Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО "АмГУ"
Факультет математики и информатики

УТВЕРЖДАЮ
Зав. кафедрой МАиМ

_____ Т.В. Труфанова

«___»_____ 2007 г.

ПРАКТИКУМ НА ЭВМ

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

для специальности 010501 – "Прикладная математика и информатика"

Составители: А.В. Рыженко, В.А. Труфанов

Благовещенск
2007 г.

ББК

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Рыженко А.В., Труфанов В.А.

Практикум на ЭВМ: Учебно-методический комплекс по дисциплине для студентов АмГУ очной формы обучения специальности 010501 "Прикладная математика и информатика". – Благовещенск: Амурский гос. ун-т, 2007. – 345 с.

Учебно-методический комплекс по дисциплине "Практикум на ЭВМ" предназначен для студентов специальности 010501 – "Прикладная математика и информатика" очной формы обучения, призван помочь ведущим преподавателям и студентам в организации процесса изучения дисциплины. Комплекс содержит рабочую программу дисциплины, материалы для проведения лабораторных работ, справочный материал и библиографический список.

I. РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Рабочая программа по дисциплине "Практикум на ЭВМ" для специальности 010501 – "Прикладная математика и информатика".

Курс 1–3. Семестр 2–6. Лекции нет. Экзамен нет. Практические (семинарские) занятия (нет). Зачет 2, 3, 4, 5, 6 семестр. Лабораторные занятия 324 (72+72+36+72+72) час. Самостоятельная работа 76 час. Всего 400 часов.

Составители А.В. Рыженко, ст. преподаватель; В.А. Труфанов, доцент. Факультет математики и информатики. Кафедра математического анализа и моделирования. Благовещенск, 2006 г.

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

1.1. Цель преподавания дисциплины.

Данный курс составляет часть дисциплин: информатика, языки программирования, методы оптимизации, численные методы. Поэтому основной целью дисциплины «Практикум на ЭВМ» является приобретение практических навыков применения ЭВМ при решении задач различного типа через средства программируемых сред для разработки программ (программных комплексов или алгоритмов), используя численные методы.

1.2. Задачи изучения дисциплины.

В результате изучения дисциплины студенты должны знать: 1) основные среды программирования; 2) как использовать различные методы программирования; 3) основные принципы разработки алгоритмов и программ; 4) как составлять и оформлять программную документацию.

В результате изучения дисциплины студенты должны уметь: 1) использовать простейших элементов и объектов программных средств для разработки программных комплексов; 2) создавать и разрабатывать

алгоритмы решения задач; 3) отлаживать программы по заранее готовым тестам; 4) разрабатывать рабочие программы, использующие классические численные методы, в том числе методы линейной алгебры и математического анализа, методы оптимизации, методы решения дифференциальных уравнений; 5) модифицировать алгоритмы и методы решения в соответствии с условиями задач.

1.3. Перечень дисциплин с указанием разделов (тем), усвоение которых студентами необходимо при изучении данной дисциплины.

Дисциплина является вспомогательной (базовой) для изучения других дисциплин специальности, связанных с программированием. Курс является составной частью цикла специальных дисциплин, определяющих подготовку студентов в области современных информационных технологий и предусмотрен государственным стандартом высшего профессионального образования для данной специальности. Курс использует теоретические знаний полученные студентами в рамках дисциплин Информатика, Языки программирования и методы трансляции, Системное и прикладное программное обеспечение, Операционные системы и сетевые технологии.

1.4. Структура дисциплины

Дисциплина рассчитана на пять семестров. В первом семестре раскрываются общие концепции языков высокого уровня. Подробно рассматривается язык высокого уровня C/C++. Уделяется особое внимание алгоритмизации и структурному подходу к написанию программ. Делается акцент на операции ввода-вывода и рекурсию. Во втором семестре изучается среда программирования Delhi. Детально рассматриваются свойства объектов с точки зрения программирования на языке Object Pascal. В третьем семестре рассматриваются прикладные математические задачи, которые решаются с помощью пакетов MATHCAD, MATLAB. В четвертом семестре изучается программирование трехмерной графики с использованием библиотеки OpenGL. В пятом семестре рассматривается

объектно-ориентированный подход к написанию программ и динамическое программирование.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. Федеральный компонент.

Практикум по программированию; практикум по решению прикладных задач (практическое освоение работы на ЭВМ, умение применять стандартные математические методы и математическое обеспечение ЭВМ для решения различных задач).

2.2. Наименование тем, их содержание, объем в лекционных часах.

Лекции не предусмотрены.

2.3. Практические и семинарские занятия, их содержание и объем в часах.

Практические и семинарские занятия не предусмотрены.

2.4. Лабораторные занятия, их наименование и объем в часах.

2 семестр: Программирование на языке высокого уровня C/C++

1. Обучение работы в интегрированной среде Visual C++ 6 – 2 часа.
2. Линейные алгоритмы – 4 часа.
3. Разветвляющиеся, циклические и итерационные алгоритмы – 14 часа.
4. Одномерные массивы и указатели – 7 часа.
5. Двумерные массивы – 7 часа.
6. Строки и файлы – 8 часа.
7. Структуры – 6 часа.
8. Функции – 8 часа.
9. Рекурсия – 8 часа.
10. Элементарная машинная графика – 6 часа.
11. Зачетное занятие – 2 часа.

3 семестр: Среда Delphi и Object Pascal

12. Среда разработки Delphi – 4 часа.

13. Использование визуальных компонентов: компоненты для обработки текста – 4 часа.
14. Использование визуальных компонентов: компоненты управляющих команд – 4 часа.
15. Использование визуальных компонентов: графические компоненты – 2 часа.
16. Использование визуальных компонентов: компоненты для доступа к файлам и каталогам – 4 часа.
17. Создание приложений с использованием меню – 4 часа.
18. Создание приложений с использованием диалоговых окон – 4 часа.
19. Создание приложений с использованием графических объектов – 8 часа.
20. Создание приложений с использованием главной и дочерней форм – 4 часа.
21. Использование различных моделей программирования: инкапсуляция, полиморфизм, наследование объектов – 12 часа.
22. Использование методов при программировании: виртуальные, динамические, абстрактные – 12 часа.
23. Работа с графикой – 8 часа.
24. Зачетное занятие – 2 часа.

4 семестр: Применение математических пакетов MathCAD и MATLAB для решения задач линейной алгебры, математического анализа и численных методов

25. Пакет MathCAD – основные возможности – 2 часа.
26. Графические возможности. Создание различного рода графиков, их форматирование и использование – 2 часа.
27. Символьные вычисления – 2 часа.
28. Действия с векторами и матрицами – 4 часа.
29. Решение алгебраических выражений и систем – 4 часа.
30. Аппроксимация функций – 4 часа.

31. Решение дифференциальных уравнений – 4 часа.
32. Программирование – 6 часа.
33. Контрольное занятие – 2 часа.
34. Основы работы с пакетом MATLAB – 4 часа.
35. Зачетное занятие – 2 часа.

5 семестр: Основы программирования трехмерной графики с использованием библиотеки OpenGL

36. Стандартные фигуры библиотеки GLAUX – 9 часа.
37. Генерация движущихся изображений – 9 часа.
38. Геометрические примитивы – 9 часа.
39. Полигональная аппроксимация поверхностей – 9 часа.
40. Цвет и освещение – 9 часа.
41. Свойства материала и спецэффекты освещения – 9 часа.
42. Растровые объекты: изображение и текстуры – 8 часа.
43. Примеры программ с использованием OpenGL – 8 часа.
44. Зачетное занятие – 2 часа.

6 семестр: Динамические типы данных и объектно-ориентированное программирование

45. Работа с динамическими переменными – 4 часа.
46. Односвязные списки (модификация мини ИС) – 6 часа.
47. Двусвязные списки (доработка мини ИС) – 6 часа.
48. Стеки, стековый калькулятор – 6 часа.
49. Очереди, задачи моделирования очередей – 6 часа.
50. Двусвязное кольцо – 6 часа.
51. Деревья – 6 часа.
52. Применение ООП – 4 часа.
53. Использование инкапсуляции – 4 часа.
54. Использование наследования – 4 часа.
55. Использование полиморфизма – 4 часа.
56. Создание динамических объектов – 12 часа.

57.Зачетное занятие – 4 часа.

Выбор темы самостоятельной работы и практического задания оговариваются с каждым студентом отдельно.

2.5. Самостоятельная работа студентов.

В качестве самостоятельной работы студентам предлагается рассмотреть и изучить следующие вопросы:

1. Хеширование и синтаксический анализ.
2. Компоненты организации управления приложением.
3. Тулбоксы пакета MatLab.
4. Моделирование поверхностей полигональными сетками.
5. AVL-деревья.

Контроль над выполнением самостоятельной работы осуществляется проведением самостоятельных лабораторных работ, кроме этого они также включены в перечень вопросов к зачету.

2.6. Вопросы к зачету.

2 семестр

1. Понятие о файлах. Процедуры открытия и закрытия файлов.
2. Процедуры и функции работы с файлами.
3. Текстовые файлы.
4. Записи. Записи с вариантами.
5. Процедурные типы.
6. Нетипизированные параметры процедур и функций.

3 семестр

1.Структурные типы данных в Delphi.
2.Описание нового типа.
3.Сложные структуры данных.
4.Основные стандартные функции для работы с типами.
5.Преобразование типов.
6.Инициализация констант сложных типов.
7.Структура подпрограмм.
8.Понятие объекта.
9.Понятие класса.

10.Типы методов.
11.Динамическое конструирование объектов.
12.Использование визуальных компонентов.
13.Компонент Меню.
14.Компонент Контекстное меню.
15.Компонент Текстовая область.
16.Компонент Переключатель.
17.Компонент Группа переключателей.
18.Компонент Поле со списком.
19.Компонент Флажок.
20.Компонент Полоса прокрутки.
21.Иерархия компонентов.
22.Класс Tobject.
23.Класс Tpresistent (наследник Tobject).
24.Класс Tcomponent (наследник Tpresistent).
25.Форма.
26.Управление проектом. Добавление новой формы.
27.Что такое отладка программы?
28.Причины ошибок.
29.Синтаксические ошибки.
30.Логические ошибки.
31.Выполнение программы по шагам.
32.Просмотр и изменение значений.
33.Просмотр и анализ кода.
34.Прерывания по условию.
35.Организация точек прерывания в группы.
36.Ведение протокола работы.
37.Отладка внешних процессов.
38.Инспектор объектов.
39.Генерация исключительных ситуаций.
40.Стандартные классы исключительных ситуаций.
41.Контроль над исключительными ситуациями.
42.Программный обработчик ошибок.

4 семестр

1.MATHCAD. Основные возможности пакета. Создание и редактирование формул. Создание математических и текстовых областей, параграфы.
2.Символьные вычисления. Графические возможности. Создание различного рода графиков, их форматирование и использование.
3.Форматирование выражений и результатов. Локальный и глобальный формат. Дискретный аргумент. Переменные и константы. Способы определения и вычисления переменных и функций. Встроенные переменные.
4.Создание векторов и матриц, способы их отображения. Оператор векторизации. Основные встроенные функции. Решение СЛАУ.
5.Аппроксимация функций.

6.Решение алгебраических уравнений.
7.Программирование.
8.Решение дифференциальных уравнений и систем.
9.Ряды Фурье.
10.Задачи из теории вероятностей.
11.Решение задач численными методами.
12.Программирование: составные операторы, условные операторы, условные циклы.
13.Ввод и вывод данных в файлы и в другие программы.
14.MatLab. Переменные и их типы. Команды для задания различного вида матриц.
15.MatLab. Построение графиков функций, виды графиков.
16.MatLab. Нахождение корней уравнения, полинома.
17.MatLab. Многомерные массивы: определение, команды обработки.
18.MatLab. Программирование: m-файлы и m-функции.
19.MatLab. Решение уравнений, задач мат. анализа и линейной алгебры с помощью тулбоксов.

5 семестр

6 семестр

1.Создание пользовательского модуля и использование его в программе.
2.Структурированные типы данных. Множества.
3.Динамические данные. Указатели.
4.Ссылочные типы.
5.Динамические массивы.
6.Списки.
7.Стек. Организация стека.
8.Операции со стеками. Удаление и добавление элемента в стек.
9.Выделение и освобождение динамической памяти. Процедуры и функции, используемые при работе с указателями и динамической памятью.
10.Очередь. Организация очереди.
11.Удаление первого и добавление последнего элемента в очередь.
12.Принципы объектно-ориентированного программирования.
13.Описание объекта.
14.Инкапсуляция. Методы. Классы и объекты.
15.Сообщения и события, реагирование объекта на сообщения и события.
16.Наследование.
17.Полиморфизм.
18.Иерархия объектов.
19.Виртуальные методы. Объявление виртуальных методов.
20.Конструкторы и деструкторы.
21.Понятие о функциях Windows API.

3. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

3.1. Перечень обязательной (основной) литературы.

1. Павловская Т.А. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2001.
2. Павловская Т.А., Щупак Ю.А. C/C++. Структурное программирование: Практикум. – СПб.: Питер, 2003.
3. Д. Мэтчо и др. Delphi 2. Руководство для профессионалов: пер. с англ. - Спб.: BHV - Санкт-Петербург, 1997. - 784с.
4. Культин Н.Б. Основы программирования в Delphi 7. – СПб.: БХВ-Петербург, 2003.– 608 с.
5. Р. Баас, М. Фервай, Х. Гюнтер. Delphi 4: полное руководство: пер.с нем. - К.: Издательская группа BHV, 1999. - 800с.
6. Фараонов В.В. Delphi 4. Учебный курс. - М.: "Нолидж", 1999. - 464с.
7. Архангельский А.Я. Delphi 5. Справочное пособие. - М: ЗАО "Издательство БИНОМ", 2001.-768с.
8. Гурский Д.А. MathCAD для студентов и школьников. Популярный самоучитель. – СПб.: Питер, 2005. – 400 с.
9. Макаров Е.Г. Инженерные расчёты в MathCAD. Учебный курс. – СПб.: Питер, 2004. – 448с.
10. Богуславский А.А. C++ и компьютерная графика.– М.: Компьютер Пресс, 2003.

3.2. Перечень дополнительной литературы.

1. Тюкачев Н., Свиридов Ю. Delphi 5. Создание мультимедийных приложений. Учебный курс. - СПб.: Питер, 2001. – 400с.
2. М. Канту, Т. Гуч и др. Delhi. Руководство разработчика: Пер. с англ. – К.: ВЕК+, М.: ЭНТРОП, М.: ДЕСС, 1999. – 752 с.
3. Краснов М.В. OpenGL. Графика в проектах Delphi. – СПб.: БХВ-Петербург, 2001. – 352с.
4. Лаптев В.В. C++. Экспресс-курс. – СПб.: БХВ-Петербург, 2004. – 512 с.

3.3. Перечень наглядных и иных пособий.

1. Карточки с заданиями к лабораторным работам / *А.В. Рыженко*.
2. Карточки с заданиями к лабораторным работам / *В.А. Труфанов*.

3.4. Средства обеспечения освоения дисциплины.

1. Среда программирования Visual C++.
2. Среда программирования Borland Delphi.
3. Пакеты MatCad, MatLab.

4. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Компьютерный класс кафедры МАиМ.

5. КРИТЕРИИ ОЦЕНКИ ЗНАНИЙ

Текущий контроль за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения лабораторных работ посредством устного опроса по изучаемым разделам и проверки отчетности по лабораторным работам и индивидуальным занятиям. Промежуточный контроль предполагает систематическое проведение контрольных работ.

Зачет сдаются в конце семестра. Форма сдачи – устная. Необходимым допуском на экзамен (зачет) является сдача всех лабораторных работ, положительные оценки за промежуточные контрольные работы. В предлагаемый билет входят три вопроса: два теоретических и один практический (разработка программы), на которые студент должен дать развернутый ответ. Показать знание теории по данной части курса, продемонстрировать свободную ориентацию в материале, знание понятий и терминологии, ответить на дополнительные вопросы. Выполнение указанных требований оценивается оценкой (зачтено – не зачтено).

II. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Форма проведения лабораторного занятия: а) приветствие студентов, 1 мин.; б) определение личного состава студенческой группы, 4 мин.; в) объявление тематики и вопросов лабораторного занятия, 1 мин.; г) выполнение заданий, 70 мин.; д) подведение итогов лабораторного занятия, 13 мин.; е) прощание со студентами, 1 мин.

Список рекомендуемой литературы (основной и дополнительной) приведен в рабочей программе.

III. ЛАБОРАТОРНЫЙ ПРАКТИКУМ: МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ИЛИ КОМПЛЕКТ ЗАДАНИЙ

Задания для лабораторных, контрольных работ и домашних заданий берутся из книг, реквизиты которых приведены в рабочей программе.

2 семестр

Лабораторная работа № 1.

Программирование алгоритмов линейной структуры.

Цель работы. Овладение практическими навыками разработки и программирования вычислительного процесса линейной структуры и навыками по отладке и тестированию программ.

Задания для самостоятельной подготовки.

1. Изучить:

– запись констант, переменных, стандартных функций;

– правила записи арифметических выражений;

– организацию простейшего ввода-вывода данных.

2. Разработать алгоритм решения в соответствии с заданием.

3. Составить программу решения задачи.

4. Подготовить тестовый вариант исходных данных и вычислить для них с помощью калькулятора значения вычисляемых в программе величин, затем следует сравнить их со значениями, вычисленными в компьютере

(результат вычисления по первой формуле должен совпадать со второй, но только для чётных вариантов).

Задание к работе.

Вычислить в компьютере значения по заданным двум расчётным формулам и наборам исходных параметров, указанных в таблице (вариант задаётся преподавателем). На печать вывести значения вводимых данных и результаты вычислений, осуществляя его в соответствии со следующей разметкой строк:

```
- - - - - - - - - - - - - - - - - - - - - -
|   ИСХОДНЫЕ ДАННЫЕ   |
- - - - - - - - - - - - - - - - - - - - - -
        < пропуск 1-ой строки >
< имя > . . . . ., < имя > . . . . ., . . .
. . . . . .
        < пропуск одной строки >
* * * * * * * * * * * * * * * * * * * * *
*  РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЙ *
* * * * * * * * * * * * * * * * * * * * *
        < пропуск 1-ой строки >
< имя > = . . . . . .
        * * * * * * *
                < имя > = . . . . . .
                * * * * * * * * * *
```

Варианты заданий	Расчётные формулы	Значения параметров
1	$a = \frac{2\cos(x - \pi/6)}{1/2 + \sin^2(y)}$, $b = 1 + \frac{z^2}{3 + z^2/5}$	$x = 1.426$
2	$y = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$, $z = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$	
3	$\gamma = \lvert x^{y/x} - \sqrt[3]{y/x} \rvert$, $\psi = (y - x)\frac{y - z/(y - z)}{1 + (y - x)^2}$	$x = 1.825$

4	$s = \dfrac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$, $\psi = 2\sin\alpha$	
5	$y = e^{-bt}\sin(at + b) - \sqrt{\lvert bt + a \rvert}$, $s = b\sin(at^2\cos 2t) - 1$	$a = -0.5$; $b = 1.7$
6	$z = 1 - \dfrac{1}{4}\sin^2 2\alpha + \cos 2\alpha$, $w = \cos^2\alpha + \cos^4\alpha$	
7	$w = \sqrt{x^2 + b} - b^2\sin^3(x + a)/x$, $y = \cos^2 x^3 - x/\sqrt{a^2 + b^2}$	$a = 1.5$; $b = 15.5$
8	$p = \cos^2\left(\dfrac{3}{8}\pi - \dfrac{\alpha}{4}\right) - \cos^2\left(\dfrac{11}{8}\pi + \dfrac{\alpha}{4}\right)$, $t = \dfrac{\sqrt{2}}{2}\sin\dfrac{\alpha}{2}$	
9	$s = x^3\,\mathrm{tg}^2(x + b)^2 + a/\sqrt{x + b}$, $Q = \dfrac{bx^2 - a}{e^{ax} - 1}$	$a = 16.5$; $b = 3.4$
10	$\gamma = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$, $\varphi = -4\sin^2\dfrac{\alpha - \beta}{2}\cdot\cos(\alpha + \beta)$	
11	$R = x^2(x + 1)/b - \sin^2(x + a)$, $S = \sqrt{xb/a} + \cos^2(x + b)^3$	$a = 0.7$; $b = 0.05$
12	$W = \dfrac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$, $Z = \dfrac{1 - \mathrm{tg}\alpha}{1 + \mathrm{tg}\alpha}$	
13	$f = \sqrt[3]{m\,\mathrm{tg}\,t + \lvert c\sin t \rvert}$, $z = m\cos(bt\sin t) + c$	$m = 2$; $c = -1$; $b = 0.7$
14	$z = \dfrac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$, $y = \dfrac{1 + \sin 2\beta}{\cos 2\beta}$	
15	$y = b\,\mathrm{tg}^2 x - \dfrac{a}{\sin^2(x/a)}$, $d = ae^{-\sqrt{a}}\cos(bx/a)$	$a = 3.2$; $b = 17.5$
16	$z = \dfrac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2}$, $p = \dfrac{1}{\sqrt{b + 2}}$	
17	$f = \ln(a + x^2) + \sin^2(x/b)$, $z = e^{-cx}\dfrac{x + \sqrt{x + a}}{x - \sqrt{\lvert x - a \rvert}}$	$a = 10.2$; $b = 9.2$; $c = 0.5$
18	$z = \dfrac{\sqrt{(3m + 2)^2 - 24m}}{3\sqrt{m} - 2/\sqrt{m}}$, $f = \ln(x + 6)/(e^x - 6)$	
19	$y = \dfrac{a^{2x} + b^{-x}\cos(a + b)x}{x + 1}$, $R = \sqrt{x^2 + b} - b^2\sin^3(x + a)/x$	$a = 0.3$; $b = 0.9$
20	$z = \left(\dfrac{1 + a + a^2}{2a + a^2} + 2 - \dfrac{1 - a + a^2}{2a - a^2}\right)^{-1}(5 - 2a^2)$, $f = \dfrac{4 - a^2}{2}$	
21	$w = \sqrt{ax\sin 2x + e^{-2x}(x + b)}$, $z = \cos^2 x^3 - x/\sqrt{a^2 + b^2}$	$a = 0.5$; $b =$

		3.1
22	$z = x - x^3/6 + x^5/120$, $y = \sin x$	
23	$U = \dfrac{a^2 x + e^{-x}\cos bx}{bx - e^{-x}\sin bx + 1}$, $f = e^{2x}\ln(a + x) - b^{3x}\ln(b - x)$	$a = 0.5$; $b = 2.9$
24	$z = \dfrac{(m-1)\sqrt{m} - (n-1)\sqrt{n}}{\sqrt{m^3 n} + nm + m^2 - m}$, $w = \dfrac{\sqrt{m} - \sqrt{n}}{m}$	
25	$s = e^{-ax}\sqrt{x+1} + e^{-bx}\sqrt{x+1.5}$, $z = \dfrac{\sin x}{\sqrt{1 + m^2\sin^2 x} - cm\ln mx}$	$m = 0.7$; $c = 2.1$; $a = 0.5$; $b = 1.08$

Лабораторная работа № 2.

Программирование алгоритмов разветвляющейся и циклической структуры.

Цель работы. Овладение практическими навыками разработки и программирования вычислительного процесса разветвляющейся и циклических структур, развитие дальнейших навыков по отладке и тестированию программ.

Задания для самостоятельной подготовки.

1. Изучить возможности языка С/С++ для реализации:
   – условной передачи управления;
   – вычислительного процесса разветвляющейся структуры;
   – вычислительных процессов циклической структуры с известным числом повторений в цикле;
   – приёма программирования – табулирования функции от одного аргумента (вычисление значений функции при изменении значения аргумента в заданном диапазоне с шагом Δx);
   – поразрядных операций.
2. Разработать алгоритм решения в соответствии с заданием.
3. Составить программу решения задачи.

4. Подготовить тесты (число тестов равно числу ветвей вычислительного процесса) для проверки правильности выполнения программы.

Подсказки. Для проверки условий в операторе if вам могут потребоваться некоторые логические и поразрядные операции.

Символ	Операция	Пример	Значение
<	меньше, чем	4 < 5	TRUE (истина)
<=	меньше или равно	43 <= 21	FALSE (ложь)
>	больше, чем	4 > 7	FALSE
>=	больше или равно	9 >= 5	TRUE
==	равно	21 == 6	FALSE
!=	не равно	21 != 6	TRUE
&&	Логическое И	5>2 && 7>9	FALSE
||	Логическое ИЛИ	5>2 || 7>9	TRUE
&	Поразрядное И	1&1 1&0, 0&1, 0&0	1 (TRUE) 0 (FALSE)
|	Поразрядное ИЛИ	1|1, 1|0, 0|1 0|0	1 (TRUE) 0 (FALSE)
^	Поразрядное исключающее ИЛИ (сложение по модулю 2)	1^1 1^0, 0^1 0^0	10 1 0
~	Исключающее НЕ	~1 ~0	0 1

Язык C++ пригоден для системных разработок, которые и требуют поразрядные логические операции (ПЛО). При использовании любой ПЛО происходит побитовая обработка внутренних данных – двоичных чисел, состоящих из единиц и нулей (только над двоичным представлением переменных типа int, либо long, либо char). ПЛО предназначены не только для системных программистов. Прикладные программисты могут также улучшить свои программы, научившись пользоваться этими операциями. Заменив единицы и нули на TRUE (истина) и FALSE (ложь), вы можете

соотнести поразрядные операции с обычными логическими операциями && и | |, которые используются в if-сравнениях.

Задание к работе.

Вычислить и вывести на экран в виде таблицы значения функции *F* на интервале от $X_{нач.}$ до $X_{кон.}$ с шагом dX. Параметры функции *a*, *b* и *c* – действительные числа. Функция *F* должна принимать действительное значение, если проверяемое логическое выражение с поразрядными операциями не равно нулю, и целое значение в противном случае. Через $A_ц$, $B_ц$, $C_ц$ и $D_ц$ обозначены целые части значений *a*, *b*, *c*, *d*. Операции И, ИЛИ, НЕ, MOD2 (сложение по модулю 2) поразрядные. Значения исходных данных *a*, *b*, *c*, *d*, $X_{нач.}$, $X_{кон.}$, dX (в зависимости от задания) ввести с клавиатуры, сопровождая их ввод с предварительным выводом соответствующих им наименований.

Организовать вывод значений аргумента и вычисленного значения функции в виде таблицы с заголовком:

ТАБЛИЦА ФУНКЦИИ Y(X)	
X	Y
…	…

Вариант задания	Функция	Условие	Значение функции, зависимое от условия
1	$F = \begin{cases} ax^2 + bx + c, \\ a/x + \sqrt{x^2+1}, \\ (a+bx)/\sqrt{x^2+1} \end{cases}$	$x < 1.2$; $x = 1.2$; $x > 1.2$	($A_ц$ MOD2 $B_ц$) И ($B_ц$ ИЛИ $C_ц$)
2	$F = \begin{cases} -\dfrac{2x-b}{cx-a}, \\ \dfrac{x-a}{x-c}, \\ -\dfrac{x}{c} + \dfrac{-c}{2x} \end{cases}$	$x < 0$ и $b \neq 0$; $x > 0$ и $b = 0$; в остальных случаях	НЕ($A_ц$ ИЛИ $B_ц$) И НЕ($A_ц$ ИЛИ $C_ц$)
3	$F = \begin{cases} \pi x^2 - 7/x^2, \\ ax^2 + 7\sqrt{x}/b, \\ \ln(c + 7\sqrt{\lvert x + ab \rvert}) \end{cases}$	$x < 1.4$; $x = 1.4$; $x > 1.4$	($A_ц$ И $B_ц$) ИЛИ НЕ $C_ц$

4	$F = \begin{cases} a(x+7)^2 - b, \\ (x - cd)/ax, \\ x/c \end{cases}$	$x < 5$ и $b \neq 0$; $x > 5$ и $b = 0$; в остальных случаях	($A_ц$ MOD2 $B_ц$) ИЛИ ($C_ц$ ИЛИ $D_ц$)
5	$F = \begin{cases} x\sqrt[3]{x - a}, \\ x\sin bx, \\ e^{-cx}\cos ax \end{cases}$	$x < a$; $x = a$; $x > a$	($A_ц$ И $B_ц$) И ($B_ц$ ИЛИ $C_ц$)
6	$F = \begin{cases} ax^3 + bx^2, \\ (x - a)/(x - c), \\ \dfrac{x+5}{c(x-10)} \end{cases}$	$x < 0$ и $b \neq 0$; $x > 0$ и $b = 0$; в остальных случаях	НЕ ($A_ц$ И $B_ц$ И $C_ц$)
7	$F = \begin{cases} \dfrac{a+b}{e^x + \cos x}, \\ (a+b)/(x+1), \\ e^x + \sin cx \end{cases}$	$x < 2.8$; $2.8 \le x < 6$; $x > 6$	($A_ц$ MOD2 $B_ц$) И ($A_ц$ ИЛИ $C_ц$)
8	$F = \begin{cases} ax^2 - cx + b, \\ \dfrac{x-a}{x-c}, \\ \dfrac{-x}{a-c} \end{cases}$	$x+10<0$ и $b \neq 0$; $x+10>0$ и $b = 0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) И НЕ($A_ц$ ИЛИ $C_ц$)
9	$F = \begin{cases} \dfrac{a}{x} + bx^2 + c, \\ x, \\ ax + bx^3 \end{cases}$	$x < 4$; $4 \le x \le 6$; $x > 6$	НЕ($A_ц$ ИЛИ $B_ц$) И ($B_ц$ ИЛИ $C_ц$)
10	$F = \begin{cases} a(x+c)^2 - c, \\ \dfrac{x-a}{-b}, \\ a + \dfrac{x}{c} \end{cases}$	$x = 0$ и $b = 0$; $x = 0$ и $b \neq 0$; в остальных случаях	($A_ц$ MOD2 $B_ц$) И НЕ($A_ц$ ИЛИ $C_ц$)

11	$F = \begin{cases} -ax^2 + b, \\ \frac{x}{x - c} + 5.5, \\ \frac{x}{-c} \end{cases}$	$x < 0$ и $b \neq 0$; $x > 0$ и $b=0$; в остальных случаях	НЕ($A_ц$ ИЛИ $B_ц$ ИЛИ $C_ц$)
12	$F = \begin{cases} -ax^3 - b, \\ \frac{x - b}{x - c}, \\ \frac{x}{c} + \frac{c}{x} \end{cases}$	$x+c$ и $a \neq 0$; $x+c$ и $a=0$; в остальных случаях	($A_ц$ MOD2 $B_ц$) ИЛИ ($A_ц$ MOD2 $C_ц$)
13	$F = \begin{cases} ax^2 + b, \\ \frac{x - a}{x}, \\ \frac{x}{c} \end{cases}$	$x-1<0$ и $b-x \neq 0$; $x-1>0$ и $b+x=0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) MOD2 ($B_ц$ И $C_ц$)
14	$F = \begin{cases} ax^3 + b^2 + c, \\ \frac{x - a}{x - c}, \\ \frac{x}{c} + \frac{x}{a} \end{cases}$	$x<0.6$ и $b+c \neq 0$; $x>0.6$ и $b+c=0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) И $C_ц$
15	$F = \begin{cases} ax^2 + b/c, \\ \frac{x - a}{(x - b)^2}, \\ \frac{x^2}{c^2} \end{cases}$	$x<1$ и $c \neq 0$; $x>1.5$ и $c=0$; в остальных случаях	($A_ц$ И $B_ц$) MOD2 $C_ц$
16	$F = \begin{cases} ax^2 - bx + c, \\ \frac{x - a}{x - c}, \\ \frac{x}{c} \end{cases}$	$x<3$ и $b \neq 0$; $x>3$ и $b=0$; в остальных случаях	НЕ($A_ц$ ИЛИ $B_ц$) И ($A_ц$ MOD2 $C_ц$)

17	$F = \begin{cases} ax^2 + b^2 x, \\ x - \dfrac{x + 0.5}{b - c}, \\ 1 + \dfrac{x}{c} \end{cases}$	$x<0$ и $a\neq 0$; $x>0$ и $a=0$; в остальных случаях	НЕ($A_ц$ ИЛИ $B_ц$) И ($B_ц$ ИЛИ $C_ц$)
18	$F = \begin{cases} -ax^2 - b^2 + c, \\ \dfrac{x - a}{x}, \\ \dfrac{-x}{c} \end{cases}$	$x<5$ и $c\neq 0$; $x>5$ и $c=0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) MOD2 ($A_ц$ ИЛИ $C_ц$)
19	$F = \begin{cases} -ax^2, \\ \dfrac{b - x}{cx}, \\ \dfrac{x}{c} \end{cases}$	$c<0$ и $a\neq 0$; $c>0$ и $a=0$; в остальных случаях	($A_ц$ MOD2 $B_ц$) И НЕ($A_ц$ ИЛИ $C_ц$)
20	$F = \begin{cases} ax^2 + b^2 x, \\ \dfrac{x + a}{x + c}, \\ \dfrac{x}{c} \end{cases}$	$c<0$ и $b\neq 0$; $c>0$ и $b=0$; в остальных случаях	($A_ц$ И $B_ц$) ИЛИ ($A_ц$ И $C_ц$)
21	$F = \begin{cases} a - \dfrac{x}{10 + b}, \\ \dfrac{x - a}{x - c}, \\ 3x + 2/c \end{cases}$	$x<0$ и $b\neq 0$; $x>0$ и $b=0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) И $C_ц$
22	$F = \begin{cases} -ax - c, \\ \dfrac{b - a}{-c}, \\ \dfrac{bx}{c - a} \end{cases}$	$c<0$ и $x\neq 0$; $c>0$ и $x=0$; в остальных случаях	НЕ($A_ц$ ИЛИ $B_ц$) ИЛИ $C_ц$
23	$F = \begin{cases} ax^2 + bx + c, \\ \dfrac{-a}{x - b}, \\ a(x + c) \end{cases}$	$a<0$ и $c\neq 0$; $a>0$ и $c=0$; в остальных случаях	$A_ц$ И ($B_ц$ ИЛИ $C_ц$)

24	$F = \begin{cases} 1/ax - b, \\ \dfrac{x - c}{x}, \\ \dfrac{10x}{c - 4} \end{cases}$	$x+5<0$ и $c=0$; $x+5>0$ и $c\neq 0$; в остальных случаях	($A_ц$ И $B_ц$) ИЛИ ($B_ц$ И $C_ц$)
25	$F = \begin{cases} ax^2 + bx + c, \\ \dfrac{-a}{x - b}, \\ a(x + c) \end{cases}$	$x<0$ и $b\neq 0$; $x>0$ и $b=0$; в остальных случаях	($A_ц$ ИЛИ $B_ц$) И ($A_ц$ ИЛИ $C_ц$)

Лабораторная работа № 3.

Программирование алгоритмов итерационной циклической структуры.

Цель работы. Овладение практическими навыками разработки и программирования алгоритмов итерационной циклической структуры; приобретение дальнейших навыков по отладке и тестированию программ.

Задания для самостоятельной подготовки.

1. Изучить:

– организацию итерационных циклов;

– возможности языка C/C++ для организации таких циклов;

– приёма программирования – уточнение корня уравнения методом итерации, вычисление суммы членов бесконечного ряда, накопление суммы.

2. Разработать алгоритмы решения задач для двух заданий 1) и 2).

3. Составить программы решения задач для заданий 1) и 2).

Задание к работе.

Задание 1). Методом итераций вычислить корень уравнения вида $F(x)=0$, расположенный на интервале $[\alpha, \beta]$ абсолютной точностью ε (в соответствии с вариантом задания). Там, где интервал нахождения корня не задан, нужно – предварительно графически отделить его. Определить также

число итераций, необходимое для нахождения корня. Предусмотреть защиту от “вечного цикла”.

Вариант задания	Уравнение	Отрезок	Точность
1	$e^{x}-e^{-x}-2=0$	[0; 1]	10^{-3}
2	$3\sin\sqrt{x}+0{,}35x-3{,}8=0$	[2; 3]	10^{-3}
3	$\ln(x)+(x+1)^{3}=0$	Определить	10^{-5}
4	$x-2+\sin(1/x)=0$	[1,2; 2]	10^{-4}
5	$1-x+\sin(x)-\ln(1+x)=0$	[0; 1,5]	10^{-5}
6	$\sqrt{x+1}-1/x=0$	Определить	10^{-3}
7	$x^{2}-\ln(1+x)-3=0$	[2; 3]	10^{-4}
8	$x-\frac{1}{3+\sin(3{,}6x)}=0$	[0; 0,85]	$0{,}5\cdot10^{-4}$
9	$3x+\cos(x)+1=0$	Определить	10^{-3}
10	$\ln(x)-x+1{,}8=0$	[2; 3]	$0{,}5\cdot10^{-4}$
11	$0{,}1x^{2}-x\cdot\ln(x)=0$	[1; 2]	$0{,}5\cdot10^{-3}$
12	$(x-1)^{2}-0{,}5\cdot e^{x}=0$	Определить	10^{-4}
13	$x+\cos(x^{0{,}52}+2)=0$	[0,5; 1]	10^{-3}
14	$\sqrt{1-0{,}4x}-\arcsin x=0$	[0; 1]	10^{-3}
15	$x^{2}+4\sin(x)=0$	Определить	10^{-5}
16	$x^{2}-\ln(x+1)=0$	Определить	10^{-4}
17	$3x-4\cdot\ln(x)-5=0$	[2; 4]	$0{,}5\cdot10^{-3}$
18	$0{,}5x+\lg(x-1)-0{,}5=0$	Определить	10^{-3}
19	$0{,}4+\operatorname{arctg}\sqrt{x}-x=0$	[1; 2]	10^{-3}
20	$\arccos x-\sqrt{1-0{,}3x^{3}}=0$	[0; 1]	$0{,}5\cdot10^{-3}$
21	$\operatorname{tg}(0{,}5x+0{,}2)-x^{2}=0$	Определить	10^{-4}
22	$2x-3\cdot\ln(x)-3=0$	[0,5; 0,6]	10^{-3}
23	$2x+\lg(2x+3)-1=0$	[0; 0,5]	10^{-4}
24	$x\cdot\lg(x)-1{,}2=0$	Определить	10^{-3}
25	$\arcsin(2x+1)=x^{2}$	[-0,5; 0]	10^{-4}

Задание 2). Вычислить значение суммы членов бесконечного ряда с заданной точностью $\varepsilon=10^{-5}$ на интервале от $x_{нач}$ до $x_{кон}$ с шагом dx. На печать вывести таблицу, снабдить её заголовком и шапкой. Каждая строка должна содержать значение аргумента, значение функции, количество просуммированных членов ряда и точное значение функции, для

сопоставления с вычисленным результатом. Сделать проверку на расходимость в программе.

Варианты заданий	Функции, разложенные в ряд	Область сходимости
1	$\ln\frac{x+1}{x-1} = 2\sum_{n=0}^{\infty}\frac{1}{(2n+1)x^{2n+1}} = 2\left(\frac{1}{x}+\frac{1}{3x^2}+\frac{1}{5x^5}+\ldots\right)$	$\lvert x\rvert > 1$
2	$e^{-x} = \sum_{n=0}^{\infty}\frac{(-1)^n x^n}{n!} = 1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\ldots$	$\lvert x\rvert < \infty$
3	$x\ln(1+x^2) = \sum_{n=1}^{\infty}\frac{(-1)^{n-1}x^{2n+1}}{n} = x^3-\frac{x^5}{2}+\frac{x^7}{3}-\ldots$	$\lvert x\rvert < 1$
4	$\ln(1+x) = \sum_{n=0}^{\infty}\frac{(-1)^n x^{n+1}}{n+1} = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots$	$-1<x\le 1$
5	$\ln\sqrt[3]{\frac{1+2x}{1-x}} = \frac{1}{3}\sum_{n=1}^{\infty}\frac{[(-1)^{n+1}2^n+1]x^n}{n} = \frac{1}{3}\left(3x-\frac{3x^2}{2}+\frac{9x^3}{3}-\frac{15x^4}{4}+\ldots\right)$	$\lvert x\rvert < ½$
6	$\ln\frac{1+x}{1-x} = 2\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1} = 2\left(x+\frac{x^3}{3}+\frac{x^5}{5}+\ldots\right)$	$\lvert x\rvert < 1$
7	$\ln(1-x) = -\sum_{n=1}^{\infty}\frac{x^n}{n} = -\left(x+\frac{x^2}{2}+\frac{x^3}{3}+\ldots\right)$	$-1\le x<1$
8	$\operatorname{arcctg} x = \frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}x^{2n+1}}{2n+1} = \frac{\pi}{2}-x+\frac{x^3}{3}-\frac{x^5}{5}+\ldots$	$\lvert x\rvert \le 1$
9	$\operatorname{arctg} x = \frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = \frac{\pi}{2}-\frac{1}{x}+\frac{1}{3x^3}-\frac{1}{5x^5}+\ldots$	$x > 1$
10	$\frac{1}{\sqrt{1+x^2}} = 1+\sum_{n=1}^{\infty}\frac{(-1)^n(2n-1)!!}{(2n)!!}x^{2n}$	$\lvert x\rvert < 1$
11	$\operatorname{arctg} x = \sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{2n+1} = x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\ldots$	$\lvert x\rvert \le 1$
12	$\operatorname{arctg} x = -\frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = -\frac{\pi}{2}-\frac{1}{x}+\frac{1}{3x^3}-\frac{1}{5x^5}+\ldots$	$x < -1$
13	$e^{-x^2} = \sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{n!} = 1-x^2+\frac{x^4}{2!}-\frac{x^6}{3!}+\ldots$	$\lvert x\rvert < \infty$
14	$\cos x = \sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!} = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots$	$\lvert x\rvert < \infty$
15	$\ln(x+\sqrt{1+x^2}) = x+\sum_{n=1}^{\infty}(-1)^n\frac{(2n-1)!!}{(2n)!!}\frac{x^{2n+1}}{2n+1}$	$\lvert x\rvert < 1$
16	$\frac{\sin x}{x} = \sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n+1)!} = 1-\frac{x^2}{3!}+\frac{x^4}{5!}-\frac{x^6}{7!}+\ldots$	$\lvert x\rvert < \infty$
17	$\ln x = 2\sum_{n=0}^{\infty}\frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}} = 2\left(\frac{x-1}{x+1}+\frac{(x-1)^3}{3(x+1)^3}+\frac{(x-1)^5}{5(x+1)^5}+\ldots\right)$	$x > 0$

18	$\ln x = \sum_{n=0}^{\infty} \frac{(-1)^n (x-1)^{n+1}}{n+1} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \ldots$	$0<x\leq 2$
19	$\ln x = \sum_{n=0}^{\infty} \frac{(x-1)^{n+1}}{(n+1)(x+1)^{n+1}} = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \ldots$	$x > ½$
20	$\operatorname{ch} x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots$	$\lvert x\rvert < \infty$
21	$\arcsin x = x + \sum_{n=1}^{\infty} \frac{1\cdot 3\cdot \ldots \cdot (2n-1)\cdot x^{2n+1}}{2\cdot 4\cdot \ldots \cdot 2n\cdot (2n+1)} = x + \frac{x^3}{2\cdot 3} + \frac{1\cdot 3\cdot x^5}{2\cdot 4\cdot 5} + \ldots$	$\lvert x\rvert < 1$
22	$\arccos x = \frac{\pi}{2} - \left(x + \sum_{n=1}^{\infty} \frac{1\cdot 3\cdot \ldots \cdot (2n-1)\cdot x^{2n+1}}{2\cdot 4\cdot \ldots \cdot 2n\cdot (2n+1)} \right)$	$\lvert x\rvert < 1$
23	$\operatorname{sh} x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots$	$\lvert x\rvert < \infty$
24	$\frac{1}{(1+x)^2} = \sum_{n=0}^{\infty} (-1)^n (n+1) x^n = 1 - 2x + 3x^2 - 4x^3 + \ldots$	$\lvert x\rvert < 1$
25	$\frac{1}{\sqrt{1+x}} = 1 + \sum_{n=1}^{\infty} (-1)^n \frac{(2n-1)!!}{(2n)!!} x^n = 1 - \frac{1}{2}x + \frac{1\cdot 3}{2\cdot 4}x^2 - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}x^3 + \ldots$	$\lvert x\rvert < 1$

Лабораторная работа № 4.

Обработка одномерных массивов.

Цель работы. Овладение практическими навыками работы с массивами, особенностями их ввода и вывода; приобретение дальнейших навыков по организации программ циклической структуры с использованием приёмов программирования.

Задания для самостоятельной подготовки.

1. Изучить:

– способы описания размеров и инициализации массивов, задание динамических массивов;

– способы ввода и вывода массивом;

– реализацию приёмов накопления суммы или произведения, запоминание результатов, нахождения наибольшего и наименьшего, пересылка и сдвиг, сортировка;

– организация вычислительных структур с вложенными циклами.

2. Разработать алгоритм решения в соответствии с заданием.

3. Составить программу решения задачи.

4. Подготовить тексты для проверки программы.

Задание к работе.

При написании программ можно использовать как динамические, так и нединамические массивы. Размерность последних задаётся именованной константой.

Вариант задания	Массив	Действия	Условия и ограничения
1	X(20)	1) Вычислить сумму и количество элементов массива X, удовлетворяющих заданному условию. 2) Найти наибольший и наименьший элементы массива и их порядковые номера. 3) Упорядочить элементы массива по убыванию модулей элементов.	$0 \le x_i \le 1$, $x_i \in \boldsymbol{R}$
2	A(n)	Вычислить: 1) сумму положительных элементов массива; 2) произведение элементов массива (см. условие). 3) Упорядочить элементы массива по возрастанию.	$a_{\min} < a_i < a_{\max}$, $a_i \in \boldsymbol{R}$, $n \le 40$
3	B(30)	Вычислить: 1) сумму отрицательных элементов массива; 2) произведение элементов массива (см. условие). 3) Упорядочить элементы массива по убыванию.	$\lvert a_{\min}\rvert < a_i < \lvert a_{\max}\rvert$ $a_i \in \boldsymbol{R}$
4	C(n)	Вычислить: 1) произведение элементов массива с	$c_i \in \boldsymbol{Z}$

		чётными номерами; 2) сумму элементов массива, расположенных между первым и последним нулевыми элементами. 3) Преобразовать массив так, чтобы сначала располагались все положительные, а потом – все отрицательные (считать нулевые элементы положительными).	
5	D(40)	Вычислить: 1) сумму элементов массива с нечётными номерами; 2) сумму элементов массива, расположенных между первым и последним отрицательными элементами. 3) Сжать массив, удалив из него все элементы (см. условие). Освободившиеся в конце массива элементы заполнить нулями.	$\lvert d_i\rvert \le 1$, $d_i \in \boldsymbol{R}$
6	Y(n)	1) Найти наибольшее и наименьшее значение модуля разности между соседними элементами. 2) Вычислить среднее арифметическое значение элементов массива (см. условие 2). 3) Преобразовать массив так, чтобы сначала располагались все элементы целая часть которых не превышает 2, а потом – все остальные.	1) y_i, y_{i+1} – соседние, $i=0..n-1$; 2) $y_i > 0$; 3) $[y_i] \le 2$ $y_i \in \boldsymbol{R}$
7	C(50)	Вычислить: 1) максимальный элемент массива; 2) сумму элементов массива, расположенных до последнего	$c_i \le \in [a, b]$, $c_i \in \boldsymbol{R}$

		положительного элемента. 3) Сжать массив, удалив из него элементы, удовлетворяющие условию. Освободившиеся в конце массива элементы заполнить нулями.	
8	E(n)	Вычислить: 1) минимальный элемент массива; 2) сумму элементов массива, расположенных между первым и последним положительными элементами. 3) Преобразовать массив, расположив сначала все нулевые элементы, а потом – все остальные.	$e_i \in \boldsymbol{R}$
9	F(40)	Вычислить: 1) номер максимального элемента массива; 2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами. 3) Преобразовать массив так, чтобы в первой его половине располагались элементы, стоящие в нечётных позициях, а во второй – элементы, стоящие в чётных позициях.	$f_i \in \boldsymbol{Z}$
10	G(n)	Вычислить: 1) номер минимального элемента массива; 2) сумму элементов массива, расположенных между первым и вторым отрицательными элементами. 3) Преобразовать массив так, чтобы сначала	$\lvert g_i \rvert \le 1$, $g_i \in \boldsymbol{R}$

		располагались все элементы, удовлетворяющие условию, а потом – все остальные.	
11	Z(30)	1) Найти наибольший и наименьший элементы, поменять их местами. 2) Вычислить среднее геометрическое элементов массива, удовлетворяющих условию 1). 3) Переписать элементы массива Z в новый массив Y, удовлетворяющие условию 2) и подсчитать их количество.	1) $z_i > 0$; 2) $\|z_i\| \le 1$ $z_i \in \boldsymbol{Z}$
12	H(n)	Вычислить: 1) максимальный по модулю элемент массива; 2) сумму элементов массива, расположенных между первым и вторым положительными элементами. 3) Преобразовать массив так, чтобы элементы, равные нулю, располагались после всех остальных.	$h_i \in \boldsymbol{R}$
13	R(20)	Вычислить: 1) минимальный по модулю элемент массива; 2) сумму модулей элементов массива, расположенных после первого элемента, равного нулю. 3) Преобразовать массив так, чтобы в первой его половине располагались элементы, стоящие в чётных позициях, а во второй половине – элементы, стоящие в	$r_i \in \boldsymbol{Z}$

		нечётных позициях.	
14	T(n)	Вычислить: 1) наибольший элемент массива, удовлетворяющий условию 1); 2) сумму модулей элементов массива, расположенных после первого отрицательного элемента. 3) Сжать массив, удалив из него все элементы, удовлетворяющие условию 2). Освободившиеся в конце массива элементы заполнить нулями.	1) $1 \le t_i \le 2$; 2) $t_i \in [a, b]$ $t_i \in \boldsymbol{R}$
15	U(30)	Вычислить: 1) номер максимального по модулю элемента массива; 2) сумму элементов массива, расположенных после первого положительного элемента. 3) Преобразовать массив так, чтобы сначала располагались все элементы, целая часть которых удовлетворяют условию, а потом все остальные.	$[u_i] \in [a, b]$ $u_i \in \boldsymbol{R}$
16	X(n)	Вычислить: 1) наибольший элемент, заменить его значением суммы элементов, предшествующих ему; 2) суммы (раздельно) положительных и отрицательных элементов. 3) Расположить в массиве X сначала положительные, а затем отрицательные	$x_i \in \boldsymbol{R}$

		элементы.	
17	V(40)	Вычислить: 1) количество элементов, удовлетворяющих условию 1); 2) сумму элементов массива, расположенных после максимального элемента. 3) Переместить в массив Y элементы, удовлетворяющие условию 2), массив V сжать и пустующие элементы обнулить.	1) $[v_i] \in [a, b]$; 2) $v_i > 0$ $v_i \in \boldsymbol{R}$
18	W(n)	Вычислить: 1) количество нулевых элементов; 2) сумму элементов массива, расположенных после минимального элемента. 3) Упорядочить элементы массива по возрастанию модулей элементов.	$w_i \in \boldsymbol{R}$
19	A(30)	Вычислить: 1) количество элементов, удовлетворяющих условию; 2) произведение элементов массива, расположенных после максимального по модулю элемента. 3) Переписать элементы массива, чтобы сначала располагались все отрицательные элементы, а потом – все положительные, включая нулевые.	$a_i > c$, $a_i \in \boldsymbol{R}$
20	B(n)	Вычислить: 1) количество отрицательных элементов	$b_i \in \boldsymbol{R}$

		массива; 2) сумму модулей элементов массива, расположенных после минимального по модулю элемента. 3) Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.	
21	Y(40)	Вычислить: 1) наибольший элемент, заменить (нормировать) все элементы массива, поделив их значения на значение наибольшего элемента; 2) сумму элементов массива, расположенных после последнего элемента, равного нулю. 3) Упорядочить массив по возрастанию элементов удовлетворяющих условию, остальные записать в конец массива.	$0.1 \le y_i \le 0.4$, $y_i \in \boldsymbol{R}$
22	C(n)	Вычислить: 1) количество положительных элементов массива; 2) определить сумму элементов массива, кратных трём. 3) Преобразовать массив так, чтобы сначала располагались элементы, удовлетворяющие условию 2), а потом все остальные.	1) c_i /3*3= c_i; 2) $[c_i] \le 1$ $c_i \in \boldsymbol{R}$
23	K(30)	Вычислить: 1) количество элементов, удовлетворяющих условию;	$k_i < c$, $k_i \in \boldsymbol{R}$

		2) сумму целых частей элементов массива, расположенных после последнего отрицательного. 3) Изменить порядок следования элементов массива на обратный.	
24	D(n)	Вычислить: 1) произведение отрицательных элементов массива; 2) сумму положительных элементов массива, расположенных до максимального элемента. 3) Преобразовать массив так, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом все остальные.	$d_i \in \boldsymbol{R}$
25	F(40)	Вычислить: 1) произведение положительных элементов массива; 2) сумму элементов массива, расположенных до минимального элемента. 3) Упорядочить по возрастанию отдельно элементы, стоящие на чётных местах и элементы, стоящие на нечётных местах.	$f_i \in \boldsymbol{R}$

Лабораторная работа № 5.

Обработка матриц.

Цель работы. Овладение навыками алгоритмизации и программирования структур с вложенными циклами, навыками использования приёмов программирования во вложенных циклах, способами ввода и вывода матриц, использование динамических массивов.

Задания для самостоятельной подготовки.

1. Изучить:

– правила организации вложенного цикла с учётом порядка перебора элементов матрицы;

– правила использования приёмов программирования в структурах с вложенными циклами;

– способы ввода и вывода матриц;

– правила организации двумерного динамического массива с помощью операции new.

2. Разработать алгоритм решения в соответствии с заданием.

3. Составить программу решения задачи.

4. Подготовить тестовый вариант программы и исходных данных.

Задание к работе.

1) Обработать в компьютере матрицу в соответствии с вариантом задания. Вывести на печать результаты и исходную матрицу в общепринятом виде. 2) Проверить правильность выполнения программы с помощью тестового примера. 3) Использовать динамический массив для неопределенных размерностей.

Вариант задания	Имя матрицы и размеры	Действия	Условия и ограничения
1	M(m, n)	Улитка. Матрицу заполнить числами от 1 до m·n по спирали, начинающейся в левом верхнем углу и закрученной по часовой стрелке.	$m_{ij} \in \boldsymbol{N}$
2	A(k, k)	Определить: 1) количество строк, не содержащих ни одного нулевого элемента; 2) максимальное из чисел, встречающихся	$a_{ij} \in \boldsymbol{Z}$

		в матрице более одного раза.	
3	A(10, 10)	1) Определить количество столбцов, не содержащих ни одного нулевого элемента. 2) Характеристикой строки матрицы назовём сумму её положительных чётных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.	$a_{ij} \in \boldsymbol{Z}$
4	B(n, m)	Определить: 1) количество столбцов, содержащих хотя бы один нулевой элемент; 2) номер строки, в которой находится самая длинная серия одинаковых элементов.	$b_{ij} \in \boldsymbol{Z}$
5	B(20, 20)	Определить: 1) произведение элементов в тех строках, которые не содержат отрицательных элементов; 2) максимум среди элементов диагоналей, параллельных главной диагонали матрицы.	$b_{ij} \in \boldsymbol{Z}$
6	C(m, n)	Нарастающий итог. Каждый элемент матрицы C заменить суммой элементов подматриц C′(i, j), расположенной в левом верхнем углу матрицы C.	$c_{ij} \in \boldsymbol{R}$
7	C(10, 10)	Определить: 1) сумму элементов в тех столбцах, которые не содержат отрицательных элементов; 2) минимум среди сумм модулей элементов диагоналей, параллельных	$c_{ij} \in \boldsymbol{Z}$

		побочной диагонали матрицы.	
8	D(m, n)	Определить: 1) сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент; 2) номер строк и столбцов всех седловых точек матрицы. Матрица D имеет седловую точку d_{ij}, если является минимальным элементом в i-й строке и максимальным в j-м столбце.	$d_{ij} \in \boldsymbol{Z}$
9	D(8, 8)	1) Для данной матрицы найти такие k, что k-я строка матрицы совпадает с k-м столбцом. 2) Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.	$d_{ij} \in \boldsymbol{Z}$
10	D(m, n)	1) Характеристикой столбца матрицы назовём сумму модулей его отрицательных нечётных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик. 2) Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.	$d_{ij} \in \boldsymbol{Z}$
11	N(10, 15)	Работа комбайнёра. Матрицу N заполнить следующим образом: элементам, находящимся на периферии (по периметру матрицы), присвоить значения 1; периметру оставшейся подматрицы – значение 2 и т.д. до заполнения всей	$n_{ij} \in \boldsymbol{N}$

		матрицы.	
12	X(n, n)	1) В массиве X каждый элемент (кроме граничных) заменить суммой непосредственно примыкающих к нему элементов по вертикали, горизонтали и диагоналям. 2) В преобразованном массиве найти суммы модулей элементов, расположенных ниже главной диагонали.	$x_{ij} \in \boldsymbol{R}$
13	K(10, 10)	1) Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов матрицы K. 2) Найти сумму модулей элементов, расположенных выше главной диагонали.	$k_{ij} \in \boldsymbol{R}$
14	K(n, m)	1) Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду (метод Гаусса). 2) Найти количество строк, среднее арифметическое элементов которых меньше заданной величины (для исходной матрицы).	$k_{ij} \in \boldsymbol{R}$
15	K(10, 15)	1) Уплотнить заданную матрицу, удаляя из неё строки и столбцы, заполненные нулями. 2) Найти номер первой из строк, содержащих хотя бы один положительный	$k_{ij} \in \boldsymbol{Z}$

		элемент.	
16	N(m, m)	1) Для матрицы N найти для каждой строки число элементов кратных пяти, и наибольший из полученных результатов. 2) Упорядочить по возрастанию элементы каждой строки матрицы.	$n_{ij} \in \boldsymbol{Z}$ n_{ij} /5*5=n_{ij}
17	A(m, n)	Осуществить циклический сдвиг элементов в матрице на *k* элементов вправо или вниз (в зависимости от введённого режима). Число *k* может быть больше количества элементов с строке или столбце.	$a_{ij} \in \boldsymbol{R}$ $k \in \boldsymbol{Z}$
18	B(m, n)	Матрицу B заполнить следующим образом. Для заданных *k* и *l* элементу b_{kl} присвоить значение 1; элементам, окаймляющим его (соседним с ним по вертикали, горизонтали и диагоналям) – значение 2; элементам следующего окаймления – значение 3 и т.д. до заполнения всей матрицы.	$k, l \in \boldsymbol{N}$
19	B(10, 10)	1) Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент. 2) Характеристикой строки матрицы назовём сумму её отрицательных чётных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.	$b_{ij} \in \boldsymbol{Z}$
20	B(n, n)	1) Упорядочить строки матрицы по возрастанию количества одинаковых элементов в каждой строке.	$b_{ij} \in \boldsymbol{Z}$

		2) Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.	
21	P(n, n)	1) В каждом столбце и каждой строке матрицы содержится по одному нулевому элементу. Перестановкой строк добиться расположение всех нулей по главной диагонали матрицы. 2) Найти строки с наибольшей и наименьшей суммой элементов.	$p_{ij} \in \boldsymbol{R}$
22	P(10, 10)	1) Путем перестановкой элементов матрицы P добиться того, чтобы её максимальный элемент находился в левом верхнем углу, следующий по величине – в позиции (2, 2), следующий по величине – в позиции (3, 3) и т.д., заполнив таким образом, получить на главной диагонали убывающие по строкам максимальные элементы . 2) Найти номер первой из строк, не содержащих ни одного положительного элемента.	$p_{ij} \in \boldsymbol{R}$
23	S(m, n)	Определить: 1) количество строк, содержащих хотя бы один нулевой элемент; 2) номер столбца, в котором находится самая длинная серия одинаковых элементов.	$s_{ij} \in \boldsymbol{Z}$
24	S(n, n)	Определить: 1) сумму элементов в тех строках, которые	$s_{ij} \in \boldsymbol{Z}$

		не содержат отрицательных элементов; 2) минимум среди сумм элементов диагоналей, параллельных главной.	
25	Y(m, n)	Определить: 1) количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент; 2) номера строк и столбцов всех седловых точек матрицы. Матрица Y имеет седловую точку y_{ij}, если y_{ij} является минимальным в i-й строке и максимальным в j-м столбце.	$y_{ij} \in \boldsymbol{Z}$

Лабораторная работа № 6.

Обработка символьных данных и текстовых файлов.

Цель работы. Овладение навыками алгоритмизации и программирования задач, обрабатывающих символьные данные; ввод данных в файл; чтение данных из файла; использование функций обработки символьных данных.

Задания для самостоятельной подготовки.

1. Изучить:

– правила записи символьных данных (констант, переменных, массивов и их описание, инициализация);

– способы ввода и вывода символьных данных;

– способы обработки символьных данных;

– использование функций обработки символьных данных;

– стандартные программные решения обработки символов;

– возможности языка С++ по обработке файла с последовательной организацией: запись данных в файл, чтение из файла, добавление записей в файл, корректировка записей и т.п.

2. Разработать алгоритм решения в соответствии с заданием.

3. Составить программу решения задачи.

4. Подготовить тестовый вариант программы и исходных данных.

Задание к работе.

1.Исходные данные для всех вариантов должны находиться в текстовых файлах, если в задании не оговорено другое условие о вводе данных

2.Проверить правильность выполнения программы с помощью тестового варианта.

Варианты заданий	Условие задачи
1	Отредактировать предложение, удаляя из него все повторяющиеся символы.
2	В строке, содержащей последовательность слов, найти конец предложения, обозначенный символом “точка”. В следующем слове первую строчную букву заменить на прописную.
3	В строке найти все числа в десятичной системе счисления, сформировать новую строку, в которой заменить их соответствующим представлением в шестнадцатеричной системе.
4	Заменить в строке принятое в С обозначение символа с заданным кодом (например, \101) на сам символ (в данном случае – А).
5	Переписать в выходную строку слова из входной строки в порядке возрастания их длины.
6	Преобразовать строку, содержащую выражение на С с операциями (=, ==, !=, *a*+=, *a*-=), в строку, содержащую эти же операции с синтаксисом языка Паскаль (:=, =, #, *a*=*a*+, *a*=*a*-).
7	Удалить из строки комментарии вида “/*…*/”. Игнорировать вложенные комментарии.
8	Проверить, имеется ли в заданной строке баланс открывающихся и закрывающихся скобок.
9	Заменить в строке последовательность одинаковых символов (не пробелов) на десятичное число, соответствующее их количеству, и сам символ (т.е., например, “abcdaaaaa xyznnnnnnn” на ”abcd5a

	xyz7n”).
10	“Перевернуть” в строке все слова (например, “Жили были и не тужили” – “илиЖ илыб и ен илижут”).
11	Для встречающихся в заданной строке пар рядом расположенных символов указать, сколько раз встречаются каждое из таких двухсимвольных сочетаний.
12	Отредактировать предложение, удаляя из него лишние пробелы, оставляя только по одному пробелу между словами.
13	В заданной строке указать слово, в котором доля гласных (a, e, i, o) максимальна.
14	Для каждого символа введённого с клавиатуры слова указать, сколько раз он встречается в строке. Сообщение об одном символе должно печататься не более одного раза.
15	Найти самое длинное симметричное слово заданной строки, например, ABBA.
16	Отредактировать заданную строку, заменяя многоточия точкой.
17	В заданной строке найти самое короткое и самое длинное слово.
18	Из заданной строки выбрать и напечатать только те символы, которые встречаются в нём только один раз (в том порядке, в котором они встречаются в строке).
19	В строке заменить последовательность символов X на A и подсчитать число произведенных замен.
20	В строке удалить символ “,” и подсчитать число удалённых символов.
21	Из строки выбрать числа и записать в массив int (целочисленный).
22	Удалить из строки символы пробела и подсчитать длину полученной строки.
23	В строке заменить символ пробела на символы “,”. Если в тексте встречается несколько пробелов подряд, то вместо них поставить одну запятую. Определить длину полученной строки.
24	Вывести на экран из текста только предложения, состоящие из заданного количества слов.
25	Вывести на экран из текста сначала вопросительные, а затем повествовательные предложения.

Лабораторная работа № 7.

Программирование с использованием функций пользователя.

Цель работы. Овладение навыками алгоритмизации и программирования задач с использованием функций пользователя различных видов, овладения написания функций и обращения к ним, выбора параметров функций.

Задания для самостоятельной подготовки.

1. Изучить:
   – правила записи (объявления) функций различных видов и способов обращений к ней;
   – способы передачи параметров в функцию;
   – объявление прототипов функций;
   – локальные и статические переменные в функциях;
   – выход из функций;
   – перегрузка функций;
   – функции типа inline.
2. Разработать алгоритм решения в соответствии с заданием.
3. Составить программу решения задачи.
4. Подготовить тестовый вариант программы и исходных данных.

Задание к работе.

1. Выполнить раздельно задания четвертой, пятой и шестой лабораторных работ оформив каждый пункт задания в виде функции. Все необходимые данные для функций должны передаваться в качестве параметров. Использование глобальных переменных в функциях не допускается.

2. Проверить правильность работы программы с помощью тестового варианта.

3. Выполнить на компьютере программу, использующую функции в соответствии с номером указанным в таблице. Использование библиотечных функций не допускается.

Варианты заданий	Условие задачи	Примечание
1	Вычислить z = (sign(x)+sign(y))·sign(x+y) где $\text{sign}(a) = \begin{cases} -1, & \text{при } a < 0 \\ 0, & \text{при } a = 0 \\ 1, & \text{при } a > 0 \end{cases}$ При решении этой задачи: а) не использовать функцию sign; б) определить и использовать функцию sign.	$x, y, a \in \boldsymbol{R}$
2	Вычислить величину $\text{sh}(x)\cdot\text{tg}(x+1)-\text{tg}^2(2+\text{sh}(x-1))$.	$x \in \boldsymbol{R}$
3	Описать функцию степень (x, n), вычисляющую (через умножение) величину x^n, и использовать её для вычисления $b=2{,}7^k + (a+1)^{-5}$.	$a, x \in \boldsymbol{R}$, $n \in \boldsymbol{N}$
4	Даны три числа a, b и c. Определить их наибольший общий делитель.	$a, b, c \in \boldsymbol{N}$
5	Даны отрезки a, b, c и d. Для каждой тройки этих отрезков можно построить треугольник и напечатать площадь этого треугольника. (Определить функцию PrintS(x, y, z) печатающую площадь треугольника со сторонами x, y, z, если такой треугольник существует.)	$a, b, c, d \in \boldsymbol{N}$
6	Описать функцию целых параметров, которая приводит дробь a/b к несократимому виду p/q и использовать её для приведения дроби 1+1/2+1/3+…+1/20 к несократимому виду.	$a, b, p, q \in \boldsymbol{Z}$
7	Найти сумму кубов всех корней уравнения $\mathbf{e}x^3 - \text{pi}x^2 - (2\mathbf{e}+1)x + 2\text{pi} = 0$ (pi=3,1415927, **e**=2,7182818), вычисленных с точностью 0,0001.	
8	Даны длины a, b, и c некоторого треугольника. Найти медианы треугольника, сторонами	

	которого являются медианы исходного треугольника. (Длина медианы, проведённой к стороне a, равна $0{,}5\cdot(2b^2+2c^2-a^2)^{1/2}$.)	
9	Даны три слова, в каждом из которых от 1 до 6 строчных латинских букв и за каждым из которых следует пробел. Напечатать эти слова в алфавитном порядке.	
10	По заданным вещественным числам $a_0, a_1,\ldots, a_{30}$, $b_0, b_1,\ldots, b_{30}$, $c_0, c_1,\ldots, c_{30}$, x, y, z вычислить величину $\dfrac{(a_0x^{30} + a_1x^{29} + \ldots + a_{30})^2 - (b_0y^{30} + b_1y^{29} + \ldots + b_{30})}{c_0(x+z)^{30} + c_1(x+z)^{29} + \ldots + c_{30}}$.	$a_i, b_i, c_i \in \boldsymbol{R}$, $i = 0..30$
11	По заданным 20-элементным массивам x и y вычислить $U = \begin{cases} \sum\limits_{i=1}^{20} x_i^2, \text{при} \sum\limits_{i=1}^{15} x_i y_i > 0 \\ \sum\limits_{i=1}^{20} y_i^2, \text{при} \sum\limits_{i=1}^{15} x_i y_i < 0 \end{cases}$.	$x_i, y_i \in \boldsymbol{Z}$
12	По заданным 50-элементным массивам a, b и c вычислить $t = \begin{cases} \dfrac{\min(b_i)}{\max(a_i)} + \dfrac{\max(c_i)}{\min(b_i + c_i)}, \text{при } \max(a_i) < \max(b_i) \\ \max(b_i + c_i) + \min(c_i), \text{в противном случае} \end{cases}$.	$a_i, b_i, c_i \in \boldsymbol{R}$, $i = 1..50$
13	Дано n чисел. Упорядочить их по возрастанию методом фон Неймана: завести два массива A и B и записать исходные числа в A; упорядочить пары соседних чисел (a_1 и a_2, a_3 и a_4 и т.д.) и записать их в B; взять из B по две соседние упорядоченные пары и, слив их в упорядоченные четвёрки, снова записать в A; затем каждые две соседние четвёрки из B слить в упорядоченные восьмёрки и перенести в A; и т.д.	n=100, $a_i \in \boldsymbol{R}$

14	В точках 1, 2,…, k, где k – заданное число, напечатать значения следующих функций: $\varphi(n)$ – количество целых чисел от 1 до n, взаимно простых с числом n; $\tau(n)$ – количество положительных делителей числа n; $\pi(n)$ – количество простых чисел, не превосходящих n.	$k = 2,\ldots,70$
15	Даны две квадратные матрицы 10-го порядка. Определить, можно ли отражениями относительно главной и побочной диагоналей преобразовать одну из них в другую.	Элементы целых матриц
16	Даны матрицы A, B и C размером 10×20. Вычислить величину $\frac{\|\| A \|\| + \|\| B \|\| + \|\| C \|\|}{\|\| A + B + C \|\|}$, где $\|\| D \|\| = \max_j \| d_{1,j} \| + \max_j \| d_{2,j} \| + \ldots + \max_j \| d_{20,j} \|$.	$a_i, b_i, c_i \in \boldsymbol{R}$
17	Дано число p и квадратные матрицы A, B и C 4-го порядка. Получить $(ABC)^p$.	$a_i, b_i, c_i \in \boldsymbol{R}$, $p \in \boldsymbol{N}$
18	По числам ε и t вычислить с точностью ε величину $\sqrt[4]{1 - \frac{\cos^4 t}{4}} + \sqrt[5]{1 + \frac{\operatorname{arctg} t}{2}} \cdot \sqrt[9]{\frac{1}{3 + t^2}}$. Для вычисления корней использовать следующий ряд Тейлора: $(1 + x)^a = 1 + \frac{a}{1!}x + \frac{a(a-1)}{2!}x^2 + \frac{a(a-1)(a-2)}{3!}x^3 + \ldots$	ε, $t \in \boldsymbol{R}$, ε > 0, a>0, $\|x\| \le 1$
19	Написать функцию, которая вычисляет функцию a^b. Числа a и b могут быть любыми дробными положительными числами.	
20	По числу a вычислить величину $\frac{\sqrt[3]{a} - \sqrt[6]{a^2 + 1}}{1 + \sqrt[7]{3 + a}}$. Корни $y = \sqrt[k]{x}$ вычислять с точностью ε =0,0001 по	$a \in \boldsymbol{R}$, a>0, k=0, 1, 2,…

	следующей итерационной формуле: y_0=1; $y_{n+1} = y_n + \frac{x / y_n^{k-1} - y_n}{k}$, приняв за ответ приближение y_{n+1}, для которого $\lvert y_{n+1}-y_n \rvert < \varepsilon$.	
21	Два натуральных числа называются “дружественными”, если каждое из них равно сумме всех делителей другого, за исключением его самого (таковы, например, числа 220 и 284). Напечатать все пары “дружественных” чисел, не превосходящих заданного числа n.	$n \in \boldsymbol{N}$
22	Два простых числа называются “близнецами”, если они отличаются друг от друга на 2 (таковы, например, числа 41 и 43). Напечатать все пары чисел “близнецов” из отрезка [n; 2n].	$n \in \boldsymbol{N}$, n>1
23	Описать функцию next без параметров, которая считывает из входного файла первую литеру, отличную от пробела, и объявляет её своим значением. Использовать эту функцию для подсчёта k – количества отличных от пробела литер текста, который задан во входном файле и за которым следует точка.	
24	Описать логическую функцию перестановки (x, y), проверяющую, можно ли, переставив литеры слова x, получить слово y.	
25	Напечатать все цифры десятичной записи чисел 2^{500} и 1!+2!+…+100! (Рекомендация: представить “длинные” натуральные числа в виде массивов из цифр и реализовать нужные операции над ними.)	

Лабораторная работа №8.

Программирование с использованием типа данных структурированной.

Цель работы – овладение навыками алгоритмизации и программирования задач с использованием данных типа структур.

Задания для самостоятельной подготовки.

1. Изучить:
   - основную терминологию, связанную со структурами, методы доступа к полям структур.
   - определения массива структур и указателя на структуру;
   - инициализация структур;
   - объединение (union);
   - использование структур в качестве аргументов функции, передача структур по ссылке, по указателям.
2. Разработать алгоритм решения в соответствии с заданием.
3. Составить программу решения задачи в виде набора функций.
4. Подготовить тестовый вариант программы и исходных данных.

Задание к работе.

Определить структурированный тип и набор функций для работы с таблицей, реализованной в массиве структур. Выбрать способ организации массива: с отметкой свободных элементов специальным значением полем либо с перемещением их к концу массива (уплотнение данных). Функции должны работать с массивом структур или с отдельной структурой через указатели, а также при необходимости возвращать указатель на структуру. В перечень функций входят:

- очистка структурированных переменных;
- поиск свободной структурированной переменной;
- ввод элементов (полей) структуры с клавиатуры;
- вывод элементов (полей) структуры по запросу с клавиатуры;
- поиск в массиве структуры с минимальным значением заданного поля;
- сортировка массива структур в порядке возрастания заданного поля;

- поиск в массиве структуры элемента с заданным значением поля или с наиболее близким к нему по значению;
- удаление заданного элемента;
- изменение (редактирование) заданного элемента;
- сохранение содержимого массива структур в текстовом файле и загрузка из текстового файла;
- вычисление с проверкой и использованием всех элементов массива по заданному условию и формуле (например, общая сумма на всех счетах) – даётся индивидуально.

Вариант задания	Перечень полей структурированной переменной:
1.	Фамилия И.О., номер группы, успеваемость (массив из пяти элементов).
2.	Фамилия И.О., дата рождения, адрес.
3.	Название пункта назначения рейса, номер рейса, тип самолёта.
4.	Фамилия И.О., номер счёта, сумма на счёте, дата последнего изменения.
5.	Фамилия И.О., название занимаемой должности, год поступления на работу.
6.	Номер страницы, номер строки, текст изменения строки, дата изменения.
7.	Название пункта назначения, номер поезда, время отправления.
8.	Название экзамена, дата экзамена, фамилия преподавателя, количество оценок, оценки.
9.	Название начального пункта маршрута, название конечного пункта маршрута, номер маршрута.
10.	Фамилия И.О., номер зачетной книжки, факультет, группа.
11.	Фамилия Имя, знак Зодиака, день рождения.
12.	Фамилия И.О., номер читательского билета, название книги, срок возврата.
13.	Наименование товара, цена, количество, процент торговой надбавки.
14.	Фамилия Имя, номер телефона, день рождения.
15.	Номер рейса, пункт назначения, время вылета, дата вылета,

стоимость билета.

16. Расчётный счёт плательщика, расчётный счёт получателя, перечисляемая сумма в рублях.
17. Фамилия И.О., количество оценок, оценки, средний бал.
18. Фамилия И.О., дата поступления, дата отчисления.
19. Регистрационный номер автомобиля, марка, пробег.
20. Фамилия И.О., количество переговоров (для каждого – дата и продолжительность).
21. Номер телефона, дата разговора, продолжительность, код города.
22. Номер поезда, пункт назначения, дни следования, время прибытия, время стоянки.
23. Название кинофильмов, сеанс, стоимость билета, количество зрителей.
24. Название товара, название магазина в котором продаётся товар, стоимость товара в руб.
25. Фамилия И.О. рабочего, наименование цеха, размер заработной платы за месяц.

3 семестр

Лабораторная работа 1

Вычисление функции с помощью разложения в ряд

Вычислить и вывести на экран в виде таблицы значения функции, заданной с помощью ряда Тейлора, на интервале от $x_{нач}$ до $x_{кон}$ с шагом dx с точностью ε. Таблицу снабдить заголовком и шапкой. Каждая строка таблицы должна содержать значение аргумента, значение функции с помощью ряда, с помощью функции библиотеки и количество просуммированых членов ряда.

1. $\ln\frac{x+1}{x-1} = 2\sum_{n=0}^{\infty}\frac{1}{(2n+1)x^{2n+1}} = 2(\frac{1}{x}+\frac{1}{3x^3}+\frac{1}{5x^5}+...) \quad |x|>1$
2. $e^{-x} = \sum_{n=0}^{\infty}\frac{(-1)^n x^n}{n!} = 1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-... \quad |x|<\infty$
3. $e^{x} = \sum_{n=0}^{\infty}\frac{x^n}{n!} = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}-... \quad |x|<\infty$
4. $\ln(x+1) = \sum_{n=0}^{\infty}\frac{(-1)^n x^{n+1}}{n+1} = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}-... \quad -1<x\le 1$
5. $\ln\frac{1+x}{1-x} = 2\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1} = 2(x+\frac{x^3}{3}+\frac{x^5}{5}+...) \quad |x|<1$

6. $\ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} = -(x + \frac{x^2}{2} + \frac{x^4}{4} + ...) \quad -1 \le x < 1$

7. $arcctg\,x = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} x^{2n+1}}{2n+1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} - ... \quad |x| \le 1$

8. $arctg\,x = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} ... \quad x > 1$

9. $arctg\,x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + ... \quad |x| \le 1$

10. $Arth\,x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + ... \quad |x| < 1$

11. $Arth\,x = \sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + ... \quad |x| > 1$

12. $arctg\,x = -\frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + ... \quad x < -1$

13. $e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - ... \quad |x| < \infty$

14. $\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... \quad |x| < \infty$

15. $\frac{\sin x}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} - ... \quad |x| < \infty$

16. $\ln x = 2\sum_{n=0}^{\infty} \frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}} = 2(\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + ...) \quad x > 0$

17. $\ln x = \sum_{n=0}^{\infty} \frac{(-1)^n (x-1)^{n+1}}{(n+1)} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} + ...) \quad 0 < x \le 2$

18. $\ln x = \sum_{n=0}^{\infty} \frac{(x-1)^{n+1}}{(n+1)(x+1)^{n+1}} = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + ...) \quad x > \frac{1}{2}$

19. $\arcsin x = x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot ... \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot ... \cdot 2n \cdot (2n+1)} = x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} ... \quad |x| < 1$

20. $\arccos x = \frac{\pi}{2} - (x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot ... \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot ... \cdot 2n \cdot (2n+1)}) = \frac{\pi}{2} - (x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} ...) \quad |x| < 1$

Лабораторная работа 2

Алгоритмы

Выполнить задания с использованием везде, где это возможно, контейнерных классов и алгоритмов библиотеки:

Вариант 1

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. сумму отрицательных элементов массива;
2. произведение элементов массива, расположенных между максимальным и минимальным элементами.

Упорядочить элементы массива по возрастанию.

Вариант 2

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. сумму положительных элементов массива;
2. произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию.

Вариант 3

В одномерном массиве, состоящем из n целых элементов, вычислить:

1. произведение элементов массива с четными номерами;
2. сумму элементов массива, расположенных между первым и последним нулевыми элементами.

Преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом – все отрицательные (элементы, равные 0, считать положительными).

Вариант 4

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. сумму элементов массива с нечетными номерами;
2. сумму элементов массива, расположенных между первым и последним отрицательными элементами.

Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 5

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. максимальный элемент массива;
2. сумму элементов массива, расположенных до последнего положительного элемента.

Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a,b]. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 6

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. минимальный элемент массива;
2. сумму элементов массива, расположенных между первым и последним положительными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, равные нулю, а потом – все остальные.

Вариант 7

В одномерном массиве, состоящем из n целых элементов, вычислить:

1. номер максимального элемента массива;
2. произведение элементов массива, расположенных между первым и вторым нулевыми элементами.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях.

Вариант 8

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. номер минимального элемента массива;
2. сумму элементов массива, расположенных между первым и вторым отрицательными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные.

Вариант 9

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. максимальный по модулю элемент массива;
2. сумму элементов массива, расположенных между первым и вторым положительными элементами.

Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

Вариант 10

В одномерном массиве, состоящем из n целых элементов, вычислить:

1. минимальный по модулю элемент массива;
2. сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине — элементы, стоявшие в нечетных позициях.

Вариант 11

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. номер минимального по модулю элемента массива;
2. сумму модулей элементов массива, расположенных после первого отрицательного элемента.

Сжать массив, удалив из него все элементы, величина которых находится в интервале [a,b]. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 12

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. номер максимального по модулю элемента массива;
2. сумму элементов массива, расположенных после первого положительного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a,b], а потом – все остальные.

Вариант 13

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. количество элементов массива, лежащих в диапазоне от A до B;
2. сумму элементов массива, расположенных после максимального элемента.

Упорядочить элементы массива по убыванию модулей элементов.

Вариант 14

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. количество элементов массива, равных 0;
2. сумму элементов массива, расположенных после минимального элемента.

Упорядочить элементы массива по возрастанию модулей элементов.

Вариант 15

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. количество элементов массива, больших C;
2. произведение элементов массива, расположенных после максимального по модулю элемента.

Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом – все положительные (элементы, равные 0, считать положительными).

Вариант 16

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. количество отрицательных элементов массива;
2. сумму модулей элементов массива, расположенных после минимального по модулю элемента.

Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

Вариант 17

В одномерном массиве, состоящем из n целых элементов, вычислить:

1. количество положительных элементов массива;
2. сумму элементов массива, расположенных после последнего элемента, равного нулю.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом — все остальные.

Вариант 18

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. количество элементов массива, меньших C;
2. сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом – все остальные.

Вариант 19

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. произведение отрицательных элементов массива;
2. сумму положительных элементов массива, расположенных до максимального элемента.

Изменить порядок следования элементов в массиве на обратный.

Вариант 20

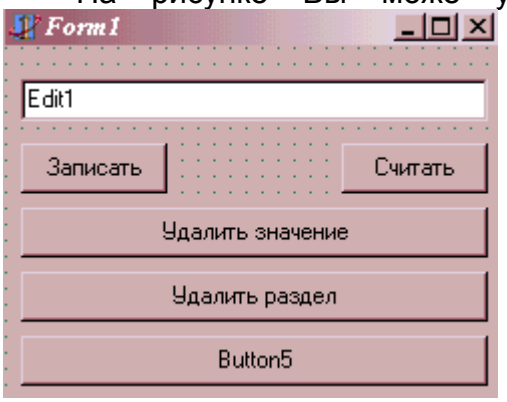
В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1. произведение положительных элементов массива;
2. сумму элементов массива, расположенных до минимального элемента.

Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

Лабораторная работа № 03
Работа с реестром

На рисунке Вы може увидеть форму, которая нам понадобиться.



Прежде чем приступать к программированию необходимо в разделе uses добавить модуль registry.

