Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Амурский государственный университет»

Кафедра информационных и управляющих систем

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

«АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ»

Основной образовательной программы по специальности

230102.65 – Автоматизированные системы обработки информации и управления

Благовещенск 2012

УМКД разработан канд. техн. наук, доцентом Т.А. Галаган

Рассмотрен и рекомендован на заседании кафедры

Протокол заседания кафедры от «__» _____201_ г. № _____

Зав. кафедрой _____ / А.В. Бушманов /
(подпись) (И.О. Фамилия)

УТВЕРЖДЕН:

Протокол заседания УМСС 230102.65 – Автоматизированные системы обработки информации и управления

от «____» _____ 201__ г. № _____

Председатель УМСС _____ /_____ /
(подпись) (И.О. Фамилия)

РАБОЧАЯ ПРОГРАММА

Курс 1 семестр 1, 2

Лекции 72 (час.) Экзамен 1, 2 семестр

Практические (семинарские) занятия 36 (час.) Зачет -

Лабораторные занятия 54 (час.)

Самостоятельная работа 98 (час.)

Всего часов 260 час.

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

1.1.Цель курса – обучение студентов программированию с использованием языка высокого уровня С++; знание основ и положение процедурного, модульного и объектно-ориентированного программирования; изучение основ алгоритмизации и проектирования программ и эффективных алгоритмов; привитие практических навыков реализации программ на современных ЭВМ.

1.2. По завершению обучения дисциплине студент должен:
уметь составлять и реализовывать алгоритмы решения задач; иметь практические навыки работы со средами Borland C++ и Borland C++ Builder; владеть основами программирования на языках С++; знать и владеть основными этапами решения задач на ЭВМ; выявлять и устранять ошибки в программах; владеть основами доказательства правильности программ.

1.3. Преподавание курса «Алгоритмические языки и программирование связано с изучением курса государственного образовательного стандарта «Высшая математика» и является основой для изучения дальнейших дисциплин, использующих ЭВМ и программирование.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. ФЕДЕРАЛЬНЫЙ КОМПОНЕНТ

Программа курса «Алгоритмические языки и программирование» составлена в соответствие с требованиями государственного образовательного стандарта специализации – Интегрированные системы автоматизированного управления, специализации 230201, блок общеобразовательных дисциплин ОПД.Ф.05.

2.2. ЛЕКЦИИ (72 часа)

2.2.1.Введение в программирование: этапы создание программы, устройство компьютера, языки программирования.(2 часа)

2.2.2. Общая характеристика языка С++. Основные конструкции языка: алфавит, идентификаторы, ключевые слова. Простые типы данных. Переменные и константы. Основные операции, математические функции языка. Арифметические и логические выражения. (4 часа)

2.2.3. Структура программы на языке С++. Простые операторы языка: оператор присваивания, оператор безусловного перехода, операторы ввода – вывода. Условные конструкции языка. (6 часов).

2.2.4. Циклические конструкции языка: с предусловием, с постусловием, с заданным числом повторений. Операторы передачи управления.(4 часа)

2.2.5. Работа с одномерными массивами и со строковыми типами данных. Задачи поиска минимального и максимального элементов массива; нахождение суммы, произведения, количества определенных элементов массива. Методы сортировки элементов массива. (4 часа)

2.2.6. Двумерные массивы. Обработка двумерных массивов случайным образом, по строкам, по столбцам. Счетчик случайных чисел. (4 часа)

2.2.7. Объявление и определение функций. Фактические и формальные параметры. Понятие прототипа функции. Вызов функции. Передача значений с использованием оператора return. Понятие рекурсии. (6 часов)

2.2.8. Директивы препроцессора #include, #define. Объявление макросов и встроенных функций. (2 часа)

2.2.9. Спецификации классов памяти. Объявление переменных на внешнем и внутреннем уровнях. Время жизни и область действия переменных. (2 часа)

2.2.10. Технология создания программы. (2 часа)

2.2.11. Указатели и основные операции над ними. Объявление и инициализация массивов. Связь между указателями и массивами. (4 часа)

2.2.12. Понятие ссылки. Использование ссылок для передачи значений из функции. Операция приведения типа. Динамическое распределение памяти. (2 часа)

2.2.13. Типы данных, определяемые пользователем (перечисляемый тип, структуры и объединения). Переименование типов с помощью typedef. (6 часов)

2.2.14. Основы объектно-ориентированного программирования. Инкапсуляция, наследование, полиморфизм. Определение класса в C++. Элементы-данные и элементы-функции класса. Спецификации управления доступом. Оператор видимости. (6 часов)

2.2.15. Конструкторы и деструкторы. (2 часа)

2.2.16. Дружественные классы. Дружественные функции. Правила относительно друзей. (2 часа)

2.2.17. Простое и множественное наследование. (2 часа)

2.2. 18. Перегрузка функций. Виртуальные функции. (4 часа)

2.2.19. Текстовые и бинарные файлы. Функции открытия и закрытия файлов. Чтение и запись в файл. Функция произвольного доступа. (4 часа)

2.2.20. Эволюция систем программирования. Этапы технологии создания программы (4 часа)

2.3.ПРАКТИЧЕСКИЕ ЗАНЯТИЯ (36 часов)

2.3.1. Понятие алгоритма, способы записи алгоритмов. Алгоритмы линейной структуры. (2 часа)

2.3.2. Алгоритмы разветвляющейся структуры (2 часа)

2.3.3. Алгоритмы циклической структуры (4 часа)

2.3.4. Структура программы языка C++. Арифметические и логические выражения языка. (2 часа)

2.3.5. Контрольная работа по пройденным темам (2 часа)

2.3.6. Программы на основе разветвляющегося алгоритма (if ... else, switch). (4 часа)

2.3.7. Обработка одномерных массивов. (6 часов)

2.3.8. Обработка двумерных массивов (6 часов)

2.3.9. Обработка строк (2 часа)

2.3.10. Контрольная работа по пройденным темам (2 часа)

2.3.11. Создание пользовательских функций. (4 часа)

2.4. ЛАБОРАТОРНЫЕ РАБОТЫ (54 часа)

2.4.1. Изучение среды программирования Borland C++, обработка простейшей линей-

ной программы. (2 часа)

2.4.2. Программирование разветвляющейся структуры. (2 часа)

2.4.3. Разветвляющиеся программы. Оператор switch. (2 часа)

2.4.4. Программирование разветвляющей и циклической структуры (2 часа)

2.4.5. Вычисление суммы ряда с заданной точностью. (2 часа)

2.4.6. Обработка одномерных массивов. (4 часа)

2.4.7. Символьные массивы. (2 часа)

2.4.8. Двумерные массивы. (4 часа)

2.4.9. Создание пользовательских функций. (2 часа)

2.4.10. Указатели и массивы. (4 часа)

2.4.11. Использование ссылок для передачи значений из функции. (2 часа)

2.4.12. Структуры. Динамическое распределение памяти (6 часов)

2.4.13. Текстовые файлы (2 часа)

2.4.14. Бинарные файлы (2 часа)

2.4.15. Объектно-ориентированное программирование. Классы в С++. (4 часа)

2.4.16. Наследование (4 часа)

2.4.17. Полиморфизм :виртуальные функции, перегрузка операций (2 часа)

2.4.18. Графические возможности С++ (4 часа)

2.5. КУРСОВАЯ РАБОТА –

2.6. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (98 часов)

2.6.1. Работа со строковыми типами данных. (4 часа)

2.6.2. Указатели и динамическая память (10 часов)

2.6.3. Объектно – ориентированное программирование (10 часов)

2.6.4. Графика в С++ (20 часов)

2.6.5. Изучение стандартных модулей. (12 часов)

2.6.6. Способы конструирования программ (6 часов)

2.6.7. Выявление и устранение ошибок (10 часов)

2.6.8. Списки:основные виды и способы реализации (10 часов)

2.6.9. Основы доказательства правильности (10 часов)

2.6.10. Директивы условной компиляции (6 часов)

2.7. ПЕРЕЧЕНЬ И ТЕМЫ ПРОМЕЖУТОЧНЫХ ФОРМ КОНТРОЛЯ ЗНАНИЙ

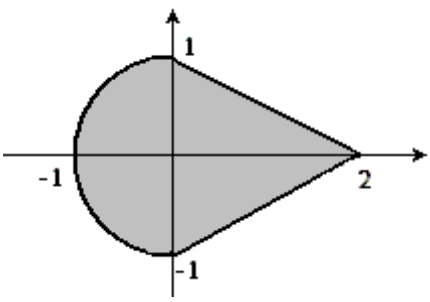
2.7.1. Вариант контрольной работы №1

1. Составить программу вычисления

$$z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$$

$$z_2 = \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$$

2. Составить блок-схему и программу попадания в «мишень», представленную на рисунке.



3. Составить блок-схему и программу нахождения значения

$$\sum_{x=4}^{24} x^3 /(x^4 - 5)$$

2.7.2. Вариант самостоятельной работы

1. В одномерном массиве B(15) определить сумму и количество отрицательных элементов, расположенных после первого элемента массива, равного нулю. Составить блок-схему алгоритма и написать программу на языке C++.

2. Если значение минимального элемента массива K(60) отрицательно, поменять местами минимальный и последний элементы массива, иначе уменьшить значение минимального элемента массива в десять раз. Составить блок-схему алгоритма.

3. Напишите программу, задающую значения элементов одномерного массива, содержащего 20 элементов, по правилу: первый элемент ввести с клавиатуры, а каждый следующий - должен быть в два раза больше предыдущего. Вывести на экран полученные значения.

Вариант контрольной работы №2

1. С использованием счетчика случайных чисел задать элементы матрицы B(8, 8) в диапазоне от -15 до 15. Определить максимум из элементов каждого второго столбца полученной матрицы. В каждой строке определить сумму положительных элементов. Поменять местами элементы первой и последней строки. Вывести на экран в общепринятом виде первоначальную и итоговую матрицы. Составить блок- схему и написать программу

2. Объявлен массив

float matrix[4][3] = { {1, 6.5}, {-1.3, 3.6, 2.8}, {2.6, 3.2, -1}, {1.1, 3, 2}};

Какое значение имеет элемент matrix[2][1]

Вариант итогового тестирования по проверке знаний языка C++

1. В приведенном фрагменте программы возможно содержится ошибка. Выберите среди предложенных вариантов правильный ответ

#define PI = 3.1415

a) строка написана правильно **b)** следует писать #define PI = 3.1415;
c) следует писать #define "PI = 3.1415"; **d)** следует писать #define PI 3.1415

2. Сколько раз распечатается слово Hello в процессе работы приведенного ниже фрагмента

```
unsigned i;
i = 0;
while (i < 10)
{     printf ("Hello");
i--; }
```

a) ни одного раза **b)** 1 раз **c)** 9 раз **d)** 10 раз **e)** 11 раз

3. Какой файл заголовков нужно включить с помощью директивы #include, чтобы использовать в программе следующий оператор

```
intl=abs( yl-8);
```

a) conio.h **b)**stdio.h **c)** iostream.h **d)**string.h **e)** math.h

4. Функция с именем prim возвращает указатель на символ и имеет два аргумента целого типа, какой из вариантов прототипа функции является правильным

a) prim (char* c, int x, int y); **b)** void prim (char*c, int x, int y);
c)*char prim (int x, int y); **d)** char* prim(int x; int y);

5. Инициализация нулем всех элементов массива типа float с именем plan, имеющего 30 строк и 10 столбцов выглядит как

a)
```
for (row=0; row<30; row++)
for (col=0;  col<10; col++)
plan[row][col]=0.0;
```

b)
```
for (row=0; row<10; row++)
for (col=0;  col<30; col++)
plan[row][col]=0.0;
```

c)
```
for (row=1; row<=30; row++)
for (col=1;  col<=10; col++)
plan[row][col]=0.0;
```

d)
```
for (row=1; row<=10; row++)
for (col=1;  col<=30; col++)
plan[row][col]=0.0;
```

6. Объявлен прототип функции int mean (int *a, int b); и переменные int x,y, z[5]; вызов данной функции будет вида:

a)mean(x,y); **b)**y=mean(z,x); **c)**y=mean(x,z); **d)**z=mean(x,y);

7. Если считать, что указатель занимает 2 байта, а символ 1 байт то в процессе работы фрагмента программы

```
char c, *pc;
printf("Размер &c = %d  ", sizeof (&c));
printf("Размер pc = %d   ", sizeof (pc));
printf("Размер *pc = %d  ", sizeof (*pc));
```

на экран выведется

a) размер &c = 2 размер pc = 2 размер *pc = 2
b) размер &c = 1 размер pc = 2 размер *pc = 1
c) размер &c = 2 размер pc = 2 размер *pc = 1
d) размер &c = 2 размер pc = 1 размер *pc = 2

8. После выполнения фрагмента программы на экран выведется

```
#define sqr(x) x*x
void main()
{
     int x = 1;
         printf("sqr x = %d", sqr(x + 1));
}
```

a) sqr x = 1 **b)** sqr x = 2 **c)** sqr x = 3 **d)** sqr x = 4 **e)** sqr x = 5 **f)** sqr x = 6

9. В приведенном фрагменте программы возможно содержится ошибка. Выберите среди приведенных вариантов правильный ответ

```
int arr[10] = {0, 1, 2, 3, 4, 5 };
int i, s = 0;
for (i = 0; i < 5; ++i)
    s += *(arr+i);
```

a) всё правильно;

b) в операторе присваивания допущена ошибка. Следует писать s += arr[i];
c) в операторе присваивания допущена ошибка. Следует писать s += *arr++;
d) в операторе присваивания допущена ошибка. Следует писать s += *arr[i];

10. Если считать, что int занимает 2 байта, то в процессе работы фрагмента программы на экран выведется

```
int arr[2][2] = {1, 2, 4, 8};
int *p;
p = arr[0][1];
printf("%d %d", *p, *(p + 2));
```

a) 1 3 **b)** 0 0 **c)** 1 4 **d)** 4 0 **e)** 2 8 **f)** 65510 65514

11. Если считать, что указатель занимает 2 байта, а длинное целое 4 байта, то в процессе работы фрагмента программы на экран выведется

```
long (*arr)[2];
printf("%d %d %d", sizeof(arr), sizeof(*arr), sizeof((*arr)[0]));
```

a) 4 2 4 **b)** 8 4 2 **c)** 4 4 4 **d)** 2 4 4 **e)** 2 8 4

12. В приведенном фрагменте программы возможно содержится ошибка. Выберите среди приведенных ответов правильный по вашему мнению

```
cout >> "Hello"; cout >> endl;
```

a) строка написана правильно
b) в строке ошибка. Следует писать cout << "Hello"; cout << endl;
c) в строке ошибка. Следует писать cout << "Hello"; cout << endline;
d) строке ошибка. Следует писать cout >> "Hello"; cout >> endline;

13. После выполнения фрагмента программы на экран выведется

```
class A {
public:
  A ( ) { printf ("construct A\n");}
 ~A ( ) { printf ("destruct A\n");}
};
class B : A {
public:
  B ( ) { printf ("construct B\n");}
 ~B ( ) { printf ("destruct B\n");}
};
B b;
```

a) ничего не выведется
b) фрагмент не будет скомпилирован
c)
```
construct A
destruct A
construct B
destruct B
```
d)
```
construct A
construct B
destruct B
destruct A
```
e)
```
construct B
destruct B
```
f)
```
construct B
construct A
destruct A
destruct B
```

14. Если считать, что int занимает в памяти 2 байта, то для того, чтобы записать в открытый для записи файл f значение целой переменной i можно воспользоваться:

a) fwrite (i, sizeof (i), 1, f);
b) fwrite (&i, sizeof (i), 1, f);
c) fwrite (i, sizeof (i), 2, f);
d) fwrite (&i, sizeof (i), 2, f);

15. Поставьте в соответствие логическому выражению в левой колонке, проверяющее выполнение того же самого условия

x!=>y || y= =z

a) !(x!=y) && y= =z **b)** !(x<=y || y<z) **c)** (y<z || y= =z) || x= =y
d) !(x>=y) && !(y>=z) **e)** !(x= =y && y !=z)

2.8. ВОПРОСЫ К ЭКЗАМЕНУ

2.8.1. Экзаменационные вопросы (1 семестр)

1. Этапы создания программы
2. Этапы компиляции и выполнения программы
3. Понятие алгоритма. Способы записи алгоритмов
4. Линейные алгоритмы
5. Разветвляющие алгоритмы
6. Циклические алгоритмы
7. Состав языка С++
8. Структура программы языка С++
9. Переменные, идентификаторы
10. Типы данных языка С++
11. Описание констант и переменных. Инициализация переменных
12. Основные операции языка С++
13. Директивы препроцессора #include, #define
14. Библиотека математических функций
15. Область действия и время жизни переменных
16. Ввод-вывод с использованием библиотеки iostream (cin>>, cout<<)
17. Условный оператор if
18. Множественный выбор: оператор switch
19. Инструкции перехода
20. Оператор цикла с предусловием
21. Оператор цикла с постусловием
22. Оператор цикла с заданным числом повторений
23. Одномерные массивы
24. Методы сортировок
25. Двумерные массивы
26. Обработка двумерных массивов по строкам
27. Обработка двумерных массивов по столбцам
28. Функции библиотеки conio.h
29. Функции ввода-вывода (printf( ), scanf( ))
30. Строки
31. Определение функций
32. Понятие прототипа функции
33. Область действия и время жизни переменных
34. Технология создания программы

2.8.2. Экзаменационные вопросы (2 семестр)

1. Указатели (объявление, операции, связь с массивами)
2. Ссылки. Передача аргументов функции по ссылке
3. Строки. Функции работы со строками

4. Объявления typedef
5. Структуры
6. Объединения
7. Перечисляемый тип
8. Основные функции работы со строками
9. Текстовые и бинарные файлы
10. Функции работы с файлами (fopen(), fclose(), fwrite(), fread(),fseek())
11. Основные принципы объектно-ориентированного программирования
12. Понятие класса
13. Спецификаторы доступа
14. Конструкторы
15. Деструкторы
16. Простое и множественное наследование
17. Дружественные классы
18. Виртуальные функции
19. Перегрузка операций и функций
20. Динамическое распределение памяти
21. Графические возможности C++
22. Директивы условной компиляции
23. Критерии качества программы
24. Этапы проектирования программы

КРИТЕРИИ ОЦЕНОК ЗНАНИЙ СТУДЕНТОВ

Отлично

Студент дает полные ответы на теоретические вопросы билета, показывая глубокое знание учебного материала, свободное владение основными понятиями и терминологией; ответ на дополнительный вопрос; при решении задачи демонстрирует навыки работы на компьютере в среде C++, находит правильный ответ, оптимальный алгоритм, правильно выделяет подпрограммы.

Хорошо

Студент дает ответы на теоретические вопросы билета, показывая прочное знание учебного материала, владение основными понятиями и терминологией; ответ на дополнительный вопрос; при решении задачи демонстрирует навыки работы на компьютере в среде C++, находит правильный ответ.

Удовлетворительно

Студент дает неполные ответы на теоретические вопросы билета, показывая поверхностное знание учебного материала, владение основными понятиями и терминологией; при неверном ответе на билет ответы на наводящие вопрос; частичное решение задачи.

Неудовлетворительно

Студент не дает полные ответы на теоретические вопросы билета, показывая лишь фрагментарное знание учебного материала, незнание основных понятий и терминологии; наводящие вопросы остаются без ответа; решение задачи не найдено.

3. УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ:

3.1. ЛИТЕРАТУРА

ОСНОВНАЯ

1. Павловская Т.А. C/C++. Программирование на языке высокого уровня. СПб.: Питер, 2004. 461 с. (Допущено МО РФ)

2. Павловская Т.А., Щупак Ю.А. C/C++. Структурное программирование. Практикум. СПб.: Питер, 2004. 239 с.

3. Павловская Т.А., Щупак Ю.А. С/С++. Объектно-ориентированное программирование. Практикум. СПб.: Питер, 2004. 265 с.

4. Подбельский В.В. Язык С++: учебное пособие: Рек. Мин. Обр. РФ – СПб: Питер, 2001, 2002, 2003

5. Франка П. Язык С++: учебное пособие – СПб: Питер, 2004.

6. С. Уэллин. Как надо программировать на С++: 111 нерабочих и 3 рабочих программ, или почему 2+2. СПб.: Питер, 2004.