Начнём с записи в реестр. Создайте событие *OnClick* для первой кнопки (у меня она с именем "Записать") и напиши там следующее:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 RegIni:TRegIniFile;
begin
 RegIni:=TRegIniFile.Create('Software');
 RegIni.OpenKey('VR-online', true);
 RegIni.WriteString('Razd', 'Param', Edit1.Text);
 RegIni.Free;
end;
```

Первая строка (RegIni:=TRegIniFile.Create('Software')) создаёт переменную RegIni типа TRegIniFile, через которую мы будем работать с реестром. В этом объекте (TRegIniFile) реализованы все необходимые нам функции, которые позволяют получить очень простой доступ к реестру. В скобках указываем имя подраздела, с которым собираемся работать. Сразу возникает вопрос: а внутри какого основного раздела Delphi будет искать подраздел "Software"? Ответ: HKEY_CURRENT_USER. Этот раздел используется по умолчанию.

Итак, первая строка инициализирует реестр и текущем разделом становиться HKEY_CURRENT_USER\ Software. Если бы мы написали нашу первую строку так: RegIni:=TRegIniFile.Create(' Network') то текущим разделом был бы уже: HKEY_CURRENT_USER\ Network.

В принципе, мы уже можем приступать к записи, но мы решили открыть ещё подраздел: RegIni.OpenKey('VR-online', true). OpenKey - открывает следующий подраздел, имя которого указано в качестве первого параметра в скобках. Второй параметр означает: создавать ли раздел если его нет?. У нас стоит значение true - это значит, что если такого подраздела нет, то он будет создан. Если поставить false, то ничего создаваться не будет.

Итак, теперь текущий раздел у нас HKEY_CURRENT_USER\ Software\ VR-online. Давай уже что-нибудь запишем: RegIni.WriteString('Razd', 'Param', Edit1.Text). Функция WriteString записывает строку в текущий раздел реестра. Первый параметр - имя подраздела в текущем разделе, т.е. наше значение будет записываться в HKEY_CURRENT_USER\ Software\ VR-online\ Razd. Обязательно

учитывай это. Второй параметр - имя записываемого параметра, а третий - значение параметра. В качестве значения используем строку записанную в Edit1.

Последняя строка RegIni.Free уничтожает созданную нами переменную.

Запись в реестре мы сделали. Можно запустить прогу и проверить на практике, как произошла эта запись.

Для записи в реестр мы использовали процедуру WriteString. Но она не единственная, есть ещё WriteBool (для записи булево значения) и WriteInteger (для записи целого значения). У всех у них одинаковое количество параметров и все они похожи: первый означает имя подраздела, второй - имя параметра, третий - значение. Вот пример их использования.

```
RegIni.WriteInteger('Razd', 'Param', 1);
RegIni.WriteBool('Razd', 'Param', true);
```

Теперь перейдём к чтению параметра. Создай событие *OnClick* для второй кнопки (у меня она с именем "Считать") и напиши там:

```
procedure TForm1.Button2Click(Sender: TObject);
var
 RegIni:TRegIniFile;
begin
 RegIni:=TRegIniFile.Create('Software');
 RegIni.OpenKey('VR-online', true);
 Edit1.Text:=RegIni.ReadString('Razd', 'Param', 'Default');
 RegIni.Free;
end;
```

Практически всё здесь, как и при записи. Точно так же создаётся переменная и открывается раздел. Затем идёт функция RegIni.ReadString, которая непосредственно читает значения. В качестве первого параметра выступает имя раздела, второй - имя параметра, и третий - значение по умолчанию. Функция возвращает считанное значение, а если его нет, то функция вернёт значение по умолчанию (то, что ты указал в третьем параметре 'Default').

Для чтения значений из реестра есть ещё две функции: ReadBool (читать булево значение) и ReadInteger (читать число). Вот пример их использования:

```
Edit1.Text:=RegIni.ReadInteger('Razd', 'Param', 1);
Edit1.Text:=RegIni.ReadBool('Razd', 'Param', true);
```

Дальше познакомимся с функциями объекта TRegIniFile, с которым мы работаем.

DeleteValue - удаляет параметр. В скобках Вы должны указать имя параметра. Например, для того чтобы удалить созданный нами параметр (Param), мы должны написать:

```
RegIni.OpenKey('VR-online', true);
RegIni.OpenKey('Razd', true);
RegIni.DeleteValue('Param');
```

DeleteKey - удаляет раздел. В скобках указывается имя раздела для удаления. У объекта TRegIniFile эта функция имеет два параметра, но лучше использовать эту же функцию его предка TRegistry. Для этого нужно написать TRegistry(RegIni). DeleteKey('Razd');.

RootKey - это свойство объекта TRegIniFile, которое указывает на головной раздел, который сейчас используется. По умолчанию это HKEY_CURRENT_USER. Чтобы изменить это значение, нужно просто присвоить другое. Нарпимер:

```
procedure TForm1.Button5Click(Sender: TObject);
```

```
var
 RegIni:TRegIniFile;
begin
 RegIni:=TRegIniFile.Create('Software');
 RegIni.RootKey:=HKEY_LOCAL_MACHINE;
 RegIni.OpenKey('VR-online', true);
 RegIni.WriteString('Razd', 'Param', Edit1.Text);
 RegIni.Free;
end;
```

Основные вопросы от программистов Windows NT типа: "Почему, когда я создаю ключи в реестре с помощью TRegIniFile все ключи строковые, хотя я хочу создать целое число". Некоторые думают, что это глюк Delphi, но это не глюк, а специфика TRegIniFile. Если Вы хотите, чтобы ваши программы правильно работали с реестром в любой ОС, то обратите внимание на TRegIniFile.

Объект TRegIniFile работает с реестром на высоком уровне так, как будто он работает с ini файлом. Если вы не застали Win3.11, то напомню, ini файлы - это файлы конфигурации, которые уже устарели и используются только для совместимости со старым софтом. Сейчас для этих целей есть реестр. TRegIniFile - объект, который позволяет работать с реестром как с ini файлом. А так, как файл может содержать только строки, то и объект работает со строками. Так что реально, TRegIniFile все данные сохраняет и читает из реестра в виде строк. Только после чтения строки происходит преобразование в формат указанный тобой.

Реестр, в отличии от ini файлов - это база данных. Поэтому она позволяет реально хранить не только строки, но и числа, и данные, и логические операторы. Если вы хочешь, чтобы данные сохранялись и читались в виде типов отличных от строк, то ты должен работать через TRegistry. Он так же объявлен в модуле registry, поэтому ты должен подключать этот модуль в раздел uses.

Работа с TRegistry проктически такая же, как и с TRegIniFile:

```
var
 Reg:TRegistry;
begin
 Reg:=TRegIniFile.Create; //Инициализация
 Reg.RootKey:=HKEY_LOCAL_MACHINE;//Выбираю корень реестра
                        //По умолчанию это HKEY_CURRENT_USER

 Reg.OpenKey('SYSTEM', true);          //Открываю раздел SYSTEM
 Reg.OpenKey('CurrentControlSet', true);//Открываю раздел CurrentControlSet
 Reg.OpenKey('Control', true);
 Reg.OpenKey('Session Manager', true);
 Reg.OpenKey('Memory Management', true);

 //Записываю параметр ClearPageFileAtShutdown
 Reg.WriteInteger('ClearPageFileAtShutdown', 112);

 //Читаю параметр Hidden
 RegIni.ReadInteger('Hidden')
```

```
 Reg.CloseKey;//Закрываю ключ
 Reg.Free;    //Освобождаю объект
end;
```

В этом примере чтение и запись будет происходить с числами, а не со строками с последующим преобразованием в число.

Задание:

1. Используя пример, создайте программу считывающую и записывающую данные в реестр.
2. Используя пример, создайте программу, выводящую список программ установленных на компьютере.

Лабораторная работа № 04

Вызов API Windows в Delphi

Разработчики библиотеки визуальных компонент (VCL) Delphi очень серьезно поработали над ее проектированием и воплощением в реальность. Как показывает практика, стандартного набора объектов обычно достаточно для создания реальных приложений. И, что более существенно, им (разработчикам) удалось решить очень сложную задачу, стоящую перед создателями любой среды визуального программирования - скрыть от программиста сложность и трудоемкость программирования в Windows и, в то же время, не лишать его возможности доступа к тем богатым возможностям системы, которые предоставляет Windows API.

На примерах показано, как с помощью вызовов Windows API можно управлять объектами из VCL. Кроме того, здесь же можно найти описание некоторых программных трюков, которые помогут придать вашей программе оригинальный вид.

1.1 Стандартная страница Палитры Компонент

Компоненты, расположенные на странице “Standard”, представляют из себя объектную оболочку для стандартных управляющих элементов Windows. Поэтому для них существуют ограничения, накладываемые самой системой. Например, 32Кб – максимальный размер текста в TMemo.

TMainMenu, TPopupMenu

Добавление картинки (BitMap) в меню.

Для добавления в меню картинки можно использовать функцию API Windows *SetMenuItemBitmaps()*, например, следующим образом:

```
…
implementation
 …
var
  BMP1, BMP2 : TBitMap;
…

procedure TForm1.FormCreate(Sender: TObject);
begin
  BMP1:=TBitMap.Create;
  BMP1.LoadFromFile('c:\images\uncheck.bmp');
  BMP2:=TBitMap.Create;
  BMP2.LoadFromFile('c:\images\check.bmp');
  SetMenuItemBitmaps(File1.Handle, 1, MF_BYPOSITION,
          BMP1.Handle, BMP2.Handle);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  BMP1.Free;
  BMP2.Free;
end;
```

File1 это объект класса TMenuItem - пункт меню “File”. Значения параметров при вызове функции можно посмотреть в справочнике по Windows API.

При уничтожении меню освобождения связанных с ним картинок не происходит и их надо уничтожать вручную.

Вторая картинка BMP2 отображается рядом с пунктом меню, когда он выбран (*Checked=True*).

TMemo

Компонент класса TMemo может содержать до 32К текста (для Windows 3.x) вследствие ограничения Windows. В Delphi 2.0 предел увеличен до 64К (в декабрьской бета-версии).

I. Определение позиции каретки в TMemo.

Можено использовать сообщения Windows API EM_LINEFROMCHAR и EM_LINEINDEX для определения текущей позиции каретки.

```
procedure TForm1.ShowPosition;
var
  LineNum: longint;
  CharNum: longint;
begin
 LineNum:= Memo1.Perform(EM_LINEFROMCHAR, Memo1.SelStart,0);
 CharNum:= Memo1.Perform(EM_LINEINDEX, LineNum, 0);
 Label1.Caption := 'Line : '+ IntToStr(LineNum+1);
 Label2.Caption := 'Position :' + IntToStr((Memo1.SelStart -CharNum)+1);
end;
```

Метод *Perform*, определенный в классе TControl, посылает сообщение своему же объекту, то есть его использование имеет тот же эффект, что и вызов функции API SendMessage():

```
SendMessage(Memo1.Handle,EM_LINEFROMCHAR,    Memo1.SelStart,0);
```

II. Операция UnDo в TMemo.

Отмена последнего редактирования (операция UnDo) в объекте класса TMemo выполняется так же с помощью сообщений, посылаемых в данный объект:

```
procedure TForm1.UnDoClick(Sender: TObject);
begin
  if Memo1.Perform(EM_CANUNDO, 0, 0)<>0 then
    Memo1.Perform(EM_UNDO, 0, 0);
end;
```

В справочнике по Windows API описаны сообщения, которые можно послать в объект TMemo для управления его поведением. Кроме вышеназванных, имеется еще несколько полезных:

EM_EMPTYUNDOBUFFER Сбрасывает флажок UnDo
EM_GETHANDLE Получает указатель на буфер с текстом

EM_LINESCROLL Прокрутка текста в окне TMemo
EM_SETHANDLE Установка указателя на буфер с текстом
EM_SETTABSTOPS Устанавливает табуляцию в окне с текстом

TListBox, TComboBox

Windows накладывает ограничение на количество элементов в списке этих управляющих элементов. В случае Windows 3.x это количество равно 5440, в Windows'95 - 32767.

I. Как получить горизонтальную прокрутку (scrollbar) в ListBox?

Так же как в случае с TMemo, здесь можно использовать сообщения. Например, сообщение может быть отослано в момент создания формы:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Perform(LB_SETHORIZONTALEXTENT, 1000, Longint(0));
end;
```

Второй параметр в вызове - ширина прокрутки в точках.

II. Вставка графики в ListBox.

У класса *TListBox* (и *TComboBox* тоже) есть свойство *Style,* определяющее порядок рисования объекта. По-умолчанию оно установлено в *lbStandard* и за внешний вид объекта отвечает Windows. Если установить это значение в *lbOwnerDrawFixed* или *lbOwnerDrawVariable*, то можно несколько разнообразить внешний вид объекта. Давайте построим для примера ListBox, отображающий названия файлов формата .BMP из какой-либо директории вместе с их картинками.

Прежде всего, оказывается, что вовсе не нужно заполнять ListBox вручную именами файлов, для этого достаточно послать ему сообщение:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  s : String;
begin
  s:='c:\windows\*.bmp'#0;
  ListBox1.Perform(LB_DIR, DDL_READWRITE, Longint(@s[1]));
end;
```

Здесь мы указали ListBox'у, какие файлы требуется отображать.

Далее, как уже было сказано, свойство *Style* нужно установить в *lbOwnerDrawFixed* и создать обработчик события *OnDrawItem:*

```
procedure TForm1.ListBox1DrawItem(Control: TWinControl;
                  Index: Integer; Rect: TRect; State: TOwnerDrawState);
var
  Bitmap: TBitmap;
  Offset: Integer;
  BMPRect : TRect;
begin
```

```
  with (Control as TListBox).Canvas do
  begin
        {очищаем прямоугольник}
    FillRect(Rect);
        {считываем картинку}
    Bitmap:=TBitMap.Create;
    Bitmap.LoadFromFile('c:\windows\'+ListBox1.Items[Index]);
    if Bitmap <> nil then begin
                {вычисляем квадрат для показа картинки}
      BMPRect:=Bounds(Rect.Left + 2, Rect.Top + 2,
                Rect.Bottom-Rect.Top-2, Rect.Bottom-Rect.Top-2);
                {рисуем картинку}
      StretchDraw(BMPRect, BitMap);
      Offset := Rect.Bottom-Rect.Top + 6;
    end;
        {выводим текст}
    TextOut(Rect.Left+Offset,Rect.Top,Listbox1.Items[Index]);
        {не забыть освободить!}
    Bitmap.Free;
  end;
end;
```

Чтобы картинки получились побольше, значение свойства ItemHeight можно увеличить.

Есть около двух десятков сообщений, которые можно послать в объекты класса TListBox и TComboBox. Подробнее о них можно узнать в справочнике по Windows API (on-line Help).

1.2 Страница “Additional” Палитры Компонент

Компоненты, размещенные на этой странице представляют из себя объектную оболочку для управляющих элементов, появившихся в более поздних версиях Windows.

TSpeedButton

I. Эмуляция потери фокуса.

Особенность этой кнопки в том, что она никогда не получает фокус и, следовательно, при нажатии на нее, текущий активный элемент фокус не теряет. В некоторых случаях бывает необходимо эмулировать потерю фокуса активным объектом. Пример, при нажатии кнопки, данные из объектов типа TEdit записываются в файл, на событие *OnExit* для них (объектов) установлена процедура верификации. В этом случае надо вызывать обработчик в явном виде:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  if ActiveControl is TEdit then
    (ActiveControl as TEdit).OnExit(ActiveControl);
end;
```

Обработчик события должен быть определен, иначе возникнет GPF.

II. Обработка двойного щелчка мышью.

Если значение свойства *GroupIndex* равно 0, то двойное нажатие будет воспринято объектом как два одиночных. Если требуется, чтобы в результате двойного щелчка кнопка не фиксировалась в нажатом состоянии, то ее свойство *AllowAllUp* устанавливается в True, и в обработчике события кнопка возвращается в прежнее состояние:

```
procedure TForm1.SpeedButton1DblClick(Sender: TObject);
begin
  SpeedButton1.Down:= Not SpeedButton1.Down;
  Do_Something;
end;
```

TTabSet, TNoteBook

Ограничения по количеству страниц для этих объектов - 16364 (еще одно "магическое число" из класса *TList*). Но, на самом деле, не имеет смысла создавать более сотни страниц.

I. Переход на страницу по ее имени.

Если объект типа *TTabSet* содержит большое количество страниц, то пролистывать их в поисках нужной – дело утомительное. Проще найти ее по имени. Предположим, что имя страницы вводится в Edit1:

```
procedure TMultPageDlg.Edit1Change(Sender: TObject);
var
  i : Integer;
  s1, s2: String;
begin
  s1:=Edit1.Text;
  if s1 = '' then Exit;
  for i:=TabSet.Tabs.Count-1 downto 0 do begin
    s2:=Copy(TabSet.Tabs[i], 1, Ord(s1[0]));
    if CompareText(s1, s2)<=0 then
      TabSet.TabIndex:=i;
  end;
end;
```

TTabbedNoteBook

I. Добавление новых объектов во время выполнения программы.

После создания нового объекта, нужно в его свойстве *Parent* указать требуемую страницу TabbedNotebook:

```
...
var
  Btn : TButton;
begin
 Btn := TButton.Create(Self);
 Btn.Parent:=TWinControl(TabbedNotebook1.Pages.Objects[1]);
 ...
```

```
end;
```

1.3 Окна в Delphi

Большинство видимых компонентов в Delphi (все компоненты, имеющие предком класс *TWinControl*) являются для Windows полноценными окнами. При доступе к этим компонентам из Windows API используется свойство *Handle* каждого компонента. Ниже приводится несколько “фокусов”, которые можно произвести с окнами компонентов, включая *TForm*.

I. “Перетаскивание” окна без помощи Caption.

Достаточно известен способ “перетаскивания” окна приложения без Caption (*caption* - поле заголовка вверху формы), этот способ описан и в разделе технической информации на CD-ROM с Delphi 1.02.

Для решения этой задачи нужно переопределить обработчик события WM_NCHITTEST для окна следующим образом:

```
type

  TForm1 = class(TForm)
  . . .
    private
      procedure WMNCHitTest(var M: TWMNCHitTest);
                                              message wm_NCHitTest;
  . . .
  end;

. . .

procedure TForm1.WMNCHitTest(var M: TWMNCHitTest);
begin
   {вызов унаследованного обработчика события}
  inherited;
   {если событие произошло в клиентской области,}
  if  M.Result = htClient then
   {то пусть Windows думает, что это произошло на Caption}
    M.Result := htCaption;
end;
```

Теперь окно можно будет переместить на экране, даже если оно совсем не имеет Caption. Однако, неприятно то, что для такого окна не будут вызываться обработчики событий, связанных с мышкой (*OnClick*, *OnMouseMove* и т.д.).

Все, что было сейчас проделано с формой (объектом класса *TForm*) можно применить и к остальным оконным компонентам. Нам никто не запрещает и для них переопределить обработку события WM_NCHITTEST.

II. Объекты, перетаскиваемые во время работы программы.

Проще всего было бы решить эту задачу переопределением соответствующего обработчика и добавлением нового свойства для класса *TWinControl.* Однако, после этого перекомпилировать библиотеку будет трудно, так как не все модули предоставлены в исходных текстах. Поэтому, приведу решение задачи на примере класса *TMemo.*

```
type
  TMoveMemo = class(TMemo)
  private
    FMoveable : Boolean;
    procedure WMNCHitTest(var M : TWMNCHitTest);
                                        message WM_NCHitTest;
  published
    property Moveable:Boolean read FMoveable write FMoveable;
  end;

. . .
procedure TMoveMemo.WMNCHitTest(var M : TWMNCHitTest);
begin
  inherited;
  if (not (csDesigning in ComponentState)) and FMoveable then
    if M.Result = htClient then
      M.Result:=htCaption;
end;
```

Свойство *Moveable* определяет, можно ли объект перетаскивать. Если бы все оконные объекты имели такой обработчик события и свойство *Moveable*, то было бы очень просто построить приложение с изменяемым во время работы программы пользовательским интерфейсом.

III. Внешний вид окна.

Как известно, внешний вид окна в Windows определяется при его создании параметром *Style* (тип Longint). Полный список возможных значений этого параметра можно найти в справочнике по Windows API, все они начинаются с префикса WS_ (например, WS_CAPTION, WS_MAXIMIZED).

Изменить внешний вид окна во время выполнения программы можно, если изменить значение его параметра *Style*. Для этого используется вызов функции API *SetWindowLong()*. Конечно, в случае формы (*TForm*) можно обойтись свойствами этого класса (но не всегда). Но в случае с обычными оконными компонентами получается забавный эффект. Попробуйте изменить стиль обычной кнопки (*TButton*):

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Style : Longint;
begin
      {старый стиль окна}
  Style:=GetWindowLong(Button1.Handle, GWL_STYLE);
      {меняем стиль окна}
  Style:=Style or WS_OVERLAPPEDWINDOW;
  SetWindowLong(Button1.Handle, GWL_STYLE, Style);
```

```
        {обновление окна (Invalidate не сработает)}
        SetWindowPos(Button1.Handle, HWND_TOP, 0, 0, 0, 0,
              SWP_NOMOVE or SWP_NOSIZE or SWP_NOZORDER or
SWP_DRAWFRAME
              or SWP_NOACTIVATE);
      end;
```

Указать свой стиль окна во время его создания можно, если переопределить для любого оконного объекта (включая TForm) процедуру *CreateParams:*

```
      type
        TForm1 = class(TForm)
          . . .
              procedure CreateParams(var Params:TCreateParams);
                                                          override;
          . . .
        end;

      . . .

      procedure TForm1.CreateParams(var Params: TCreateParams);
      begin
        inherited CreateParams(Params);
        Params.Style:=Params.Style and not WS_CAPTION;
      end;
```

IV. Распахивание (maximize) и захлопывание (minimize) окон.

Иногда нужно, чтобы окно при нажатии кнопки “Maximize” распахивалось не на весь экран, а только на часть его. Вспомните редактор исходных текстов в среде Delphi. Эта задача, как и многие другие, решается написанием обработчика соответствующего события - WM_GETMINMAXINFO. Это сообщение посылается системой в окно при максимизации окна или при изменении его размеров с помощью рамки. Окно возвращает требуемые размеры и положение.

```
      TForm1 = class(TForm)
      . . .
      private
        procedure WMGetMinMaxInfo(var M: TWMGetMinMaxInfo);
                                              message WM_GETMINMAXINFO;
      . . .
      end;

      . . .

      procedure TForm1.WMGetMinMaxInfo(var M: TWMGetMinMaxInfo);
      begin
            {на всякий случай}
        inherited;
            {указываем верхнюю границу окна ниже другого окна}
        M.MinMaxInfo^.PTMaxPosition.Y := Form2.Top+Form2.Height;
```

```
end;
```

С помощью обработчика данного события устанавливается не только место и размер распахнутого полностью окна, но и минимальные/максимальные размеры окна при изменении их с помощью рамки. Пример, установка в обработчике события WM_GETMINMAXINFO

```
m.minmaxinfo^.ptMinTrackSize.y := 100;
m.minmaxinfo^.ptMaxTrackSize.y := 300;
```

не даст сделать вертикальный размер окна менее 100 и более 300 точек.

1.4 Обработка событий от клавиатуры

I. Эмуляция нажатия клавиши.

Внутри приложения это выполняется достаточно просто с помощью вызова функции Windows API *SendMessage()* (можно воспользоваться и методом *Perform* того объекта (или формы), кому посылается сообщение о нажатой клавише).

Код

```
  Memo1.Perform(WM_CHAR, Ord('A'), 0);
```

или

```
 SendMessage(Memo1.Handle, WM_CHAR, Ord('A'), 0);
```

приведет к печати символа "A" в объекте Memo1.

II. Перехват нажатий клавиши внутри приложения.

Задача решается очень просто. Можно у формы установить свойство *KeyPreview* в True и обрабатывать событие OnKeyPress. Второй способ - перехватывать событие *OnMessage* для объекта Application.

III. Перехват нажатия клавиши в Windows.

Существуют приложения, которым необходимо перехватывать все нажатия клавиш в Windows, даже если в данный момент активно другое приложение. Это может быть, например, программа, переключающая раскладку клавиатуры, резидентный словарь или программа, выполняющая иные действия по нажатию "горячей" комбинации клавиш.

Перехват всех событий в Windows (в том числе и событий от клавиатуры) выполняется с помощью вызова функции *SetWindowsHook()*. Данная функция регистрирует в системе Windows ловушку (*hook*) для определенного типа событий/сообщений. Ловушка - это пользовательская процедура, которая будет обрабатывать указанное событие.

Основное здесь то, что эта процедура должна всегда присутствовать в памяти Windows. Поэтому ловушку помещают в DLL и загружают эту DLL из программы. Пока хоть одна программа использует DLL, та не может быть выгружена из памяти. Приведем пример такой DLL и программы, ее использующей. В примере ловушка перехватывает нажатие клавиш на клавиатуре, проверяет их и, если это клавиши "+" или

“-”, посылает соответствующее сообщение в конкретное приложение (окно). Окно ищется по имени его класса (“TForm1”) и заголовку (*caption*, “XXX”).

```
{текст библиотеки}
library SendKey;

uses
 WinTypes, WinProcs, Messages;

const
 {пользовательские сообщения}
 wm_NextShow_Event = wm_User + 133;
 wm_PrevShow_Event = wm_User + 134;
 {handle для ловушки}
 HookHandle: hHook = 0;

var
 SaveExitProc : Pointer;


{собственно ловушка}
function Key_Hook(Code: integer; wParam: word; lParam:
Longint): Longint; export;
var
 H: HWND;
begin
 {если Code>=0, то ловушка может обработать событие}
 if Code >= 0 then
 begin
   {это те клавиши?}
   if ((wParam = VK_ADD)or(wParam = VK_SUBTRACT)) and
      (lParam and $40000000 = 0) then
   begin
     {ищем окно по имени класса и по заголовку}
     H := FindWindow('TForm1', 'XXX');
     {посылаем сообщение}
     if wParam = VK_ADD then
       SendMessage(H, wm_NextShow_Event, 0, 0)
     else
       SendMessage(H, wm_PrevShow_Event, 0, 0);
   end;
  {если 0, то система должна дальше обработать это событие}
  {если 1 - нет}
  Result:=0;
 end
 else
   {если Code<0, то нужно вызвать следующую ловушку}
   Result := CallNextHookEx(HookHandle,Code, wParam, lParam);
end;
```

```
{при выгрузке DLL надо снять ловушку}
procedure LocalExitProc; far;
begin
 if HookHandle<>0 then
 begin
   UnhookWindowsHookEx(HookHandle);
   ExitProc := SaveExitProc;
 end;
end;

{инициализация DLL при загрузке ее в память}
begin
 {устанавливаем ловушку}
 HookHandle := SetWindowsHookEx(wh_Keyboard, Key_Hook,
                                                hInstance, 0);
 if HookHandle = 0 then
   MessageBox(0, 'Unable to set hook!', 'Error', mb_Ok)
 else begin
  SaveExitProc := ExitProc;
  ExitProc := @LocalExitProc;
 end;
end.
```

Размер такой DLL в скомпилированном виде будет около 3Кб, поскольку в ней не используются объекты из VCL.

Далее приведен код модуля в Delphi, который загружает DLL и обрабатывает сообщения от ловушки, просто отображая их в Label1.

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

{пользовательские сообщения}
const
 wm_NextShow_Event = wm_User + 133;
 wm_PrevShow_Event = wm_User + 134;


type
  TForm1 = class(TForm)
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
  private
        {обработчики сообщений}
    procedure WM_NextMSG (Var M : TMessage);
                                          message wm_NextShow_Event;
    procedure WM_PrevMSG (Var M : TMessage);
```

```
                                        message wm_PrevShow_Event;
  end;

var
  Form1: TForm1;
  P : Pointer;

implementation

{$R *.DFM}

{загрузка DLL}
function Key_Hook : Longint; far; external 'SendKey';

procedure TForm1.WM_NextMSG (Var M : TMessage);
begin
  Label1.Caption:='Next message';
end;

procedure TForm1.WM_PrevMSG (Var M : TMessage);
begin
  Label1.Caption:='Previous message';
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {если не использовать вызов процедуры из DLL в программе,
   то компилятор удалит загрузку DLL из программы}
  P:=@Key_Hook;
end;

end.
```

Конечно, свойство Caption в этой форме должно быть установлено в “XXX”.

IV. Изменение вида окна приложения

В раздел *private* напишите следующее:

```
  private
    { Private declarations }
    FormRgn,EllipseRgn:HRGN;
```

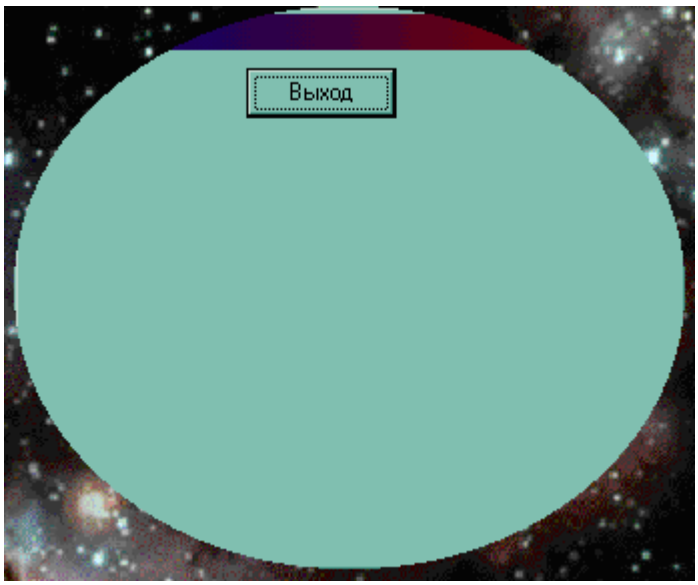
Во второй строке идёт комментарий. Комментарий - это любой текст (хоть матершина), он не влияет на работу программы, ты их пишешь для себя, а я для тебя, обращай на них внимание. Комментарии оформляются в фигурные кавычки {}. Всё, что находится между ними - это комментарий. Тут может быть хоть несколько строк. Второй способ оформления - это двойной слеш "//". Всё, что находиться в этой строке после двойного слеша - комментарий. В этом случае следующая строка уже не будет комментарием.

Теперь для главной формы создай событие *"OnCreate"*. Для этого достаточно дважды щёлкнуть по главной форме и Delphi создаст процедуру *FormCreate* . В ней напишите следующее:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 FormRgn:= CreateEllipticRgn(0,0,Width,Height);
 SetWindowRgn(Handle,FormRgn,True);
end;
```

У Вас должен получиться вот такой текст:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    FormRgn,EllipseRgn:HRGN;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
 FormRgn:=CreateEllipticRgn(0,0,Width,Height);
 SetWindowRgn(Handle,FormRgn,True);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 Close;
end;

end.
```



Теперь рассмотрим подробнее:

```
CreateEllipticRgn(
 NLeftRect:Integer, //Левая позиция
 nTopRect:Integer, // Верхняя
 nRightRect:Integer, // Правая
 nBottomRect:Integer // Нижняя
  ): HRGN;
```

Эта процедура создаёт регион в виде эллипса.

```
SetWindowRgn(
 HWnd: HWND,       //Указатель на нашу форму
 HRgn: HRGN,       // Предварительно созданный регион
 BRedraw:Boolean   // Флаг перерисовки окна
):Integer;
```

Эта процедура привязывает созданный нами регион с нашей формой. Флаг *bRedraw* должен быть true, иначе регион ну будет прорисован.

Рассмотрим ещё две функции.

```
CreateRectRgn (
 nLeftRect:Integer, //Левая позиция
 nTopRect:Integer, // Верхняя
 nRightRect:Integer, // Правая
 nBottomRect:Integer // Нижняя
  ): HRGN;
```

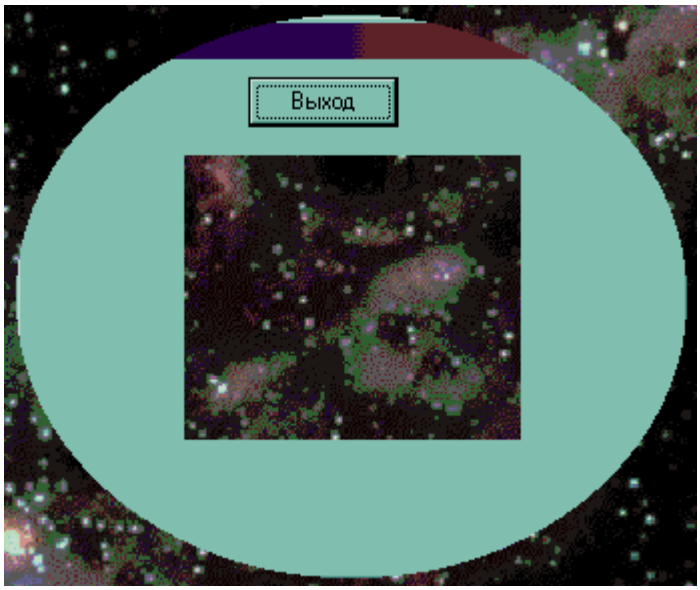
Эта функция похожа на *CreateEllipticRgn* . Она также создаёт регион, но уже квадратный.

```
CombineRgn(
 HrgnDest: HRGN,   // Указатель на результирующий регион
 hrgnSrc1: HRGN,   // Указатель на регион 1
 hrgnSrc2: HRGN,   //Указатель на регион 2
 fnCombineMode:Integer       // Метод объединения
  ) :Integer;
```

Эта функция комбинирует два региона из *hrgnSrc1* и *hrgnSrc2* и помещает результат в *HrgnDest* . *fnCombineMode* - метод комбинирования, который может быть: RGN_AND, RGN_COPY, RGN_DIFF, RGN_OR или RGN_XOR.

Теперь подправим нашу программу с учётом новых функций. Для этого, нужно изменить процедуру *FormCreate* .

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 FormRgn:=CreateEllipticRgn(0,0,Width,Height);
 EllipseRgn:=CreateRectRgn(round(Width/4),round(Height/4),
   round(3*Width/4),round(3*Height/4));
 CombineRgn(FormRgn,FormRgn,EllipseRgn,RGN_DIFF);
 SetWindowRgn(Handle,FormRgn,True);
end;
```



И напоследок немного теории. Мы использовали в наших функциях *Width* и *Height* - это ширина и высота нашего окна. Откуда они взялись? Для этого нужно понимать объёктность. Delphi - это объектно-ориентированный язык. Вот именно об объёктах мы сейчас немного поговорим.

В самом начале текста программы есть строка *TForm1 = class(TForm)* . Она означает, что класс TForm1 происходит от TForm. Такой класс унаследует себе все свойства предка (TForm), и может добавлять свои. Вот именно у предка и есть такие свойства как *Width* и *Height* . Мы просто пользовались ими.

VII. Создать иконку с помощью WinAPI

Как создать иконку с помощью WinAPI, без использования компонентов. Для этого есть функция WINAPI Shell_NotifyIcon. У неё два параметра. Первый может принимать значения:

- NIM_ADD - добавить иконку.
- NIM_DELETE - удалить иконку.
- NIM_MODIFY - редактировать

Второй параметр - это указатель на структуру TNotifyIconData. Она состоит из:

- cbSize - сюда нужно занести размер структуры.
- hWnd - указатель на окно создающее иконку.
- uID - приложение отвечающее за иконку.
- uFlags - флаги, указывающие на те поля, которые заполнены в структуре. Возможны сочетания трёх значений: NIF_ICON (в свойстве hIcon

указана иконка), NIF_MESSAGE (в свойстве uCallbackMessage указана функция обработчик сообщений для иконки) и NIF_TIP (в свойстве szTip указан текст подсказки).

- uCallbackMessage - функция обработчик сообщений для иконки.
- hIcon - иконка.
- szTip - подсказка (максимум 64 символа).

Ну и давай рассмотрим пример использования этой функции:

```
var
 NI : TNotifyIconData;
begin
 NI.uFlags := nif_icon or nif_tip or nif_message;
 NI.cbSize := SizeOf(NI);
 NI.Wnd := Window;
 NI.szTip := 'Подсказка';
 NI.hIcon := LoadIcon(HInstance, 'MAINICON');
 NI.uCallBackMessage := wm_nid;

 Shell_NotifyIcon(nim_add, @NI);
 Shell_NotifyIcon(nim_delete, @NI);
end;
```

Задание:

1. Использовать WinAPI вместо каких-то действий своей программы

Лабораторная работа № 05

Использование DLL в Delphi

Понятие DLL

Вспомним процесс программирования в DOS. Преобразование исходного текста программы в машинный код включал в себя два процесса - компиляцию и линковку. В процессе линковки, редактор связей, компоновавший отдельные модули программы, помещал в код программы не только объявления функций и процедур, но и их полный код. Вы готовили таким образом одну программу, другую, третью ... И везде код одних и тех же функций помещался в программу полностью (см. рис 1).

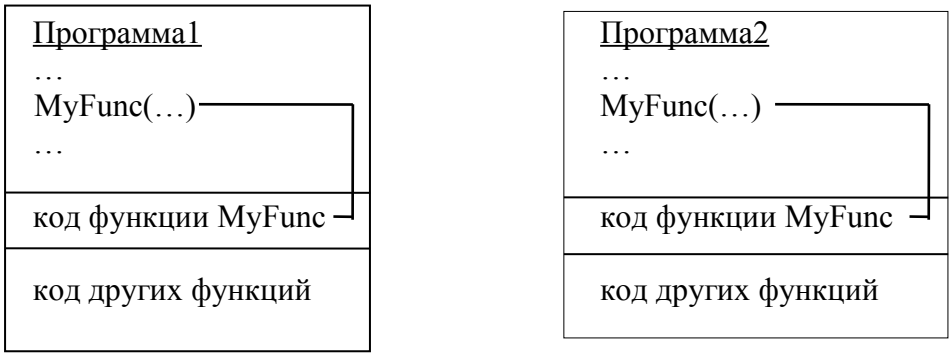


Рис.1 : Вызов функций при использовании статической компоновки

В многозадачной среде такой подход был бы по меньшей мере безрассудным, так как очевидно, что огромное количество одних и тех же функций, отвечающих за прорисовку элементов пользовательского интерфейса, за доступ к системным ресурсам и т.п. дублировались бы полностью во всех приложениях, что привело бы к быстрому истощению самого дорогого ресурса - оперативной памяти. В качестве решения возникшей проблемы, еще на UNIX-подобных платформах была предложена концепция динамической компоновки (см. рис . 2).

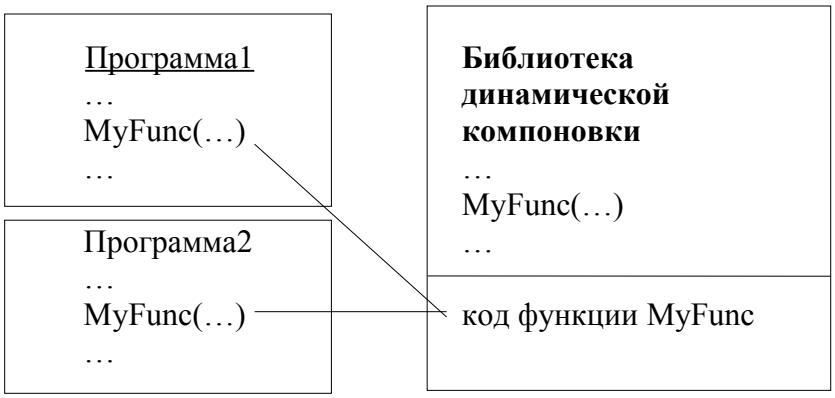


Рис.2: Вызов функций при использовании динамической компоновки

Но, чем же отличаются Dynamic Link Library (DLL) от обычных приложений? Для понимания этого требуется уточнить понятия задачи (task), экземпляра (копии) приложения (instance) и модуля (module).

При запуске нескольких экземпляров одного приложения, Windows загружает в оперативную память только одну копию кода и ресурсов - модуль приложения, создавая несколько отдельных сегментов данных, стека и очереди сообщений (см. рис. 3), каждый набор которых представляет из себя задачу, в понимании Windows. Копия приложения представляет из себя контекст, в котором выполняется модуль приложения.

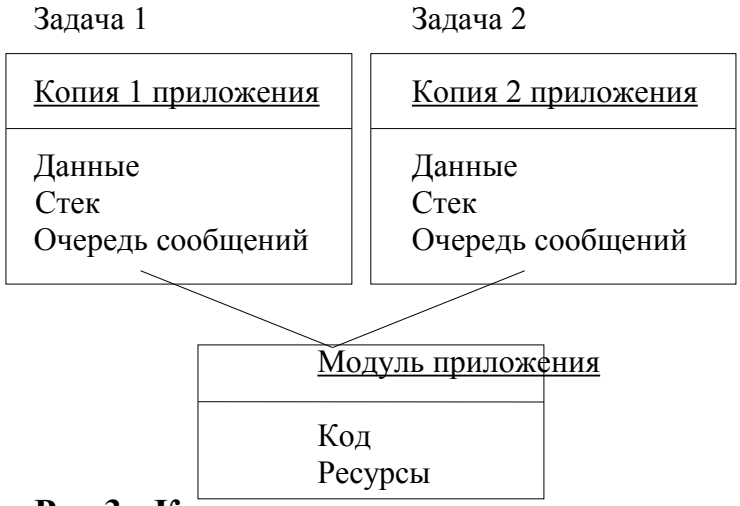
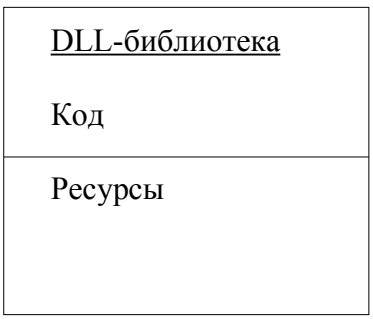


Рис.3 : Копии приложения и модуль приложения

DLL - библиотека также является модулем. Она находится в памяти в единственном экземпляре и содержит сегмент кода и ресурсы, а также сегмент данных (см. рис. 4).



Данные

Рис.4 : Структура DLL в памяти

DLL - библиотека, в отличие от приложения не имеет ни стека, ни очереди сообщений. Функции, помещенные в DLL, выполняются в контексте вызвавшего приложения, пользуясь его стеком. Но эти же функции используют сегмент данных, принадлежащий библиотеке, а не копии приложения.

В силу такой организации DLL, экономия памяти достигается за счет того, что все запущенные приложения используют один модуль DLL, не включая те или иные стандартные функции в состав своих модулей.

Часто, в виде DLL создаются отдельные наборы функций, объединенные по тем или иным логическим признакам, аналогично тому, как концептуально происходит планирование модулей (в смысле unit) в Pascal. Отличие заключается в том, что функции из модулей Pascal компонуются статически - на этапе линковки, а функции из DLL компонуются динамически, то есть в run-time.

Создание DLL в Delphi (экспорт)

Для программирования DLL Delphi предоставляет ряд ключевых слов и правил синтаксиса. Главное - DLL в Delphi такой же проект как и программа.

Рассмотрим шаблон DLL:

```
library MyDll;
uses
  <используемые модули>;

<объявления и описания функций>

exports
  <экспортируемые функции>

begin
  <инициализационная часть>
end.
```

Имя файла проекта для такого шаблона должно быть MYDLL.DPR.

!!!! К сожалению, в IDE Delphi автоматически генерируется только проект программы, поэтому Вам придется проект DLL готовить вручную. В Delphi 2.0 это неудобство устранено.

Как и в программе, в DLL присутствует раздел **uses**. Инициализационная часть необязательна. В разделе же **exports**

перечисляются функции, доступ к которым должен производится из внешних приложений.

Экспортирование функций (и процедур) может производится несколькими способами:

- по номеру (индексу);
- по имени.

В зависимости от этого используется различный синтаксис:

```
{экспорт по индексу}
procedure ExportByOrdinal; export;
begin
  .....
end;

exports
  ExportByOrdinal index 10;

{экспорт по имени}
procedure ExportByName; export;
begin
  .....
end;

exports
  ExportByName name 'MYEXPORTPROC'; { имя для экспорта может не совпадать с
именем функции ! }
```

Так как в Windows существует понятие "резидентных функций" DLL, то есть тех функций, которые находятся в памяти на протяжении всего времени существования DLL в памяти, в Delphi имеются средства для организации и такого рода экспорта:

```
exports
  ExportByName name 'MYEXPORTPROC' resident;
```

Стоит отметить тот факт, что поиск функций, экспортируемых по индексу, производится быстрее, чем при экспорте по имени. С другой стороны, экспорт по имени удобнее, особенно если Вы периодически дополняете и расширяете набор экспортируемых из DLL функций, при гарантии работы приложений, использующих DLL, и не хотите специально следить за соблюдением уникальности и соответствия индексов.

Если же Вы будете экспортировать функции следующим образом:

```
exports
  MyExportFunc1,
```

```
  MyExportFunc2,
  .....;
```

то индексирование экспортируемых функций будет произведено Delphi автоматически, а такой экспорт будет считаться экспортом по имени, совпадающему с именем функции. Тогда объявление импортируемой функции в приложении должно совпадать по имени с объявлением функции в DLL. Что же касается директив, накладываемых уже на импортируемые функции, то об этом мы поговорим ниже.

Использование DLL в Delphi (импорт)

Для организации импорта, т.е. доступа к функциям, экспортируемым из DLL, так же как и для их экспорта, Delphi предоставляет стандартные средства.

Для показанных выше примеров, в Вашей программе следует объявить функции, импортируемые из DLL таким образом:

```
{ импорт по специфицированному имени }
procedure ImportByName;external 'MYDLL' name 'MYEXPORTPROC';

{ импорт по индексу }
procedure ImportByOrdinal;  external  'MYDLL'  index  10;

{ импорт по оригинальному имени }
procedure MyExportFunc1;  external  'MYDLL';
```

Этот способ называется статическим импортом.

Как Вы могли заметить, расширение файла, содержащего DLL, не указывается - по умолчанию подразумеваются файлы *.DLL и *.EXE. Как же тогда быть в случае, если файл имеет другое расширение (например, как COMPLIB.DCL в Delphi), или если требуется динамическое определение DLL и импортируемых функций (например, Ваша программа работает с различными графическими форматами, и для каждого из них существует отдельная DLL.)?

Для решения такого рода проблем Вы можете обратиться напрямую к API Windows, используя, так называемый, динамический импорт:

```
uses
  WinTypes,  WinProcs, ... ;

type
  TMyProc  = procedure ;

var
  Handle  :  THandle;
  MyImportProc  :  TMyProc;

begin
  Handle:=LoadLibrary('MYDLL');
```

```
  if Handle>=32 then        { if <=32 - error ! }
  begin
    @MyImportProc:=GetProcAddress(Handle,'MYEXPORTPROC');
    if MyImportProc<>nil then
      ..... {using imported procedure}
  end;

  FreeLibrary(Handle);
end;
```

!!! Синтаксические диаграммы объявлений экспорта/импорта, подмена точки выхода из DLL, и другие примеры, Вы можете найти в OnLine Help Delphi, Object Pascal Language Guide, входящему в Borland RAD Pack for Delphi, и, например, в книге "Teach Yourself Delphi in 21 Days".

Если не говорить о генерируемом компилятором коде (сейчас он более оптимизирован), то все правила синтаксиса остались те же , что и в Borland Pascal 7.0

DLL, использующие объекты VCL для работы с данными

При создании своей динамической библиотеки Вы можете использовать вызовы функций из других DLL. Пример такой DLL есть в поставке Delphi (X:\DELPHI\DEMOS\BD\BDEDLL). В эту DLL помещена форма, отображающая данные из таблицы и использующая для доступа к ней объекты VCL (TTable, TDBGrid, TSession), которые, в свою очередь, вызывают функции BDE. Как следует из комментариев к этому примеру, для такой DLL имеется ограничение: ее не могут одновременно использовать несколько задач. Это вызвано тем, что объект Session, который создается автоматически при подключении модуля DB, инициализируется для модуля, а не для задачи. Если попытаться загрузить эту DLL вторично из другого приложения, то возникнет ошибка. Для предотвращения одновременной загрузки DLL несколькими задачами нужно осуществить некоторые действия. В примере - это процедура проверки того, используется ли DLL в данный момент другой задачей.

Исключительные ситуации в DLL

Возникновение исключительной ситуации в DLL, созданной в Delphi, приведет к прекращению выполнения всего приложения, если эта ситуация не была обработана внутри DLL. Поэтому желательно предусмотреть все возможные неприятности на момент разработки DLL. Можно порекомендовать возвращать результат выполнения импортируемой функции в виде строки или числа и, при необходимости, заново вызывать исключительную ситуацию в программе.

Код в DLL:

```
function MyFunc : string;
begin
  try

  {собственно код функции}

  except
    on EResult: Exception do
      Result:=Format(DllErrorViewingTable,
                          [EResult.Message]);
    else
      Result := Format(DllErrorViewingTable,
                          ['Unknown error']);
  end;
end;
```

Код в программе:

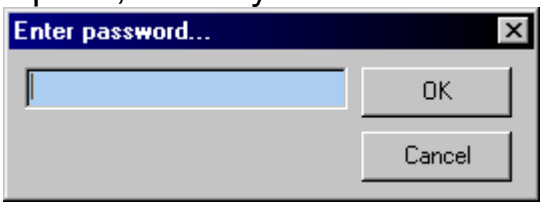
```
StrResult:=MyFunc;
if StrResult<>'' then
  raise Exception.Create(StrResult);
```

Задание:

1. Реализовать выполнение определенных функций, например, различные математические операции, через использование DLL.
2. Реализовать выполнение функции

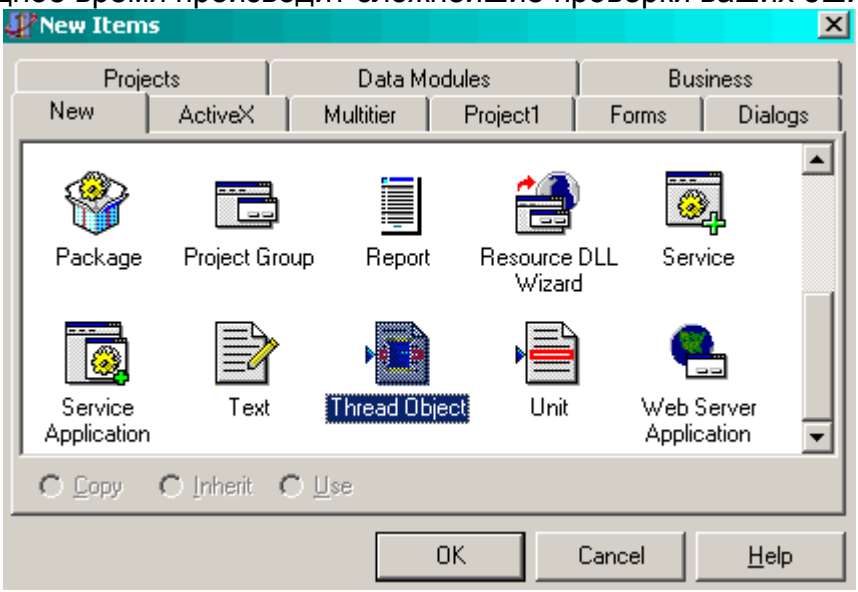
`function AskPassword: integer;`

которая открывает окно для ввода пароля и возвращает 0, в случае правильного ввода пароля, -1 в случае отказа.



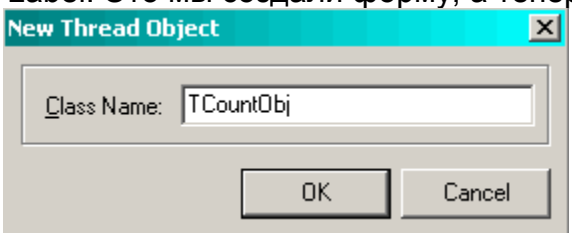
Лабораторная работа № 6
Потоки или нити (Thread)

Представьте, что вы хотите вставить в свою программу проверку орфографии. Если вы после каждой нажатой клавиши будете проверять правильность слов, то ваша программа будет очень тормозить. А как же это делает MS Word? Очень просто, после запуска программа запускается поток, который в фоновом режиме производит проверку орфографии. Вы спокойно набираете текст и даже не ощущаете (почти) как параллельный процесс в свободное время производит сложнейшие проверки ваших ошибок.



Любая программа содержит хотя бы один поток (главный), в котором она выполняется. Помимо этого, она может порождать любое количество дополнительных потоков, которые будут выполняться в фоновом режиме. У дополнительных потоков приоритет выставляется такой же как и у главного потока программы, но вы его можете увеличить или уменьшить. Чем выше приоритет потока, тем больше на него отводится процессорного времени.

Создай новый проект. Поставьте на форму TRichEdit из палитры Win32 и один TLabel. Это мы создали форму, а теперь создадим поток.



Выберем File->New (рисунок 1). Находим там *Thread Object* , выделяем и щёлкаем "OK". Появляется окошко, как на рисунке 2. Вводим имя потока (например, TCountObj). Сохраняем весь проект. Главную форму под именем main (как всегда), а поток под именем MyThread.

После этого Delphi создаст вот такой код:

```
unit MyThread;

interface

uses
  Classes;
```

```
type
  TCountObj = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;

implementation

{ Important: Methods and properties of objects in VCL can only be used in a
  method called using Synchronize, for example,

     Synchronize(UpdateCaption);

  and UpdateCaption could look like,

    procedure TCountObj.UpdateCaption;
    begin
      Form1.Caption := 'Updated in a thread';
    end; }

{ TCountObj }

procedure TCountObj.Execute;
begin
  { Place thread code here }
end;

end.
```

Это новый поток. У объекта есть только одна функция *Execute* . В этой функции мы и будем писать код потока. Напишем вот что:

```
procedure TCountObj.Execute;
begin
 index:=1;
 //Запускаем бесконечный счётчик
 while index>0 do
  begin
   Synchronize(UpdateLabel);
   Inc(index);
   if index>100000 then
    index:=0;

   //Если поток остановлен, то выйти.
   if terminated then exit;
```

```
  end;
end;
```

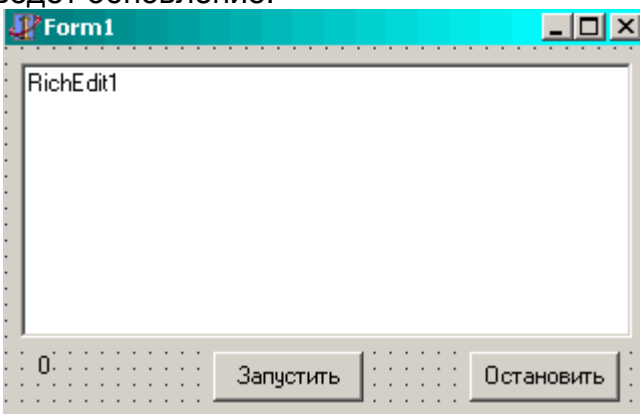
index мы объявил как integer в разделе *private* потока. Там же объявили процедуру UpdateLabel. Эта процедура выглядит так:

```
procedure TCountObj.UpdateLabel;
begin
 Form1.Label1.Caption:=IntToStr(Index);
end;
```

И последнее, что мы сделали - подключил главную форму в раздел uses, потому что обращаемся к ней в коде выше (Form1.Label1.Caption).

Теперь о магической функции Synchronize. В качестве параметра ей передаётся процедура UpdateLabel, которая производит вывод в главную форму. Для чего нужно вставлять вывод на экран в Synchronize? Библиотека VCL имеет один недостаток - она не защищена от потоков. Если главная форма и поток попробуют одновременно вывести что-нибудь в одну и ту же область экрана или компонент, то программа выполнит недопустимую операцию. Поэтому весь вывод на форму нужно выделять в отдельную процедуру и вызывать эту процедуру с помощью Synchronize.

Что происходит во время вызова Synchronize? В этот момент поток останавливается и управление передаётся главному потоку, который и произведёт обновление.



Наш поток готов. Возвращаемся к главной форме, как на рис 3. В раздел uses (самый первый, который идёт после interface) добавим свой поток MyThread.

В разделе private объявим переменную CO типа TCountObj (объект моего потока).

По нажатию кнопки "Запустить" напишем такой код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 co:=TCountObj.Create(true);
 co.Resume;
 co.Priority:=tpLower;
end;
```

В первой строке создаём поток *CO*. В качестве параметра может быть true или false. Если false, то поток сразу начинает выполнение, иначе поток создаётся, но не запускается. Для запуска нужно использовать Resume, что и делаем во второй строке.

В третьей строке устанавливаем приоритет потока поменьше, чтобы он не мешал работе основному потоку и выполнялся в фоне. Если установить приоритет повыше, то основной поток начнёт притормаживать, потому что у них будут одинаковые приоритеты.

По кнопке "Остановить" напишем:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 co.Terminate;
 co.Free;
end;
```

В первой строке останавливаем выполнение потока, а во второй уничтожаю его.

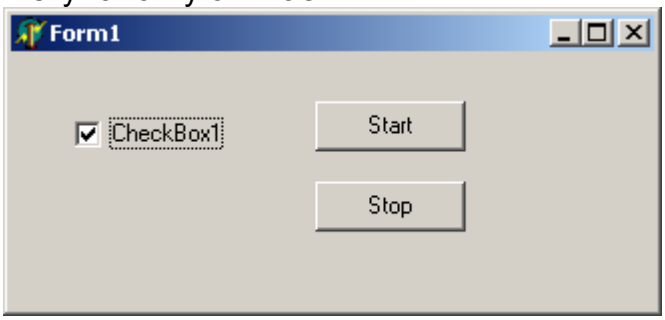
Попробуем запустить программу, запустить поток (нажатием кнопки "Запустить") и понабирать текст в RichEdit. Текст будет набиратся без проблем, и в это время в TLabel будет работать счётчик. Если бы мы запустили счётчик без отдельного потока, то не смогли бы набирать текст в RichEdit, потому что все ресурсы программы (основного потока) уходили бы на работу счётчика.

Итак, наш первый поток готов. При программировании звука мы напишем более полезные примеры с потоками. А сейчас ещё несколько полезных свойств:

- Suspend - приостанавливает поток. Для вызова просто нужно написать co.Suspend. Чтобы возобновить работу с этой же точки нужно вызвать Resume.
- Priority- устанавливает приоритет потока. Например Priority:=tpIdle;
  - tpIdle - поток будет работать только когда процессор бездельничает.
  - tpLowest - самый слабый приоритет
  - tpLower - слабый приоритет
  - tpNormal - нормальный
  - tpHigher - высокий
  - tpHighest - самый высокий
  - tpTimeCritical - критичный (лучше не использовать).
- Suspended - если этот параметр true, то поток находится в паузе.
- Terminated - если true, то поток должен быть остановлен.

Задание:

Используя пример, создайте программу, в которой работают две нити. Одна нить устанавливает галочку в элемент – checkbox, а другая нить эту галочку снимает.



Лабораторная работа № 07
Хранитель экрана

Для того, чтобы Delphi мог создать хранитель экрана, Вы должны сделать несколько установок для своей формы:

- Перед объявлением типов вставить вот такую строку: *{$D SCRNSAVE Raver}*.
- Свойство формы *BorderStyle* поменять на *bsNone* . Это означает, что у формы не должно быть никаких заголовков и обрамлений.
- Все параметры в *Border Icons* установить в False.
- Свойство формы *WindowState* поменять на *wsMaximized* .
- Создать событие *OnKeyPress, onMouseDown* и *OnMouseMove* и вставить туда всего лишь одну процедуру - *Close()* .
- На событие *FormActivate* значения Left и Top нужно установить в ноль.
- Также неплохо было бы на время работы хранителя скрыть курсор мыши, а при выходе снова показать *ShowCursor(false), ShowCursor(false)* соответственно.

Теперь Delphi создаст хранитель экрана. Остаётся одна деталь - у хранителя расширение SCR. Вы можете после создания переименовать файл в *.scr или возложить этот тяжёлый труд на Delphi. Для этого необходимо выбрать пункт *Option* из меню *Project* и на закладке *Application* в строке *Target file extension* написать SCR. В этом случае Delphi сам подставит это расширение.

Подготовительные работы закончены. Можно приступать к написанию хранителя экрана. Рассмотрим простейший пример.

Для примера понадобится объявить три переменные в разделе *private* :

```
  private
    { Private declarations }
    BGbitmap:TBitmap;
    DC : hDC;
    BackgroundCanvas : TCanvas;
```

Затем посмотрим обработчик события *OnCreate*

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 BGbitmap:=TBitmap.Create;//Инициализация

// Выставляем размеры картинки как у экрана
 BGbitmap.Width := Screen.Width;
 BGbitmap.Height := Screen.Height;
ShowCursor(false);

 DC := GetDC (0);
 BackgroundCanvas := TCanvas.Create;
 BackgroundCanvas.Handle := DC;

 BGBitmap.Canvas.CopyRect(Rect (0, 0, Screen.Width, Screen.Height),
                          BackgroundCanvas,
                          Rect (0, 0, Screen.Width, Screen.Height));
 BackgroundCanvas.Free;
 randomize;
end;
```

В самом начале инициализируется BGbitmap и устанавливаем ему размер как у экрана. Потом в переменную DC заноситься указатель на контекст вывода экрана. На первый взгляд это происходит не явно, но всё очень просто. GetDC возвращает указатель на контекст указанного в качестве параметра устройства или формы. Если указать 0, то GetDC вернёт указатель на контекст воспроизведения экрана. Если вы захотите получить контекст воспроизведения твоей формы, то должны написать GetDC(Handle). В качестве параметра выступает Handle (указатель) окна. Но в таком виде никогда не надо использовать GetDC, потому что уже есть контекст воспроизведения формы - это Canvas, а указатель на него - Canvas.Handle.

Дальше создаём BackgroundCanvas - это TCanvas, который будет указывать на экран. Можно бы использовать для рисования DC, но всё же TCanvas более понятен.

С помощью следующей строки, сохраняем копию экрана:

```
BGBitmap.Canvas.CopyRect(Rect (0, 0, Screen.Width,
Screen.Height),
                                BackgroundCanvas,
                                 Rect (0, 0, Screen.Width,
Screen.Height));
```

Затем уничтожаем указатель на контекст экрана (BackgroundCanvas.Free).

Самая последняя функция - randomize инициализирует таблицу случайных чисел. Если этого не сделать, то после запроса у системы случайного числа, тебе вернут значение из текущей таблицы, которая не всегда удачна. Вам не надо будет видеть таблицу случайных чисел, она прекрасно будет работать без твоих глаз.

В обработчике события *OnDistroy* уничтожаем картинку:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
 BGBitmap.Free;
end;
```

Последняя функция - это обработчик таймера, который мы поставили на форму:

```
procedure TForm1.Timer1Timer(Sender: TObject);
const
  DrawColors: array[0..7] of TColor =(clRed, clBlue,
clYellow, clGreen,
                                                clAqua,
clFuchsia, clMaroon, clSilver);
begin
  BGBitmap.Canvas.Pen.Color:=DrawColors[random(7)];
   BGBitmap.Canvas.MoveTo(random(Screen.Width),random(
Screen.Height));
   BGBitmap.Canvas.LineTo(random(Screen.Width),random(
Screen.Height));
  Canvas.Draw(0,0,BGBitmap);
end;
```

В разделе констант объявляем массив *DrawColors* который хранит восемь цветов. Объявление происходит следующим образом:

Имя_Массива : array [Индекс_первого_значения .. Индекс_последнего_значения] of Тип_значения_Массива = (Перечисление_значений)

В "перечислении_значений" должно быть описано "Индекс_последнего_значения"-"Индекс_первого_значения"+1 параметров. В нашем случае это 7-0+1=8 значений цвета.

Дальше меняем цвет у контекста рисования формы BGBitmap.Canvas.Pen.Color случайным цветом из массива DrawColors. Функция random возвращает число из таблицы случайных чисел, при этом, это число будет больше нуля и меньше значения переданного в качестве параметра. Это значит, что random(7) вернёт случайное число от 0 до 7. С помощью BGBitmap.Canvas.MoveTo перемещаемся в случайную точку внутри BGBitmap и с помощью BGBitmap.Canvas.LineTo рисуем из этой точки в новую точку линию. Canvas.Draw(0,0,BGBitmap); выводит наше произведение на экран.

Теперь немного усложнений.

Технически, хранитель экрана является нормальным EXE файлом (с расширением .SCR), который управляется через командные параметры строки. Например, если пользователь хочет изменить параметры вашего хранителя, Windows выполняет его с параметром "-c" в командной строке. Поэтому начать создание вашего хранителя экрана следует с создания примерно следующей функции:

```
Procedure RunScreenSaver;
Var S : String;
Begin
  S := ParamStr(1);
  If (Length(S) > 1) Then Begin
    Delete(S,1,1); { delete first char - usally "/" or "-" }
    S[1] := UpCase(S[1]);
  End;
  LoadSettings; { load settings from registry }
  If (S = 'C') Then RunSettings
  Else If (S = 'P') Then RunPreview
  Else If (S = 'A') Then RunSetPassword
  Else RunFullScreen;
End;
```

Во избежании различных неприятных ситуаций, связанных с принадлежностью компонента TTimer библиотеке визуальных компонентов VCL, действия, выполняемые хранителем лучше реализовывать в дополнительной нити. Это в свою очередь также освобождает основную нить процесса, а также в меньшей степени зависит от длительности процессорного такта. То есть, в функции TMyThread.Execute можно записать примерно следующее:

```
procedure TMyThread.Execute;
begin
  while not Terminated do
    synchronize (RedrawSaverForm)
end;
```

Соответственно, весь код, который мы бы записали в обработчике OnTimer компонента TTimer, теперь будет располагаться в TMyThread.RedrawSaverForm.

Задание:

1. Используя пример, создайте хранитель экрана, заполняющий форму не линиями, а точками или другими фигурами.
2. Используя пример, создайте хранитель экрана, в котором по экрану перемещается некоторая фигура или текст
3. Используя предыдущие л/р сделать форму установок (напр. содержимое текста, форма фигуры, скорость перерисовки изображения). А также установки пароля. Все данные сохранять в реестре.
4. Используя предыдущие л/р сделать запрос пароля при выходе из хранителя (пароль хранить в реестре).

Лабораторная работа № 08

Создание консольных приложений

Выберите в меню File пункт New[->Other], затем Console Application. Сохраните проект с расширением *.dpr, например Console.dpr. Добавьте процедуру ReadLn:

```
program Console;

{$APPTYPE CONSOLE}

begin
  ReadLn
end.
```

Если теперь вы запустите эту программу на выполнение, то увидите пустое текстовое окно, для закрытия которого следует нажать клавишу Enter.

В рамках Win32 API Microsoft предоставляет набор процедур и функций, которыми можно пользоваться при создании консольных программ – Console API. Отметим, что с окном консольного приложения связаны как минимум две ссылки (handles): одна указывает на устройство ввода, другая на устройство вывода. Ниже приведена функция, возвращающая эти ссылки.

```
var
  input_handle, output_handle: longword;

input_handle  := GetStdHandle(STD_INPUT_HANDLE);
output_handle := GetStdHandle(STD_OUTPUT_HANDLE);
```

Кроме того, нам потребуется ряд функций для позиционирования курсора, управления его отображением и очистки экрана. Большинство из них использует в качестве одного из параметров ссылку на консольное окно (output_handle). Для задания местоположения курсора в окне консольной программы существует функция SetConsoleCursorPosition.

```
var
  pos: _COORD

begin
  pos.X := 10;
  pos.Y := 10;
```

```
  SetConsoleCursorPosition(output_handle, pos);
  ReadLn
end;
```

Очистка экрана – это заполнение его символами « » (пробел). Для заполнения экрана в Console API существует функция FillConsoleOutputCharacter, которой указывается ссылка на консольное окно, символ, которым заполняется экран, размер экрана и начальная точка.

```
  pos.X := 0;
  pos.Y := 0;

  FillConsoleOutputCharacter(output_handle, '_',
    80 * 25, pos, really_write);
```

Переменная really_write, описанная как

```
var
  really_write: longword;
```

используется для хранения результатов выполнения функций.

Так как консольная программа – это программа, работающая в текстовом режиме, то и цвета в ней задаются как в DOS-программах – комбинацией составляющих фона и цвета текста. Удобно задавать цвет шестнадцатеричным значением, например $1e – желтые символы на синем фоне. Младший разряд – цвет символа, старший – фона. Таблица возможных цветов приведена в приложении А. Для вывода на консоль какой-либо строки без учета атрибутов используется функция WriteConsoleOutputCharacter, а для заполнения этой строки атрибутами – функция FillConsoleOutputAttribute.

Например, процедура изменения цвета символов на желтый, а фона на синий будет выглядеть так:

```
  FillConsoleOutputAttribute(output_handle, $1e,
    80*25, pos, really_write);
```

Обработчики событий

Для обработки событий в консольных программах нам необходимо создать цикл обработки сообщений. Для этого создадим цикл while, в котором будем считывать клавиатурных ввод и действия с мышью и

анализировать их. Основной функцией будет стандартная функция ReadConsoleInput, которая собирает всю информацию, посылаемую консольному окну (через ссылку на консольный ввод). Полученная информация сохраняется в буфере, который содержит различные значения в зависимости от источника информации – клавиатуры или мыши. Так, для ввода с клавиатуры сохраняется тип действия (нажатие/отпускание), число повторов, виртуальный код клавиши (ASCII), ее скан-код и т.п., для мыши в буфер заносятся координаты курсора, информация о нажатых кнопках и т.п.

Рассмотрим следующих пример цикла обработки сообщений. Пусть программа завершается нажатием клавиши F10, а все действия с мышью отображались в статусной строке. Вот код такого обработчика:

```
continue := true;
while (continue) do
  begin
    ReadConsoleInput(input_handle, input_buf,  1,
      really_read);
    case input_buf.EventType of
      KEY_EVENT:
      with input_buf.Event.KeyEvent do
        begin
          //If key pressed
          if  (bKeyDown)
          and (wVirtualKeyCode = VK_ESCAPE) then
            continue := false
        end;
      _MOUSE_EVENT:
      with input_buf.Event.MouseEvent do
        begin
          pos.X := 0;
          pos.Y := 0;

          SetConsoleCursorPosition(output_handle, pos);
          Write(Format('Mouse position X=%.2d, Y=%.2d',
            [dwMousePosition.X, dwMousePosition.Y]))
        end
    end
  end
```

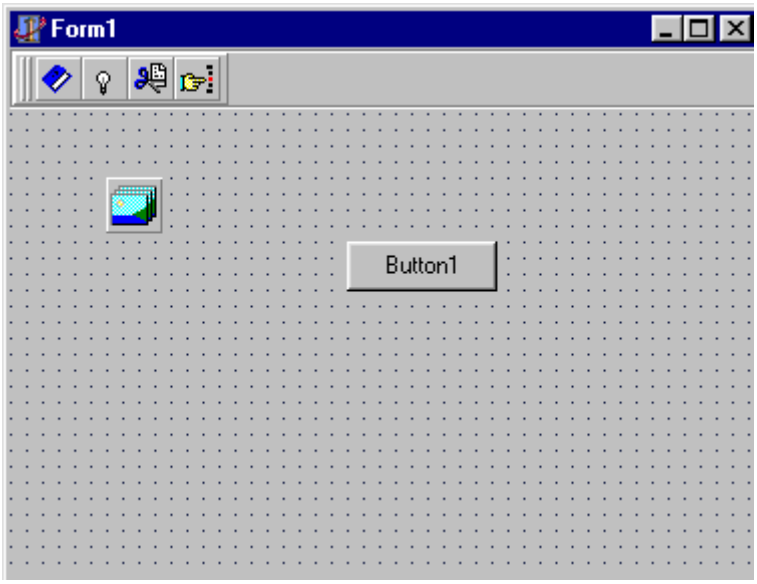
Задание

Написать консольное приложение по вариантам:

- файловый менеджер;
- текстовый редактор;
- калькулятор;
- командный интерпретатор.

Лабораторная работа № 09
Технология Dock

Для большей ясности – технология Dock - это когда палитру с кнопками можно оторвать от окна и прилепить в другое место или вообще превратить в отдельное окно.



Для того, чтобы TToolBar можно было перемещать, достаточно установить в нём свойство *DragKind* в *dkDock*. Вот и всё. Но главная проблема не в этом. Самое сложное здесь - это сохранить положение TToolBar после выхода из проги и восстановить его при запуске. Для примера мы рассмотрим программу, которую вы должны доделать до полноценной.

Для демонстрации нам понадобилась кнопка, по нажатию которой будет выводится положение TToolBar:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 r:TRect;
begin
if ToolBar1.HostDockSite<>ControlBar1 then
  begin
   GetWindowRect(ToolBar1.Handle, R);
    Application.MessageBox(PChar(IntToStr(r.Left)+'--
'+IntToStr(r.Top)),
                           'MM',IDOK);
  end;
end;
```

В первой строке мы проверяем, лежит ли ToolBar1 на ControlBar1 с помощью (ToolBar1.HostDockSite<>ControlBar1). Если он лежит, то получить положение ToolBar1 очень просто. Для этого можно узнать всего лишь ToolBar1.Left и ToolBar1.Top.

Если ToolBar1 не лежит на ControlBar1 (ToolBar1 выглядит как отдельное окно), то задача усложняется. Тебе придётся вызывать GetWindowRect, чтобы получить реальное положение ToolBar1 на экране. В качестве первого параметра вы должен передать указатель на ToolBar1, а второй - это переменная типа TRect в которую запишется

реальное положение окна. Для удобства мы выводим эти значения в окне сообщения Application.MessageBox.

Теперь вы можете запустить программу и переместить ToolBar1 по экрану. Каждый раз, когда ты будешь нажимать кнопку, программа будет выводить окно и показывать тебе реальное положение ToolBar1.

По событию OnShow мы напишем:

```
procedure TForm1.FormShow(Sender: TObject);
begin
 ToolBar1.ManualDock(nil,nil,alNone);
        ToolBar1.ManualFloat(Bounds(100,        500,
ToolBar1.UndockWidth, ToolBar1.UndockHeight));
end;
```

ToolBar1.ManualDock заставляет переместится ToolBar1 на новый компонент. В качестве первого параметра указывается указатель на компонент или окно, к которому мы хотим прилепить ToolBar1. Поскольку мы хотим, чтобы после загрузки ToolBar1 превратился в отдельное окно, поэтому мы указываем nil. Второй параметр можешь ставить nil. Он означает компонент внутри компонента указанного в качестве первого параметра, на который мы хотим поместить ToolBar1. Мы указали nil. Третий параметр - выравнивание.

С помощью ToolBar1.ManualFloat просто двигаем ToolBar1 внутри нового компонента. У нас новый компонент nil, т.е. окно, поэтому мы двигаем ToolBar1 по окну. Может не совсем понятно? Попробуйте запустить пример и потестировать его, тогда всё встанет на свои места.

И ещё ToolBar1.UndockWidth и ToolBar1.UndockHeight возвращают размер ToolBar1, когда он выглядит как окно, а не лежит на ControlBar1.

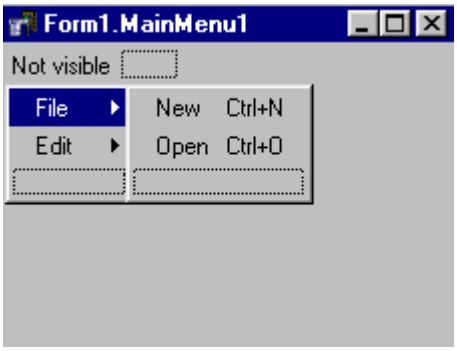
Когда вы будешь использовать это в своей программе для сохранения положения ToolBar1, надо будет написать примерно следующее по событию OnClose:

```
var
 r:TRect;
begin
if ToolBar1.HostDockSite<>ControlBar1 then
  begin
   GetWindowRect(ToolBar1.Handle, R);
   {Здесь надо сохранить в реестре R.Left и R.Top.
   А также признак, что ToolBar1 не лежит на ControlBar1}
  end
 else
  begin
      {Здесь надо сохранить в реестре ToolBar1.Left и
ToolBar1.Top.
       А также признак, что ToolBar1 лежит на ControlBar1}
  end;
end;
```

На запуск программы вы должен написать примерно следующее:

```
procedure TForm1.FormShow(Sender: TObject);
begin
```

```
    // Прочитать положение ToolBar1. ControlBar1 то
      Begin
       ToolBar1.Left:=Сохранённая левая позиция
       ToolBar1.Topt:=Сохранённая верхняя позиция
      End;
    Иначе
     begin
      ToolBar1.ManualDock(nil,nil,alNone);
          ToolBar1.ManualFloat(Bounds(Сохранённая   левая
позиция,
                        Сохранённая   правая   позиция,
ToolBar1.UndockWidth,
               ToolBar1.UndockHeight));
     End;
    end;
```



Теперь мы сделаем меню в стиле M$. Для этого нужно поставить ещё один ToolBar и установим его свойство *ShowCaption* в true.Создадим на нём две кнопки и назовём их *File* и *Edit* . Теперь установим компонент MainMenu и сделаем его таким как на рисунке. Меню *Not visible* сделаем невидимым (Visible=false), в этом случае всё меню будет подключено к форме, но будет не видно. Для чего мы это делаем, ведь можно было использовать PopupMenu? А потому что при использовании PopupMenu приходится мучится с клавишами быстрого вызова, а в нашем способе они подключаются автоматически вместе с главным меню.

Чтобы создать подменю для меню File, нужно щёлкнуть по нём правой кнопкой и выбрать *Create Submenu* или нажать CTRL+Стрелка в право.

Теперь кнопке File в свойстве MenuItem ставим File1 (имя пункта меню), а кнопке Edit ставим Edit1. И на последок обеим кнопкам нужно установить свойство Grouped в true.

Задание:

1. Используя пример, создайте программу – простейший текстовый редактор, использующий технологию Docking при работе с панелями инструментов.
2. Дополните программу возможностью включения/выключения панелей (выбор осуществляется как при нажатии правой кнопкой мыши на панели, так и при выборе соответствующего пункта в меню Вид).
3. Дополните программу возможностью пристыковки (прилипания) панели инструментов к краям окна приложения.
4. Дополните программу возможностью настройки панели инструментов (добавления и удаления кнопок) в ходе работы приложения. Для реализации

задания рекомендуется использовать TActionManager.

Лабораторная работа № 10

Создание приложений с помощью WinAPI

Изучение базового шаблона программы для Windows. Принцип отображения информации в окнах.

Работа выполняется в системе программирования Borland Delphi 2.0 в среде Windows 95. Использование средств визуального программирования и классов VCL не допускается, то есть Delphi используется просто как 32-разрядный компилятор с языка Паскаль для операционной системы Windows.

Теоретические положения

Любая программа Windows при своем запуске получает от операционной системы ряд параметров. Эти параметры могут быть проанализированы внутри программы. Для доступа к ним в Паскаль-программе следует использовать следующие идентификаторы:

hInstance: Longint — Хэндл экземпляра приложения. Это число идентифицирует процесс, который представляет собой запущенная программа, в операционной системе. Хэндл экземпляра требуется некоторым функциям API.

CmdLine: PChar — Командная строка, с помощью которой запущено приложение. Содержит полное имя исполняемого модуля, а также параметры командной строки.

CmdShow: integer — Числовая константа, определяющее, каким должно быть показано главное окно программы (см. описание функции ShowWindow). Этот параметр может игнорироваться приложением.

Как правило, программы, написанные для Windows, используют окна для организации пользовательского интерфейса. Из этого правила могут быть исключения, но в подавляющем большинстве случаев программисту приходится работать с окнами.

Основой оконного графического пользовательского интерфейса Windows является механизм сообщений. Когда с окном должно быть произведено некоторое действие, операционная система посылает окну сообщение соответствующего содержания, которое это окно должно обработать. Например, в качестве реакции на сообщение WM_HIDE окно должно исчезнуть с экрана (сделаться невидимым). На самом деле посылка сообщения приводит к вызову некоторой процедуры ("оконной процедуры"), в качестве параметра которой передается небольшой блок данных, который и называется сообщением.

Сообщения могут передаваться оконной процедуре немедленно, а могут помещаться в очередь сообщений, откуда программа должна выбирать их по мере обработки предыдущих сообщений. Хотя сообщения адресуются ОКНАМ, Windows создает очередь сообщений для каждого ПРОЦЕССА (точнее — для каждого потока выполнения, их в рамках одной задачи, т.е. процесса, может быть несколько). Когда в каком-либо потоке программы порождается окно, Windows фиксирует его принадлежность к этому потоку и помещает сообщения для этого окна в очередь сообщений породившего его потока. В простейшем случае, когда в программе не организуются несколько потоков выполнения, очередь сообщений — одна на всю программу, и все сообщения, адресуемые окнам программы, помещаются в эту очередь, откуда программа должна уметь их забирать.

Из очереди сообщения извлекаются в виде блоков данных, структура которых такова:

```
TMsg = packed record
    hwnd: HWND; // хэндл окна-адресата
    message: UINT; // код (тип) сообщения
```

```
        wParam: WPARAM; // параметр сообщения
        lParam: LPARAM; // параметр сообщения
        time: DWORD; // момент посылки сообщения
        pt: TPoint; // положение курсора мыши в момент посылки сообщения
    end;
```

Сообщения содержит в себе информацию о том, какому окну оно адресовано (хэндл окна в поле hwnd), идентификатор сообщения, определяющий тип требуемого действия (поле message), и два параметра, назначение которых специфично для каждого типа сообщений. В ранних версиях Windows поле wParam было 16-разрядным, поле lParam - 32-разрядным, однако в Windows 95 (Win32) оба эти параметра 32-разрядные.

Тип сообщения задается целым числом в поле Message. В Windows определены более 1000 стандартных сообщений; кроме того, программист может вводить дополнительные сообщения. Все сообщения Windows перечислены в файле MESSAGES.PAS в качестве числовых констант. В любом трансляторе для Windows используется устоявшаяся система идентификаторов для обозначения типов сообщений, например: WM_CLOSE, WM_PAINT и т.д. При этом нет необходимости помнить, что WM_CLOSE=10h, WM_PAINT=0Fh и т.д.

В тело сообщения включается также дополнительная информация: о времени посылки сообщения и положении курсора мыши в этот момент. Нетрудно видеть, что размер сообщения фиксирован и вся "полезная информация" должна укладываться в 8 байт (поля wParam и lParam). Если сообщение подразумевает необходимость передачи большего объема информации, то передается указатель на блок памяти, содержащий эту информацию, и откуда процедура, обрабатывающая сообщение, должна эту информацию прочитать.

Практически в любой программе, работающей с окнами Windows, можно логически выделить следующие разделы:

1. Порождение окон.
2. Цикл выборки сообщений из очереди.
3. Оконные процедуры, реализующие логику обработки сообщений в окнах программы.

Вся смысловая нагрузка программы, все ее прикладное значение обычно сосредоточено в третьем из перечисленных разделов, точнее — в процедурах, которые вызываются внутри оконной процедуры. Почти все, что делает программа в Windows, она делает в ответ на сообщение, свидетельствующее о каком-либо внешнем событии. Пользователь щелкнул мышью по кнопке — окно получило сообщение WM_COMMAND — начали выполняться какие-то действия. Пользователь закрыл окно — все окна, которые были под только что закрытым окном на экране получили сообщение WM_PAINT и должны перерисоваться.