ДОПОЛНИТЕЛЬНАЯ

1. Бруно Бабэ. Просто и ясно о Borland С++. Версии 4.0 и 4.5. М: Бином, 1995, 395с
2. Козелл Е.И. и др. От Си к С++. М: Финансы и статистика, 1993, 270с.
3. В.А. Скляров. Язык С++ и объектно-ориентированное программирование. Минск: Вышэйшая школа, 1997, 478 с.
4. Бочков С.О., Субботин Д.М. Язык программирования Си для персонального компьютера. М: СП «Диалог» «Радио и связь», 1990, 383 с.
5. Майкл Хаймен, Боб Арнсон. Borland С++5 для «чайников». Киев: Диалектика, 1997, 319 с.
6. Пахомов Б.И. С/С++ и Borland С++ Builder для начинающих. СПб.: БХВ-Петербург, 2005. 640 с.

3.2. МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

1. Галаган Т.А. Алгоритмические языки и программирование. Язык С++. Курс лекций. Благовещенск: Изд-во АмГУ, 2007. 147 с. (Рекомендовано ДВ РУМЦ)
2. Галаган Т.А., Соловцова Л.А. Язык программирование С++ в примерах и задачах. Учебно-методическое пособие. Благовещенск: Изд-во АмГУ, 2002. 40 с.
3. Галаган Т.А., Соловцова Л.А. Язык программирование С++ в примерах и задачах. Практикум. Благовещенск: Изд-во АмГУ, 2005. 104 с.

ТЕХНИЧЕСКИЕ СРЕДСТВА ОБЕСПЕЧЕНИЯ ДИСЦИПЛИНЫ

1. Кафедра обладает необходимым количеством современных ЭВМ для проведения лабораторных работ
2. Программное обеспечение: среды программирования Borland С++ 3.1 , Borland С++ Builder.

4. УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

4.1. 1 семестр

Номер недели	Вопросы, изучаемые на лекции	Занятия (номера)		Используемые нагляд. и метод. пособия	Самостоятельная работа студентов		Форма контроля
		Практич (семин.)	Лаборат.		Содержание	часы	
1	2.2.1	2.3.1					
2	2.2.2	2.3.2					
3	2.2.2	2.3.3					
4	2.2.3	2.3.3			2.6.1	2	
5	2.2.3	2.3.4			2.6.1	2	
6	2.2.3	2.3.5					2.7.1.
7	2.2.4	2.3.6			2.6.4	2	

8	2.2.4	2.3.6			2.6.4	2	
9	2.2.5	2.3.7			2.6.4	2	
10	2.2.5	2.3.7			2.6.4	2	
11	2.2.6	2.3.7			2.6.4	2	2.7.2
12	2.2.6	2.3.8			2.6.4	2	
13	2.2.7	2.3.8			2.6.4	2	
14	2.2.7	2.3.8		3.2.1	2.6.4	2	
15	2.2.7	2.3.9		3.2.1	2.6.4	2	
16	2.2.8	2.3.10		3.2.1	2.6.4	2	
17	2.2.9	2.3.11		3.2.1	2.6.4	2	
18	2.2.10	2.3.11		3.2.1	2.6.4	2	2.7.3

4.2. 2 семестр

Номер недели	Вопросы, изучаемые на лекции	Занятия (номера)		Используемые нагляд. и метод. пособия	Самостоятельная работа студентов		Форма контроля
		Практич (семин.)	Лаборат.		Содержание	часы	
1	2.2.11		2.4.1	3.2.1, к	2.6.2	4	отчет
2	2.2.11		2.4.2 – 2.4.3	3.2.1, к	2.6.2	4	отчет
3	2.2.12		2.4.4	3.2.1, к	2.6.2, 2.6.3	4	отчет
4	2.2.13		2.4.5 – 2.4.6	3.2.1, к	2.6.2, 2.6.3	4	отчет
5	2.2.13		2.4.6	3.2.1, к	2.6.2	4	отчет
6	2.2.13		2.4.7 – 2.4.8	3.2.1, к	2.6.1 - 2.6.2, 2.6.4	4	отчет
7	2.2.14		2.4.8	3.2.2, к	2.6.1 - 2.6.2, 2.6.4	4	отчет
8	2.2.14		2.4.10	3.2.1, к	2.6.1 - 2.6.2, 2.6.4	4	отчет,
9	2.2.14		2.4.11	3.2.2, к	2.6.3 - 2.6.9	4	отчет
10	2.2.15		2.4.12	3.2.2, к	2.6.3 - 2.6.8	4	отчет
11	2.2.16		2.4.12 – 2.4.13	3.2.2, к	2.6.3 - 2.6.7	4	отчет,
12	2.2.17		2.4.13	3.2.2, к	2.6.3 - 2.6.9	4	отчет
13	2.2.18		2.4.13 – 2.4.14	3.2.2, к	2.6.3 - 2.6.8	4	отчет
14	2.2.18		2.4.15	3.2.1, к	2.6.3 - 2.6.8	4	отчет,
15	2.2.19		2.4.15	3.2.1, к	2.6.9	4	отчет
16	2.2.19		2.4.16	3.2.2, к	2.6.10	4	отчет,
17	2.2.20		2.4.17	3.2.2, к	2.6.10	4	отчет
18	2.2.20		2.4.18	3.2.2, к	2.6.10	4	отчет,

Обозначения
К – карточки с индивидуальными заданиями

УТВЕРЖДАЮ

Зав. кафедрой ИУС

_____ **А.В. Бушманов**

«___» _____ 20 ___ г.

Список литературы к рабочей программе дисциплины
Алгоритмические языки и программирование
Специальности 230102.65 - Автоматизированные системы обработки информации и управления
По состоянию на « 01» 09 2012 г.

ОСНОВНАЯ ЛИТЕРАТУРА

1 Павловская, Т.А. С/С++. Программирование на языке высокого уровня (Допущено МинОбр РФ) – СПб.: Питер, 2009, 2010. – 461 с.

2 Лаптев, В.В. С++ Объектно-ориентированное программирование: учеб. пособие / В.В. Лаптев – СПб.: Питер, 2008. – 458 с.

ДОПОЛНИТЕЛЬНАЯ

1 Павловская, Т.А. С/С++ Структурное программирование. Практикум. / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. – 239 с.

2 Павловская, Т.А. С++. Объектно-ориентированное программирование. Практикум (Допущено МинОбр РФ) / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. 265 с.

3 Лаптев, В.В. С++ Объектно-ориентированное программирование: задачи и упражнения (Допущено Мин. обр. РФ) / В.В. Лаптев, А.В. Морозов, А.В. Бокова – СПб.: Питер, 2007. – 281 с.

4 Шамис, В.А. Borland C++ Builder 6 / В.А. Шамис – СПб.: Питер, 2005 – 798 с.

МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

4.Галаган, Т.А. Алгоритмические языки и программирование. Язык С++. Курс лекций (Рек. ДВРУМЦ) / Т.А. Галаган – Благовещенск: изд-во АмГУ, 2007. – 147 с.

5.Галаган, Т.А. Язык программирование С++ в примерах и задачах. Практикум / Т.А. Галаган, Л.А. Соловцова – Благовещенск: Изд-во АмГУ, 2005. – 104 с.

ИНТЕРНЕТ-РЕСУРСЫ

	Наименование ресурса	Характеристика
1	http://www.biblioclub.ru	Электронная библиотечная система «Университетская система -online» специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а также содержит материалы по точным и естественным наукам.
2	http://www.window.edu.ru	Единое окно доступа к образовательным ресурсам/ каталог/ профессиональное образование

Преподаватель _____ Т.А. Галаган

СОГЛАСОВАНО
Директор научной библиотеки _____Л.А. Проказина

I. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Практические занятия проводятся по группам. Преподаватель кратко излагает теоретические аспекты рассматриваемой темы, приводя решения задач. Затем задания решаются студентами самостоятельно с проведением демонстрации правильных решений на доске.

На каждом занятие студенты получают домашнее задание, обязательная проверка которого проводится на следующем занятие.

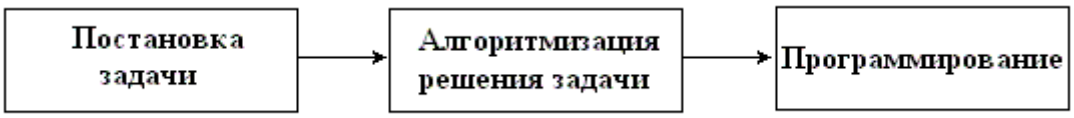
В течение первого семестра проводится три контрольных работы.

АЛГОРИТМЫ И АЛГОРИТМИЗАЦИЯ

Возможности компьютера как технической основы обработки данных связаны с используемым программным обеспечением (программами).

Программа – упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Процесс создания программ можно представить как последовательность действий, представленных на рисунке.



Постановка задачи – это точная формулировка решения задачи на компьютере с описанием входной и выходной информации.

Алгоритм – система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных (входной информации) в желаемый результат (выходную информацию) за конечное число шагов.

Таким образом, алгоритмом называют последовательность команд (инструкций) для достижения некоторой цели.

Алгоритм решения задачи имеет ряд обязательных свойств:

дискретность – разбиение процесса обработки информации на более простые этапы (шаги), выполнение которых компьютером или человеком не вызывает затруднений;

определенность алгоритма – однозначность выполнения каждого отдельного шага преобразования информации;

выполнимость – конечность действий алгоритма решения задач, позволяющая получить желаемый результат при допустимых исходных данных за конечное число шагов;

массовость – пригодность алгоритма для решения определенного класса задач.

В алгоритме отражаются логика и способ формирования результатов решения с указанием необходимых расчетных формул, логических условий, соотношений для контроля достоверности выходных результатов. В алгоритме обязательно должны быть предусмотрены все ситуации, которые могут возникнуть в процессе решения комплекса задач.

Программирование – теоретическая и практическая деятельность с созданием программ, осуществляющая перевод алгоритма решения задачи на язык программирования.

Существует несколько способов представления алгоритмов: словесный, графический и др. Один из них – представление в виде блок-схем.

Блок-схема алгоритма – это графическое представление алгоритма в виде геометрических фигур и стрелок.

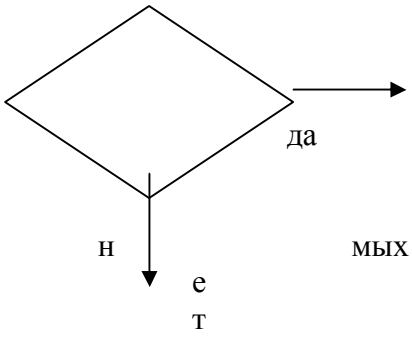
Основные блоки, используемые в блок-схемах:

Начало обработки данных.
(Начало алгоритма.)

Окончание алгоритма.

Блок процесс. Выполнение операции или группы операций, в результате которых изменяется значение, форма или представления данных.

Блок ввода-вывода. Ввод данных с клавиатуры, или отображение их на экране монитора.



Блок решение.
Выбор направления алгоритма в зависимости некоторых переменных условий, размещаемых внутри блока.

В теории программирования доказано, что программу для решения задачи любой сложности можно составить из трех структур, называемых следованием, ветвлением и циклом. Этот результат установлен Боймом и Якопини в 1966 году.

Следование, ветвление и цикл называют *базовыми конструкциями* структурного программирования.

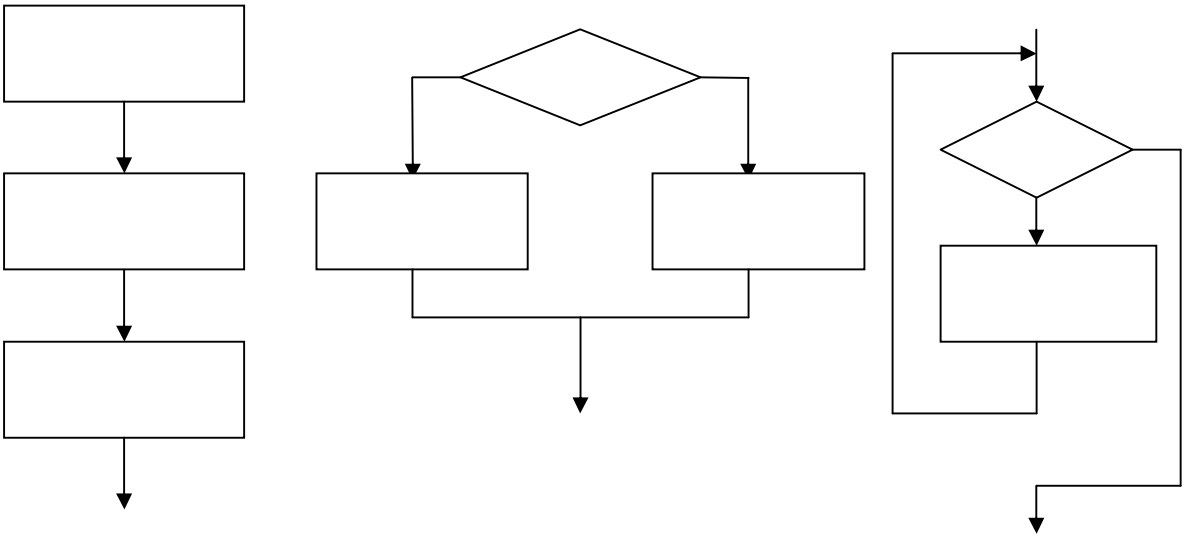
Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных).

Ветвление задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

Цикл задает многократное выполнение оператора.

Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход, поэтому конструкции могут вкладываться друг в друга произвольным образом, например, цикл может содержать следование двух ветвлений, каждое из которых содержит вложенные циклы.

Базовые конструкции используются с целью получения программы простой структуры.



Следование Ветвление Цикл

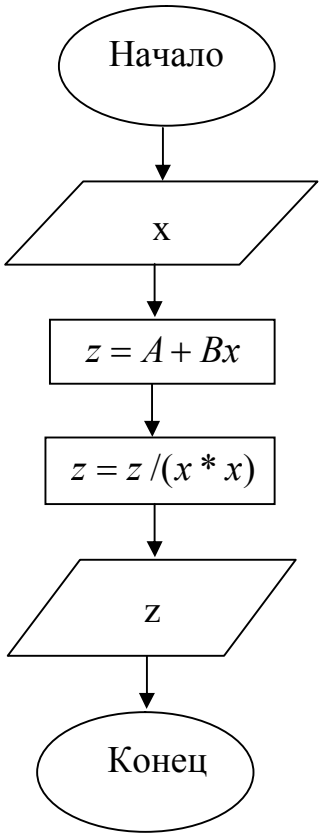
ЛИНЕЙНЫЙ АЛГОРИТМ

Линейным называется алгоритм, в котором все действия последовательно выполняются друг за другом.

Пример

Составить блок-схему алгоритма вычисления значения

$z = \dfrac{A + Bx}{x^2}$, где A, B – константы, x – переменная вводимая с клавиатуры.



В данной блок-схеме приведен линейный алгоритм. Причем два последовательных блока процесс можно было бы объединить в один блок.

Задание

Составить блок-схему алгоритма вычисления значений величины:

1. $y(x) = A / x + Bx, \quad где\ A, B - const.$
2. $z(x) = \dfrac{125x + 321}{x^2}.$
3. $h(a) = \dfrac{34}{a} - \dfrac{670}{a^3}.$
4. $r(b) = 294b - \sin b + \cos 2b.$
5. $v(t) = \dfrac{t^4}{2} + \dfrac{t^2}{4} + 9.$
6. $y(x) = \dfrac{25x}{(x+1)^3} + \dfrac{3}{x+2}$
7. $y(t) = e^{-t} + e^t / 1000 - 4$
8. $s(x) = \dfrac{\sin 2x}{\cos x + 1} - \dfrac{2\cos x}{5}$
9. $f(x) = \dfrac{x + 12x^3 - 5}{x^2 - x + 1}$
10. $h(a) = 3Ba - \dfrac{7}{B} + Ba^2, \quad B - const$

РАЗВЕТВЛЯЮЩИЙСЯ АЛГОРИТМ

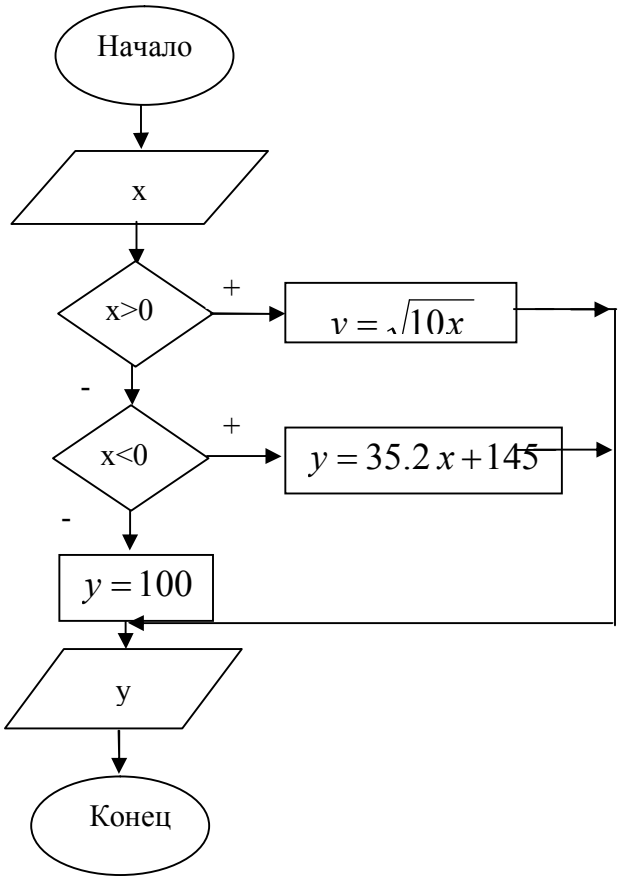
Алгоритм является разветвляющимся, если в нем предусмотрено несколько направлений. Направление выбирается логической проверкой условия, в результате которого возможны два ответа: истина или ложь.

Характерной особенностью разветвляющихся алгоритмов является то, что некоторая последовательность действий выполняется в зависимости от тех или иных условий, однако во всех случаях это происходит только один раз.

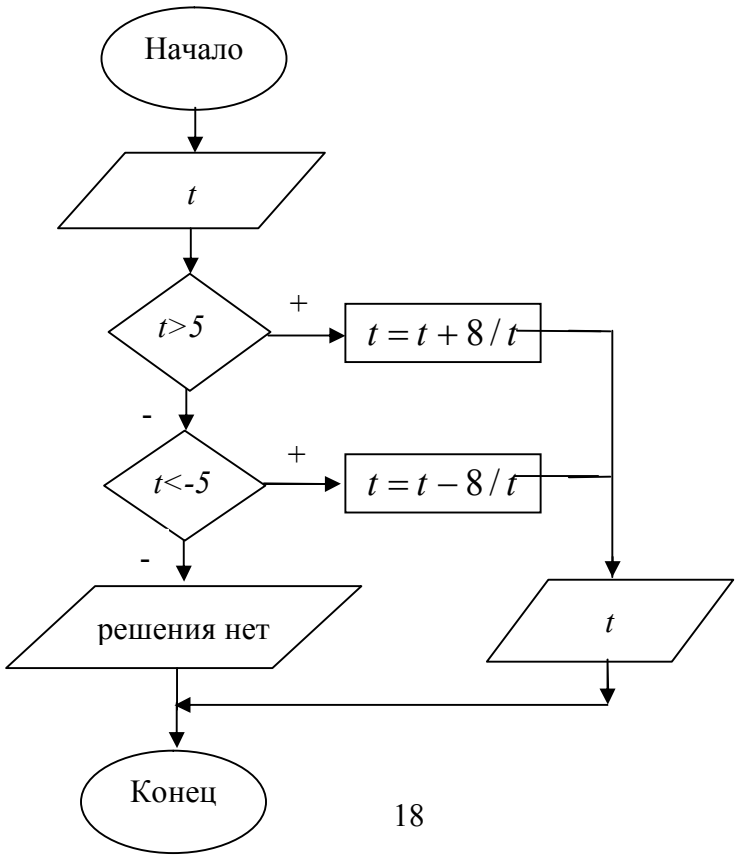
Пример. Составить блок-схему алгоритма для вычисления значения функции, заданной выражением:

$$y = \begin{cases} 35.2x + 145, & x < 0 \\ 100, & x = 0 \\ \sqrt{10x}, & x > 0 \end{cases}$$

В приведенном алгоритме не требуется проверки третьего условия, так как область значений функции y – вся числовая ось, и если аргумент x не удовлетворяет каким-либо двум из трех условий, то он обязательно удовлетворяет третьему.