Общая структура шаблона программы

В данной работе предлагается составить программу, соответствующую следующему шаблону:

```
uses Messages, Windows;

procedure WndProc(...); stdcall; //Оконная процедура
begin
```

```
    case
      //..................... обработка WM_PAINT
      else
        result:=DefaultWindowProc(...);
    end; //case
  end;

  begin
    {регистрация оконного класса}
    {порождение окна}
    {вывод окна на экран}
    {цикл обработки сообщений}
  end.
```

Оконные процедуры

Оконная процедура в Delphi описывается как функция, возвращающая 32-битовое целое значение, с использованием директивы **STDCALL.** STDCALL указывает компилятору, что вызов такой функции происходит по правилам вызова процедур системой Windows. Набор параметров оконной процедуры следующий:

```
function WndProc1(hWnd: THandle; Msg: integer;
                  wParam: longint; lParam: longint
): longint; stdcall;
```

Оконная процедура реализует логику обработки сообщений окном и вызывается операционной системой всякий раз, когда окну передается сообщение для обработки. Нетрудно видеть, что параметры оконной процедуры соответствуют полям структуры сообщения, определяющим суть сообщения. Это хэндл окна-получателя, типа сообщения и параметры wParam и lParam. Результат, возвращаемый оконной процедурой, зависит от типа обрабатываемого сообщения. Для некоторых сообщений необходимо вернуть определенный результат, например, чтобы сообщить отправившему сообщение процессу, что оно успешно обработано. Для других сообщений результат не важен, в этом случае рекомендуется возвращать 0.

Должна ли оконная процедура уметь обрабатывать все 1000 с лишним возможных сообщений? С одной стороны — да, ведь все сообщения, адресованные окну, проходят через его оконную процедуру. Но с другой стороны — в Windows определена реакция по умолчанию на все возможные сообщения, поэтому подавляющее большинство сообщений оконная процедура, которую пишет программист, может поручить обработать Windows, вызвав внутри себя

```
DefWindowProc(hwnd,msg,wparam,lparam);
```

Поэтому тело оконной процедуры обычно представляет собой один большой оператор **case**, завершаемый командой "а иначе выполнить обработку по умолчанию". Например так:

```
  case Message of
    WM_PAINT: doPaint;        // вызов процедуры реакции
на сообщение
      WM_DESTROY:  PostQuitMessage(0);  // приведет к
завершению программы
```

```
    else
       result:=DefWindowProc(hwnd,msg,wparam,lparam); // обработка по умолчанию
  end; // case
```

Регистрация оконных классов

Для порождения окна прежде всего должен быть описан класс окна. Класс окна — это некоторый блок памяти, существующий в недрах Windows, в котором содержится описание различных параметров, определяющих внешний вид и поведение окон (экземпляров данного класса), и в частности — указатель на оконную процедуру, обрабатывающую сообщения любого окна данного класса. Как уже понятно из вышесказанного, одновременно может существовать множество окон - экземпляров одного и того же класса, со сходными свойствами и поведением, и окно в Windows всегда порождается как экземпляр некоторого зарегистрированного в системе класса. В Windows имеется несколько предописанных оконных классов, однако обычно программисту приходится описывать и регистрировать свои собственные классы окон.

Для регистрации оконного класса программа должна вызвать функцию

```
function RegisterClassEx(const WndClass: TWndClassEx): ATOM;
```

В случае успешной регистрации класса функция возвращает ненулевое целое значение, которое можно использовать для идентификации оконного класса в других процедурах API; нулевой результат свидетельствует о невозможности создания класса. Параметр WndClass описывает необходимые для создания класса параметры и имеет следующую структуру:

```
  TWndClassEx = packed record
      cbSize: UINT; // размер структуры TWndClassEx
      style: UINT;  // битовые флаги, определяющие стиль окна
      lpfnWndProc: TFNWndProc; // адрес оконной процедуры
         cbClsExtra: Integer;  // Размер дополнительной памяти класса
      cbWndExtra: Integer; // Размер дополнительной памяти окна
         hInstance: HINST;    // Хэндл экземпляра приложения, которому принадлежит
                      // оконная процедура
      hIcon: HICON;  // Хэндл значка окна (32x32 пиксела)
      hCursor: HCURSOR; //  Хэндл курсора окна по умолчанию
        hbrBackground: HBRUSH; // Хэндл кисти окна по умолчанию или цвет фона
      lpszMenuName: PAnsiChar; //  Оконное меню (имя ресурса)
         lpszClassName: PAnsiChar; // Строка - имя класса, заканчивается #0
        hIconSm: HICON; // Хэндл маленького значка окна (16x16 пикселов)
    end;
```

cbSize должно содержать размер передаваемого блока данных, т.е. SizeOf(TWndClassEx).

style является комбинацией битовых флагов, получаемых при логическом сложении (операция OR) следующих констант (приведены не все возможные стили):

CS_CLASSDC	Один контекст устройства выделяется для всех окон класса и используется при операциях рисования в них. При этом программист должен заботиться о том, чтобы операции рисования в окнах этого класса не происходили в разных потоках одновременно.
CS_DBLCLKS	Окна при двойном щелчке мышью внутри окна получают соответствующие сообщения.
CS_GLOBALCLASS	Позволяет объявить глобальный класс окна (актуально для DLL).
CS_HREDRAW	Перерисовывает все окно, если изменяется горизонтальный размер окна.
CS_NOCLOSE	Убирает команду "Закрыть" из системного меню окна и запрещает кнопку закрытия окна в его правом верхнем углу.
CS_OWNDC	Заранее выделяет уникальный контекст устройства для каждого окна класса..
CS_PARENTDC	Дочернее окно использует контекст устройства окна-родителя. В ряде случаев позволяет ускорить отрисовку дочерних окон.
CS_SAVEBITS	Окно при своем появлении сохраняет образ закрываемой им части экрана в буфере, а при скрытии восстанавливает этот образ из буфера; при этом тем окнам, что были под ним, не посылается сообщение WM_PAINT и они не перерисовываются. Этот стиль обычно используется для небольших по размеру окон, которые появляются на экране на короткое время (диалоги, меню).
CS_VREDRAW	Перерисовывает все окно, если изменяется вертикальный размер окна.

cbClsExtra и cbWndExtra указывают размер дополнительной памяти, резервируемой Windows для всего класса и каждого окна соответственно. Программа может обращаться к этой памяти и использовать ее для своих целей с помощью функций GetClassLong, GetClassWord, GetWindowLong, GetWindowWord, SetClassLong, SetClassWord, SetWindowLong, SetWindowWord (см. Win32 API Help).

hInstance должно содержать хэндл экземпляра приложения, в котором расположена оконная процедура. Инициализируется как

```
wndClass.hInstance:=hInstance;
```

В поле lpszClassname должен быть указатель на Z-строку (оканчивающуюся нулем), в которой содержится имя класса. Помните, что в Delphi тип String может означать как длинные строки, которые можно передавать в качестве Z-строк (ANSIString),

так и короткие строки в стиле языка Паскаль предыдущих версий (ShortString), когда первый байт строки содержит ее длину, а нулевого символа в конце нет. Трактовка типа String определяется опцией компилятора {$H+/-}.

В оставшихся нерассмотренными полях указываются хэндлы ресурсов (значки, курсор), которые используются окнами класса по умолчанию. Если эти параметры не указывать (сделать нулевыми), то программа должна сама заботиться об изменении вида курсора при вводе указателя мыши в область окна, о рисовании значка при минимизации окна и т.д.

В системе Windows имеется небольшой набор предопределенных ресурсов и объектов GDI, которые можно использовать при описании классов окон. Их хэндлы можно получить с помощью функций

```
function LoadIcon(hInstance: HINST; lpIconName: PChar): HICON;
```

```
function LoadCursor(hInstance: HINST; lpCursorName: PChar): HCURSOR;
```

```
function GetStockObject(Index: Integer): HGDIOBJ;
```

При этом при загрузке предопределенного ресурса в функции LoadIcon и LoadCursor передается нулевое значение hInstance, а вместо имени ресурса передается индекс из числа предопределенных констант, например

```
wndClass.hCursor:=loadCursor(0, idc_Arrow);
```

(использовать в качестве курсора стандартный системный курсор в виде стрелки).

Классы окон, созданные приложением, уничтожаются после его завершения автоматически.

Создание окон

Зарегистрировав таким образом класс окна, программа может приступить к созданию самих окон. Создание окна производится функцией

```
function CreateWindow(
  lpClassName: PChar; // Имя зарегистрированного класса окна
  lpWindowName: PChar; // Заголовок окна
  dwStyle: DWORD;  // Стиль окна (комбинация флагов)
   X, Y, nWidth, nHeight: Integer;  // Начальное положение и размеры окна
  hWndParent: HWND; // Хэндл окна-родителя
  hMenu: HMENU;  // Хэндл меню окна
  hInstance: HINST;  // Экземпляр приложения, создающий окно
  lpParam: Pointer // Указатель на параметры для WM_CREATE
): HWND;
```

Существует также функция CreateWindowEx, среди параметров которой присутствует дополнительное поле флагов dwExStyle. остальные параметры и действия функции идентичны рассматриваемой.

Функция CreateWindow или CreateWindowEx создает окно и возвращает в случае успеха операции ненулевой хэндл созданного окна. В противном случае возвращается 0. Хэндл окна используется для идентификации окна при всех последующих операциях с ним. Рассмотрим подробнее некоторые ее параметры.

В качестве X, Y, nWidth и nHeight может быть передано значение CW_USEDEFAULT, при этом начальное положение окон и размеры будут определяться Windows.

Указание ненулевого хэндла родительского окна hWndParent заставляет порождаемое окно автоматически выполнять некоторые дополнительные действия, например: когда родительское окно минимизируется, подчиненные окна скрываются, подчиненные окна всегда располагаются на экране поверх родительского и т.д.

В поле dwStyle может указываться логическая сумма (OR) следующих флагов (указаны не все флаги):

В параметре dwExStyle функции CreateWindowEx может указываться логическая сумма(OR) следующих флагов (указаны не все флаги), определяющих дополнительный стиль окна:

Окно показывается или убирается с экрана при помощи функции

```
function ShowWindow(hWnd: HWND; nCmdShow: Integer): BOOL;
```

В параметре nCmdShow указывается действие, которое необходимо произвести с окном, хэндл которого hWnd. В частности, SW_SHOW означает "показать окно".

Организация обработки очереди сообщений

Сообщения, которые посылаются окнам с буферизацией (post), в отличие от сообщений, посылаемых без буферизации (send), передаются операционной системой не напрямую в оконную процедуру окна-получателя, а помещаются в очередь сообщений. Эта очередь организуется для каждого потока, в котором порождались окна. Если в программе не организовано множество потоков, то можно говорить о единой очереди сообщений программы. Выбор сообщений из очереди и передача их на обработку в соответствующие оконные процедуры возложены на само приложение.

В простейшем случае выбор сообщений организуется в виде следующего цикла:

```
while GetMessage(msg,0,0,0) do
begin {получить очередное сообщение}
   TranslateMessage(msg); {Windows транслирует сообщения от клавиатуры}
  DispatchMessage(msg); {Windows вызовет оконную процедуру}
end; {выход по wm_quit, на которое GetMessage вернет FALSE}
```

Функция GetMessage с указанными параметрами извлекает очередное сообщение из очереди и помещает его в структуру msg типа TMsg. Если сообщений в очереди нет, то текущий поток переводится в состояние ожидания и досрочно отдает управление другим потокам. Функция GetMessage возвращает истину при получении любого сообщения, кроме WM_QUIT.

Сообщение WM_QUIT свидетельствует о том, что либо пользователь, либо операционная система подали приложению команду завершиться. При получении сообщения WM_QUIT цикл выборки сообщений завершается и программа, возможно, выполнив какие-то завершающие действия, также завершается.

Любое другое сообщение, отличное от WM_QUIT, приводит к результату TRUE в функции GetMessage и должно быть передано в оконную процедуру. Это выполняется функцией DispatchMessage, при выполнении которой Windows анализирует данные сообщения и вызывает нужную оконную процедуру.

Обычно перед передачей сообщения оконной процедуре выполняются дополнительные действия, самое распространенное из которых — обработка сообщений от клавиатуры функцией TranslateMessage. Эта функция при получении низкоуровневого сообщения о нажатии клавиши WM_KEYDOWN в случае нажатия алфавитно-цифровой клавиши помещает в очередь сообщений сообщение WM_CHAR, содержащее вместо кода клавиши код соответствующего символа в зависимости от установленного в системе языкового драйвера. В дальнейшем это сообщение выбирается из очереди и обрабатывается на общих основаниях.

Посылка сообщений внутри программы

Посылка сообщений окнам не есть прерогатива операционной системы. Любой пользовательский процесс также может посылать сообщения. Для этого есть две основные функции Windows API, реализующие буферизованную и небуферизованную посылку сообщений.

Под буферизованной посылкой сообщений понимается помещение их в очередь того потока, в котором создавалось окно-получатель. Эта операция реализуется функцией

```
function PostMessage(hWnd: HWND; Msg: UINT;
                     wParam: WPARAM; lParam:
LPARAM): BOOL;
```

Сообщение типа Msg с параметрами wParam и lParam отправляется окну с хэндлом hWnd. Функция возвращает TRUE, если сообщение отправлено успешно, и FALSE в противном случае. Важно, что после выполнения функции PostMessage, сообщение помещено в очередь, но еще не обработано оконной процедурой окна-получателя.

Посылка сообщения без буферизации выполняется функцией

```
function SendMessage(hWnd: HWND; Msg: UINT;
                     wParam: WPARAM; lParam:
LPARAM): LRESULT;
```

При этом сообщение не помещается в очередь, а вместо этого непосредственно вызывается оконная процедура окна-приемника, которая обрабатывает сообщение. Значение, возвращаемое оконной функцией, возвращается в качестве результата функции SendMessage. Таким образом, после возврата из функции SendMessage посланное сообщение уже обработано и известен результат его обработки.

Организация отображения информации в окнах

Для операционной системы DOS были разработаны приложения и оконные библиотеки для текстового режима, в которых при открытии окна закрываемая им область экрана сохранялась в особом буфере в памяти, а после закрытия окна восстанавливалась из этого буфера. Таким образом, окно обязано было заботиться о восстановлении экрана после своего исчезновения.

Метод сохранения и восстановления части экрана, закрываемой окном, в буфере неудобен для построения графического интерфейса, так как слишком велик объем информации, который требовалось бы сохранять: в полноцветном режиме, когда в видеопамяти отводится 24 или 32 бита на каждый пиксел, буфер для окна, занимающего весь экран, требовал бы более мегабайта оперативной памяти, что нереально. Поэтому в основе графического интерфейса Windows лежит другой принцип: каждое окно в любой момент времени должно уметь отображать себя само. Операционная система при этом

отслеживает, какие участки каких окон требуют перерисовки и посылает этим окнам сообщения WM_PAINT, обрабатывая которые, оконные процедуры этих окон должны обеспечить отображение информации в окне.

Это накладывает на программиста некоторые дополнительные ограничения. Так, в программах для DOS программист мог один раз вывести на экран изображение и быть уверенным, что оно останется в целости и сохранности, пока сама программа его не сотрет. В то время как в Windows изображение в окне может быть стерто в силу внешних причин, например, когда другое окно закрыло окно программы, а затем было убрано с экрана. Поэтому данные, необходимые для прорисовки изображения в окне (редактируемый текст или массив статистических данных для диаграммы) должны быть наготове всегда, пока окно видимо на экране.

Обработка сообщения WM_PAINT

Когда операционная система считает, что окно или его часть должны быть перерисованы, оно посылает ему сообщение WM_PAINT, не имеющее параметров. Обработка сообщения WM_PAINT собственно и должна состоять в перерисовке изображения в окне. Оконная процедура при обработке WM_PAINT должна возвращать 0.

Для рисования в программе для Windows используются функции GDI (графического интерфейса устройства), являющиеся унифицированным интерфейсом драйверов дисплея, а также принтеров, графопостроителей и прочих устройств вывода графической информации. Чтобы воспользоваться большинством из этих функций, необходимо получить так называемый "контекст устройства". Фактически, это внутренняя структура данных GDI, содержащая такие сведения, как возможная глубина цвета, разрешающая способность устройства, способ пересчета логических координат в физические, а также об активных инструментах рисования, таких как перо, кисть и шрифт.

Часто требуется перерисовать не все окно целиком, а некоторую его часть, допустим, когда два окна частично перекрывались, и затем одно из них было закрыто. Поэтому при получении сообщения WM_PAINT определен так называемый недействительный регион, в пределах которого требуется производить рисование. Чаще всего недействительный регион является совокупностью прямоугольных областей. Также существует понятие региона отсечения, за пределами которого рисование не производится. Чаще всего регион отсечения по координатам совпадает с недействительным регионом, но это разные понятия.

Обработка сообщения WM_PAINT в оконной процедуре должна строиться примерно следующим образом:

```
case message of
  .......
  WM_PAINT: begin
      hdc:=BeginPaint(hwnd, paintStruct);
      {вызовы функций GDI}
      EndPaint(hwnd, paintStruct);
    end;
  ..............
end; //case
```

Функция BeginPaint возвращает хэндл контекста устройства для окна с хэндлом hWnd, кроме того, она заполняет структуру paintStruct некоторой полезной информацией. Помимо этих действий, функция BeginPaint устанавливает регион отсечения в

соответствии с недействительным регионом, а сам недействительный регион делает действительным. Вызов BeginPaint должен производиться только при обработке сообщения WM_PAINT и обязательно должен сопровождаться вызовом функции EndPaint. Параметр paintStruct должен быть записью типа

```
TPaintStruct = packed record
    hdc: HDC;
    fErase: BOOL;
    rcPaint: TRect;
    fRestore: BOOL;
    fIncUpdate: BOOL;
    rgbReserved: array[0..31] of Byte;
  end;
```

Важными для программиста являются первые три поля этой структуры. В поле hdc дублируется хэндл полученного контекста устройства. В поле fErase содержится признак того, была ли произведена операция стирания фона окна в пределах региона отсечения, это стирание производится с использованием кисти, указанной при создании класса окна, и выполняется Windows автоматически при вызове BeginPaint. Наконец, rcPaint содержит координаты минимальной прямоугольной области, описанной вокруг региона отсечения.

Смысл параметра rcPaint в том, что за пределами этого прямоугольника рисование гарантированно не производится. В то же время, так как регион отсечения может иметь сложную форму, операции рисования могут физически не производиться и в некоторых областях внутри этого прямоугольника. Так как выполняется отсечение, обработчик WM_PAINT может просто перерисовывать окно, ни о чем не заботясь, так как все равно будет прорисована только часть изображения внутри региона отсечения. Тем не менее, использование знаний о размерах региона отсечения позволяет оптимизировать процесс рисования за счет того, что части изображения, лежащие заведомо вне rcPaint, можно просто не рисовать.

Дисциплина ожидания в очереди для сообщения WM_PAINT является особой: это сообщение имеет низший приоритет и может быть получено только в том случае, когда в очереди нет других сообщений. Кроме того, если в очереди уже присутствует сообщение WM_PAINT и приходит новое сообщение WM_PAINT для того же окна, то это приводит просто к добавлению нового региона обновления к региону обновления, соответствующему старому, еще не обработанному, сообщению WM_PAINT. При этом в очереди по-прежнему останется одно сообщение WM_PAINT. Наконец, если в очереди есть сообщение WM_PAINT, но соответствующий регион обновления каким-то образом стал корректным (например, приложение вызвало функцию ValidateRect), то сообщение WM_PAINT будет удалено из очереди, так как рисование больше не требуется. Эту логику прохождения сообщения WM_PAINT обеспечивает операционная система.

Функции рисования GDI

(Конкретные особенности использования функций см. в Help.)

Функция	Примечание
GetClientRect	Позволяет получить размеры клиентской области окна
MoveToEx	Перемещение текущей позиции пера

LineTo	Рисование отрезка из текущей позиции и перемещение пера
Ellipse	Рисование закрашенного эллипса, вписанного в указанный прямоугольник
Arc	Рисование дуги эллипса, вписанного в указанный прямоугольник
Rectangle	Рисование закрашенного прямоугольника
RoundRect	Рисование закрашенного прямоугольника со скругленными краями
PolyLine	Рисование ломаной, соединяющей указанные точки
Polygon	Рисование заполненного замкнутого многоугольника по заданным вершинам
TextOut	Выводит строку символов
DrawText	Выводит строку символов с расширенными возможностями форматирования
SetBkMode	Управляет "прозрачностью" текста
SetTextAlign	Устанавливает режим вывода текста
GetPixel	Прочитать цвет пиксела
SetPixel	Нарисовать пиксел

Задания:

1. Стандартное масштабируемое окно, в котором по центру нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны изменяться при изменении размеров окна.
2. Масштабируемое окно без заголовка, в котором по центру нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны изменяться при изменении размеров окна.
3. Два стандартных окна. Программа должна завершаться после закрытия всех окон.
4. "Главное окно" с заголовком и масштабируемое, а также два "вспомогательных" окна стиля Toolbar (плавающая панель инструментов). Программа должна завершаться после закрытия главного окна.
5. Стандартное масштабируемое окно, в левом верхнем углу которого нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны составлять 200 пикселов по горизонтали и вертикали.
6. Масштабируемое окно без заголовка, в правом нижнем углу которого нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны составлять 200 пикселов по горизонтали и вертикали.

Изучение механизмов ввода информации от клавиатуры, мыши и таймера

Операционная система Windows извещает запущенные процессы о событиях, в том числе о событиях от устройств ввода, путем посылки им сообщений. Сообщения от клавиатуры, мыши и таймера помещаются в очереди сообщений потоков, откуда извлекаются потоками-получателями и отправляются на обработку в оконные процедуры. Логика прохождения сообщений от клавиатуры, мыши и таймера заслуживает отдельного описания.

Клавиатура

Сообщения от клавиатуры можно разделить на аппаратные и символьные.

Аппаратные сообщения возникают при нажатии и отпускании клавиш на клавиатуре и содержат аппаратно-независимый виртуальный код нажатой клавиши Windows, а также скэн-код. Аппаратные сообщения помещаются в очередь сообщений процесса операционной системой.

Символьные сообщения содержат код символа (или управляющего символа), если была нажата алфавитно-цифровая клавиша. Символ зависит не только от нажатой клавиши, но и от положения клавиш Shift, CapsLock, установленного языкового драйвера. Символьные сообщения помещаются в очередь при обработке аппаратных сообщений о нажатии клавиш процедурой TranslateMessage, которая вызывается по инициативе самого процесса в цикле обработки сообщений:

```
while GetMessage(msg,0,0,0) do
begin {получить очередное сообщение}
   TranslateMessage(msg);     {Windows транслирует сообщения от клавиатуры}
    DispatchMessage(msg);         {Windows вызовет оконную процедуру}
end; {выход по wm_quit, на которое GetMessage вернет FALSE}
```

Аппаратные сообщения клавиатуры

Аппаратные сообщения от клавиатуры накапливаются в общесистемной очереди, откуда посылаются в очередь сообщений окна, имеющего фокус ввода. Пока окно не обработает сообщение, следующее сообщение не посылается. Это связано с тем, что обработка сообщения от клавиатуры может приводить к смене окна, имеющего фокус ввода, и последующие сообщения должны быть адресованы уже новому окну. Поэтому, если система по каким-либо причинам не может осуществить обработку сообщений от клавиатуры в темпе их поступления, наличие общесистемной очереди клавиатурных сообщений оказывается оправданным. (Окно извещается о получении или потере фокуса ввода посылкой ему сообщений WM_SETFOCUS и WM_KILLFOCUS.)

При нажатии клавиши окну посылается сообщение WM_KEYDOWN или WM_SYSKEYDOWN, при отпускании — WM_KEYUP или WM_SYSKEYUP. При длительном удержании клавиши нажатой сообщения о нажатии клавиши посылаются многократно. Сообщения, в идентификаторах которых фигурирует слово "SYS", свидетельствуют о том, что клавиша нажимается при нажатой клавише Alt.

Все перечисленные аппаратные сообщения от клавиатуры в параметре WPARAM имеют виртуальный код клавиши Windows. Эти коды имеют идентификаторы, начинающиеся с VK_ , и их можно найти в файле WINDOWS.PAS.

В LPARAM содержится дополнительная информация о нажатии клавиши, структура которой такова:

Биты	Размер	Значение

0..15	16	Счетчик повторений
16..23	8	Скэн-код
24	1	Флаг принадлежности расширенной клавиатуре
29	1	Флаг контекста. "0" для обычных и "1" для системных сообщений.
30	1	Предыдущее состояние клавиши. "1" - была нажата, "0" - была отпущена.
31	1	Состояние клавиши. "1" - нажата (KEYDOWN), "0" - отпущена (KEYUP).

Счетчик повторений содержит количество повторов нажатия клавиши, которое может отличаться от 1 в случае, когда пользователь нажимает и удерживает клавишу, а программа не в состоянии обработать сообщения клавиатуры в темпе их поступления. В такой ситуации Windows не помещает в очередь отдельные сообщения, а накапливает нажатия в одном единственном сообщении, увеличивая поле счетчика в параметре LPARAM.

Скэн-код клавиши предоставляется в соответствующем поле, однако программа для идентификации клавиши должна использовать ее виртуальный код, передаваемый в WPARAM.

Флаг принадлежности к расширенной клавиатуре выставляется в единицу, если нажаты такие клавиши, как правый Alt, Ctrl или Shift, "серые" клавиши со знаками арифметических операций и Enter, NumLock, а также клавиши управления курсором, не относящиеся к клавиатуре цифрового набора.

Флаг контекста показывает, нажата ли клавиша Alt в момент посылки сообщения. Когда активное окно минимизировано, оно получает все клавиатурные сообщения как системные, и чтобы определить, нажата ли действительно клавиша Alt, используется данный флаг.

Как правило, пользовательский процесс не должен реагировать на системные сообщения клавиатуры, передавая их в DefWindowProc. Чаще всего программа должна обрабатывать только сообщения WM_KEYDOWN от клавиш управления курсором.

Символьные сообщения клавиатуры

После обработки аппаратного сообщения в TranslateMessage в очередь могут быть помещены символьные сообщения WM_CHAR, WM_SYSCHAR, WM_DEADCHAR и WM_SYSDEADCHAR. Сообщения со словом "SYS" являются следствием сообщений WM_SYSKEYDOWN, без него – WM_KEYDOWN.

Символьные сообщения содержат код нажатого символа в WPARAM, а в LPARAM содержится та же информация, что и у породившего их аппаратного сообщения, в том числе — количество повторов символа.

DEADCHAR (немой символ) имеет смысл для некоторых европейских раскладок клавиатуры, когда символ с диакритическим знаком вводится путем последовательного нажатия клавиши соответствующего знака (например — апострофа) и алфавитной клавиши. В результате окно получает два символьных сообщения: WM_DEADCHAR с символом диакритического знака и, при последующем нажатии алфавитной клавиши, WM_CHAR с символом с диакритическим знаком, например символом Ó . Если введенный после немого символа алфавитный символ не может иметь диакритического

знака, окно получит не два, а три сообщения: WM_DEADCHAR и WM_CHAR с диакритическим знаком и WM_CHAR с буквой алфавита. Очевидно, что поскольку логика обработки немых символов реализована в Windows, пользовательской программе обычно нет необходимости обрабатывать сообщения о немых символах.

Состояние клавиатуры

Windows содержит ряд функций для опроса и переключения состояния клавиатуры. Эти функции возвращают не текущее состояние клавиатуры в момент их вызова, а состояние клавиатуры на момент посылки последнего извлеченного из очереди клавиатурного сообщения. Таким образом можно считать, что к каждому сообщению о нажатии клавиши "привязана" информация о состоянии всех клавиш в этот момент, и именно с этой информацией о "логическом состоянии клавиш" можно производить операции.

function GetKeyState(nVirtKey: Integer): SmallInt;

Функция GetKeyState возвращает логическое состояние клавиши с соответствующим виртуальным кодом. Старший бит 16-разрядного результата взведен, если клавиша нажата, и сброшен, если клавиша отпущена. Младший бит для клавиш управления режимом (CapsLock, NumLock, ScrollLock) показывает, активен соответствующий режим ("1") или нет ("0").

Следующие две функции позволяют прочитать и модифицировать логическое состояние клавиатуры целиком. Возвращаемое логическое значение свидетельствует об успехе или неуспехе производимой операции.

type TKeyboardState = **array**[0..255] **of** Byte;

function GetKeyboardState(**var** KeyState: TKeyboardState): BOOL;

function SetKeyboardState(**var** KeyState: TKeyboardState): BOOL;

Старший бит соответствующего виртуальному коду клавиши байта взведен, когда клавиша нажата, и сброшен, когда отпущена. Младший бит по-прежнему показывает режим для клавиш управления режимом.

Для определения физического, т.е. аппаратного состояния клавиши в текущий момент служит функция GetAsyncKeyState с параметрами, аналогичными GetKeyState.

Мышь

Использование указательного устройства в графических средах, подобных Windows, является стандартом. В качестве такового в Windows используется устройство типа "мышь" с одной, двумя или тремя кнопками. Для Windows 95 стандартной является двухкнопочная мышь.

Функция GetSystemMetrics позволяет определить следующие параметры:

getSystemMetrics(SM_MOUSEPRESENT) — не 0, если в системе установлена мышь

getSystemMetrics(SM_CMOUSEBUTTONS) — количество кнопок мыши

getSystemMetrics(SM_SWAPBUTTON) — не 0, если включено зеркальное отображение кнопок мыши (правая воспринимается как левая и наоборот).

С событиями от мыши связано 21 сообщение Windows. Если курсор мыши находится в рабочей области окна, окно получает следующие сообщения:

Движение курсора: WM_MOUSEMOVE

Нажатие кнопок:

Кнопка	Нажатие	Отпускание	Двойное (второе)

			нажатие*
Левая	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
Правая	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDBLCLK
Средняя	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK

* Сообщения о двойных щелчках окна получают лишь в том случае, когда в стиле их оконного класса указан флаг CS_DBLCLKS. В противном случае для отслеживания многократных щелчков в оконной процедуре удобно использовать функцию GetMessageTime для определения моментов посылки сообщений и интервалов между ними.

Для всех этих сообщений параметры следующие:

wParam - состояние кнопок, а также клавиш Ctrl и Shift клавиатуры; OR-комбинация констант MK_LBUTTON, MK_RBUTTON, MK_MBUTTON, MK_SHIFT и MK_CONTROL.

lParam - координаты курсора в пикселах относительно левого верхнего угла рабочей области, X=loWord(lParam); Y=hiWord(lParam);

В отличие от сообщений клавиатуры, получение сообщений от мыши определяется не активностью окна, а положением курсора мыши в области окна. Сообщения о нажатии кнопок мыши могут не сопровождаться соответствующими сообщениями об их отпускании, если курсор вдруг будет выведен за границы окна.

Windows помещает сообщения от мыши в очередь сообщений процесса только в том случае, если там нет однотипных сообщений. Из-за этого в случае перегруженности системы возможно "пропадание" некоторых действий с мышью. Например, если пользователь щелкнул кнопкой мыши в поле окна, то соответствующие сообщения помещаются в очередь, но пока они не будут извлечены оттуда программой, следующие нажатия и отпускания кнопки мыши в этом окне будут пропадать, т.е. очередные сообщения WM_xBUTTONDOWN и WM_xBUTTONUP в очередь помещены не будут.

Захват мыши

Часто окну бывает необходимо получать сообщения от мыши даже в том случае, если курсор мыши покинет пределы рабочей области. Например, если пользователь нажимает кнопку мыши для перетаскивания какого либо объекта в окне графического редактора, это окно всегда должно знать, когда кнопка мыши будет отпущена, чтобы завершить или отменить операцию перетаскивания. Для этого в обработчике сообщения WM_LBUTTONDOWN окном должен быть выполнен захват мыши, а в WM_LBUTTONUP — освобождение захвата.

Захват мыши осуществляется функцией SetCapture(hWnd), которая возвращает нулевое значение или хэндл окна, осуществлявшего захват мыши до вызова этой функции. Освобождение захвата осуществляется вызовом функции ReleaseCapture без параметров. Окно, захватившее мышь, получает сообщения от мыши, когда курсор находится вне его, только в том случае, если кнопка мыши нажата. В противном случае сообщения мыши проходят стандартным образом, невзирая на захват.

Сообщения мыши нерабочей области

Окно получает извещение о действиях с мышью и тогда, когда курсор находится на нерабочей области окна — на заголовке или на рамке. Названия этих сообщений имеют в

своем составе аббревиатуру NC (non-client), и трактовка их параметров отличается от трактовки параметров сообщений для рабочей области.

Это сообщения:

WM_NCMOUSEMOVE, WM_NCLBUTTONDOWN, WM_NCRBUTTONDOWN, WM_NCLMUTTONDOWN, WM_NCLBUTTONUP, WM_NCRBUTTONUP, WM_NCLMUTTONUP, WM_NCLBUTTONDBLCLK, WM_NCRBUTTONDBLCLK, WM_NCLMUTTONDBLCLK.

wParam - обозначает зону окна. Полный список возможных вариантов можно получить в описании сообщения WM_NCHITTEST.

lParam - как и для сообщений рабочей области — координаты курсора, но в координатной системе всего экрана, а не рабочей области окна.

Преобразование координат между клиентской и экранной системами координат осуществляется функциями

function ClientToScreen(hWnd: HWND; **var** lpPoint: TPoint): BOOL;
function ScreenToClient(hWnd: HWND; **var** lpPoint: TPoint): BOOL;

Сообщение WM_NCHITTEST

Прежде чем программа получит любое сообщение от мыши, Windows запрашивает окно о том, к какой его области относится положение курсора мыши. Для этого при возникновении каждого системного события от мыши Windows посылает соответствующему окну сообщение WM_NCHITTEST, в ответ на которое оконная процедура должна вернуть значение, определяющее, где в данном окне (в рабочей области, на рамке, на заголовке и т.д.) находится указанное Windows положение курсора. Положение курсора задается в параметре lParam так же, как и в остальных сообщениях от мыши; параметр wParam не используется. В зависимости от результатов обработки сообщения WM_NCHITTEST будут сгенерированы дальнейшие сообщения для рабочей или нерабочей области окна.

Обработка этого сообщения внутри оконной процедуры пользовательского процесса открывает ряд интересных возможностей. Например, следующий фрагмент позволяет создать окно, которое можно перетаскивать по экрану не только "ухватываясь за заголовок", но и "за рабочую область":

```
case Msg of

  .....
  wm_nchittest: begin
    result:=DefWindowProc(hwnd,msg,wparam,lparam);
      {Если попали в рабочую область, то обманываем Windows
       и говорим, что в заголовок}
    if result=HTCLIENT then result:=HTCAPTION;
  end;
  .....
end;
```

Чаще всего пользовательская программа обрабатывает:
HTCAPTION - строка заголовка окна
HTCLIENT - рабочая область
HTMENU - меню
HTNOWHERE - вне окна на свободном поле экрана

HTREDUCE - кнопка минимизации окна
HTSYSMENU - значок системного меню в левом верхнем углу окна
HTZOOM - кнопка максимизации окна

Таймер

Программа Windows может запросить операционную систему, чтобы та ставила ее в известность об истечении заданных промежутков времени. При этом операционная система создает логический таймер с заданным периодом, а программа начинает получать сообщения WM_TIMER. Следует заметить, что использование логического таймера не гарантирует точного отслеживания заданных промежутков времени; эти сообщения следует использовать для грубого задания интервалов времени в несколько секунд, а также для инициации периодического обновления отображаемой информации о выполняемом процессе, когда точность не важна, лишь бы информация менялась "примерно раз в N секунд".

Сообщения WM_TIMER имеют, подобно сообщениям WM_PAINT, низкий приоритет. Это означает, что сообщение WM_TIMER может быть извлечено из очереди только тогда, когда в очереди нет других сообщений. Если процесс не успел обработать предыдущее сообщение WM_TIMER, находящееся в очереди, то следующее сообщение WM_TIMER не ставится в очередь, и событие от таймера просто теряется.

Период логического таймера задается в миллисекундах и может лежать в интервале от 1 до $FFFFFFFF (что соответствует приблизительно 50 суткам). Однако для отсчета периода логических таймеров Windows использует аппаратный таймер компьютера, который в компьютерах IBM PC AT срабатывает 18.2 раза в секунду, т.е. с периодом приблизительно в 55 мсек. Из этого следует, во-первых, что задание периода таймера менее 55 мсек. бессмысленно, так как программа в этом случае все равно будет получать одно сообщение таймера каждые 55 мсек. Во-вторых, задаваемый период логического таймера всегда округляется вниз до ближайшего значения, соответствующего целому количеству срабатываний аппаратного таймера. Так, задав интервал в 1000 мсек., пользовательский процесс будет получать сообщения WM_TIMER каждые 989 мсек., точнее, с таким интервалом они будут попадать в его очередь сообщений, извлечь их оттуда вовремя — его собственная проблема.

Логический таймер создается и активизируется функцией SetTimer, а уничтожается (освобождается) функцией KillTimer:

function SetTimer(hWnd: THandle; nIDEvent, uElapse: integer; lpTimerFunc: pointer): integer;
function KillTimer(hWnd: THandle; uIDEvent: integer): BOOL;

Здесь hWnd - хэндл окна, создающего таймер, nIDEvent - числовой идентификатор таймера в программе, uElapse - период таймера в миллисекундах, lpTimerFunc - необязательный указатель на процедуру реакции на события от таймера (может быть nil).

SetTimer возвращает числовой идентификатор таймера или 0, если по каким-либо причинам таймер не может быть создан. В предыдущих версиях Windows количество логических таймеров в системе было ограничено (16, затем 32). В Windows 95 эти ограничения сняты, и невозможность создания таймера перестала быть обычной ситуацией.

Сообщение WM_TIMER имеет следующие параметры:

wParam - числовой идентификатор таймера.

lParam - указатель на функцию обработки, если таковая имеется.

Windows предоставляет две возможности по обработке сообщений таймера: в оконной процедуре и в специальной процедуре обработки. В первом случае в качестве lpTimerFunc в SetTimer передается nil, и сообщение WM_TIMER должно обрабатываться наравне со всеми остальными в общем операторе CASE оконной процедуры. Во втором случае в программе должна быть описана процедура со следующим набором параметров (имя — произвольное):

procedure TimerProc(hwnd:THandle; uMsg, idEvent, time: integer); **stdcall**;

Указатель на эту процедуру (@TimerProc) передается в качестве lpTimerFunc в SetTimer, и при каждом получении сообщения WM_TIMER процедура DispatchMessage будет вызывать не оконную процедуру, а процедуру TimerProc. Параметры этой процедуры — хэндл окна, тип сообщения (всегда WM_TIMER), числовой идентификатор таймера и системное время в момент получения сообщения.

Windows при использовании процедуры реакции таймера предоставляет возможность создавать таймер без окна, указав в параметре hwnd функции SetTimer значение 0. В этом случае значение параметра uIDEvent игнорируется системой, и функция возвращает номер таймера, назначенный автоматически. Этот номер должен быть впоследствии передан в KillTimer, параметр hwnd при этом также должен быть установлен в 0.

Задания:

1. Программа-секундомер. При нажатии клавиши Enter в рабочей области окна начинает отображаться отсчитываемое время. Точность отсчета - десятые доли секунды. При повторном нажатии Enter отсчет останавливается. Третье нажатие Enter обнуляет время. Окно секундомера — масштабируемое, без заголовка. Перемещение окна реализовать при помощи перетаскивания мышью "за клиентскую область".

Примечание: Пытаясь отслеживать время с точностью до 0.1 секунды, помните о невысокой точности сообщений таймера Windows.

2. В рабочей области окна существует единственный объект, представляющий собой геометрическую фигуру, отличную от прямоугольника (например, круг или треугольник). Нажатием цифровых клавиш 1..8 цвет объекта изменяется на один из 8 возможных. Стрелками осуществляется перемещение объекта на 1 пиксел в любую из 4 сторон. Предусмотреть также возможность перетаскивания объекта мышью.

Примечание: объект удобно представлять как регион, см. соответствующую группу функций в справке.

3. Программа - телетайп. Используя моноширинный шрифт (например, Courier) программа организует окно с размером клиентской области 80x25 символов. Алфавитно-цифровые символы выводятся на этот экран. Символ конца строки приводит к переходу в начало следующей строки, остальные управляющие символы обрабатывать не нужно. При достижении конца строки набор продолжается в следующей строке. При достижении конца экрана содержимое прокручивается на 1 строку, при этом верхняя строка теряется. При щелчке левой кнопкой мыши по какому-либо знакоместу весь экран телетайпа заполняется соответствующим символом, находящимся в указанной позиции. Окно очищается при нажатии клавиши Esc.

4. Программа - монитор состояния клавиатуры. В окне в произвольном виде отображается состояние (нажато-отпущено) клавиш на клавиатуре. Можно ограничиться только алфавитно-цифровыми клавишами. Отслеживание должно производиться даже тогда, когда окно не имеет фокуса ввода, т.е. следует асинхронно сканировать состояние клавиатуры. Окно при этом должно быть управляемым, т.е. приходящие в очередь сообщения должны обслуживаться.

5. "Уворачивающееся" окно. При попытке ввести курсор мыши в область, занятую окном (в том числе в неклиентскую область) окно изменяет свое положение таким образом, чтобы курсор мыши оказался вне окна. Окно не должно полностью уходить за границы экрана, а производимый сдвиг должен быть по возможности минимальным.

6. "Вручную", т.е. без собственной обработки WM_NCHITTEST, реализовать перетаскивание окна без заголовка "за клиентскую область". При этом желательно, чтобы в процессе перетаскивания текущее положение окна отображалось прямоугольной рамкой. Закрывать окно по нажатию клавиши Esc.

7. Прототип программы-"калькулятора", складывающей целые неотрицательные числа. В рабочей области отображается "индикатор" и 10 "клавиш" с цифрами, клавиши "Сброс(Esc)", "+" и "=". Последовательность применения калькулятора: набрать цифры первого числа (не более 8), нажать "+", набрать цифры второго числа (не более 8), нажать "=" - отобразится сумма, а калькулятор готов к приему очередного первого числа. При нажатии клавиши "Сброс" калькулятор переводится в состояние ожидания ввода первого числа, а сумма становится нулевой. Реализовать возможность ввода управляющих воздействий как с клавиатуры, так и мышью с помощью нарисованных в рабочей области клавиш.

Примечание: Удобно было бы реализовать ввод с клавиатуры, а затем при щелчке мышью по нарисованной клавише посылать окну сообщение WM_CHAR с кодом соответствующей клавиши клавиатуры. Пусть окно будет не масштабируемым.

8. Окно с изменяемым цветом фона, задаваемым в формате RGB (красный-зеленый-синий). Клавиши с буквами R, G и B ответственны за изменение каждой из составляющих цвета: если нажата клавиша Shift и "R", "G" или "B", то соответствующая составляющая цвета фона увеличивается на 1, при отпущенной клавише Shift - уменьшается. Из трех клавиш "R", "G" и "B" допускается одновременно нажимать (и удерживать) более одной, тогда должны изменяться несколько цветовых составляющих сразу. Цветовая составляющая не должна становиться меньше 0 и больше 255, т.е. выходить за диапазон BYTE. Предусмотреть вывод текущих значений составляющих цвета (допустимо использовать функцию IntToStr из модуля SysUtils в составе Delphi).

Примечание: в ответ на сообщение WM_KEYDOWN необходимо проверять состояние всех клавиш клавиатуры.

9. "Рисование" мышью. При нажатии левой кнопки мыши производится рисование точки. При движении мышью с нажатой левой кнопкой производится рисование линии по координатам, проходимым мышью. При нажатии правой кнопки производится заливка области (см. FloodFill). Цвет пера и кисти выбирается из 8 возможных при нажатии соответственно клавиш 1..8 и F1..F8.

Примечание: не требуется реализовывать запоминание рисунка и его восстановление в случае, если рабочая область окна по каким-то причинам станет недействительной.

10. Программа-"метроном". Рабочая область окна (или ее участок) должны периодически изменять цвет (например, с черного на белый и наоборот) через постоянные интервалы времени, например — каждую секунду. Допустимый интервал — от 0.5 секунды до 3 секунд. Изменение интервала с шагом в 0.1 секунду производится при нажатии клавиш "стрелка вверх" и "стрелка вниз", с шагом 0.5 сек. — при нажатии тех же клавиш в комбинации с клавишей Ctrl.

Изучение средств Windows для построения пользовательского интерфейса

Оконные органы управления (controls)

В системе Windows реализованы средства для создания современного пользовательского интерфейса, включающего такие элементы как кнопки (buttons), списки (listboxes), полосы прокрутки (scrollbars), окна редактирования (editboxes) и т.д. Все эти элементы управления в программе должны порождаться как дочерние окна окон программы - экземпляры предописанных классов. Такими классами являются (перечисляются идентификаторы, используемые в процедуре CreateWindow):

button	- кнопка, переключатель (radio button), флажок (checkbox).
static	- просто прямоугольная область.
scrollbar	- горизонтальная или вертикальная полоса прокрутки.
edit	- поле редактирования, одно- или многострочное.
listbox	- список.
combobox	- выпадающий список или список, комбинированный со строкой редактора.

При создании органа управления в процедуру CreateWindow, в отличие от создания обычного окна программы, должны быть переданы следующие параметры:

```
function CreateWindow(
   lpClassName: PChar;   // имя класса окна управления, малыми буквами, как представлено выше
    lpWindowName: PChar; // текст окна (надпись на кнопке и т.д.)
     dwStyle: DWORD;    // Стиль окна должен включать флаги WS_CHILD и (обычно)
                        // WS_VISIBLE + стиль органа управления
   X, Y: integer      // Положение и размеры органа управления в координатах
                        // рабочей области родительского окна
   nWidth, nHeight: Integer;  // Ширина и высота
   hWndParent: HWND; // Хэндл окна-родителя
     hMenu:  HMENU;           // Идентификатор (номер) органа управления, определяется
                        // программистом
   hInstance: HINST;  // Экземпляр приложения, создающий окно
   lpParam: Pointer   // nil
 ): HWND;
```

Дочерние окна обрабатывают сообщения от мыши и клавиатуры. Когда пользователь производит с органом управления какие-либо действия, окну-родителю посылается сообщение WM_COMMAND, содержащее информацию о производимом действии и идентификатор (номер) окна управления, присвоенный ему при создании. Оконная процедура родительского окна должна уметь обрабатывать такие сообщения. Параметры сообщения WM_COMMAND:

loword(wParam)	- идентификатор дочернего окна управления
hiword(wParam)	- код уведомления (операции)
lParam	- хэндл дочернего окна управления.

Родительское окно управляет дочерними, тоже посылая им сообщения. Эти сообщения зависят от класса окон управления.

Кроме того, для управления дочерними окнами часто используются функции ShowWindow, MoveWindow, IsWindowVisible, EnableWindow, IsWindowEnabled, SetWindowText, GetWindowText. Подробнее об этих функциях см. Справку по Win32 API.

Для переключения фокуса ввода между окнами управления в диалогах обычно используются клавиши Tab и Shift-Tab. Windows содержит средства для автоматизации этого действия без значительного усложнения пользовательской программы.

Дочерние окна управления автоматически уничтожаются при уничтожении родительского окна.

Цикл обработки сообщений для диалогов

Чаще всего оконными органами управления "оборудуются" окна, предназначенные исключительно для организации диалога с пользователем — ввода значений, установки параметров и т.д. Такие окна называют диалоговыми или просто диалогами. Windows предоставляет программисту возможность без особых затрат труда организовать переключение органов управления при нажатии пользователем стандартных комбинаций клавиш (самые распространенные комбинации — Tab и Shift-Tab). Для этого цикл обработки сообщений программы должен быть модифицирован следующим образом (это может быть как основной цикл программы, так и специально организуемый для конкретного диалога цикл обработки сообщений).

Пусть хэндл диалогового окна хранится в переменной hDlg. тогда цикл обработки сообщений должен выглядеть как:

```
  while GetMessage(msg,0,0,0)=true do begin {получить очередное сообщение}
    if not IsDialogMessage(hDlg,msg)
                  {Если Windows не распознает и не обрабатывает клавиатурные сообщения
                как команды переключения между оконными органами управления,
              тогда чообщение идет на стандартную обработку}
    then begin
      TranslateMessage(msg);    {Windows транслирует сообщения от клавиатуры}
           DispatchMessage(msg);        {Windows вызовет оконную процедуру}
    end;
  end; {выход по wm_quit, на которое GetMessage вернет FALSE}
```

Функция IsDialogMessage в этом случае анализирует текущее сообщение MSG и, если сообщение содержит информацию о нажатии стандартной комбинации клавиш, служащей для перемещения между органами управления диалога, обрабатывает его. Если сообщение обрабатывается, функция возвращает TRUE и программа обрабатывать такое сообщение уже не должна, в противном случае возвращается FALSE и сообщение обрабатывается на общих основаниях.

В ходе выполнения функции IsDialogMessage Windows проверяет, есть ли в окне с указанным хэндлом (hDlg) оконные органы управления вообще, и если они есть, то выполняется последовательная посылка ряда сообщений им для смены текущего органа управления.

Кнопки (класс button)

Поддерживаются следующие стили кнопок:

Стиль	Характеристика
BS_PUSHBUTTON	Обычная кнопка ("нажимаемая кнопка").

BS_DEFPUSHBUTTON	Кнопка с более жирной рамкой. В диалогах, описываемых в ресурсах, считается нажатой при нажатии пользователем клавиши Enter.
BS_CHECKBOX	Флажок, т.е. надпись, слева или справа от которой расположено квадратное поле, которое может быть отмечено крестиком.
BS_AUTOCHECKBOX	То же, что предыдущее, но отметка или снятие отметки происходит автоматически, не требуя участия родительского окна.
BS_3STATE	То же, что флажок, но возможно также "третье состояние", когда поле флажка отображается серым цветом.
BS_AUTO3STATE	То же, что предыдущее, но переключение между тремя возможными состояниями происходит автоматически при щелчке по флажку.
BS_RADIOBUTTON	Индикатор исключающего выбора: кружок, слева от которого размещена строка текста. Кружок может быть отмечен точкой в центре.
BS_AUTORADIOBUTTON	То же, что предыдущее, но при щелчке по индикатору он автоматически становится выбранным, а все другие индикаторы в той же группе (того же родительского окна) — невыбранными.
BS_GROUPBOX	Рамка с текстом в верхнем левом углу, обычно служит для визуального объединения радио-кнопок в группу. Само по себе не реагирует на события от клавиатуры и мыши.
BS_OWNERDRAW	Кнопка пользователя, отрисовкой внешнего вида кнопки должна заниматься оконная процедура родительского окна, обрабатывая сообщение WM_DRAWITEM.

При нажатии кнопок в сообщении WM_COMMAND содержится код уведомления BN_CLICKED.

Родительское окно может посылать окнам кнопок управляющие сообщения:

Сообщение	**Описание**
BM_GETCHECK	Возвращает для флажков и индикаторов выбора состояние выбора: 0 (не помечено), 1 (помечено), 2 (третье состояние).
BM_SETCHECK	Устанавливает состояние выбора для флажков и индикаторов выбора. В wParam должен содержаться код состояния (см.

выше).

BM_GETSTATE	Возвращает состояние как комбинацию битовых флагов: биты 0 и 1 — содержат код, возвращаемый BM_GETCHECK; бит 2 — нажата (1) или отпущена (0) кнопка; бит 3 — имеет ли кнопка фокус.
BM_SETSTATE	Для нажимаемых кнопок устанавливает состояние. wParam = 0 — кнопка отпущена, wParam = 1 — кнопка нажата.
BM_SETSTYLE	Позволяет изменить стиль кнопки после ее создания. wParam - содержать новый стиль как комбинация битовых флагов, lParam - 1 - перерисовать кнопку, 0 - не перерисовывать.

Например, изменить состояние окна-флажка с хэндлом hCheckBox1 на противоположное можно с помощью следующей конструкции:

```
SendMessage(hCheckBox1, BM_SETCHECK,
                                SendMessage(hCheckBox1,
BM_GETCHECK,0,0) xor 1,
                0);
```

Для того, чтобы изменить способ отображения кнопок или их реакцию на внешние события, необходимо создать собственный оконный класс на основе класса button (рассматривается ниже).

Статические окна (класс static)

Эти окна отображаются как закрашенные прямоугольники или рамки с текстом. Обычно эти окна не обрабатывают событий от клавиатуры и мыши. Они могут использоваться как контейнеры для других дочерних окон управления. Подробнее см. Справку по Win32 API, функция CreateWindow.

Полосы прокрутки

Окна Windows могут иметь оконные горизонтальные и вертикальные полосы прокрутки, что определяется включением в стиль окна флагов WS_VSCROLL и WS_HSCROLL. Помимо этого существуют также дочерние окна управления класса scrollbar, которые выглядят точно так же, но, в отличие от полос прокрутки окна, могут располагаться в любом месте и иметь произвольный размер. Далее будет рассмотрено использование обоих типов полос прокрутки.

Окна с полосами прокрутки (полосы — не дочерние окна)

Чтобы окно имело вертикальную полосу прокрутки, в его стиле при вызове функции CreateWindow должен присутствовать флаг WS_VSCROLL, горизонтальную — WS_HSCROLL. Такие полосы прокрутки (назовем их оконными) размещаются в окне вдоль правой и нижней границ и имеют протяженность вдоль всей границы окна в пределах рабочей области. Пространство, занятое оконными полосами прокрутки, не включается в рабочую область. Ширина полос прокрутки является общесистемным

параметром и ее можно узнать с помощью функции GetSystemMetrics (см. также SystemParametesInfo).

Оконные полосы прокрутки управляются при помощи мыши, они не реагируют на нажатия клавиш, поэтому программист должен "вручную" реализовывать прокрутку при нажатии на клавиши управления курсором в оконной процедуре.

Полосы прокрутки характеризуются диапазоном, определяемым двумя целыми числами — минимальным и максимальным положением бегунка. Положение бегунка всегда дискретно и соответствует целому числу из диапазона возможных положений от минимального до максимального. Так, если диапазон задан как "от 0 до 100" (по умолчанию), то бегунок может находиться в одном из 101 возможных положений, и его положение будет возвращаться как целое число от 0 до 100. Диапазон по умолчанию не всегда удобен, поэтому есть возможность переопределить его при помощи вызова функции

```
function SetScrollRange(hWnd: HWND; nBar, nMinPos,
                        nMaxPos: Integer; bRedraw: BOOL):
                        BOOL;
```

Узнать текущий диапазон оконной полосы прокрутки можно с помощью функции

```
function GetScrollRange(hWnd: HWND; nBar: Integer;
                        var nMinPos, nMaxPos: Integer):
                        BOOL;
```

Здесь hWnd - хэндл окна, nBar - SB_VERT или SB_HORZ, nMinPos и nMaxPos - минимальное и максимальное положение полос прокрутки, bRedraw - определяет, требуется ли перерисовка полосы прокрутки сразу после установления новых параметров. При неуспешном завершении операции возвращается FALSE.

Если не выполняется условие nMinPos<nMaxPos, то полоса прокрутки становится невидимой.

Для работы с положением бегунка используются функции

```
function SetScrollPos(hWnd: HWND; nBar, nPos: Integer;
bRedraw: BOOL): Integer;
function GetScrollPos(hWnd: HWND; nBar: Integer):
Integer;
```

Здесь hWnd — хэндл окна, nBar - SB_VERT или SB_HORZ, bRedraw - определяет, требуется ли перерисовка полосы прокрутки сразу после установления новых параметров. Возвращаемое значение — старое положение бегунка (оно же во втором случае — текущее).

Сообщения полос прокрутки

При действиях с полосами прокрутки оконной процедуре посылаются сообщения WM_VSCROLL (вертикальная прокрутка) и WM_HSCROLL (горизонтальная прокрутка). В младшем слове wParam содержится код операции с полосой прокрутки, lParam для сообщений оконных полос прокрутки можно игнорировать. Кодами операции в младшем слове wParam могут быть:

SB_BOTTOM	Прокрутить в самый конец
SB_ENDSCROLL	Кнопка мыши отпущена, операция завершена (это

	сообщение как правило можно игнорировать)
SB_LINEDOWN	Прокрутка вниз на одну строку, возникает при нажатии на стрелку внизу (справа) полосы прокрутки.
SB_LINEUP	Прокрутка вверх на одну строку, возникает при нажатии на стрелку вверху (слева) полосы прокрутки.
SB_PAGEDOWN	Прокрутка на страницу вниз, возникает при щелчке мыши на полосе прокрутки ниже (справа от) бегунка.
SB_PAGEUP	Прокрутка на страницу вверх, возникает при щелчке мыши на полосе прокрутки выше (слева от) бегунка.
SB_THUMBPOSITION	Установить позицию бегунка. Позиция содержится в старшем слове wParam. Возникает после перетаскивания бегунка мышью.
SB_THUMBTRACK	Позиция в время перетаскивания. Позиция содержится в старшем слове wParam. Возникает во время перетаскивания ползунка мышью.
SB_TOP	Прокрутка в самое начало.

Как видно из приведенной таблицы, положение бегунка в сообщениях SB_THUMBPOSITION и SB_THUMBTRACK передается в виде 16-разрядного целого, поэтому "полнофункциональная" полоса прокрутки должна иметь диапазон, определяемый числами в диапазоне от 0 до 65535. Функции SetScrollRange, SetScrollPos, GetScrollRange и GetScrollPos оперируютс 32-разрядными целыми, однако положение бегунка, выходящее за 16-разрядный диапазон, не может быть корректно передано в теле сообщения. При обработке сообщения SB_THUMBPOSITION есть возможность узнать 32-разрядное положение бегунка при помощи функции GetScrollPos, однако в случае SB_THUMBTRACK с ее помощью узнать положение бегунка невозможно, так как функция GetScrollPos не отражает положения бегунка в процессе его перетаскивания.

При перетаскивании бегунка в очередь сообщений окна помещается множество сообщений с признаком SB_THUMBTRACK, после чего по окончании операции посылается сообщение с SB_THUMBPOSITION.

Простейший способ реакции на сообщения WM_VSCROLL и WM_HSCROLL — вызов функции ScrollWindow для прокрутки клиентской области окна. Прокрутка состоит в том, что изображение в окне соответствующим образом сдвигается, а "открывшиеся" после сдвига участки становятся недействительными, и в очередь сообщений помещается сообщение WM_PAINT. Дочерние окна и начало координат окна также сдвигаются.

Полосы прокрутки — окна класса scrollbar

Полоса прокрутки может быть помещена в произвольном месте окна. Для этого необходимо создать ее как дочернее окно управления класса scrollbar, указав стиль SBS_VERT или SBS_HORZ (см. Справку по WIN32 API: CreateWindow). Полоса прокрутки при этом занимает весь указанный в параметрах CreateWindow (или MoveWindow) прямоугольник, т.е. ее ширина и высота могут быть произвольными. Системные настройки ширины стандартных оконных полос прокрутки можно узнать, вызвав функции GetSystemMetrix(SM_CYHSCROLL) или GetSystemMetrix(SM_CXVSCROLL).

Окна класса scrollbar, как и стандартные полосы прокрутки окна, посылают родительскому окну сообщения WM_VSCROLL или WM_HSCROLL вместо сообщений WM_COMMAND, посылаемых остальными оконными органами управления. Отличия этих сообщений от сообщений, посылаемых оконными полосами прокрутки, состоят в том, что для оконных полос прокрутки lParam содержит 0, а для дочерних окон - полос прокрутки — хэндл такого дочернего окна. В отличие от стандартных оконных полос прокрутки, окна класса scrollbar могут получать фокус и обрабатывать нажатия клавиш клавиатуры. Нажатия клавиш интерпретируются следующим образом:

Клавиша	**wParam сообщения VM_xSCROLL**
Home	SB_TOP
End	SB_BOTTOM
Page Up	SB_PAGEUP
PageDown	SB_PAGEDOWN
Стрека влево, стрелка вверх	SB_LINEUP
Стрелка вправо, стрелка вниз	SB_LINEDOWN

Управление полосами прокрутки класса scrollbar осуществляется теми же функциями GetScrollPos, GetScrollRange, SetScrollPos и SetScrollRange, что и для стандартных оконных полос прокрутки, однако в параметре hWnd следует указывать хэндл самой полосы прокрутки, а в nBar — константу SB_CTL.

Окна редактирования (класс edit)

Окна редактирования представляют собой одно- или многострочные текстовые редакторы. Они могут иметь клавиатурный фокус и позволяют вводить текст, редактировать его при помощи клавиш управления, выделять при помощи клавиш и мыши и производить операции копирования и вставки с буфером обмена при помощи клавиш Ctrl-Ins, Shift-Ins и Shift-Del. Окна редактирования могут иметь собственные стандартные полосы прокрутки, что определяется включением флагов WS_VSCROLL и WS_HSCROLL в стиль окна редактирования при его создании.

Окно редактирования создается как экземпляр класса edit. В стиле окна могут присутствовать следующие специфичные для окон-редакторов флаги:

ES_MULTILINE	Многострочный редактор.
ES_AUTOHSCROLL	Автоматически используется горизонтальная полоса прокрутки, если текст не умещается по ширине окна. При отсутствии этого флага производится автоматический перенос на следующую строку.
ES_AUTOVSCROLL	Автоматически появляется вертикальная полоса прокрутки, если текст не умещается по высоте окна.
ES_NOHIDESEL	Выделенный текст подсвечивается, даже когда окно теряет фокус, иначе при потере фокуса подсветка текста гасится.

С помощью функции SetWindowText можно задать текст в окне редактирования, с помощью GetWindowtextLength и GetWindowText — получить текст. Многострочные окна редактирования воспринимают последовательность #13#10 как конец строки; следующие

символы выводятся со следующей строки, а символы конца строки не отображаются. Окна редактирования не обрабатывают символ табуляции (#9) и отображают его в виде вертикальной черты ("|"). Чтобы в обычном окне (не в диалоге) обрабатывать нажатия клавиш Tab, Shift-Tab, Enter как управляющие, необходимо вводить новую оконную процедуру для окон редактирования (см. ниже). Размер редактируемого текста ограничен примерно 32 килобайтами, при этом используется область памяти, выделенная приложению, содержащему окно редактора.

Окна редактора посылают родительскому окну сообщения WM_COMMAND со следующими кодами уведомления в старшем слове wParam:

EN_SETFOCUS	Редактор получил фокус ввода
EN_KILLFOCUS	Редактор потерял фокус ввода
EN_CHANGE	Содержимое изменилось, посылается после вывода на экран
EN_UPDATE	Содержимое изменилось, посылается перед обновлением экрана
EN_ERRSPACE	Буфер редактирования переполнился
EN_MAXTEXT	Буфер редактирования переполнился при вставке

Программа может управлять окном редактора, посылая ему сообщения. Наиболее важные из них:

EM_GETLINECOUNT — возвращает число строк в многострочном редакторе.

EM_LINEINDEX, wParam = номер строки — возвращает смещение начала строки относительно начала буфера редактирования. Строки нумеруются с нуля. wParam=-1 позволяет узнать смещение строки, содержащей курсор.

EM_LINELENGTH, wParam = номер строки — возвращает длину указанной строки.

EM_GETLINE, wParam = номер строки, lParam = указатель на буфер — копирует указанную строку в заданный буфер, заканчивая ее нулем.

См. также сообщения WM_CUT, WM_COPY, WM_PASTE, EM_GETSEL, EM_SETSEL, EM_REPLACESEL, с помощью которых реализуются операции с буфером обмена.

Списки (класс listbox)

Окна класса listbox представляют собой списки Windows. Список — это набор текстовых строк, которые выводятся в прямоугольном прокручиваемом окне в один или несколько столбцов. Окно списка может получать фокус и способно обрабатывать сообщения клавиатуры. Когда окно списка имеет фокус, одна из строк отображается с пунктирной рамкой и считается имеющей фокус. Кроме того, строки могут быть выделены цветом, в этом случае они считаются выбранными. Наличие у строки фокуса в общем случае не означает, что она выбрана.

По умолчанию списки не посылают родительскому окну сообщения WM_COMMAND; чтобы посылка извещений происходила, необходимо включать в стиль окна списка флаг LBS_NOTIFY. Другими важными флагами стиля являются:

LBS_SORT	Строки списка автоматически сортируются
LBS_MULTIPLESEL	Позволяет выделять несколько строк списка (список с множественным выбором)

Для того, чтобы список имел рамку и полосу прокрутки, необходимо включить в стиль окна списка флаги WS_BORDER и WS_VSCROLL. Интересные результаты получаются при использовании с окном списка флагов WS_CAPTION, WS_SIZEBOX, WS_HSCROLL.

Для управления списком используются следующие сообщения, посылаемые окну списка при помощи функции SendMessage:

LB_ADDSTRING, lParam = указатель на Z-строку — добавляет строку в конец несортированного списка, в сортированном списке строка оказывается на нужном месте в соответствии с алгоритмом сортировки.

LB_INSERTSTRING, wParam = позиция, lParam = указатель на Z-строку — вставляет строку в указанной позиции, все остальные строки сдвигаются. Если список сортированный, пересортировка не производится. При wParam=-1 строка добавляется в конец списка.

LB_DELETSTRING, wParam = позиция — удаляет строку в заданной позиции.

LB_RESETCONTENT — полная очистка списка.

WM_SETREDRAW, wParam = 0 - разрешить, 1 - запретить — устанавливает флаг, разрешающий или запрещающий перерисовку списка при изменении его содержимого. При последовательном добавлении или удалении нескольких строк полезно сбросить этот флаг перед началом операции и взвести по окончании.

LB_GETCOUNT — возвращает количество элементов в списке.

LB_SETCURSEL, wParam = номер строки — в списке с одиночным выбором делает указанную строку выбранной, при wParam=-1 ни одна строка не выбрана.

LB_SELECTSTRING, wParam = номер первой строки, lParam = указатель на Z-строку для поиска — делает выбранной первую подходящую строку, начиная с wParam. Если wParam=-1, то поиск начинается с начала списка. Выбирается строка, начинающаяся со строки, задаваемой lParam. При невозможности найти подходящую строку возвращается LB_ERR.

LB_GETCURSEL — возвращает номер текущей выбранной строки или -1.

LB_GETTEXTLEN, wParam = номер строки — возвращает длину указанной строки.

LB_GETTEXT, wParam = номер строки, lParam = адрес буфера — записывает в буфер заданную строку списка, завершая ее нулем.

LB_SETSEL, wParam = 0 - снять выделение, 1 - включить выделение, lParam - номер строки — в списках с множественным выбором выделяет или снимает выделение указанного элемента списка, если lParam=-1, то выделяется или снимается выделение у всех элементов списка.

LB_GETSEL, wParam = номер строки — возвращает 0, если строка не выбрана, и другое значение, если выбрана.

Окна списков в свою очередь извещают родительские окна о действиях с ними путем посылки им сообщений WM_COMMAND со следующими кодами уведомления в старшем слове wParam:

LBN_ERRSPACE	При добавлении строки не хватило памяти.
LBN_SELCHANGE	Изменен текущий выбор.
LBN_DBLCLICK	Был двойной щелчок мышью по списку.
LBN_SETFOCUS	Список получил фокус.
LBN_KILLFOCUS	Список потерял фокус.

Создание собственных окон управления. Subclassing и superclassing

Windows позволяет программисту создавать собственные оконные классы на основе уже существующих и модифицировать поведение уже созданных окон. В Справке по Win 32 API (из Software Development Kit) при описании этих вопросов используются термины subclassing (создание подклассов) и superclassing (создание суперклассов). Эти термины представляются не очень удачными, поскольку в объектно-ориентированном программировании термин "суперкласс" имеют несколько иное значение, а здесь наблюдается прямая аналогия с принципами ООП. "Subclassing" в документации по Windows означает подмену оконной процедуры для конкретного окна или для всего существующего класса другой оконной процедурой, "superclassing" — создание нового оконного класса на основе уже существующего с заменой оконной процедуры. (В ООП суперклассом принято называть класс-предок, в документации же по Windows под суперклассом подразумевается надстройка над существующим классом, т.е. то, что в ООП называлось бы классом-потомком.)

Замена оконной процедуры существующего окна

Этот прием используется, когда необходимо модифицировать поведение существующего окна. Используя его можно, например, сделать так, чтобы некоторые клавиатурные сообщения от дочернего окна управления, имеющего фокус, попадали в родительское окно (например, о нажатиях клавиш Tab и Enter). Для этого необходимо:

1. Описать новую оконную процедуру, предусмотрев в ней новую обработку для нужных сообщений; остальные сообщения должны передаваться старой оконной процедуре.

2. Прочитать адрес заменяемой оконной процедуры из локальной памяти окна.

3. Записать адрес новой оконной процедуры в локальную память окна.

Новая оконная процедура может поступать с сообщениями следующим образом:

- передавать их на обработку старой оконной процедуре;

- модифицировать сообщения и передавать их старой процедуре;

- обрабатывать сообщения самостоятельно, не передавая старой процедуре.

Вместо вызова DefWindowProc в новой оконной процедуре сообщения как правило передаются старой оконной процедуре при помощи функции

```
function CallWindowProc(lpPrevWndFunc: TFNWndProc;
        hWnd: HWND; Msg: UINT; wParam: WPARAM; lParam: LPARAM):
LRESULT;
```

Первый параметр этой функции — указатель на старую оконную процедуру, остальные параметры — параметры для вызова оконной процедуры.

Для получения адреса оконной процедуры окна hwnd используется вызов

```
pOldProc := pointer( GetWindowLong(hwnd, GWL_WNDPROC) );
```

получаемое 32-битное целое должно быть затем преобразовано к типу pointer. Можно совместить получение адреса старой оконной процедуры с заменой его на новый, так как функция SetWindowLong возвращает старое значение изменяемой ячейки памяти окна:

```
pOldProc := pointer( SetWindowLong(hwnd, GWL_WNDPROC, integer(@NewWndProc) ) );
```

Создание нового оконного класса на основе существующего

Часто бывает необходимо изменить поведение не одного окна, а всех окон какого-либо класса, например, чтобы все кнопки передавали родительскому окну на обработку сообщения от клавиатуры. Для этого можно изменить оконную процедуру у каждой из создаваемых кнопок, но логичнее было бы создать новый класс кнопок, которые ведут себя так же, как и кнопки button, но еще и отдают сообщения от клавиатуры главному окну.

Для этого необходимо:

1. Получить описание базового класса.

2. Модифицировать описание базового класса, создав таким образом описание нового класса. В частности, обычно вводится новая оконная процедура, обрабатывающая часть сообщений и передающая остальные старой процедуре при помощи вызова CallWindowProc, как это описано выше.

3. Зарегистрировать новый класс в системе.

4. Использовать зарегистрированный класс для порождения окон.

Описание класса в виде структуры TWndClassEx можно получить с помощью функции GetClassInfoEx, затем в полученной структуре заменяются поля указателя на оконную процедуру и имени класса. Указатель на оконную процедуру базового класса должен быть сохранен для ее вызова из оконной процедуры нового класса. Модифицированное описание класса регистрируется с помощью RegisterClassEx.

Задания:

1. "Регулятор цвета". Окно, содержащее три полосы прокрутки и три информационных (не модифицируемых) поля ввода для красной, зеленой и синей составляющей цвета. При перемещении полосы прокрутки значение соответствующей цветовой составляющей должно отображаться в соответствующем информационном окошке. Заданный цвет должен произвольным образом отображаться в рабочей области главного окна. Каждая составляющая — число от 0 до 255.

2. "Регулятор цвета". Окно, содержащее три информационных (не модифицируемых) полосы прокрутки и три поля ввода для красной, зеленой и синей составляющей цвета. После ввода составляющей положение соответствующей полосы

прокрутки должно изменяться. При неверном вводе значения должно выдаваться сообщение об ошибке. Заданный цвет должен произвольным образом отображаться в рабочей области главного окна. Каждая составляющая — число от 0 до 255.

3. "Калькулятор" с числовыми кнопками, полем индикации, клавишами четырех арифметических действий, стирания и равенства. Все клавиши должны быть реализованы в виде органов управления - кнопок.

4. "Список с поиском". Окно содержит список (изначально чем-то заполненный) и поле ввода. При выборе элемента списка в поле ввода должна появляться выбранная строка. В процессе набора в поле ввода подсветка в списке должна перемещаться к строке, начало которой совпадает с вводимой строкой. если совпадения нет, подсветка должна исчезать.

5. "Ханойская башня" или что-то отдаленно ее напоминающее. Три списка, строку из конца каждого списка можно переносить в конец любого другого списка. Проверок строк на допустимость переноса выполнять не требуется. Предусмотреть органы управления, позволяющие задать, в какой список (и из какого, если это требуется) осуществляется перенос.

6. "Транслитератор". В многострочном редакторе пользователь имеет возможность набрать произвольный русский текст ограниченной длины (2000 символов). После нажатия соответствующей кнопки текст переносится в другой редактор, при этом все русские буквы заменяются на эквивалентные латинские буквы или буквосочетания, например В→V, Ж→ZH и т.д. Редактор с транслитерированным текстом также допускает редактирование.

7. "Транслитератор". В многострочном редакторе пользователь имеет возможность набрать произвольный русский текст ограниченной длины (2000 символов). В процессе ввода (при вводе и удалении символов) текст подвергается транслитерации, при этом все русские буквы заменяются на эквивалентные латинские буквы или буквосочетания, например В→V, Ж→ZH и т.д. Транслитерированный текст отображается в другом поле ввода, редактирование в котором запрещено.

8. В программе два окна — "администратор" и "диалог". В окне администратора пользователь имеет список паролей, в который может добавлять пароль через строку редактора или удалять его (предусмотреть как). В окне диалога пароль вводится в "секретном" режиме (символы заменяются звездочками), после нажатия кнопки "Проверка" пользователю выдается сообщение (например, "Доступ открыт" и "Доступ запрещен"), свидетельствующее о наличии или отсутствии введенного пароля в списке паролей.

9. "Калькулятор" для перевода целых чисел из одной системы счисления в другую. В поле ввода набирается число в заданной системе счисления. В поле вывода появляется преобразованное число. Поддерживать системы счисления с основанием 2, 8, 10, 16. Предусмотреть орган управления для задания системы счисления.

10. "Создатель окон". Пользователь вводит параметры окна, которое он хочет создать, нажимает соответствующую кнопку, и окно с заданными параметрами появляется на экране. Числовые атрибуты вводить в редакторах, остальные выбирать из списков возможных значений. Предусмотреть как минимум ввод характеристик:

> ширина-высота, положение на экране, наличие заголовка, сист. меню, кнопок минимизации и максимизации, типа рамки, ввод строки заголовка, цвет клиентской области (хотя бы черный-белый-серый).

Лабораторная работа 11.
Создание MDI приложений.

Термин MDI (Multiple Document Interface) дословно означает многодокументный интерфейс и описывает приложения, способные загрузить и использовать одновременно несколько документов или объектов. Примером такого приложения может служить диспетчер файлов (File Manager).
Обычно MDI-приложения состоят минимум из двух форм — родительской и дочерней. Свойство родительской формы FormStyle установлено равным fsMDIForm. Для дочерней формы установите стиль fsMDIChild.
Родительская форма служит контейнером, содержащим дочерние формы, которые заключены в клиентскую область и могут перемещаться, изменять размеры, минимизироваться или максимизироваться. В вашем приложении могут быть дочерние формы разных типов, например одна — для обработки изображений, а другая — для работы с текстом.

Создание форм

В MDI-приложении, как правило, требуется выводить несколько экземпляров классов формы. Поскольку каждая форма представляет собой объект, она должна быть создана перед использованием и освобождена, когда в ней больше не нуждаются. Delphi может делать это автоматически, а может предоставить эту работу вам.

Автоматическое создание форм

По умолчанию при запуске приложения Delphi автоматически создает по одному экземпляру каждого класса форм в проекте и освобождает их при завершении программы. Автоматическое создание обрабатывается генерируемым Delphi кодом в трех местах. Первое — раздел интерфейса в файле модуля формы.

```
type
TForm1 = class (TForm)
private
{Закрытые объявления.}
public
{Открытые объявления.}
end;
```

В данном фрагменте кода объявляется класс TForm1.
Вторым является место, в котором описывается переменная класса.

```
var Form1: TForm1;
```

Здесь описана переменная Form1, указывающая на экземпляр класса TForm1 и доступная из любого модуля. Обычно она используется во время работы программы для управления формой.
Третье место находится в исходном тексте проекта, доступ к которому можно получить с помощью меню View/ Project Source. Этот код выглядит как:

```
Application.CreateForm(TForm1, Form1);
```

Процесс удаления форм обрабатывается с помощью концепции владельцев объектов: когда объект уничтожается, автоматически уничтожаются все объекты, которыми он владеет. Созданная описанным образом форма принадлежит объекту Application и уничтожается при закрытии приложения.

Динамическое создание форм
Хотя автоматическое создание форм полезно при разработке SDI-приложений, при создании MDI-приложении оно, как правило, неприемлемо.
Для создания нового экземпляра формы используйте конструктор Create класса формы. Приведенный ниже код создает новый экземпляр TForm1 во время работы программы и устанавливает его свойство Caption равным 'New Form'.

```
Form1:= TForm1.Create(Application);
Form1.Caption:= 'New Form';
```

Конструктор Create получает от вас в качестве параметра потомка TComponent, который и будет владельцем вашей формы. Обычно в качестве владельца выступает Application, чтобы все формы были автоматически закрыты по окончании работы приложения. Вы можете также передать параметр Nil, создав форму без владельца (или владеющую собой — как вам больше нравится), но тогда закрывать и уничтожать ее придется вам. В случае возникновения необрабатываемой ошибки такая форма останется в памяти, что не говорит о высоком профессионализме программиста...
В приведенном ниже коде Form1 указывает только на последнюю созданную форму. Если вам это не нравится, воспользуйтесь приведенным ниже кодом — возможно, он более точно отвечает вашим запросам:

```
with TFormI.Create(Application) do
Caption:= 'New Form';
```

При разработке MDI-приложения метод Show не нужен, так как Delphi автоматически показывает все вновь созданные дочерние MDI-формы. В случае SDI-приложения вы обязаны использовать метод Show.

Даже при динамическом создании форм Delphi попытается навязать вам свои услуги по созданию экземпляра каждой формы. Чтобы отказаться от них, воспользуйтесь диалоговым окном Project Options, изображенным на рис. 1.14, и удалите классы форм из списка Auto-create forms.

Если вы захотите получить доступ к отдельному дочернему экземпляру класса, используйте свойство MDIChildren, описываемое в следующем разделе.
MDI-свойства TForm
Объект TForm имеет несколько свойств, специфичных для MDI-приложений.
ActiveMDIChild
Это свойство возвращает дочерний объект TForm, имеющий в текущее время фокус ввода. Оно полезно, когда родительская форма содержит панель инструментов или меню, команды которых распространяются на открытую дочернюю форму.
Например, представим, что проект использует дочернюю форму, содержащую элемент TMemo, названный memDailyNotes. Имя класса этой дочерней формы— TfrmMDIChild. Родительская форма содержит кнопку Clear в панели инструментов, которая удаляет содержимое memDailyNotes в активной дочерней форме. Вот как это реализуется.

```
procedure TfrmMDIParent.spbtnClearClick(Sender: TObject);
begin
  if not (ActiveMDIChild = Nil) then
    if ActiveMDIChild is TfrmMDIChild then
```

```
      TfrmMDIChild(ActiveMDIChild).memDailyNotes.Clear;
end;
```

В первой строке проверяется, равен ли ActiveMDIChild значению Nil, так как в этом случае обращение к объекту вызовет исключительную ситуацию.

Совет: ActiveMDIChild равен Nil, если нет открытых дочерних форм или свойство FormStyle не равно fsMDIForm.

Поскольку ActiveMDIChild возвращает объект TForm, компилятор не имеет доступа к memDailyNotes — объекту TfrmMDIChild. Вторая строка проверят соответствие типов, т.е. действительно ли ActiveMDIChild указывает на объект TfrmMDIChild.
Третья строка выполняет преобразование типа и вызывает метод Clear компонента memDailyNotes.

MDIChildren и MDIChildCount

Свойство MDIChildren является массивом объектов TForm, предоставляющих доступ к созданным дочерним формам. MDIChildCount возвращает количество элементов в массиве MDIChildren.
Обычно это свойство используется при выполнении какого-либо действия над всеми открытыми дочерними формами. Вот код сворачивания всех дочерних форм командой Minimize All.

```
procedure TFormI.mnuMinimizeAllClick(Sender: TObject);
var
  iCount: Integers;
begin
  for iCount:= MDIChildCount-1 downto 0 do
    MDIChildren[iCount].WindowState:= wsMinimized;
end;
```

Если вы будете сворачивать окна в порядке возрастания элементов массива, цикл будет работать некорректно, так как после сворачивания каждого окна массив MDIChildren обновляется и пересортировывается, и вы можете пропустить некоторые элементы.

TileMode

Это — свойство перечислимого типа, определяющее, как родительская форма размещает дочерние при вызове метода Tile. Используются значения tbHorizontal (по умолчанию) и tbVertical для размещения форм по горизонтали и вертикали.

WindowMenu

Профессиональные MDI-приложения позволяют активизировать необходимое дочернее окно, выбрав его из списка в меню. Свойство WindowMenu определяет объект TMenuItem, который Delphi будет использовать для вывода списка доступных дочерних форм.
Для вывода списка TMenuItem должно быть меню верхнего уровня. Это меню имеет свойство Caption, равное swindow.

MDI-события TForm

В MDI-приложении событие OnActivate запускается только при переключении между дочерними формами. Если фокус ввода передается из не MDI-формы в MDI-форму, генерируется событие OnActivate родительской формы, хотя ее свойство Active никогда и не устанавливается равным True. Эта странность на самом деле строго логична: ведь, если бы OnActivate генерировался только для дочерних форм, не было бы никакой возможности узнать о переходе фокуса ввода от другого приложения.

MDI-методы TForm
Специфичные для MDI-форм методы перечислены ниже.
Arrangelcons выстраивает пиктограммы минимизированных дочерних форм в нижней части родительской формы.
Cascade располагает дочерние формы каскадом, так что видны все их заголовки.
Next и Previous переходит от одной дочерней формы к другой, как будто вы нажали <Ctrl+Tab> или <Ctrl+Shift+Tab>.
Tile выстраивает дочерние формы так, что они не перекрываются.

Пример MDI-приложения
В этом разделе мы расширим возможности созданной ранее программы просмотра изображений.
Создание интерфейса
Интерфейс MDI-приложения очень похож на интерфейс разработанного ранее SDI-приложения, но каждое изображение выводится в отдельной, а не в главной форме.
Выполните следующие действия для создания родительской формы.
1. Выберите команду File/New Application, и появится пустое приложение.
2. Установите следующие свойства.
Свойство Значение
Caption Image Viewer
FormStyle fsMDIForm
Name frmMDIParent ShowHint True
3. Поместите компонент TPanel в форму. Установите следующие его свойства.
Свойство Значение
Align alTop
Caption -
4. Поместите три компонента TSpeedButton в TPanel и назовите их spbtnLoad, spbtnStretch и spbtnCenter. Установите следующие их свойства.
Свойство Значение
spbtnLoad.Hint Load
spbtnLoad.Left 8
spbtnLoad.Top 8
spbtnStretch.AllowAllUp True
spbtnStretch.Grouplndex 1
spbtnStretch.Hint Stretch
spbtnStretch.Left 48
spbtnStretch.Top 8
spbtnCenter.AllowAllUp True
spbtnCenter.Grouplndex 2
spbtnCenter.Hint Center
spbtnCenter.Left 80
spbtnCenter.Top 8
Свойства Glyph установите те же, что и для SDI-приложения.
5. Добавьте в форму компонент TOpenDialog и установите следующие его свойства.
Свойство Значение
Filter Bitmaps (*.bmp)]*.bmp
Name opndlgLoad
Options [ofPathMustExist,ofFileMustExist]
Теперь создадим дочернюю форму.
1. Выберите из меню File/New Form, и появится пустая форма.
2. Установите следующие ее свойства.
Свойство Значение

FormStyle fsMDIChild
Name frmMDIChild
Position poDefaultPosOnly
3. Поместите компонент TImage во вновь созданную форму и установите его следующие свойства.
Свойство Значение
Align alClient
Name imgMain
Удалите дочернюю форму из списка автоматически создаваемых форм следующим образом.