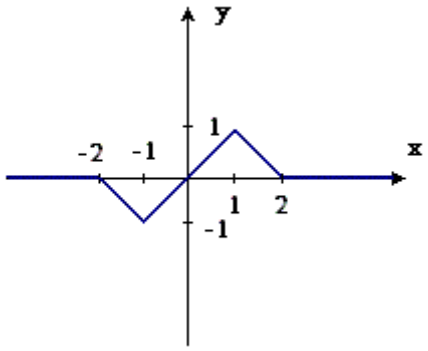


Пример. Составить блок-схему алгоритма для вычисления значения функции, заданной выражением: $\upsilon(t)=\begin{cases} t+8/t, & t>5 \\ t-8/t, & t<-5 \end{cases}$



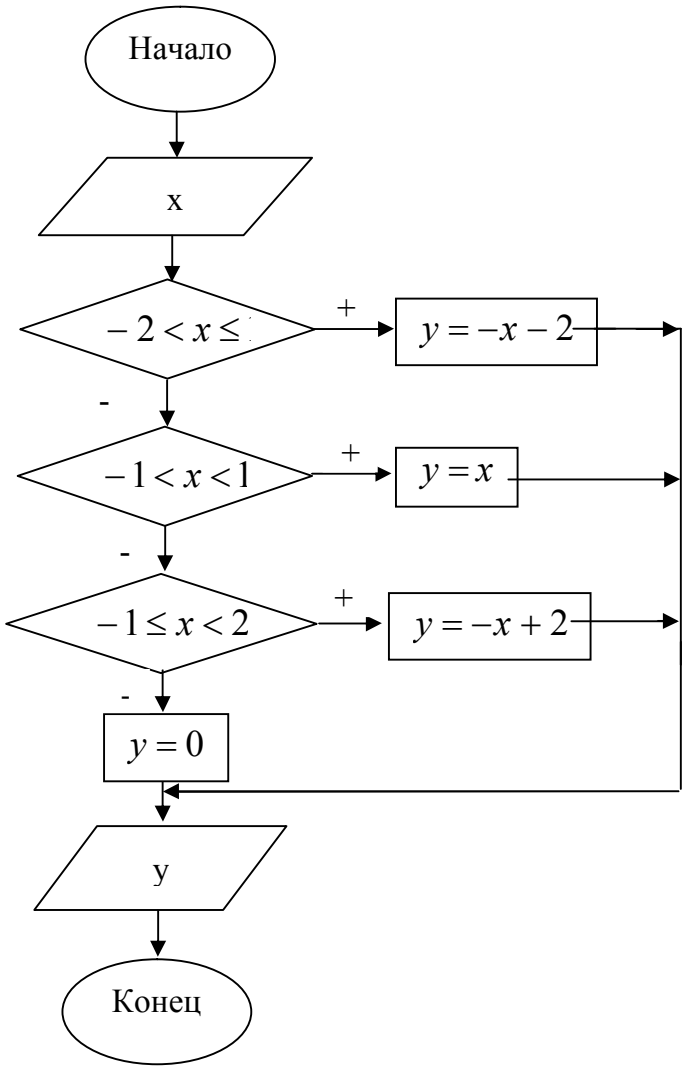
Не всегда формулировка задачи представлена в аналитической форме. Например, требуется составить алгоритм программы, которая по введенному значению аргумента вычисляет значение функции, заданной на следующем графике:

Сначала необходимо получить уравнения графика функции.



На промежутках $(-\infty;-2]\ [2;+\infty)$ значение функции равно нулю. На промежутке $(-2;-1]$ функция представлена прямой, проходящей через точки (-2, 0) и (-1, -1), следовательно, коэффициенты уравнения прямой ($y = kx + b$) можно определить решением системы уравнений: $\begin{cases} 0 = -2k + b \\ -1 = -k + b \end{cases}$ т.е., $k = -1,\quad b = -2$.

$$y = \begin{cases} 0, & x \le -2 \\ -x-2, & -2 < x \le -1 \\ x, & -1 < x < 1 \\ -x+2, & 1 \le x < 2 \\ 0, & x \ge 2 \end{cases}$$



Промежутки $(-\infty;-2]\ [2;+\infty)$ объединены в алгоритме в одну ветвь, поскольку здесь значение функции одинаково (равное нулю).

Задание

I. Составить блок-схему алгоритма вычисления заданной величины по введенному с клавиатуры аргументу.

1. $y(x)=\begin{cases} 9x^2-8A, & x>5 \\ 13x+A^2, & x<0 \\ x+A, & \textit{в остальных случаях} \end{cases}$ $A-const.$

2. $\upsilon(t)=\begin{cases} t^{-2}+t+3, & 25\le t<50 \\ t+25, & 10\le t<25 \\ 0, & 0\le t<10 \\ t^3-100, & -50\le t<0 \end{cases}$

3. $z(x)=\begin{cases} \sqrt{x^3-15}, & x\ge 3 \\ x+25, & x<3 \end{cases}$

4. $r(k)=\begin{cases} (k+25)/(k-25), & k>25 \\ k+25, & 0\le k<25 \\ (k-25)/(k+25) & -25<k<0 \end{cases}$

5. $u(j)=\begin{cases} 25j^{-4}-15, & j=30 \\ \sqrt{3j}, & j=100 \end{cases}$

6. $y(x)=\begin{cases} \sin x & x<\pi \\ \cos x & \pi\le x<3\pi \\ 1 & x\ge 3\pi \end{cases}$

7. $v(t)=\begin{cases} 3e^{-2t} & 0\le t\le 10 \\ 4t+2 & 12<t<25 \end{cases}$

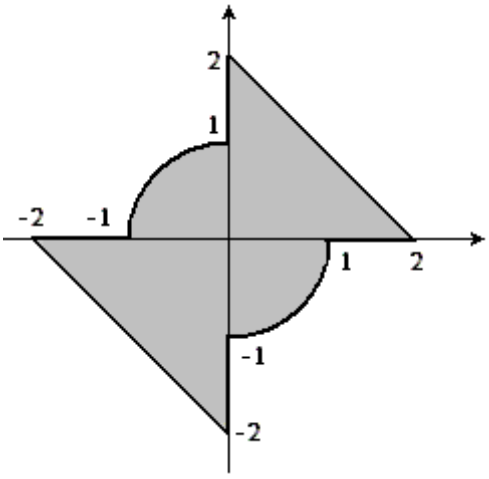
8. $z(x)=\begin{cases} \dfrac{4x+5}{3} & 0<x\le 25 \\ 23x^2+7 & 25<x\le 45 \end{cases}$

9. $x(a) = \begin{cases} a + a^2 + 5 & a < 0 \\ \dfrac{a}{10 + a} & a \geq 0 \end{cases}$

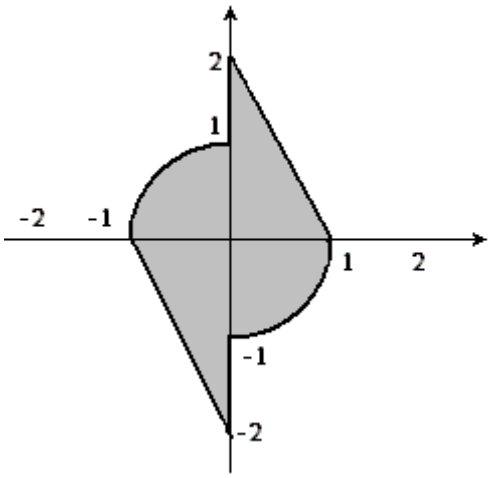
10. $y(t) = \begin{cases} 10t & t \leq 0 \\ 10 + t/7 & t > 0 \end{cases}$

II. Составить алгоритм программы, которая по введенному значению координат точки определяет, попадает ли точка в мишень, представленную рисунком:

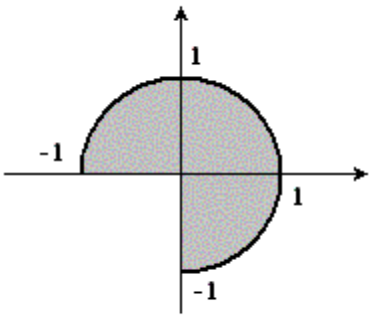
1.



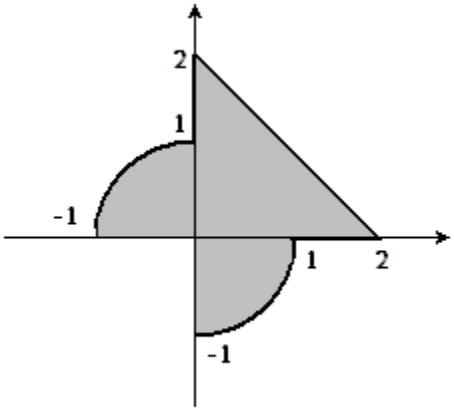
2.



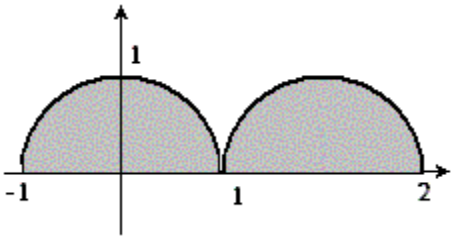
3.



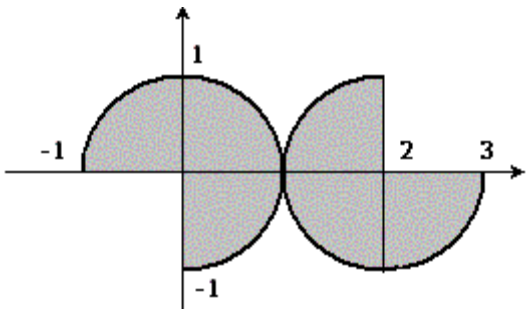
4.



5.

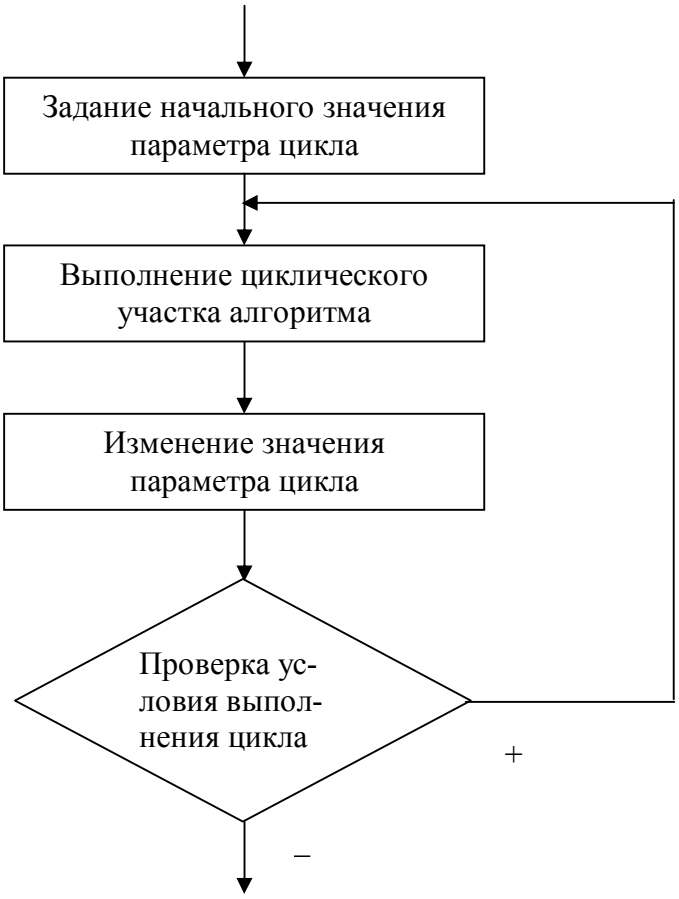


6.



ЦИКЛИЧЕСКИЙ АЛГОРИТМ

Циклический алгоритм используют для организации многократно повторяющихся вычислений. Циклический алгоритм существенно сокращает объем программы.



Любой цикл состоит из *тела цикла* (действий, которые повторяются несколько раз), начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

На схеме изображены типовые составляющие циклического алгоритма.

Один повтор выполнения операторов тела цикла называется *итерацией*. Проверка условия выполняется на каждой итерации.

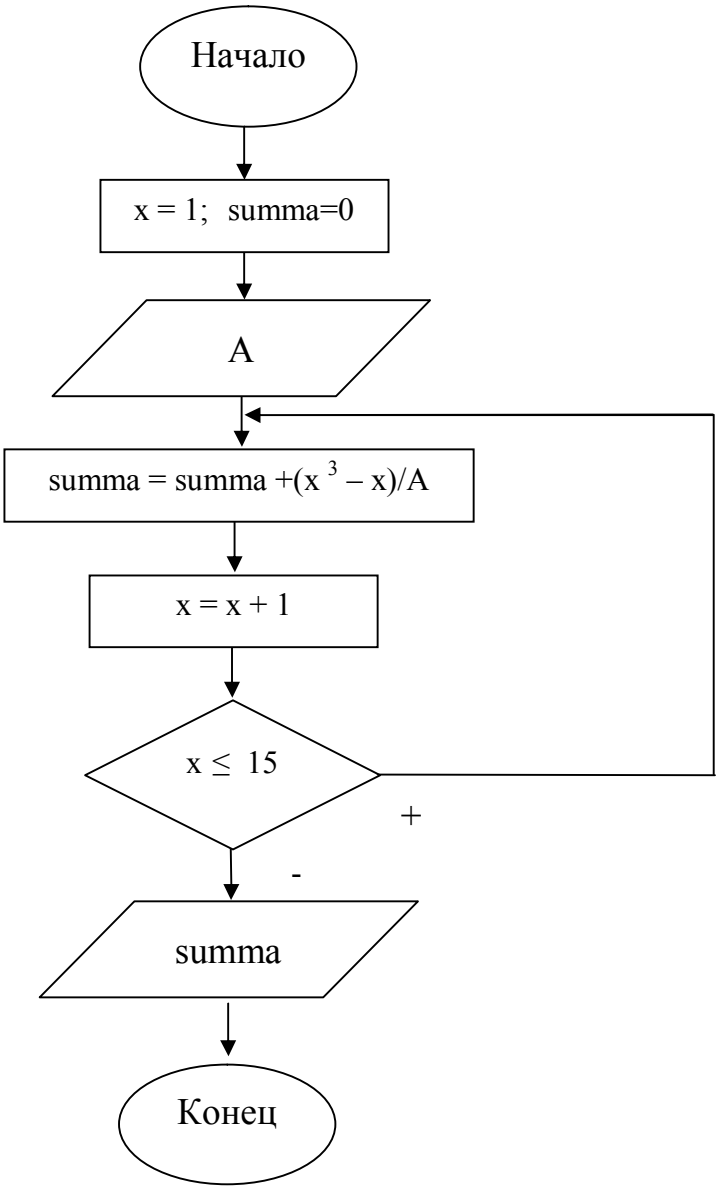
Переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения цикла, называются *параметрами цикла*.

Начальные установки используются для того, чтобы до входа в цикл задать значения переменным, которые в нем используются, в том числе и начальные значения параметров цикла.

Цикл завершается, если условие его выполнения становится ложным.

Составить блок-схему алгоритма вычисления $y = \sum_{x=1}^{15} (x^3 - x) / A$,

где $A - const$.



Задание

I. Посчитать значение функции

1. $y = \sum_{1}^{14} (x^2 - x) / 2$
2. $y = 25 \sum_{5} \sin^2 x$
3. $y = \sum_{3}^{20} (9x - 5)^2$
4. $y = \prod_{1}^{10} \frac{1}{2i}$
5. $y = \prod_{i=10}^{20} \frac{\left(i^2 + 2\right)}{i^3}$

II. Посчитать все значения функции на указанном интервале

1. $y = t^2 - 5t + 1 \qquad t \in [2,\ 12],\quad \Delta t = 0.5$

2. $y = \begin{cases} \sin 2t, & t < 0 \\ \cos 3t, & t \ge 0 \end{cases} \qquad t \in [-4,\ 6],\quad \Delta t = 1$

3. $y = \begin{cases} 2t - 5, & t > 0 \\ 0, & t = 0 \\ |t|, & t < 0 \end{cases} \qquad t \in [-10,\ 45],\ \Delta t = 5$

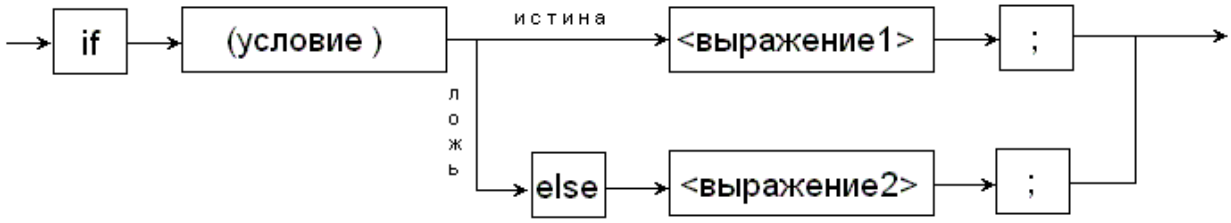
4. $y = \begin{cases} e^{-t}/6, & t \le 10 \\ 45, & t > 10 \end{cases} \qquad t \in [4,\ 11],\quad \Delta t = 0.25$

5. $y = \begin{cases} \lg x - 3, & x > 0 \\ 1, & x = 0 \\ \lg|x| - 3, & x < 0 \end{cases} \qquad t \in [-5,\ 7],\quad \Delta t = 2$

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ И ЦИКЛИЧЕСКОЙ СТРУКТУРЫ НА ЯЗЫКЕ С++

Условный оператор if

Условный оператор if используется для разветвления процесса вычислений на два направления. Структурная схема оператора представлена на рисунке.



Сначала вычисляется выражение. Если оно не равно нулю или имеет значение истина, выполняется первый оператор, иначе второй. После этого управление передается следующему оператору. Например, вычисление модуля переменной f можно записать как

```
if ( f >=0.0 )  f = f;  else f = -f;
```

Ветвь с ключевым словом else может отсутствовать.

```
if ( f != 0 )  I = 100 / f;
```

Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок (операторные скобки { }), иначе компилятор не сможет определить окончание ветвления.

```
if ( a ) { a++;  v=60*a };  else v = a;
if ( e==1000 ) { e /=10;  cout << "e= " << e; }  else { e =10+y*y;  y++; }
```

Блок может содержать любые операторы, в том числе и другие условные.

```
if ( a<b ) { if ( a<c ) m = a;  else m = c; }  else { if ( b<c ) m = b;  else m = c; }
```

Необходимо помнить, что в этом случае else относится к ближайшему if. Операторные скобки после первого if необязательны.

Если требуется проверить несколько условий, их объединяют знакам логических операций.

```
if ( a<b && ( a>d || a==0 ) )  b++; else { b*=a; a=0; }
```

Записанное условие будет истинно в том случае, если выполнится одновременно условие a<b и одно из условий в скобках. Если опустить внутренние скобки, будет выполнено сначала логическое И, а потом ИЛИ.

Операторы цикла

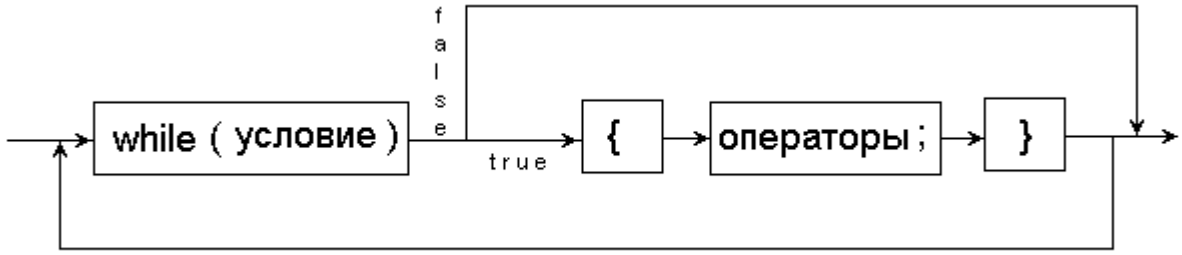
Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из *тела цикла*, т. е. операторов, которые повторяются несколько раз, начальных установок, модификации *параметра цикла* и проверки условия продолжения выполнения цикла.