1. Выберите команду Project/ Options, и появится диалоговое окно Project Options, показанное на рис. 1.14.
2. Выберите frmMDIChild в списке Auto-create forms.
3. Щелкните на кнопке. Форма frmMDIChild при этом будет перенесена в список Available forms.
4. Щелкните на кнопке ОК.
Теперь самое время сохранить проект, выбрав команду File/Save Project As. Сохраните Unit1 как MDIParent, а проект — как EgMDIApp.
Написание кода
Создав интерфейс, перейдем к написанию исходного текста приложения, который будет очень похож на код для SDI-приложения.
Сначала загрузим изображение. Введите следующий код в обработчик события OnClick компонента spbtnLoad.

```
procedure TfrmMDIParent.spbtnLoadClick(Sender: TObject);
begin
  if opndlgLoad.Execute then
  with TfrmMDIChild.Create(Application) do
    begin
      Caption:= opndlgLoad.FileName;
      imgMain.Picture.LoadFromFile(opndlgLoad.FileName);
      ClientWidth:= imgMain.Picture.Width;
      ClientHeight:= imgMain.Picture.Height;
    end;
end;
```

После запуска диалогового окна создается новый экземпляр дочерней формы и загружается файл изображения. После загрузки размеры дочерней формы изменяются так, чтобы можно было видеть все изображение.
Еще пара штрихов— и приложение заработает, как и предусматривалось. Поскольку модуль ссылается на тип TfrmMDIChild, находящийся в модуле MDIChild, после строки implementation следует добавить еще одну строку:
uses MDIChild;
Теперь можно приступить к компиляции и запуску приложения. Однако заметьте, что, когда вы щелкаете на кнопке Close, дочерняя форма не закрывается, а сворачивается в пиктограмму. Чтобы заставить ее закрыться, следует добавить в код обработчика OnClose класса TfrmMDIChild маленькую деталь— изменить свойство Action:
Action:= caFree;
Компоненты TSpeedButton Stretch и Center выполняют те же функции, что и в SDI-приложении, однако их обработчики события OnClick следует изменить следующим образом

```
if not (ActiveMDIChild = Nil) then
  if ActiveMDIChild is TfrmMDIChild then
    TfrmMDIChild(ActiveMDIChild).imgMain.Stretch:=
spbthStretch.Down;
и
if not (ActiveMDIChild = Nil) then
  if ActiveMDIChild is TfrmMDIChild then
    TfrmMDIChild(ActiveMDIChild).imgMain.Center:=
spbthCenter.Down;
```

Остается последняя проблема — состояния кнопок Stretch и Center одинаковы для всех дочерних форм Для решения этой задачи добавьте в обработчик события OnActivate класса TfrmMDIChild строки.

```
frmMDIParent.spbtnStretch.Down:= imgMain.Stretch;
frmMDIParent.spbtnCenter.Down:= imgMain.Center;
```

И, наконец, самый последний из последних штрихов— в модуле MDIChild добавьте после строки implementation строку.

```
uses MDIParent;
```

ПРЕДОСТЕРЕЖЕНИЕ: В этом примере присвоение нового значения свойству Down класса TSpeedButton вызывало событие OnClick. Будьте осторожны при написании кода обработчика события, который генерирует новое событие путем присвоения значения свойству, ведь при этом можно создать бесконечную рекурсию.

Задание:

Написать программу, реализующий многооконное приложение для редактирования текста в формате RTF, а также для редактирования изображений в формате BMP. Предусмотреть сохранение состояния программы после выхода из программы и восстановления при запуске.

Лабораторная работа 12.
Создание клиент серверных приложений.

Надо заметить, что в отличие от любых других протоколов (FTP, POP, SMTP, HTTP, и т.д.), сокеты - это база для этих протоколов. Таким образом, пользуясь сокетами, можно самому создать (симитировать) и FTP, и POP, и любой другой протокол, причем не обязательно уже созданный, а даже свой собственный!

Алгоритм работы с сокетными протоколами

При работе с сокетом Вы просто посылаете другому компьютеру последовательность символов. Так что этим методом Вы можете посылать как простые сообщения, так и целые файлы! Причем, контролировать правильность передачи Вам не нужно (как это было при работе с COM-портами)!
Ниже следует примерная схема работы с сокетами в Дельфи-приложениях:

Определение св-в Host и Port >>> Запуск Сокета (ClientSocket1.Open) >>> Авторизация >>> Посылка/прием данных >>> Закрытие Сокета

Разберем схему подробнее:
Определение св-в Host и Port - чтобы успешно установить соединение, нужно присвоить свойствам Host и Port компонента TClientSocket требуемые значения. Host - это хост-имя (например: nitro.borland.com) либо IP-адрес (например: 192.168.0.88) компьютера, с которым надо соединиться. Port - номер порта (от 1 до 65535) для установления соединения. Обычно номера портов берутся, начиная с 1001 - т.к. номера меньше 1000 могут быть заняты системными службами (например, POP - 110). Подробнее о практической части см.ниже;
Открытие сокета - после того, как Вы назначили свойствам Host и Port соответствующие значения, можно приступить непосредственно к открытию сокета (сокет здесь рассматривается как очередь, в которой содержатся символы, передающиеся от одного компьютера к другому). Для этого можно вызвать метод Open компонента TClientSocket, либо присвоить свойству Active значение True. Здесь полезно ставить обработчик исключительной ситуации на тот случай, если соединиться не удалось. Подробнее об этом можно прочитать ниже, в практической части;
Авторизация - этот пункт можно пропустить, если сервер не требует ввода каких-либо логинов и/или паролей. На этом этапе Вы посылаете серверу свой логин (имя пользователя) и пароль. Но механизм авторизации зависит уже от конкретного сервера;
Посылка/прием данных - это, собственно и есть то, для чего открывалось сокетное соединение. Протокол обмена данными также зависит от сервера;
Закрытие сокета - после всех выполненных операций необходимо закрыть сокет с помощью метода Close компонента TClientSocket (либо присвоить свойству Active значение False).

Описание свойств и методов компонента TClientSocket

Здесь мы познакомимся с основными свойствами, методами и событиями компонента TClientSocket.

Свойства
Active - показывает, открыт сокет или нет. Тип: Boolean. Соответственно, True - открыт, а False - закрыт. Это свойство доступно для записи;
Host - строка (Тип: string), указывающая на хост-имя компьютера, к которому следует подключиться;
Address - строка (Тип: string), указывающая на IP-адрес компьютера, к которому следует подключиться. В отличие от Host, здесь может содержаться лишь IP. Отличие в том, что

если Вы укажете в Host символьное имя компьютера, то IP адрес, соответствующий этому имени будет запрошен у DNS;
Port - номер порта (Тип: Integer (Word)), к которому следует подключиться. Допустимые значения - от 1 до 65535;
Service - строка (Тип: string), определяющая службу (ftp, http, pop, и т.д.), к порту которой произойдет подключение. Это своеобразный справочник соответствия номеров портов различным стандартным протоколам;
ClientType - тип соединения. ctNonBlocking - асинхронная передача данных, т.е. посылать и принимать данные по сокету можно одновременно с помощью OnRead и OnWrite. ctBlocking - синхронная передача данных. События OnRead и OnWrite не работают. Этот тип соединения полезен для организации обмена данными с помощью потоков (т.е. работа с сокетом как с файлом);

Методы
Open - открытие сокета (аналогично присвоению значения True свойству Active);
Close - закрытие сокета (аналогично присвоению значения False свойству Active);

События

OnConnect - как следует из названия, это событие возникает при установлении соединения. Т.е. в обработчике этого события уже можно начинать авторизацию или прием/передачу данных;
OnConnecting - возникает при установлении соединения. Отличие от **OnConnect** в том, что соединение еще не установлено. Обычно такие промежуточные события используются для обновления статуса;
OnDisconnect - возникает при закрытии сокета. Причем, закрытия, как из Вашей программы, так и со стороны удаленного компьютера (либо из-за сбоя);
OnError - возникает при ошибке в работе сокета. Следует отметить, что это событие не поможет Вам отловить ошибку в момент открытия сокета (**Open**). Для того, чтобы избежать выдачи виндозного сообщения об ошибке, надо заключить операторы открытия сокета в блок **try..except** (обработка исключительных ситуаций);
OnLookup - возникает при попытке получения от DNS IP-адреса указанного хоста;
OnRead - возникает, когда удаленный компьютер послал Вам какие-либо данные. При возникновении этого события возможна обработка данных;
OnWrite - возникает, когда Вам разрешена запись данных в сокет.

Практика и примеры

Пример 1. Простейшая сокетная программа

```
{... Здесь идет заголовок файла и определение формы
TForm1 и ее экземпляра Form1}

{В форму нужно поместить кнопку TButton и два TEdit.
При нажатии на кнопку вызывается обработчик события
OnClick - Button1Click. Перед этим в первый из TEdit-ов
нужно ввести хост-имя, а во второй - порт удаленного
компьютера.
НЕ ЗАБУДЬТЕ ПОМЕСТИТЬ В ФОРМУ КОМПОНЕНТ TClientSocket!}
procedure Button1Click(Sender: TObject);
```

```
begin
{Присваиваем свойствам Host и Port нужные значения}
  ClientSocket1.Host := Edit1.Text;
  ClientSocket1.Port := StrToInt(Edit2.Text);
{Пытаемся открыть сокет и установить соединение}
  ClientSocket1.Open;
end;

procedure ClientSocket1Connect(Sender: TObject; Socket:
TCustomWinSocket);
begin
{Как только произошло соединение - закрываем сокет и
прерываем связь}
  ClientSocket1.Close;
end;
```

Если Вы думаете, что данный пример программы совершенно бесполезен и не может принести никакой пользы, то глубоко ошибаетесь. Приведенный код - простейший пример сканера портов (PortScanner). Суть такой утилиты в том, чтобы проверять, включен ли указанный порт и готов ли он к приему/передаче данных.
Далее следует другой пример, в котором по сокету передаются и принимаются текстовые сообщения:

Пример 2. Посылка/прием текстовых сообщений по сокетам

```
{... Здесь идет заголовок файла и определение формы TForm1 и ее экземпляра
Form1}

    {В форму нужно поместить две кнопки TButton и три TEdit. При нажатии
    на первую кнопку вызывается обработчик события OnClick - Button1Click.
    Перед этим в первый из TEdit-ов нужно ввести хост-имя, а во второй - порт
    удаленного компьютера. После установления соединения можно посылать
текстовые
    сообщения, вводя текст в третий TEdit и нажимая вторую кнопку TButton.
    Чтобы отсоединиться, нужно еще раз нажать первую TButton. Еще нужно
    добавить TListBox, в который будем помещать принятые и
    отправленные сообщения.
    НЕ ЗАБУДЬТЕ ПОМЕСТИТЬ В ФОРМУ КОМПОНЕНТ TClientSocket!}
    procedure Button1Click(Sender: TObject);
    begin
      {Если соединение уже установлено - прерываем его.}
      if ClientSocket1.Active then begin
        ClientSocket1.Close;
        Exit; {...и выходим из обработчика}
      end;
      {Присваиваем свойствам Host и Port нужные значения}
      ClientSocket1.Host := Edit1.Text;
      ClientSocket1.Port := StrToInt(Edit2.Text);
      {Пытаемся открыть сокет и установить соединение}
      ClientSocket1.Open;
    end;

    procedure ClientSocket1Connect(Sender: TObject; Socket:
TCustomWinSocket);
```

```
begin
  {Как только произошло соединение - посылаем
приветствие}
  Socket.SendText('Hello!');
  ListBox1.Items.Add('< Hello!');
end;

procedure ClientSocket1Read(Sender: TObject; Socket: TCustomWinSocket);
begin
  {Если пришло сообщение - добавляем его в ListBox}
  ListBox1.Items.Add('> '+Socket.ReceiveText);
end;

procedure Button2Click(Sender: TObject);
begin
  {Нажата кнопка - посылаем текст из третьего TEdit}
  ClientSocket1.Socket.SendText(Edit3.Text);
  ListBox1.Items.Add('< '+Edit3.Text);
end;
```

ПРИМЕЧАНИЕ: В некоторых случаях (зависящих от сервера) нужно после каждого сообщения посылать перевод строки:
ClientSocket1.Socket.SendText(Edit3.Text+#10);

Работа с сокетным потоком

"А как еще можно работать с сокетом?", - спросите Вы. Естественно, приведенный выше метод - не самое лучшее решение. Самих методов организации работы с сокетами очень много. Я приведу лишь еще один дополнительный - работа через поток. Наверняка, многие из Вас уже имеют опыт работы, если не с потоками (stream), то с файлами - точно. Для тех, кто не знает, поток - это канал для обмена данными, работа с которым аналогична работе с обычным файлом. Нижеприведенный пример показывает, как организовать поток для работы с сокетом:

Пример 3. Поток для работы с сокетом

```
procedure ClientSocket1Connect(Sender: TObject; Socket: TCustomWinSocket);
     var c: Char;
            MySocket: TWinSocketStream;
    begin
      {Как только произошло соединение - создаем
    поток и ассоциируем его с сокетом (60000 - таймаут в мсек)}
      MySocket := TWinSocketStream.Create(Socket,60000);
      {Оператор WaitForData ждет данных из потока
     указанное время в мсек (в данном примере - 100) и возвращает True,
     если получен хотя бы один байт данных, False - если нет никаких
     данных из потока.}
       while not MySocket.WaitForData(100) do
          Application.ProcessMessages;
       {Application.ProcessMessages позволяет
     Windows перерисовать нужные элементы окна и дает время другим
программам. Если бы этого оператора не было и данные бы довольно долго не
поступали, то система бы слегка "подвисла".}
       MySocket.Read(c,1);
       {Оператор Read читает указанное количество байт из потока (в данном
примере - 1) в указанную переменную определенного типа (в примере - в
переменную с типа Char). Обратите внимание на то, что Read, в отличие от
ReadBuffer, не устанавливает строгих ограничений на количество принятой
информации. Т.е. Read читает не больше n байтов из потока(где n - указанное
число). Эта функция возвращает количество полученных байтов данных.}
       MySocket.Write(c,1);
```

```
    {Оператор Write аналогичен оператору
   Read, за тем лишь исключением, что Write пишет данные в поток.}
     MySocket.Free;
     {Не забудем освободить память, выделенную под
   поток}
  end;
```

ПРИМЕЧАНИЕ: Для использования потока не забудьте установить свойство ClientType в ctBlocking.

Посылка/прием сложных данных

Иногда необходимо пересылать по сети не только простые текстовые сообщения, но и сложные структуры (тип record в Паскале), или даже файлы. И тогда Вам необходимо использовать специальные операторы. Некоторые из них перечислены ниже:

Методы TClientSocket.Socket (TCustomWinSocket, TClientWinSocket):

SendBuf(var Buf; Count: Integer) - Посылка буфера через сокет. Буфером может являться любой тип, будь то структура (record), либо простой Integer. Буфер указывается параметром Buf, вторым параметром Вы должны указать размер пересылаемых данных в байтах (Count);

SendText(const S: string) - Посылка текстовой строки через сокет. Этот метод рассматривался в примере 2 (см.выше);

SendStream(AStream: TStream) - Посылка содержимого указанного потока через сокет. Пересылаемый поток должен быть открыт. Поток может быть любого типа - файловый, из ОЗУ, и т.д. Описание работы непосредственно с потоками выходит за рамки данной статьи;

Всем перечисленным методам соответствуют методы Receive... Их описание можно посмотреть в справочном файле по Дельфи (VCL help).

Авторизация на сервере

В данном примере пароль посылается нешифрованным текстом, так что если Вам нужен действительно надежный механизм входа, то Вам придется внести кое-какие изменения в исходник данного примера. Пример реализован как работа с сокетным потоком.

Пример 4. Авторизация

```
{В данном примере нужно добавить в форму еще два TEdit - Edit3 и
Edit4 для ввода логина и пароля}
procedure ClientSocket1Connect(Sender: TObject; Socket:
TCustomWinSocket);
var c: Char;
  MySocket: TWinSocketStream;
  login,password: string;
begin
  MySocket := TWinSocketStream.Create(Socket,60000);
{Добавляем к логину и паролю символ перевода строки, чтобы
сервер смог отделить логин и пароль.}
  login := Edit3.Text+#10;
  password := Edit4.Text+#10;
  MySocket.Write(login,Length(Edit3.Text)+1);
  MySocket.Write(password,Length(Edit4.Text)+1);
  while not MySocket.WaitForData(100) do
    Application.ProcessMessages;
  MySocket.Read(c,1);
{Здесь сервер посылает нам один байт, значение 1 которого
```

```
соответствует подтверждению успешной авторизации, а 0 - ошибку
(это лишь пример). Далее мы выполняем нужные действия
(прием/пересылку данных) и закрываем поток.}
  MySocket.Free;
end;
```

Алгоритм работы сокетного сервера

Что же позволяет делать сокетный сервер?.. По какому принципу он работает?.. Сервер, основанный на сокетном протоколе, позволяет обслуживать сразу множество клиентов. Причем, ограничение на их количество Вы можете указать сами (или вообще убрать это ограничение, как это сделано по умолчанию). Для каждого подключенного клиента сервер открывает отдельный сокет, по которому Вы можете обмениваться данными с клиентом. Также отличным решением является создание для каждого подключения отдельного процесса (Thread).
Ниже следует примерная схема работы сокетного сервера в Дельфи-приложениях:

Определение св-в Port и ServerType >>> Открытие сокета >>>Подключение клиента и обмен данными с ним<<>> Отключение клиента >>>Закрытие сервера и сокета

Разберем схему подробнее:

- ***Определение св-в Port и ServerType*** - чтобы к серверу могли нормально подключаться клиенты, нужно, чтобы порт, используемый сервером точно совпадал с портом, используемым клиентом (и наоборот). Свойство ServerType определяет тип подключения (подробнее см.ниже);
- ***Открытие сокета*** - открытие сокета и указанного порта. Здесь выполняется автоматическое начало ожидания подсоединения клиентов (*Listen*);
- ***Подключение клиента и обмен данными с ним*** - здесь подключается клиент и идет обмен данными с ним.
- ***Отключение клиента*** - Здесь клиент отключается и закрывается его сокетное соединение с сервером;
- ***Закрытие сервера и сокета*** - По команде администратора сервер завершает свою работу, закрывая все открытые сокетные каналы и прекращая ожидание подключений клиентов.

Следует заметить, что пункты 3-4 повторяются многократно, т.е. эти пункты выполняются для каждого нового подключения клиента.
Примечание: Документации по сокетам в Дельфи на данный момент очень мало, так что, если Вы хотите максимально глубоко изучить эту тему, то советую просмотреть литературу и электронную документацию по Unix/Linux-системам - там **очень** хорошо описана теория работы с сокетами. Кроме того, для этих ОС есть множество примеров сокетных приложений (правда, в основном на C/C++ и Perl).

Краткое описание компонента TServerSocket

Здесь мы познакомимся с **основными** свойствами, методами и событиями компонента *TServerSocket*.

Свойства:
Socket - класс TServerWinSocket, через который Вы имеете доступ к открытым сокетным каналам. Далее мы рассмотрим это свойство более подробно, т.к. оно, собственно и есть одно из главных. *Тип:* ***TServerWinSocket***;
ServerType - тип сервера. Может принимать одно из двух значений: *stNonBlocking* - синхронная работа с клиентскими сокетами. При таком типе сервера Вы можете работать с клиентами через события *OnClientRead* и *OnClientWrite*. *stThreadBlocking* - асинхронный тип. Для каждого клиентского сокетного канала создается отдельный процесс (Thread).

Tun: ***TServerType***;
ThreadCacheSize - количество клиентских процессов (Thread), которые будут кэшироваться сервером. Здесь необходимо подбирать среднее значение в зависимости от загруженности Вашего сервера. Кэширование происходит для того, чтобы не создавать каждый раз отдельный процесс и не убивать закрытый сокет, а оставить их для дальнейшего использования. *Tun:* ***Integer***;
Active - показатель того, активен в данных момент сервер, или нет. Т.е., фактически, значение *True* указывает на то, что сервер работает и готов к приему клиентов, а *False* - сервер выключен. Чтобы запустить сервер, нужно просто присвоить этому свойству значение *True*. *Tun:* ***Boolean***;
Port - номер порта для установления соединений с клиентами. Порт у сервера и у клиентов должны быть одинаковыми. Рекомендуются значения от 1025 до 65535, т.к. от 1 до 1024 - могут быть заняты системой. *Tun:* ***Integer***;
Service - строка, определяющая службу (**ftp**, **http**, **pop**, и т.д.), порт которой будет использован. Это своеобразный справочник соответствия номеров портов различным стандартным протоколам. *Tun:* ***string***;

Методы:
Open - Запускает сервер. По сути, эта команда идентична присвоению значения *True* свойству *Active*;
Close - Останавливает сервер. По сути, эта команда идентична присвоению значения *False* свойству *Active*.

Свойства:
OnClientConnect - возникает, когда клиент установил сокетное соединение и ждет ответа сервера (*OnAccept*);
OnClientDisconnect - возникает, когда клиент отсоединился от сокетного канала;
OnClientError - возникает, когда текущая операция завершилась неудачно, т.е. произошла ошибка;
OnClientRead - возникает, когда клиент передал серверу какие-либо данные. Доступ к этим данным можно получить через передаваемый параметр *Socket: TCustomWinSocket*;
OnClientWrite - возникает, когда сервер может отправлять данные клиенту по сокету;
OnGetSocket - в обработчике этого события Вы можете отредактировать параметр *ClientSocket*;
OnGetThread - в обработчике этого события Вы можете определить уникальный процесс (Thread) для каждого отдельного клиентского канала, присвоив параметру *SocketThread* нужную подзадачу TServerClientThread;
OnThreadStart, ***OnThreadEnd*** - возникает, когда подзадача (процесс, Thread) запускается или останавливается, соответственно;
OnAccept - возникает, когда сервер принимает клиента или отказывает ему в соединении;
OnListen - возникает, когда сервер переходит в режим ожидания подсоединения клиентов.

TServerSocket.Socket (TServerWinSocket)

Итак, как же сервер может отсылать данные клиенту? А принимать данные? В основном, если Вы работаете через события *OnClientRead* и *OnClientWrite*, то общаться с клиентом можно через параметр ClientSocket (TCustomWinSocket). Про работу с этим классом можно прочитать выше, т.к. отправка/посылка данных через этот класс аналогична - методы (Send/Receive)(Text,Buffer,Stream). Также и при работе с TServerSocket.Socket. Однако, т.к. здесь мы рассматриваем сервер, то следует выделить некоторые полезные свойства и методы:

- **ActiveConnections** (*Integer*) - количество подключенных клиентов;

- **ActiveThreads** (*Integer*) - количество работающих процессов;
- **Connections** (*array*) - массив, состоящий из отдельных классов TClientWinSocket для каждого подключенного клиента. Например, такая команда:
  ServerSocket1.Socket.Connections[0].SendText('Hello!');
  отсылает первому подключенному клиенту сообщение 'Hello!'. Команды для работы с элементами этого массива - также (Send/Receive)(Text,Buffer, Stream);
- **IdleThreads** (*Integer*) - количество свободных процессов. Такие процессы кэшируются сервером (см. *ThreadCacheSize*);
- **LocalAddress**, **LocalHost**, **LocalPort** - соответственно - локальный IP-адрес, хост-имя, порт;
- **RemoteAddress**, **RemoteHost**, **RemotePort** - соответственно - удаленный IP-адрес, хост-имя, порт;
- Методы **Lock** и **UnLock** - соответственно, блокировка и разблокировка сокета.

Практика и примеры

А теперь рассмотрим вышеприведенное на конкретном примере.
Итак, рассмотрим очень неплохой пример работы с TServerSocket (этот пример - наиболее наглядное пособие для изучения этого компонента). В приведенных ниже исходниках демонстрируется протоколирование всех важных событий сервера, плюс возможность принимать и отсылать текстовые сообщения:

Пример 1. Протоколирование и изучение работы сервера, посылка/прием сообщений через сокеты.

```
{... Здесь идет заголовок файла и определение
        формы TForm1 и ее экземпляра Form1}
procedure TForm1.Button1Click(Sender: TObject);
begin
  {Определяем порт и запускаем сервер}
  ServerSocket1.Port := 1025;
  {Метод Insert вставляет строку в массив в
                                  указанную позицию}
  Memo2.Lines.Insert(0,'Server starting');
  ServerSocket1.Open;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {Останавливаем сервер}
  ServerSocket1.Active := False;
  Memo2.Lines.Insert(0,'Server stopped');
end;

procedure TForm1.ServerSocket1Listen(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  {Здесь сервер "прослушивает"
               сокет на наличие клиентов}
  Memo2.Lines.Insert(0,'Listening on port '+
                       IntToStr(ServerSocket1.Port));
end;

procedure TForm1.ServerSocket1Accept(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  {Здесь сервер принимает клиента}
  Memo2.Lines.Insert(0,'Client connection accepted');
```

```
end;

procedure TForm1.ServerSocket1ClientConnect(Sender: TObject;
     Socket: TCustomWinSocket);
begin
  {Здесь клиент подсоединяется}
  Memo2.Lines.Insert(0,'Client connected');
end;

procedure TForm1.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  {Здесь клиент отсоединяется}
  Memo2.Lines.Insert(0,'Client disconnected');
end;

procedure TForm1.ServerSocket1ClientError(Sender: TObject;
  Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
  var ErrorCode: Integer);
begin
  {Произошла ошибка - выводим ее код}
  Memo2.Lines.Insert(0,'Client error. Code = '+IntToStr(ErrorCode));
end;

procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
{От клиента получено сообщение -
                        выводим его в Memo1}
  Memo2.Lines.Insert(0,'Message received from client');
  Memo1.Lines.Insert(0,'> '+Socket.ReceiveText);
end;

procedure TForm1.ServerSocket1ClientWrite(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  {Теперь можно слать данные в сокет}
  Memo2.Lines.Insert(0,'Now can write to socket');
end;

procedure TForm1.ServerSocket1GetSocket(Sender: TObject;
   Socket: Integer; var ClientSocket: TServerClientWinSocket);
begin
  Memo2.Lines.Insert(0,'Get socket');
end;

procedure TForm1.ServerSocket1GetThread(Sender: TObject;
  ClientSocket: TServerClientWinSocket;
  var SocketThread: TServerClientThread);
begin
  Memo2.Lines.Insert(0,'Get Thread');
end;

procedure TForm1.ServerSocket1ThreadEnd(Sender: TObject;
   Thread: TServerClientThread);
begin
  Memo2.Lines.Insert(0,'Thread end');
end;

procedure TForm1.ServerSocket1ThreadStart(Sender: TObject;
  Thread: TServerClientThread);
begin
  Memo2.Lines.Insert(0,'Thread start');
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
 var i: Integer;
begin
  {Посылаем ВСЕМ клиентам сообщение из Edit1}
  for i := 0 to ServerSocket1.Socket.ActiveConnections-1 do begin
  ServerSocket1.Socket.Connections[i].SendText(Edit1.Text);
end;
  Memo1.Lines.Insert(0,'< '+Edit1.Text);
end;
```

Далее мы будем рассматривать уже не примеры, а приемы работы с TServerSocket.

Приемы работы с TServerSocket (и просто с сокетами)

Хранение уникальных данных для каждого клиента.

Наверняка, если Ваш сервер будет обслуживать множество клиентов, то Вам потребуется хранить какую-либо информацию для каждого клиента (имя, и др.), причем с привязкой этой информации к сокету данного клиента. В некоторых случаях делать все это вручную (привязка к handle сокета, массивы клиентов, и т.д.) не очень удобно. Поэтому для каждого сокета существует специальное свойство - **Data**. На самом деле, Data - это всего-навсего указатель. Поэтому, записывая данные клиента в это свойство будьте внимательны и следуйте правилам работы с указателями (выделение памяти, определение типа, и т.д.)!

Посылка файлов через сокет.

Здесь мы рассмотрим посылку файлов через сокет (по просьбе JINX-а) :-). Итак, как же послать файл по сокету? Очень просто! Достаточно лишь открыть этот файл как файловый поток (TFileStream) и отправить его через сокет (SendStream)! Рассмотрим это на примере:

```
  {Посылка файла через сокет}
 procedure SendFileBySocket(filename: string);
  var srcfile: TFileStream;
 begin
   {Открываем файл filename}
   srcfile := TFileStream.Create(filename,fmOpenRead);
   {Посылаем его первому подключенному клиенту}
   ServerSocket1.Socket.Connections[0].SendStream(srcfile);
   {Закрываем файл}
   srcfile.Free;
 end;
```

Нужно заметить, что метод **SendStream** используется не только сервером, но и клиентом (**ClientSocket1.Socket.SendStream(srcfile)**)

Почему несколько блоков при передаче могут обьединяться в один

Итак, во-первых, надо заметить, что посылаемые через сокет данные могут не только объединяться в один блок, но и разъединяться по нескольким блокам. Дело в том, что сокет - обычный поток, но в отличие, скажем, от файлового (TFileStream), он передает данные медленнее (сами понимаете - сеть, ограниченный трафик, и т.д.). Именно поэтому две команды:
ServerSocket1.Socket.Connections[0].SendText('Hello, ');
ServerSocket1.Socket.Connections[0].SendText('world!');
совершенно идентичны одной команде:
ServerSocket1.Socket.Connections[0].SendText('Hello, world!');
И именно поэтому, если Вы отправите через сокет файл, скажем, в 100 Кб, то тому, кому Вы посылали этот блок, придет несколько блоков с размерами, которые зависят от трафика и загруженности линии. Причем, размеры не обязательно будут одинаковыми.

Отсюда следует, что для того, чтобы принять файл или любые другие данные большого размера, Вам следует принимать блоки данных, а затем объединять их в одно целое (и сохранять, например, в файл). Отличным решением данной задачи является тот же файловый поток - TFileStream (либо поток в памяти - TMemoryStream). Принимать частички данных из сокета можно через событие OnRead (OnClientRead), используя универсальный метод **ReceiveBuf**. Определить размер полученного блока можно методом **ReceiveLength**.

```
{Прием файла через сокет}
procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
 var l: Integer;
     buf: PChar;
     src: TFileStream;
begin
  {Записываем в l размер полученного блока}
  l := Socket.ReceiveLength;
  {Заказываем память для буфера}
  GetMem(buf,l+1);
  {Записываем в буфер полученный блок}
  Socket.ReceiveBuf(buf,l);
  {Открываем временный файл для записи}
  src := TFileStream.Create('myfile.tmp',fmOpenReadWrite);
  {Ставим позицию в конец файла}
  src.Seek(0,soFromEnd);
  {Записываем буфер в файл}
  src.WriteBuffer(buf,l);
  {Закрываем файл}
  src.Free;
  {Освобождаем память}
  FreeMem(buf);
 end;
```

Задания:

1. Создать программу, для поиска и загрузки файлов на удаленном компьютере.
2. Создать программу, имитирующую работу службы net send.
3. Создать программу многопользовательский чат.
4. Создать программу Remote Administrator.

Лабораторная работа 13.
Операторы языка Паскаль. Элементы управления редактированием.

Задание № 1

Сконструировать форму с шестью кнопками, имена которых ShortInt, SmallInt, Integer, Byte, Word, Cardinal; с четырьмя статическими надписями (компонент Label) Type, Size, Max, Min и четырьмя надписями для вывода информации о типе при каждом нажатии одной из кнопок. Для этого записать для каждой кнопки метод отклика на событие OnClick, используя свойство Caption надписей для вывода информации и функции SizeOf – размер внутреннего представления переменной данного типа, High – самое высокое значение в диапазоне перечислимого типа, Low – самое низкое значение, а также функцию IntToStr – преобразование числа в строку. Пример строки кода:

SizeLabel . Caption : = IntToStr (SizeOf (Number)) ;

Приведение и преобразование типов

Вы не можете присвоить переменной значение другого типа. Если в этом все же возникла необходимость, имеются две возможности. Первая возможность – приведение типов, которое выглядит как простой вызов функции, но вместо имени функции используется имя типа данных адресата:

```
var
  N:   Integer;
  C:   Char;
  B:   Boolean;
begin
  N  : =  Integer ( 'X' ) ;
   C  : =  Char ( N ) ;
   B  : =  Boolean ( 0 ) ;
```

Строго говоря, операцию приведения можно осуществлять между теми типами данных, которые имеют одинаковый размер. Обычно безопасным является приведение между перечислимыми или между вещественными типами, но вы также можете выполнить приведение между типами указателей (а также объектов).

Вторая возможность – использовать подпрограмму преобразования типов, например – Trunc – преобразует значение вещественного типа в значение целочисленного типа, отсекая дробную часть; IntToStr – преобразует число в строку; StrToInt – преобразует строку в число, вызывая исключение в случае неправильной строки и т.д.

Операторы языка Паскаль

Рассмотрим пример, который демонстрирует различие между фиксированным счетчиком и циклом с псевдослучайным счетчиком. Начните новый пустой проект и поместите в его основную форму список и две

кнопки. Теперь в событие OnClick кнопок можно добавить некоторый код. Первая кнопка содержит простой цикл for для отображения списка чисел. До выполнения этого цикла, который добавляет несколько строк в свойство Items списка, вы должны очистить содержимое самого списка.

```
procedure TForm1. Botton1. Click (Sender : TObject) ;
  var
    I :  Integer ;
begin
  ListBox1. Items. Clear ;
   For  I : = 1  to  20  do
      ListBox1. Items. Add ( ‘ String ‘ + IntToStr ( I ) ) ;
end;
```

Код, связанный со второй кнопкой использует цикл while, который основан на счетчике, увеличивающемся случайным образом.

```
procedure TForm1. Botton2. Click (Sender : TObject) ;
  var
     I :  Integer ;
begin
  ListBox1. Items. Clear ;
   Randomize;
    I : = 0 ;
     while  I < 1000  do
      begin
         I : = I + Random ( 100 );
          ListBox1. Items. Add ( ‘ Random number : ‘ + IntToStr ( I ) ) ;
       end;
end;
```

При каждом щелчке по второй кнопке числа будут различными , так как они зависят от генератора случайных чисел.

Элементы управления редактированием

Класс TCustomEdit – это абстрактный класс для всех элементов управления редактированием в Delphi. Он включает простой элемент управления редактированием, элементы управления редактированием по маске и все элементы управления мемо.

Некоторые свойства и методы, реализованные классом TCustomEdit

Используйте или установите это…	*Чтобы сделать это …*
Brush	Определить цвет и шаблон, используемые в качестве фона оконного элемента управления
CanFocus	Определить, может ли оконный элемент управления получить фокус

Clear	Очистить содержимое элемента управления редактированием
Enabled	Определить доступность элемента управления
Focused	Определить находится ли оконный элемент управления в фокусе
Font	Определить шрифт, используемый для вывода текста в элементе управления
GetSelTextBuf	Скопировать выбранный текст из элемента управления в буфер
GetTextBuf	Скопировать текст из элемента управления в буфер
GetTextLen	Получить длину текста элемента управления
Hide	Сделать элемент управления невидимым
Hint	Определить текст, который отображается в подсказке для элемента управления
SelectAll	Выбрать весь текст в элементе управления
SelLength	Определить длину выбранного текста в элементе управления
SelStart	Определить исходную позицию выбранного текста
SelText	Получить доступ к выбранному тексту в элементе управления редактирования
SetFocus	Установить фокус на оконный элемент управления
Show	Сделать элемент управления видимым
Text	Обратиться к изменяемому тексту на элементе управления

Класс TEdit инкапсулирует большинство возможностей стандартного элемента управления редактированием известного как “поле” или “ текстовое поле”. Элемент управления редактированием предоставляет одну доступную для редактирования строку текста внутри элемента управления с необязательной рамкой. При желании текст на элементе управления редактированием может быть предназначен только для чтения, так что пользователь изменять его не сможет.

Класс TEdit предусматривает только основные функциональные возможности элемента управления редактированием. При необходимости ограничить диапазон ввода, воспринимаемый этим элементом управления, используйте вместо него элемент управления редактированием по маске (TMaskEdit). Класс TEdit порожден непосредственно от TCustomEdit.

Задание № 2

Сконструировать форму, которая будет содержать следующие управляющие элементы:

- Элемент управления редактированием по маске (TMaskEdit) или элемент управления редактированием (TEdit) со связанной с ним меткой Operand 1.

- Элемент управления редактированием по маске (TMaskEdit) или элемент управления редактированием (TEdit) со связанной с ним меткой Operand 2.
- Элемент управления редактированием по маске (TMaskEdit) или элемент управления редактированием (TEdit) со связанной с ним меткой Result.
- Кнопка Close, которая закрывает приложение.
- Шесть кнопок операций + , - , * , / , mod , div .
- Кнопка Clear для очистки окон элементов управления редактированием.

Тип операндов – вещественный. Написать код, который будет реализовывать указанные операции по нажатию соответствующей кнопки (связать код с событием OnClick кнопки).

Задание № 3

Необходимо модифицировать калькулятор, добавив к нему, согласно варианту:

1. Возможность работы с памятью
2. Возможность ввода значений (цифр) с клавиатуры, а также изменение знака и ввода «,»
3. Возможность использования арифметических операций, стирания и вывода результата с клавиатуры
4. Запрещение ввода с клавиатуры букв
5. Возможность расчёта квадрата, корня введённого числа, а также «%», «1/x»
6. Возможность использования операций sin, cos, tg, ln, log и режима градусов и радиан
7. Возможность переключение режима между различными системами счисления
8. Возможность использования функций y^x, exp

Лабораторная работа 14.
Использование компонентов

Создание редактора для текста формата RTF

Windows 95 содержит новый управляющий элемент, который способен поддерживать формат Rich Text Format (RTF). Компонент Delphi Rich Edit инкапсулирует поведение этого стандартного управляющего элемента. Поместите в форму три компонента: Panel, которая занимает верхнюю часть формы; Button, расположенная на панели; RichEdit, который занимает всю остальную часть формы. Нажатие на кнопку должно изменять шрифт выделенного фрагмента текста в окне RichEdit. Программа отображает стандартное диалоговое окно Font, использовав в качестве начального значения исходный шрифт компонента RichEdit. Затем выбранный пользователем шрифт скопируется в атрибуты текущего выбранного текста. Хотя свойства DefAttributes и SelAttributes компонента RichEdit не имеют тип TFont, они совместимы с ним, поэтому для копирования значения можно применить метод Assign:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if RichEdit1.SelLength>0 then
  begin
    FontDialog1.Font.Assign(RichEdit1.DefAttributes);
     if FontDialog1.Execute then
       RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  end
  else
    ShowMessage ('No text selected');
end;
```

Задание № 1

Создать вышеописанную форму и записать код для события OnClick кнопки. В форму поместить компонент FontDialog.

Выбор

Существуют два стандартных управляющих элементаWindows, которые позволяют пользователю выбирать разные опции. Первый – чекбокс. Он соответствует опции, которую можно выбрать независимо от других. Второй управляющий элемент – кнопка опций, которая соответствует исключительному выбору. Например, если вы видите две кнопки опций с метками A и B, вы можете выбрать или A, или B, но не обе одновременно. Еще одна особенность множественного выбора заключается в том, что вы обязательно должны выбрать одну из опций.

Группирование кнопок опций

Кнопки опций подразумевают исключительный выбор. Однако форма может содержать несколько групп кнопок опций. Сама система Windows не

способна определить как связаны друг с другом различные кнопки опций. В Windows и в Delphi эта проблема решается так: связанные кнопки опций помещаются внутри компонента-контейнера. Для совместного хранения кнопок опций – и функционального и визуального – стандартный интерфейс пользователя Windows использует управляющий элемент блока группы. В Delphi этот управляющий элемент реализован в компоненте GroupBox. Аналогичный компонент, который используется для кнопок опций, - RadioGroup, представляющий собой блок группы с нарисованными внутри него несколькими копиями кнопок опций. Компонент RadioGroup способен автоматически выравнивать свои кнопки опций и вы легко можете добавить в него новые элементы на этапе выполнения.

Правила построения блока группы с кнопками опций очень просты: разместите в форме компонент GroupBox, а затем поместите в него кнопки опций. Компонент GroupBox способен содержать другие управляющие элементы и вместе с компонентом Panel является одним из самых распространенных компонентов-контейнеров. Если вы отключите или скроете блок группы, все управляющие элементы внутри него также будут заблокированы или скрыты.

Вы по-прежнему можете обрабатывать отдельные кнопки опций, но можете работать также со всем массивом управляющих элементов, принадлежащих блоку группы. Соответствующее свойство называется Controls. другое свойство ControlCount – хранит число элементов. К этим двум свойствам можно получить доступ только на этапе выполнения.

Задание № 2

Выполнить построитель английских предложений по типу The book is on the table. Цель данного прмера состоит в том, чтобы создать инструмент для построения подобных фраз путем выбора из различных доступных опций.

1) Поместите в форму компонент GroupBox с заголовком First Object и затем кнопки опций с заголовками The book, The pen, The pencil, The chair.
2) Поместите еще один компонент GroupBox с заголовком Position с опциями on, under, near.
3) Поместите компонент RadioGroup с заголовком Second Object с опциями the table, the big box, the carpet, the computer. В этом случае для создания элементов вы должны ввести список значений в свойство Items (тип TStringList).
4) Одну из кнопок опций в каждой группе нужно обязательно пометить, установив свойство Checked в True, соответствующее значение должно иметь и свойство ItemIndex группы опций (которое указывает на текущий выбор).
5) Поместите в верхнюю часть формы компонент Label, где будет отображаться построенная фраза.
6) Выберите в форме все кнопки опций (щелкая по каждой из них при нажатой клавише Shift) и введите имя метода TForm1.ChangeText в

окне Object Inspector после события OnClick. Код этого метода представлен ниже.

```
procedure TForm1.ChangeText (Sender: TObject);
var
  Phrase:String;
  I:Integer;
begin
  For i:=0 to GroupBox1.ControlCount - 1 do
    if (GroupBox1.Controls[ i ] as TRadioButton).Checked then
      Phrase:= (GroupBox1.Controls[ i ] as TRadioButton).Caption;
      Phrase:= Phrase + '  is  ';
   For i:=0 to GroupBox2.ControlCount-1 do
     with GroupBox2.Controls[ i ] as TRadioButton do
       if Checked then
         Phrase:= Phrase + Caption;
         Label1.Caption:= Phrase + ' ' +
RadioGroup1.Items[RadioGroup1.ItemIndex];
end;
```

Список со многими опциями

Когда вам нужно добавить много опций, кнопки опций не подходят. Для решения этой проблемы используется список. Список способен хранить в небольшом месте большое количество опций и может содержать полоску прокрутки, чтобы показывать на экране только ограниченную часть всего выбора. Другое преимущество списка состоит в том, что вы легко можете добавить в него новые элементы или удалить некоторые из текущих. Списки чрезвычайно гибки и мощны.

Еще одна важная особенность: используя компонент ListBox, вы можете осуществлять как однозначный выбор – поведение, аналогичное группе кнопок опций, - так и множественный выбор – подобно группе чекбоксов.

Задание № 3

Выполнить построитель английских предложений, используя компонент ListBox.

1) Поместите в центре формы компонент RadioGroup с заголовком Position и опциями on, under, near (одну опцию пометьте).
2) Слева и справа от блока группы поместите два списка и добавьте несколько строк в свойства Items обоих списков. Вы можете скопировать все строки из редактора свойств Items одного списка и вставить в редактор такого же свойства другого списка. Для удобства строки можно отсортировать, установив свойство Sorted списков в True. Не забудьте также поместить над списками пару надписей, раскрывающих их содержание.

3) В верхнею часть формы поместите компонент Label, в котором будет отображаться построенная фраза.
4) В нижней части формы поместите поле редактирования с надписью и кнопку с заголовком Add . В поле редектирования будет вводиться строка текста, которую необходимо добавить в оба списка по нажатию кнопки Add.
5) Чтобы первоначально выбрать по строке из каждого списка, записать метод для события OnCreate формы:

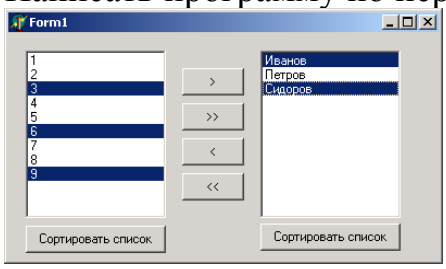
```
procedure TForm1.FormCreate (Sender: TObject);
var
  N : Integer ;
begin
   N := ListBox1. Items . IndexOf (‘book’) ;
   ListBox1. ItemIndex := N ;
   N := ListBox2 . Items . IndexOf (‘table’) ;
   ListBox2 . ItemIndex := N ;
end;
```

6) Создать процедуру ChangeText, соединенную с событиями OnClick группы опций и двух списков. Чтобы получить из списка выбранный текст, вы должны только взять номер выбранного элемента (хранится в свойстве этапа выполнения ItemIndex) и затем отыскать строку в соответствующей ячейке массива Item (ListBox1. Items [ListBox1. ItemIndex]).
7) Добавлять строку String1 в список с помощью метода Add класса TStrings –

ListBox1. Items . Add (String1) .

Задание № 4

Написать программу по переносу данных из одного списка в другой:



Программа должна переносить записи, которые выделены, а также все записи между двумя списками. Кнопка «Сортировать список» должна сортировать соответствующий список в порядке возрастания.

Лабораторная работа 15.
Создание и обработка меню.

Структура меню

Обычно меню имеет два уровня. Полоска меню, которая находится под заголовком окна, содержит имена выпадающих меню. Каждое выпадающее меню содержит несколько элементов. Однако структура меню очень гибка. Элемент меню можно поместить непосредственно в полоску, а выпадающее меню внутрь другого выпадающего меню. Выпадающее меню внутри другого выпадающего меню (выпадающее меню второго уровня) встречается довольно часто, и для этого случая Windows предоставляет специальный визуальный значок – маленький треугольник справа от меню. Нередко вместо выпадающего меню второго уровня вы можете просто сгруппировать ряд опций в первоначальном выпадающем меню и поместить два разделителя: один до группы и один после.

Различные роли элементов меню

Существует три основных типа элементов меню:

- Команды – элементы меню, которые используются для выдачи команды и выполнения действия. Визуально они никак не выделяются.
- Установщики состояния – элементы меню, которые используются для переключения опции в положения включено – выключено и изменения состояния какого – либо элемента. Если эти команды имеют два состояния, то в активном положении слева от них обычно стоит галочка. В этом случае выбор команды изменяет состояние на противоположное.
- Элементы вызова диалога – элементы меню, которые вызывают диалоговою панель. Реальное различие между этими и другими элементами меню состоит в следующем: с помощью этих элементов пользователь должен получить возможность исследовать вероятные действия соответствующей диалоговой панели. Такие команды должны иметь визуальный ключ в виде трех точек после текста.

Редактирование меню с помощью Menu Designer

Система Delphi включает специальный редактор для меню Menu Designer. Чтобы вызвать этот инструмент, поместите компонент меню в форму и щелкните по нему. Не волнуйтесь о точном положении данного компонента в форме, поскольку на результат это не влияет : само меню всегда помещается правильно – под заголовком формы. Menu Designer позволяет создавать меню путем простого написания текста команд, перемещать элементы или выпадающие меню с помощью буксировки и легко устанавливать свойства элементов. Для создания выпадающего меню второго уровня нужно выбрать команду Create submenu в SpeedMenu инструмента (локальном меню, которое вызывается правой кнопкой мыши).

Горячие клавиши меню

Общее свойство элементов меню – наличие подчеркнутой буквы. Эту букву можно использовать для выбора меню с помощью клавиатуры. При

одновременном нажатии клавиши Alt и клавиши с буквой на экране появляется соответствующее выпадающее меню. Чтобы определить подчеркнутую клавишу, просто поместите перед ней символ амперсанта (&), например &File. Элементам меню можно назначить горячие клавиши. Для этого нужно указать значение для свойства ShortCut, выбрав одну из стандартных комбинаций.

Задание № 1

1) С помощью Menu Designer добавьте в форму полоску меню. Эта полоска имеет три выпадающих меню: меню File с единственной командой Exit; меню Options с командами Font, Color, Left, Center, Hight (между Color и Left установите разделитель, для Left, Center, Hight назначьте горячие клавиши); меню Help с элементом About. Чтобы поместить разделитель, вместо текста команды вставьте просто дефис.
2) В форму поместите компонент RichEdit и две пиктограммы диалогов FontDialog и ColorDialog.
3) Для отклика на команды меню вы должны определить метод для события OnClick каждого элемента меню. Код метода TForm1.Font1Click :

```
procedure TForm1.Font1Click (Sender : TObject) ;
begin
  FontDialog1.Font : = RichEdit1.Font ;
  FontDialog1.Execute;
  RichEdit1.Font : = FontDialog1.Font ;
end;
```

4) Код метода TForm1.Color1Click аналогичен предыдущему.
5) Код метода TForm1.Left1Click :

```
procedure TForm1.Left1Click (Sender : TObject) ;
begin
  RichEdit1.Paragraph.Alignment : = taLeftJustify ;
  Left1.Checked : = True ;
  Center1.Checked : = False ;
  Right1Checked : = False ;
end;
```

Чтобы поставить галочку в ряде выбираемых опций, установите свойство Checked элемента меню в окне Object Inspector в True.

6) Код методов для элементов Center и Right аналогичен предыдущему.

Изменение элементов меню

Для модификации элемента меню чаще всего используются три свойства. Свойство Checked используется, чтобы добавить или удалить галочку рядом с элементом меню. С помощью свойства Enabled элемент меню можно пригасить, после чего пользователь не сможет его выбрать. Последнее свойство этой группы Caption представляет текст элемента меню. Изменяя текст элемента меню, вы указываете пользователю, что программа перешла в другое состояние.

Задание № 2

1) На форме расположить две панели , две кнопки и компонент RichEdit. Первая панель содержит два поля редактирования, а вторая два чекбокса. Также необходимо построить выпадающее меню File, Buttons, View, Pulldowns. Команды меню File – Open , SaveAs. Команды меню Buttons содержит изменяемый текст (с ‘Enable First’ на ‘Disable First’). Команды меню View – Edit Boxes, Check Boxes. Команды Pulldowns – Remove File, Disable Buttons, Disable View. Поместите в форму пиктограммы необходимых диалогов.
2) Код методов, которые загружают и сохраняют файлы :

```
procedure TForm1.Open1Click (Sender : TObject) ;
begin
  if  OpenDialog1.Execute then
     RichEdit1. Lines. LoadFromFile (OpenDialog1.FileName) ;
end;

procedure TForm1.SaveAs1Click (Sender : TObject) ;
begin
  if  SaveDialog1.Execute then
     RichEdit1. Lines. SaveToFile (SaveDialog1.FileName) ;
end;
```

3) Компоненты внутри панелей в действительности не используются. Однако вам необходимо воспользоваться двумя кнопками, чтобы скрыть или отобразить каждую из двух панелей вместе с управляющими элементами, которые в них содержатся. Те же действия можно выполнить с помощью двух команд меню : View/ Edit Boxes и View/ Check Boxes. Когда вы выбираете одну из этих команд меню или нажимаете одну из кнопок, происходит три разных действия. Во-первых, отображается или скрывается панель. Во-вторых, текст кнопки изменяется с Hide на Show, и наоборот. В-третьих, рядом с соответствующим элементом меню появляется или исчезает галочка. Ниже приведен код одного из двух методов, который связан с событиями щелчка как команды меню, так и кнопки :

```
procedure TForm1.ViewEdit1Click (Sender : TObject) ;
begin
  Panel1.Visible : = not Panel1.Visible ;
   ViewEdit1.Checked : = not ViewEdit1.Checked ;
    if Panel1.Visible then
       Button1.Caption : = 'Hide' ;
    else
       Button1.Caption : = 'Show' ;
end;
```

4) Команды меню Buttons применяют другой подход. Для показа текущего состояния они используют не галочку, а изменение текста. Кроме того, они разрешают или запрещают соответствующую команду View и кнопку.

```
procedure TForm1.ButtonsFirst1Click (Sender : TObject) ;
begin
ButtonsFirst1.Checked : = not ButtonsFirst1.Checked ;
  if ButtonsFirst1.Checked then
  begin
     ViewEdit1.Enabled : = False ;
      ButtonsFirst1.Caption : = 'Enable &First' ;
  end
  else
  begin
     ViewEdit1.Enabled : = True ;
      ButtonsFirst1.Caption : = 'Disable &First' ;
  end
end;
```

5) Команды меню Pulldowns должны скрывать выпадающее меню указанные в элементах и показывать галочку для выбранного элемента. Запишите код для каждого элемента этого меню самостоятельно.
6) Кнопки, расположенные на форме должны скрывать и показывать соответствующие панели

Лабораторная работа 16.

Получение ввода от мыши. Рисование в форме.

Когда пользователь нажимает одну из кнопок мыши, указатель которой находится над формой (или над компонентом), Windows посылает приложению несколько сообщений. Для написания кода, откликающегося на эти сообщения, Delphi определяет несколько событий. Основных событий два : OnMouseDown, которое происходит при нажатии одной из кнопок мыши, и OnMouseUp, которое происходит при освобождении кнопки.

Еще одно важное системное сообщение связано с перемещением мыши – сообщение OnMouseMove. Событие OnClick также доступно и в форме. Его основной смысл состоит в том, что левая кнопка мыши нажимается и отпускается над одним и тем же окном или компонентом. Однако в период между этими двумя действиями курсор может переместиться за пределы окна или компонента, причем левая кнопка мыши будет удерживаться нажатой. Если вы в определенной позиции нажмете кнопку мыши, а затем переместите мышь в другое место и отпустите кнопку, то щелчок не произойдет. В этом случае окно получает только сообщение о нажатии, несколько сообщений о перемещении и сообщение об освобождении.

События, связанные с мышью

Метод, соответствующий событию OnMouseDown имеет несколько параметров:

```
procedure TForm1.FormMouseDown (
  Sender : TObject ;
  Button : TMouseButton ;
  Shift : TShiftState ;
  X, Y : Integer ) ;
```

Кроме обычного параметра Sender здесь присутствуют еще четыре :

1) Button – показывает, какая из трех кнопок мыши была нажата. Возможные значения : mbRight, mbLeft, mbCenter.
2) Shift – показывает, какие влияющие на мышь клавиши были нажаты при возникновении события. Такой клавишей может быть Alt, Ctrl или Shift, нажатая вместе с самой кнопкой мыши. Данный параметр имеет тип множества, т.к. несколько клавиш могут быть нажаты одновременно. Это означает, что при анализе условия вы должны применять не проверку на равенство, а оператор in.
3) X и Y – показывают координаты позиции мыши относительно клиентской области.

Рисование в форме

Canvas (холст) – это область рисунка в форме и многих других графических компонентах. Чтобы получить доступ к пикселам формы, используйте свойство формы Canvas и свойство Pixels для Canvas. Свойство Pixels – это двумерный массив, соответствующий цветам отдельных

пикселов в Canvas. Canvas.Pixels[10,20] соответствует цвету пиксела, который находится на 10 пикселов правее и на 20 пикселов ниже точки отсчета. Обращайтесь с массивом пикселов как с любым другим свойством; чтобы изменить цвет пиксела присвойте новое значение. Чтобы определить цвет пиксела – прочитайте значение.

Каждое свойство Canvas имеет воображаемое перо для рисования линий и контуров. Свойство Pen (перо) определяет цвет и размер линий и границ фигур. Свойствами пера являются его цвет, размер (если это сплошная линия) или стиль. Работая с пером, вы можете прочитать (но не изменить) его текущую позицию (свойство PenPos). Позиция пера определяет исходную точку следующей линии, которую программа может нарисовать с помощью метода LineTo. Для изменения позиции вы можете применить метод MoveTo канвы.

Свойство Brush (кисть) определяет цвет очерченой поверхности. Кисть используется для закрашивания замкнутых фигур. Свойствами кисти являются ее цвет, стиль и иногда растровое изображение.

Свойство Font определяет шрифт, который используется методом холста TextOut для написания текста в форме. Шрифт имеет имя, размер, стиль, цвет и т.п.

Задание № 1

1) Поместить в форму меню Color с командами PenColor и BrushColor, которые будут соответственно изменять цвет пера и кисти с помощью стандартной диалоговой панели.
2) В форме реализовать рисование окружностей, эллипсов и прямоугольников различных размеров и цветов с помощью мыши, используя свойство Canvas формы. Дальше даются подсказки для реализации этой задачи.
3) Запишите следующий код для события OnMouseDown :

```
if Button = mbLeft then
  begin
    Center.X := X;
    Center.Y := Y;

 if ssShift in Shift then
      Circle := False
    else
      Circle := True;
  end;
```

Поле формы Circle типа Boolean определяет вид фигуры, значения координат центра фигуры записываются в поля формы Center типа TPoint.

4) Запишите следующий код для события OnMouseUp :

```
Radius.X := abs(Center.X - X);
```

```
Radius.Y := abs (center.Y-Y);
if Circle then
      Canvas.Ellipse(Center.X- Radius.X, Center.Y- Radius.Y, Center.X+
Radius.X, Center.Y+ Radius.Y)
    else
      Canvas.Rectangle(Center.X- Radius.X, Center.Y- Radius.Y, Center.X+
Radius.X, Center.Y+ Radius.Y);
```

5) Запишите следующий код для события OnMouseMove :

```
Caption := Format ('Координаты : x=%d, y=%d ' , [X, Y]);
```

6) Запустите приложение. Если все сделано правильно, то вы будете наблюдать изменение координат в заголовке формы при продвижении мыши; сможете рисовать окружности и эллипсы нужного размера (щелкая кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали); сможете рисовать прямоугольник нужного размера, используя ту же технологию.

Черчение и рисование в системе Windows

- Черчение – вы обращаетесь к канве и вызываете некоторые ее методы. Поскольку изображение не сохраняется, форма может частично или целиком потерять свое содержимое (при закрытии окна формы другим окном или при уменьшении размера окна формы).
- Рисование – это технология, которая позволит приложению перерисовывать всю ее поверхность при любых возможных условиях.

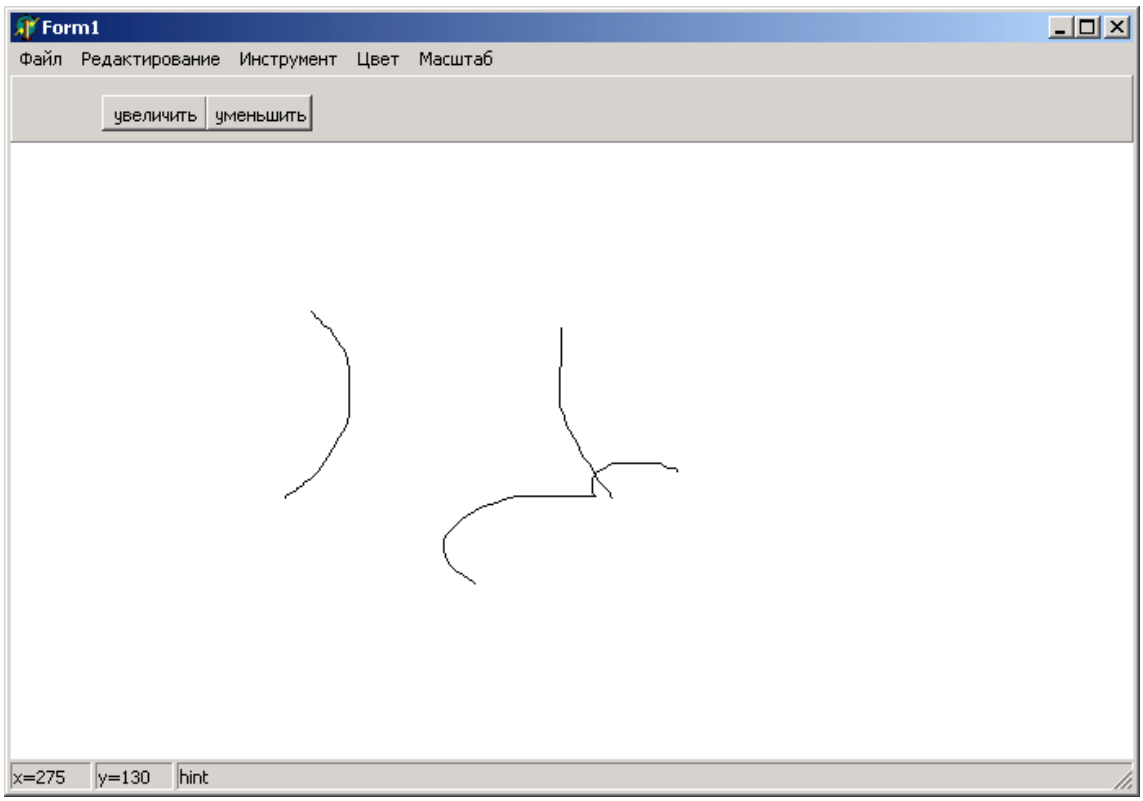
Для вызова перерисовки можно использовать методы Invalidate, UpDate, ReFresh и Repaint.

Задание № 2

1) Начиная с оператора if., код для события OnMouseUp перенести в код для события OnPaint.
2) В код для события OnMouseUp вставить в конце вызов метода Invalidate (который вызывает косвенно метод FormPaint, связанный с событием OnPaint).
3) Запустить приложение. При правильном выполнении всех инструкций, в форме будет рисоваться только одна фигура.
4) Выполните первое задание для компонента PainBox.

Задание № 3

Составить программу – простой графический редактор.



Редактор должен как открывать существующие графические изображения, так и рисовать и сохранять сам (по вариантам):

1. Прямые линии и пером
2. Окружности и эллипсы
3. Прямоугольники и квадраты

В редакторе необходимо предусмотреть изменение цвета линий и заливки. А также дополнительные возможности (по вариантам):

1. Изменение масштаба рисунка
2. Возможность копирования, вырезания и вставки
3. Вывода координат курсора

Лабораторная работа №17

Обработка исключительных ситуаций в Delphi

С целью поддержки структурной обработки исключительных ситуаций (*exception*) в Delphi введены новые расширения языка Pascal. В данной статье будет дано описание того, что из себя представляет такая обработка, почему она полезна, будут приведены соответствующий синтаксис Object Pascal и примеры использования исключительных ситуаций в Delphi.

Структурная обработка исключительных ситуаций

Структурная обработка исключительных ситуаций - это система, позволяющая программисту при возникновении ошибки (исключительной ситуации) связаться с кодом программы, подготовленным для обработки такой ошибки. Это выполняется с помощью языковых конструкций, которые как бы “охраняют” фрагмент кода программы и определяют обработчики ошибок, которые будут вызываться, если что-то пойдет не так в “охраняемом” участке кода. В данном случае понятие исключительной ситуации относится к языку и не нужно его путать с системными исключительными ситуациями (hardware exceptions), такими как General Protection Fault. Эти исключительные ситуации обычно используют прерывания и особые состояния “железа” для обработки критичной системной ошибки; исключительные ситуации в Delphi же независимы от “железа”, не используют прерываний и используются для обработки ошибочных состояний, с которыми подпрограмма не готова иметь дело. Системные исключительные ситуации, конечно, могут быть перехвачены и преобразованы в языковые исключительные ситуации, но это только одно из применений языковых исключительных ситуаций.

При традиционной обработке ошибок, ошибки, обнаруженные в процедуре обычно передаются наружу (в вызывавшую процедуру) в виде возвращаемого значения функции, параметров или глобальных переменных (флажков). Каждая вызывающая процедура должна проверять результат вызова на наличие ошибки и выполнять соответствующие действия. Часто, это просто выход еще выше, в более верхнюю вызывающую процедуру и т.д. : функция A вызывает B, B вызывает C, C обнаруживает ошибку и возвращает код ошибки в B, B проверяет возвращаемый код, видит, что возникла ошибка и возвращает код ошибки в A, A проверяет возвращаемый код и выдает сообщение об ошибке либо решает сделать что-нибудь еще, раз первая попытка не удалась.

Такая “пожарная бригада” для обработки ошибок трудоемка, требует написания большого количества кода в котором можно легко ошибиться и который трудно отлаживать.

Одна ошибка в коде программы или переприсвоение в цепочке возвращаемых значений может привести к тому, что нельзя будет связать ошибочное состояние с положением дел во внешнем мире. Результатом будет ненормальное поведение программы, потеря данных или ресурсов, или крах системы.

Структурная обработка исключительной ситуации замещает ручную обработку ошибок автоматической, сгенерированной компилятором системой уведомления. В приведенном выше примере, процедура A установила бы “охрану” со связанным обработчиком ошибки на фрагмент кода, в котором вызывается B. B просто вызывает C. Когда C обнаруживает ошибку, то создает (*raise*) исключительную ситуацию. Специальный код, сгенерированный компилятором и встроенный в Run-Time Library (RTL) начинает поиск обработчика данной исключительной ситуации. При поиске “защищенного” участка кода используется информация, сохраненная в стеке. В процедурах C и B нет такого участка, а в A - есть. Если один из обработчиков ошибок, которые используются в A, подходит по типу для возникшей в C исключительной ситуации, то программа переходит на его выполнение. При этом, область стека, используемая в B и C, очищается; выполнение этих процедур прекращается.

Если в A нет подходящего обработчика, то поиск продолжается в более верхнем уровне, и так может идти, пока поиск не достигнет подходящего обработчика ошибок среди используемых по умолчанию обработчиков в RTL. Обработчики ошибок из RTL только показывают сообщение об ошибке и форсированно прекращают выполнение программы. Любая исключительная ситуация, которая осталась необработанной, приведет к прекращению выполнения приложения.

Без проверки возвращаемого кода после каждого вызова подпрограммы, код программы должен быть более простым, а скомпилированный код - более быстрым. При наличии исключительных ситуаций подпрограмма B не должна содержать дополнительный код для проверки возвращаемого результата и передачи его в A. B ничего не должна делать для передачи исключительной ситуации, возникшей в C, в процедуру A - встроенная система обработки исключительных ситуаций делает всю работу.

Данная система называется *структурной,* поскольку обработка ошибок определяется областью “защищенного” кода; такие области могут быть вложенными. Выполнение программы не может перейти на произвольный участок кода; выполнение программы может перейти только на обработчик исключительной ситуации активной программы.

Модель исключительных ситуаций в Delphi

Модель исключительных ситуаций в Object Pascal является невозобновляемой(*non-resumable*). При возникновении исключительной ситуации Вы уже не сможете вернуться в точку, где она возникла, для продолжения выполнения программы (это позволяет сделать возобновляемая(*resumable*) модель). Невозобновляемые исключительные ситуации разрушают стек, поскольку они сканируют его в поисках обработчика; в возобновляемой модели необходимо сохранять стек, состояние регистров процессора в точке возникновения ошибки и выполнять поиск обработчика и его выполнение в отдельном стеке. Возобновляемую систему обработки исключительных ситуаций гораздо труднее создать и применять, нежели невозобновляемую.

Синтаксис обработки исключительных ситуаций

Теперь, когда мы рассмотрели, что такое исключительные ситуации, давайте дадим ясную картину, как они применяются. Новое ключевое слово, добавленное в язык Object Pascal - **try**. Оно используется для обозначения первой части защищенного участка кода. Существует два типа защищенных участков:

- *try..except*
- *try..finally*

Первый тип используется для обработки исключительных ситуаций. Его синтаксис:

```
try
  Statement 1;
  Statement 2;
  ...
except
  on Exception1 do Statement;
  on Exception2 do Statement;
  ...
else
  Statements; {default exception-handler}
end;
```

Для уверенности в том, что ресурсы, занятые вашим приложением, освободятся в любом случае, Вы можете использовать конструкцию второго типа. Код, расположенный в части *finally,* выполняется в любом случае, даже если возникает исключительная ситуация. Соответствующий синтаксис:

```
try
  Statement1;
  Statement2;
  ...
finally
  Statements;  { These statements always execute }
end;
```

Примеры обработки исключительных ситуаций

Ниже приведены процедуры A,B и C, обсуждавшиеся ранее, воплощенные в новом синтаксисе Object Pascal:

```
type
  ESampleError = class(Exception);

var
  ErrorCondition: Boolean;
```

```
procedure C;
begin
  writeln('Enter C');
  if (ErrorCondition) then
  begin
    writeln('Raising exception in C');
    raise ESampleError.Create('Error!');
  end;
  writeln('Exit C');
end;

procedure B;
begin
  writeln('enter B');
  C;
  writeln('exit B');
end;

procedure A;
begin
  writeln('Enter A');
  try
    writeln('Enter A''s try block');
    B;
    writeln('After B call');
  except
    on ESampleError do
      writeln('Inside A''s ESampleError handler');
    on ESomethingElse do
      writeln('Inside A''s ESomethingElse handler');
  end;
  writeln('Exit A');
end;

begin
  writeln('begin main');
  ErrorCondition := True;
  A;
  writeln('end main');
end.
```

При ErrorCondition = True программа выдаст:

```
begin main
Enter A
Enter A's try block
enter B
Enter C
Raising exception in C
Inside A's ESampleError handler
Exit A
end main
```

Возможно вас удивила декларация типа 'ESampleError =class' вместо '=object'; это еще одно новое расширение языка. Delphi вводит новую модель объектов, доступную через декларацию типа '=class'. Описание новой объектной модели дается в других уроках. Здесь же достаточно сказать, что исключительные ситуации (exceptions) являются классами, частью новой объектной модели.

Процедура C проверяет наличие ошибки (в нашем случае это значение глобальной переменной) и, если она есть (а это так), C вызывает(raise) исключительную ситуацию класса ESampleError.

Процедура A помещает часть кода в блок *try..except.* Первая часть этого блока содержит часть кода, аналогично конструкции *begin..end*. Эта часть кода завершается ключевым словом *except*, далее следует один или более обработчиков исключительных ситуаций *on xxxx do yyyy*, далее может быть включен необязательный блок *else,* вся конструкция заканчивается *end;*. В конструкции, назначающей определенную обработку для конкретной исключительной ситуации (*on xxxx do yyyy*), после резервного слова *on* указывается класс исключительной ситуации, а после *do* следует собственно код обработки данной ошибки. Если возникшая исключительная ситуация подходит по типу к указанному после *on*, то выполнение программы переходит сюда (на код после *do*). Исключительная ситуация подходит в том случае, если она того же класса, что указан в *on,* либо является его потомком. Например, в случае *on EFileNotFound* обрабатываться будет ситуация, когда файл не найден. А в случае *on EFileIO* - все ошибки при работе с файлами, в том числе и предыдущая ситуация. В блоке *else* обрабатываются все ошибки, не обработанные до этого.

Приведенные в примере процедуры содержат код (строка с writeln), который отображает путь выполнения программы. Когда C вызывает exception, программа сразу переходит на обработчик ошибок в процедуре A, игнорируя оставшуюся часть кода в процедурах B и C.

После того, как найден подходящий обработчик ошибки, поиск оканчивается. После выполнения кода обработчика, программа продолжает выполняться с оператора, стоящего после слова *end* блока *try..except* (в примере - writeln('Exit A')).

Конструкция *try..except* подходит, если известно, какой тип ошибок нужно обрабатывать в конкретной ситуации. Но что делать, если требуется выполнить некоторые действия в любом случае, произошла ошибка или нет? Это тот случай, когда понадобится конструкция *try..finally*.

Рассмотрим модифицированную процедуру B:

```
procedure NewB;
var
  P: Pointer;
begin
  writeln('enter B');
  GetMem(P, 1000);
  C;
  FreeMem(P, 1000);
  writeln('exit B');
end;
```

Если C вызывает исключительную ситуацию, то программа уже не возвращается в процедуру B. А что же с теми 1000 байтами памяти, захваченными в B? Строка FreeMem(P,1000) не выполнится и Вы потеряете кусок памяти. Как это исправить? Нужно ненавязчиво включить процедуру B в процесс, например:

```
procedure NewB;
var
  P: Pointer;
begin
  writeln('enter NewB');
  GetMem(P, 1000);
  try
    writeln('enter NewB''s try block');
    C;
    writeln('end of NewB''s try block');
  finally
    writeln('inside NewB''s finally block');
    FreeMem(P, 1000);
```

```
  end;
  writeln('exit NewB');
end;
```

Если в A поместить вызов NewB вместо B, то программа выведет сообщения следующим образом:

```
begin main
Enter A
Enter A's try block
enter NewB
enter NewB's try block
Enter C
Raising exception in C
inside NewB's finally block
Inside A's ESampleError handler
Exit A
end main
```

Код в блоке *finally* выполнится при любой ошибке, возникшей в соответствующем блоке *try*. Он же выполнится и в том случае, если ошибки не возникло. В любом случае память будет освобождена. Если возникла ошибка, то сначала выполняется блок *finally,* затем начинается поиск подходящего обработчика. В штатной ситуации, после блока *finally* программа переходит на следующее предложение после блока.

Почему вызов GetMem не помещен внутрь блока *try*? Этот вызов может окончиться неудачно и вызвать exception EOutOfMemory. Если это произошло, то FreeMem попытается освободить память, которая не была распределена. Когда мы размещаем GetMem вне защищаемого участка, то предполагаем, что B сможет получить нужное количество памяти, а если нет, то более верхняя процедура получит уведомление EOutOfMemory.

А что, если требуется в B распределить 4 области памяти по схеме все-или-ничего? Если первые две попытки удались, а третья провалилась, то как освободить захваченную область память? Можно так:

```
procedure NewB;
var
  p,q,r,s: Pointer;
begin
  writeln('enter B');
  P := nil;
  Q := nil;
  R := nil;
  S := nil;
  try
    writeln('enter B''s try block');
    GetMem(P, 1000);
    GetMem(Q, 1000);
    GetMem(R, 1000);
    GetMem(S, 1000);
    C;
    writeln('end of B''s try block');
  finally
    writeln('inside B''s finally block');
    if P <> nil then FreeMem(P, 1000);
    if Q <> nil then FreeMem(Q, 1000);
    if R <> nil then FreeMem(R, 1000);
    if S <> nil then FreeMem(S, 1000);
  end;
  writeln('exit B');
```

```
end;
```

Установив сперва указатели в NIL, далее можно определить, успешно ли прошел вызов GetMem.

Оба типа конструкции *try* можно использовать в любом месте, допускается вложенность любой глубины. Исключительную ситуацию можно вызывать внутри обработчика ошибки, конструкцию *try* можно использовать внутри обработчика исключительной ситуации.

Вызов исключительной ситуации

В процедуре C из примера мы уже могли видеть, как должна поступать программа при обнаружении состояния ошибки - она вызывает исключительную ситуацию:

```
raise ESampleError.Create('Error!');
```

После ключевого слова *raise* следует код, аналогичный тому, что используется для создания нового экземпляра класса. Действительно, в момент вызова исключительной ситуации создается экземпляр указанного класса; данный экземпляр существует до момента окончания обработки исключительной ситуации и затем автоматически уничтожается. Вся информация, которую нужно сообщить в обработчик ошибки передается в объект через его конструктор в момент создания.

Почти все существующие классы исключительных ситуаций являются наследниками базового класса Exception и не содержат новых свойств или методов. Класс Exception имеет несколько конструкторов, какой из них конкретно использовать - зависит от задачи. Описание класса Exception можно найти в on-line Help.

Доступ к экземпляру объекта exception

До сих пор мы рассматривали механизмы защиты кода и ресурсов, логику работы программы в исключительной ситуации. Теперь нужно немного разобраться с тем, как же обрабатывать возникшую ошибку. А точнее, как получить дополнительную информацию о коде ошибки, текст сообщения и т.п.

Как уже говорилось, при вызове исключительной ситуации (raise) автоматически создается экземпляр соответствующего класса, который и содержит информацию об ошибке. Весь вопрос в том, как в обработчике данной ситуации получить доступ к этому объекту.

Рассмотрим модифицированную процедуру A в нашем примере:

```
procedure NewA;
begin
  writeln('Enter A');
  try
    writeln('Enter A''s try block');
    B;
    writeln('After B call');
  except

    on E: ESampleError do writeln(E.Message);

    on ESomethingElse do
      writeln('Inside A''s ESomethingElse handler');
  end;
  writeln('Exit A');
end;
```

Здесь все изменения внесены в строку

```
on ESE: ESampleError do writeln(ESE.Message);
```

Пример демонстрирует еще одно новшество в языке Object Pascal - создание локальной переменной. В нашем примере локальной переменной является ESE - это тот самый экземпляр класса ESampleError, который был создан в процедуре C в момент вызова исключительного состояния. Переменная ESE доступна только внутри блока *do.* Свойство Message объекта ESE содержит сообщение, которое было передано в конструктор Create в процедуре C.

Есть еще один способ доступа к экземпляру exception - использовать функцию *ExceptionObject:*

```
on ESampleError do
  writeln(ESampleError(ExceptionObject).Message);
```

Предопределенные обработчики исключительных ситуаций

Ниже Вы найдете справочную информацию по предопределенным исключениям, необходимую для профессионального программирования в Delphi.

- **Exception** - базовый класс-предок всех обработчиков исключительных ситуаций.
- **EAbort** - "скрытое" исключение. Используйте его тогда, когда хотите прервать тот или иной процесс с условием, что пользователь программы не должен видеть сообщения об ошибке. Для повышения удобства использования в модуле *SysUtils* предусмотрена процедура *Abort*, определенная, как:

```
procedure Abort;
begin
  raise EAbort.CreateRes(SOperationAborted) at ReturnAddr;
end;
```

- **EComponentError -** вызывается в двух ситуациях:
  1) при попытке регистрации компоненты за пределами процедуры Register;
  2) когда имя компоненты не уникально или не допустимо.
- **EConvertError -** происходит в случае возникновения ошибки при выполнении функций *StrToInt* и *StrToFloat*, когда конвертация строки в соответствующий числовой тип невозможна.
- **EInOutError** - происходит при ошибках ввода/вывода при включенной директиве {$I+}.
- **EIntError** - предок исключений, случающихся при выполнении целочисленных операций.
  - **EDivByZero** - вызывается в случае деления на ноль, как результат RunTime Error 200.
  - **EIntOverflow** - вызывается при попытке выполнения операций, приводящих к переполнению целых переменных, как результат RunTime Error 215 при включенной директиве {$Q+}.
  - **ERangeError** - вызывается при попытке обращения к элементам массива по индексу, выходящему за пределы массива, как результат RunTime Error 201 при включенной директиве {$R+}.

- **EInvalidCast** - происходит при попытке приведения переменных одного класса к другому классу, несовместимому с первым (например, приведение переменной типа TListBox к TMemo).

- **EInvalidGraphic** - вызывается при попытке передачи в *LoadFromFile* файла, несовместимого графического формата.

- **EInvalidGraphicOperation** - вызывается при попытке выполнения операций, неприменимых для данного графического формата (например, Resize для TIcon).

- **EInvalidObject -** реально нигде не используется, объявлен в *Controls.pas*.

- **EInvalidOperation** - вызывается при попытке отображения или обращения по Windows-обработчику (handle) контрольного элемента, не имеющего владельца (например, сразу после вызова MyControl:=TListBox.Create(...) происходит обращение к методу Refresh).

- **EInvalidPointer** - происходит при попытке освобождения уже освобожденного или еще неинициализированного указателя, при вызове Dispose(), FreeMem() или деструктора класса.

- **EListError** - вызывается при обращении к элементу наследника TList по индексу, выходящему за пределы допустимых значений (например, объект TStringList содержит только 10 строк, а происходит обращение к одиннадцатому).

- **EMathError** - предок исключений, случающихся при выполнении операций с плавающей точкой.

    - **EInvalidOp** - происходит, когда математическому сопроцессору передается ошибочная инструкция. Такое исключение не будет до конца обработано, пока Вы контролируете сопроцессор напрямую из ассемблерного кода.

    - **EOverflow** - происходит как результат переполнения операций с плавающей точкой при слишком больших величинах. Соответствует RunTime Error 205.

    - **Underflow** - происходит как результат переполнения операций с плавающей точкой при слишком малых величинах. Соответствует RunTime Error 206.

    - **EZeroDivide** - вызывается в результате деления на ноль.

- **EMenuError** - вызывается в случае любых ошибок при работе с пунктами меню для компонент TMenu, TMenuItem, TPopupMenu и их наследников.

- **EOutlineError** - вызывается в случае любых ошибок при работе с TOutLine и любыми его наследниками.

- **EOutOfMemory** - происходит в случае вызовов New(), GetMem() или конструкторов классов при невозможности распределения памяти. Соответствует RunTime Error 203.

- **EOutOfResources** - происходит в том случае, когда невозможно выполнение запроса на выделение или заполнение тех или иных Windows ресурсов (например таких, как обработчики - handles).

- **EParserError** - вызывается когда Delphi не может произвести разбор и перевод текста описания формы в двоичный вид (часто происходит в случае исправления текста описания формы вручную в IDE Delphi).

- **EPrinter** - вызывается в случае любых ошибок при работе с принтером.

- **EProcessorException** - предок исключений, вызываемых в случае прерывания процессора- hardware breakpoint. Никогда не включается в DLL, может обрабатываться только в “цельном” приложении.

    - **EBreakpoint** - вызывается в случае останова на точке прерывания при отладке в IDE Delphi. Среда Delphi обрабатывает это исключение самостоятельно.

    - **EFault** - предок исключений, вызываемых в случае невозможности обработки процессором тех или иных операций.

        - **EGPFault** - вызывается, когда происходит “общее нарушение защиты” - General Protection Fault. Соответствует RunTime Error 216.

        - **EInvalidOpCode** - вызывается, когда процессор пытается выполнить недопустимые инструкции.

        - **EPageFault** - обычно происходит как результат ошибки менеджера памяти Windows, вследствие некоторых ошибок в коде Вашего приложения. После такого исключения рекомендуется перезапустить Windows.

        - **EStackFault** - происходит при ошибках работы со стеком, часто вследствие некорректных попыток доступа к стеку из фрагментов кода на ассемблере. Компиляция Ваших программ со включенной проверкой работы со стеком {$S+} помогает отследить такого рода ошибки.

    - **ESingleStep** - аналогично EBreakpoint, это исключение происходит при пошаговом выполнении приложения в IDE Delphi, которая сама его и обрабатывает.

- **EPropertyError** - вызывается в случае ошибок в редакторах свойств, встраиваемых в IDE Delphi. Имеет большое значение для написания надежных property editors. Определен в модуле DsgnIntf.pas.

- **EResNotFound** - происходит в случае тех или иных проблем, имеющих место при попытке загрузки ресурсов форм - файлов .DFM в режиме дизайнера. Часто причиной таких исключений бывает нарушение соответствия между определением класса формы и ее описанием на уровне ресурса (например,вследствие изменения порядка следования полей-ссылок на компоненты, вставленные в форму в режиме дизайнера).

- **EStreamError** - предок исключений, вызываемых при работе с потоками.

    - **EFCreateError** - происходит в случае ошибок создания потока (например, при некорректном задании файла потока).

    - **EFilerError** - вызывается при попытке вторичной регистрации уже зарегистрированного класса (компоненты). Является, также, предком специализированных обработчиков исключений, возникающих при работе с классами компонент.

        - **EClassNotFound** - обычно происходит, когда в описании класса формы удалено поле-ссылка на компоненту, вставленную в форму в режиме дизайнера. Вызывается, в отличие от EResNotFound, в RunTime.

        - **EInvalidImage** - вызывается при попытке чтения файла, не являющегося ресурсом, или разрушенного файла ресурса, специализированными функциями чтения ресурсов (например, функцией ReadComponent).

        - **EMethodNotFound** - аналогично EClassNotFound, только при несоответствии методов, связанных с теми или иными обработчиками событий.

        - **EReadError** - происходит в том случае, когда невозможно прочитать значение свойства или другого набора байт из потока (в том числе ресурса).

    - **EFOpenError** - вызывается когда тот или иной специфированный поток не может быть открыт (например, когда поток не существует).

- **EStringListError** - происходит при ошибках работы с объектом TStringList (кроме ошибок, обрабатываемых TListError).

Исключения, возникающие при работе с базами данных

Delphi, обладая прекрасными средствами доступа к данным, основывающимися на интерфейсе IDAPI, реализованной в виде библиотеки Borland Database Engine (BDE), включает ряд обработчиков исключительных ситуаций для регистрации ошибок в компонентах VCL работающим с БД. Дадим краткую характеристику основным из них:

- **EDatabaseError** - наследник *Exception* ; происходит при ошибках доступа к данным в компонентах-наследниках TDataSet. Объявлено в модуле *DB*. Ниже приведен пример из Delphi On-line Help, посвященный этому исключению:

```
repeat {пока не откроем таблицу или не нажмем кнопку Cancel}
  try
    Table1.Active := True; {Пытаемся открыть таблицу}
    Break; { Если нет ошибки - прерваем цикл}
  except
    on EDatabaseError do
      {Если нажата OK - повторяем попытку открытия Table1}
      if MessageDlg('Не могу открыть Table1', mtError,
               [mbOK, mbCancel], 0) <> mrOK
      then
        raise;
  end;
until False;
```

- **EDBEngineError** - наследник *EDatabaseError* ; вызывается, когда происходят ошибки BDE или на сервере БД. Объявлено в модуле *DB*:

```
EDBEngineError = class(EDatabaseError)
private
  FErrors: TList;
  function GetError(Index: Integer): TDBError;
  function GetErrorCount: Integer;
public
  constructor Create(ErrorCode: DBIResult);
  destructor Destroy;
  property ErrorCount: Integer;
  property Errors[Index: Integer]: TDBError;
end;
```

Особенно важны два свойства класса *EDBEngineError* :
Errors - список всех ошибок, находящихся в стеке ошибок BDE. Индекс первой ошибки 0;
ErrorCount - количество ошибок в стеке.

Объекты, содержащиеся в *Errors*, имеют тип *TDBError.* Доступные свойства класса *TDBError*:
ErrorCode - код ошибки, возвращаемый Borland Database Engine;
Category - категория ошибки, описанной в ErrorCode;
SubCode - 'субкод' ошибки из ErrorCode;
NativeError - ошибка, возвращаемая сервером БД. Если NativeError 0, то ошибка в ErrorCode не от сервера;
Message - сообщение, переданное сервером, если NativeError не равно 0; сообщение BDE - в противном случае.

- **EDBEditError** - наследник *Exception* ; вызывается, когда данные не совместимы с маской ввода, наложенной на поле. Объявлено в модуле *Mask*.

Задание:

1. Используя два типа обработки исключительных систуаций продемонстрировать их использование в своих программах
2. Написать свой собственный обработчик исключительной ситуации

Лабораторная работа 18.
Классы и модули.

Классы и сокрытие информации

Класс может содержать сколько угодно данных и любое количество методов. Однако для соблюдения всех правил объектно-ориентированного подхода данные должны быть скрыты, или инкапсулированы, внутри использующего их класса. Использование метода для получения доступа к внутреннему представлению объекта уменьшает риск возникновения ошибочных ситуаций и позволяет автору класса модифицировать внутреннее представление в будущих версиях. В Object Pascal имеются две различные конструкции, подразумевающих инкапсуляцию, защиту и доступ к переменным: классы и модули. С классами связаны некоторые специальные ключевые слова – спецификаторы доступа:

- private – элементы интерфейса класса видны только в пределах модуля, в котором определен класс. Вне этого модуля private-элементы не видны и недоступны. Если в одном модуле создано несколько классов, то все они видят private-разделы друг друга.
- Раздел public не накладывает ограничений на область видимости перечисленных в нем полей, методов и свойств. Их можно вызывать в любом другом модуле программы.
- Раздел published также не ограничивает область видимости, однако в нем перечислены свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы. Раздел published используется при разработке компонентов. Без объявления раздел считается объявленным как published. Такой умалчиваемый раздел располагается в самом начале объявления класса любой формы и продолжается до первого объявленного раздела. В раздел published среда помещает описание вставляемых в форму компонентов. Сюда не нужно помещать собственные элементы или удалять элементы, вставленные средой.
- Раздел protected доступен только методам самого класса, а также любым потомкам, независимо от того, находятся ли они втом же модуле или нет.

Порядок следования разделов может быть любой, любой раздел может быть пустым.

Классы и модули

Приложения Delphi интенсивно используют модули. За каждой формой скрывается соответствующий ей модуль. Однако модули не обязаны иметь соответсвующие формы.

Модуль содержит раздел interface, где объявлено все, что доступно для других модулей, и раздел implementation с реальным кодом. Наконец, модуль может иметь два необязательных раздела : initialization с некоторым кодом запуска, который выполняется при загрузке в память программы, использующей данный модуль, и finalization, который выполняется при завершении программы.

Предложение uses в начале раздела interface к каким другим модулям мы должны получить доступ из раздела interface текущего модуля. Если же на другие модули необходимо сослаться из клда подпрограмм и методов, вы должны добавить новое предложение uses в начале раздела implementation.

В интерфейсе модуля можно объявить несколько различных элементов, в том числе процедуры, функции, глобальные переменные и типы данных. Также можно поместить в модуль класс. Delphi это делает автоматически при создании каждой формы. Чтобы создать новый не относящийся к форме модуль, выберите команду File/New и отметьте на странице New появившегося окна Object Repository элемент Unit.

Модули и область видимости

Область видимости идентификатора (переменной, процедуры, функции или типа данных, определяет ту часть кода, в которой доступен этот идентификатор. Основное правило состоит в том, что идентификатор является значимым только внутри его области видимости.

Если вы объявили идентификатор внутри блока определения процедуры, вы не сможете использовать данную переменную вне этой процедуры. Область видимости идентификатора охватывает всю процедуру, включая вложенные блоки.

Если вы объявили идентификатор в области реализации модуля, вы не можете применить его вне модуля, номожете использовать в любом блоке и процедуре, определенных внутри модуля. Если вы объявили идентификатор в интерфейсной части модуля, его область
видимости распространяется на любой другой модуль, где объявлен идентификатор. Любой
идентификатор, объявленный в интерфейсе модуля, является глобальным; все другие идентификаторы принято называть локальными.

Модули и программы

Приложение Delphi создается из файлов исходного текста двух разных видов. Это один или несколько модулей и один файл программы.

Модули можно считать вторичными файлами, к которым обращается основная часть приложения – программа. На практике файл программы обычно является автоматически сгенерированным файлом с ограниченной ролью. Он нужен только для запуска программы, которая выполнет основную форму. Код файла программы, или файла проекта Delphi (DPR), можно отредактировать вручную или с помощью Project Manager и некоторых опций проекта.

Информация о типе на этапе выполнения

Правило языка Object Pascal о совместимости типов для классов-потомков позволяет вам использовать класс-потомок там, где ожидается класс предок, обратное невозможно. Теперь предположим, что класс Dog содержит функцию Eat, которая отсутствует в классе Animal.

Если переменная MyAnimal ссылается на объект типа Dog, вызов этой функции должен быть разрешен. Но если вы попытаетесь вызвать эту

функцию, а переменная ссылается на объект другого класса, возникнет ошибка. Поскольку компилятор не способен определить, будет ли значение правильным на этапе выполнения, то делая явное приведение типов, мы рискуем вызвать опасную ошибку этапа выполнения программы.

Для решения данной проблемы можно воспользоваться некоторыми подходами, основанными на системе RTTI. Каждый объект энает свой тип и своего предка и можем получить эту информацию с помощью операции is. Параметрами операции is являются объект и тип:

```
if MyAnimal is Dog then …
```

Выражение is становится истинным, только если в настоящее время объект MyAnimal имеет тип Dog или тип потомка от Dog. Другими словами, это выражение приобретает значение True, если вы можете без риска присвоить объект (MyAnimal) переменной заданного типа данных (Dog). Такое прямое приведение можно выполнить так:

```
if MyAnimal is Dog then
  MyDog := Dog (MyAnimal) ;
```

То же действие можно выполнить напрямую с помощью другой операции RTTI – as. Мы можем написать так:

```
MyDog := MyAnimal as Dog ;
Text := MyDog. Eat ;
```

Если мы хотим вызвать функцию Eat, можно использовать и другую нотацию:

```
(MyAnimal as Dog) . Eat ;
```

Результатом выражения будет объект с типом данных класса Dog, поэтому выможете применить к нему любой метод этого класса.

Приведение с операцией as отличается от традиционного приведения тем, что в случае несовместимости типа объекта с типом, к которому вы пытаетесь его привести, порождается исключение EInvalidCast.

Чтобы избежать этого исключения, используйте операцию is и в случае успеха делайте простое приведение:

```
if MyAnimal is Dog then
 (Dog (MyAnimal) ) . Eat ;
```

Задание

Открыть модуль, не связанный с формой и поместить в него три класса:

- Класс Animal, который содержит в разделе public объявление конструктора Create и объявление метода-функции: Verse – звук, издаваемый животным. Тип результата возвращаемого функцией – string. Метод Verse объявить виртуальным и абстрактным. В разделе private класса определить переменную Kind : string.
- Класс Dog объявить потомком класса Animal. В разделе public этого класса объявить конструктор и методы Verse и Eat. Метод Eat типа string объявить виртуальным (пища животного).

- Класс Cat объявить потомком класса Animal. Раздел public класса содержит те же определения, что и соответствующий раздел класса Dog.

В реализациях конструктора каждого класса переменной Kind присваивается имя соответствующего животного, напрамир для класса Animal : Kind := ‘An Animal’.

В реализациях методов Verse возвращается звук, издаваемый животным, например Verse := ‘Mieow’.

В реализациях методов Eat возвращается название пищи, которой питается соответствующее животное.

- Задать имя модуля и имя проекта, в который этот модуль будет включен.
- Добавить в проект форму, которой присвоить имя Animals, также задать имя модулю, связанному с формой.
- В форме расположить три кнопки опций (компонент RadioButton) с названиями Animal, Dog, Cat ; кнопкой команды (компонент Button) с названием Kind и две крупный надписи (компонент Label) . Нажатию одной из кнопок опций будет соответствовать выбор животного. При нажатии кнопки команды надписи должны отобразить звук, издаваемый животным и его пищу.
- Определите в классе формы private-переменную MyAnimal. Запишите код для обработчика события OnCreate формы, где создается объект типа Dog на который ссылается переменная MyAnimal.
- В обработчиках события OnClick каждой кнопки опций записать код, который удаляет текущий объект и создает новый.
- В обработчике события OnClick кнопки команды записать код, который будет помещать в надписи звук, издаваемый животным и его пищу. Для работы с методом Eat используйте операцию is для приведения типов.
- Если вы все сделали правильно, при запуске приложения надписи будут отображать пищу и звук для Dog и Cat и приложение завершит работу по ошибке при выборе Animal.
- Уберите ключевое слово abstract в объявлении метода Verse. Запустите приложение снова. Посмотрите что изменилось в работе приложения. Объясните различия.
- Попробуйте использовать метод Eat без приведения типов (без is).
- Разработайте два класса потомка от Dog или Cat (согласно варианту), которые будут отображать особенности двух пород собак или кошек соответственно. Разработайте методы для этих классов, позволяющие получить некоторые характеристики породы (рост, длина шерсти, длина ушей и т.д.). Дополните форму компонентами позволяющими увидеть все характеристики разработанных классов.
- Потомки должны показываться на форме, если пользователь выбирает родителя, и должны быть скрыты, если выбран другой класс.

Поскольку Delphi является открытой средой и позволяет не только использовать объекты из Библиотеки Визуальных Компонент (VCL) в своей программе, но и создавать новые объекты. Причем, ничего другого, кроме Delphi, для этого не требуется. Создание нового объекта в Delphi не является очень сложной задачей, хотя для этого и требуется знание Windows API, объектно-ориентированного программирования и иерархии классов в VCL.

Может возникнуть вопрос; если в Delphi уже есть своя библиотека, то зачем еще создавать какие-то объекты? Ответ прост: нельзя создать библиотеку на все случаи жизни и на все вкусы. Новые компоненты, во-первых, позволяют расширить область применения Delphi: например, с помощью библиотек объектов третьих фирм разрабатывать приложения для работы в Internet. Во-вторых, позволяют дополнить или настроить для себя имеющиеся в VCL объекты (например, переопределить значения свойств, устанавливаемые по умолчанию).

Добавление новых объектов в VCL

Предположим, что у вас появился уже готовый компонент. Как его добавить в VCL? Для этого выберите пункт меню Options|Install Components… Появится диалог, как на рис.1

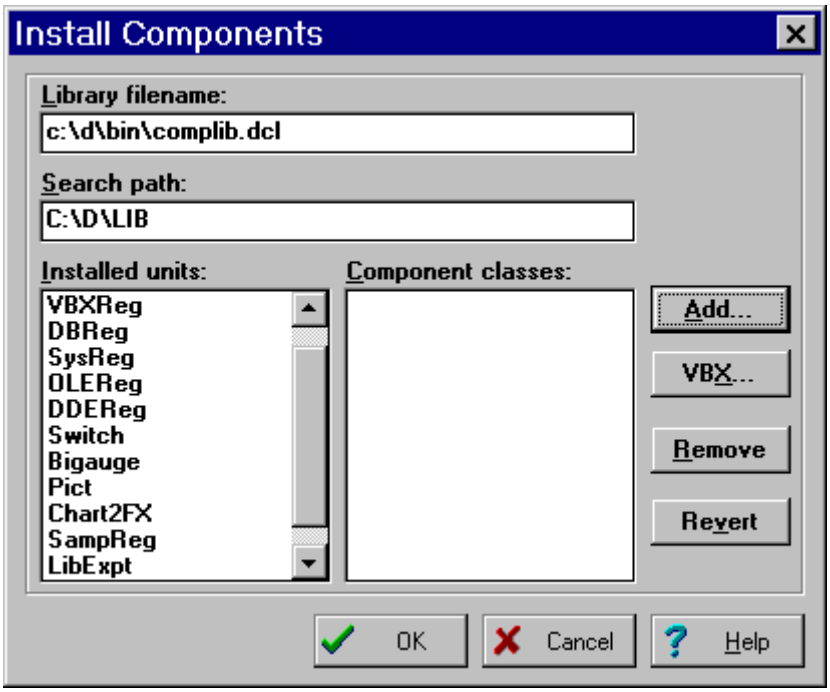


Рис.1: Диалог установки нового компонента

Нажмите “Add” и укажите модуль, содержащий процедуру регистрации, нажмите “OK” и после успешной перекомпиляции новый объект появится в палитре.

Заготовка для нового компонента

В среде Delphi есть специальный эксперт, создающий заготовку для нового компонента. Вызвать его можно в пункте меню File|New Component… (см рис.2)

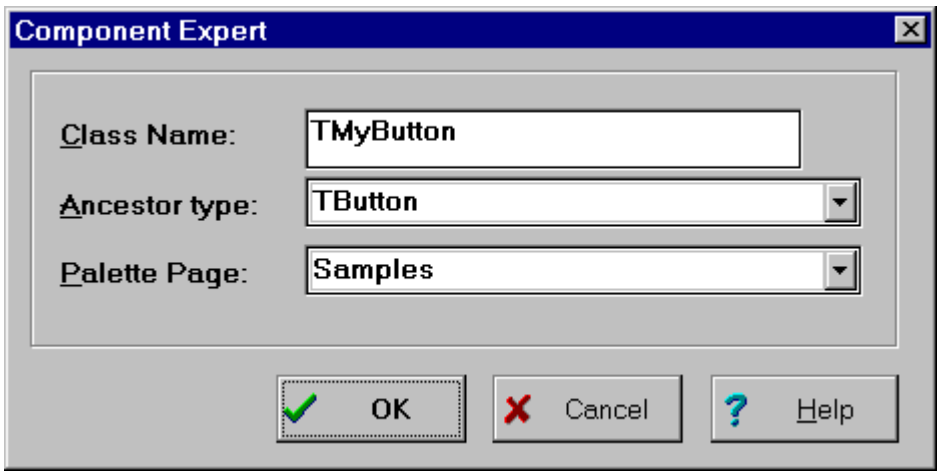


Рис.2: Эксперт для создания нового компонента

В диалоге нужно указать имя нового класса (например, TMyButton), предка класса (TButton) и страницу палитры, куда поместить новый компонент (Samples). Если нажать “OK”, то эксперт создаст модуль - заготовку для нового компонента:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls;

type
  TMyButton = class(TButton)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TMyButton]);
end;

end.
```

Модуль содержит декларацию нового класса и процедуру его регистрации в Палитре Компонент. В процедуре *RegisterComponents* первый параметр - имя страницы (можно указать свое имя - появится новая страница); второй параметр - множество объектов для регистрации.

Теперь модуль нужно сохранить под новым именем (например, NEW_BTN.PAS) и приступить к дописыванию новых свойств и методов. После того, как эта работа закончена и новый компонент отлажен,

можно добавить его в Палитру (см. предыдущую главу). Но перед этим желательно создать файл ресурсов, в котором будет лежать пиктограмма для представления данного объекта в Палитре Компонент. Файл ресурсов можно создать с помощью программы Resource Workshop, называться он должен точно так же, как модуль регистрации компонента и иметь расширение .DCR (т.е., если объект регистрируется в модуле NEW_BTN.PAS, то тогда имя файла ресурсов будет NEW_BTN.DCR). В файле ресурсов должен находиться ресурс типа BITMAP - картинка размером 28x28 точки (можно меньше), название картинки должно совпадать с именем класса (в нашем случае TMYBUTTON).

Соглашения по наименованиям

Если вы рассматривали исходные тексты VCL, то могли видеть, что они следуют нескольким простым соглашениям при определении новых классов. Delphi этого не требует, имена методов, свойств и т.п. могут быть любыми, компилятору это безразлично. Но если следовать этим соглашениям, то разработка новых компонентов и чтение исходных текстов станет существенно проще.

Итак:

- Все декларации типов начинаются на букву T. Еще раз, Delphi не требует этого, но это делает очевидным, что "TEdit", например, есть определение типа, а не переменная или поле класса.
- Имена свойствам нужно давать легко читаемые и информативные. Нужно помнить, что пользователь будет их видеть в Инспекторе Объектов. И имя вроде "TextOrientation" много удобнее, нежели "TxtOr". То же самое относится к методам. Методы, доступные пользователю, должны иметь удобные названия.
- При создании свойств типа Event, имя такого свойства должно начинаться с “On” (например, OnClick, OnCreate и т.д.).
- Имя метода для чтения свойства должен начинаться со слова “Get”. Например, метод GetStyle должен выполнять чтение для свойства Style.
- Имя метода для записи свойства должен начинаться со слова “Set”. Например, метод SetStyle должен выполнять запись в свойство Style.
- Внутреннее поле для хранения данных свойства должно носить имя, начинающееся с буквы “F”. Например, свойство Handle могло бы храниться в поле FHandle.

Конечно же, есть исключения из правил. Иногда бывает удобнее их нарушить, например, класс TTable имеет свойства типа Event, которые называются BeforePost, AfterPost и т.п.

Выбор предка

Прежде, чем приступить к написанию кода, нужно определиться, хотя бы приблизительно, что за компонент вы собираетесь делать. Далее, исходя из его предполагаемых свойств, определите класс-предок. В VCL имеется несколько базовых классов, рекомендуемых для наследования:

- **TObject** - Можно использовать в качестве предка, если с этим компонентом не нужно работать во время дизайна. Это может быть, например, класс, содержащий значения переменных среды (environment) или класс для работы с INI файлами.
- **TComponent** - Отправная точка для многих невидимых компонент. Данный класс обладает встроенной возможностью сохранять/считывать себя в потоке во время дизайна.
- **TGraphicControl** - Используйте этот класс для создания видимых компонент, которым не нужен handle. Такие компоненты рисуют прямо на своей поверхности и требуют мало ресурсов Windows.
- **TWinControl** - Базовый класс для компонент, которые имеют окно. Данное окно имеет свой handle, его используют при доступе к возможностям Windows через API.
- **TCustomControl** - Потомок TWinControl, вводит понятие канвы (Canvas) и метод Paint() для лучшего контроля за прорисовкой компонента. Именно этот класс используется в качестве базового для построения большинства видимых компонент, имеющих оконный handle.

• **TXxxxx** - Класс вроде TEdit или TButton. Используются с целью доопределения их свойств и методов или переопределения значения свойств, принимаемых по умолчанию.

Пример создания компонента

Для примера создадим новый класс, мутант TButton, в котором изменим значение по умолчанию свойства ShowHint на True и добавим новое свойство - счетчик нажатий на кнопку. Заготовка модуля для создания нового компонента уже есть (см. пункт **Заготовка для нового компонента**). Теперь исходный текст выглядит так:

```
unit New_btn;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TMyButton = class(TButton)
  private
    { Private declarations }
    FClickCount : Longint;
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
    procedure Click; override;
    property ClickCount : Longint read FClickCount write
                          FClickCount;
  published
    { Published declarations }
  end;

procedure Register;

implementation

constructor TMyButton.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  ShowHint:=True;
  FClickCount:=0;
end;

procedure TMyButton.Click;
begin
  Inc(FClickCount);
  inherited Click;
end;

procedure Register;
begin
```

```
  RegisterComponents('Samples', [TMyButton]);
end;
end.
```

Для того, чтобы переопределить начальное значение свойства при создании объекта, нужно переписать конструктор *Create,* в котором и присвоить этому свойству нужное значение (не забыв перед этим вызвать конструктор предка).

Новое свойство для подсчета нажатий на клавишу называется *ClickCount.* Его внутреннее поле для сохранения значения - FClickCount имеет тип Longint, емкости поля хватит надолго.

Задание:

1. Создать новый компонент от класса (согласно варианту) и для него определить новые действия:

1. TLabel
2. TEdit
3. TMemo
4. TButton
5. TCheckBox
6. TRadioButton
7. TListBox
8. TComboBox

Компонент TForm имеет свойство Canvas, представляющее собой область окна, в которую можно выводить различные графические изображения.

1.1 Класс TCanvas

Свойство Brush

Это свойство позволяет задать цвет и шаблон заполнения («кисть» при графических операциях). Это свойство имеет тип TBrush и представляет собой экземпляр этого класса. В данном примере показано, как изменить цвет кисти на красный:

```
Canvas.Brush.Color := clRed;
```

Свойство ClipRect

Доступно только для чтения. Задает координаты отсекающего прямоугольника. Результаты всех графических операций, выполняемых за пределами данной прямоугольной области, не отображаются.

Свойство CopyMode

Устанавливает режим копирования – то, как класс TCanvas будет воспроизводить графические изображения. По умолчанию используется значение cmSrcCopy, указывающее на то, что изображение просто копируется, замещая текущее изображение. С помощью изменения значений свойства CopyMode можно добиться различных визуальных эффектов.

В следующем примере показано, как скопировать графическое изображение с инвертированием:

```
with Canvas do
  begin
    CopyMode := cmNotSrcCopy;
    CopyRect(ClientRect, Canvas, ClientRect);
  end;
```

Свойство Font

Свойство Font является экземпляром класса TFont и позволяет указать параметры текста, отображаемого классом TCanvas. В данном примере показано, как изменить атрибуты шрифта:

```
with Canvas do
```

```
  begin
    Font.Color := clBlue;
    Font.Name  := 'Arial';
    Font.Size  := 20;
    Font.Style := [fsItalic];
    TextOut(20, 5, 'Hello!');
  end;
```

Свойство Pen

Свойство Pen позволяет задать тип «карандаша» - инструмента, используемого для отрисовки линий и границ геометрических фигур. Свойство Pen является экземпляром класса TPen. В приведенном примере показано, как отобразить на экране прямоугольник с рамкой шириной в 20 пикселов:

```
with Canvas do
  begin
    Pen.Width := 20;
    Rectangle(10, 10, 100, 100);
  end;
```

Свойство PenPos

Свойство PenPos задает текущую позицию «карандаша». Для изменения значения этого свойства используется метод MoveTo.

Свойство Pixels

Свойство Pixels представляет собой массив, содержащий цвета пикселов, находящихся в классе TCanvas. Этот массив является двухмерным. Для доступа к отдельным точка достаточно указать их X- и Y- координаты. Например, чтобы отобразить прямую линию, не прибегая к использованию метода LineTo, можно написать следующий код:

```
var
  i: word;

begin
  with Canvas do
    for i := 10 to 100 do
      begin
        Pixels[i, 10] := clBlue;
      end;
end;
```

1.2 Методы класса TCanvas

Метод Arc

С помощью этого метода отображается дуга эллипса. Центром эллипса считается центр охватывающего прямоугольника, левый верхний угол которого имеет координаты X1, Y1, а правый нижний угол – координаты X2, Y2. Дуга отображается из точки с координатами X3, Y3 в точку с координатами X4, Y4 против часовой стрелки. Если начальная и конечная точки имеют одинаковые координаты, то вместо дуги отображается полный эллипс (окружность в случае, если охватывающий прямоугольник является квадратом).

```
with Canvas do
  begin
    Pen.Width := 5;
    Arc(10, 20, 200, 100, 100, 100, 0, 0);
  end;
```

Метод CopyRect

С помощью метода CopyRect выполняется копирование фрагмента изображения. Ниже показано, как скопировать изображение с инвертированием:

```
with Canvas do
  begin
    CopyMode := cmNotSrcCopy;
    CopyRect(ClientRect, Canvas, ClientRect);
  end;
```

Метод Draw

Этот метод используется для отображения графического объекта типа TGraphic в точке, заданной координатами (X, Y).

Метод Ellipse

Метод Ellipse используется для отрисовки эллипса, заданного охватывающим прямоугольником. Координата левого верхнего угла прямоугольника задается парой (X1, Y1), а координата правого нижнего угла - (X2, Y2). Пример использования этого метода показан ниже.

```
with Canvas do
  begin
    Ellipse(10, 10, 100, 50);
  end;
```

Метод FillRect

С помощью метода FillRect выполняется заливка указанного прямоугольника цветом, который указан аналогом стандартной функции FillRect. Ниже показано, как использовать этот метод:

```
var
  rec: TRect;

begin
  rec := Rect(20, 20, 150, 150);
  with Canvas do
    begin
      Brush.Color := clRed;
      FillRect(rec);
    end;
end;
```

Метод FloodFill

Метод FloodFill используется для заполнения области цветом, заданным значением свойства Brush. Заполнение начинается с точки с координатами X, Y. Каким образом будет происходить заполнение, определяется значением параметра FillStyle. Если свойство FillStyle имеет значение fsBorder, то область, которая будет заполнена, ограничена цветом, заданным параметром Color. Если же свойство FillStyle имеет значение fsSurface, то область заполняется до тех пор, пока в ней присутствует цвет, заданный параметром Color. Обычно значение fsSurface используется для заполнения областей, имеющих многоцветную границу.

Метод LineTo

Метод LineTo используется для отрисовки прямых линий. Линия отображается из текущей точки (определяемой значением свойства PenPos) в точку с координатами X, Y. После отображения прямой линии свойство PenPos принимаем значение X, Y.

Метод MoveTo

Метод MoveTo изменяет координаты текущей точки, определяемой значением свойства PenPos. Пример использования этого метода показан ниже:

```
with Canvas do
  begin
    MoveTo(100, 100);
    LineTo(200, 100);
  end;
```

Метод Pie

Метод Pie используется для отрисовки сегмента эллипса, заданного охватывающим прямоугольником с координатами X1, Y1 и X2, Y2. Сегмент определяется двумя линиями, соединяющими центр эллипса с точками с координатами X3, Y3 и X4, Y4. Пример использования этого метода показан ниже.

```
with Canvas do
  begin
    Pie(10, 10, 200, 200, 60, 3, 200, 60);
  end;
```

Метод Polygon

Метод Polygon используется для отрисовки замкнутого многогранника, заданного массивом координат. Пример использования этого метода показан ниже.

```
with Canvas do
  begin
    Brush.Color := clRed;
    Polygon([Point( 10,  10),
             Point( 30,  10),
             Point(130,  30),
             Point(240, 120)]);
  end;
```

Метод PolyLine

Метод PolyLine используется для отрисовки незамкнутого многогранника, заданного массивом координат. Пример использования этого метода показан ниже.

```
with Canvas do
  begin
    Pen.Color := clBlack;
    PolyLine([Point(  5,   5),
              Point(100,  40),
              Point(150, 120),
              Point(140, 200),
              Point( 80, 100),
              Point(  5,   5)]);
  end;
```

Метод Rectangle

Метод Rectangle используется для отрисовки прямоугольника, заданного координатами его левого верхнего (X1, Y1) и правого нижнего углов (X2, Y2).

Метод TextOut

Метод TextOut выводит строку, переданную в параметре Text в точке, указанной координатами X, Y. Эта точка задает координаты левого верхнего угла прямоугольной области, в которой будет размещен текст. Например, чтобы вывести строку «Delphi» в точке с координатами (20, 20) можно написать следующий код:

```
Canvas.Font.Height := 128;
Canvas.Font.Name := 'Courier New Bold';
Canvas.TextOut(20, 20, 'Delphi');
```

Цветовые константы

clAqua	зеленовато-голубой
clBlack	черный
clBlue	синий
clFuchsia	светло-фиолетовый
clGray	серый
clGreen	зеленый
clLime	светло-зеленый
clMaroon	малиновый
clNavy	темно-синий
clOlive	желтовато-зеленый
clPurple	фиолетовый
clRed	красный
clSilver	светло-серый
clTeal	светло-синий
clWhite	белый
clYellow	желтый

Примечание: для получения любого другого цвета (из 16 миллионов возможных) используйте функцию RGB(R, G, B: Byte), в которой задается интенсивность трех компонент цвета – красной [red], зеленой [green] и синей [blue]. Каждая компонента может принимать значение от 0 до 255. В следующем примере функция RGB используется для отрисовки квадрата, цвет которого меняется от черного до ярко-красного:

```
var
  x: integer;
```

```
begin
  for x := 0 to 255 do
  with Canvas do
    begin
      Pen.Color := RGB(x, 0, 0);
      MoveTo(x, 0);
      LineTo(x, 255);
    end;
end;
```

Задания

1. Нарисовать линию синего цвета толщиной 8 пикселов. Начальную и конечную точку задать самостоятельно.
2. Нарисовать прямоугольник с зеленой границей толщиной 16 пикселов и заполненный желтым цветом. Вершины прямоугольника задать самостоятельно.
3. Вывести произвольный текст (например «Delphi») высотой 32 пиксела (шрифт «Courier New Bold») примерно на середину формы. Вокруг текста нарисовать эллипс.

Примечание: для решения этого задания нужно изучить свойства объекта Brush («Кисть»), который отвечает за заполнение простых геометрических фигур. См. встроенную справку Delphi.

4. Написать процедуру очистки формы.
5. Нарисовать квадрат от точки (0, 0) до (255, 255) используя свойства Canvas.Pixels[x, y]. Цвет точек определять по формуле (256*x + y).
6. Аналогично 5, только размер квадрата 512x512 и цвет каждой точки определять по формуле RGB(z, 0, 0), где z равно:
   а. sin(x*y/8192);
   б. sin(x/32)*cos(y/32);
   в. sin((x-256)*(y-256)/8192).

Можно использовать свою формулу.

Примечание: при использовании функций sin или cos результат будет принадлежать интервалу [-1, 1]. А параметры функции RGB должны находиться в интервале [0, 255].

Кроме того, необходимо округлить z до ближайшего целого, так как функция RGB работает только с целыми числами.

7. Усложненный вариант 6. Дополнительно определяются координаты (после вычисления z):

```
rx:=256+((x-y)*cos(pi/6))/2;
ry:=128+((x+y)*sin(pi/6) - z/4)/2.
```

И отображение точек на экране примет вид: Canvas.Pixels[rx, ry].

Лабораторная работа 21.
Построение ортогональной проекции трехмерного объекта.

Цель работы – построить ортогональную проекцию модели трехмерного объекта, научиться менять ракурс. Продемонстрировать это на созданной модели куба.

Задание:

1. *Получить изображение ортогональной проекции объекта (по вариантам):*
   1.1 Куб;
   1.2 Тетраэдр;
   1.3 Правильная усеченная 3-х угольная пирамида, боковые ребра равны стороне нижнего основания;
   1.4 Правильная 4-х угольная пирамида, все ребра равны;
   1.5 Октаэдр;
   1.6 «Звезда» (куб, на гранях которого построены пирамиды), все ребра равны;
   1.7 Прямоугольный параллелепипед (в основании прямоугольник и боковые ребра перпендикулярны основанию);
   1.8 Правильная 3-хгранная призма, все ребра равны.
   Примечание: номер варианта определить по формуле N = (M mod 8) + 1, где M – номер по журналу, mod – остаток от деления.
2. *Предусмотреть возможность смены ракурса (вращение с помощью клавиатуры или мышки).*
3. *Оформить отчет, который должен содержать:*
   3.1. Название и цель работы;
   3.2. Листинги ваших процедур.

Общие теоретические положения.

Рассмотрим способ получения перспективных изображений на основе аналитической геометрии. Все знакомы с представлениями точек в двухмерном и трехмерном пространствах их координатами *(X, Y)* и *(x, y, z)* соответственно. При необходимости получения перспективной проекции задается большое количество точек *P*(*x*, *y, z),* принадлежащих объекту, для которых предстоит вычислить координаты точек изображения *P′(X, Y)* на картинке. Для этого нужно только преобразовать координаты точки *P* из так называемых *мировых координат (x, y, z)* в *экранные координаты (X, Y)* ее центральной проекции *P′*. Будем предполагать, что экран расположен между объектом и глазом E. Для каждой точки *P* объекта прямая линия *PE* пересекает экран в точке *P′*.

Это отображение удобно выполнять в два этапа. Первый этап назовем *видовым преобразованием* - точка *P* остается на своем месте, но система мировых координат переходит в систему *видовых координат.* Второй этап называется *перспективным преобразованием.* Это точное преобразование точки *P* в точку *P′* объединенное с переходом из системы трехмерных видовых координат в систему двухмерных экранных координат:

Мировые координаты (x_w, y_w, z_w)

Видовое преобразование

Видовые координаты (x_e, y_e, z_e)

Перспективное преобразование

Экранные координаты (X, Y)

Для выполнения видовых преобразований должны быть заданы точка наблюдения, совпадающая с глазом, и объект. Будет удобно, если начало ее координат располагается где-то вблизи центра объекта, поскольку объект наблюдается в направлении от E к O. Пусть точка наблюдения E будет задана в сферических координатах ϕ, θ, ρ по отношению к мировым координатам. То есть мировые координаты (точки E) могут быть вычислены по формулам:

$$x_e = \rho \sin \phi \cos \theta$$
$$y_e = \rho \sin \phi \sin \theta$$
$$z_e = \rho \cos \phi$$

Обозначения сферических координат схематически изображены на рис. 1.

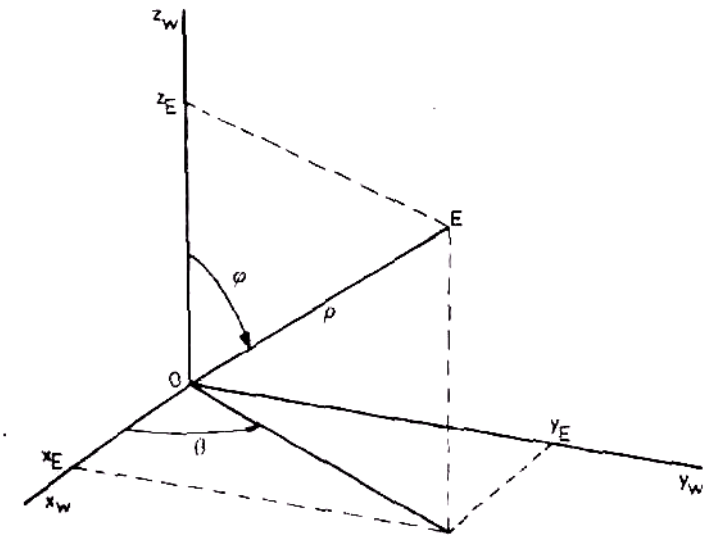


Рисунок 1 - *Сферические координаты точки наблюдения E*

Видовое преобразование может быть записано в форме

$$[x_e\ y_e\ z_e\ 1] = [x_w\ y_w\ z_w\ 1]V,$$

где V - матрица видового преобразования размерами 4x4.

Матрица V, полученная в процессе видового преобразования, выглядит следующим образом:

$$V = \begin{bmatrix} -\sin\theta & -\cos\varphi\cos\theta & -\sin\varphi\cos\theta & 0 \\ \cos\theta & -\cos\varphi\sin\theta & -\sin\varphi\sin\theta & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix}$$

Сейчас можно использовать видовые координаты x_e и y_e просто игнорируя координату z_e для получения *ортогональной проекции.* Каждая точка *P* объекта проецируется в точку *P′* проведением прямой линии из точки *P* перпендикулярно плоскости, определяемой осями x и y. Эту проекцию можно также считать перспективной картинкой, которая была бы получена при удалении точки наблюдения в бесконечность. Параллельные линии остаются параллельными и на картинке, полученной при ортогональном проецировании.

Для построения куба необходимо задать координаты его вершин в мировой системе координат, а затем получить из них видовые (и экранные для ортогональной проекции) координаты. Это можно сделать, описав координаты в массивах типа `T3DPoint` (запись, содержащая три координаты точки) и `TPoint` (запись, содержащая две координаты точки).

```
var
w: array [1..8] of T3DPoint;  //мировые (world) координаты вершин
v: array [1..8] of TPoint;          //видовые (view) координаты вершин
```

Для примера зададим координаты вершин куба с центром в начале координат:

```
  w[1].x := -50;
  w[1].y := -50;
  w[1].z := -50;

  w[2].x := 50;   w[2].y := -50;   w[2].z := -50;

  w[3].x := 50;   w[3].y := 50;   w[3].z := -50;

  w[4].x := -50;   w[4].y := 50;   w[4].z := -50;

  w[5].x := -50;   w[5].y := -50;   w[5].z := 50;

  w[6].x := 50;   w[6].y := -50;   w[6].z := 50;

  w[7].x := 50;   w[7].y := 50;   w[7].z := 50;

  w[8].x := -50;   w[8].y := 50;   w[8].z := 50;
```

Далее необходимо перевести координаты в видовую систему координат, воспользовавшись матрицей V. Данная процедура выглядит следующим образом:

```
procedure ViewTransformation;
for i := 1 to 8  Do
begin
      v[i].x := Round(
w[i].x*(-sin(teta)) + w[i].y*(cos(teta)))+Form1.ClientWidth div 2;
      v[i].y := Round(
w[i].x*(-cos(phi) * cos(teta)) - w[i].y*( cos(phi) * sin(teta)) +
w[i].z*(sin(phi))) + Form1.ClientHeight div 2;
end;
```

Процедура рисования граней куба состоит в соединении между собой соответствующих вершин ребрами. Строить куб необходимо по экранным координатам, которые сосчитаны в процедуре `ViewTransformation`. Пример:

```
procedure DrawCube;
Line(v[1],v[2]);
Line(v[2],v[3]);
```

```
Line(v[3],v[4]);
Line(v[4],v[1]);
Line(v[5],v[6]);
Line(v[6],v[7]);
Line(v[7],v[8]);
Line(v[8],v[5]);
Line(v[1],v[5]);
Line(v[2],v[6]);
Line(v[3],v[7]);
Line(v[4],v[8]);
```

На этом этапе (если вы написали процедуру `line`), вы можете, задав начальные углы, вызвать процедуры `ViewTransformation` и `DrawCube`, например, из метода `OnCreate`, чтобы нарисовать куб.

Анимация

Для того, чтобы добиться анимации необходимо выполнить следующие шаги:

- Нарисовать объект в одном месте
- Стереть объект
- Рассчитать новые координаты объекта
- Нарисовать объект в новом месте

Обработка нажатия клавиши

Получить скан-код клавиши, нажатой пользователем, а также состояние кнопки `Shift` позволяет процедура `FormKeyDown (Sender: TObject; var Key: Word; Shift: TShiftState)`.

Здесь в переменной `Key` передаётся код клавиши.

Получив скан-код, мы должны сравнить его с кодами клавиш, которые мы хотим обрабатывать (коды стрелок: 38, 40 и 37, 39). Если таких сравнений больше двух, удобно воспользоваться оператором `case.`

Теперь в зависимости от нажатой стрелки можно изменять углы `phi` и `teta`.

Не забывайте перед изменением угла стереть старый рисунок. Это можно сделать с помощью закрашивания цветом фона целиком всей формы или прорисовки такого же кубика по линиям.

Если изображение на экране не соответствует вашим ожиданиям, убедитесь, что в процедуре `FormKeyDown` присутствуют все пункты алгоритма анимации.

Обработка сообщений мыши

Если вы хотите изменять положение объекта с помощью мыши, вы можете обрабатывать сообщения `OnL(R)ButtonDown` (нажатие на левую (или правую) кнопку мыши) или сообщение `OnMouseMove` (движение мыши, в этом случае можно проверять, в какую сторону происходит движение и в эту сторону вращать фигуру).

4 семестр

Лабораторная работа № 1.

Тема: Использование Mathcad в качестве суперкалькулятора.

Пример 1. Для набора выражения используем обычную математическую нотацию: $\frac{1}{\sqrt{2}} = 0.707$

$1/\sqrt{2}=$. Знак квадратного корня мы найдем, раскрыв арифметическую палитру. В конце выражения поставим знак равенства "="

Пример 2. Можно присвоить значения переменным: *a:*=1 *b*:=2.35 *p*:=π

Ввод заканчивается клавишей Enter или щелчком мыши вне определения.

Здесь мы обозначили переменные буквами: *a*, *b, p;* но можно использовать произвольный набор символов для обозначения переменных. Имена переменных чувствительны к регистру. Вначале вводится имя переменной, затем символ ":" (или знак =), затем число или выражение (в частности, мы использовали предопределенную константу π из арифметической пары, Ctrl+p). Синий уголок показывает текущий операнд выражения, он может быть расширен клавишей **"Пробел".** *Обратите внимание, что в качестве разделителя целой и дробной части числа используется точка*. Теперь этими переменными можно пользоваться при арифметических вычислениях.

:= это оператор присваивания, = это команда «Вывести значение».

$\frac{a+b}{2} = 1.675$ $c := \frac{a-b}{2}$ $d := \sin\left(\frac{p}{2}\right)$ c=-0,675 d=1

Сейчас видна разница в использовании оператора присваивания := и знака =.

Пример 3. Вычислите для x=1 следующие функции:

$y := \frac{1}{\sqrt[3]{x^2}+\sqrt[6]{x^5}}$ $y := \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ $y := \frac{\operatorname{atan}(x)^2}{2}$. Функция arctg(x) обозначена здесь как atan(x). Набирается «вручную».

Вычислите значения при x=3, а также при x=5.

Для корректного ввода формул необходимо пользоваться арифметической палитрой и кнопкой **«Вставить функцию»** и копировать формулы, используя кнопки панели инструментов.

Правила видимости: значение переменной доступно правее и ниже ее определения.

Глобальные переменные доступны везде на рабочем листе и вводятся знаком ~, например, введем N ~100, получим N≡100

Если вы хотите изменить количество знаков результата вычислений после десятичной точки, это можно сделать в меню **Format\Number…\Displayed Precision(3)** или просто дважды щелкнуть мышкой по выражению, после чего заменить 3 на 6.

Установим, например, для значения выражения 6 значащих цифр: $\frac{a}{b} = 0.425532$

Для ввода текстового комментария необходимо ввести знак двойной кавычки “, затем вводить текст. При достижении конца строки происходит автоматический перенос на следующую. Текстовая область, как и любая другая, может быть перемещена на рабочем листе или скопирована в буфер. Маркеры текстовой области позволяют менять ее размеры.

Упражнения для самостоятельной работы. Вычислите при x=2:

$$y1 := \frac{2.087 \cdot x^3 + 3.24 \cdot \sqrt[3]{x}}{1 + \sqrt{x}} \qquad y2 := \frac{1}{\pi} \cdot \cos\left(\frac{x}{\pi}\right)^2 - \frac{1}{2 \cdot \pi} \cdot \sin\left(\frac{x}{2 \cdot \pi}\right) \qquad y3 := \frac{\sqrt{1 - \sin(a \cdot x)^2}}{b - p \cdot \tan(x)}$$

Числовые массивы. Матрицы.

Матрица – прямоугольная таблица чисел (массив). Поэтому будем понимать эти термины как синонимы. В Mathcad реализованы одно- и двумерные матрицы, причем одномерные матрицы – это просто массивы у которых один столбец. Создаются матрицы при помощи кнопки палитры инструментов или команды **Insert\Matrix…**, где указывается количество строк, столбцов **Rows и Columns**. Для примера создадим матрицу размером 3*3 и 3*1:

$$A := \begin{pmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{pmatrix} \qquad A := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Матрицу с одним столбцом называют вектор-столбец.

Принято обозначать матрицы большими латинскими буквами.

С матрицами можно проделывать множество операций, имеется даже специальная матричная алгебра, но мы ограничимся лишь обычными операциями с массивом чисел:

1. *Умножение матрицы на число*. G:=2·A $G = \begin{pmatrix} 2 & 4 & 6 \\ 16 & 18 & 8 \\ 14 & 12 & 10 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \cdot 0.5 = \begin{pmatrix} 0.5 \\ 1 \\ 1.5 \end{pmatrix}$.

2. *Сложение матриц*. $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 1 \\ 1.5 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 3 \\ 4.5 \end{pmatrix}$ C:=A+2·A $C = \begin{pmatrix} 3 & 6 & 9 \\ 24 & 27 & 12 \\ 21 & 18 & 15 \end{pmatrix}$.

Примечание. Совершенно очевидно, что в операциях сложения размеры матриц должны совпадать.

Доступ к элементам матриц.

Имея дело с массивами чисел неплохо было бы иметь возможность извлечения отдельного числа из матрицы. Для этого реализован механизм индексирования. Так в одномерной матрице (вектор-столбец) все значения пронумерованы от 0 до n-1, где n количество значений. Обращение к элементу массива производится по индексу. Например, в матрице B три значения с индексами 0, 1, 2 и обращение к ним производится , как к переменной с индексом:

B_0=1 B_1=2 B_2=3 Индекс вводится символом квадратной скобки [- B[0, B[1, B[2 или из арифметической палитры.

Примечание. *Переменная с индексом может присутствовать в арифметическом выражении наряду с другими переменными.*

Обращение к двумерному массиву производится аналогично, только приходится указывать два индекса через запятую: первый индекс – это номер строки, второй – номер столбца. Как и ранее нумерация начинается с 0. Например:

$A_{0,0}=1$ $A_{0,2}=3$ $A_{2,0}=7$ $A_{2,2}=5$

Упражнения для самостоятельной работы. Введите матрицы размером 2*2, 3*3, 4*4, 1*3 с произвольным набором чисел.

Что касается последней матрицы размером 1*3, то хотя и одномерная матрица – вектор-строка, доступ к ее элементам возможен только как к двумерной, где первый индекс равен 0, например:

Z:= (1 2 3) $Z_{0,0}=1$ $Z_{0,1}=2$ $Z_{0,2}=3$

Примечание. Элементами матрицы могут быть и арифметические выражения.

Лабораторная работа № 2.

Тема: Функции. Интервальная переменная.

Определить функцию в Mathcad достаточно просто, для этого необходимо ввести имя функции, в скобках её параметры и оператор присваивания. После чего вводится арифметическое выражение. Затем функция может использоваться наравне с встроенными функциями. Введем для примера параболическую функцию:

***a*:=1 *b*:=1 *c*:=-1** Нам пришлось предварительно описать три константы *a*, *b*, *c*, $f(x) := a \cdot x^2 + b \cdot x + c$ иначе функция не может быть вычислена.

Теперь, для того чтобы получить значение функции, достаточно записать:

***f(0)*=-1** ***f*(1.5)=2.75** и так далее.

Однако, если бы этим ограничивались возможности Mathcad, то этот пакет так бы и остался большим калькулятором. К счастью был предусмотрен механизм многократного повторения вычислений, нечто подобное циклам в языках программирования. Введено понятие интервальной переменной в формате:

var*:=начальное значение[, начальное значение+шаг]..конечное значение*

в скобках указан необязательный параметр шаг, по умолчанию равный 1.

Двоеточие ".." вводится клавишей точка с запятой ":" или кнопкой арифметической палитры

Введем для примера интервал изменения аргумента x на отрезке **[-2;2]** с шагом **h=0.1**. $x := -2, -1.9..2$

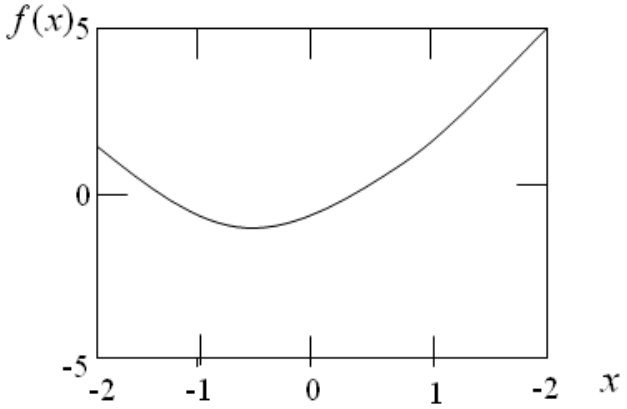
Фактически мы получаем набор из 41 значения аргумента. Чтобы убедиться в этом, достаточно ввести x=

Для того чтобы вывести таблицу значений функции, введите ***f*(x)** и знак "=", вы получите значения функции.

Таким образом можно увидеть только первые 50 значений.

Построим график, воспользовавшись уемся графической палитрой, раскрыв которую выберем **x-y график**.

График строится довольно просто, нужно только указать x переменную в маркере оси x и функцию *f(x)* в маркере оси y. Заканчивается построение клавишей Enter или щелчком мыши вне графика.



Можно также явно указать начальное и конечное значение по осям в маркерах начала и конца оси, иначе они определятся автоматически.
Выделив график двойным щелчком мыши, можно произвести его настройку, в частности, определить тип, цвет и толщину линии, а также выбрать оси.
<u>Упражнение для самостоятельной работы</u>. Постройте графики функций:

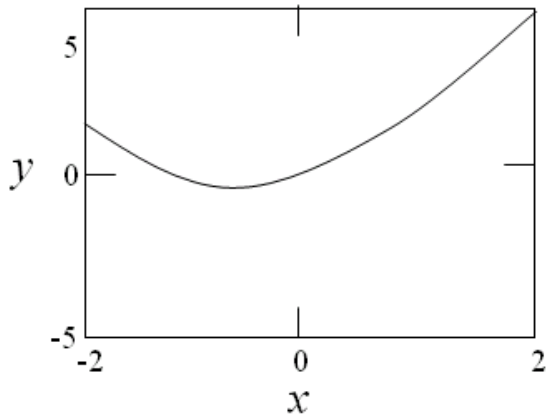
$u(x) := x \cdot \sin(x)$ $v(x) := \sqrt{1 + \sin(x)^2}$ $w(x) := e^{-x^2}$ $z(x) := \log(1 + \sqrt{x - 1})$

Диапазон для изменения аргумента по умолчанию будет **[-10;10].** Шаг принять равным **0,2.**
<u>Примечание</u>: на одном x-y графике можно построить до 16 кривых. *Функции вводятся через **запятую***.

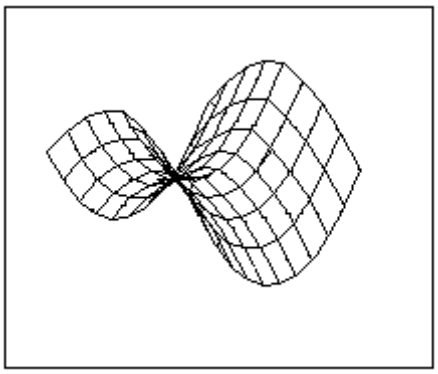
Матрицы и переменные с индексом

К сожалению, использование интервальной переменной для построения функции вызывает определенные трудности. Это связано с тем, что при каждом обращении к интервальной переменной система производит многократные вычисления и, зачастую, совершенно напрасно пересчитывает одни и те же значения. Чтобы избежать лишней работы проще всего вычислить таблицу значений функции один раз, а затем обращаться к табличным значениям.

Для реализации данной задачи используется интервальная переменная и переменная с индексом. Например, если вернуться к предыдущей задаче, можно решить её следующим образом:
введем индекс, как интервальную переменную **i:=0..40**
введем переменную с индексом, которая будет меняться от -2 до 2 с шагом 0.1, для описания переменной **x_i:=-2+0.1·i**
Если вычислить **y_i:=*f*(x_i)** мы получим вектор-столбец значений функции.
Введите x= и y=, получите эти матрицы.
При помощи переменной с индексом мы создали две одномерные матрицы с одним столбцом и 41 строкой. Теперь можно вновь построить график, где в качестве функции и аргумента мы и укажем эти переменные с индексом.



Аналогично можно использовать двумерную матрицу для построения функции двух переменных, например, определим функцию:
f(x,y):=x²-y² и две интервальных переменных ***i*:=0..10** ***j*:=0..10**
Теперь определяем две переменные с индексом x_i***:=-5+i*** y_j***:=-5+j***
Определим двумерную матрицу: $M_{i,j}$***:=f(x_i,y_j)*** и построим поверхность, в качестве единственного аргумента графика указываем имя матрицы *M*.



M

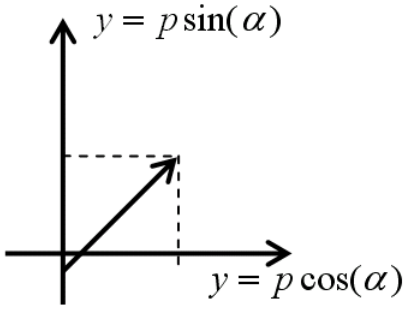
M=

	0	1	2	3	4	5	6	7	8	9
0	0	9	16	21	24	25	24	21	16	9
1	-9	0	7	12	15	16	15	12	7	0
2	-16	-7	0	5	8	9	8	5	0	-7
3	-21	-12	-5	0	3	4	3	0	-5	-12
4	-24	-15	-8	-3	0	1	0	-3	-8	-15
5	-25	-16	-9	-4	-1	0	-1	-4	-9	-16
6	-24	-15	-8	-3	0	1	0	-3	-8	-15
7	-21	-12	-5	0	3	4	3	0	-5	-12
8	-16	-7	0	5	8	9	8	5	0	-7
9	-9	0	7	12	15	16	15	12	7	0
10	0	9	16	21	24	25	24	21	16	9

Лабораторная работа № 3.

Тема: Графики. Вычисление сумм и произведений.

В Mathcad, кроме x-y – графика можно построить и график в полярных координатах, что иногда оказывается очень удобным.
Связь полярных и декартовых координат показана на картинке.



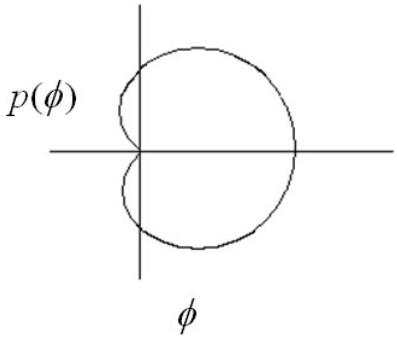
Построим полярный график, выбрав в качестве кривой кардиоиду.

$p(\phi) := 1 + \cos(\phi)$ $\phi := 0,0.1..2 \cdot \pi$

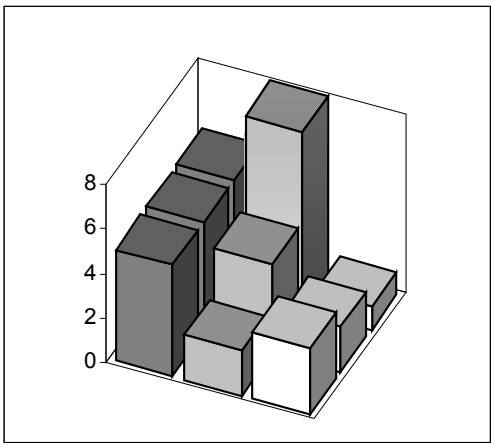
Для ввода *f, r* используем палитру греческих символов.

Постройте самостоятельно графики функций:

$p1(\phi) := \cos(\phi) \cdot \sin(\phi)$ $p2(\phi) := \phi$ $p3(\phi) := e^{\phi}$



Для построения столбиковой диаграммы необходимо задать матрицу значений $D := \begin{pmatrix} 5 & 8 & 1 \\ 5 & 4 & 2 \\ 5 & 2 & 3 \end{pmatrix}$



Как видно из графика, каждая колонка матрицы создает ряд значений.

Постройте столбиковые диаграммы для произвольного набора чисел, например: $M := \begin{pmatrix} 2 & 0 \\ 8 & 3 \\ 15 & 5 \end{pmatrix}$

Примечание. Просмотрите примеры в электронных книгах центра ресурсов.

Вычисление сумм и произведений

Воспользуемся палитрой вычислений  .

Например: $\sum_{n=1}^{100} \frac{1}{n^2} = 1.635$ $\sum_{n=1}^{20} \frac{(-1)^n}{(2 \cdot n)!} = 0.54$ $\sum_{k=1}^{1000} \frac{1}{(2 \cdot k - 1) \cdot (2 \cdot k + 1)} = 0.5$

По определению, n!=1*2*…*n , т.е. произведение всех чисел до n включительно. Здесь мы использовали значок суммы с указанием границ суммирования. Значок суммирования только с указанием индекса используется в тех случаях, когда пределы изменения индекса заданы переменной интервального типа.

Например: $X := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ i:=0..2 $\sum_i X_i = 6$

n:=1..100 $u(n):=n^2+2\cdot n+1$ $\sum_n \frac{1}{u(n)} = 0.635$

Аналогично вычисляются произведения. По определению: $\prod_{i=1}^{n} (a_i = a_1 \cdot a_2 \cdot ...a_n)$

Например:

$\prod_{k=2}^{10000}\left(1-\frac{1}{k^2}\right) = 0.5$ $\prod_{n=1}^{10000}\left[1+\frac{(-1)^{n+1}}{2\cdot n-1}\right] = 1.414$ x:=0.5 $\prod_{k=0}^{1000}\left(1+x^{2^k}\right) = 2$

Вычислим для матриц M и D, которые мы использовали для построения объемных графиков, сумму и произведение.

i:=0..2 $\sum_i M_{i,0} = 25$ $\prod_i M_{i,1} = 0$ $\sum_i D_{i,i} = 12$ $\prod_i D_{i,i} = 60$

Символьные вычисления

Суммы и произведения можно вычислять в символьном виде, для этого, вместо оператора “=”, необходимо воспользоваться кнопкой математической палитры, например:

$\sum_{n=1}^{\infty} \frac{1}{n^2} \rightarrow \frac{1}{6}\cdot\pi^2$ $\sum_{n=1}^{\infty} \frac{1}{e^n} \rightarrow \frac{1}{(-1+\exp(1))}$

$\sum_{n=0}^{\infty} \frac{(-1)^n\cdot z^{2\cdot n}}{(2\cdot n)!} \rightarrow \cos(z)$ $\sum_{n=0}^{\infty} \frac{(-1)^n\cdot z^{2\cdot n+1}}{(2\cdot n+1)!} \rightarrow \sin(z)$

$\sum_{n=0}^{7} \frac{(-1)^n\cdot z^{2\cdot n}}{(2\cdot n)!} \rightarrow 1-\frac{1}{2}\cdot z^2+\frac{1}{24}\cdot z^4-\frac{1}{720}\cdot z^6+\frac{1}{40320}\cdot z^8-\frac{1}{3628800}\cdot z^{10}-\frac{1}{479001600}\cdot z^{12}-\frac{1}{871782912}$

Получаем просто ряд из нескольких слагаемых. Это значит, система не смогла упростить выражение.

Лабораторная работа № 4.

Тема: Операции с матрицами.

При решении задач линейной алгебры часто приходится проводить операции с матрицами. При работе с матрицами необходимо использовать панель "Векторы и матрицы" (Vector and Matrix Toolbar). Кроме того, часто используются следующие встроенные функции:

matrix(m,n,f) – создает и заполняет матрицу размерности m*n , элемент которой, расположенный в i-той строке, j-том столбце, равен значению f(i,j) функции f(x,y);

diag(v) - создает диагональную матрицу , элементы главной диагонали которой хранятся в векторе v;

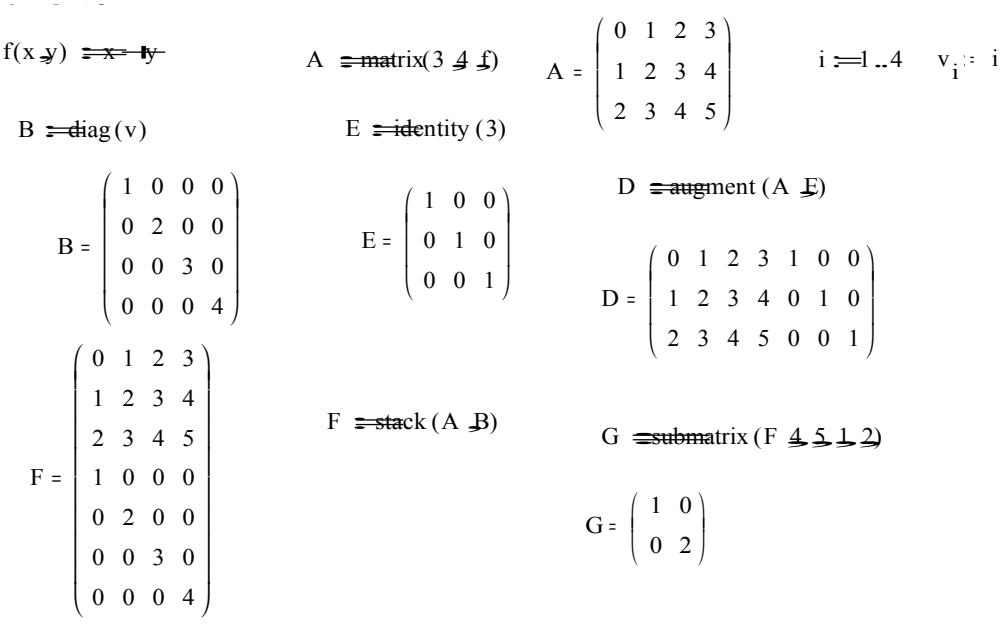
identity(n) - создает единичную матрицу порядка n;

augment(A,B) - к матрице A приписывает справа матрицу B (матрицы A и B должны иметь одинаковое число строк);

stack(A,B) - к матрице A приписывает снизу матрицу B (матрицы A и B должны иметь одинаковое число столбцов);

submatrix(A,ir,jr,ic,jc) - выделяет из матрицы A подматрицу, расположенную в строках с ir по jr и в столбцах с ic по jc;

При работе с матрицами обычно принято нумеровать строки и столбцы, начиная с 1 (а с MathCad-е нумерация по умолчанию начинается с нуля, поэтому рекомендуется для удобства восприятия матричных вычислений выполнить команду: **ORIGIN:=1**



Для вычисления различных числовых характеристик матриц используются следующие функции:

last(v) – вычисление номера последней компоненты вектора v;

length(v) – вычисление количества компонент вектора v;

rows(A) – вычисление числа строк в матрице A;

cols(A) – вычисление числа столбцов в матрице A;

max(A) – вычисление максимального элемента в матрице A;

min(A) – вычисление минимального элемента в матрице A;

tr(A) – вычисление следа квадратной матрицы A (т.е. суммы ее диагональных элементов);

rank(A) – вычисление ранга матрицы A;

norm1(A) – первая норма матрицы $\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{i,j}|$ (т.е. максимальная сумма модулей чисел по столбцам матрицы);

norm2(A) – вторая норма матрицы $||A||_2 = \sqrt{\lambda_{max}(A \cdot A^T)}$, где $\lambda_{max}(A \cdot A^T)$ - максимальное собственное значение матрицы $A \cdot A^T$.

norme(A) – евклидова норма матрицы $||A||_e = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{i,j}|^2}$

normi(A) – $\max_i \sum_{j=1}^{n} |a_{i,j}|$, (т.е. максимальная сумма модулей чисел по строкам матрицы);

Выполните следующие действия:

last(V) = 4	length(V) = 4		
rows(D) = 3	cols(D) = 7	max((D) = 5	tr(B) = 10 rank(A) = 2
norm1(B) = 4	norm2(B) = 4	normi(B) = 4	norme(B) = 5.477

При численном решении задач линейной алгебры используются следующие **функции:**

rref(A) – приведение матрицы к ступенчатому виду с единичным базисным минором (выполняются элементарные операции над строками матрицы);
eigenvals(A) – вычисление собственных значений квадратной матрицы A:
eigenvecs(A) – вычисление собственных векторов квадратной матрицы A: значением функции является матрица, столбцы которой есть собственные векторы матрицы A (порядок их следования отвечает порядку следования собственных значений, вычисленных функцией **eigenvals(A)**;
eigenvecs(A,I) – вычисление того собственного вектора матрицы A, который соответствует I – тому собственному значению матрицы;
lsolve(A,b) – решение системы линейных алгебраических уравнений Ax = b

$$C := \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 1 \end{pmatrix} \qquad \text{eigenvals}(C) = \begin{pmatrix} 0.271 \\ -2.71 \end{pmatrix} \qquad L = \begin{pmatrix} 0.271 \\ -2.71 \end{pmatrix}$$

$$\text{eigenvecs}(C) = \begin{pmatrix} 0.321 & -0.611 & -0.432 \\ 0.711 & 0.222 & 0.801 \end{pmatrix}$$

$$\text{eigenvec}(C, L_2) = \begin{pmatrix} 0.611 \\ -0.222 \end{pmatrix} \qquad b := \begin{pmatrix} 8 \\ 14 \end{pmatrix} \qquad X = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

Для транспонирования матрицы (т.е. перестановки строк со столбцами, когда первая строка становится первым столбцом, вторая - вторым столбцом и т.д.) можно воспользоваться панелью "Матрица" ("Matrix") , выбрав значок $\mathbf{M^T}$. Чтобы вычислить определитель матрицы, можно воспользоваться на этой панели значком **|x|**. Для вычисления обратной матрицы необходимо применить значок $\mathbf{x^{-1}}$ (при этом следует помнить, что обратная матрица вычисляется только для квадратной матрицы с ненулевым определителем).

$|C| = -4$ $\quad C^{-1} = \begin{pmatrix} -1.5 & 2 & 0.5 \\ 0.75 & -0.5 & -0.25 \end{pmatrix}$ $\quad C^{T} = \begin{pmatrix} 2 & 1 & 4 \\ 3 & 2 & 1 \end{pmatrix}$

Решение системы линейных уравнений методом Крамера

При решении задач линейной алгебры для вычисления неизвестных в системе линейных алгебраических уравнений можно воспользоваться методом Крамере, основанном на вычислении определителя. Если определитель $\Delta = \det A$ матрицы системы уравнений $Ax=b$ отличен от нуля, тосистема имеет единственное решение $x_1, x_2, x_3, \ldots x_n$, определяемое по формулам Крамера $x_i = \Delta_i/\Delta$, где $\tilde{\Delta}_i$определитель матрицы n -ного порядка, полученной из матрицы системы заменой i-го столбца столбцом правых частей **b**.

Пример. Решение СЛАУ методом Крамера.

$$A := \begin{pmatrix} -1 & 2 & -3 & 4 \\ 0 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad b := \begin{pmatrix} 10 \\ 3 \\ 10 \end{pmatrix} \qquad \Delta := |A|$$

Определитель отличен от нуля, следовательно, система имеет единственное решение

$$\Delta 1 := \left|\begin{pmatrix} 30 & 2 & 3 & 4 \\ 10 & 2 & -3 & 4 \\ 3 & 1 & -1 & 1 \\ 10 & 1 & 1 & 1 \end{pmatrix}\right| \quad \Delta 2 := \left|\begin{pmatrix} 1 & 30 & 3 & 4 \\ -1 & 10 & -3 & 4 \\ 0 & 3 & -1 & 1 \\ 1 & 10 & 1 & 1 \end{pmatrix}\right| \quad \Delta 3 := \left|\begin{pmatrix} 1 & 2 & 30 & 4 \\ -1 & 2 & 10 & 4 \\ 0 & 1 & 3 & 1 \\ 1 & 1 & 10 & 1 \end{pmatrix}\right|$$

$$\Delta 4 := \left|\begin{pmatrix} 1 & 2 & 3 & 30 \\ -1 & 2 & -3 & 10 \\ 0 & 1 & -1 & 3 \\ 1 & 1 & 1 & 10 \end{pmatrix}\right| \quad \Delta 1 = -4 \quad \Delta 2 = -8 \quad \Delta 3 = -12 \quad \Delta 4 = -16$$

$$x1 := \frac{\Delta 1}{\Delta} \quad x2 := \frac{\Delta 2}{\Delta} \quad x3 := \frac{\Delta 3}{\Delta} \quad x4 := \frac{\Delta 4}{\Delta}$$

$x1 = 1 \quad x2 = 2 \quad x3 = 3 \quad x4 = 4$

Эту задачу можно решить другим способом, используя понятие обратной матрицы:

$x := A^{-1} b \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$

Для сравнения найдем решение системы Ax=b с использованием функции

lsolve(A,b) : $\qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$

Решение СЛАУ методом Гаусса

Метод Гаусса (или метод Гауссовых исключений) состоит в том, что систему уравнений сначала приводят к ступенчатому виду (прямой ход метода Гаусса), а затем находят неизвестные X (обратный ход). В MathCad прямой и обратный ходы метода Гаусса выполняет функция **rref(Ar)**. При этом матрица **Ar** представляет собой расширенную матрицу - к матрице коэффициентов при неизвестных **A** добавлен справа столбец свободных членов **b:**

$$Ag = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 & 4 \end{pmatrix} \quad Ar = \begin{pmatrix} -1 & 2 & -3 & 4 & 10 \\ 0 & 1 & -1 & 1 & 3 \\ 1 & 1 & 1 & 1 & 10 \end{pmatrix} \quad x = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

Как видно из решения, функция **rref(Ar)** привела систему уравнений к ступенчатому виду с единичной матрицей в первых столбцах. Последний столбец полученной матрицы и есть искомое решение. Для того, чтобы его найти, воспользуемся функцией **submatrix**:

Выполните самостоятельно.

Исследуйте и, если решение существует, найдите по формулам Крамера, Гаусса и функции **lsolve** решение системы Ax=b

$$A := \begin{pmatrix} -0.090 & -0.033 & 0.0067 & -0.098 \\ 0.150 & 0.033 & 0.050 & 0 \\ 2.857 & 0.100 & -0.300 & 0.025 \end{pmatrix} \quad b := \begin{pmatrix} -0.098 \\ 0.183 \\ -0.041 \end{pmatrix}$$

Лабораторная работа № 5.
Тема: Дополнительные функции MathCad.

Числовые функции с условиями сравнения:

ceil(x) – наименьшее целое, большее или равное **x**

floor(x) – наибольшее целое, меньшее или равное **x**

mod(x,y) – остаток от деления **x/y** со знаком **x**

Функция условных выражений **if**

if (Условие, Выражение 1, Выражение 2) – если в этой функции условие выполняется, то будет вычисляться выражение1, в противном случае – выражение 2.

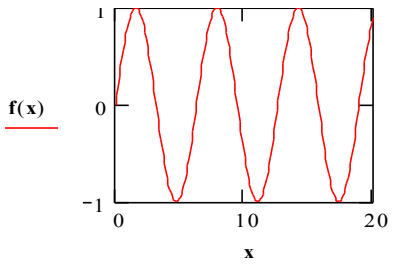
Пример.

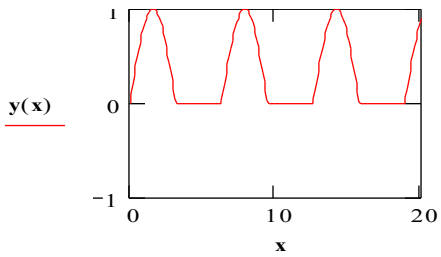
Синусоидальный сигнал

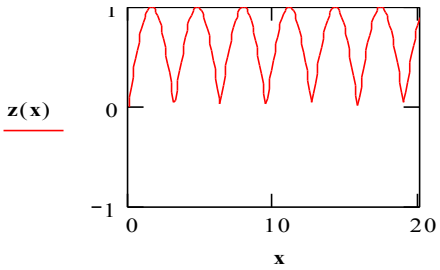
Сигнал однополупериодного выпрямления

Сигнал двухполупериодного выпрямления

Графики функций







Дополнительные матричные функции

lu(M) - выполняет треугольное разложение матрицы M: L и U -соответственно нижняя и верхняя треугольная матрицы.

qr(M) - выполняет ортогональное разложение матрицы **M: q** и **r** - соответственно ортогональная и верхняя треугольная матрицы

$$\mathbf{a} := \begin{pmatrix} 6 & 5 & 3 \\ 9 & 7 & 0 \end{pmatrix} \qquad \mathbf{lu}(\mathbf{a}) = \begin{pmatrix} 1 & 0 & 0 & 0.333 & 1 & 0 & 0 & 1.333 & 0 \\ 0 & 0 & 1 & 1.5 & -0.375 & 1 & 0 & 0 & -4.5 \end{pmatrix}$$

$$\mathbf{l} = \begin{pmatrix} 0.333 & 1 & 0 \\ 1.5 & -0.375 & 1 \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} 0 & 1.333 & 0 \\ 0 & 0 & -4.5 \end{pmatrix}$$

Проверка: $\mathbf{l} \cdot \mathbf{u} = \begin{pmatrix} 2 & 3 & 1 \\ 9 & 7 & 0 \end{pmatrix}$ $\qquad \mathbf{qr}(\mathbf{a}) = \begin{pmatrix} 0.545 & -0.064 & -0.836 & 0 & -1.414 & -1.157 \\ 0.818 & 0.257 & 0.514 & 0 & 0 & -2.314 \end{pmatrix}$

$$\mathbf{q} = \begin{pmatrix} 0.545 & -0.064 & -0.836 \\ 0.818 & 0.257 & 0.514 \end{pmatrix} \qquad \mathbf{r} = \begin{pmatrix} 0 & -1.414 & -1.157 \\ 0 & 0 & -2.314 \end{pmatrix}$$

$$\mathbf{q} \cdot \mathbf{r} = \begin{pmatrix} 6 & 5 & 3 \\ 9 & 7 & 0 \end{pmatrix}$$

Проверка:

Функции сортировки для векторов и матриц

sort(v) – сортировка элементов векторов в порядке возрастания их значений

reverse(v) – перестановка элементов (после sort) в обратном порядке
csort(M,n) – перестановка строк матрицы M таким образом, чтобы отсортированным оказался n-ный столбец;
rsort(M,n) – перестановка столбцов матрицы M таким образом, чтобы отсортированной оказалась n - ная строка.

$$\mathbf{V} := \begin{pmatrix} 2 \\ 4 \\ 6 \\ 1 \end{pmatrix} \qquad \mathbf{VS} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 6 \end{pmatrix} \qquad \mathbf{VR} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}$$

Исходная матрица Сортировка по первому столбцу и по первой строке

$$\mathbf{M} := \begin{pmatrix} 3 & 15 & 27 \\ 17 & 2 & 13 \end{pmatrix} \qquad \mathbf{csort}(\mathbf{M}, 1) = \begin{pmatrix} 1 & 12 & 4 \\ 3 & 15 & 27 \end{pmatrix} \qquad \mathbf{rsort}(\mathbf{M}, 1) = \begin{pmatrix} 3 & 15 & 27 \\ 17 & 2 & 13 \end{pmatrix}$$

Решение систем линейных алгебраических уравнений методом простых итераций
Исходная системы уравнений Cx=d преобразуется к виду:
X=b + Ax и ее решение вычисляется как предел последовательности
$X^{(k)} = b + Ax^{(k-1)}$, k=1,2,….
Преобразовать систему Cx=d к виду x=b+Ax можно, выделив диагональные элементы:

$$x_i = \frac{1}{c_{i,i}}(d_i - \sum_{i \neq j} c_{i,j} \cdot x_j)$$

В качестве условия окончания итерационного процесса можно взять условие $\frac{\left\| x^{(k)} - x^{(k-1)} \right\|}{\left\| x^{(k)} \right\|} \leq \varepsilon$, где eps – заданная погрешность приближенного решения

Пример. Решить методом простых итераций систему уравнений:

$100x_1 + 6x_2 - 2x_3 = 200$
$6x_1 + 200x_2 - 10x_3 = 600$
$x_1 + 2x_2 - 100x_3 = 500$

Преобразованная система имеет вид:
$x_1 = 2 - 0.06x_2 + 0.02x_3$
$x_2 = 3 - 0.03x_1 + 0.05x_3$
$x_3 = 5 - 0.01x_1 - 0.02x_2$

$$b := \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} \qquad A := \begin{pmatrix} 0 & 0.06 & - 0.02 \\ - 0.03 & 0 & 0.05 \\ - 0.01 & - 0.02 & 0 \end{pmatrix}$$

Для сходимости метода простых итераций достаточно, чтобы выполнялось условие ||A||<1 по какой-либо норме матрицы A:

$x^{(0)} := b \qquad x^{(k)} := b + A \cdot x^{(k-1)}$

$x =$

	0	1	2	3	4	5	6	7	8	9
0	2	1.92	1.907	1.907	1.907	1.907	1.907	1.907	1.907	1.907
1	3	3.19	3.188	3.189	3.189	3.189	3.189	3.189	3.189	3.189
2	5	4.92	4.917	4.917	4.917	4.917	4.917	4.917	4.917	4.917

$$\varepsilon := \frac{\left| x^{\langle 10 \rangle} - x^{\langle 9 \rangle} \right|}{\left| x^{\langle 9 \rangle} \right|} \qquad \varepsilon = 8.662 \times 10^{-}$$

Решите самостоятельно

Методом простых итераций решить СЛАУ:
$50x_1+25x_2-10x_3=40$
$12x_1+40x_2+10x_3=50$
$36x_1-15x_2+60x_3=120$

Решить эту систему уравнений любым из точных методов и сравнить результаты

Произвести qr - и lu-разложения матрицы коэффициентов при неизвестных. Убедиться, что матрица q – ортогональная.

Построить графики однополупериодного и двухполупериодного выпрямителей для функции
Y=2+3sin(x) (синусоида с постоянной составляющей)

Лабораторная работа № 6.

Тема: Решение переопределенных систем.

Очень часто при решении практических задач приходится решать систему алгебраических линейных уравнений вида **Ax=b**, когда число уравнений (т.е. строчек в матрице A) превышает число неизвестных в векторе - столбце x (или, что то же самое, число столбцов в матрице A). Это получается при достаточно большом наборе наблюдений с переменными параметрами. Когда уравнений и неизвестных немного, обычно убирают "лишние" уравнения - те, которые наиболее сильно отличаются от остальных. При решении больших систем этот метод не подходит - можно выбросить "не то" уравнение. При решения подобных систем используют метод наименьших квадратов для минимизации вектора невязки.

Вектор невязки обозначим **r = Ax - b.** Невязкой называется длина **|r|** вектора невязки **r**. Вектор x*, доставляющий минимум невязке |r| = |Ax - b|, называется нормальным обобщенным решением системы Ax = b и вычисляется по формуле:

$$\mathbf{x^* = (A^T * A)^{-1} A^T b.}$$

Пример

Найти нормальное обобщенное решение линейной системы Ax = b для двух различных вариантов правых частей уравнения:

$$A := \begin{pmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \\ 2 & 4 & 6 \end{pmatrix} \qquad b1 := \begin{pmatrix} 3 \\ 7 \\ 6 \end{pmatrix} \qquad b2 := \begin{pmatrix} 3 \\ 7 \\ 7 \end{pmatrix}$$

$$AP := A \cdot A \qquad bp1 := A \cdot b1 \qquad bp2 := A \cdot b2$$

$$xn1 := AP \cdot bp1 \qquad xn2 := AP \cdot bp2$$

$$AP = \begin{pmatrix} 22 & 49 & 73 \\ 33 & 73 & 110 \end{pmatrix} \qquad xn1 = \begin{pmatrix} 1 \\ 1.144 \times 10^{-14} \end{pmatrix} \qquad xn2 = \begin{pmatrix} 0.6 \\ -0.4 \end{pmatrix}$$

Получено два решения: xn1 и xn2. Вычислим невязку для обоих решений:

$$r1 := |A \cdot xn1 - b1| \qquad r1 = 8.995 \times 10^{-15} \qquad r2 := |A \cdot xn2 - b2|$$

Как видно из расчетов, для системы Ax = =b1 невязка минимальна и равна нулю с точностью до погрешностей округления. Следовательно, эта система совместна.

Минимум невязки для системы Ax = b2 равен 0,447. Следовательно, эта система несовместна.

Проверим решение для первой системы

$$r := |A \cdot x - b1|$$

$$Ag = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Классическое решение x совместной системы Ax = b1 совпадает с нормальным
обобщенным решением xn1, минимум невязки для точного решения равен нулю.

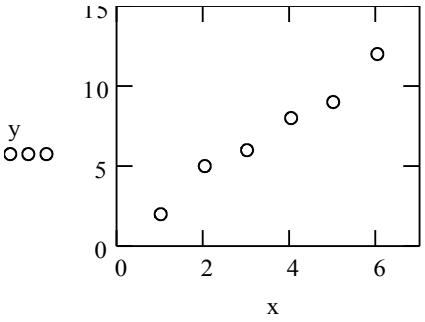
Решите самостоятельно
Найдите нормальное обобщенное решение линейной системы AX = b
для двух вариантов правых частей b1 и b2.
Найдите классическое решение системы, если оно существует, и сравните его с обобщенным

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 1.5 & 1.333 & 1.25 \\ 2.5 & 2.333 & 2.25 \end{pmatrix} \qquad b1 := \begin{pmatrix} 2 \\ 2.833 \\ 4.833 \end{pmatrix} \qquad b2 := \begin{pmatrix} 3 \\ 3.833 \\ 5.833 \end{pmatrix}$$

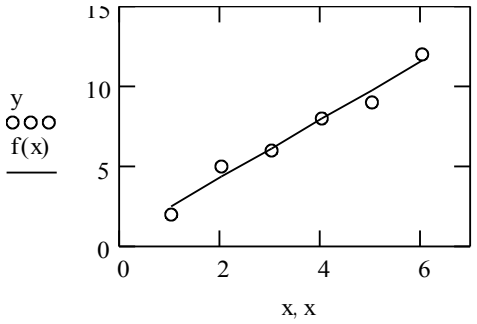
2. ВЫПОЛНЕНИЕ РЕГРЕССИИ

При решении многих практических задач решение получается в виде таблицы. График полученного решения представляет собой "облако" точек, распределенных случайным образом, но подчиняющихся определенной закономерности. Задача регрессии заключается в получении параметров некоторой функции y=y(x) такими, чтобы функция приближала бы "облако" исходных точек с наименьшей среднеквадратической погрешностью.

$$x := \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} \qquad y := \begin{pmatrix} 5 \\ 6 \\ 8 \\ 9 \\ 12 \end{pmatrix}$$



Очевидно, что переменная y зависит от переменной x (с ростом x переменная y также возрастает). Задача регресии- установить функциональную связь y=y(x).
Простейший вид уравнения регрессии - линейная регрессия. При этом график y(x) описывается уравнением прямой линии: **y = ax+b.**
Для проведения линейной регрессии в систему MathCad встроен ряд функций:
corr(VX,VY) – возвращает скаляр – коэффициент корреляции Пирсона;
intercept(VX,VY) – возвращает значение параметра b (свободного члена прямой регрессии)
slope(VX,VY) – возвращает значение параметра a (угловой коэффициент прямой регрессии)
VX,VY – исходные векторы



Как видно на рисунке, прямая регрессии проходит в "облаке" исходных точек с минимальным среднеквадратичным отклонением от них.
Вычислим невязку как сумму квадратов отклонений значений исходного вектора y и значений функции регрессии в точках x_i

$$RRR := \sum_i \left| y_i - f\left(x_i\right) \right|$$

В случае нелинейной зависимости между параметрами x и y можно использовать встроенную в MathCad функцию полиномиальной регрессии:
regress(VX,VY,n) – эта функция возвращает вектор **VS**, содержащий коэффициенты многочлена n-ной степени. Для вычисления коэффициентов полинома регрессии используется функция **submatrix**
Пример (полиномиальная регрессия).

$$data := \begin{pmatrix} 2 & 3.5 \\ 3 & 4.4 \\ 4 & 15 \\ 5 & 20 \\ 6 & 18 \end{pmatrix}$$

n=6

Степень полинома

coeffs = (11.267 −16.042 6.863 −0.666)

Таким образом, график полинома описывается уравнением:

$$y = 11{,}267 - 16{,}042x + 6{,}863x^2 - 0{,}666x^3$$

$$f1(x) := coeffs_0 + coeffs_1 \cdot x + coeffs_2 \cdot x^2 + coeffs_3 \cdot x^3$$

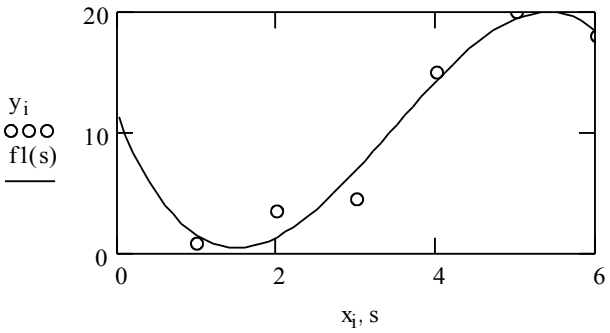


График полинома и исходные точки

На практике не рекомендуется делать степень аппроксимирующего полинома выше 4-6, так как сильно возрастают погрешности реализации регрессии.

$$RR := \sum_i \left| y_i - fl\left(x_i\right) \right|$$

Невязка равна:

Выполните самостоятельно:

Аппроксимировать функциональную зависимость между векторами vx и vy полиномом второго порядка. Вычислить невязку.

$$vx := \begin{pmatrix} 2 \\ 3 \\ 4 \\ 4.8 \\ 5.2 \end{pmatrix} \qquad vy := \begin{pmatrix} 7.5 \\ 9.4 \\ 16.2 \\ 26 \\ 32.1 \end{pmatrix}$$

Лабораторная работа № 7.

Тема: Решение нелинейных уравнений и систем.

Многие уравнения (например, трансцендентные) и системы из них не имеют аналитических решений. Однако они могут решаться численными методами с заданной погрешностью (но не более значения, заданного системной переменной TOL). Для простейших уравнений вида F(x) = 0 решение находится с помощью функции:

root(Выражение, Имя переменной)

Эта функция возвращает значение переменной, при котором выражение дает 0. Функция реализует вычисления итерационным методом, причем можно задать начальное значение переменной. Это особенно важно, если существует несколько решений. В этом случае выбор решения определяется выбором начального значения.

Пример.

Коэффициенты полинома

$$F(x) := a3\,x^3 + a2\,x^2 + a1\,x + a0$$

Задание полинома

Вычисление действительного корня

Вычисление других (возможно, комплексных) корней

$i := \sqrt{-1}$

Задание мнимой единицы

$x2 := root\left(\frac{}{x - x1}, x\right)$

$x3 := root\left(\frac{}{(x - x1)(x - x2)}, x\right)$

Для поиска всех корней полинома n-ной степени можно воспользоваться функцией:

polyroots(V) – эта функция возвращает вектор корней многочлена (полинома) степени n, коэффициенты которого находятся в векторе V, имеющем длину n+1 (так как существует "нулевой" коэффициент полинома).

Задание вектора V

$$\text{polyroots}(V) = \begin{pmatrix} 1 - 3.464i \\ 4 \end{pmatrix}$$

вычисление корней полинома

Не рекомендуется пользоваться этой функцией, если степень полинома выше пятой-шестой, так как при этом сильно возрастает погрешность вычисления корней уравнения.

Решение систем нелинейных уравнений

При решении систем нелинейных уравнений используется специальный вычислительный блок, открываемый служебным словом – директивой **Given:**

Given
Уравнения
Ограничительные условия
Выражения с функциями Find и Minerr

Рекомендуется дополнить блок проверкой решения системы.

Find(v1,v2,...,vn) – возвращает значение одной или ряда переменных для точного решения;
Minerr(v1,v2,...,vn) – возвращает значение одной или ряда переменных для приближенного решения.
Примечание! Внутри блока вместо знака "присвоения" (двоеточия со знаком равенства :=) используется знак логического (символического) равенства (символ - жирный знак равенста =)
Между функциями **Find** и **Minerr** существуют принципиальные различия. Первая функция используется, когда решение реально существует (хотя и не является аналитическим). Вторая функция пытается найти максимальное

приближение даже к несуществующему решению путем минимизации среднеквадратичной погрешности решения.

Пример. Найти корень уравнения $x^2=3$. Мы знаем, что решением будет значение плюс-минус корень квадратный из числа 3.

начальное приближение

$x^2 = 3$ Уравнение

Выражение с функцией Find

Найденное решение

Решение с применением функции Minerr также позволяет найти второе решение (обратите внимание, что начальное приближение выбрано равным –10).

$x^2 = 3$

Пример. Найти точки пересечения параболы $y=x^2$ и прямой $y=8+3x$.

диапазон построения графика

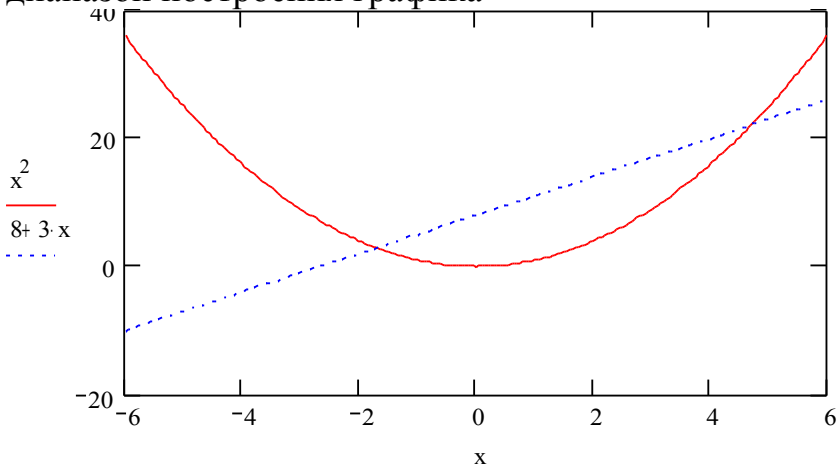


График показывает, что парабола и прямая пересекаются в двух точках: при x около -1.7 и при x около +4.8

Ищем первое решение при x=0 и y=0 и при ограничении x<0

$y = x^2$ $y = 8 + 3 \cdot x$

$\begin{pmatrix} \\ y0 \end{pmatrix} := \text{Find}(x, y$

Найденное первое решение

$\begin{pmatrix} \\ y0 \end{pmatrix} = \begin{pmatrix} \\ 2.895 \end{pmatrix}$

Проверка решения (вычислением y):

$x0 = 2.895$

Найдем второе решение системы (подобно первому, но задав ограничение x>0)

$y = x^2 \qquad y = 8 + 3 \cdot x$

$\begin{pmatrix} \\ y1 \end{pmatrix} := \text{Find}(x, y$

$\begin{pmatrix} \\ y1 \end{pmatrix} = \begin{pmatrix} \\ 22.105 \end{pmatrix}$

Найденное второе решение

Пример. Найти приближенное решение системы нелинейных уравнений $(x^2+1)^2+(y^2+1)^2=5.5$ и $x+y=0.95$

Начальные значения переменных

Начало вычислительного блока

$\left(x^2 + 1\right)^2 + \left(y^2 + 1\right)^2 = 5.5$

$x + y = 0.95$

Заданная система уравнений

$z = \begin{pmatrix} \\ 1.056 \end{pmatrix}$

Найденное решение

Проверка решения:

$\left[\left(z_0\right)^2 + 1\right]^2 + \left[\left(z_1\right)^2 + 1\right]^2 = 5.5 \qquad 0 \quad 1$

Решите самостоятельно:

1) Найти все корни уравнения $x^3=6$

2) Найти все корни уравнения $5x^3-14x^2+12x+9=0$

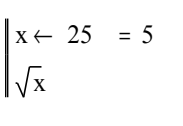
3) Построить графики и найти решение системы уравнений $y=5x^2+2x-3$ и $x+y=1$.

4) Найти приближенное решение системы нелинейных уравнений $(x^2-4)^2-(y^2-1)^2=16$ и $x+y=0.5$

Лабораторная работа № 8.
Тема: Программирование.

Хотя MathCad является достаточно мощным средством, с помощью которого можно решить множество задач, существуют нестандартные задачи, не решаемые, не решаемые или решаемые малоэффективно стандартными средствами Маткада. Для решения подобных задач создана панель **Программирование** (в английской версии **Programming Toolbar**). Вызванная кнопкой панели управления, панель Программирование содержит 10 операторов, из которых строится программный блок. Следует отметить, что все операторы должны вводиться **только из палитры**, писать их **вручную не рекомендуется**. Панель содержит следующие элементы (операторы):

Add Line - создает и при необходимости расширяет жирную вертикальную линию, справа от которой в шаблонах задается запись программного блока. Вообще данный оператор имеет аналогию в языках программирования - так называемые **"операторные скобки"**, поэтому внутри операторного блока также можно проводить подобные жирные линии, выделяющие в единый блок несколько команд.



← – символ локального присваивания (в теле модуля). Внутри модуля **вместо** символа присваивания **:=** используется символ локального присваивания ←. Следует заметить, что присваивание действует только в теле модуля, то есть если какой-то переменной внутри модуля присвоено некоторое значение, то при выходе из модуля (после выполнения всех команд в нем) переменная теряет это значение и его необходимо присваивать заново. Кроме того, в теле модуля можно вводить новые локальные переменные, причем эти локальные переменные в разных модулям могут иметь одинаковые имена и никак не влияют друг на друга.

If - оператор условного выражения. Его также нельзя вводить с клавиатуры, так как этот оператор имеет другой формат, отличающийся от формата в основной программе. Формат оператора if внутри блока следующий: **(Оператор) If (Условие)**. Если выполняется **Условие**, то будет выполнен **Оператор.**

Рассмотрим пример программы, вычисляющий модуль (абсолютное значение) некоторого числа s. В блоке использован оператор **otherwise** – оператор иного выбора, использующийся совместно с оператором **if**.

$$m(s) := \begin{cases} s & \text{if } s > 0 \\ -s & \text{otherwise} \end{cases}$$

Другой пример – программа определяет четность и нечетность числа. Возвращается 0, если число четное и 1, если нечетное.

$$chet(a) := \begin{cases} 0 & \text{if } mod(a,2) = 0 \\ 1 & \text{otherwise} \end{cases}$$

Функция mod(s,n) возвращает остаток от деления числа s на число n:

Если необходимо вычислить несколько значений функции, то необходимо поставить курсор на квадратик **Оператора** (слева от условия if) и "добавить линию" (Add Line)

```
q(a, b) := if a > b
             w ← a
             a ← b
             b ← w
           (a)
           (b)
```

$q(4, 5) = \begin{pmatrix} 5 \end{pmatrix}$ $q(7, 3) = \begin{pmatrix} 7 \end{pmatrix}$

Данный модуль записывает два числа в вектор-столбец, располагая числа по возрастанию

for – оператор задания цикла с фиксированным числом повторений

Переменная **Var** меняется с шагом +1 от значения Nmin до Nmax, при этом выполняется выражение, помещенное ниже в шаблоне.

Пример программы, вычисляющей сумму элементов от 1 до n

```
sum(n) := s ← 0
          for i ∈ 1..n
            s ← s + i
```

Пример программы, вычисляющей факториал числа n:

```
fakt(n) := g ← 1
           for i ∈ 1..n
             g ← g·i
```

Следующий пример – определение максимальной координаты вектора y (max) и ее позиции (индекса) - ind:

```
kopos(y) := ind ← 0
            max ← y_0
            for i ∈ 1..last(y)
              if y_i > max
                max ← y_i
                ind ← i
            (max)
            (ind)
```

$g := \begin{pmatrix} 4 \\ -7 \\ 0 \\ 3 \\ 1 \end{pmatrix}$ $kopos(g) = \begin{pmatrix} 1 \end{pmatrix}$

Цикл по элементам вектора (оператор **last(y)** возвращает последний индекс вектора y). Присваивание максимальному элементу max текущего значения вектора, а индексу ind- текущего индекса (в случае, когда выполняется условие if). Возвращение матрицы, содержащей два элемента максимум и индекс).

While – оператор, служащий для организации циклов, действующий до тех пор, пока выполняется некоторое условие. После оператора **while**

записывается **условие**, а **выполняемое выражение** пишется ниже (в шаблоне).

Применение оператора while для вычисления факториала

```
Fakt(n) := | d ← 1
           | while n > 0
           |     d ← d·n
           |     n ← n - 1
           | d
Fact(n) := | f ← 1
           | while n ← n - 1
           |   f ← f·(n + 1)
           | f
```

break – оператор вызывает прерывание работы программы всякий раз, как он встречается. Обычно он используется совместно с оператором условного выражения **if** или операторами циклов **while** и **for,** обеспечивая переход в конец тела цикла.

continue – оператор продолжения используется для продолжения работы после прерывания программы. Он также обычно используется совместно с операторами циклов **while** и **for,** обеспечивая возврат после прерывания в начало цикла.

Пример программы, вычисляющей факториал вводимого числа с использованием оператора **break**:

```
F(n) := | w ← n
        | while 1
        |     w ← w·(n - 1)
        |     n ← n - 1
        |     break if n = 1
        | w
```

F(10) = 3.6288 × 10

Оператор-функция возврата **return** прерывает выполнение программы и возвращает значение своего операнда, стоящего следом за ним.

Первый замечательный предел (при x=0) и значение функции в других точках.

```
S(x) := | return 1 if x = 0
        | sin(x)/x  otherwise
```

Например, в данном случае функция возвращает значение 0 при любом x<0 :

Оператор обработки ошибок **on error** позволяет создавать конструкции обработчиков ошибок. Этот оператор задается в виде:

Выражение 1 **on error** *Выражение 2* Если при выполнении Выражения 1 возникает ошибка, то выполняется Выражение 2.

```
y1(x) := (1/x)·sin(x)
```

Функция **error(S)** – функция обработки ошибок, которая, будучи в программном модуле, возвращает окошко с надписью, хранящейся в символьной переменной **S** или в символьной константе (любой фразе в кавычках).

```
F(g) := return "One"   if g = 1
        return "Two"   if g = 2
        error("No value")   otherwise
```

Примеры программ по обработке матриц

Пусть требуется переставить местами первый и последний столбцы матрицы D (причем число столбцов и строк матрицы заранее неизвестно):

```
change(D) := k ← cols(D) - 1
             n ← rows(D) - 1
             for i ∈ 0..n
                 s ← D_{i,0}
                 D_{i,0} ← D_{i,k}
                 D_{i,k} ← s
             D
```

$$D := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} \qquad \text{change}(D) = \begin{pmatrix} 4 & 2 & 3 & 1 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

Пусть требуется поменять местами i-ую и j-ую строки матрицы A

```
Perestan(A, i, j) := k ← cols(A) - 1
                     for s ∈ 0..k
                         w ← A_{i,s}
                         A_{i,s} ← A_{j,s}
                         A_{j,s} ← w
                     A
```

$$A := \begin{pmatrix} 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix} \qquad \text{Perestan}(A, 0, 2) = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

В примере переставлены местами нулевая и вторая строки

Выполните самостоятельно.

1) Ввести матрицу произвольного размера. Создать программный модуль, подсчитывающий число положительных, отрицательных и нулевых элементов в каждой строке матрицы. (Замечание: требуется вывести матрицу, содержащую столько же строк, сколько в исходной, и всего три столбца).

2) Подсчитать сумму положительных и отрицательных элементов матрицы в каждом столбце.

3) Найти сумму и произведение элементов главной и побочной диагонали.(Замечание: требуется вывести всего четыре числа).

4) Задать произвольную матрицу (любых размеров). Для заданной матрицы создать две других (можно в двух разных программных модулях), являющихся зеркальным отражением исходной матрицы относительно: а) главной диагонали; б) вспомогательной (побочной) диагонали.

5) Ввести произвольное число w. Написать программный модуль, разбивающий данное число на простые множители.

Лабораторная работа № 9.

Тема: Обыкновенные дифференциальные уравнения.

Почти все функции MathCad предназначены для решения задачи Коши нормальных систем обыкновенных дифференциальных уравнений следующего вида:

y1' = f1(x,y1,y2,...yn)
y2' = f2(x,y1,y2,...yn)
.....
yn' = fn(x,y1,y2,...yn)
при заданных начальных условиях
$y1(x0) = y1_0$
$y2(x0) = y2_0$
.....
$yn(x0) = yn_0$

Численное решение этой задачи состоит в постороении таблицы приближенных значений y1,y2,...yn на отрезке [x0,xn] в точках $x_0,x_1,x_2,...,x_n$, которые называют узлами сетки.

В библиотеке встроенных функций MathCad есть функция **odesolve**, предназначенная для решения линейных дифференциальных уравнений. Функция odesolve решает задачу Коши с заданными начальными условиями y(x0) = y0, y'(x0) = y01, ...

Перед обращением к функции **odesolve(x,b,step)** необходимо записать ключевое слово **Given**, а затем ввести уравнение и начальные либо граничные условия. Здесь **x** – имя переменной интегрирования (аргумента искомой функции), **b** – правый конец отрезка интегрирования, **step** – шаг, который используется при интегрировании уравнения методом Рунге-Кутта (этот аргумент необязателен, его можно опустить).

Пример. Найти с помощью функции odesolve на отрезке $[0,4\pi]$ решение задачи Коши

$y'' - y' \sin(x) + y = x/(2\pi)$.

начальные условия: y(0)=0, y'(0)=1

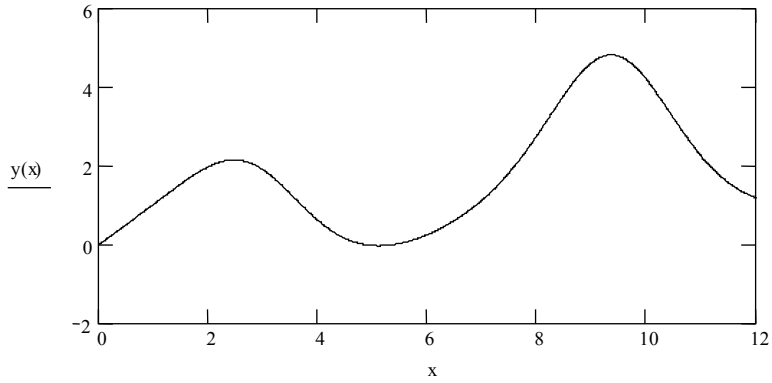
Решение

Given

$y''(x) - \sin(x)\cdot y'(x) + y(x) = \frac{x}{2\cdot\pi}$

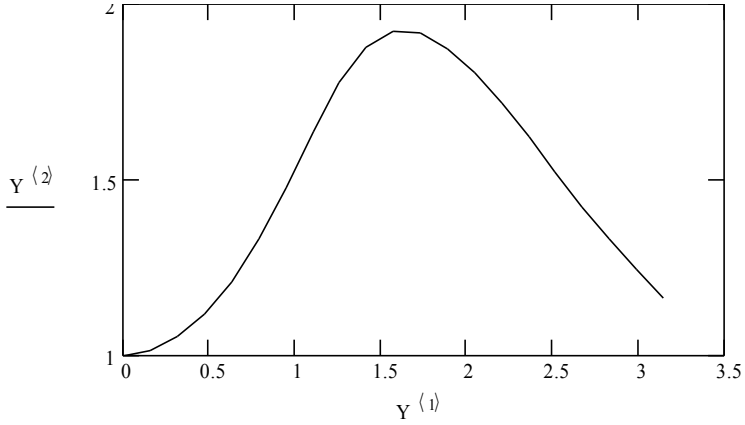
$y(0) = 0 \qquad y'(0) = 1$

y := odesolve(x, 4, 0.1)



Данную задачу можно также решить методом Рунге-Кутта при помощи функции **rkfixed(y,x1,x2,npoints,D) -** здесь **y** – вектор начальных значений $\mathbf{Y_0}$, **x1,x2** - начальная и конечная точки отрезка интегрирования системы ,**npoints** - число узлов на отрезке x1,x2, **D** – имя вектора-функции D(x,y), содержащей правые части уравнений, т.е. $\mathbf{D_i(x,y)=f_i\ (x,y1,y2,...,yn)}$

Пример. Решить уравнение **y' =sinxy** на отрезке $[0,\pi]$, удовлетворяющее начальным условиям y(0)=1. Решим задачу методом Рунге-Кутта с фиксированным шагом на сетке из 20 равноотстоящих узлов.



Обратите внимание, что перед обращением к функции rkfixed переменной y было присвоено начальное значение y_1, равное единице, а переменной D(x,y) – выражение для правой части уравнения, равное $\sin(x*y_1)$. Результаты вычислений присвоены матрице Y, которая в первом столбце содержит координаты 20 узлов равномерной сетки, а во втором – приближенные значения решения в этих узлах.

Решение задачи Коши для нормальной системы дифференциальных уравнений

Пример. Пусть требуется найти приближенное решение задачи Коши методом Рунге-Кутта

$y1' = -y2 + \sin x*y3$ $\quad y1(0) = 1$

$y2' = -y1^2$ $\quad y2(0) = 0$

$y3' = -y3 - y1$ $\quad y3(0) = 1$

на отрезке [0,3] с фиксированным шагом на сетке из 30 равноотстоящих узлов.

$y1' = y2 + \sin(x \cdot y3)$

$y2' = -y1^2$

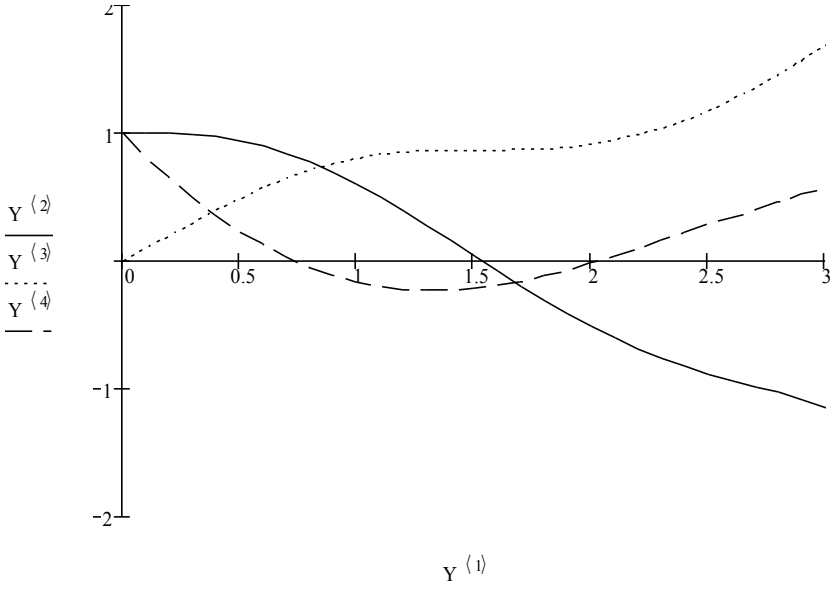
$y3' = -y3 - y1$

$y1(0) = 1$

$y2(0) = 0$

$y3(0) = 1$

$$D(x, y) := \begin{pmatrix} \\ (y_1)^2 \\ -y_3 - y_1 \end{pmatrix} \qquad y := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$



Y =

	1	2	3	4
1	0	1	0	1
2	0.1	0.999	0.1	0.81
3	0.2	0.995	0.199	0.638
4	0.3	0.984	0.298	0.483
5	0.4	0.964	0.393	0.344
6	0.5	0.932	0.483	0.221
7	0.6	0.889	0.566	0.113
8	0.7	0.833	0.64	0.021
9	0.8	0.764	0.704	-0.057
10	0.9	0.683	0.757	-0.121
11	1	0.591	0.797	-0.17
12	1.1	0.49	0.827	-0.205
13	1.2	0.382	0.846	-0.227
14	1.3	0.268	0.857	-0.236
15	1.4	0.151	0.861	-0.234
16	1.5	0.032	0.862	-0.22

Выполните самостоятельно

1) Решите на отрезке **[0,5]** задачу Коши **y' = (y*lny)/sinx** с начальным условием **y(π/2)=e** по методу Рунге-Кутта с постоянным шагом (величину

шага выберите самостоятельно). Изобразите графики решений , вычисленных с шагами **h, 2h и h/2.**

2) Решите задачу Коши для системы уравнений

y1' = y2*ln(x)

y2' = y1+y2^2

на отрезке **[1,4]** с постоянным шагом **h = 0.1** с заданными начальными условиями **y1(1) = - 2, y2(1) = - 1**

Лабораторная работа № 10.

Тема: Разложение функций в ряды.

В практике инженерных расчетов часто приходится прибегать к разложению функций в функциональные ряды, т.е. в ряды, составленные из функций определенного вида. Простейшим функциональным рядом является степенной ряд Тейлора:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

Точка x_0 называется центром разложения. При x_0=0 такой ряд называют рядом Маклорена:

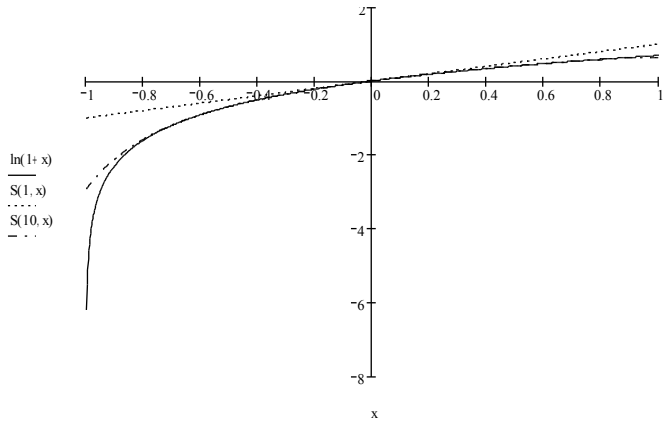
$$\sum_{n=0} \frac{f^{(n)}(0)}{n!}x^n$$

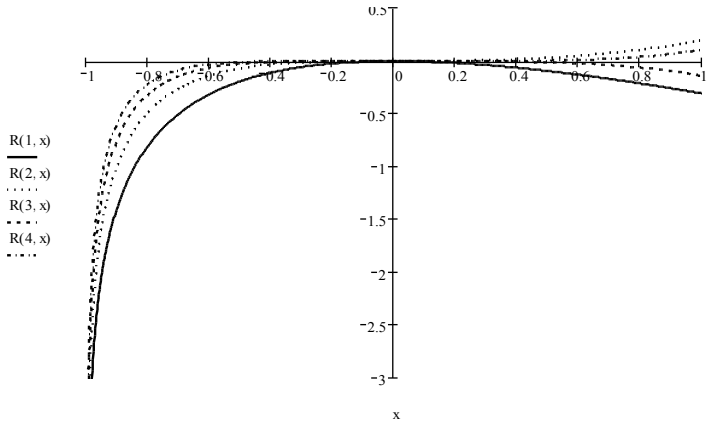
Функция f(x) может быть разложена в степенной ряд на интервале (x_0+ R, x_0 - R), если существует степенной ряд, сходящийся к f(x) на данном интервале.

Пример. Рассмотрим разложение функции ln(1+x) в ряд Тейлора

$$c(n) := \frac{(-1)^{n-1}}{n} \qquad S(n,x) := \sum_{k=1}^{n} c(k)\cdot x^k \qquad \sum_{n=1} c(n)\cdot x^n \rightarrow \ln(1+x)$$

Для оценки точности вычислений вводится понятие остатка (остаточного члена) ряда R:





Пример. Разложить функцию 1\x в ряд Тейлора в окрестности точки $x_0 = -2$. Построить график функции и частичных сумм ее ряда Тейлора в окрестности этой точки.

$t := x + 2$

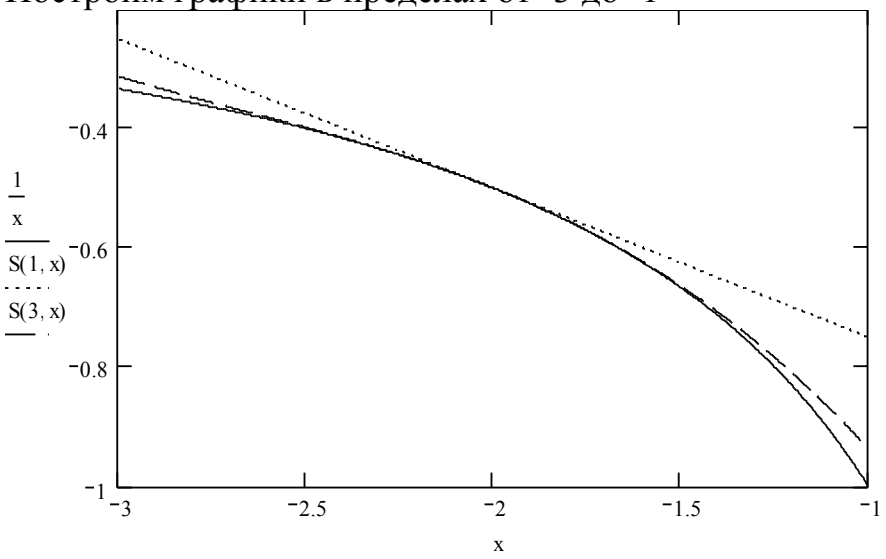
$x := t - 2$

$c(n) := \frac{1}{2^{n+1}}$

$S(n, x) := \sum_{k=0} c(k) \cdot (x + 2)^k$

$\sum_{k=0} c(k) \cdot (x + 2)^k \rightarrow \frac{1}{x}$

Построим графики в пределах от -3 до -1



Разложение функции в ряд Фурье

Ряд Фурье наиболее часто встречается в электротехнических расчетах. Каждой абсолютно интегрируемой на отрезке $[-\pi,\pi]$ функции f(x) можно поставить в соответствие ее тригонометрический ряд Фурье:

$$f(x) = \frac{a_0}{2} + \sum_{k=1} \left(a_k \cdot \cos kx + b_k \cdot \sin kx\right)$$

Коэффициенты ряда Фурье вычисляют по формулам Эйлера-Фурье:

$a_k := \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \cdot \cos(k \cdot x)\, dx$, $b_k := \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \cdot \sin(k \cdot x)\, dx$

Так как ряд Фурье есть бесконечная сумма, то иногда для расчетов удобнее пользоваться конечным числом членов ряда. Обозначим через $S_n(x)=$ $\frac{a_0}{2} + \sum_{k=1} \left(a_k \cdot \cos kx + b_k \cdot \sin kx\right)$ n-ную частичную сумму ряда Фурье.

Пример. Построить график функции y= $\begin{matrix} 1, & |x| < \frac{\pi}{2} \\ 0, & |x| > \frac{\pi}{2} \end{matrix}$ и частичных сумм с различным числом членов.

$$f(x) := \begin{cases} 1 & \text{if } \frac{-\pi}{2} \le x \le \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

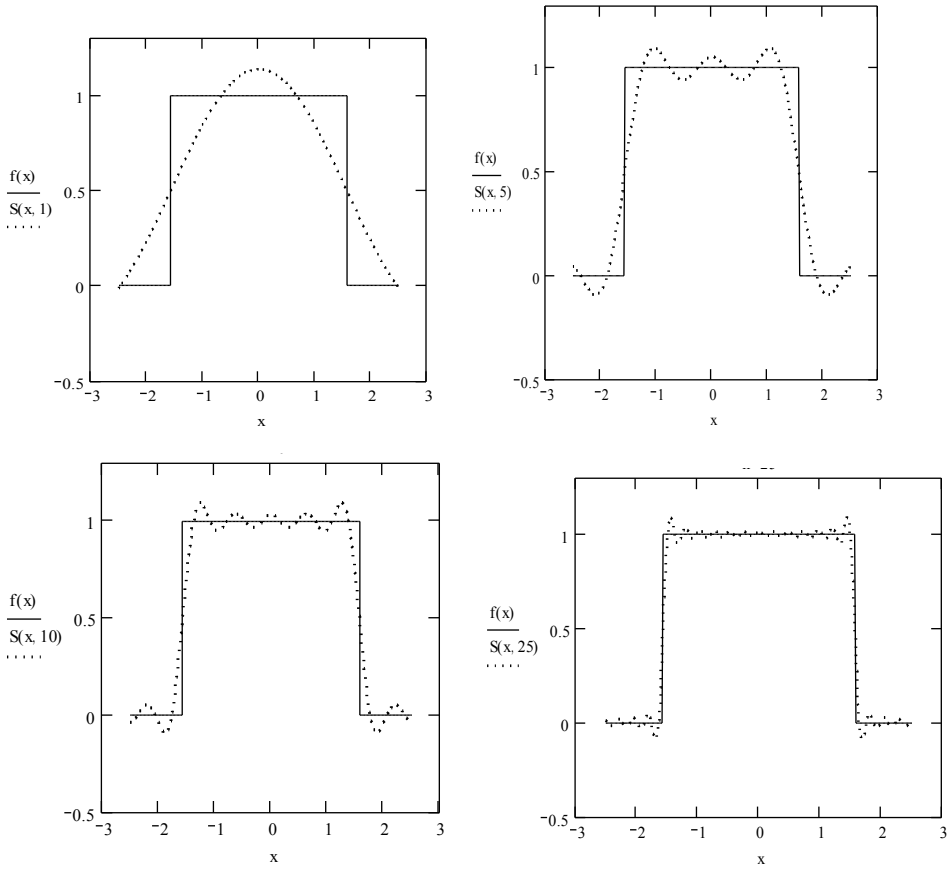
Вычисление коэффициентов ряда Фурье

$$a_k := \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \cdot \cos(k \cdot x)\, dx \qquad b_k := \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \cdot \sin(k \cdot x)\, dx$$

Вычисление частичной суммы ряда Фурье

$$S(x, n) := \frac{a_0}{2} + \sum_{k=1}^{n} \left(a_k \cdot \cos(k \cdot x) + b_k \cdot \sin(k \cdot x) \right)$$

Как видно из графиков, уже при n=25 цифровой сигнал достаточно хорошо описывается набором аналоговых сигналов (синусов и косинусов).



Выполните самостоятельно

1. Разложите в ряд Тейлора функцию f(x)= e^x в окрестности точки x=0. Постройте графики для частичных сумм для n = 1,5,10,50.

2. Разложите в ряд Фурье функцию f(x)= $\begin{cases} -\frac{\pi + x}{2}, -\pi \le x \le 0 \\ \frac{\pi - x}{2}, 0 < x < \pi \end{cases}$

Постройте графики для частичных сумм для n = 1,2,5,10,25,50.

Лабораторная работа № 11

Тема: Решение дифференциальных уравнений при помощи преобразования Лапласа.

Одним из важнейших применений операционного исчисления является решение линейных дифференциальных уравнений с постоянными коэффициентами:

$$a_0 y^{(n)} + a_1 y^{(n-1)} + \dots + a_n y = f(t) \qquad (1)$$

где $a_0, \dots, a_n$ – заданные числа (коэффициенты уравнения),

f(t) – заданная функция,

y(t) – искомая (неизвестная) функция.

Требуется решить задачу Коши, т.е. найти решение этого уравнения, удовлетворяющее начальным условиям $y(0) = y_0$, $y'(0) = y'_0$, …, $y^{(n-1)}(0) = y_0^{(n-1)}$, где $y_0, y'_0, \dots, y_0^{(n-1)}$ – заданные числа.

Перейдем от функций y(t) и f(t) к их изображениям Y(p) и F(p), используя свойство линейности преобразования Лапласа и теоремы о дифференцировании оригинала:

$$a_0 (p^n Y - p^{n-1} y_0 - p^{n-2} y'_0 - \dots - y_0^{(n-1)}) + a_1 (p^{n-1} Y - p^{n-2} y_0 - p^{n-3} y'_0 - \dots - y_0^{(n-2)}) + \dots + a_n Y = F(p) \qquad (2)$$

Для перехода от функции f(t) к ее изображению F(p) используют таблицы перехода и основные теоремы, характеризующие свойства преобразования Лапласа.

№ п/п	Оригинал	Изображение	№ п/п	Оригинал	Изображение
1	1	$1/p$	11	$t \sin \omega t$	$2p\omega/(p^2+\omega^2)^2$
2	e^{at}	$1/(p-a)$	12	$t \cos \omega t$	$p^2 - \omega^2/(p^2+\omega^2)^2$
3	$\sin \omega t$	$\omega/(p^2+\omega^2)$	13	$t \operatorname{sh} \omega t$	$2p\omega/(p^2 - \omega^2)^2$
4	$\cos \omega t$	$p/(p^2+\omega^2)$	14	$t \operatorname{ch} \omega t$	$p^2+\omega^2/(p^2 - \omega^2)^2$
5	$\operatorname{sh} \omega t$	$\omega/(p^2-\omega^2)$	15	$(\sin \omega t)/t$	$\pi/2 - \operatorname{arctg}(p/\omega)$
6	$\operatorname{ch} \omega t$	$p/(p^2-\omega^2)$	16	$\alpha f(t)+\beta g(t)$	$\alpha F(p)+\beta G(p)$
7	$e^{at} \sin \omega t$	$\omega/((p-a)^2+\omega^2)$	17	$f(\lambda t)$	$(1/\lambda) F(p/\lambda)$
8	$e^{at} \cos \omega t$	$(p-a)/((p-a)^2+\omega^2)$	18	$f(t-\tau)$	$e^{-p\tau}F(p)$
9	t^n , n = 1,2,…	$n!/p^{n+1}$	19	$e^{-\lambda t} f(t)$	$F(p+\lambda)$
10	$t^n e^{at}$	$n!/(p-a)^{n+1}$	20	$f'(t)$	$P F(p) – f(0)$

Уравнение (2) проще исходного дифференциального уравнения (1), так как оно является линейным алгебраическим уравнением относительно Y.

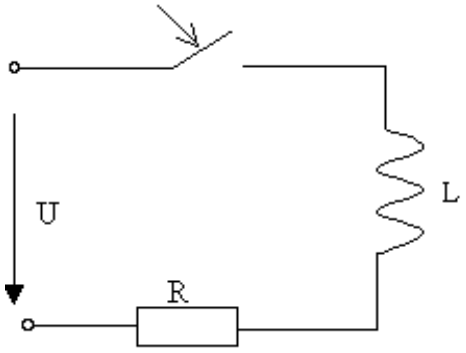
Решая данное уравнение, находят Y(p), после чего выполняют обратное преобразование Лапласа – от найденного изображения Y(p) к оригиналу y(t).

Пример. Пусть требуется найти закон изменения электрического тока от времени для следующей электрической схемы:

Исходные данные для расчета: U = 100 В, R = 10 Ом, L = 0,5 Гн.

Дифференциальное уравнение, которое описывает процессы, происходящие в данной схеме, имеет следующий вид:

$$L \cdot \frac{di}{dt} + R \cdot i = U \qquad (3)$$



Перейдем от дифференциального уравнения относительно оригинала i(t) к уравнению относительно изображения I(p): **L(p I(p) – i(0)) + R I(p) = U/p**.

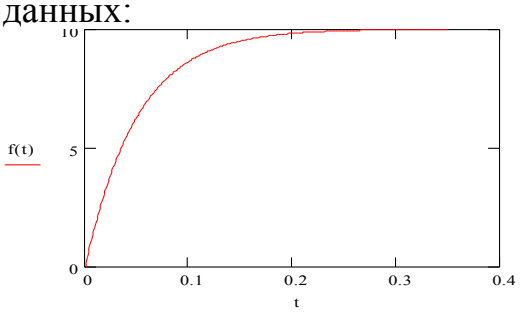
Выразив из этого уравнения I(p), получим решение уравнения (3) относительно его изображения (в переменных p):

$$I(p) = \frac{U + p \cdot L \cdot i(0)}{p \cdot (pL + R)} \qquad (4)$$

Зная изображение I(p), можно путем обратного преобразования Лапласа (функцией invlaplase) перейти обратно к оригиналу f(t) и построить график переходного процесса.

Перейдем к решению задачи в системе MathCad.

Ввод начальных данных:



Следует обратить внимание, что в Маткаде вместо переменной **p** используется переменная **s**.

Обратное преобразование Лапласа:

$$f(t) := \frac{0}{s \cdot (s \cdot L + R)} \text{ invlaplace , s } \rightarrow 10. - 10. \cdot \exp(-20. \cdot t)$$

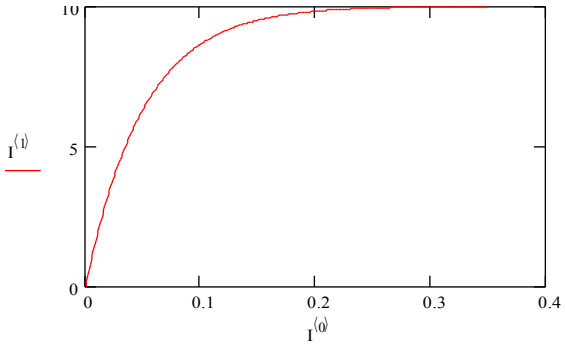
диапазон для построения графика

Проверка решения методом Рунге-Кутта:

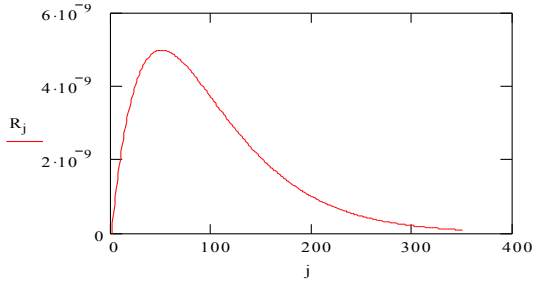
$$L\frac{d}{dt}i + R\cdot i \equiv U \qquad 0$$

$$D(t, i) := \frac{U}{L} - \frac{0}{L}$$

Решение методом Рунге-Кутта

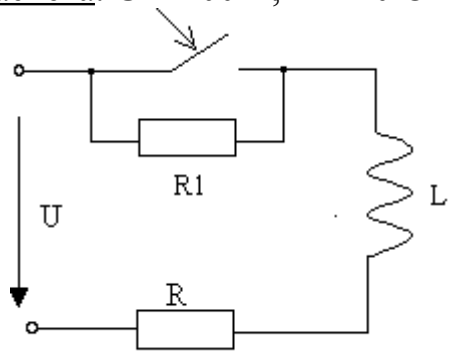


Невязка между методами:



Таким образом, максимальная невязка не превышает $5*10^{-9}$ А (при максимальном значении тока 10 А).

<u>Выполнить самостоятельно</u>. Построить графики переходных процессов для заданной схемы, в которой производится шунтирование сопротивления R1.
<u>Исходные данные для расчета</u>: U = 200 В, R = 10 Ом, R1 = 1,5 R, L = 0,5 Гн.



Требуется:

1) Используя преобразование Лапласа, решить заданную систему и найти закон изменения тока i(t) в цепи при включении шунта. Примечание: уравнения (3) и (4) справедливы и в данном случае. Необходимо лишь найти i(0). Также необходимо подобрать временной диапазон, в котором ток меняется от начальной величины до конечной.
2) Выполнить проверку правильности вычисления i(t), используя метод Рунге-Кутта. Вычислить невязку между методами и построить ее график, найти максимум невязки.
3) Исследовать влияние индуктивности L, построив три графика i(t) с различной величиной L: 0,5 Гн, 0,25 Гн и 0,1 Гн.
4) Исследовать влияние сопротивления R1, приняв три различных его значения: R1 = R, R1 = 2R, R1 = 0.1R. Построить три графика.

5 семестр

Лабораторная работа 1. Библиотека OpenGL

Выполнить четыре задания.

1. Создайте проект, состоящий из файла с исходным текстом программы 1 и библиотечные файлы OpenGL: opengl32.lib, glu32.lib, glaux.lib . Скомпилируйте и запустите программу. Попробуйте изменить цвет сферы, пользуясь функцией glColor3..( ) .

```
// Программа 1
#include <windows.h> // Заголовочный файл с описаниями
функций Windows
#include <GL/gl.h>        // Заголовочные файлы
библиотеки OpenGL
#include <GL/glu.h>
#include <GL/glaux.h>

// Прототипы функций обратной связи (для
автоматического вызова из GLAUX)
void CALLBACK resize( int width, int height );
void CALLBACK display( void );

void main()
{
  // Параметры обсчета сцены в OpenGL: цветовой режим
RGBA, удаление
  // невидимых поверхностей и линий, двойная
буферизация
  auxInitDisplayMode( AUX_RGBA | AUX_DEPTH | AUX_DOUBLE
);

  // Создание окна OpenGL с заголовком "Программа 1"
  // Размер окна - 400x400 пикселей. Левый верхний угол
окна
  // задается экранными координатами (50, 10).
  auxInitPosition( 50, 10, 400, 400);
  auxInitWindow( "Программа 3.1" );

  // В случае, когда окно не получает сообщений от
клавиатуры, мыши или
  // таймера, то будет вызываться функция display. Так
можно получить
```

```
  // анимацию. Если анимация не нужна, то эта строка
лишняя.
  auxIdleFunc( display );

  // Задание функции, которая будет вызываться при
изменении
  // размеров окна Windows.
  auxReshapeFunc( resize );

  // Включение ряда параметров OpenGL
  glEnable( GL_ALPHA_TEST );      // Учет прозрачности
  glEnable( GL_DEPTH_TEST );      // Удаление невидимых
поверхностей
  glEnable( GL_COLOR_MATERIAL );  // Синхронное задание
цвета рисования
                                  // и цвета
материала объектов
  glEnable( GL_BLEND );           // Разрешение
смешения цветов
  glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
  glEnable( GL_LIGHTING );        // Учет освещения
  glEnable( GL_LIGHT0 ); // Включение нулевого
источника света

  // Задание положения и направления нулевого источника
света
  float pos[4] = { 3, 3, 3, 1 };
  float dir[3] = { -1, -1, -1 };
  glLightfv( GL_LIGHT0, GL_POSITION, pos );
  glLightfv( GL_LIGHT0, GL_SPOT_DIRECTION, dir );

  // Задание функции отрисовки окна. Эта функция будет
вызываться всякий
  // раз, когда потребуется перерисовать окно на экране
(например, когда
  // окно будет развернуто на весь экран)
  auxMainLoop( display );
}


void CALLBACK resize( int width, int height )
{
  // Указание части окна для вывода кадра OpenGL
  glViewport( 0, 0, width, height );
  glMatrixMode( GL_PROJECTION );
```

```
  glLoadIdentity();

  // Задание типа проекции (glOrtho - параллельная,
glFrustum -
  // перспективная). Параметры функций определяют
видимый объем
  // (левая стенка - пять единиц влево, правая - пять
единиц вправо,
  // далее задаются нижняя стенка, верхняя, передняя и
задняя)
  glOrtho( -5, 5, -5, 5, 2, 12 );

  // Задание позиции наблюдателя (0, 0, 5), направление
луча
  // зрения (на точку (0, 0, 0)), вектор, принимаемый
за направление
  // "вверх" (0, 1, 0) (т.е. параллельно оси Y).
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}


void CALLBACK display(void)
{
  // Очистка буфера кадра
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

  // Перенос системы координат, связанной с объектом,
на 0.01
  glTranslated( 0.01, 0, 0 );
  // Рисование в начале координат, связанных с
объектом, сферы
  // радиусом 1, окрашенной в красный цвет
  glColor3d( 1, 0, 0 );
  auxSolidSphere( 1 );

  // Копирование содержимого буфера кадра на экран
  auxSwapBuffers();
}
```

2. С помощью перечисленных ниже функций нарисуйте стандартные фигуры библиотеки GLAUX: куб, параллелепипед и др. Значения параметров функции выбирайте в диапазоне 0.5 – 1.7 (фигура слишком большего размера будет выходить за пределы видимого объёма)

Таблица

Фигура	Функция GLAUX
Куб	auxSolidCube (width)
Параллелепипед	auxSolidTorus (width, height, depth)
Тор	auxSolidTorus (r, R)
Цилиндр	auxSolidCylinder (r, height)
Конус	auxSolidCone (r, height)
Икосаэдр	auxSolidIcosahedron (width)
Октаэдр	auxSolidOctahedron (width)
Тетраэдр	auxSolidDodecahedron (width)
Додекаэдр	auxSolid(width)
Чайник	auxSolidTeapot (width)

В таблице приведены имена функций для рисования сплошных фигур. В GLAUX есть также аналогичные функции (вместо Solid имена этих функций содержат слово Wire) для рисования каркасных фигур. Например, для рисования куба надо вызвать функцию: auxWireCube (width); .

Переход к новым координатам

Продолжим рисовать трехмерные фигуры. В предыдущем задании вы научились рисовать примитивные трехмерные объекты. Но проблема в том, что они рисуются только в начале координат, т.е. в точке (0,0,0). Для того чтобы изобразить сферу в точке (x_0,y_0,z_0), надо переместить начало координат в эту точку, т.е. надо перейти к новым координатам. Эта процедура довольно распространенная при программировании графики и анимации. Часто, бывает очень удобно, сместить координаты в новую точку и повернуть их на требуемый угол, и ваши расчеты резко упростятся. А пока вы узнаете, как переходить к новым координатам. Для перехода к новым координатам в OpenGL есть две функции:

- glTranslated(Δx,Δy,Δz)
- glRotated(φ,x_0,y_0,z_0)

Первая функция сдвигает начало системы координат на (Δx,Δy,Δz). Вторая - поворачивает на угол φ против часовой стрелки вокруг вектора (x0,y0,z0). Теперь, стоит сказать еще о двух очень важных функциях:

- glPushMatrix()
- glPopMatrix()

Они предназначены для сохранения и восстановления текущих координат. Отметим, что при переходе от одной системы координат к другой необходимо помнить все переходы. Поэтому удобнее с помощью glPushMatrix() сохранить текущие координаты, затем, осуществляя необходимые преобразования (смещение, вращение и тому подобное), вызовом glPopMatrix вернуться к старым координатам. Итак, настало время

поэкспериментировать. Создайте новый проект, повторив пункты программы задания 1. Только измените его имя. Сначала мы рассмотрим сдвиг координат. Вставьте в функцию display следующий код:

```
 glPushMatrix();       // сохраняем текущие координаты
  glTranslated(1.4,0,0);  //сдвигаемся по оси X на 1.4
  glColor3d(0,1,0);
  auxSolidSphere(0.5); // рисуем сферу в (1.4,0,0)
                       // в абсолютных координатах
  glTranslated(1,0,0);  // еще раз сдвигаемся
  glColor3d(0,0,1);
  auxSolidSphere(0.3);
 glPopMatrix();  // возвращаемся к старой системе координат
 glColor3d(1,0,0);
 auxSolidSphere(0.75); // рисуем сферу в точке (0,0,0)
                      // в абсолютных координатах
```

Если не получится, то воспользуйтесь программой

```
#include <windows.h>

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>



void CALLBACK resize(int width,int height)
{
   glViewport(0,0,width,height);
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   glOrtho(-5,5, -5,5, 2,12);
   gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
   glMatrixMode( GL_MODELVIEW );
}



void CALLBACK display(void)
{
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    glPushMatrix();       // сохраняем текущие координаты

     glTranslated(1.4,0,0);  // сдвигаемся по оси X на единицу
     glColor3d(0,1,0);
     auxSolidSphere(0.5); // рисуем сферу в (1,0,0)
                     // в абсолютных координатах

     glTranslated(1,0,0);  // еще раз сдвигаемся
     glColor3d(0,0,1);
     auxSolidSphere(0.3);

    glPopMatrix();  // возвращаемся к старой системе координат

    glColor3d(1,0,0);
```

```
    auxSolidSphere(0.75); // рисуем сферу в точке (0,0,0)
                          // в абсолютных координатах

    auxSwapBuffers();
}



void main()
{
        float pos[4] = {3,3,3,1};
        float dir[3] = {-1,-1,-1};

    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Sphere2" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);


    glEnable(GL_ALPHA_TEST);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);

        auxMainLoop(display);
}
```

В общем-то, из комментариев многое понятно. Обратим ваше внимание только на два последних вызова auxSolidSphere. Перед вызовом glPopMatrix сфера рисуется в точке (2,0,0), а после, в точке (0,0,0).

3. "Список трехмерных фигур"

С помощью функций, перечисленных в таблице 2-го задания, нарисуйте стандартные фигуры библиотеки GLAUX в два столбца (по пять фигур в каждом столбце): в1-м – сплошные фигуры, во 2-м – каркасные.

Примечание: тут следует заметить, что в версии glaux.lib от фирмы Microsoft имеется следующий баг: цилиндр и конус рисуются всегда либо каркасными, либо сплошными. Если вы первый цилиндр/конус в программе нарисовали каркасным, то далее все цилиндры/конусы будут каркасными. Соответственно, если первой была сплошная фигура, то далее все будут сплошные. Поэтому, не стоит “паниковать” – это ошибка Microsoft.

Пример программы в 4-е столбца трехмерных фигур:

```
#include <windows.h>

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
```

```
void CALLBACK resize(int width,int height)
{
   glViewport(0,0,width,height);
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   glOrtho(-5,5, -5,5, 2,12);
   gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
   glMatrixMode( GL_MODELVIEW );
}



void CALLBACK display(void)
{

    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

  glPushMatrix();

    glTranslated(0.5,4,0);
        glColor3d(0,0,1);
    auxSolidCube(1);      // куб

    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxSolidBox(1,0.75,0.5);  // коробка

        glTranslated(0,-2,0);
        glColor3d(0,1,1);
    auxSolidTorus(0.2,0.5);         // тор

    glTranslated(0,-2,0);
        glColor3d(1,0,0);
    auxSolidCylinder(0.5,1); // цилиндр


    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxSolidCone(1,1);     // конус

    glTranslated(2,8,0);
        glColor3d(1,0,1);
    auxSolidIcosahedron(1); // многогранники

    glTranslated(0,-2,0);
        glColor3d(1,1,1);
    auxSolidOctahedron(1);

    glTranslated(0,-2,0);
        glColor3d(0,1,1);
    auxSolidTeapot(0.7);       // рисует чайник

    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxSolidTetrahedron(1);

    glTranslated(0,-2,0);
        glColor3d(1,1,0);
    auxSolidDodecahedron(1);
```

```
        glTranslated(-6,8,0);
        glColor3d(0,0,1);
    auxWireCube(1);      // куб

    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxWireBox(1,0.75,0.5);  // коробка

        glTranslated(0,-2,0);
        glColor3d(0,1,1);
    auxWireTorus(0.2,0.5);          // тор

    glTranslated(0,-2,0);
        glColor3d(1,0,0);
    auxWireCylinder(0.5,1); // цилиндр


    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxWireCone(1,1);     // конус

    glTranslated(2,8,0);
        glColor3d(1,0,1);
    auxWireIcosahedron(1); // многогранники

    glTranslated(0,-2,0);
        glColor3d(1,1,1);
    auxWireOctahedron(1);

    glTranslated(0,-2,0);
        glColor3d(0,1,1);
    auxWireTeapot(0.7);        // рисует чайник

    glTranslated(0,-2,0);
        glColor3d(0,1,0);
    auxWireTetrahedron(1);

    glTranslated(0,-2,0);
        glColor3d(1,1,0);
    auxWireDodecahedron(1);

  glPopMatrix();


    auxSwapBuffers();
}



void main()
{
        float pos[4] = {3,3,3,1};
        float dir[3] = {-1,-1,-1};

    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Shapes" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);


    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
```

```
    glEnable(GL_LIGHT0);


    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);

    auxMainLoop(display);
}
```

Поворот координат

Теперь рассмотрим вращение координат. Создайте новый проект. В функцию display вставьте следующий код:

```
glColor3d(1,0,0);
auxSolidCone(1, 2);  // рисуем конус в центре координат
glPushMatrix();     // сохраняем текущие координаты
 glTranslated(1,0,0);  // сдвигаемся в точку (1,0,0)
 glRotated(75, 1,0,0); // поворачиваем систему координат на 75 градусов
 glColor3d(0,1,0);
 auxSolidCone(1, 2);   // рисуем конус
glPopMatrix();     // возвращаемся к старым координатам
```

Как видите, конус повернулся в абсолютных координатах. Так что, для того, чтобы нарисовать фигуру не в начале координат, надо:

1. сохранить текущие координаты
2. сдвинуть(glTranslated), повернуть(glRotated)
3. нарисовать то, что хотели
4. вернуться к старым координатам

Вызовы glPushMatrixglPopMatrix могут быть вложенными, т.е.:

```
glPushMatrix();
  ...
   glPushMatrix();
   glPopMatrix();
  ...
glPopMatrix();
```

Естественно число вызовов glPopMatrix должно соответствовать числу вызовов glPushMatrix, иначе у вас сцена улетит в неизвестном направление. Максимально допустимая глубина вложенности glPushMatrix/glPopMatrix определяется следующим образом:

```
int n;
glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, &n);
printf("n=%d\n",n);
```

Спецификация на OpenGL гарантирует, что глубина стека не может быть меньше 32.

4. "Снеговик"

Используя функцию glRotate, нарисуйте снеговика. Три сферы, шапка - конус, нос - тоже конус, глаза - сфера, рот можно квадратным сделать - glBox.

Анимация

Давайте оживим нашего снеговика и добавим интерактивность. Для этого надо отрисовывать кадры и реагировать на внешние события от клавиатуры или мыши. Для отрисовки кадров их надо как-то различать. Для этого мы в функции display вводим переменную time типа int с модификатором static. Создайте новый проект и в функцию display введите следующее: "static int time=0;". Модификатор static означает, что значение переменной будет сохраняться при выходе из функции. Начальное значение мы устанавливаем в ноль. Если функция display не зависит от времени, то счетчик кадров можно и не вести. Теперь добавьте следующие строчки:

```
glPushMatrix();
 glTranslated( ((double)time)/100.0 ,0,0);
  ... // здесь код из предыдущего упражнения "Снеговик"
glPopMatrix();
```

Запустив приложение, вы увидите, что снеговик пополз вдоль оси X. Вот так вы можете делать анимацию, т.е. теперь координаты объектов вычисляются в зависимости от времени. Это простой пример. Вообще говоря, для программирования сложной графики вам понадобится вычислять координаты каждого отдельного объекта в зависимости от времени.

Лабораторная работа 2. Геометрические примитивы

Общие положения

Точки, линии, треугольники, четырехугольники, многоугольники - простые объекты, из которых состоят любые сложные фигуры. В предыдущей лабораторной работе мы рисовали сферу, конус и тор и т.д.. OpenGL непосредственно не поддерживает функций для создания таких сложных объектов, т.е. таких функций нет в opengl32.dll. Эти функции есть в библиотеки утилит glu32.dll, и устроены они следующим образом. Для того чтобы нарисовать сферу функция auxSolidSphere использует функции из библиотеки glu32.dll, а те в свою очередь, используют базовую библиотеку opengl32.dll и из линий или многоугольников строят сферу. Примитивы создаются следующим образом:

```
glBegin(GLenum mode);  // указываем, что будем рисовать
  glVertex[2 3 4][s i f d](...); // первая вершина
  ...                           // тут остальные
вершины
  glVertex[2 3 4][s i f d](...); // последняя вершина
 glEnd();                // закончили рисовать примитив
```

Сначала определяем, что будем рисовать – glBegin с соответствующим параметром. Возможные значения mode перечислены ниже в таблице. Далее указываете вершины, определяющие объекты указанного типа. Обычно будем задавать вершину одним из четырех способов.

```
glVertex2d(x,y);     // две переменных типа double
glVertex3d(x,y,z);   // три переменных типа double
glVertex2dv(array); // массив из двух переменных типа
double
glVertex3d(array);   // массив из трех переменных типа
double
```

И завершаете glEnd, чтобы указать, о завершении рисования объекты типа, указанного в glBegin. Далее мы подробно разберем создание всех примитивов.

Значение mode	Описание
GL_POINTS	Каждый вызов glVertex задает отдельную точку.
GL_LINES	Каждая пара вершин задает отрезок.
GL_LINE_STRIP	Рисуется ломанная.
GL_LINE_LOOP	Рисуется ломанная, причем ее последняя точка соединяется с первой.
GL_TRIANGLES	Каждые три вызова glVertex задают треугольник.
GL_TRIANGLE_STRIP	Рисуются треугольники с общей стороной.
GL_TRIANGLE_FAN	Тоже самое, но по другому правилу соединяются вершины, вряд ли понадобится.
GL_QUADS	Каждые четыре вызова glVertex задают четырехугольник.
GL_QUAD_STRIP	Четырехугольники с общей стороной.
GL_POLYGON	Многоугольник.

Точки

Вы можете нарисовать столько точек, сколько вам нужно, вызывая для этого glVertex3d, которая устанавливает новую точку. При создании точек вы можете изменять следующие параметры: можете вызывать glColor3d внутри glBegin/glEnd. Размер точки можно устанавливать с помощью функции:

```
void glPointSize(GLfloat size)
```

Режим сглаживания можно устанавливать вызовом функции

```
glEnable(GL_POINT_SMOOTH)
```

Отключается соответственно вызовом glDisable() с таким параметром. Последние функции - glPointSize и glEnable/glDisable надо вызывать вне glBegin/glEnd, иначе они будут проигнорированы. Функции glEnable/glDisable включают/выключают множество опций, но вы должны учитывать, что некоторые опции влекут за собой большие вычисления и, следовательно, изрядно затормаживают ваше приложение, поэтому без надобности не стоит их включать. Очевидно, что совершенно не к чем включать освещение, наложение текстуру и сглаживания цветов при рисовании точек. Пока вы с этими возможностями OpenGL ещё не знакомы, и использовать их не нужно.

```
// рисуем точки
 glPointSize(2);
 glBegin(GL_POINTS);
  glColor3d(1,0,0);
  glVertex3d(-4.5,4,0); // первая точка
```

```
 glColor3d(0,1,0);
 glVertex3d(-4,4,0);   // вторая точка
 glColor3d(0,0,1);     // третья
glVertex3d(-3.5,4,0);
glEnd();
glPointSize(5);
glBegin(GL_POINTS);
 glColor3d(1,0,0);
 glVertex3d(-2,4,0); // первая точка
 glColor3d(0,1,0);
 glVertex3d(-1,4,0);   // вторая точка
 glColor3d(0,0,1);     // третье
 glVertex3d(0,4,0);
glEnd();
glPointSize(10);
glEnable(GL_POINT_SMOOTH);
glBegin(GL_POINTS);
 glColor3d(1,0,0);
 glVertex3d(2,4,0); // первая точка
 glColor3d(0,1,0);
 glVertex3d(3,4,0);   // вторая точка
 glColor3d(0,0,1);     // третья
 glVertex3d(4,4,0);
glEnd();
glDisable(GL_POINT_SMOOTH);
```

Линии

Для линий вы также можете изменять ширину, цвет, размер, сглаживание. Если вы зададите разные цвета для начала и конца линии, то ее цвет будет переливающемся. OpenGL по умолчанию делает интерполяцию. Так же вы можете рисовать прерывистые линии, делается это путем наложения маски при помощи следующей функции:

```
void glLineStipple(GLint factor, GLushort pattern   );
```

Второй параметр задает саму маску. Например, если его значение равно 255 (0x00FF), то чтобы вычислить задаваемую маску воспользуемся калькулятором. В двоичном виде это число выглядит так: 0000000011111111, т.е. всего 16 бит. Старшие восемь установлены в ноль, значит тут линии не будет. Младшие установлены в единицу, тут будет рисоваться линия. Первый параметр определяет, сколько раз повторяется каждый бит. Скажем, если его установить равным 2, то накладываемая маска будет выглядеть так:

```
0000000000000000111111111111111
```

Далее приведен исходный текст с комментариями для наглядной демонстрации.

```
glLineWidth(1);        // ширину линии
                       // устанавливаем 1
 glBegin(GL_LINES);
  glColor3d(1,0,0);      // красный цвет
  glVertex3d(-4.5,3,0); // первая линия
  glVertex3d(-3,3,0);
  glColor3d(0,1,0);      // зеленый
  glVertex3d(-3,3.3,0); // вторая линия
  glVertex3d(-4,3.4,0);
 glEnd();
 glLineWidth(3);       // ширина 3
 glBegin(GL_LINE_STRIP); // см. ниже
  glColor3d(1,0,0);
  glVertex3d(-2.7,3,0);
  glVertex3d(-1,3,0);
  glColor3d(0,1,0);
  glVertex3d(-1.5,3.3,0);
  glColor3d(0,0,1);
  glVertex3d(-1,3.5,0);
 glEnd();
 glLineWidth(5);
 glEnable(GL_LINE_SMOOTH);
 glEnable(GL_LINE_STIPPLE); // разрешаем рисовать
                            // прерывистую линию
 glLineStipple(2,58360);    // устанавливаем маску
                            // пояснения см. ниже
 glBegin(GL_LINE_LOOP);
  glColor3d(1,0,0);
  glVertex3d(1,3,0);
  glVertex3d(4,3,0);
  glColor3d(0,1,0);
  glVertex3d(3,2.7,0);
  glColor3d(0,0,1);
  glVertex3d(2.5,3.7,0);
  glEnd();
 glDisable(GL_LINE_SMOOTH);
 glDisable(GL_LINE_STIPPLE);
```

Треугольники

Для треугольника можно задавать те же параметры, что и для линии плюс есть еще одна функция glPolygonMode. Она устанавливает опции для отрисовки многоугольника. Первый параметр может принимать значения - GL_FRONT, GL_BACK и GL_FRONT_AND_BACK. Второй параметр указывает, как будет рисоваться многоугольник. Он принимает значения: GL_POINT(рисуются только точки); GL_LINE(рисуем линии); GL_FILL(рисуем заполненный многоугольник). Первый параметр указывает: к лицевой, тыльной или же к обеим сторонам применяется опция, заданная вторым параметром. Треугольники можно рисовать, передав GL_TRIANGLE_STRIP или GL_TRIANGLE_FAN в glBegin. В первом случае первая, вторая и третья вершины задают первый треугольник. Вторая, третья и четвертая вершина - второй треугольник. Третья, четвертая и пятая вершина - третий треугольник и т.д. Вершины n, n+1 и n+2 определят n-ый

треугольник. Во втором случае первая, вторая и третья вершина задают первый треугольник. Первая, третья и четвертая вершины задают второй треугольник и т.д. Вершины 1, n+1, n+2 определяют n-ый треугольник. Далее следует пример с комментариями.

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // см. выше
 glBegin(GL_TRIANGLES);
  glColor3d(1,0,0);      // рисуем треугольник
  glVertex3d(-4,2,0);
  glVertex3d(-3,2.9,0);
  glVertex3d(-2,2,0);
 glEnd();
glLineWidth(2);
 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); //рисуем
                              // проволочные треугольники
 glBegin(GL_TRIANGLE_STRIP); // обратите внимание на порядок
                              // вершин
  glColor3d(0,1,0);
  glVertex3d(1,2,0);
  glVertex3d(0,2.9,0);
  glVertex3d(-1,2,0);
  glVertex3d(0,1.1,0);
 glEnd();
 glEnable(GL_LINE_STIPPLE);
 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
 glBegin(GL_TRIANGLE_FAN);
  glColor3d(0,0,1);
  glVertex3d(4,2,0);
  glVertex3d(2.6,2.8,0);
  glVertex3d(2,2,0);
  glVertex3d(3,1.1,0);
 glEnd();
 glDisable(GL_LINE_STIPPLE);
```

Задания

1. Примитивы

Изобразите точки, линии, треугольники, многоугольники в одном окне, как показано ниже.

Исходный файл

```
#include <windows.h>

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>



void CALLBACK resize(int width,int height)
{
```

```
   glViewport(0,0,width,height);
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   glOrtho(-5,5, -5,5, 2,12);
   gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
   glMatrixMode( GL_MODELVIEW );
}



void CALLBACK display(void)
{
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

 glPointSize(2);
  glBegin(GL_POINTS);
      glColor3d(1,0,0);
      glVertex3d(-4.5,4,0); // первая точка

     glColor3d(0,1,0);
      glVertex3d(-4,4,0);   // вторая точка

     glColor3d(0,0,1);     //третья
      glVertex3d(-3.5,4,0);
  glEnd();

  glPointSize(5);
  glBegin(GL_POINTS);
      glColor3d(1,0,0);
      glVertex3d(-2,4,0); // первая точка

     glColor3d(0,1,0);
      glVertex3d(-1,4,0);   // вторая точка

     glColor3d(0,0,1);     //третья
      glVertex3d(0,4,0);
  glEnd();

  glPointSize(10);
  glEnable(GL_POINT_SMOOTH);
  glBegin(GL_POINTS);
      glColor3d(1,0,0);
      glVertex3d(2,4,0); // первая точка

     glColor3d(0,1,0);
```

```
      glVertex3d(3,4,0);   // вторая точка

     glColor3d(0,0,1);     //третья
      glVertex3d(4,4,0);
  glEnd();
  glDisable(GL_POINT_SMOOTH);


glLineWidth(1);
 glBegin(GL_LINES);

  glColor3d(1,0,0);
  glVertex3d(-4.5,3,0);
  glVertex3d(-3,3,0);

  glColor3d(0,1,0);
  glVertex3d(-3,3.3,0);
  glVertex3d(-4,3.4,0);

 glEnd();


 glLineWidth(3);

 glBegin(GL_LINE_STRIP);
  glColor3d(1,0,0);
  glVertex3d(-2.7,3,0);
  glVertex3d(-1,3,0);

  glColor3d(0,1,0);
  glVertex3d(-1.5,3.3,0);
  glColor3d(0,0,1);
  glVertex3d(-1,3.5,0);

 glEnd();


 glLineWidth(5);
 glEnable(GL_LINE_SMOOTH);
 glEnable(GL_LINE_STIPPLE);
 glLineStipple(2,58360);


 glBegin(GL_LINE_LOOP);
```

```
  glColor3d(1,0,0);
  glVertex3d(1,3,0);
  glVertex3d(4,3,0);

  glColor3d(0,1,0);
  glVertex3d(3,2.7,0);
  glColor3d(0,0,1);
  glVertex3d(2.5,3.7,0);

  glEnd();

 glDisable(GL_LINE_SMOOTH);
 glDisable(GL_LINE_STIPPLE);



glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
 glBegin(GL_TRIANGLES);
  glColor3d(1,0,0);
  glVertex3d(-4,2,0);
  glVertex3d(-3,2.9,0);
  glVertex3d(-2,2,0);
 glEnd();


glLineWidth(2);
 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
 glBegin(GL_TRIANGLE_STRIP);
  glColor3d(0,1,0);
  glVertex3d(1,2,0);
  glVertex3d(0,2.9,0);
  glVertex3d(-1,2,0);
  glVertex3d(0,1.1,0);
 glEnd();


 glEnable(GL_LINE_STIPPLE);
 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
 glBegin(GL_TRIANGLE_FAN);
  glColor3d(0,0,1);
  glVertex3d(4,2,0);
  glVertex3d(2.6,2.8,0);
  glVertex3d(2,2,0);
  glVertex3d(3,1.1,0);
 glEnd();
```

```
 glDisable(GL_LINE_STIPPLE);

  auxSwapBuffers();

}



void main()
{
      float pos[4] = {3,3,3,1};
      float dir[3] = {-1,-1,-1};

    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH |
AUX_DOUBLE );
    auxInitWindow( "Primitives" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);


    glEnable(GL_ALPHA_TEST);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);

      auxMainLoop(display);
}
```

2. Многогранники

Постройте каркасные многогранники с помощью каркасных треугольников, многоугольников и линий.

Резюме

Ну вот, вы еще на один шаг продвинулись в изучение библиотеки OpenGL. Теперь вы имеете представление о том, как рисовать элементарные фигуры. Из примитивов вы можете составить фигуры любой сложности.

Лабораторная работа 3

ПОЛИГОНАЛЬНАЯ АППРОКСИМАЦИЯ ПОВЕРХНОСТЕЙ

1. Векторы нормали

Вектором нормали к поверхности в данной точке называется вектор перпендикулярный этой поверхности. Векторы нормали определяют ориентацию поверхности в пространстве, в частности её расположение относительно источников света. Эти векторы OpenGL использует для расчётов количества света, получаемого объектом в его вершинах.

Нормаль может быть одинаковой в нескольких вершинах. В каких-либо иных точках, кроме вершин,
OpenGL задавать вектор нормали не позваляет. Для указания вектора нормали используется одна из функций glNormal*(), имеющие формат:
void glNormal3{bsidf}(TYPE nx,TYPE ny, TYPE nz);
void glNormal3{bsidf}v(const TYPE* v);

В качестве параметров этой функции передаются координаты нормали. Этот вектор будет присваиваться всем последующим вершинам. Если в разных вершинах направление нормали разное, то его необходимо указывать перед рисованием каждой вершины, например:

```
glBegin( GL_POLYGON );
        glNormal3fv( n0 );
        glVertex3fv( v0 );
        glNormal3fv( n1 );
        glVertex3fv( v1 );
        glNormal3fv( n2 );
        glVertex3fv( v2 );
        glNormal3fv( n3 );
        glVertex3fv( v3 );
glEnd();
```

2. Пример построения икосаэдра

Икосаэдр - это правильный многогранник, имеющий 12 вершин и 20 граней (каждая грань - это равносторонний треугольник). Икосаэдр можно считать грубой аппроксимацией сферы. В фрагменте программы 3.1.1a в массивах заданы вершины икосаэдра и списки вершин каждой грани. После описания и инициализации массивов приведён цикл рисования этих граней.

Фрагмент программы 3.1.1a.

```
const double X = .525731112119133606;
const double Z = .850650808352039932;

// Координаты вершин икосаэдра
double vdata[12][3] = {
```

```
        {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
        {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
        {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};

// Правило построения граней икосаэдра из вершин vdata
int tindices[20][3] = {
        {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
        {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
        {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
        {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}

for (int i = 0; i<20; i++){
// Здесь могут быть вызовы функций для задания цвета
        glBegin( GL_TRIANGLES );
          glVertex3dv( &vdata[tindices[i][0]][0] );
          glVertex3dv( &vdata[tindices[i][1]][0] );
          glVertex3dv( &vdata[tindices[i][2]][0] );
        glEnd();
}
```

Конец фрагмента программы 3.1.1а.

Константы X и Z выбраны такими, чтобы расстояние от начала координат до каждой вершины икосаэдра равнялась 1.0 . Координаты 12 вершин хранятся в массиве vdata[][]: 0-я вершина имеет координаты {-X, 0.0, Z}, 1-я - {X, 0.0, Z} и т.д. Массив tindices[][] задаёт правила построения треугольных граней из вершин. Например, первый треугольник формируется из 0б 4 и 1-й вершин. Вершины треугольников в массиве tindices перечислены так, чтобы все грани имели одинаковую ориентацию.

В строке с комментарием о цветовой информации можно вызвать функцию, задающую разный цвет для каждой (i-й) грани. Если всем граням присвоить одинаковый цвет, то на изображении они будут сливаться. При закраске граней одним цветомнеобходимо задавать нормали вершин и применять направленное освещение.

Вычисление нормалей к граням икосаэдра

Для применения направленного освещения следует задать вектор нормали для каждой вершины икосаэдра. На плоских гранях икосаэдра вектор нормали одинаков у всех трёх вершин (это нормаль к плоскости этой грани). Следовательно, для каждого из трёх вершин нормаль надо задавать только один раз. Фрагмент программы 3.1.1б можно разместить в тех строках фрагмента 3.1.1а, которые отмечены комментарием "вызовы функций для задания цвета".

Фрагмент программы 3.1.1б.

```
double d1[3], d2[3], norm[3];
        for ( int j = 0; j < 3; j++ )
        {
                d1[j] = vdata[tindices[i][1]][j] - vdata[tindices[i][0]][j];
                d2[j] = vdata[tindices[i][2]][j] - vdata[tindices[i][0]][j];
        }
        normcrossprod( d2, d1, norm );
        glNormal3dv( norm );
```

Конец фрагмента программы 3.1.1б.

Функция normcrossprod() вычисляет нормированное векторное произведение двух векторов (см. фрагмент 3.1.1в).

Фрагмент программы 3.1.1в.

```
void normalize( double v[3] )
{
        double d = sqrt( v[0]*v[0]+v[1]*v[1]+v[2]*v[2] );
        if ( d == 0.0 )
   {
                // Ошибка: вектор нулевой длины
                return;
   }

        v[0] /= d; v[1] /= d; v[2] /= d;
}


// Вычисление векторного произведения out=v1xv2
void normcrossprod(const double v1[3], const double v2[3], double out[3])
{
        out[0] = v1[1]*v2[2] - v1[2]*v2[1];
        out[1] = v1[2]*v2[0] - v1[0]*v2[2];
        out[2] = v1[0]*v2[1] - v1[1]*v2[0];
        normalize(out);
}
```

Конец фрагмента программы 3.1.1в.

Нормали вершин не обязательно вычислять ка нормаль к плоскости грани. Способ вычмсления нормалей зависит от решаемой задачи. Допустим, что надо использовать икосаэдр

```
//
// Программа 3.1
//
// Рисование икосаэдра в одном из четырех режимов, различающихся направлением
// нормалей вершин и количеством цветов для закраски граней. Режимы отображения
// переключаются циклически по нажатию пробела.
//
#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void CALLBACK resize( int width, int height );
void CALLBACK change_display_mode();
void CALLBACK display_main();
void display_without_normals_single_color();
void display_without_normals_mult_colors();
void display_with_icosahedron_normals();
void display_with_sphere_normals();
void normalize( double v[3] );
```

```
void normcrossprod(const double v1[3], const double v2[3], double out[3]);


// Константы для задания координат вершин икосаэдра так, чтобы они
// лежали на поверхности единичной сферы
const double X = .525731112119133606;
const double Z = .850650808352039932;

// Координаты вершин икосаэдра
double vdata[12][3] = {
        {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
        {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
        {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};

// Правило построения граней икосаэдра из вершин vdata
int tindices[20][3] = {
        {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
        {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
        {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
        {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}
};


// Текущий режим отображения икосаэдра (0, 1, 2 или 3)
int display_mode = 0;


void main()
{
        // Создание экранного окна
        auxInitDisplayMode( AUX_RGBA | AUX_DEPTH | AUX_DOUBLE );
        auxInitPosition( 50, 10, 400, 400);
        auxInitWindow( "Программа 3.1" );

        // Включение ряда параметров OpenGL
        glEnable( GL_ALPHA_TEST );              // Учет прозрачности
        glEnable( GL_DEPTH_TEST );              // Удаление невидимых поверхностей
        glEnable( GL_COLOR_MATERIAL );      // Синхронное задание цвета рисования
                                                                        // и цвета материала
объектов
        glEnable( GL_BLEND );                   // Разрешение смешения цветов
        glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

        glEnable( GL_LIGHTING );            // Учет освещения
        glEnable( GL_LIGHT0 );                  // Включение нулевого источника света

        // Задание положения и направления нулевого источника света
        float pos[4] = { 0, 5, 5, 1 };
        float dir[3] = { 0, -1, -1 };
        glLightfv( GL_LIGHT0, GL_POSITION, pos );
        glLightfv( GL_LIGHT0, GL_SPOT_DIRECTION, dir );
```

```
	// Регистрация обработчиков событий
	auxReshapeFunc( resize );
	auxKeyFunc( AUX_SPACE, change_display_mode );
	auxIdleFunc( display_main );

	// Вход в главный цикл GLAUX
	auxMainLoop( display_main );
}


void CALLBACK resize( int width, int height )
{
	glViewport( 0, 0, width, height );

	glMatrixMode( GL_PROJECTION );
	glLoadIdentity();
	gluPerspective( 60.0, (float)width/(float)height, 1.0, 20.0 );

	gluLookAt( 3,3,3, 0,0,0, 0,1,0 );

	glMatrixMode( GL_MODELVIEW );
	glLoadIdentity();
}


// Обработчик нажатия клавиши (пробела) для переключения режима отображения
void CALLBACK change_display_mode()
{
	display_mode = ( display_mode + 1 ) % 4;
}


// Основная функция отображения
void CALLBACK display_main()
{
	switch ( display_mode )
	{
		case 0 : display_without_normals_single_color(); break;
		case 1 : display_without_normals_mult_colors(); break;
		case 2 : display_with_icosahedron_normals(); break;
		case 3 : display_with_sphere_normals(); break;
	}
}


// Все грани закрашиваются одним цветом, нормали вершин не задаются
// (т.е. нормали всех вершинам равны по умолчанию (0,0,1))
void display_without_normals_single_color()
{
	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
        glColor3d( 0.8, 0.8, 0.8 );
        glNormal3d( 0, 0, 1 );
        for ( int i = 0; i < 20; i++ )
        {
                glBegin( GL_TRIANGLES );
                  glVertex3dv( &vdata[tindices[i][0]][0] );
                  glVertex3dv( &vdata[tindices[i][1]][0] );
                  glVertex3dv( &vdata[tindices[i][2]][0] );
                glEnd();
        }

        auxSwapBuffers();
}


// Каждая грань закрашивается своим цветом, нормали не задаются
// (т.е. все нормали равны (0,0,1)).
void display_without_normals_mult_colors()
{
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        // Цвета граней икосаэдра
        static double clr[20][3] = {
                {0.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 1.0, 0.0},
                {0.0, 0.0, 1.0}, {1.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {0.5, 0.5, 0.5},
                {1.0, 1.0, 1.0}, {0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.5, 0.5, 0.0},
                {0.0, 0.0, 0.5}, {0.5, 0.0, 0.5}, {0.0, 0.5, 0.5}, {0.8, 0.8, 0.8},
                {0.3, 0.3, 0.3}, {0.8, 0.0, 0.0}, {0.0, 0.8, 0.0}, {0.8, 0.8, 0.0}
        };

        glNormal3d( 0, 0, 1 );
        for ( int i = 0; i < 20; i++ )
        {
                glColor3dv( clr[i] );
                glBegin( GL_TRIANGLES );
                  glVertex3dv( &vdata[tindices[i][0]][0] );
                  glVertex3dv( &vdata[tindices[i][1]][0] );
                  glVertex3dv( &vdata[tindices[i][2]][0] );
                glEnd();
        }

        auxSwapBuffers();
}


// Все грани закрашиваются одним цветом, нормали задаются как
// перпенди-куляры к плоским граням икосаэдра.
void display_with_icosahedron_normals()
{
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        glColor3d( 0.8, 0.8, 0.8 );
```

```
        for ( int i = 0; i < 20; i++ )
        {
                double d1[3], d2[3], norm[3];
                for ( int j = 0; j < 3; j++ )
                {
                        d1[j] = vdata[tindices[i][1]][j] - vdata[tindices[i][0]][j];
                        d2[j] = vdata[tindices[i][2]][j] - vdata[tindices[i][0]][j];
                }

                normcrossprod( d2, d1, norm );

                glBegin( GL_TRIANGLES );
                  glNormal3dv( norm );
                  glVertex3dv( &vdata[tindices[i][0]][0] );
                  glVertex3dv( &vdata[tindices[i][1]][0] );
                  glVertex3dv( &vdata[tindices[i][2]][0] );
                glEnd();
        }

        auxSwapBuffers();
}


// Все грани закрашиваются одним цветом, нормали задаются как перпендикуляры к
// поверхности сферы, на которой лежат вершины икосаэдра
void display_with_sphere_normals()
{
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        glColor3d( 0.8, 0.8, 0.8 );
        for ( int i = 0; i < 20; i++ )
        {
                glBegin( GL_POLYGON );
                  glNormal3dv( &vdata[tindices[i][0]][0] );
                  glVertex3dv( &vdata[tindices[i][0]][0] );
                  glNormal3dv( &vdata[tindices[i][1]][0] );
                  glVertex3dv( &vdata[tindices[i][1]][0] );
                  glNormal3dv( &vdata[tindices[i][2]][0] );
                  glVertex3dv( &vdata[tindices[i][2]][0] );
                glEnd();
        }

        auxSwapBuffers();
}


// Функция нормировки вектора
void normalize( double v[3] )
{
        double d = sqrt( v[0]*v[0]+v[1]*v[1]+v[2]*v[2] );
        if ( d == 0.0 )
   {
```

```
                // Ошибка: вектор нулевой длины
                return;
    }

        v[0] /= d; v[1] /= d; v[2] /= d;
}


// Вычисление векторного произведения out=v1xv2
void normcrossprod(const double v1[3], const double v2[3], double out[3])
{
        out[0] = v1[1]*v2[2] - v1[2]*v2[1];
        out[1] = v1[2]*v2[0] - v1[0]*v2[2];
        out[2] = v1[0]*v2[1] - v1[1]*v2[0];
        normalize(out);
}
```

3. Повышение точности аппроксимации сферической поверхности

Двенадцатигранная аппроксимация сферы не слишком точна, и её изображение напоминает сферу только
при небольшом размере. Существует простой путь для увеличения точности аппроксимации. Допустим,
имеется вписанный в сферу икосаэдр. Разобьём каждую грань икосаэдра на четыре равносторонних
треугольника. Новые вершины будут лежать внутри сферы, поэтому их надо приподнять на поверхность
(умножить на такое число, чтобы их радиус-векторы стали равны 1).