Для удобства в С++ существуют три разных оператора цикла – while, do while, for.

Цикл с предусловием (while)

В цикле с предусловием проверка условия продолжения цикла выполняется перед телом цикла.

Цикл с предусловием реализован в С++ оператором цикла while, структурная схема которого представлена на рисунке.



Выражение, стоящее в круглых скобках, определяет условие повторения тела цикла, представленного простым или составным оператором. Если оператор простой операторные скобки { } не ставятся.

Выполнение оператора цикла начинается с вычисления выражения. Если оно истинно, выполняется тело цикла. Если при первой проверке выражение ложно (false), цикл не выполнится ни разу.

Тип выражения должен быть арифметическим или приводимым к нему. Выражение вычисляется перед каждой итерацией цикла.

Пример. Программа печатает таблицу значений функции $y = x^3 - x$ во введенном диапазоне.

```
#include <iostream.h>
#include <iomanip.h>
void main ( )
{ float x1, xn, d;
  cout << " введите диапазон и шаг изменения x"<< endl;
  cin >> x1 >> xn >> d;
  cout << "|        x        |        y       |" << endl;     // шапка таблицы
  float x = x1;
  while ( x <=  xn )
        { cout << "|" << setw(6) << setprecision(3) << x << "|";
          cout << "|" << setw(6) << setprecision(3)<< x*x*x – x << "|" << endl;
          x+=d;
        }
```

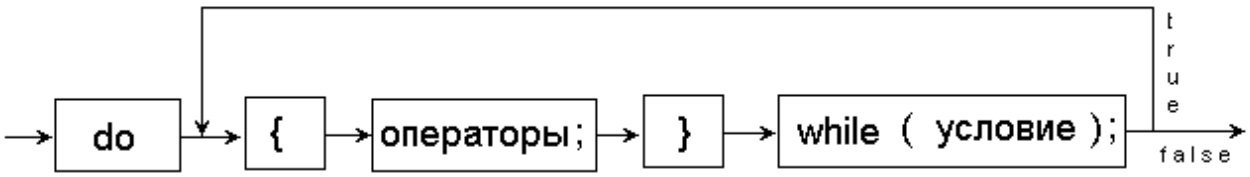
}

Цикл while обычно используется в тех случаях, когда число повторений заранее не известно.

Цикл с постусловием (do while)

Тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз.

Цикл с постусловием реализован в С++ оператором цикла do … while и имеет вид:



Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение, составляющее условие выполнение цикла. Даже если условие заведомо ложно цикл выполняется один раз. Если условие истинно, тело цикло выполнится еще раз. Цикл завершается, когда выражение станет равным false или в теле цикла будет выполнен оператор передачи управления.

Пример. Программа угадывания загаданного числа.

```
#include <iostream.h>
#include <stdlib.h>
void main ( )
{ float x, y;
  randomize( );
  y = random (10);
do
  { cout << " введите произвольное число меньшее 10"<< endl;
    cin >> x;
    if ( x = = y) {cout <<" вы угадали!"; break; }
   else          if  ( x< y ) cout<< "введите меньшее"<<endl;
                 else     cout<< "введите большее"<<endl;

  } while (x!=y);
}
```

Цикл с параметром (for)

Цикл for называют также циклом с заданным числом повторений. Он имеет следующий формат:

for (инициализация; выражение (условие) ; модификации) оператор;

Инициализация используется для объявления и присвоения начальных значений величинам, используемым в цикле. Инициализация выполняется один раз перед выполнением тела цикла.

Выражение определяет условие выполнения цикла: если его результат равен истине, то цикл выполняется. Цикл с параметром реализуется как цикл предусловием.

Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла.

Тело цикла представляет собой простой или составной оператор.

Любое из трех выражений, указанных в скобках, является необязательным и может быть опущено. Точки с запятой должны оставаться на своих местах, даже если все три выражения отсутствуют.

Примеры

```
for ( ; ; );        // пример бесконечного цикла
for (y=2; y<20; y++)  r+=y;
```

В инициализации и модификации параметра можно писать несколько операторов, разделенных запятой. Если в теле цикла требуется выполнить несколько операторов, используют операторные скобки.

```
for (i = 1, s=1; i<11; i++) s*=i;        // вычисления факториала 10
```

Цикл for обычно используется в тех случаях, когда можно точно определить необходимое число повторов.

В C++ допускается объявление переменной прямо в строке инициализации цикла for. Значение счетчика цикла может не только увеличиваться, но и уменьшаться, причем на произвольный шаг, который может быть не только целым, но и числом с плавающей точкой.

```
for (float k = 11.5; k>0.5; k - = 0. 5) s+=k;
```

Допускается в цикле и работа с символьными переменными.

```
for (char ch='a'; ch< 'z'; ch++) …
```

Если тело цикла содержит более одной команды, следует использовать фигурные скобки и руководствоваться определенными правилами оформления, чтобы сделать текст более наглядным.

Любой цикл while может быть приведен к эквивалентному ему циклу for и наоборот. Операторы цикла взаимозаменяемы, но можно привести рекомендации по выбору наилучшего в каждом конкретном случае.

Оператор do while обычно используют, когда цикл требует обязательно выполнить хотя бы раз (например, если в цикле производится ввод данных).

Оператором while удобнее пользоваться в случаях, когда число итераций заранее неизвестно, или нет очевидных параметров цикла, или модификацию параметров удобнее записывать не в конце тела цикла.

Задание

Написать программы по заданиям предыдущего раздела.

МАССИВЫ

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой переменных одного типа требуется выполнить различные действия, им дают одно имя и отличают по порядковому номеру. Это позволяет записывать множество операций через циклы. Конечная именованная последовательность однотипных величин называется массивом. Для создания массива компилятору необходимо знать тип данных, количество элементов в массиве и требуемый класс памяти. Массивы могут иметь те же типы данных, что и простые переменные.

Синтаксис объявления массива следующий:

<Тип данных> <имя массива> [размерность массива];

Примеры:

```
int array[45];              //массив из 45 элементов целого типа с именем array
double rt[12];              // массив rt, состоящий из 12 элементов типа double
const int ARRAY_SIZE=90;
float alp[ARRAY_SIZE]; // вещественный массив alp из 90 элементов
```

Размерность массива предпочтительнее задавать с помощью именованных констант, как это сделано в примере, так как при таком подходе для ее изменения во всей программе

достаточно изменить значение константы в одном месте.

Возможна также инициализация массива при его объявлении.

int a[5]={4, 90, 71, 45, 3};

При инициализации допускается использование пустых скобок в объявлении массива, и тогда компилятор сам определяет размерность массива. Например,

float x[ ]= {4, 8, 19};

В памяти компьютера элементы массива располагаются последовательно друг за другом. Место в памяти для хранения элементов массива выделяется компилятором сразу после обработки его объявления.

Так объявление вида

float y[5]= {0.7, 1.9, 8.4, 3.1 };

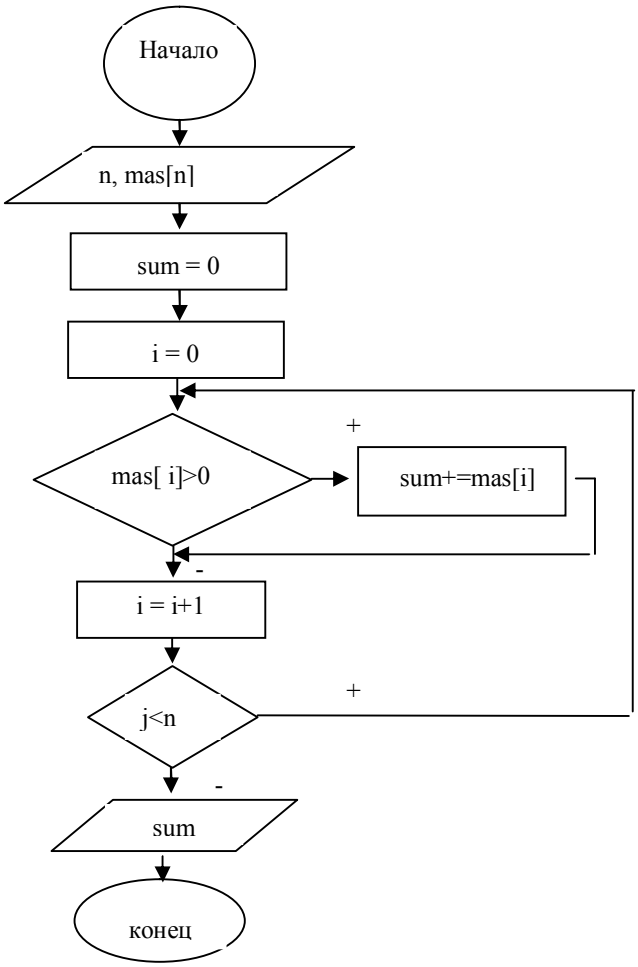
выделит место для расположения пяти элементов типа float. Так как выполнена инициализация четырех первых элементов, они сразу займут свое место в памяти компьютера, а ячейки памяти для пятого элемента пока останутся свободными:

0. 7	1. 9	8. 4	3. 1	
y[0]	y[1]	y[2]	y[3]	y[4]

Для доступа к элементу массива после его имени указывается номер элемента (индекс), заключенный в квадратные скобки. Нумерация элементов массива начинается с нуля. Следовательно, диапазон значений для объявленного в примере массива x лежит в пределах от 0 до 2. Следовательно, x[0]=4 x[1]=8 x[2]=19. К элементам массива можно применять все операции, допустимые для обычной переменной данного типа.

Работа с элементами массивов осуществляется на основе циклических алгоритмов.

Пример: Составим алгоритм подсчета суммы положительных элементов массива.