```
//
//  программа 3.2
//
// Программа для рисования икосаэдральной аппроксимации сферы
// в виде многогранников с 20-ю, 80-ю и 320-ю гранями.
//
#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void CALLBACK resize( int width, int height );
void CALLBACK display();
void normalize( double v[3] );
void draw_triangle( double v1[3], double v2[3], double v3[3] );
void subdivide( double v1[3], double v2[3], double v3[3], long depth );


// Константы для задания координат вершин икосаэдра так, чтобы они
// лежали на поверхности единичной сферы
const double X = .525731112119133606;
const double Z = .850650808352039932;
```

```
// Координаты вершин икосаэдра
double vdata[12][3] = {
        {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
        {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
        {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};

// Правило построения граней икосаэдра из вершин vdata
int tindices[20][3] = {
        {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
        {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
        {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
        {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}
};


void main()
{
        // Создание экранного окна
        auxInitDisplayMode( AUX_RGBA | AUX_DEPTH | AUX_DOUBLE );
        auxInitPosition( 50, 10, 400, 400);
        auxInitWindow( " Программа 3.2" );

        // Включение ряда параметров OpenGL
        glEnable( GL_ALPHA_TEST );              // Учет прозрачности
        glEnable( GL_DEPTH_TEST );              // Удаление невидимых поверхностей
        glEnable( GL_COLOR_MATERIAL );      // Синхронное задание цвета рисования
                                                                        // и цвета материала
объектов
        glEnable( GL_BLEND );                   // Разрешение смешения цветов
        glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

        glEnable( GL_LIGHTING );            // Учет освещения
        glEnable( GL_LIGHT0 );                  // Включение нулевого источника света

        // Задание положения и направления нулевого источника света
        float pos[4] = { 0, 5, 5, 1 };
        float dir[3] = { 0, -1, -1 };
        glLightfv( GL_LIGHT0, GL_POSITION, pos );
        glLightfv( GL_LIGHT0, GL_SPOT_DIRECTION, dir );

        // Регистрация обработчиков событий
        auxReshapeFunc( resize );

        // Вход в главный цикл GLAUX
        auxMainLoop( display );
}


void CALLBACK resize( int width, int height )
{
```

```
	glViewport( 0, 0, width, height );

	glMatrixMode( GL_PROJECTION );
	glLoadIdentity();
	gluPerspective( 60.0, (float)width/(float)height, 1.0, 20.0 );

	gluLookAt( 0,3,6, 0,0,0, 0,1,0 );

	glMatrixMode( GL_MODELVIEW );
	glLoadIdentity();
}


void CALLBACK display()
{
	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

	glPushMatrix();

	glColor3d( 0.8, 0.8, 0.8 );

	// 20-ти-гранник
	glTranslated( -2.5, 0, 0 );
	for ( int i = 0; i < 20; i++ )
	{
		glBegin( GL_POLYGON );
		  glNormal3dv( &vdata[tindices[i][0]][0] );
		  glVertex3dv( &vdata[tindices[i][0]][0] );
		  glNormal3dv( &vdata[tindices[i][1]][0] );
		  glVertex3dv( &vdata[tindices[i][1]][0] );
		  glNormal3dv( &vdata[tindices[i][2]][0] );
		  glVertex3dv( &vdata[tindices[i][2]][0] );
		glEnd();
	}

	// 80-ти-гранник
	glTranslated( 2.5, 0, 0 );
	for ( i = 0; i < 20; i++ )
		subdivide( &vdata[tindices[i][0]][0],
				&vdata[tindices[i][1]][0],
				&vdata[tindices[i][2]][0], 1 );

	// 320-ти-гранник
	glTranslated( 2.5, 0, 0 );
	for ( i = 0; i < 20; i++ )
		subdivide( &vdata[tindices[i][0]][0],
				&vdata[tindices[i][1]][0],
				&vdata[tindices[i][2]][0], 2 );

	glPopMatrix();

	auxSwapBuffers();
```

```
}


// Функция нормировки вектора
void normalize( double v[3] )
{
        double d = sqrt( v[0]*v[0]+v[1]*v[1]+v[2]*v[2] );
        if ( d == 0.0 )
   {
                // Ошибка: вектор нулевой длины
                return;
   }

        v[0] /= d; v[1] /= d; v[2] /= d;
}


// Рисование треугольника с заданными вершинами, в предположении,
// что его вершины лежат на поверхности сферы
void draw_triangle( double v1[3], double v2[3], double v3[3] )
{
        glBegin( GL_POLYGON );
          glNormal3dv(v1); glVertex3dv(v1);
          glNormal3dv(v2); glVertex3dv(v2);
          glNormal3dv(v3); glVertex3dv(v3);
        glEnd();
}


// Рекурсивная функция разбиения треугольной грани полигональной
// сетки на сферической поверхности.
// Параметр depth задает количество разбиений.
void subdivide( double v1[3], double v2[3], double v3[3], long depth )
{
        double v12[3], v23[3], v31[3];

        if ( depth == 0 )
        {
                draw_triangle( v1, v2, v3 );
                return;
        }

        for ( int i = 0; i < 3; i++ )
        {
                v12[i] = ( v1[i]+v2[i] )/2;
                v23[i] = ( v2[i]+v3[i] )/2;
                v31[i] = ( v3[i]+v1[i] )/2;
        }

        normalize( v12 );
        normalize( v23 );
        normalize( v31 );
```

```
        subdivide( v1, v12, v31, depth-1 );
        subdivide( v2, v23, v12, depth-1 );
        subdivide( v3, v31, v23, depth-1 );
        subdivide( v12, v23, v31, depth-1 );
}
```

4. Плоскости отсечения

```
//
//программа 3.3
//
// Использование плоскостей отсечения для построения
// каркасного сферического сегмента
//
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void CALLBACK resize( int width, int height );
void CALLBACK display();


void main()
{
        auxInitDisplayMode( AUX_SINGLE | AUX_RGBA );
        auxInitPosition( 0, 0, 400, 400 );
        auxInitWindow( " Программа 3.3" );
        auxReshapeFunc( resize );
        auxMainLoop( display );
}


void CALLBACK resize( int width, int height )
{
        glViewport( 0, 0, width, height );

        glMatrixMode( GL_PROJECTION );
        glLoadIdentity();
        gluPerspective( 60.0, (float)width/(float)height, 1.0, 20.0 );

        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
}


void CALLBACK display()
{
        double eqn1[4] = {0.0, 1.0, 0.0, 0.0};          // y < 0
        double eqn2[4] = {1.0, 0.0, 0.0, 0.0};          // x < 0
```

```
        glClear( GL_COLOR_BUFFER_BIT );

        glColor3d( 1.0, 1.0, 1.0 );
        glPushMatrix();
        glTranslated( 0.0, 0.0, -5.0 );

        glClipPlane( GL_CLIP_PLANE0, eqn1 );
        glEnable( GL_CLIP_PLANE0 );
        glClipPlane( GL_CLIP_PLANE1, eqn2 );
        glEnable( GL_CLIP_PLANE1 );

        glRotated( 90.0, 1.0, 0.0, 0.0 );
        auxWireSphere( 1.0 );
        glPopMatrix();
        glFlush();
}
```

ЗАДАНИЯ.

1. Выполнить программу 3.1 для рисования икосаэдра в 4-х режимах (режимы циклически переключаются
пробелом):
 - все грани закрашиваются одним цветом, нормали вершин не задаются (т.е. нормали всех вершин
   равны по умолчанию (0, 0, 1));
 - каждая грань закрашивается своим цветом, нормали не задаются;
 - все грани закрашиваются одним цветом, нормали задаются как перпендиуляры к плоским граням
   икосаэдра;
 - все грани закрашиваются одним цветом, нормали задаются как перпендиуляры к поверхности сферы,
   на которой лежат вершины икосаэдра.

2. Выполнить программу 3.2, которая будет рисовать три варианта икосаэдральной аппроксимации сферы.
Затем попробуйте выполнить ещё один шаг разбиения для получения 1280 граней.

3. Выполнить программу 3.3. Изменить её так, чтобы с помощью одной плоскости отсечения программа
изображала нижнюю полусферу. Затем добавьте в свою программу функцию фоновой обработки и с её
помощью сделайте так, чтобы плоскость отсечения вращалась (функцией glRotated() перед glClipPlane()
и отсекала полусферу под разными углами.

Лабораторная работа 4

ЦВЕТ И ОСВЕЩЕНИЕ

1. Цветовая модель RGB

На мониторе компьютера видимые цвета формируются путём смешивания трёх основных цветов: красного
(Red), зелёного (Green) и синего (Blue). В памяти компьютера значение цвета хранится в виде трёх
чисел - компонентов R,G и B. К ним иногда добавляется четвёртый компонент A (прозрачность).
В OpenGL текущей цвет задается функциями glColor...().
  В OpenGL двумерное изображение для вывода на экран хранится в виде массива значений цветов,
соответствующих каждому пикселю изображения. Этот массив называется цветовым буфером. Значения
компонентов R,G и B лежат в диапазоне от 0.0 (минимальная интенсивность) до 1.0 (максимльная
интенсивность). Если в OpenGL включено освещение, то для вычисления цвета пикселя в цветовом
буфере производится ряд вычислений, в которых участвует цвет примитива, например, красный мяч
в ярком синем цвете будет выгляеть иначе, чем при белом цвете.

2. Задание способа окраски.
При рисовании примитивов можно задавать цвет в их вершинах. Цвет внутренних точек отрезка или
многоугольника вычисляется в соответствии с установленным способом закраски. При плоской
закраски отрезок или залитый многоугольник рисуется одним цветом, а при плавной закраске -
- различными цветами. Способ закраски задается с помощью функции:

void glShadeModel(Glenum mode);

где mode может быть GL_SMOOTH (плавная закраска, значение по умолчанию) или GL_FLAT (плоская
закраска).
  При плоской закраске примитив рисуется цветом первой вершины. При плавной закраске цвет каждой
вершины рассматривается независимо и цвет промежуточных точек рассчитывается интерполяцией в
цветовом пространстве (см. программу 4.1)

```
// Программа 4.1. Рисование треуголника с плавной закраской.
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

void triangle()
{
  glBegin( GL_TRIANGLES );
    glColor3f( 1.0, 0.0, 0.0 );
    glVertex2f( 5.0, 5.0 );
    glColor3f( 0.0, 1.0, 0.0 );
    glVertex2f( 25.0, 5.0 );
```

```
    glColor3f( 0.0, 0.0, 1.0 );
    glVertex2f( 5.0, 25.0 );
  glEnd();
}

void CALLBACK display()
{
  glClear( GL_COLOR_BUFFER_BIT );
  triangle();
  glFlush();
}

void CALLBACK resize( int w, int h )
{
  glViewport( 0, 0, w, h );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  if ( w <= h )
    gluOrtho2D( 0.0, 30.0, 0.0, 30.0*(float)h/(float)w );
  else
    gluOrtho2D( 0.0, 30.0*(float)w/(float)h, 0.0, 30.0 );
  glMatrixMode( GL_MODELVIEW );
}

void main()
{
  auxInitDisplayMode( AUX_SINGLE | AUX_RGBA );
  auxInitPosition( 0, 0, 400, 400 );
  auxInitWindow ( "Программа 4.1");
  glShadeModel( GL_SMOOTH );
  auxReshapeFunc( resize );
  auxMainLoop( display );
}
// Конец программы 4.1
```

3. Освещение в OpenGL.

В OpenGL источники света проявляют себя только при взаимодействии с поверхностями объектов, способными поглощать и отражать свет. Предполагается, что каждая поверхность изготовлена из материала с определёнными свойствами. Материал может излучать собственный свет (подобно передней повехности фары автомобиля), рассеивать падающий свет по всем направлениям, а также отражать некоторую часть падающего света по избранному направлению - подобно зеркалу или блестящей поверхности.

В результате в модели освещения OpenGL освещение рассматривается как состоящее из четырёх независимых компонентов: излучаемый, рассеянный, диффузионно отраженный и зеркально отражённый

свет. Каждый компонент описывается тройкой значений RGB. Все четыре компонента вычисляются
независимо, а затем складываются вместе и применяются для вычисления цвета пикселя в цветовом
буфере.

Цветовые значения световых компонентов имеют разный для источников света и для материалов.
Для источника света они определяют интенсивность излучения каждого из основных цветов. Например,
если R,G и B равны 1.0, то цвет будет белым, максимальной интенсивности. Если R=G=1 и B=0
(максимальный красный и зелёный, совсем нет синего), то свет будет выглядеть жёлтым.

Для материалов значения компонентов определяют долю отражённого света каждого цвета. Например,
если для материала задано R=1, G=0.5 и B=0, то этот материал будет отражать весь падающий
красный цвет, половину зелёного и совсем не будет отражать синий цвет. Допустим, что цвет одного
из компонентов источника равен (LR,LG,LB), а цвет соответствующего компонента материала равен
(MR,MG,MB). Тогда, если не учитывать другие эффекты отражения, глаз наблюдателя будет
воспринимать свет (LR*MR,LG*MG,LB*MB).

Аналогично, если есть два источника света - (R1,G1,B1) и (R2,G2,B2), то OpenGL сложит эти
компоненты и глаз наблюдателя будет воспринимать свет (R1+R2,G1+G2,B1+B2). Если одна из сумм
больше 1, то она уменьшится до 1 - максимально возможной интенсивности.

Пример рисования освещенной сферы.
Добавление освещения в сцену производится в следующем порядке:
- у всех объектов для каждой вершины задаётся вектор нормали. Эти векторы определяют ориентацию
  обьекта относительно источников света;
- задаются местоположение и свойства одного или нескольких источников света. Каждый источник
  необходимо включить;
- задаются параметры модели освещения, которые определяют уровень фонового рассеянного света и
  эффективную точку наблюдения (она используется при вычислениях освещения);
- задаются свойства материалов объектов сцены.

Выполнение этих действий показано в программе 4.2. Она рисует сферу, освещённую одним
источником света. Вызовы всех функций, имеющих отношение к освещению, вынесены в функцию
lightsInit().

```
// Программа 4.2. Рисование освещённой сферы.
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
```

```
void lightsInit()
{
  float mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  float mat_shininess[] = { 50.0 };
  float light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  float light_global_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f };

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glLightModelfv( GL_LIGHT_MODEL_AMBIENT, light_global_ambient );

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  glDepthFunc(GL_LEQUAL);
  glEnable(GL_DEPTH_TEST);
}

void CALLBACK display()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  auxSolidSphere(1.0);
  glFlush();
}

void CALLBACK resize( int width, int height )
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if ( width <= height )
            glOrtho( -1.5, 1.5, -1.5*(float)height/width,
             1.5*(float)height/width, -10.0, 10.0);
  else
     glOrtho( -1.5*(float)width/height,
             1.5*(float)width/height, -1.5, 1.5, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void main()
{
  auxInitDisplayMode( AUX_SINGLE | AUX_RGBA | AUX_DEPTH );
  auxInitPosition( 0, 0, 400, 400 );
  auxInitWindow( "Программа 4.2" );
  lightsInit();
  auxReshapeFunc( resize );
  auxMainLoop( display );
}
```

// Конец программы 4.2

Векторы нормали в вершинах объектов.

Нормали в вершинах объекта определяют его ориентацию относительно источников света. OpenGL
использует нормаль вершины, чтобы вычислить, сколько света от каждого источника попадает в
эту вершину. В программе 4.2 нормали для вершин явно не задаются, это делается внутри функции
auxSolidSphere().

Создание, расположение и включение источников света.

В программе 4.2 используется только один источник белого света. Его местоположение задаётся
функцией glLightfv(). В данном примере для нулевого источника света (GL_LIGHT0) используется
цвет по умолчанию (белый). Для изменения цвета источника надо вызвать функцию glLight*().
OpenGL позволяет включить до восьми источников света (у всех, кроме нулевого,по умолчанию
задан чёрный цвет). Эти источники можно располагать в любых местах - вблизи сцены (как
настольную лампу) или бесконечно далеко (как Солнце). Кроме того, можно управлять свойствами
светового пучка - сделать его узким, сфокусированным или широким. Необходимо учитывать, что
каждый источник света приводит к усложнению вычислений, поэтому производительность прогаммы
зависит от включённых источников.
После задания характеристик источнков света требуется влючить каждый источник функцией
glEnable(). Предварительно необходимо вызвать эту функцию с параметром GL_LIGHTING, чтобы
подготовить OpenGL к выполнению расчётов освещения.

Выбор модели освещения.

Функция glLightModel*() предназначена для задания параметров модели освещения. В программе 4.2
явно задаётся только один подобный параметр - яркое фоновое рассеянное освещение. Модель
освещения также определяет, где следует располагать наблюдателя: на бесконечно большом
расстоянии или близко от сцены, а также не должны ли расчёты освещения выполняться по-разному
для передних и задних сторон объектов сцены. В программе 4.2 используются параметры модели по

умолчанию - бесконечно удалённый наблюдатель и одностороннее освещение. Использование локального
наблюдателя существенно усложняет расчёты, т.к.OpenGL должна будет вычислять угол между точкой
наблюдения и каждым объектом. Для бесконечно далёкого наблюдателя этот угол не учитывается,
поэтому результаты слегка теряют реалистичность. Поскольку в этом примере внутренняя поверхность
сферы никогда не видна, то далее будет достаточно одностороннего освещения.

Задание свойств материалов для объектов сцены.

Для материала можно независимо задавать излучаемый, рассеянный, диффузный и зеркальный
компоненты, а также блеск. В программе 4.2 с помощью функции glLightfv() задаются два свойства
материала - цвет зеркального компонента (GL_SPECULAR) и блеск (GL_SHININESS).

4. Создание источников света.

У источников света существуют набор свойств: цвет разных компонентов света, местоположение и
направление излучения. Для задания всех свойств источника света используется функция glLight*():

void glLight{if}[v](Glenum light, Glenum pname, TYPE param);

Параметр light - это номер источника света (от GL_LIGHT0 до GL_LIGHT7). Параметр pname
обозначает изменяемое свойства (табл.). Новое значение свойства pname передаётся в параметре
param.

Таблица. Константы, обозначающие свойства источника света и значения этих свойств

Свойство	По умолчанию	Смысл
GL_AMBIENT	(0.0,0.0,0.0,1.0)	Рассеянный компонент света
GL_DIFFUSE	(1.0,1.0,1.0,1.0)	Диффузный компонент света
GL_SPECULAR	(1.0,1.0,1.0,1.0)	Зеркальный компонент света
GL_POSITION	(0.0,0.0,1.0,0.0)	Координаты источника света (x,y,z,w)
GL_SPOT_DIRECTION	(0.0,0.0,-1.0)	Направление источника светы (x,y,z)
GL_SPOT_EXPONENT	0.0	Показатель экспонциального затухания прожектора
GL_SPOT_CUTOFF	180.0	Угол отсечки прожектора
GL_CONSTANT_ATTENUATION	1.0	Постоянный коэффициент затухания
GL_LINEAR_ATTENUATION	0.0	Линейный коэффициент затухания
GL_QUADRATIC_ATTENUATION	0.0	Квадратичный коэффициент затухания

Значение по умолчанию для свойств GL_DIFFUSE и GL_SPECULAR в табл. верны только для GL_LIGHT0.
Для остальных источников света значения этих компонентов по умолчанию равны (0.0,0.0,0.0,1.0)
(т.е. остальные источники света по умолчанию ничего не излучают даже во включённом состянии).

Рассмотрим на примере использование функции glLightfv(), показывающий, что для задания
свойств GL_AMBIENT,GL_DIFFUSE, GL_SPECULAR необходимы отдельные вызовы этой функции:

```
float light_ambient[]={0.0,0.0,0.0,1.0};
float light_diffuse[]={1.0,1.0,1.0,1.0};
float light_specular[]={1.0,1.0,1.0,1.0};
float light_position[]={1.0,1.0,1.0,1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse};
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular};
glLightfv(GL_LIGHT0, GL_POSITION, light_position};
```

Цвет

OpenGL позволяет указать для каждого источника света цветовые значения трёх компонентовсвета -
- GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR. Компонент GL_AMBIENT определяет вклад источника в общее
рассеянное освещение. Как видно из таблицы , по умолчанию рассеянный компонент отсутствует, т.е.
GL_AMBIENT равен (0.0,0.0,0.0,1.0). В программе 4.2 задаётся фоновое яркое белое рассеянное
освещение. Если изменить его на синее, то сфера будет выглядеть иначе:

```
float light_global_ambient[]={0.0,0.0,1.0,1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_global_ambient};
```

Местоположение и затухание

Источник света можно расположить или бесконечно далеко, или вблизи сцены. Источники первого типа
называются направленными источниками света. Источник второго типа - локальные. Пример локального
источника света - настольная лампа. В программе 4.2 создаётся направленный источник света:

```
float light_position[]={1.0,1.0,1.0,0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position};
```

В качестве значения для свойства GL_POSITION указывается массив (x,y,z,w). Если 4-я координата
w=0, то источник света будет считаться направленным и его направление будет задаваться вектором

(x,y,z). По умолчанию GL_POSITION равно (0,0,1,0), т.е. это направленный источник света,
излучающий свет вдоль отрицательной оси Z.

Если w не равно 0, то источник света будет считаться локальным и вектор (x,y,z) задаёт местоположение источника света. Локальный источник излучает свет по всем направлениям, но можно
ограничить конус излучения, сделав источник прожектором.

ИЗМЕНЕНИЕ МЕСТОПОЛОЖЕННИЯ ИСТОЧНИКОВ СВЕТА

Местоположение и направление источников света, как и координаты вершин, в OpenGL могут
подвергаться модельным преобразованиям. Точнее, при вызове функции glLight*() для задания
местоположения или направления источника света переданные координаты преобразуются в
соответсвии с текущей видовой матрицей. Рассмотрим два варианта расположения источника света:
- источник света находится в фиксированной точке;
- источник света перемещается относительно неподвижного объекта.

В прстейшем случае (например, в программе 4.2) положение (или направление) источника света
не изменяется. Координаты (или направление) такого источника определяются сразу после выбора
видовой матрицы:

```
glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if ( width <= height )
            glOrtho( -1.5, 1.5, -1.5*(float)height/width,
             1.5*(float)height/width, -10.0, 10.0);
  else
     glOrtho( -1.5*(float)width/height,
             1.5*(float)width/height, -1.5, 1.5, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

// Далее, внутри функции lightsInit()
float light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Итак, сначала задаются оконное преобразование и проекционная матрица. Далее в качестве видовой
матрицы загружается единичная и затем определяется направление нулевого источника света.
Поскольку используется единичная видовая матрица, то направление источника координат будет
(1.0, 1.0, 1.0).

Предположим, что необходимо поворачивать и/или сдвигать локальный источник света так, чтобы он

двигался относительно неподвижного объекта. Для этого надо задавать координаты источника после
видовых преобразований точно также, как это делается при размещении объектов сцены. Ниже прведена
функция display(), в которой источник света поворачивается на угол spin вокруг неподвижного тора
(переменная spin изменяется в фоновой функции или в обработчике событий):

```
void CALLBACK display()
{
 float light_position[]={0.0, 0.0, 1.5, 1.0}  ;
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glPushMatrix();
   glTranslatef(0.0, 0.0, -5.0);
   glPushMatrix();
     glRotated((doubl)spin, 1.0, 0.0, 0.0);
     glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   glPopMatrix();
   auxSolidTorus(0.275, 0.85);
 glPopMatrix();
 glFlush();
}
```

```
// Программа 4.3
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

static int spin = 0;


void CALLBACK moveLight( AUX_EVENTREC* )
{
  spin = (spin + 30) % 360;
}


void lightsInit()
{
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  glDepthFunc(GL_LEQUAL);
  glEnable(GL_DEPTH_TEST);
}


void CALLBACK display()
{
  float position[] = { 0.0, 0.0, 1.5, 1.0 };
```

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glPushMatrix();
    glTranslatef(0.0, 0.0, -5.0);

    glPushMatrix();
      glRotated((double) spin, 1.0, 0.0, 0.0);
      glRotated(0.0, 1.0, 0.0, 0.0);
      glLightfv(GL_LIGHT0, GL_POSITION, position);
      glTranslated(0.0, 0.0, 1.5);
      glDisable(GL_LIGHTING);
      glColor3f(0.0, 1.0, 1.0);
      auxWireCube(0.1);
      glEnable(GL_LIGHTING);
    glPopMatrix();

    auxSolidTorus(0.275, 0.85);
  glPopMatrix();

  glFlush();
}


void CALLBACK resize( int width, int height )
{
  glViewport( 0, 0, width, height );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  gluPerspective(40.0, (float)width/(float)height, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
}


void main()
{
  auxInitDisplayMode( AUX_SINGLE | AUX_RGBA | AUX_DEPTH );
  auxInitPosition( 0, 0, 400, 400 );
  auxInitWindow( "Программа 4.3" );
  lightsInit();
  auxMouseFunc( AUX_LEFTBUTTON, AUX_MOUSEDOWN, moveLight );
  auxReshapeFunc( resize );
  auxMainLoop( display );
}
// Конец программы 4.3
```

ЗАДАНИЕ.

1. С помощью программы 4.1 выясните различие между плавной и плоской закраской треугольника.

2. Проверьте, как будет работать при каждом из трёх изменений:
   - замените направленный источник света на локальный белый источник;

- задайте синееифоновое освещение максимальной интенсивности;
- к направленному белому источнику света добавьте ещё один цветной прожектор.

3. Измените программу 4.3, показывающую тор, освещаемый вращающимся вогруг него локальным
источником света, следующим образом:
- сделайте так, чтобы источник света не вращался вокруг тора, а перемещался относительно
него прямолинейно;

Указание. В функции display() вместо glRotated()примените функцию glTranslated() и заведите
вместо spin подходящую переменную для хранения значения смещения источника света.
- задайте коэффициент затухания так, чтобы интенсивность света уменьшалась по мере удаления
от источника.

Указание. Коэффициент затухания задаётся с помощью функции glLight*().

Лабораторная работа 5. Введение в программирование с использованием OpenGL.

Задания.

1. Реализовать простейший графический редактор, позволяющий с помощью мыши рисовать:
а) цветные точки (размер и цвет точки задается, например, нажатием клавиш ‘1’-‘9’),
б) цветные ломаные (ломаная заканчивается, например, при нажатии Enter; цвет задается для каждой ломаной),
в) цветные ломаные (цвет задается для каждой точки ломаной),
г) цветные многоугольники (задается цвет каждого многоугольника),
д) цветные многоугольники (задается цвет каждой вершины).
2. Реализовать простейший графический редактор, рисующий многоугольники из предопределенного набора. Мышью задается положение многоугольника. Предусмотреть возможность выбора цвета. При рисовании использовать смешение цветов посредством логических операций:
а) and и not and,
б) or и not or,
в) xor и not xor.

Указания.
1.а. См. функцию glPointSize.
2. См. функции glEnable(GL_COLOR_LOGIC_OP), glLogicOp(...).
Все задания, касающиеся отрисовки многоугольников, должны в тестах иметь выпуклый многоугольник, невыпуклый многоугольник, перекрещивающийся многоугольник (неправильно заданный).

Лабораторная работа 6. Задание полигональных моделей объектов. Трехмерная визуализация с использованием OpenGL.

Цель работы: ознакомиться с основным способом задания полигональных моделей – методом тиражирования сечений; ознакомиться со средствами трехмерной визуализации в OpenGL (источники света, свойства материалов, текстуры).

Задания.

№	Путь	Сечение	Способ тиражирования сечения
1	прямой	выпуклый многоугольник	без изменения
2	прямой	невыпуклый многоугольник	без изменения
3	прямой	выпуклый	с поворотом

		многоугольник	
4	прямой	невыпуклый многоугольник	с поворотом
5	прямой	выпуклый многоугольник	с масштабированием
6	прямой	невыпуклый многоугольник	с масштабированием
7	кривой	выпуклый многоугольник	без изменения
8	кривой	невыпуклый многоугольник	без изменения
9	кривой	выпуклый многоугольник	с поворотом
10	кривой	невыпуклый многоугольник	с поворотом
11	кривой	выпуклый многоугольник	с масштабированием
12	кривой	невыпуклый многоугольник	с масштабированием
13	окружность	невыпуклый многоугольник	без изменения

Требования.

1. Входные данные – текстовый файл с описанием сечения, пути его тиражирования, положения сечений вдоль пути с углом поворота или коэффициентом масштабирования.
2. Сечение – плоское, путь – трехмерный.
3. Сечение всегда перпендикулярно пути.
4. Положение сечения задается длиной пути от начальной вершины в процентах.
5. Нормали в вершинах объекта должны совпадать с нормалями аппроксимируемой поверхности.
6. При визуализации использовать источники света, свойства материалов и/или текстуры.

Указание.
Для разбивки невыпуклых многоугольников на выпуклые можно использовать процедуру триангуляции из дополнительной библиотеке GLU. См. функции gluNewTess/gluDeleteTess; gluBeginTessPolygon/gluEndTessPolygon, gluBeginTessContour/gluEndTessContour, gluTessVertex, gluTessCallback.

Лабораторная работа 7
Построение изображение методом трассировки лучей.

Цель: ознакомиться с основными аспектами метода трассировки лучей.

Варианты заданий.

№	Тип объектов	Возможности трассировщика	Текстурирование	Охватывающие оболочки	Модель освещения
1	1	2	1	1	1
2	1	3	1	1	1
3	1	1	2+3	1	1
4	2	1	1	1	2
5	2	2	1	1	1
6	3	1	1	1+2	1
7	3	1	1	1+3	1
8	3	1	1	1+4	1
9	2	3	2	1	1
10	2	1	2	1	2
11	3	2	1	4	1
12	1+2+3	1	1	4	3

Типы объектов.

1. Базовые: сфера и конический цилиндр.
2. Составные из сфер, цилиндров и прямоугольных параллелепипедов.
3. Сеточные.

Возможности трассировщика.

1. Трассировка только первичных лучей (от наблюдателя до первого пересечения с объектом).
2. Трассировка первичных лучей и расчет теней.
3. Трассировка первичных и вторичных (отражения и преломления) лучей.

Текстурирование.

1. Без текстур.
2. Стандартное наложение изображения на объект (текстура модулирует диффузную составляющую отраженного света).
3. Рельефная текстура, имитирующая неровности поверхности (текстура модулирует возмущения нормали).

Охватывающие оболочки.

1. Не используются.
2. Проекционные.
3. Трехмерные в канонической системе координат.
4. Трехмерные в глобальной системе координат.

Модель освещения.

1. Фонга.
2. Кука-Торренса.

Указания.

Рельефная текстура.

Пусть имеется параметрически заданная поверхность $P(u,v)$. Необходимо изобразить поверхность

$\tilde{P}(u,v) = P(u,v) + texture(u,v) \cdot \bar{n}(u,v)$,

где $texture(u,v)$ - текстурная функция, $\bar{n}(u,v)$ - вектор нормали к поверхности.

Для этого нужно знать нормаль в каждой точке поверхности $\tilde{P}(u,v)$:

$\tilde{\bar{n}}(u^*,v^*) = \bar{n}(u^*,v^*) + \bar{d}(u^*,v^*)$, где

$\bar{d}(u^*,v^*) = \left(\left(\bar{n} \times P_v\right) \cdot texture_u - \left(\bar{n} \times P_u\right) \cdot texture_v\right)_{u=u^*, v=v^*}$

$P_u = \left(\frac{\partial P_x}{\partial u}, \frac{\partial P_y}{\partial u}, \frac{\partial P_z}{\partial u}\right)$, $P_v = \left(\frac{\partial P_x}{\partial v}, \frac{\partial P_y}{\partial v}, \frac{\partial P_z}{\partial v}\right)$,

$texture_u = \frac{\partial\left(texture\right)}{\partial u}$, $texture_v = \frac{\partial\left(texture\right)}{\partial v}$.

Модель освещения.

Коэффициенты преломления для некоторых металлов (модель Кука-Торренса).

$\eta = \frac{1+\sqrt{F_0}}{1-\sqrt{F_0}}$

Металл	F_0 (красный)	F_0 (зеленый)	F_0 (синий)
Золото	0.989	0.876	0.349
Серебро	0.950	0.930	0.880
Медь	0.755	0.490	0.095
Железо	0.530	0.505	0.480

6 семестр

Цели лабораторных работ:
- изучение наиболее распространенных алгоритмов решения задач с использованием сложных структур данных.

Содержание работ одинаково для всех лабораторных работ.

Язык программирования – язык Си/Си++.

Порядок выполнения работы

1. Изучить необходимый материал по теме лабораторной работы, руководствуясь методическими указаниями к теме.
2. Получить допуск к работе, ответив на контрольные вопросы.
3. Разработать структуру данных для представления основных исходных данных и/или результатов для выданного преподавателем варианта задания.
4. Разработать алгоритм и написать программу решения задачи.
5. Подготовить набор тестов и отладить программу.
6. Оформить отчет по лабораторной работе. Отчет по работе включает следующие пункты:
    1. Условие задачи
    2. Анализ задачи
    3. Структура основных входных и выходных данных
    4. Алгоритм решения задачи
    5. Текст программы
    6. Набор тестов
    7. Результаты отладки и их анализ
7. Защитить лабораторную работу. Защита состоит в обсуждении выбранной структуры данных и алгоритма решения задачи, ответе на контрольные вопросы, решении контрольных задач.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 1

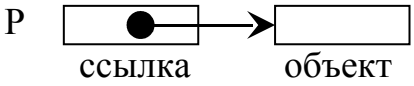
ЛИНЕЙНЫЕ ОДНОНАПРАВЛЕННЫЕ СПИСКИ

Цель работы: изучить тип указатель; получить навыки в организации и обработке однонаправленных списков.

1. Методические указания

1.1. Тип указатель

Значением типа указатель является адрес участка памяти, выделенного для объекта конкретного типа (ссылка на объект). Связь указателя P с объектом можно изобразить следующим образом:



По указателю осуществляется обращение (доступ) к объекту.

Определение переменной типа указатель:

type *имя_указателя ;

где type – обозначение типа, на который будет указывать переменная с именем (идентификатором) имя_указателя. Символ ‘*’ - это унарная операция раскрытия ссылки (операция разыменования, операция обращения по адресу, операция доступа по адресу), но в данном контексте служит признаком типа указатель.

При определении переменной-указателя можно выполнить инициализацию:

type *имя_указателя инициализатор ;

Инициализатор имеет две формы записи, поэтому допустимо:

type *имя_указателя = инициализирующее_ выражение ;

type *имя_указателя (инициализирующее_ выражение) ;

Инициализирующее_ выражение – это константное выражение, которое может быть задано указателем, уже имеющим значение или выражением, позволяющим получить адрес объекта с помощью операции & - получение адреса операнда.

Пример 1:

```
char cc = ‘f’, *p;     //Определение символьной переменной cc  и неинициализи-
                       // рованного указателя на  объект типа char
int *p1, *p2;          //Определение двух неинициализированных указателей
                       // p1 и p2 на объекты типа int
char *pc = &cc;        //Инициализированный указатель на объект типа char
char *ptr (NULL);      //Нулевой указатель на объект типа char
int *mas [10];         //Массив из 10 указателей на объекты типа int
int (*point) [10];     //Указатель на массив из 10 элементов типа int
struct list *n, *pr;   //Два указателя n и pr на объекты именованного структурного
                       // типа list. Определение  типа list должно либо
                       // предшествовать данному определению, либо находиться в
                       // пределах области действия данного определения
  struct list          //Определение переменной line структурного
  {                    //типа list, один из элементов которой типа
  char b; int *k;      //указатель на объект типа int
          } line;
int **pp = &p1;        //Указатель pp – это указатель на указатель. Он инициирован
                       //значением адреса указателя p1 на объект целого типа. Это
```

// допустимо, так как указатель – это тоже объект в памяти

Переменной типа указатель (в дальнейшем просто указатель) можно задать значение:

- присваивая ей адрес объекта с помощью операции & (тип объекта должен быть тот же, что и тип объекта, на который ссылается указатель);
- присваивая ей значение другой переменной или выражения типа указатель (типы объектов, на которые они ссылаются должны быть одинаковыми);
- с помощью операции new.

Над указателями допускается выполнение следующих операций:

- присваивание;
- получение адреса;
- сложение и вычитание;
- инкремент и декремент;
- операции отношения.

Все операции применимы к указателям на объекты одного типа и к указателю и целой константе, в противном случае требуется явное преобразование типов. Операция сложения допустима только для указателя и целочисленного значения. Вычитая два указателя одного типа, можно получить “расстояние” между двумя участками памяти. “Расстояние“ определяется в единицах, кратных размеру памяти объекта того типа, к которому отнесен указатель.

С помощью переменной типа указатель и операции разыменования * можно обращаться к объекту, на который она ссылается.

Пример 2 (с учетом описаний в примере 1) :

```
int a = 10, b = 20, c = 30;
int x[3][10];
ptr = pc;          //Указатели ptr и pc ссылаются на одну и ту же переменную
                   // cc типа char
*ptr = '+';        //Объект, на который указывают ptr и pc (т.е. cc), получил
                   // новое значение
p1 = &a;           //Указатель p1 получил значение адреса целой переменной a
p2 = &b;           //Указатель p2 – значение адреса целой переменной b
mas[0] = p1;       //Элемент массива mas[0] типа указатель получил то же
значение,
                   //что и указатель p1
mas[1] = &b;       //Указатель mas[1] получил значение адреса b
*mas[0] = 15;      //Переменная типа int, на которую указывают и p1 и mas[0],
                   //т.е. переменная a получила новое значение
*p2 = 25;          //Переменная b получила новое значение
point = &x[0];     //Указатель point получил значение адреса нулевого
элемента
                   //массива x, т.е. адреса элемента x[0][0]
```

```
point[0] = a;       //Элементу массива с индексом 0, на который указывает
point
                    //присваивается значение  переменной a
*(point + 1) = b;  //Элементу массива, на который указывает point + 1,
                    //т.е. элементу x[0][1] присваивается значение b
n = &line;          //Указателю n присвоен адрес структуры line
pr = n;             //Указатели n и pr ссылаются на одну  переменную
(*n).b = *ptr;      //Элементу b структуры, на которую ссылается n,
присваивается
                   //значение объекта, на который ссылается ptr,
                   //т.е. line.b получила значение '+'
pr -> k = p1;      //Элементу k структуры, на которую ссылается pr
присваивается
                   //значение указателя p1, т.е. line.k получила значение адреса
                   //целой переменной a
```

1.2. Понятие динамической структуры данных

Описывая в программе с помощью определений структуру данных, мы создаем так называемые статические объекты или заранее определяемые объекты. Статическая структура данных характеризуется тем что:

- она имеет имя, которое используется для обращения к ней;
- ей выделяется память в процессе трансляции программы;
- количество элементов сложной структуры (размерность) фиксировано при ее описании;
- размерность структуры не может быть изменена во время выполнения программы.

Динамическая структура данных характеризуется тем что:

- она не имеет имени;
- ей выделяется память в процессе выполнения программы;
- количество элементов структуры может не фиксироваться;
- размерность структуры может меняться в процессе выполнения программы;
- в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Каждой динамической структуре данных сопоставляется статическая переменная типа указатель (ее значение - адрес этого объекта) посредством которой осуществляется доступ к динамической структуре. Для создания динамического объекта используется унарная операция new:

```
new  имя_типа
```

или

```
new  имя_типа  инициализатор
```

Операция new позволяет выделить свободный участок в основной памяти (динамической), размеры которого соответствуют типу данных, определяемому именем типа. В случае успешного выполнения операция new

возвращает адрес начала выделенного участка памяти, таким образом, создан (порожден) динамический объект. Если участок нужного размера не может быть выделен (нет свободной памяти нужного размера), то операция new возвращает нулевое значение адреса (NULL). В случае инициализации в выделенный участок заносится значение, определяемое инициализатором, т.е. динамическому объекту присваивается начальное значение, иначе значение динамического объекта не определено.

Пример 3:

```
int *p1, *p2;
char *ptr (NULL);
int *mas [10];
int (*point) [10];
struct list *n, *pr;
  struct list
   {   char b; int *k;      }

p1 = new int;          //Создан динамический объект типа int, адрес
                       // которого присвоен указателю p1
*p1 = 25;              //Динамический объект, на который указывает p1
                       //получил значение 25
p2 = new int(15);      //Создана динамическая  переменная целого типа с
начальным
                       //значением 15, указателю p2 присвоен ее адрес
mas[0] = new int;      //Создана динамическая переменная целого типа,
                       // указателю mas[0] присвоен ее адрес
mas[1] = p1;           //Указатели mas[1],  p1 ссылаются на один объект
ptr = new char('*'); //Создан динамический объект типа char, адрес которого
                       //присвоен указателю ptr, объект инициирован
символьным
                       //значением '*'
point = new int[10]; //Создан динамический объект типа массив из 10 целых
                       //элементов, указателю point присвоен адрес первого
элемента
                       //этого массива
n= new list;           //Создан динамический объект типа структура,
                       //указателю n присвоен его адрес
pr = n;                //Указатели pr и n ссылаются на объект типа list
n->b = *p1;            //Элементу с именем b динамической структуры, на
которую
                       //указывает n присвоено значение динамического объекта,
                       //на который указывает p1, т.е. значение 25
n->k = p2;             //Элементу с именем k динамической структуры, на
которую
                       //указывает n присвоено значение указателя p2, т.е. адрес
                       //динамического объекта целого типа со значением 15
```

```
*(n->k) ++;          //Значение динамического объекта целого типа, на
который
                     //ссылается элемент с именем k динамической структуры,
                     //на которую ссылается указатель n, увеличено на 1, т.е.
стало
                     //равно 16 (на этот объект ссылается также указатель p2)
```

Продолжительность существования динамического объекта – от точки его создания операцией new до конца программы или до явного его уничтожения.

Уничтожение динамического объекта (освобождение памяти, выделенной под динамическую переменную) осуществляется операцией delete:

```
delete  имя_указателя ;
```

Здесь указатель адресует освобождаемый участок памяти, ранее выделенный с помощью операции new. После выполнения операции delete значение указателя становится неопределенным (хотя в некоторых реализациях языка может и не меняться).

Для освобождения памяти, выделенной для массива, используется следующая модификация этой операции:

```
delete  [ ] имя_указателя ;
```

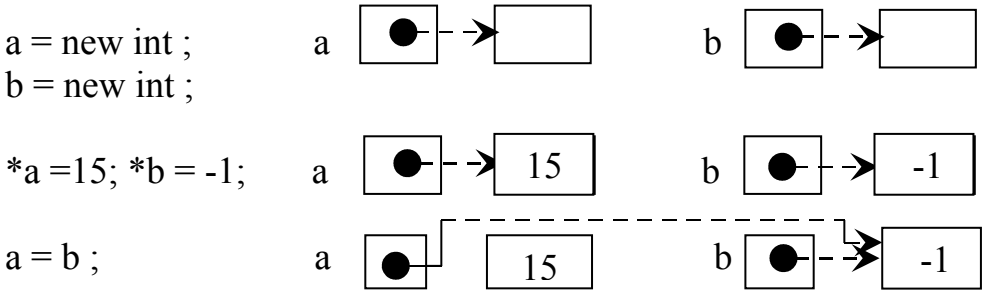
Уничтожая динамический объект не оставляйте “ висячих ” ссылок. “ Висячая ” ссылка - это ссылка на несуществующий объект.

Над динамическими объектами выполняются операции допустимые для типа данного динамического объекта.

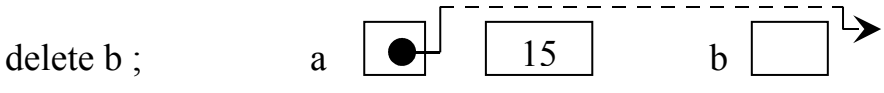
Пример 4:

```
int  *a, *b  ;
  . . . . . .
a =  new int ;  b = new int ;
*a = 15 ; *b = -1 ;  a = b ;  delete b  ;
  . . . . . .
```

Результаты этих действий можно продемонстрировать следующим образом:



Динамический объект со значением 15 существует, но к нему нет доступа. Этот недоступный объект называется “мусором”. Создавая мусор, вы уменьшаете объем свободной динамической памяти. Указатели a и b ссылаются на один объект.



Память, занимаемая динамическим объектом, на который указывают a и b освобождена, но указатель a сохранил значение адреса объекта. Указатель a – “висячая” ссылка.

1.3. Линейный однонаправленный (односвязный) список

Список – это множество (возможно пустое) данных, имеющих некоторый общий смысл для решаемой задачи. Статический список – это множество данных, которые располагаются в списке последовательно и непрерывно друг за другом. На языке Си статический список можно представить неоднородной структурой:

```
struct  list
        { int  n;
          type  elem [ k ];  }
```

Здесь элемент с именем n определяет фактическое количество данных (элементов) в списке, если n равно 0, то список пуст, если n равно k, то список полон; элемент с именем elem (в данном случае массив) определяет само множество элементов списка, type – тип элемента списка.

Динамический список – это множество данных, связанных между собой указателями.

В линейном списке у каждого, составляющего его данного (элемента списка) есть один предшествующий и один следующий элемент (это справедливо для всех элементов кроме первого и последнего).

Линейный односвязный список – это динамический список, каждый элемент которого состоит из двух полей. Одно поле содержит информацию (или ссылку на нее), другое поле содержит ссылку на следующий элемент списка. Элемент списка называют «звено» списка. Таким образом, список – это цепочка связанных между собой звеньев от первого до последнего.

Пример 5 (строку символов BETA представим в виде списка):

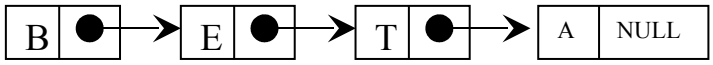


Рис.1.1.

Последнее звено (рис.1.1.) не ссылается на следующий элемент, поэтому поле ссылки имеет значение NULL - пустой указатель.

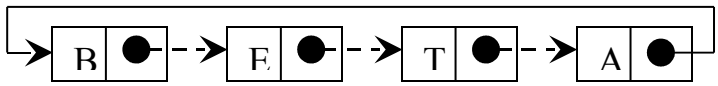
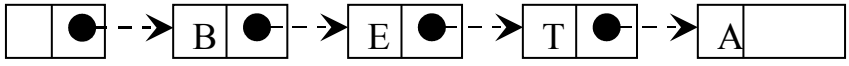


Рис.1.2.

Последнее звено ссылается на первый элемент списка (рис.1.2.) - это циклический список.



Заглавное звено

Рис.1.3.

Список имеет заглавное звено (рис.1.3.), которое ссылается на первый элемент списка. Его информационное поле, как правило, не используется. Заглавное звено позволяет обрабатывать первое звено списка также как и другие его звенья.

По однонаправленному списку можно двигаться только в одном направлении – от первого (или заглавного) звена к последнему.

Чтобы задать динамический список надо описать его звено. Так как звено состоит из полей разных типов, то описать его можно неоднородным типом – структурой.

Задание типа элемента списка:

```
struct list
    { list  *next ;
      type  elem ; }
```

Здесь type – тип информационного поля элемента списка, поле next – ссылка на аналогичную структуру типа list.

Для примера 5 элемент списка может быть определен:

```
struct list
    { list  *next ;
      char  elem ; }
```

Чтобы работать со списком как с единым объектом, надо ввести в употребление статическую переменную-указатель, значение которой – это адрес первого (или заглавного) звена списка. Если список пустой, она должна иметь значение NULL.

Определение статической переменной-указатель:

```
list  *headlist ;
```

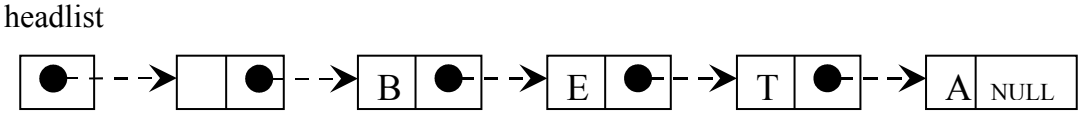


Рис.1.4.

Статическая переменная-указатель headlist, называемая заголовком (или головой) списка (не путать с заглавным звеном) определяет список как единый объект. Используя указатель, можно обращаться к элементам списка (для списка на рис.1.4.):

```
headlist . . .                  //указатель на заглавное звено списка
headlist->next . . .            //указатель на первое звено списка
headlist->next->next . . .      //указатель на второе звено списка
headlist->next->next->next . . . //указатель на третье звено списка
headlist->next->elem . . .      //обращение к информационному
                                //полю первого элемента списка
```

Для более удобной работы со списком лучше ввести указатель, который ссылается на текущий элемент списка.

Пример 6. Формирование списка из примера 5 (не циклического, с заглавным звеном):

```
      void main ( )
        struct list
           { list  *next ;
             char  elem ; } *ph;
       list *p;
       char ch;
 { ph = new list;       //Создание заглавного звена, ph получил значение адреса
  ph->next = NULL; //заглавного звена, его полю next присвоено значение
                          //пустой ссылки, таким образом создан пустой список
                          //с заглавным звеном
  p = ph;                //текущему указателю p присвоена ссылка на заглавное
звено
  while ( (ch = getchar ( )) != '\n' )
    { p->next = new list;   //Создание очередного звена, поле next текущего
звена
                           //получило значение адреса вновь созданного звена
      p = p->next;          //текущему указателю p присвоена ссылка на
                           //очередное звено
      p->elem = ch; }       //информационное поле elem текущего звена
                           //получило значение символа ch
  p->next = NULL; }     //Ссылочному полю на следующий элемент списка
текущего
                          //(последнего сформированного) звена присвоено
значение
                          //пустой ссылки, таким образом определен конец
списка
```

Пример 7. Фрагменты программ, выполняющих различные действия с односвязным списком:

```
        struct list
           { list  *next ;
             char  elem ; } *ph;      //Считаем, что ph – голова списка
       list *p, pr;
      . . . . .
for (p = ph; p != NULL; p = p->next) . . .       //просмотр всех элементов списка
for (p = ph, pr = NULL; p != NULL; pr = p, p = p->next) . . .
                          //просмотр списка с сохранением указателя
                          //на предыдущий элемент списка
for (p = ph; p->next != NULL; p = p->next) . . .
                           //переход к последнему элементу непустого списка
if (ph != NULL)
         for (p = ph; p->next != NULL; p = p->next) . . .
                          //просмотр списка (возможно пустого)  с
                          //сохранением указателя на предыдущий элемент
```

Основными операциями обработки списка являются:

1) поиск заданного элемента по его значению или порядковому номеру; операция заканчивается, когда элемент найден или просмотрен весь список (если элемента нет); результат операции должен определять, есть ли элемент в списке или нет и, если есть, то возможно его адрес или значение;

2) включение (вставка) в список нового элемента перед или после заданного элемента (в том числе перед первым элементом или после последнего); включению, как правило, предшествует поиск элемента после и/или перед которым происходит включение; при включении элемента перед первым в список без заглавного звена меняется заголовок списка; при включении после некоторого элемента меняется ссылочное поле у элемента, после которого происходит включение, поэтому надо определять ссылку на элемент после которого происходит включение;

3) исключение (удаление) заданного элемента из списка (в том числе удаление первого элемента или последнего); исключению, как правило, предшествует поиск, исключаемого элемента; результатом поиска должна быть ссылка на элемент предшествующий исключаемому, так как при удалении элемента из списка меняется ссылочное поле у элемента, предшествующего удаляемому; при удалении первого элемента в списке без заглавного звена меняется заголовок списка;

4) определение числа звеньев списка;

5) упорядочение элементов списка по значению информационного поля.

2. Контрольные вопросы

1. Понятие типа указатель.
2. Задание переменных типа указатель. Операции над указателями.
3. Понятие статического и динамического объекта.
4. Создание и уничтожение динамического объекта. Операции над динамическим объектом.
5. Понятие списка.
6. Понятие линейного односвязного списка. Задание односвязного списка.
7. Операции над односвязным списком.

3. Варианты задания

1. Задан текст, состоящий из строк, разделенных пробелом и оканчивающийся точкой.

Написать подпрограмму поиска заданного элемента в списке. Используя эту подпрограмму :

а) подсчитать количество вхождений заданного символа в каждую строку текста. Вхождение задавать номером строки и номером позиции в строке;

б) найти все вхождения (см. пункт 1.а) заданного символа в текст;

в) найти первое вхождение (см. пункт 1.а) каждой десятичной цифры в текст;

г) найти первое вхождение (см. пункт 1.а) гласных латинских букв в текст;

д) подсчитать количество вхождений четных (нечетных) десятичных цифр в каждую строку текста;

е) заменить заданный символ, если он имеется в тексте, на новое значение (символ), считая, что символ входит в каждую строку не более одного раза;

ж) удалить все вхождения заданного символа из текста;

з) после последнего вхождения каждой гласной латинской буквы в строку текста вставить цифру, изображающую число вхождений этой гласной в данную строку (в строке содержится не более девяти одинаковых гласных);

и) если в строке текста содержится заданный символ, то переместить его на место первого символа в этой строке;

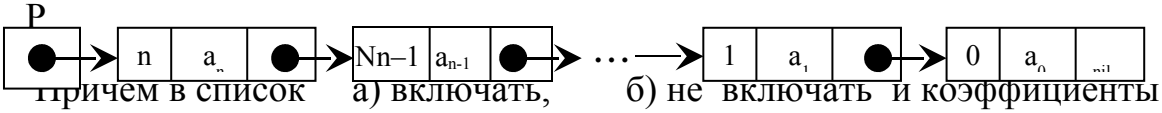
к) если в строке текста содержится заданный символ, то переместить его на место последнего символа в этой строке.

2. Даны действительные числа $x_1, x_2, \ldots, x_n$ ($n >= 2$ и заранее неизвестно). Получить последовательность $(x_1 - x_n)$, $(x_2 - x_n)$, $\ldots$, $(x_{n-1} - x_n)$.

3. Дана последовательность действительных чисел $a_1, a_2, \ldots, a_n$ ($n >= 2$ и заранее неизвестно). Если последовательность упорядочена по неубыванию, то оставить ее без изменения, иначе получить последовательность $a_n, a_{n-1}, \ldots, a_1$.

4. Для заданных полиномов $P_n(x)$ и $Q_n(x)$ найти полином R - сумму полиномов P и Q. Каждый полином представить в виде списка:

P



Причем в список а) включать, б) не включать и коэффициенты равные нулю. Считать, что входные данные не содержат равных нулю коэффициентов.

5. Дана последовательность символов $s_1, s_2, \ldots, s_n$ ($n >= 2$ и заранее неизвестно). Получить те символы, принадлежащие последовательности, которые входят в нее по одному разу.

6. Дана последовательность символов $s_1, s_2, \ldots, s_n$ ($n >= 2$ и заранее неизвестно). Получить последовательность символов, содержащую только последние вхождения каждого символа в строку с сохранением их исходного взаимного порядка.

7. Дана последовательность символов $s_1, s_2, \ldots$. Известно, что s_1 отличен от точки и , что среди $s_2, s_3, \ldots$ имеется хотя бы одна точка. Пусть $s_1, s_2, \ldots, s_n$ - символы, предшествующие первой точке. Получить последовательность $s_1, s_3, \ldots, s_n$, если n нечетно и последовательность $s_2, s_4, \ldots, s_n$, если n четно.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 2
ЛИНЕЙНЫЕ ДВУНАПРАВЛЕННЫЕ СПИСКИ

Цель работы: получить практические навыки организации двунаправленных (двусвязных) списков и их использования при решении задач.

1. Методические указания

Динамический список, в котором каждый элемент (кроме возможно первого и последнего) связан с предыдущим и следующим за ним элементами называется двусвязным. Каждый элемент такого списка имеет два поля с указателями: одно поле содержит ссылку на следующий элемент, другое поле – ссылку на предыдущий элемент и третье поле - информационное. Наличие ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют еще и двунаправленным.

Пример 1 (строка символов BETA представлена двусвязным списком):

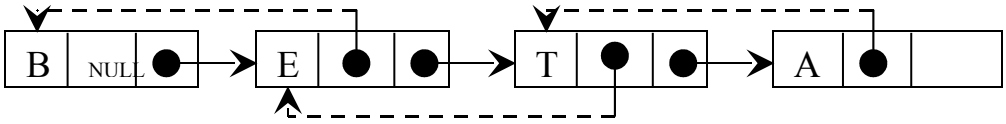


Рис.2.1.

Первое звено не имеет ссылки на предыдущее, последнее звено не имеет ссылки на следующее звено (рис.2.1.).

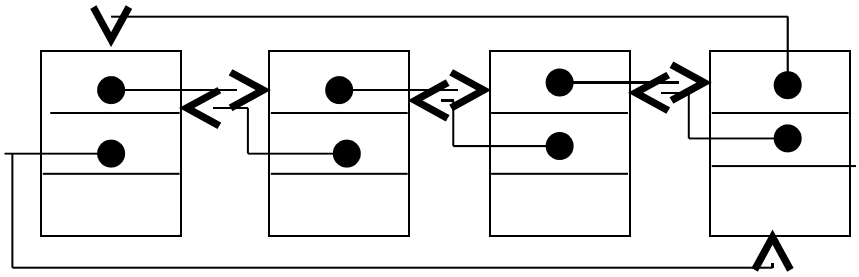


Рис.2.2.

Такой список (рис.2.2.) называют кольцевым (циклическим). Здесь первое звено имеет ссылку на последнее, а последнее звено на первое. Поэтому начинать обработку такого списка можно как с первого звена, так и с последнего.

Двусвязный список может иметь заглавное звено (рис.2.3.).

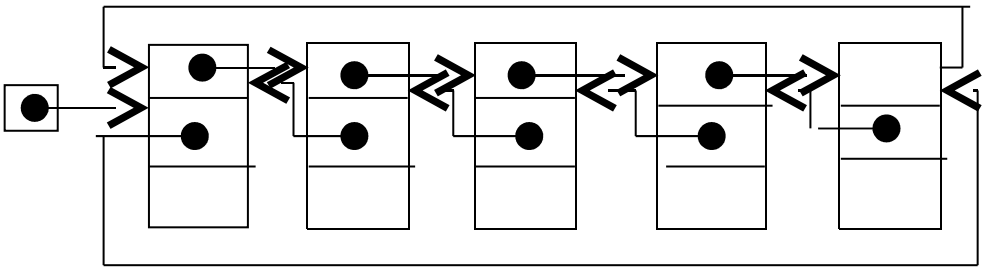


Рис.2.3.

Заглавное звено позволяет обрабатывать первое и последнее звенья также как другие.
Возможны и другие структуры двусвязных списков.

Задание двусвязного списка:

```
struct list2
        { type   elem ;
          list2   * next, * pred ;
                  }

list2   * headlist2 ;
```

Здесь type – тип информационного поля элемента списка.
Переменная-указатель headlist2 задает список как единый программный объект, ее значение - указатель на первый (или заглавный) элемент списка (см. рис.2.3.).

Пример 2. Формирование двунаправленного кольцевого списка

```
void  main ( )
struct list2
      { char  elem ;
        list2  * next, * pred ; }
list2 *l, *r, *x;
char sym;
{ l = new list2;              // Создание пустого
 l -> next = l;               //  кольцевого списка
 l -> pred = l;               //  с заглавным  звеном
 r = l;
   while ( (sym = getchar ( )) != '\n' )
 {   x = new list2; x -> elem = sym;        // Создание очередного звена
 x -> next = r -> next;      // Поле next - указатель на следующее звено
                    //получило значение указателя на первое звено списка
 x -> pred = r;      //Поле pred - указатель на предыдущее звено получило
                    // значение указателя на текущее последнее звено списка
 r -> next = x;     // Поле next текущего последнего элемента - указатель на
                   //следующий элемент получило значение указателя на новый
                   //элемент, таким образом, новое звено добавлено в конец
списка
 r -> next -> pred = x;       // Поле pred заглавного элемента - указатель
                  // на последнее звено списка получило значение указателя
                  // на добавленный в конец списка элемент, таким образом,
                  // сформирован кольцевой список
  r = r -> next;   // Текущий указатель получил значение ссылки на
                  // новый последний элемент списка
          } }
```

Основные операции, выполняемые над двусвязным списком, те же, что и для односвязного списка. Так как двусвязный список более гибкий, чем односвязный, то при включении элемента в список, можно использовать

указатель как на элемент, за которым происходит включение, так и указатель на элемент перед которым происходит включение. При исключении элемента из списка можно использовать как указатель на сам исключаемый элемент, так и указатель на элемент предшествующий или следующий за исключаемым. Но так как элемент двусвязного списка имеет два указателя, то при выполнении операций включения/исключения элемента надо изменять больше связей, чем в односвязном списке. Поиск элемента в двусвязном списке можно вести:
а) просматривая элементы от начала к концу списка,
б) просматривая элементы от конца списка к началу,
в) просматривая список в обоих направлениях одновременно: от начала к середине списка и от конца к середине (учитывая, что элементов в списке может быть четное или нечетное количество).

Пример 3. Фрагменты программ, выполняющих различные действия с двусвязным списком:

```
 void  exam1( list2 *p)  // Просмотр циклического (возможно пустого)
                         // списка без заглавного звена
 { list2 *ph;            // в направлении от начала к концу списка
   if ( p = = NULL ) return;
   ph = p;
  do  { . . .
  p = p -> next; }
  while ( p != ph ); //Просмотр продолжается пока текущий указатель
                     //  p не равен указателю на начало списка - ph
       }
. . .
void  exam2( list2 *p)  // Просмотр кольцевого (возможно пустого)
                         // списка с заглавным звеном
 { list2 *pr;            // в направлении от конца списка к началу
   if ( p –> next = = p ) return;
   pr = p -> pred;      // Текущий указатель pr получил значение ссылки
                        // на последний элемент списка
  while (pr != p)  //Просмотр продолжается пока текущий указатель pr
                   // не равен указателю на заглавное звено списка - p
{ . . . pr = pr -> pred; }
. . .
x = new list2;             // Включение нового элемента (в список с
x -> pred = p -> pred; // заглавным звеном) перед  элементом, на
x -> next = p;             // который ссылается p
p -> pred -> next = x;
p -> pred = x;
. . .
p -> pred -> next = p -> next;  // Исключение из списка с заглавным
p -> next -> pred = p -> pred;  // звеном элемента, на который
delete p;                       // ссылается указатель p
```

2. Контрольные вопросы

1.Понятие двусвязного списка. Возможные структуры двусвязного списка.
2.Задание двусвязного списка.
3.Основные операции над двусвязным списком.

3. Варианты задания

1.Дана последовательность символов, оканчивающаяся точкой:
а) найти всех соседей заданного символа (первый и последний символы считать соседями);
б) подсчитать количество символов, у которых левый сосед больше правого соседа (первый и последний элемент считать соседями);
в) удалить все символы, у которых равные соседи (первый и последний символы считать соседями);
г) переставить в обратном порядке все символы между первым и последним вхождениями заданного символа;
д) в конец последовательности добавить все ее символы, располагая их в обратном порядке (например, из последовательности 1, 2, 3 получить 1, 2, 3, 2, 1).

2.Дана последовательность латинских букв, оканчивающаяся точкой. Среди букв есть специальный символ, появление которого означает отмену предыдущей буквы; k знаков подряд отменяют k предыдущих букв, если такие есть. Учитывая вхождение этого символа преобразовать последовательность.

3.Даны две разреженные квадратные матрицы A и B порядка n (разреженная матрица это матрица высокого порядка с большим количеством нулевых элементов). Получить матрицу C = A+B. Для представления разреженной матрицы использовать двусвязный циклический список. Каждое звено списка состоит из пяти полей :

- поле с номером строки ненулевого элемента,
- поле с номером столбца ненулевого элемента,
- поле со значением элемента,
- поле со ссылкой на предыдущий ненулевой элемент в этой же строке,
- поле со ссылкой на предыдущий ненулевой элемент в этом же столбце.

Каждая строка (и столбец) имеют заглавное звено, соответствующее ссылочное поле которого содержит ссылку на последний ненулевой элемент в строке (в столбце) .

4. Дан многочлен P(x) произвольной степени с целыми коэффициентами, причем его одночлены могут быть не упорядочены по степеням x , а одночлены с одинаковой степенью могут повторяться

(например, $-75x + 8x^4 - x^2 + 6x^4 - 5 - x$). Привести подобные члены в этом многочлене и расположить одночлены по убыванию степеней x .

5. Даны действительные числа $x_1, x_2, \ldots, x_n$ (n >= 2 и заранее неизвестно). Вычислить:

а) $x_1 x_n + x_2 x_{n-1} + \ldots + x_n x_1$;

б) $(x_1 + x_n) (x_2 + x_{n-1}) \ldots (x_n + x_1)$;

в) $(x_1 + x_2 + 2x_n) (x_2 + x_3 + 2x_{n-1}) \ldots (x_{n-1} + x_n + 2x_2)$.

6. Даны действительные числа $y_1, y_2, \ldots, y_n$ (n >= 2 и заранее неизвестно). Получить последовательность:

а) $y_1, y_2, \ldots, y_n, y_1, \ldots, y_n$;

б) $y_1, y_2, \ldots, y_n, y_n, \ldots, y_1$;

в) $y_n, y_{n-1}, \ldots, y_1, y_1, \ldots, y_n$.

7. Даны действительные числа $a_1, a_2, \ldots, a_{2n}$ (n >= 2 и заранее неизвестно). Вычислить:

а) $a_1 a_{2n} + a_2 a_{2n-1} + \ldots + a_n a_{n+1}$;

б) $\min (a_1 + a_{n+1}, a_2 + a_{n+2}, \ldots, a_n + a_{2n})$;

в) $\max (\min (a_1, a_{2n}), \min (a_3, a_{2n-2}), \ldots, \min (a_{2n-1}, a_2))$.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 3

СТРУКТУРЫ ДАННЫХ СТЕКИ и ОЧЕРЕДИ

Цель работы: сформировать практические навыки организации таких распространенных структур как стеки и очереди и их использования при решении задач.

1. Методические указания

Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется вершиной стека. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека.

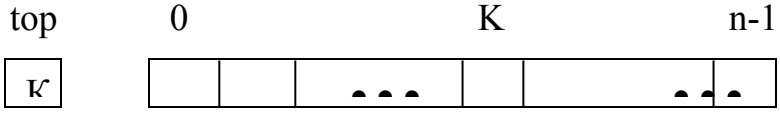
Очередь – это список, у которого доступны два элемента (две позиции) начало и конец очереди. Поместить элемент можно только в конец очереди, а взять элемент только из ее начала.

Стеки и очереди могут быть организованы различными способами. При этом они могут быть размещены как в статической памяти, так и в динамической.

Статический стек можно задать следующим образом:

```
struct  stack
      { int   top;
        type  data  [ n ]  ;}
```

Здесь type – тип элементов, хранимых в стеке. Поле top определяет вершину стека. Ограничение на количество элементов в массиве (константа n) задает размер стека. Статический стек может быть полон.

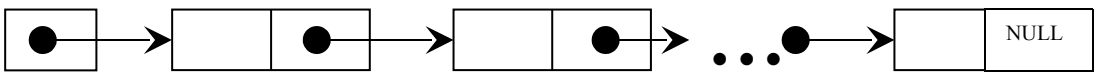


вершина

Вершина стека это либо позиция стека (массива), куда был помещен последний элемент либо позиция, куда будет помещен очередной элемент.

Динамический стек можно задать линейным односвязным списком:

```
struct list
   { type  pole1;
     list *pole2 ; }
typedef   list     stack ;
```



вершина стека

В стеке, представленном линейным односвязным списком, вершина стека – первый элемент списка.

stack *S;

Ввести в употребление переменную типа стек можно следующим образом:

Чтобы инициализировать статический стек (задать начальное значение стека), достаточно определить такое начальное значение поля top, которое показывало бы, что в стеке еще нет элементов, т.е. стек пуст. Если S.top = -1 (или 0 в случае, когда top показывает на позицию, куда будет помещен очередной элемент), то стек пуст. В таком случае при помещении в стек элемента значение S.top увеличивается на единицу, при взятии элемента уменьшается на единицу. Если top равно n-1 (или n), то стек полон. Если указатель равен 0 (или 1), то в стеке один элемент. При взятии из стека единственного элемента, он становиться пустым, поле top должно получить значение -1 (или 0).Обращение к элементу в вершине статического стека: S.data [S.top].

Чтобы инициализировать динамический стек, т.е. создать пустой динамический стек достаточно указателю на вершину стека задать значение NULL: S = NULL. Обращение к значению элемента в вершине динамического стека: S -> pole1.

Основные операции над стеком (S - переменная типа stack):

Таблица 1

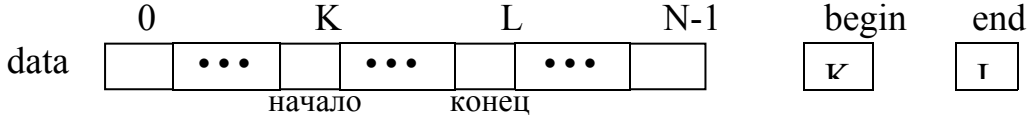
Название Операции	Статистический способ определения стека	Динамический способ определения стека
Очистить стек (создать стек)	В поле S.top ← ∅	S:= NULL
Поместить элемент в стек	Если S.top < n-1 , то S.top = S.top + 1; S.data[S.top] = элемент; , иначе ошибка - «переполнение стека»	Добавить элемент в начало линейного списка (см. лабораторная работа № 1)
Взять элемент из стека	Если стек не пуст, то элемент = S.data[S.top]; S.top = S.top – 1; , иначе ошибка - “операция не определена”	Элемент = S ->pole1; Удалить первый элемент из линейного списка, сделав первым следующий элемент списка: S = S -> pole2; Операция определена, если стек не пуст.
Проверка пуст ли стек?	пуст (S)=true, если S.top =∅	Пуст(S)=true, если S = NULL Пуст(S)=false,если S!=

	пуст(S)= false, если S.top ≠∅	NULL

Статическую очередь можно представить:

а) линейным массивом с двумя указателями: на начало и на конец очереди. Указатель на начало определяет позицию очереди (массива), откуда можно взять элемент, указатель на конец – позицию, куда кладем элемент:

```
struct  ch1
    { int   begin, end ;
      type   data  [ N ] ; }
```



Задать очередь – это определить переменную типа ch1:

```
ch1    Q;
```

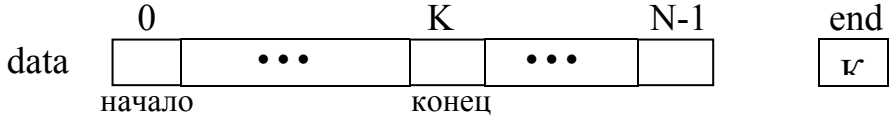
Чтобы инициализировать очередь достаточно определить начальное значение указателя на начало и/или конец очереди. Для этого полю begin дадим значение, не принадлежащее диапазону от 0 до N-1 (а полю end можно задать значение, указывающее на первый элемент очереди – это упростит работу с указателями при включении элемента в очередь). При включении элемента в непустую очередь увеличивается на единицу значение указателя end, при взятии элемента из очереди значение указателя на начало. При включении элемента в пустую очередь увеличиваются оба указателя.

В этом варианте не используются освободившиеся после взятия элементов из очереди позиции, поэтому возможно мнимое переполнение очереди. Если end равно N-1 и begin равно 0 – очередь полна, а если begin больше 0, то очередь псевдополна (т.е. из очереди были взяты элементы и в начале массива есть свободные позиции).

Обращение к элементу при взятии из очереди: Q.data [Q.begin], при занесении в очередь: Q.data [Q.end]. Если в очереди один элемент, то оба указателя показывают на него, т.е. их значения равны, если из очереди взять единственный элемент, она должна стать пустой;

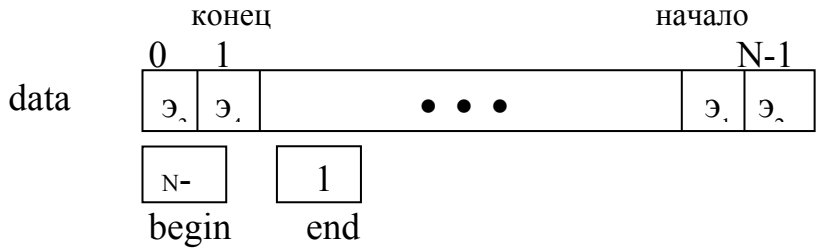
б) линейным массивом с одним указателем на начало или на конец очереди, вторая позиция очереди фиксируется (например, начало очереди это первый элемент массива. В таком случае очередь сдвигается после каждого взятия из нее элемента):

```
struct ch2
    { int  end ;
      type  data [ N ] ;
```



в) линейным массивом с двумя указателями (аналогично варианту а). При достижении конца массива (последнего элемента) очередь сдвигается на начало массива (к первому элементу), если есть свободные позиции в начале массива. Таким образом, освобождаемся от псевдопереполнения очереди;

г) циклическая очередь – линейный массив с двумя указателями (аналогично варианту а), в котором при достижении каким-либо указателем конца массива (последнего элемента) значение этого указателя очереди округляется по модулю N, где N – размер массива. Таким образом, в циклической очереди при достижении конца массива (end = N-1) и наличии свободных позиций в начале (begin > 0), новый элемент помещается в массив на место с индексом 0, при этом указателю end присваивается значение 0.

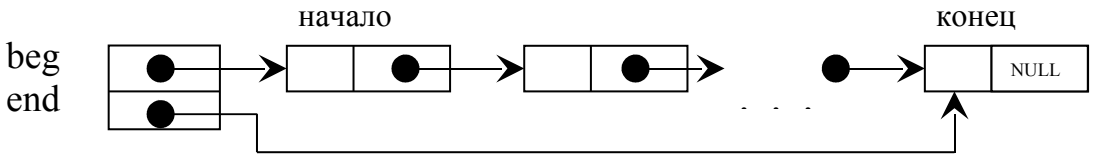


Очередь будет полна, если поле end = N-1, а поле begin = 0 или поле end = K, а begin = K+1. Здесь 0 < K < N-1.

В динамической памяти очередь можно представить:

а) линейным односвязным списком с двумя указателями:

```
struct   list1
  { type  pole1;
    list1  *pole2; }

struct   ch3
     { list1  *beg, *end ; }
```



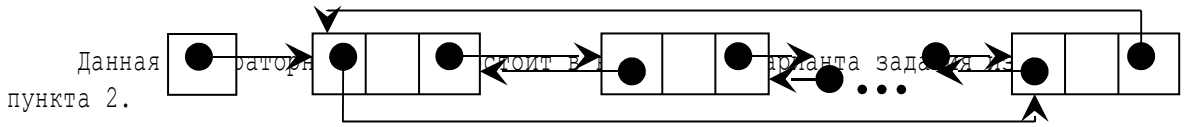
Задать динамическую очередь - это определить переменную типа ch3:

```
ch3   Q1;
```

Чтобы инициировать очередь достаточно указателю на начало Q1.beg и/или на конец Q1.end присвоить значение NULL. При взятии единственного элемента из очереди она должна стать пустой;

б) линейным двусвязным циклическим списком с одним указателем:

```
struct    list2
   {  type  pole1;
      list2 *pole1, *pole2; }
```



Здесь начало или конец очереди можно сделать как в начале, так и в конце списка.

Кроме рассмотренных вариантов простых очередей существует понятие очереди с приоритетом. Элемент такой очереди имеет не только значение, но и вес или приоритет этого значения. При включении элемента в очередь с приоритетом сначала отыскивается место элемента в соответствии с его приоритетом. Элемент с наивысшим приоритетом помещается в начало очереди, элемент с низшим весом в конец очереди. Такой очереди наилучшим образом отвечает двусвязный циклический список.

Основные операции над очередью аналогичны операциям со стеком (с учетом особенностей этой структуры).

Еще одной разновидностью этих структур данных является дек. Дек это список, у которого обе позиции: начало и конец списка доступны для добавления и взятия элемента. Таким образом, дек может быть и стеком, и очередью, и комбинацией этих структур. Наиболее часто дек представляется структурой с ограничением на вход или выход: дек с ограниченным входом (только одна позиция доступна для добавления элемента) и дек с ограниченным выходом (только одна позиция доступна для взятия элемента из дека).

2. Контрольные вопросы

1. Понятие структуры данных стек, очередь, дек.
2. Представление в памяти структур данных стек, очередь, дек.
3. Задание структур данных стек, очередь, дек.
4. Основные операции над структурами данных стек, очередь, дек.
5. Достоинства и недостатки различного представления в памяти структур данных стек, очередь, дек.
6. Использование структур данных стек, очередь для решения задач.

3. Варианты задания

1. Написать операции работы с заданной структурой данных, включив их в один

модуль (файл). К основным операциям (см. таблицу 1) добавить операцию, показывающую содержимое структуры после выполнения какого-либо действия с ней. Эту операцию реализовать на основе базовых операций:

а) основные операции над статическим стеком;

б) основные операции над динамическим стеком;

в) операция “принадлежит ли заданный элемент ” статическому стеку;

г) операция “принадлежит ли заданный элемент ” динамическому стеку;

д) основные операции над статической очередью (вид очереди: а, б, в, г);

е) основные операции над динамической очередью (вид очереди: а, б);

ж) операция “добавить элемент в очередь” для статической очереди с приоритетом (вид очереди: а, б, в, г). На вход такой очереди подается элемент и его приоритет, определяющий место элемента в очереди. В очереди с приоритетом элементы должны располагаться в порядке возрастания или убывания приоритетов;

з) операция “добавить элемент в очередь” для динамической очереди с приоритетом (вид очереди: а, б) см. 1.ж;

и) операция “принадлежит ли заданный элемент ” статической очереди (вид очереди а, б, в, г);

к) операция “принадлежит ли заданный элемент ” динамической очереди (вид очереди: а, б);

л) основные операции над статическим деком;

м) основные операции над динамическим деком.

2. Написать программу, демонстрирующую выполнение операций, над заданной структурой данных. Эту программу надо поместить в свой модуль (файл). Модуль с основными операциями включать в программу, используя директиву include.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 4

СТРУКТУРЫ ДАННЫХ: ДЕРЕВЬЯ

Цель работы: изучить основные алгоритмы работы с деревьями; получить практические навыки разработки и использования этих структур и алгоритмов для решения задач.

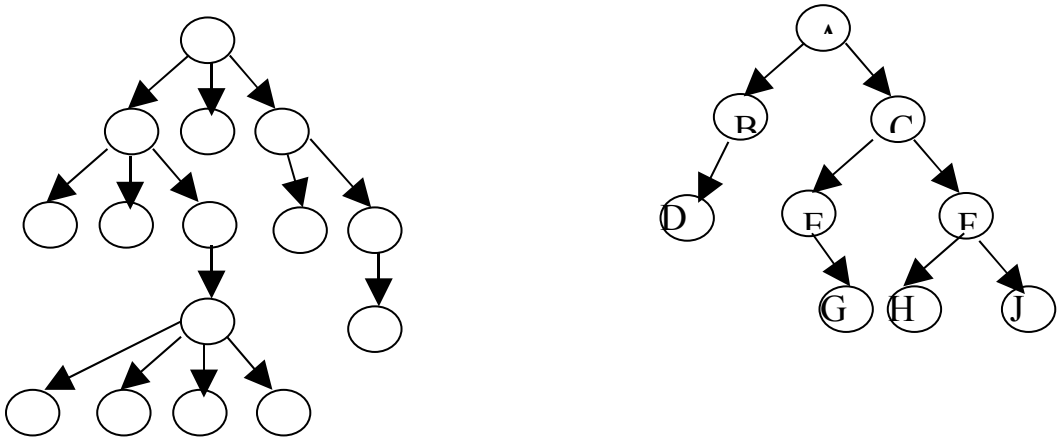
1. Методические указания

1.1. Понятие дерева. Виды деревьев

Дерево – это конечное множество T , возможно пустое, в противном случае,
состоящее из одного или более элементов (узлов или вершин дерева) таких, что:

а) имеется один специально обозначенный элемент, называемый корнем данного дерева;

б) остальные элементы содержатся в $m > 0$ попарно непересекающихся множествах $T_1, \ldots, T_m$, каждое из которых в свою очередь является деревом; деревья $T_1, \ldots, T_m$ называются поддеревьями данного корня (рис.4.1.а).



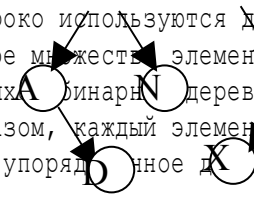
а б

Рис.4.1

Если имеет значение относительный порядок поддеревьев $T_1, \ldots, T_m$, то говорят, что дерево является упорядоченным. Число поддеревьев данного узла называется степенью этого узла. Узел с нулевой степенью называется концевым узлом (или листом или терминальным узлом), все остальные элементы – внутренние узлы (нетерминальные). Максимальная степень всех вершин называется степенью дерева. Корень дерева имеет уровень равный 0. Остальные вершины имеют уровень на единицу больше уровня непосредственного предка. Максимальный уровень какой-либо из вершин называется глубиной или высотой дерева. Минимальная высота при заданном числе вершин достигается, если на всех уровнях, кроме последнего,

помещается максимально возможное число вершин. Максимальное число

Наиболее широко используются двоичные (бинарные) деревья (рис.4.1.б). Бинарное дерево это конечное множество элементов, которое либо пусто, либо состоит из корня и из двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня. Таким образом, каждый элемент бинарного дерева имеет 0, 1 или 2 поддерева. Бинарное дерево - упорядоченное дерево, так как в нем различают левое и правое поддеревья.



Дерево поиска - это двоичное дерево, в котором выполняются следующие условия (см. рис.4.2):

а) l(u) < l(v) для всех вершин u, v, если вершина u находится в поддереве, корень которого - левый преемник v;

б) l(u) > l(v) для всех вершин u, v, если вершина u находится в поддереве, корень которого - правый преемник v;

в) для всякого значения aє S существует единственная вершина v, для которой l(v) = a (S - множество значений, допускающих сравнения).

вершин в дереве высотой h достигается в том случае, если из каждой вершины, за исключением уровня h, исходят d поддеревьев, где d – степень дерева: на 0-м уровне 1 вершина, на 1-м – d потомков, на 2-м – d^2 потомков, на 3-м уровне d^3 потомков и т.д.

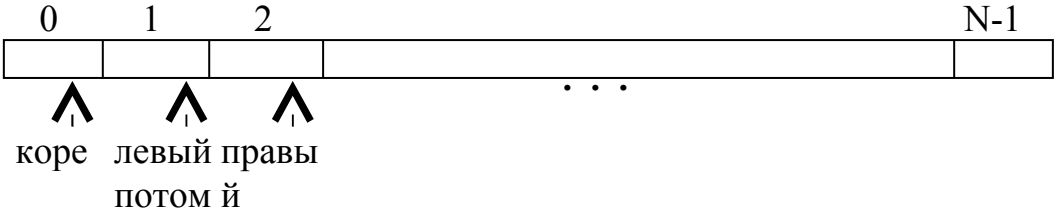
Определение структуры дерева, данное выше, является рекурсивным и отражает рекурсивную природу самой структуры.

Структуру данных – дерево можно представить как в статической, так и в динамической памяти. В статической памяти дерево можно представить массивом, для которого определено понятие пустого элемента:

```
struct stree
  { type  elem [ N ]; }
stree   d;
```

или

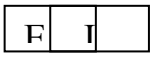
```
type    d [ N ];
```



Вершины двоичного дерева располагаются в массиве следующим образом: если k – индекс вершины дерева, то ее потомки находятся в элементах массива с индексами 2k + 1 и 2(k + 1); корень дерева находится в элементе с индексом 0 (при индексации в массиве от 1 до N индексы потомков k-ой вершины: 2k и 2k + 1, корень имеет индекс 1).

В динамической памяти дерево представляется иерархическим списком. Задать элемент двоичного дерева можно как элемент списка с тремя полями:

два ссылочных поля, содержащие указатели на его левое (L) и правое (R) поддеревья, и информационное поле (E):



Определение типа элемента бинарного динамического дерева:

```
struct btree
    {type   elem;
      btree  *left, *right;}
```

Здесь type – тип информационного поля (если информационное поле имеет сложную структуру, то type может быть типом - указатель на объект, содержащий значение элемента дерева).

Чтобы определить дерево как единую структуру, надо задать указатель на корень дерева:

```
btree     * d ;
```

На рис.4.3 представлено двоичное динамическое дерево d в соответствии с определением типа элемента, сделанным выше. Элементы дерева со степенью 0 и 1 имеют два или одно ссылочное поле со значением равным пустому указателю (NULL).

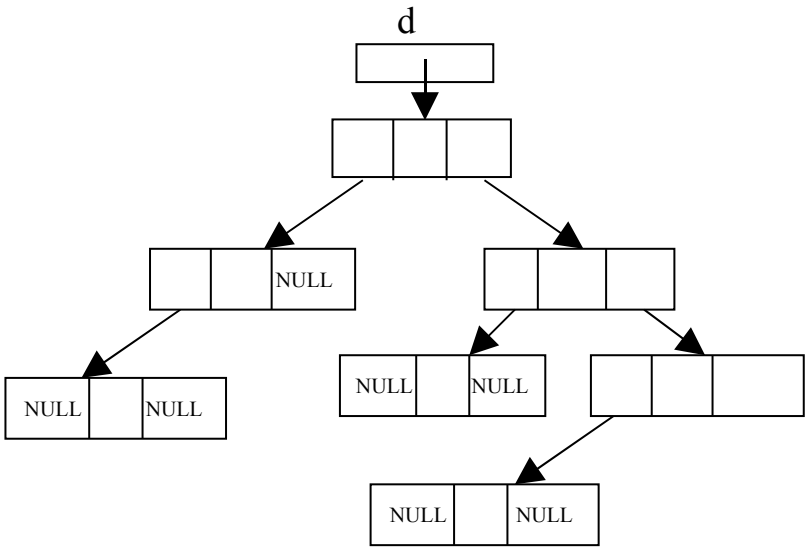


Рис.4.3

Обрабатывая дерево, приходится просматривать его элементы – эта операция называется обход дерева (или прохождение дерева).

Обход дерева – это способ методичного исследования его узлов, при котором каждый узел проходится точно один раз. Полное прохождение динамического дерева дает линейную расстановку его узлов.

Возможны два способа прохождения бинарного дерева (если дерево не пусто):

а) в глубину:

- прямой (префиксный) порядок: корень, левое поддерево в прямом порядке,

правое поддерево в прямом порядке (линейная расстановка узлов дерева, представленного на рис.4.1.б: ABDCEGFHJ);

Алгоритм прямого обхода:

```
{ S = NULL;
   while ( d != NULL)
   {  обработка ( d );
      if ( d->left != NULL && d->right != NULL)
            {встек (S, d->right); d = d->left;}
         else  if ( d->left = = NULL && d->right = =NULL)
                  if ( S!= NULL) изстека (S, d); else d = NULL;
               else  if (d->left != NULL) d = d->left;
                                 else d = d->right;
            }
   }
```

Рекурсивное описание алгоритма прямого обхода:

```
пр_обход ( btree *d)
{ if ( d != NULL ) { обработка ( d ); пр_обход ( d->left );
                                       пр_обход ( d->right ); }
  }
```

- обратный (инфиксный) порядок: левое поддерево в обратном порядке, корень,

правое поддерево в обратном порядке (линейная расстановка узлов дерева, представленного на рис.4.1.б: DBAEGCHFJ);

Алгоритм обратного обхода:

```
{ S = NULL;  F = 1;
   while ( F )
     {     while ( d != NULL )
              { встек ( S, d ); d = d->left; }
       if ( S != NULL ) { изстека ( S, d );
                          обработка ( d ); d = d-right;}
            else F = 0;
            }
      }
```

Рекурсивное описание алгоритма обратного обхода:

```
обр_обход ( btree *d )
{ if ( d != NULL ) { обр_обход ( d->left );  обработка ( d );
                     обр_обход ( d->right ); }
  }
```

- концевой (постфиксный) порядок: левое поддерево в концевом порядке, правое

поддерево в концевом порядке, корень (линейная расстановка узлов дерева, представленного на рис.4.1.б: DBGEHJFCA);

Алгоритм концевого обхода:

```
{ S = NULL;
  do
        while ( d != NULL )
            { встек ( S, d, 0 ); d = d->left; }
     if ( S != NULL )
            { do
                    изстека ( S, d, F );  if ( F ) обработка ( d );
                while ( F && S != NULL );
              if ( ! F ) { встек ( S, d, 1 ); d = d->right; }
   while ( S != NULL );
     }
```

Рекурсивное описание алгоритма концевого обхода:

```
 конц_обход ( btree *d )
{ if ( d != NULL ) { конц_обход ( d->left );  конц_обход ( d->right );
                                        обработка ( d ); }
        }
```

б) в ширину: узлы дерева посещаются слева направо, уровень за уровнем вниз от корня (линейная расстановка узлов дерева, представленного на рис.4.1.б: ABCDEFGHJ).

Алгоритм обхода в ширину:

```
{ Q = NULL;
  if ( d != NULL ) { в_очередь ( Q, d );
        do
            из_очереди ( Q, d ); обработка ( d );
            if ( d->left != NULL ) в_очередь ( Q, d->left );
            if ( d->right != NULL ) в_очередь ( Q, d->right );
        while ( ! пуста_очередь ( Q ) );
                    }
     }
```

Основные операции при работе с деревьями:

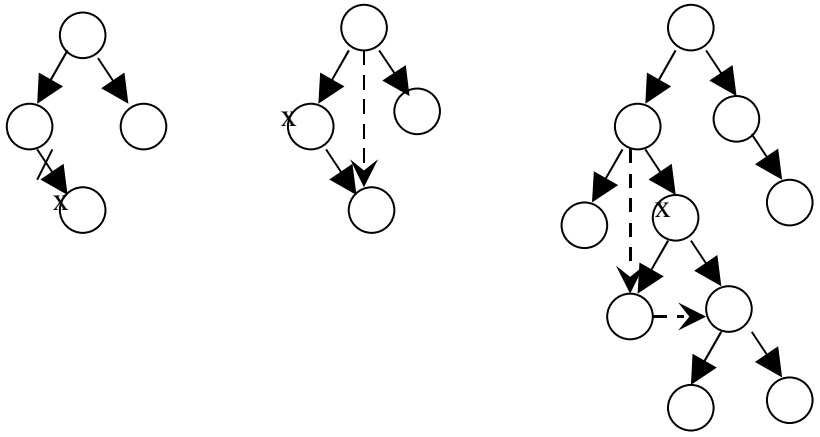
а) поиск элемента в дереве. Операция заключается в прохождении узлов дерева в одном из рассмотренных выше порядке. При прохождении дерева поиска достаточно пройти только то поддерево, которое возможно содержит искомый элемент;

б) включение элемента в дерево. Операция заключается в добавлении листа (исключая сбалансированное дерево – включение элемента в сбалансированное дерево приводит, как правило, к его перестройке) в какое-то поддерево, если такого элемента нет в исходном дереве. При включении нового элемента в дерево поиска лист добавляется в то поддерево, в котором не нарушается отношение порядка;

в) удаление элемента из дерева. Операция заключается в изменении связей между дочерними и родительскими, по отношению к удаляемому,

элементами (исключая сбалансированное дерево – удаление элемента из сбалансированного дерева приводит, как правило, к его перестройке); здесь необходимо рассматривать три случая: удаление листа (см. рис.4.4.а), удаление элемента с одним потомком (см. рис.4.4.б), удаление элемента с двумя потомками (см. рис.4.4.в).

При удалении элемента степени 2 из дерева поиска изменять связи необходимо так, чтобы не нарушалось установленное в дереве отношение порядка. Лучшим вариантом в этом случае будет перемещение на место удаляемого элемента либо самого правого листа из левого поддерева удаляемого элемента, либо самого левого листа из правого поддерева удаляемого элемента;



а б в

Рис.4.4

г) сравнение деревьев (проверка деревьев на равенство). Операция заключается в прохождении обоих деревьев в одном порядке до тех пор, пока либо не будут пройдены оба дерева, либо одно из них, либо соответствующие элементы окажутся не равными, либо в одном из деревьев не будет обнаружено отсутствие соответствующего элемента. Только в случае равенства деревьев оба дерева будут пройдены одновременно;

д) копирование дерева. Операция заключается в прохождении исходного дерева и построении нового дерева с элементами, информационные поля которых равны информационным полям соответствующих элементов исходного дерева.

1.2. Ввод дерева

Чтобы ввести дерево надо выбрать какой-то способ перечисления его узлов. Одним из возможных способов является так называемая списочная запись (представление). Список – это конечная последовательность атомов либо списков, число которых, может быть и равно нулю (атом – это понятие, определяющее элемент из любого множества предметов). Используя скобки, список можно задать перечислением через запятую его атомов (порядок

перечисления атомов определяет их упорядочение). Таким образом, дерево, представленное на рис.4.1.б можно записать в виде следующего списка:
(A, (B, (D, 0, 0), 0), (C, (E, 0, (G, 0,0)), (F, (H, 0, 0), (I, 0, 0))))

Ввод дерева, представленного таким списком, можно описать рекурсивной функцией (тип дерева btree описан выше, здесь type – тип информационного поля элемента – char):

```
btree * build_tree ( )
  { char  sym;
    btree *d;
        scanf ("%c", &sym );
        switch ( sym )
       {case ' (' : { d =  new btree;
                      scanf ("%c", &sym );  d->elem  = sym;
                       d->left =  build_tree (  );
                       d->right = build_tree (  ); scanf( "%c", &sym );
                        return  d  ;  }
         case  '0'  :  return NULL ;
         case  ','   : d  = build_tree (  );  break ;
                  }
                     }
```

2.Контрольные вопросы

1. Понятие дерева, двоичного дерева, упорядоченного дерева, дерева поиска.
2. Способы задания дерева.
3. Способы обхода дерева (в глубину: прямой, обратный, концевой; в ширину).
4. Показать, что бинарное дерево можно построить, если заданы прямой и обратный порядки расположения его узлов. Можно ли это сделать, если заданы прямой и концевой порядки? Если заданы обратный и концевой порядки?
5. Найдите все бинарные деревья, узлы которых располагаются точно в одной и той же последовательности а) в прямом и обратном порядках; б) в прямом и концевом порядках; в) в обратном и концевом порядках.
6. Каково наибольшее число вершин, находящихся одновременно в стеке при обходе двоичного дерева, содержащего n элементов: а) в прямом, б) обратном, в) концевом порядках?

2. Варианты задания

Во всех задачах тип значений элементов дерева простой. Исходное дерево после ввода распечатать в одном из порядков.

1. В заданном бинарном дереве подсчитать число его листьев и напечатать их значения:
   а) при прямом обходе дерева;
   б) при обратном обходе дерева;
   в) при концевом обходе дерева;
   г) реализуя обход, рекурсивно.
2. В заданном бинарном дереве найти первое вхождение заданного элемента и напечатать пройденные при поиске узлы дерева:
   а) при прямом обходе дерева;
   б) при обратном обходе дерева;
   в) при концевом обходе дерева;
   г) реализуя обход, рекурсивно.
3. В заданном непустом бинарном дереве найти длину (число ветвей) пути от корня до ближайшей вершины со значением равным заданному :
   а) при прямом обходе дерева;
   б) реализуя обход, рекурсивно.
4. В заданном непустом бинарном дереве подсчитать число вершин на n- ом уровне, считая корень вершиной 0-го уровня.
5. Задано бинарное дерево. Определить, есть ли в этом дереве хотя бы два одинаковых элемента.
6. Распечатать все элементы заданного непустого бинарного дерева по уровням:

сначала из корня дерева, затем (слева направо) из вершин, дочерних по отношению к корню, затем (слева направо) из вершин, дочерних по отношению к этим вершинам, и т.д.

7. Формулу вида:

< формула > ::= < терминал > | (< формула > < знак > < формула >)
< знак > ::= + | - | *
< терминал > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

можно представить в виде двоичного дерева (‘дерева – формулы ‘):

- формула из одного терминала (цифры) представляется деревом из одной вершины (корнем) с этим терминалом;
- формула вида (f_1 s f_2) - деревом, в котором корень – это знак s, а левое и правое поддеревья – это соответствующие представления f_1 и f_2 :

а) по заданной формуле построить дерево – формулу, обходя дерево – формулу в 1)прямом, 2)обратном, 3)концевом порядке, напечатать его элементы и вычислить (как целое число) значение;

б) проверить, является ли заданное двоичное дерево (с элементами типа char) деревом – формулой и напечатать его элементы в порядке 1)прямого, 2)обратного, 3)концевого обхода;

в) пусть в дереве – формуле в качестве терминалов используются не только цифры, но и буквы, играющие роль переменных; преобразовать заданное дерево – формулу, заменяя в нем все поддеревья, соответствующие

формулам (($f_1 \pm f_2$) * f_3) и (f_1 *($f_2 \pm f_3$)) на поддеревья, соответствующие формулам ((f_1 * f_3) ± (f_2 * f_3)) и ((f_1 * f_2) ± (f_1 * f_3)); распечатать преобразованное дерево в 1)прямом, 2)обратном, 3)концевом порядке;

г) выполнить в заданном дереве – формуле преобразования, обратные преобразованиям из пункта в; распечатать преобразованное дерево в 1)прямом, 2)обратном, 3)концевом порядке.

8. Заданную последовательность из строчных латинских букв, оканчивающуюся

точкой, представить в виде дерева поиска, затем преобразовать его, удалив символ с а) наименьшим, б) наибольшим значением; распечатать преобразованное дерево в 1)прямом, 2)обратном, 3)концевом порядке.

9. Заданную последовательность целых чисел, оканчивающуюся нулем, представить в виде дерева поиска, затем преобразовать его, добавив еще одно целое число; распечатать преобразованное дерево в 1)прямом, 2)обратном, 3)концевом порядке.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 5

УПРАВЛЕНИЕ ТАБЛИЦАМИ

Цель работы: получить практические навыки организации таблиц, обработки таблиц и их использования при решении задач.

1. **Методические указания**

Таблица – это список состоящий из конечного множества элементов, при чем каждый элемент характеризуется рядом признаков (свойств). Один из признаков, называемый ключом, позволяет отличить один элемент от другого (идентифицировать элемент). Ключ может однозначно определять элемент таблицы (ключи всех элементов различны) или неоднозначно (в таблице есть элементы с равными ключами).

Все действия над элементами выполняются в соответствии с их ключами: по ключу элементы выбираются из таблицы и добавляются в нее.

Таблицы можно хранить как в статической, так и в динамической памяти. В данной работе рассматриваются статические таблицы. Статическая таблица отображается массивом. Статическую таблицу можно представить следующим образом (см. рис.5.1):

	признак 1	признак 2	. . .	признак m
элемент 1				
элемент 2				
⋮				
элемент n				
⋮				
N				

Рис.5.1

где N - максимальное количество элементов в таблице (размер таблицы);
n - фактическое количество элементов в таблице.

Если n = N , таблица полна, если n < N , в таблице есть свободные позиции. Вместо задания n можно ввести, если это возможно, понятие пустого элемента. Пустой элемент будет определять

Либо таблицу можно представить массивом, для которого определено понятие пустого элемента. Это используется в хеш-таблице.

Задать таблицу можно следующим образом:

```
struct table
  { type  elem [ N ];
```

```
        int   n;  }
    table  T;
```
или во втором случае:
```
    type  T [ N ];
```
Здесь type – тип элемента таблицы, который, как правило, является сложным типом.

Неупорядоченная таблица – это таблица, элементы которой располагаются в порядке их поступления в таблицу.

Упорядоченная таблица – это таблица, между элементами которой установлено отношение порядка. Это отношение, как правило, устанавливается на признаке ключ. Поэтому говорят, что таблица упорядочена по ключам элементов. Таблица упорядочена по возрастанию значений ключа, если ключ(T_i) < ключ(T_{i+1}) для всех i = 1, 2 , . . . , n-1 (здесь T_i - i -й элемент таблицы T). Таблица упорядочена по убыванию значений ключа, если ключ(T_i) > ключ(T_{i+1}) для всех i = 1, 2, . . . , n-1.

Основные операции:

1. Поиск в таблице элемента с заданным ключом.
2. Включение заданного элемента в таблицу.
3. Исключение из таблицы элемента с заданным ключом.

Различные способы организации таблиц принято сравнивать по времени выполнения в них основных операций и прежде всего по времени поиска. В неупорядоченной таблице поиск элемента в среднем будет выполнен за n/2 сравнений, так как поиск заключается в последовательном просмотре элементов таблицы (максимальное число сравнений n). В упорядоченной таблице кроме последовательного просмотра можно применять бинарный поиск (поиск делением пополам), максимальное число сравнений при котором $1 + \log_2 n$.

Включение элемента в неупорядоченную таблицу заключается в добавлении элемента в таблицу (если n < N) на очередное свободное место. Включению элемента в упорядоченную таблицу предшествует поиск места k для нового элемента с учетом отношения порядка, затем освобождение этого места (если оно занято) смещением всех последующих элементов, начиная с k-го вниз на одну позицию. Таким образом, включение в упорядоченную таблицу требует больше времени, чем в неупорядоченную таблицу.

Исключению элемента из таблицы также предшествует поиск по ключу элемента и, если элемент в таблице есть, то исключение заключается в смещении всех последующих элементов на одну позицию вверх.

Таблица с вычисляемым входом (хеш-таблица) – это таблица, элементы которой располагаются в соответствии с некоторой функцией расстановки (хеш-функцией). Функция расстановки f (ключ) вычисляет для каждого элемента таблицы по его ключу номер (позицию) элемента в массиве. Таким образом, диапазон значений функции f (ключ) –

$$0 \div N\text{-}1 \text{ или } 1 \div N.$$

Если ключ(T_i) ≠ ключ (T_j) и f (ключ (T_i)) ≠ f (ключ (T_j)) при $i \neq j$, для всех i и j = 0, 1, 2, . . . , n , то элементы таблицы взаимно -

однозначно отображаются в элементы массива. Организованная таким образом таблица называется таблицей с прямой адресацией. Прямая адресация применима, если количество возможных ключей невелико. Если в такой таблице число реально присутствующих элементов мало по сравнению с N, то в таблице много пустых элементов (т.е. много памяти тратится зря).

Если ключ (T_i) $\neq$ ключ (T_j), а f (ключ (T_i)) = f (ключ (T_j)) при $i \neq j$, то нет однозначного отображения элемента таблицы в элемент массива. Эта ситуация называется коллизией (столкновением). Если заранее неизвестен диапазон значений ключа, то выбрать функцию расстановки, дающую однозначное отображение практически невозможно. В таких таблицах возникает проблема разрешения коллизий, которая включает две задачи:

1) выбор функции расстановки;
2) выбор способа разрешения коллизий (способа рехеширования).

Функция расстановки подбирается в каждом случае в зависимости от ключа так, чтобы алгоритм ее вычисления был несложным. Функцию расстановки подбирают из условия случайного и возможно более равномерного отображения (перемешивания) ключей в адреса хранения, но «случайная» хеш-функция должна быть детерминированной в том смысле, что при ее повторных вычислениях с одним и тем же ключом, она должна давать одно и то же значение.

Известно несколько методов получения хеш-функции, при этом обычно предполагают, что область определения хеш-функции – целые неотрицательные числа, если ключи не являются такими их всегда можно преобразовать к целому типу:

а) метод деления: ключу ставится в соответствие остаток от деления на N – значение функции вычисляется по формуле:

$$f(\text{ключ}) = \varphi(\text{ключ}) \bmod N,$$

здесь φ (ключ) – функция, преобразующая ключ элемента в целое число; (mod N) - означает по модулю N, т.е. остаток от деления на N. Здесь хорошо задавать N простым числом, плохо, если $N = 2^p$ или $N = 10^p$.

б) метод умножения: значение хеш-функции – это какая-то часть преобразованного значения ключа, либо часть самого значения ключа. Значение хеш-функции вычисляется по формуле:

$$f(\text{ключ}) = [N \cdot (\text{ключ} \cdot A \bmod 1)],$$

здесь A – константа из интервала (0,1). Выбор константы зависит от исходных данных, но любое значение константы дает не плохие результаты. Выражение (ключ·A mod 1) – это дробная часть ключ·A.
Квадратные скобки – это взятие целой части $N \cdot (\text{ключ} \cdot A \bmod 1)$.
Если $N = 2^m$, (степень двойки) или $N = 10^p$, то операция умножения на N – сдвиг значения на m (или p) разрядов. Таким образом, значением хеш-функции будет m (или p) разрядов выражения (ключ·A mod 1). В некоторых случаях при $N = 2^m$ в качестве значения функции можно взять m - разрядное двоичное число либо методом выделения каких-либо m битов из

значения ключа, либо применяя логическую операцию над некоторыми частями ключа, либо, разделив ключ на m частей, просуммировать их и в качестве значения функции взять младшие m разрядов (это допустимо, если эта часть ключа уникальна или повторяется редко).

Разрешать коллизии можно в исходной или в дополнительной таблице. Если в исходной, то такая таблица называется таблицей с перемешиванием или с открытой адресацией. Таблица с перемешиванием должна быть инициирована, чтобы элементы таблицы получили в начале значение «пусто» (при взятии элемента из такой таблицы надо различать еще одно состояние: элемент удален из таблицы). В таблицах с перемешиванием применяют линейное (метод последовательных испытаний или проб), случайное или квадратичное рехеширование. Наиболее простой метод - линейное рехеширование, который заключается в том, что функция вторичной расстановки $f^{|}$ определяется как

$$f^{|}(\text{ключ}, i) = (f(\text{ключ}) + i) \bmod N ,$$

здесь значение i равно 1, 2, . . . , а f (ключ) это значение, дающее коллизию. Рехеширование продолжается до тех пор, пока очередная позиция массива с номером $f^{|}$ (ключ, i) не окажется пустой или равной снова своему начальному значению (в этом случае таблица полна). Линейное рехеширование применяют, если таблица никогда не бывает полностью заполненной (лучше, если она заполнена не более чем на 70%).

В случае разрешения коллизии в дополнительной таблице удобно использовать хеширование с цепочками. В таких таблицах элементы, которым соответствует одно и то же хеш-значение, связываются в цепочку – список, а в самом элементе таблицы хранится ссылка на список (см. рис.5.2).

Время поиска в хеш-таблице без коллизий постоянно – это время вычисления функции расстановки. Время поиска в таблицах с перемешиванием для метода линейного рехеширования зависит от коэффициента загрузки таблицы $k = n/N$, среднее число сравнений $e = (1 - k/2) (1 - k)$.

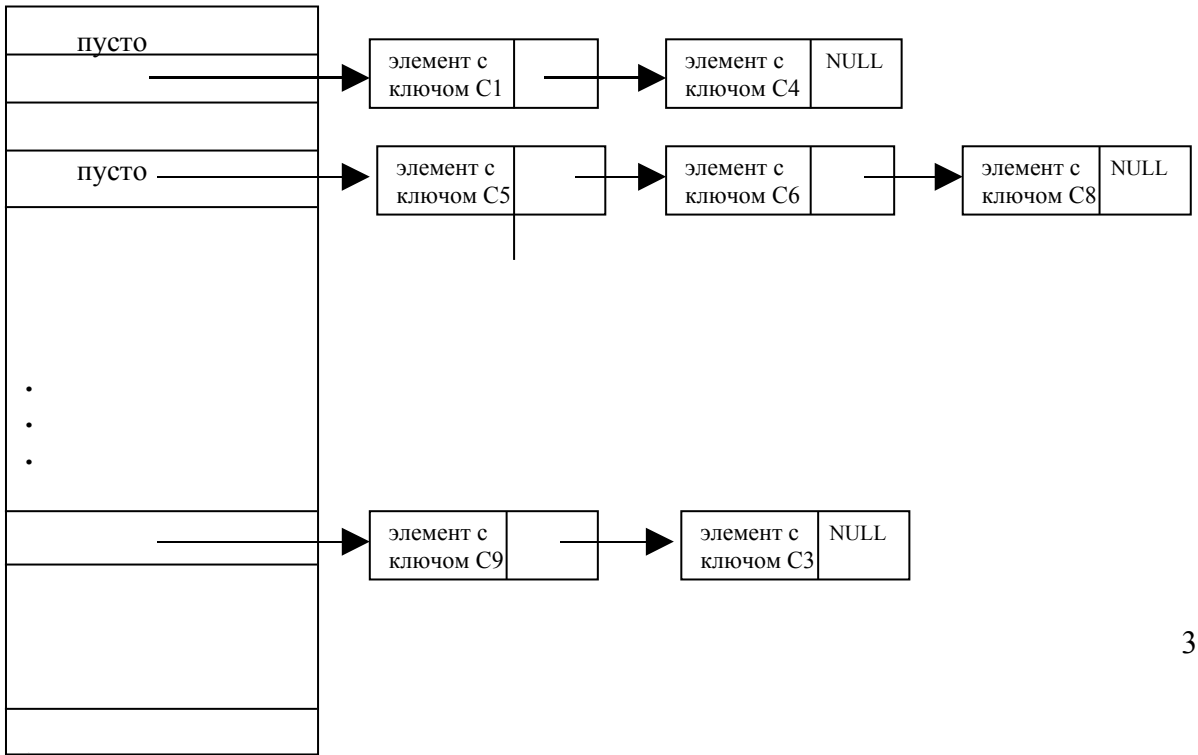


Рис.5.2.

2. **Варианты задания**

1. В файле PROG содержится текст программы на языке Си. Файл разделен на строки произвольной длины (но не более 256). Используя определения и описания, построить:
   1) таблицу имен. Элемент таблицы должен содержать имя программного объекта, его тип, размер памяти, требуемой объекту, число компонент, тип компонент (последние два признака для сложных типов). Организовать таблицу как:
   а) неупорядоченную;
   б) упорядоченную;
   в) таблицу с вычисляемым входом (таблица с перемешиванием);
   2) таблицу констант. Элемент таблицы содержит: значение константы, имя константы, тип константы, размер памяти. Организовать таблицу как:
   а) неупорядоченную;
   б) упорядоченную;
   в) таблицу с вычисляемым входом, считая, что тип констант целый и диапазон значений от -255 до 255.
2. В файле WORK содержатся результаты работы цеха за день. Элемент файла включает: шифр изделия (8 - символьный код), наименование изделия, количество (штук). Построить таблицу, содержащую результаты работы за день, считая ключом шифр изделия. Элемент таблицы имеет ту же структуру, что и элемент файла. Содержащаяся в файле информация с равными ключами должны быть помещена в таблицу один раз с общим количеством штук изделия. Организовать таблицу как:
   а) неупорядоченную;
   б) упорядоченную;
   в) таблицу с вычисляемым входом (таблица с перемешиванием).
3. В спортивных соревнованиях участвуют n команд. В файле SPORT содержатся прогнозы результатов соревнований. Каждый прогноз включает номер команды, занявшей первое место, номер команды, занявшей последнее место, номера команд, входящих в первую тройку сильнейших команд. Построить таблицу, содержащую проценты голосов, отданных командам - претендентам на первое место, командам - претендентам на последнее место и проценты голосов, отданных командам - претендентам на первую тройку. Организовать таблицу как:
   а) неупорядоченную;
   б) упорядоченную;

в) таблицу с вычисляемым входом.

3. **Контрольные вопросы**

1. Понятие таблицы.
2. Способы организации таблиц.
3. Операции над таблицами.
4. Чем отличается последовательный поиск в упорядоченной таблице от последовательного поиска в неупорядоченной таблице?
5. Чем отличается включение элемента в упорядоченную таблицу от включения элемента в неупорядоченную таблицу?
6. Хеш-функция - как ее построить?
7. Методы рехеширования.
8. Сравнительные оценки операций работы с таблицами.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 6

УПОРЯДОЧЕНИЕ ДАННЫХ

Цель работы: изучить основные алгоритмы упорядочения (сортировки) данных.

1. **Методические указания**

Сортировкой называется процесс (операция) перегруппировки объектов в некотором определенном порядке. Различают внутреннюю сортировку и внешнюю.

Алгоритмы внутренней сортировки упорядочивают данные, хранимые в оперативной памяти, алгоритмы внешней сортировки обрабатывают данные, хранимые во внешней памяти. В лабораторной работе № 6 рассматриваются алгоритмы внутренней сортировки данных, хранимых в статических таблицах.

Если таблица представлена множеством элементов $a_1, a_2, \ldots, a_n$, тогда сортировка есть перестановка элементов $a_{k1}, a_{k2}, \ldots, a_{kn}$, в которой над элементами установлено отношение порядка:

$$f(a_{k1}) <= f(a_{k2}) <= \ldots <= f(a_{kn})$$

или

$$f(a_{k1}) >= f(a_{k2}) >= \ldots >= f(a_{kn})$$

или какое-то другое отношение порядка (здесь f - упорядочивающая функция).

В простом случае f задается явно как компонент элемента таблицы, обычно этот компонент – ключ элемента. В таком случае говорят об упорядочении по ключам.

Сравнительная оценка сложности алгоритмов внутренней сортировки осуществляется по следующим параметрам:

- число сравнений ключей элементов;
- число перемещений элементов.

В эффективных алгоритмах стремятся сократить число сравнений и перестановок элементов, а также экономно использовать память. Поэтому алгоритмы сортировки производят перегруппировку элементов в исходном массиве. Алгоритм сортировки n данных, оперирующий сравнениями, имеет минимальную сложность n или выше (т.к. должно быть выполнено минимально n – 1 сравнений), максимальная сложность алгоритма, оперирующего сравнениями, не меньше n* log n.

Основные алгоритмы сортировки базируются на одном из трех способов:

1. Сортировка включением (вставками - by insertion). Исходное множество элементов делят на две части: уже упорядоченную и неупорядоченную. Вначале упорядоченная часть состоит, как правило, из одного элемента – первого элемента, а все остальные элементы находятся в неупорядоченной части. Из неупорядоченной части берут очередной элемент и включают его в упорядоченную часть, помещая (вставляя) на нужное место (т.е. так, чтобы выполнялось отношение порядка). Так продолжают до тех пор, пока последний элемент из неупорядоченной части не будет включен в упорядоченную часть.

Простой вариант сортировки включением – это метод прямого (простого) включения, улучшенный – сортировка методом Шелла.

Алгоритм прямого включения – здесь рассматриваются элементы таблицы (ее упорядоченной и неупорядоченной частей), отстоящие друг от друга на шаг равный 1:

```
{ for (i = 1; i <  T.n ; i ++ )
{ x = T.elem_i ;  c = ключ ( T.elem_i ) ;  j = i - 1;
   while ( j >= 0 && ключ ( T.elem_j ) > c )
      { T.elem_j+1 = T.elem_j ;  j = j - 1; }
  T.elem_j+1 = x; }
   }
```

Включаемый элемент сравнивается с элементами упорядоченной части, начиная с ее последнего элемента. Число просмотров (упорядоченной части) равно n – 1.

2. Сортировка выбором (by selection). Исходное множество элементов делят также на две части: уже упорядоченную и неупорядоченную. Из неупорядоченной части выбирают нужный элемент (например, с минимальным или максимальным значением в соответствии с отношением порядка) и помещают на очередное место в упорядоченную часть массива. Процесс продолжают до тех пор, пока в неупорядоченной части останется один элемент. Вначале неупорядоченная часть – весь исходный массив, а очередное место в упорядоченной части – первый (или последний) элемент. Простой вариант сортировки выбором – это метод прямого (простого) выбора; улучшенный вариант – сортировка выбором с использованием представления таблицы двоичным деревом (при этом дерево отображается в статической памяти) – сортировка с использованием структуры данных – дерево.

Алгоритм прямого выбора – здесь среди всех элементов неупорядоченной части осуществляется поиск нужного элемента:

```
{ for (i = 0; i <  T.n - 1; i ++)
   { min = T.elem_i ;  k = i;
      for (j = i + 1; j <  T.n ; j++)
            if ( ключ ( T.elem_j ) < ключ ( min ))
                        {min = T.elem_j ;   k = j; }
    T.elem_k = T.elem_i ;  T.elem_i = min;  }
    }
```

Число просмотров таблицы (неупорядоченной части) равно n – 1.

3. Сортировка обменом (by exchange). В исходном множестве элементов рассматриваются пары элементов. Если пара содержит инверсию, то она устраняется выполнением обмена этих элементов (инверсия – это пара индексов, на которой нарушено отношение порядка. Например, пусть отношение порядка – возрастание значений ключей элементов. Если i < j, а ключ (i) >= ключ (j) , то эта пара i и j содержит инверсию). Таким образом, после каждого просмотра хотя бы один элемент окажется на своем месте. Просмотры продолжаются до тех пор, пока в массиве не останется ни одной инверсии. Простой вариант сортировки обменом – метод прямого (простого) обмена («пузырьковая» сортировка), улучшенный вариант – быстрая сортировка.

Алгоритм прямого обмена – здесь сравниваются пары соседних элементов:

вариант 1:

```
{ for ( i = 0; i < T.n - 1; i ++ )
   for ( j =0; j < T.n - 1 - i ; j ++ )
     if ( ключ( T.elem_j ) > ключ ( T.elem_j+1) )
                { x = T.elem_j ;   T.elem_j = T.elem_j+1 ;
                     T.elem_j+1 = x; }
```

```
        }
```

В данном варианте всегда выполняется максимальное число сравнений. Число просмотров таблицы (неупорядоченной части) равно n – 1.

В случае, когда исходное множество является частично упорядоченным можно улучшить алгоритм прямого обмена, учитывая имеющийся частичный порядок:

вариант 2:

```
{ инверсия = 1; j =0;
  while ( инверсия )
   { инверсия = 0;
     for ( i = 0; i < T.n - 1 - j; i ++ )
        if ( ключ (T.elem_i ) > ключ ( T.elem_i+1 )
              { r = T_i ; T_i = T_i+1 ; T_i+1 = r ;
                инверсия = 1; j ++ ; }
   } }
```

В данных алгоритмах таблица T определена следующим образом:

```
struct table
    { type  elem [ N ];
      int    n;  }
  table  T;
```

Здесь поле n определяет фактическое число элементов в таблице.

2. Контрольные вопросы

1. Понятие упорядочения.
2. Характеристика прямых методов сортировки: включения, выбора, обмена.
3. Сравнительная оценка сложности прямых методов сортировки по числу сравнений и числу перемещений элементов.
4. Характеристика улучшенных методов сортировки, оценки их сложности.

3. Варианты задания

1. Упорядочить таблицу, построенную в лабораторной работе № 5 варианты 1.1.а, 1.2.а по ключу (ключ для 1.1.а – имя объекта; для 1.2.а – значение константы) методом:

а) прямого включения;
б) прямого выбора;
в) прямого обмена;
г) методом Шелла.

Используя раздел операторов, дополнить элементы таблицы числом раз использования каждого ключа. Для поиска элементов в таблице использовать:

а) последовательный поиск;
б) бинарный поиск.

2. Упорядочить таблицу, построенную в лабораторной работе № 5 варианты 1.1.а, 1.2.а по ключу (ключ для 1.1.а – имя объекта; для 1.2.а – значение константы) методом:

а) прямого включения;
б) прямого выбора;

в) прямого обмена;
г) методом Шелла.

Используя раздел операторов, напечатать в порядке возрастания ключей элементы таблицы, описанные, но не используемые в программе. Для поиска элементов в таблице использовать:

а) последовательный поиск;
б) бинарный поиск.

3. Упорядочить таблицу, построенную в лабораторной работе № 5 варианты 1.1.б, 1.1.в, 1.2.б, 1.2.в по новому ключу - по не возрастанию частоты использования элемента методом:

а) прямого включения;
б) прямого выбора;
в) прямого обмена;
г) методом Шелла.

Используя раздел операторов, дополнить элементы таблицы числом раз использования каждого ключа. Для поиска элементов в таблице использовать:

а) последовательный поиск;
б) бинарный поиск.

4. Упорядочить таблицу, построенную в лабораторной работе № 5 вариант 2.а по не убыванию значений ключа методом:

а) быстрой сортировки;
б) сортировки с использованием структуры дерево;
в) методом Шелла.

Включить информацию, хранимую в этой таблице, в таблицу продукции, имеющейся на складе. Таблица продукции упорядочена по возрастанию ключа. Элемент таблицы включает: шифр изделия (это ключ), наименование, количество (штук), цена (за штуку). Цену изделия брать из таблицы – прейскурант, элемент которой содержит: шифр изделия, цена (за штуку). Эта таблица также упорядочена по возрастанию шифров изделий. Для поиска элементов в таблице использовать:

а) последовательный поиск;
б) бинарный поиск.

5. Дополнить таблицу, построенную в лабораторной работе № 5 варианты 2.б, 2.в информацией о цене изделия. Цену изделия брать из таблицы – прейскурант, элемент которой содержит: шифр изделия, цена (за штуку). Эта таблица упорядочена по возрастанию шифров изделий. Упорядочить преобразованную таблицу по новому ключу – количество изделий методом:

а) прямого включения;
б) прямого обмена;
в) методом Шелла;
г) сортировки с использованием структуры дерево.

Для поиска элементов в таблице использовать:

а) последовательный поиск;
б) бинарный поиск.

6. Упорядочить таблицу, построенную в лабораторной работе № 5 вариант 3.а по убыванию процентов голосов, отданных командам – претендентам:

а) на первое место;
б) на последнее место;
в) на первую тройку.

Для упорядочения использовать метод:

а) быстрой сортировки;
б) сортировки с использованием структуры дерево;
в) методом Шелла.

7. Упорядочить таблицу, построенную в лабораторной работе № 5 варианты 3.б, 3.в по новому ключу – по возрастанию номеров команд, не вошедших в претенденты ни на первое, ни на последнее место.

Для упорядочения использовать метод:
а) быстрой сортировки;
б) сортировки с использованием структуры дерево;
в) методом Шелла.

Лабораторная работа 7. Функции и файлы

Вариант 1

Для хранения данных о планшетных сканерах описать структуру вида:

```
struct scan_info{
 char model[25];     // наименование модели
 int price;          // цена
 double x_size;      // гориз. размер обл. сканирования
 double y_size;      // вертик. размер обл. сканирования
 int optr;           // оптическое разрешение
 int grey;           // число градаций серого
};
```

Написать функцию, которая записывает в бинарный файл данные о сканере из приведенной структуры. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.
Написать функцию, которая извлекает из этого файла данные о сканере в структуру типа scan_info. Обязательный параметр — номер требуемой записи. Функция должна возвращать нулевое значение, если чтение прошло успешно, и –1 в противном случае.
Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) — 6–8 записей и выводящей на дисплей данные о запрошенной записи.
Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 2

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.
Написать функцию, которая записывает в бинарный файл данные о сканере из приведенной структуры. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.
Написать функцию, которая сортирует записи в описанном выше бинарном файле по одной из следующих характеристик: цена либо число градаций серого. Обязательный параметр — признак, задающий критерий сортировки.
Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) из не менее восьми записей и осуществляющий его сортировку.
Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 3

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.

Написать функцию, которая записывает в бинарный файл данные о сканере из приведенной структуры. Структура файла: в первых четырех байтах размещается значение типа long, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.

Написать функцию, которая сортирует записи в описанном выше бинарном файле по наименованию модели сканера.

Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) из не менее восьми записей и осуществляющий его сортировку.

Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 4

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.

Написать функцию, которая динамически выделяет память под массив структур (не меньше шести элементов), заполняет его данными в режиме диалога и записывает массив в бинарный файл. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.

Написать функцию, которая извлекает данные о сканере из описанного выше бинарного файла в структуру типа scan_info. Обязательный параметр — номер требуемой записи. Функция должна возвращать нулевое значение, если чтение прошло успешно, и –1 в противном случае.

Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) из не менее восьми записей и осуществляющий вывод на дисплей данных о требуемой записи.

Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 5

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.

Написать функцию, которая записывает данные о сканере из приведенной структуры в требуемую позицию в бинарном файле. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах. Запись может осуществляться в любую позицию, причем если между вводимой записью и последней (или началом файла) имеются пропуски, они заполняются нулями.

Написать функцию, которая «уплотняет» описанный выше бинарный файл путем удаления из него записей, содержащих все нули.
Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) из не менее шести записей и осуществляющий его уплотнение.
Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 6

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.
Написать функцию, которая динамически выделяет память под массив структур (не меньше шести элементов), заполняет его данными в режиме диалога, и записывает массив в бинарный файл. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.
Написать функцию, которая запрашивает данные о сканере в режиме диалога и замещает записи в бинарном файле по заданному номеру. Обязательный параметр — номер замещаемой записи. Функция должна возвращать нулевое значение, если запись прошла успешно, и –1 в противном случае.
Привести пример программы, создающей файл с данными о сканерах (данные вводятся с клавиатуры) из не менее восьми записей и осуществляющий вставку новых данных о сканере.
Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 7

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.
Написать функцию, которая записывает в бинарный файл данные о сканере из приведенной структуры. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.
Написать функцию, которая вводит данные о сканере с клавиатуры в структуру типа scan_info, и если данных об этом сканере отсутствуют в файле, помещает содержимое структуры в конец файла; в противном случае выдает соответствующее сообщение.
Привести пример программы, создающей файл с данными о сканерах (данные вводятся из текстового файла) — 6–8 записей и дополняющей файл записями о 2–3 сканерах, вводимых с клавиатуры.
Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 8

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.

Написать функцию, которая записывает в бинарный файл данные о сканере из приведенной структуры. Структура файла: в первых двух байтах размещается значение типа int, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.

Написать функцию, которая вводит данные о сканере с клавиатуры в структуру типа scan_info и помещает ее содержимое на место первой записи в файле. Файл должен существовать. При этом, запись ранее занимавшая первую позицию, помещается на вторую, вторую на третью и т. д.

Привести пример программы, создающей файл с данными о сканерах (данные вводятся из текстового файла) — 6–8 записей и дополняющей этот файл 1–2 новыми записями, вводимыми с клавиатуры.

Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Вариант 9

Для хранения данных о планшетных сканерах описать структуру вида, описанного в варианте 1.

Написать функцию, которая запрашивает количество сканеров, информация о которых будет вводиться, динамически выделяет память под массив структур соответствующего размера и заполняет его данными в режиме диалога (с клавиатуры). При этом имя сканера может содержать пробелы.

Написать функцию, которая записывает данный массив в создаваемый бинарный файл. Если цена сканера меньше 200, то данные об этом сканере в файл не записываются. Информация об остальных сканерах помещается в бинарный файл, причем сначала пишутся данные о всех сканерах, имя которых начинается с заглавной буквы, а затем — с прописной.

Структура файла: в первых четырех байтах размещается значение типа long, определяющее количество сделанных в файл записей; далее без пропусков размещаются записи о сканерах.

Привести пример программы, создающей файл с данными о сканерах и осуществляющий вывод на дисплей данных о требуемой записи (либо всех, либо по номеру).

Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается

Вариант 10

Для хранения данных о ноутбуках описать структуру вида:

```
struct NOTEBOOK{
 char model[21];  // наименование
 struct size{     // габаритные размеры
    float x;
    float y;
```

```
    float z;
  };
  float w;         // вес
  int price;       // цена
 }
```

Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в стуктуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла. Структура бинарного файла: первые два байта (целое) — число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется чтение данных только для тех ноутбуков, частота процессора которых больше 120МГц, и запись в бинарный файл по убыванию цены.

Вариант 11

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 10.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла. Структура бинарного файла: первые два байта (целое) — число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется чтение данных только для тех ноутбуков, объем HDD которых меньше 1 Гб, и запись считанных данных в бинарный файл в алфавитном порядке по наименованию.

Вариант 12

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 10.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла. Структура бинарного файла: первые два байта (целое) — число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, целое количество которых в одном кубическом метре не превышает 285-ти штук.

Вариант 13

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 10.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.

Структура бинарного файла: первые два байта (целое) — число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, максимальный объем ОЗУ которых не менее 40Мб, отсортированных по объему.

Вариант 14

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 10.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.
Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, диагональ дисплея которых больше одиннадцати дюймов.

Вариант 15

Для хранения данных о ноутбуках описать структуру вида:

```
struct NOTEBOOK{
 struct disp_res{ // разрешающая способность дисплея
    int x;        // по горизонтали
    int y;        // по вертикали
 };
 int f;           // частота регенерации
 float d;         // размер диагонали дисплея
 int price;       // цена
 char model[21]; // наименование
}
```

Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.
Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, вес которых менее 7кг, отсортированных в порядке возрастания цены.

Вариант 16

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 15.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.

Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, объем видеопамяти которых 2Мб, отсортированных в порядке уменьшения тактовой частоты процессора.

Вариант 17

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 15.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.
Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, объем HDD которых больше 1Гб, отсортированных в порядке возрастания размера диагонали дисплея.

Вариант 18

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 15.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла.
Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, тактовая частота процессора которых больше 120МГц, отсортированных в порядке уменьшения веса.

Вариант 19

Для хранения данных о ноутбуках описать структуру вида:

```
struct NOTEBOOK{
 struct disp_res{ // разрешающая способность дисплея
    int x;        // по горизонтали
    int y;        // по вертикали
 };
 int f;           // частота регенерации
 float d;         // размер диагонали дисплея
 float hdd;       // объем диска
 char model[21]; // наименование
}
```

Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию,

которая записывает содержимое структуры в конец бинарного файла. Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, тактовая частота процессора которых больше 120МГц, отсортированные в порядке возрастания цены.

Вариант 20

Для хранения данных о ноутбуках описать структуру вида, описанного в варианте 19.
Написать функцию, которая читает данные о ноутбуках из файла note.txt (см. с. <$Rprim_note>) в структуру приведенного вида. Написать функцию, которая записывает содержимое структуры в конец бинарного файла. Структура бинарного файла: первые два байта — целое число записей в файле; далее записи в формате структуры NOTEBOOK.
Написать программу, в которой на основе разработанных функций осуществляется запись в двоичный файл данных только о тех ноутбуках, цена которых больше 3500$, отсортированные в порядке возрастания тактовой частоты процессора.
<$Mprim_note>Пример файла note.txt:

```
Aser Note Light     2699 5.6 02.0x11.8x08.3 100 40 10.4 1 1024x0768 60 0.774
ASW ND5123T         3489 7.2 02.3x11.8x10.1 133 32 12.1 2 1024x0768 70 1.300
ARMNote TS80CD      3699 7.2 02.0x11.5x08.8 133 64 11.3 1 1024x0768 75 1.300
AST Ascentia P50    4499 7.5 02.3x11.3x09.0 133 40 11.3 1 0800x0600 70 0.774
BSI NP8657D         2605 8.0 02.3x11.8x09.3 133 40 11.3 1 1024x0768 60 0.810
BSI NP5265A         3765 8.2 02.5x12.0x09.0 150 32 12.1 2 1024x0768 70 1.300
Dell Xpi P100SD     3459 6.0 02.3x11.0x08.8 100 40 10.3 1 1024x0768 60 0.773
Digital HiNote      4799 4.0 01.3x11.0x08.8 120 40 10.4 1 0800x0600 56 1.000
Gateway Solo S5     4499 5.6 02.0x11.9x08.8 133 40 11.3 2 1024x0768 60 0.686
Hertz Z-Optima NB   3995 8.0 02.3x11.9x09.0 150 40 11.2 2 1024x0768 75 1.000
HP OmniBook 5500    6120 7.1 02.0x11.5x09.0 133 64 11.4 1 1024x0768 75 1.300
IBM ThinkPad 560    3749 4.1 01.3x11.8x08.8 120 40 12.1 2 1024x0768 85 0.774
NEC Versa 4080H     4780 6.6 02.3x11.8x09.5 120 48 10.4 1 0800x0600 70 0.776
Polywell Poly 500   3300 7.9 02.3x11.9x09.0 120 40 10.4 1 1024x0768 72 1.000
Samsung SENS 810    3667 8.7 02.3x11.5x09.5 100 32 11.4 2 1024x0768 75 0.773
Twinhead Slimnote   2965 7.4 02.0x11.5x08.0 075 64 10.4 1 1024x0768 70 0.772
```

<$Mopis_note>Описание файла note.txt:
В файле note.txt содержится текстовая информация о ноутбуках. Каждая строка содержит данные об одной модели. Данные в строке размещаются в следующих полях:
1 : 20 — наименование модели;
21 : 24 — цена в долларах (целое число);
26 : 28 — масса ноутбука в кг (число с десятичной точкой из четырех символов);
30 : 43 — габаритные размеры ноутбука в дюймах (ВЫСОТАхДЛИНАхШИРИНА — три числа с десятичной точкой (4 символа, включая точку, разделенные 'x');

44 : 47 — частота процессора в Гц (целое число из трех символов);
49 : 50 — максимальный объем ОЗУ в Мб (целое число из двух символов);
52 : 55 — размер диагонали дисплея в дюймах (число с десятичной точкой из четырех символов, включая точку);
57 — размер видеопамяти в Мб — целое число из одного символа;
59 : 67 — разрешающая способность дисплея в пикселах (два целых числа, разделенные 'x');
69 : 70 — частота регенерации дисплея в Гц (целое число из двух символов);
72 : 76 — объем HDD в Гб (число с десятичной точкой из пяти символов).
Все не описанные позиции заполнены пробелами.

Лабораторная работа 8. Классы

Примечание: во всех классах должно быть хотя бы одно поле, память под которое выделяется динамически.

Вариант 1

Описать класс, реализующий стек. Написать программу, использующую этот класс для моделирования Т-образного сортировочного узла на железной дороге. Программа должна разделять на два направления состав, состоящий из вагонов двух типов (на каждое направление формируется состав из вагонов одного типа). Предусмотреть возможность формирования состава из файла и с клавиатуры.

Вариант 2

Описать класс, реализующий бинарное дерево, обладающее возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, а также последовательного доступа ко всем элементам.
Написать программу, использующую этот класс для представления англо-русского словаря. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса. Предусмотреть возможность формирования словаря из файла и с клавиатуры.

Вариант 3

Построить систему классов для описания плоских геометрических фигур: круг, квадрат, прямоугольник. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и вращения на заданный угол.
Написать программу, демонстрирующую работу с этими классами. Программа должна содержать меню, позволяющее осуществить проверку всех методов классов.

Вариант 4

Построить описание класса, содержащего информацию о почтовом адресе организации. Предусмотреть возможность раздельного изменения составных частей адреса, создания и уничтожения объектов этого класса. Длину строк задавать динамически.
Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 5

Составить описание класса для представления комплексных чисел. Обеспечить выполнение операций сложения, вычитания и умножения комплексных чисел.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 6

Составить описание класса для объектов-векторов, задаваемых координатами концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 7

Составить описание класса прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменение размеров, построение наименьшего прямоугольника, содержащего два заданных прямоугольника, и прямоугольника, являющегося общей частью (пересечением) двух прямоугольников.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 8

Составить описание класса для определения одномерных массивов целых чисел (векторов). Предусмотреть возможность обращения к отдельному элементу массива с контролем выхода за пределы массива, возможность задания произвольных границ индексов при создании объекта и выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов, умножения и деления всех элементов массива на скаляр, вывода на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 9

Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы массива, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением

повторяющихся элементов, вывод на экран элемента массива по заданному индексу и всего массива.
Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 10

Составить описание класса многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, вывод на экран описания многочлена.
Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 11

Составить описание класса одномерных массивов строк, каждая строка задается длиной и указателем на выделенную для нее память. Предусмотреть возможность обращения к отдельным строкам массива по индексам, контроль выхода за пределы массивов, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, вывод на экран элемента массива и всего массива.
Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 12

Составить описание класса, обеспечивающего представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы.
Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 13

Написать класс для эффективной работы со строками, позволяющий форматировать и сравнивать строки, хранить в строках числовые значения и извлекать их. Для этого необходимо реализовать:

3. перегруженные операторы присваивания и конкатенации;
4. операции сравнения и приведения типов;
5. преобразование в число любого типа;
6. форматный вывод строки.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 14

Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (например, по автору или по году издания), добавления книг в библиотеку, удаления книг из нее, сортировки книг по разным полям. Длину строк задавать динамически.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 15

Описать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям. Длину строк задавать динамически.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 16

Описать класс «студенческая группа». Предусмотреть возможность работы с переменным числом студентов, поиска студента по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям. Длину строк задавать динамически.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 17

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

7. сложение, вычитание, умножение, деление (+, –, *, /) (умножение и деление как на другую матрицу, так и на число);
8. комбинированные операции присваивания (+=, –=, *=, /=);
9. операции сравнения на равенство/неравенство;
10. операции вычисления обратной и транспонированной матрицы, операцию возведения в степень;
11. методы вычисления детерминанта и нормы;
12. методы, реализующие проверку типа матрицы (квадратная, диагональная, нулевая, единичная, симметрическая, верхняя треугольная, нижняя треугольная);
13. операции ввода/вывода в стандартные потоки.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 18

Описать класс «множество», позволяющий выполнять основные операции — добавление и удаление элемента, пересечение, объединение и разность множеств.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Вариант 19

Описать класс, реализующий стек. Написать программу, использующую этот класс для отыскания прохода по лабиринту.

Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. После отыскания прохода программа печатает найденный путь в виде координат квадратов.

Вариант 20

Описать класс «предметный указатель». Каждая компонента указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть возможность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Лабораторная работа 9. Наследование

Примечание: во всех классах должно быть хотя бы одно поле, память под которое выделяется динамически.

Вариант 1

Создать класс CFile, инкапсулирующий в себе такие функции работы с файлами, как Open, Close, Seek, Read, Write, GetPosition и GetLength. На базе этого класса создать производный класс CMyDataFile — файл, содержащий в себе данные некоторого определенного типа MyData, а также заголовок, облегчающий доступ к этому файлу.

Написать программу, демонстрирующую работу с этими классами. Программа должна содержать меню, позволяющее осуществить проверку всех методов классов.

Вариант 2

Создать класс CPoint — точка. На его основе создать классы CcoloredPoint и CLine. На основе класса CLine создать класс CColoredLine и класс CPolyLine — многоугольник. Все классы должны иметь методы для установки и

получения значений всех координат, а также изменения цвета и получения текущего цвета.
Написать демонстрационную программу, в которой будет использоваться список объектов этих классов в динамической памяти.

Вариант 3

Создать абстрактный класс CVehicle. На его основе реализовать классы CPlane, CCar и CShip. Классы должны иметь возможность задавать и получать координаты, параметры средств передвижения (цена, скорость, год выпуска). Для самолета должна быть определена высота, для самолета и корабля — количество пассажиров. Для корабля — порт приписки.
Написать программу, создающую список объектов этих классов в динамической памяти. Программа должна содержать меню, позволяющее осуществить проверку всех методов классов.

Вариант 4

1. Описать базовый класс «Элемент».

Поля:

14. имя элемента (указатель на строку символов);
15. количество входов элемента;
16. количество выходов элемента.

Методы:

17. конструктор класса;
18. деструктор класса;
19. метод, задающий имя элемента.

2. На основе класса «Элемент» описать производный класс «Комбинационный», представляющий собой комбинационный элемент (двоичный вентиль), который может иметь несколько входов и один выход.

Поля:

20. указатель, используемый для динамического размещения полей, содержащих значения входов.

Методы:

21. конструктор;
22. конструктор копирования;
23. деструктор;
24. метод, задающий значение на входах экземпляра класса;
25. метод, позволяющий опрашивать состояние отдельного входа экземпляра класса;
26. метод, вычисляющий значение выхода (по варианту задания);
27. 3. На основе класса «Элемент» описать производный класс «Память», представляющих собой триггер. Триггер имеет входы, соответствующие типу триггера (см. ниже вариант задания), и входы установки и сброса. Все триггеры считаются синхронными, сам синхровход в состав триггера не включается.Поля:массив значений входов объекта класса (задается статически), в массиве учитываются все входы (управляющие и информационные);
28. состояние на прямом выходе триггера;
29. состояние на инверсном выходе триггера.

Методы:

30. конструктор (по умолчанию сбрасывает экземпляр класса);
31. конструктор копирования;

32. деструктор;
33. метод, задающий значение на входах экземпляра класса;
34. методы, позволяющие опрашивать состояния отдельного входа экземпляра класса;
35. метод, вычисляющий состояние экземпляра класса (по варианту задания) в зависимости от текущего состояния и значений на входах;
36. метод, переопределяющий операцию == для экземпляров класса.

4. Создать класс «Регистр», используя класс «Память» как включаемый класс.

Поля:

37. состояние входа «Сброс» — один для экземпляра класса;
38. состояние входа «Установка» — один для экземпляра класса;
39. статический массив типа «Память» заданной в варианте размерности;
40. статический(е) массив(ы), содержащие значения на соответствующих входах элементов массива типа «Память».

Методы:

41. метод, задающий значение на входах экземпляра класса (желательно в качестве параметров передавать методу указатели на массивы значений);
42. метод, позволяющий опрашивать состояние отдельного выхода экземпляра класса;
43. метод, вычисляющий значение нового состояния экземпляра класса;

Все поля классов «Элемент», «Комбинационный» и «Память» должны быть описаны с ключевым словом private.

В задании перечислены только обязательные члены и методы класса. Можно задавать дополнительные члены и методы, если они не отменяют обязательные и обеспечивают дополнительные удобства при работе с данными классами, например, описать функции вычисления выхода/состояния как виртуальные.

5. Для проверки функционирования созданных классов написать программу, использующую эти классы. В программе должны быть продемонстрированы все свойства созданных классов.

Конкретный тип комбинационного элемента, тип триггера и разрядность регистра выбираются в соответствии с вариантом задания:

Вариант	Комбинационный элемент	Число входов	Триггер	Разрядность регистра
1	И-НЕ	4	RS	8
2	ИЛИ	5	RST	10
3	МОД2-НЕ	6	D	12
4	И	8	T	8
5	ИЛИ-НЕ	8	V	9
6	И	4	RS	10
7	ИЛИ-НЕ	5	JK	11
8	МОД2	5	D	8
9	И	4	T	10
10	ИЛИ	3	JK	8
11	И-НЕ	3	RS	12

12	ИЛИ-НЕ	4	RST	4
13	МОД2	5	D	10
14	МОД2-НЕ	6	T	10
15	ИЛИ-НЕ	8	V	10
16	И	8	JK	6
17	И-НЕ	8	RS	10
18	ИЛИ	8	T	10
19	МОД2	6	JK	8
20	МОД2-НЕ	5	V	10

Вариант 5

Описать базовый класс СТРОКА.

Обязательные поля класса:

44. указатель на char — хранит адрес динамически выделенной памяти для размещения символов строки;
45. значение типа int — хранит длину строки в байтах.

Обязательные методы должны выполнять следующие действия:

46. конструктор без параметров;
47. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
48. конструктор, принимающий в качестве параметра символ;
49. конструктор копирования;
50. получение длины строки;
51. очистка строки (сделать строку пустой);
52. деструктор.

Описать производный от СТРОКА класс СТРОКА_ИДЕНТИФИКАТОР. Строки данного класса строятся по правилам записи идентификаторов в языке С и могут включать в себя только те символы, которые могут входить в состав С-идентификаторов. Если исходные данные противоречат правилам записи идентификатора, то создается пустая СТРОКА_ИДЕНТИФИКАТОР.

Обязательные методы:

53. конструктор без параметров;
54. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
55. конструктор, принимающий в качестве параметра символ;
56. конструктор копирования;
57. перевод всех символов строки в верхний регистр;
58. перевод всех символов строки в нижний регистр;
59. поиск первого вхождения символа в строку;
60. деструктор.

Переопределить следующие операции:

61. присваивание (=);
62. сложение (+) — операция конкатенации строк;
63. вычитание (–) — из строки (первый операнд) удаляются все символы, входящие в строку — второй операнд, при этом может получиться пустая строка;

64. оператор > — проверка на больше. Строка считается больше другой, если код символа первой строки в i-й позиции (i изменяется от 0 до n–1, где n — длина более короткой строки) больше кода символа в той же позиции во второй строке, длины строк могут не совпадать.

65. оператор < — проверка на меньше. Строка считается меньше другой, если код символа первой строки в i-й позиции (i изменяется от 0 до n–1, где n — длина более короткой строки) меньше кода символа в той же позиции кода символа в той же позиции во второй строке, длины строк могут не совпадать.

Разработчик вправе вводить любое (с обоснованием необходимости) число дополнительных полей и методов.

Написать тестовую программу, которая:

66. динамически выделяет массив указателей на базовый класс (4–6);

67. в режиме диалога заполняет этот массив указателями на производные классы, при этом экземпляры производных классов создаются динамически с заданием начальных значений;

68. для созданных экземпляров производных классов выполняет проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Для конструкторов копирования каждого класса предусмотреть диагностическую печать количества его вызовов в определенное место дисплея (рекомендуется использовать статические члены класса).

Режим диалога обеспечивается с помощью иерархического меню.

Вариант 6

Описать базовый класс СТРОКА.

Обязательные поля класса:

69. указатель на char — хранит адрес динамически выделенной памяти для размещения символов строки;

70. значение типа int — хранит длину строки в байтах.

Обязательные методы должны выполнять следующие действия:

71. конструктор без параметров;

72. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);

73. конструктор, принимающий в качестве параметра символ;

74. конструктор копирования;

75. получение длины строки;

76. очистка строки (сделать строку пустой);

77. деструктор.

Описать производный от СТРОКА класс БИТОВАЯ_СТРОКА.

Строки данного класса могут содержать только символы '0' или '1'. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, БИТОВАЯ_СТРОКА принимает нулевое значение.

Содержимое данных строк рассматривается как двоичное число.

Отрицательные числа хранятся в дополнительном коде.

Обязательные методы:

78. конструктор без параметров;

79. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);

80. конструктор копирования;

81. деструктор;

82. изменение знака на противоположный (перевод числа в дополнительный код).

Переопределить следующие операции (длина строки результата равна длине большей из строк; в случае необходимости более короткая битовая строка расширяется влево знаковым разрядом):

83. присваивание (**=**);
84. сложение (**+**) — арифметическая сумма строк;
85. операция (**==**) — проверка на равенство.

Разработчик вправе вводить любое (с обоснованием необходимости) число дополнительных полей и методов.

Написать тестовую программу, которая:

86. динамически выделяет массив указателей на базовый класс (4–6);
87. в режиме диалога заполняет этот массив указателями на производные классы, при этом экземпляры производных классов создаются динамически с заданием начальных значений;
88. для созданных экземпляров производных классов выполняет проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Для конструкторов копирования каждого класса предусмотреть диагностическую печать количества его вызовов в определенное место дисплея (рекомендуется использовать статические члены класса).

Режим диалога обеспечивается с помощью иерархического меню.

Вариант 7

Описать базовый класс СТРОКА.

Обязательные поля класса:

89. указатель на char — хранит адрес динамически выделенной памяти для размещения символов строки;
90. значение типа int — хранит длину строки в байтах.

Обязательные методы должны выполнять следующие действия:

91. конструктор без параметров;
92. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
93. конструктор, принимающий в качестве параметра символ;
94. конструктор копирования;
95. получение длины строки;
96. очистка строки (сделать строку пустой);
97. деструктор.

Описать производный от СТРОКА класс ДЕСЯТИЧНАЯ_СТРОКА. Строки данного класса могут содержать только символы десятичных цифр и символы – и +, задающие знак числа. Символы – или + могут находиться только в первой позиции числа, причем символ + может отсутствовать, в этом случае число считается положительным. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, ДЕСЯТИЧНАЯ_СТРОКА принимает нулевое значение. Содержимое данных строк рассматривается как десятичное число.

Обязательные методы:

98. конструктор без параметров;
99. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
100. конструктор копирования;
101. деструктор;

102. метод, определяющий, можно ли представить данное число в формате int;

Переопределить следующие операции:

103. присваивание (=);

104. вычитание (–) — арифметическая разность строк;

105. операция > — проверка на больше (по значению);

106. операция < — проверка на меньше (по значению);

Разработчик вправе вводить любое (с обоснованием необходимости) число дополнительных полей и методов.

Написать тестовую программу, которая:

107. динамически выделяет массив указателей на базовый класс (4–6);

108. в режиме диалога заполняет этот массив указателями на производные классы, при этом экземпляры производных классов создаются динамически с заданием начальных значений;

109. для созданных экземпляров производных классов выполняет проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Для конструкторов копирования каждого класса предусмотреть диагностическую печать количества его вызовов в определенное место дисплея (рекомендуется использовать статические члены класса).

Режим диалога обеспечивается с помощью иерархического меню.

Вариант 8

Описать базовый класс СТРОКА.

Обязательные поля класса:

110. указатель на char — хранит адрес динамически выделенной памяти для размещения символов строки;

111. значение типа int — хранит длину строки в байтах.

Обязательные методы должны выполнять следующие действия:

112. конструктор без параметров;

113. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);

114. конструктор, принимающий в качестве параметра символ;

115. конструктор копирования;

116. получение длины строки;

117. очистка строки (сделать строку пустой);

118. деструктор.

Производный от СТРОКА класс КОМПЛЕКСНОЕ_ЧИСЛО.

Строки данного класса состоят из двух полей, разделенных символом i. Первое поле задает значение реальной части числа, а второе — мнимой. Каждое из полей может содержать только символы десятичных цифр и символы – и +, задающие знак числа. Символы – или + могут находиться только в первой позиции числа, причем символ + может отсутствовать, в этом случае число считается положительным. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, КОМПЛЕКСНОЕ_ЧИСЛО принимает нулевое значение.

Примеры строк: 33i12, –7i100, +5i-21.

Обязательные методы:

119. конструктор без параметров;

120. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);

121. конструктор копирования;

122. деструктор.

Переопределить следующие операции:

123. присваивание (**=**);

124. операция (**==**) — проверка на равенство;

125. умножение (*) — умножение чисел.

Разработчик вправе вводить любое (с обоснованием необходимости) число дополнительных полей и методов.

Написать тестовую программу, которая:

126. динамически выделяет массив указателей на базовый класс (4–6);

127. в режиме диалога заполняет этот массив указателями на производные классы, при этом экземпляры производных классов создаются динамически с заданием начальных значений;

128. для созданных экземпляров производных классов выполняет проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Для конструкторов копирования каждого класса предусмотреть диагностическую печать количества его вызовов в определенное место дисплея (рекомендуется использовать статические члены класса).

Режим диалога обеспечивается с помощью иерархического меню.

Лабораторная работа 10

Контейнерные классы

Выполнить задания с использованием соответствующих контейнерных классов библиотеки.

Вариант 1

Составить программу, которая содержит динамическую информацию о наличии автобусов в автобусном парке.

Сведения о каждом автобусе содержат:

129. номер автобуса;

130. фамилию и инициалы водителя;

131. номер маршрута.

Программа должна обеспечивать:

132. начальное формирование данных о всех автобусах в парке в виде списка;

133. при выезде каждого автобуса из парка вводится номер автобуса, и программа удаляет данные об этом автобусе из списка автобусов, находящихся в парке, и записывает эти данные в список автобусов, находящихся на маршруте;

134. при въезде каждого автобуса в парк вводится номер автобуса, и программа удаляет данные об этом автобусе из списка автобусов, находящихся на маршруте, и записывает эти данные в список автобусов, находящихся в парке;

135. по запросу выдаются сведения об автобусах, находящихся в парке, или об автобусах, находящихся на маршруте.

Вариант 2

Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

136. номер УДК;
137. фамилию и инициалы автора;
138. название;
139. год издания;
140. количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

141. начальное формирование данных о всех книгах в библиотеке в виде двоичного дерева;
142. добавление данных о книгах, вновь поступающих в библиотеку;
143. удаление данных о списываемых книгах;
144. по запросу выдаются сведения о наличии книг в библиотеке, упорядоченные по годам издания.

Вариант 3

Составить программу, которая содержит текущую информацию о заявках на авиабилеты.

Каждая заявка содержит:

145. пункт назначения;
146. номер рейса;
147. фамилию и инициалы пассажира;
148. желаемую дату вылета.

Программа должна обеспечивать:

149. хранение всех заявок в виде списка;
150. добавление заявок в список;
151. удаление заявок;
152. вывод заявок по заданному номеру рейса и дате вылета;
153. вывод всех заявок.

Вариант 4

Составить программу, которая содержит текущую информацию о заявках на авиабилеты.

Каждая заявка содержат:

154. пункт назначения;
155. номер рейса;
156. фамилию и инициалы пассажира;
157. желаемую дату вылета;

Программа должна обеспечивать:

158. хранение всех заявок в виде двоичного дерева;
159. добавление и удаление заявок;
160. по заданному номеру рейса и дате вылета вывод заявок с их последующим удалением;
161. вывод всех заявок.

Вариант 5

Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

162. номер УДК;

163. фамилию и инициалы автора;

164. название;

165. год издания;

166. количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

167. начальное формирование данных о всех книгах в библиотеке в виде списка;

168. при взятии каждой книги вводится номер УДК, и программа уменьшает значение количества книг на единицу или выдает сообщение о том, что требуемой книги в библиотеке нет, или требуемая книга находится на руках;

169. при возвращении каждой книги вводится номер УДК, и программа увеличивает значение количества книг на единицу;

170. по запросу выдаются сведения о наличии книг в библиотеке.

Вариант 6

Составить программу, которая содержит динамическую информацию о наличии автобусов в автобусном парке.

Сведения о каждом автобусе содержат:

171. номер автобуса;

172. фамилию и инициалы водителя;

173. номер маршрута;

174. признак того, где находится автобус — на маршруте или в парке.

Программа должна обеспечивать:

175. начальное формирование данных о всех автобусах в виде списка;

176. при выезде каждого автобуса из парка вводится номер автобуса, и программа устанавливает значение признака «автобус на маршруте»;

177. при въезде каждого автобуса в парк вводится номер автобуса, и программа устанавливает значение признака «автобус в парке»;

178. по запросу выдаются сведения об автобусах, находящихся в парке, или об автобусах, находящихся на маршруте.

Вариант 7

Составить программу, отыскивающую проход по лабиринту.

Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице.

Программа находит проход через лабиринт, двигаясь от заданного входа. После отыскания прохода программа выводит найденный путь в виде координат квадратов. Для хранения пути использовать стек.

Вариант 8

Гаражная стоянка имеет одну стояночную полосу, причем единственный въезд и единственный выезд находятся в одном конце полосы. Если владелец автомашины приходит забрать свой автомобиль, который не является ближайшим к выходу, то все автомашины, загораживающие проезд, удаляются, машина данного владельца выводится со стоянки, а другие машины возвращаются на стоянку в исходном порядке.

Написать программу, которая моделирует процесс прибытия и отъезда машин. Прибытие или отъезд автомашины задается командной строкой, которая содержит признак прибытия или отъезда и номер машины. Программа должна выводить сообщение при прибытии или выезде любой машины. При выезде автомашины со стоянки сообщение должно содержать число раз, которое машина удалялась со стоянки для обеспечения выезда других автомобилей.

Вариант 9

Составить программу, моделирующую заполнение гибкого магнитного диска.

Общий объем памяти на диске 360 Кбайт. Файлы имеют произвольную длину от 18 байт до 32 Кбайт. В процессе работы файлы либо записываются на диск, либо удаляются с него.

В начале работы файлы записываются подряд друг за другом. После удаления файла на диске образуется свободный участок памяти, и вновь записываемый файл либо размещается на свободном участке, либо, если файл не вмещается в свободный участок, размещается после последнего записанного файла.

В случае, когда файл превосходит длину самого большого свободного участка, выдается аварийное сообщение. Требование на запись или удаление файла задается в командной строке, которая содержит имя файла, его длину в байтах, признак записи или удаления. Программа должна выдавать по запросу сведения о занятых и свободных участках памяти на диске.

Указание: следует создать список занятых участков и список свободных участков памяти на диске.

Вариант 10

В файловой системе каталог файлов организован как линейный список.

Для каждого файла в каталоге содержатся следующие сведения:

179. имя файла;

180. дата создания;

181. количество обращений к файлу.

Составить программу, которая обеспечивает:

182. начальное формирование каталога файлов;

183. вывод каталога файлов;

184. удаление файлов, дата создания которых меньше заданной;

185. выборку файла с наибольшим количеством обращений.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 11

Предметный указатель организован как линейный список.

Каждая компонента указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти.

Составить программу, которая обеспечивает:

186. начальное формирование предметного указателя;

187. вывод предметного указателя;

188. вывод номеров страниц для заданного слова.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 12

Текст помощи для некоторой программы организован как линейный список. Каждая компонента текста помощи содержит термин (слово) и текст, содержащий пояснения к этому термину. Количество строк текста, относящихся к одному термину, от одной до пяти.

Составить программу, которая обеспечивает:

189. начальное формирование текста помощи;

190. вывод теста помощи;

191. вывод поясняющего текста для заданного термина.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 13

Картотека в бюро обмена квартир организована как линейный список. Сведения о каждой квартире содержат:

192. количество комнат;

193. этаж;

194. площадь;

195. адрес.

Составить программу, которая обеспечивает:

196. начальное формирование картотеки;

197. ввод заявки на обмен;

198. поиск в картотеке подходящего варианта: при равенстве количества комнат и этажа и различии площадей в пределах 10% выводится соответствующая карточка и удаляется из списка, в противном случае поступившая заявка включается в список;

199. вывод всего списка.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 14

Англо-русский словарь построен как двоичное дерево.

Каждая компонента содержит английское слово, соответствующее ему русское слово и счетчик количества обращений к данной компоненте. Первоначально дерево формируется согласно английскому алфавиту. В процессе эксплуатации словаря при каждом обращении к компоненте в счетчик обращений добавляется единица.

Составить программу, которая:

200. обеспечивает начальный ввод словаря с конкретными значениями счетчиков обращений;

201. формирует новое представление словаря в виде двоичного дерева по следующему алгоритму: а) в старом словаре ищется компонента с наибольшим значением счетчика обращений; б) найденная компонента заносится в новый словарь и удаляется из старого; в) переход к п. а) до исчерпания исходного словаря.

202. производит вывод исходного и нового словарей.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 15

Анкета для опроса населения содержит две группы вопросов.

Первая группа содержит сведения о респонденте:

203. возраст;

204. пол;

205. образование (начальное, среднее, высшее).

Вторая группа содержит собственно вопрос анкеты, ответ на который либо ДА, либо НЕТ.

Составить программу, которая:

206. обеспечивает начальный ввод анкет и формирует из них линейный список;

207. на основе анализа анкет выдает ответы на следующие вопросы: а) сколько мужчин старше 40 лет, имеющих высшее образование, ответили ДА на вопрос анкеты; а) сколько женщин моложе 30 лет, имеющих среднее образование, ответили НЕТ на вопрос анкеты; а) сколько мужчин моложе 25 лет, имеющих начальное образование, ответили ДА на вопрос анкеты;

208. производит вывод всех анкет и ответов на вопросы.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 16

Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

209. номер УДК;

210. фамилию и инициалы автора;

211. название;

212. год издания;

213. количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

214. начальное формирование данных о всех книгах в библиотеке в виде списка;

215. добавление данных о книгах, вновь поступающих в библиотеку;

216. удаление данных о списываемых книгах;

217. по запросу выдаются сведения о наличии книг в библиотеке, упорядоченные по годам издания.

Вариант 17

На междугородной телефонной станции картотека абонентов, содержащая сведения о телефонах и их владельцах, организована как линейный список.

Составить программу, которая:

218. обеспечивает начальное формирование картотеки в виде линейного списка;

219. производит вывод всей картотеки;

220. вводит номер телефона и время разговора;

221. выводит извещение на оплату телефонного разговора.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 18

На междугородной телефонной станции картотека абонентов, содержащая сведения о телефонах и их владельцах, организована как двоичное дерево. Составить программу, которая:

222. обеспечивает начальное формирование картотеки в виде двоичного дерева;производит вывод всей картотеки;
223. вводит номер телефона и время разговора;
224. выводит извещение на оплату телефонного разговора.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 19

Автоматизированная информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования.Для каждого поезда указывается:

225. номер поезда;
226. станция назначения;
227. время отправления.

Данные в информационной системе организованы в виде линейного списка. Составить программу, которая:

228. обеспечивает первоначальный ввод данных в информационную систему и формирование линейного списка;
229. производит вывод всего списка;
230. вводит номер поезда и выводит все данные об этом поезде;
231. вводит название станции назначения и выводит данные обо всех поездах, следующих до этой станции.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 20

Автоматизированная информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования.
Для каждого поезда указывается:

232. номер поезда;
233. станция назначения;
234. время отправления.

Данные в информационной системе организованы в виде двоичного дерева. Составить программу, которая:

235. обеспечивает первоначальный ввод данных в информационную систему и формирование двоичного дерева;
236. производит вывод всего дерева;
237. вводит номер поезда и выводит все данные об этом поезде;
238. вводит название станции назначения и выводит данные о всех поездах, следующих до этой станции.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

IV. ФОНД ТЕСТОВЫХ И КОНТРОЛЬНЫХ ЗАДАНИЙ ДЛЯ ОЦЕНКИ КАЧЕСТВА ЗНАНИЙ ПО ДИСЦИПЛИНЕ

Инструкция: все задания имеют одну и ту же форму – с выбором одного ответа из четырех предложенных. Заданиями с 4-го по 26-е проверяется знание языка программирования Object Pascal среды программирования Borland Delphi. Текст на русском языке, заключенный в символы < и >, является смысловым описанием некоторой конструкции языка программирования. Время тестирования – 60 минут.

1. В конфигурацию компьютерной системы не входит…

1) кошка 2) системный блок 3) монитор 4) мышь

2. Устройством управления компьютером манипуляторного типа является…

1) монитор 2) сканер 3) мышь 4) системный блок

3. Внутри системного блока не находится…

1) принтер 2) материнская плата 3) видеокарта 4) процессор

4. Комментарием является последовательность символов, стоящих правее пары символов…

1) => 2) >> 3) // 4) ||

5. Буквы русского алфавита могут использоваться в…

1) идентификаторах
2) символьных выражениях
3) зарезервированных словах
4) директивах компилятора

6. Идентификатором может быть последовательность символов…

1) чай 2) –d– 3) cAT 4) 7id

7. Раздел описания (объявления, декларации) чего-либо оканчивается символом…

1) ;
2) ,
3) :
4) _

8. Функция pred возвращает…

1) символ, код которого равен значению аргумента функции
2) код символьного значения аргумента функции
3) значение порядкового типа, имеющее порядковый номер на единицу больший, чем порядковый номер значения аргумента функции
4) значение порядкового типа, имеющее порядковый номер на единицу меньший, чем порядковый номер значения аргумента функции

9. Раздел описания (объявления, декларации) типов начинается словом…

1) var 2) type

3) const
4) label

10. Ввести данные с клавиатуры позволяет процедура…

1) get
2) write
3) read
4) put

11. Символьным является выражение…

1) ord(false)
2) sin(5.6)<7
3) 'dog'+' cat'
4) chr(33)

12. Логическим является тип…

1) real
2) boolean
3) string
4) ansichar

13. Результатом операции not not not (3>7) является значение…

1) false
2) 7
3) 'false'
4) true

14. К порядковому типу не относится тип…

1) integer
2) char
3) real
4) boolean

15. Составной оператор ограничен…

1) символами { и }
2) символами : и .
3) символами < и >
4) словами begin и end

16. Пустой оператор содержится во фрагменте…

1) begin y:=5; end.
2) y:=0;
3) writeln(' ');
4) y:=5<2;

17. В операторе выбора значение выражения, стоящего после слова case, не может относится к типу…

1) real
2) boolean
3) integer
4) char

18. Если в операторе цикла с параметром (оператор for) используется слово to, то при выполнении очередной итерации порядковый номер значения счетчика (параметра цикла)…

1) уменьшается на единицу
2) увеличивается на единицу
3) уменьшается на заданное значение
4) увеличивается на заданное значение

19. Дан оператор цикла с постусловием:
repeat <операторы> until <логическое выражение>;
Начиная со второй итерации, тело цикла выполняется оператором цикла, если значение логического выражения…

1) true
2) false
3) 'false'
4) 'true'

20. Нет категории файла…

1) графический файл
2) текстовый файл
3) типизированный файл
4) нетипизированный файл

21. Группу формальных параметров – результатов предваряют словом…

1) const
2) var
3) out
4) in

22. В описании метода указать на объект, вызвавший этот метод можно словом…

1) inherited
2) self
3) me
4) object

23. Какие методы не могут быть перекрыты методами классов-наследников…

1) виртуальные
2) абстрактные
3) статические
4) динамические

24. При описании типа-записи используют слово…

1) array
2) set
3) record
4) class

25. Тип индекса у массива не может относится к…

1) типу вещественных чисел
2) перечислимому типу
3) интервальному типу
4) логическому типу

26. Операцию пересечения множеств обозначают символом…

1) –
2) +
3) *
4) ~

27. В программе Mathcad перед набором индексов элемента массива с клавиатуры вводится символ…

1) (
2) {
3) <
4) [

28. В программе Mathcad значение нижней границы индексации элементов массива определяется переменной…

1) origin
2) index
3) matrixindex
4) lim

29. В программе Mathcad для реализации нелинейной регрессии общего вида используют функцию(ии)…

1) intercept и slope
2) linfit
3) regress и interp
4) genfit

V. КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА

Лабораторные занятия проводят:

2, 5 семестр – доцент кафедры математического анализа и моделирования Труфанов В.А.;

3, 4 семестр – старший преподаватель кафедры математического анализа и моделирования, к.т.н. Рыженко А.В;

6 семестр – ассистент кафедры математического анализа и моделирования Кузьменко В.А.

СОДЕРЖАНИЕ