Тогда программа подсчета суммы положительных элементов массива.

```
#include<iostream.h>
void main ( )
{const int n=10;
 int i, mas[n], sum=0;
 cout << " введите" << n << "чисел" <<endl;
                  // ввод элементов массива с клавиатуры
 for ( i=0; i < n; i++)       cin >> mas[i];
 for ( i=0; i < n; i++)
         if ( mas[i] > 0)     sum+= mas[i];
 cout << "сумма положительных элементов равна "<<s<<endl;
}
```

Для улучшения эффективности алгоритма ввод элементов массива и подсчет их суммы можно объединить в один цикл:

Для объявления многомерного массива необходимо после имени массива задать несколько размеров, заключенных в скобки. Двумерный массив в С++ представляет собой массив одномерных массивов.

```
float dam[4] [5];
```

В этом случае массив будет содержать 4 строки и 5 столбцов. Инициализация двумерного массива происходит следующим образом:

```
float art [5] [2]={  { 1. 2, 1. 5 },
                     { -4. 0, 3. 6 },
                     { 2. 3,  -6.1 },
                     { 7. 3,  0. 4 },
                     { 0. 0, -2. 7 }  };
```

При этом внутренние фигурные скобки могут быть опущены. Например,

```
float art [5] [2]= {1. 2, 1. 5, -4. 0, 3. 6, 2. 3, -6. 1, 7. 3, 0. 4, 0. 0, -2. 7} ;
```

Все сказанное может быть распространено на трехмерный массив.

```
int rum [3] [7] [2]; // массив размерности три, каждый элемент которого двумерный массив.
```

Если одномерный массив используется для представления списка, то двумерный массив – для представления таблицы, содержащей строки и столбцы. При этом подразумевается, что все элементы принадлежат к одному типу данных. При обращении к отдельному элементу двумерного массива нужно указать его позицию в строке и столбце. Нумерация строк и столбцов также начинается с нуля. Массив можно представить как таблицу на рисунке.

	[0]	[1]
[0]	1. 2	1. 5
[1]	-4. 0	3.6
[2]	2. 3	- 6. 1
[3]	7. 3	0. 4
[4]	0. 0	- 2. 7

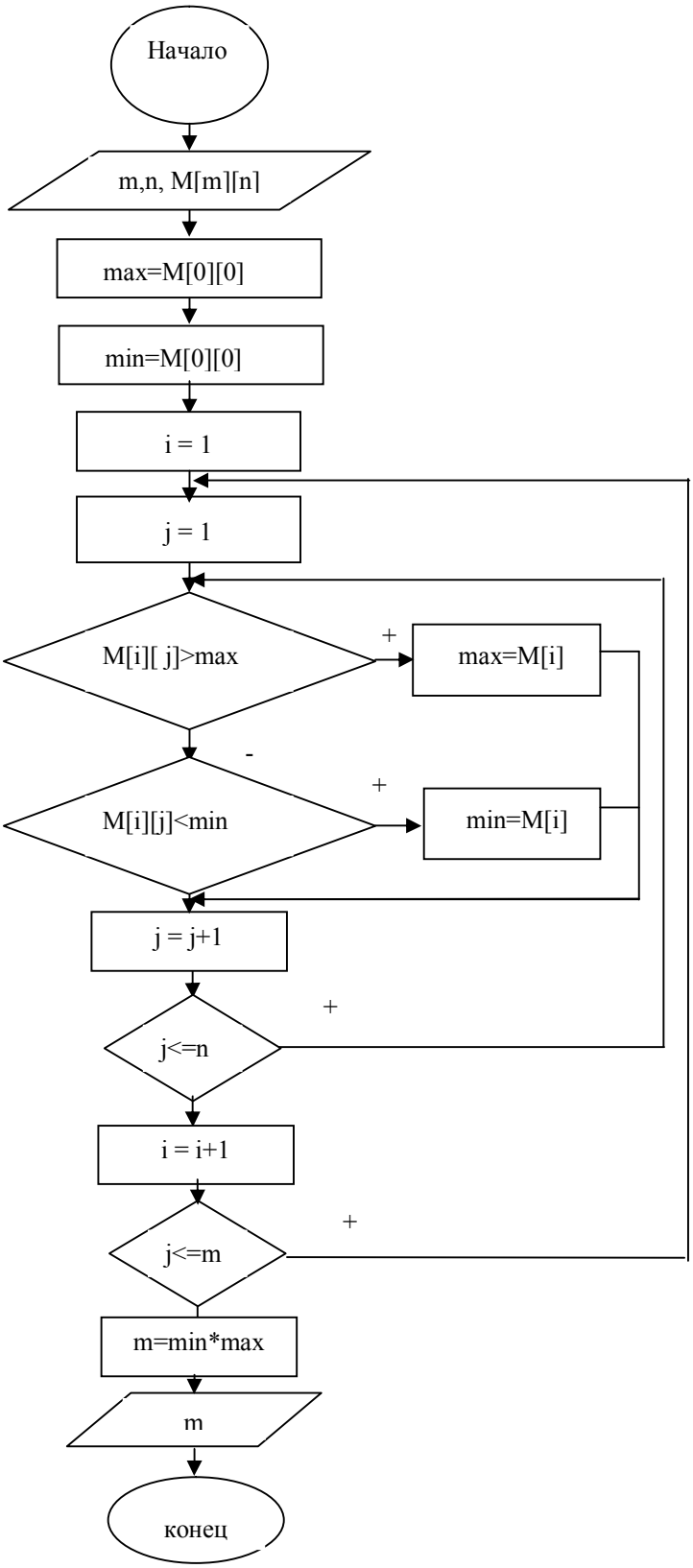
столбцы

строки

Выделенный элемент есть art [1] [2].

Обработка данных в двумерных массивах означает обращение к массиву одним из четырех способов: случайным образом, по строкам, по столбцам, по всему массиву.

При работе по строкам внешним будет цикл, параметром которого является номер строки, а внутренний с параметром – номером столбца. При работе по столбцам наоборот. Пример. Определение произведения минимального и максимального значений среди элементов матрицы размерности m на n.



```
#include <iostream. h>
#include <stdlib.h>
void main ( )
{
const int n = 6;
const int m = 8;

int M[m][n], i, j, min, max;
randomize( );
for (  i = 0; i <m; i++ )
        for (   j = 0; j < n; j++)
        M [ i ][ j ]=random(20) - 10; //задание значений
for (  i = 0; i < n; i++ )                          // вывод на экран полученной матрицы
        {       for ( int  j = 0; j < n; j++)       cout << M [ i ][ j ]<< “  “;
                cout << endl;
        }
min = M[0] [0], max = M[0] [0],;
for (  i = 1; i < m; i++ )
        for (   j =1; j < n; j++)
        if ( M[ i ][ j ]<min)  min = M[ i ][ j];
        else if ( M[ i ][ j ]>max)  max = M[ i ][ j];

 cout<< “значение произведения минимума на максимум ”<<min*max;
}
```

Если требуется преобразовать элементы, расположенные в матрице под главной диагональю и на ней, используют вложенные циклы вида

```
for (  i = 0; i < n; i++ )
        for (   j =i; j < n; j++)
```

В случаях работы с элементам матрицы, лежащими на главной диагонали, достаточно использовать один цикл.

Пример. Поменять местами максимальный и минимальный элементы главной диагонали матрицы T(7, 7).

```
#include <iostream. h>
#include <stdlib.h>
void main ( )
{
const int n = 7;
int T[n][n], i, j, min, max, nmin, nmax;
randomize( );
for (  i = 0; i < n; i++ )
        {       for ( int  j = 0; j < n; j++)
                T [ i ][ j ]=random(30) - 15;
                cout << T [ i ][ j ]<< “  “;
                cout << endl;
        }
min = max=T[0] [0];
nmin=nmax=0;
for (  i = 1; i < n; i++ )
        {
        if ( min > T[ i ][ i ] )   { min = T[ i ][ i ];     nmin = i; }
```

```
            if ( max < T[ i ][ i ] )  { max = T[ i ][ i ];    nmax = i; }
            }

      int t = T[nmin][nmin];                  // вспомогательная переменная для обмена
      T[nmin][nmin] =T[nmax][nmax];
      T[nmax][nmax] =t;

      for (  i = 0; i < n; i++ )                        // вывод на экран измененной матрицы
            {     for ( int  j = 0; j < n; j++)       cout << T [ i ][ j ]<< “  “;
                  cout << endl;
            }
}
```

Задание

Составить алгоритм и программу на языке С++ :

1. Определите номер максимального элемента одномерного массива.
2. Найти произведение отрицательных элементов одномерного массива.
3. В одномерном массиве поменять местами максимальный и минимальный элементы
4. Найти произведение элементов одномерного массива, расположенных между его максимальным и минимальным элементами.
5. Найти разность количества отрицательных и нулевых элементов одномерного массива.
6. Определить значение и номера столбца и строки минимального элемента матрицы размерности 6 на 10.
7. В каждой второй строке матрицы размерности 8 на 5 заменить положительные элементы нулями.
8. Посчитать среднее арифметическое отрицательных элементов, расположенных под главной диагональю матрицы размерности 5 на 5.
9. В каждом третьем столбце матрицы размерности 5 на 7 определить произведение элементов
10. В каждой строке матрицы размерности 6 на 8 поменять местами максимальный и минимальный элементы.

ФУНКЦИИ

Компьютерная программа состоит из команд, которые сообщают компьютеру, что он должен делать. Задача большинства программ заключается в распознавании данных, их обработке и вывода их на экран.

Любая программа, написанная на языке С++, предполагает последовательность выполнения нескольких функций, причем одна из них обязательно должна носить имя main( ). Выполнение программы всегда начинается с выполнения ряда операторов этой функции. Операторы языка отделяются друг от друга точкой с запятой.

Описание функции состоит из заголовка и тела.

Заголовок <тип результата> main (список аргументов) имя функции

{ *Тело функции* }

После заголовка функции точка с запятой не ставится. Тело функции может содержать любое количество операторов языка.

Если тип результата не указывается, предполагается, что функция возвращает (передает в вызываемую функцию) значение типа int. Функция не обязательно возвращает значение – она просто может выполнять какие-либо действия. Если функция не возвращает результат – в заголовке перед ее именем указывается ключевое слово void (пустой).

Аргументы – значения, которые можно передавать в функцию, их количество аргументов может быть любым. Наличие списка аргументов необязательно, но круглые скобки всегда должны присутствовать после имени функции.

Часто употребляется описание типов аргументов сразу при их объявлении внутри круглых скобок. Если аргументов несколько, их описания (тип и имя) разделяются запятыми. Если функции не передаются величины, то вместо списка аргументов можно задавать ключевое слово void. Локальные переменные, отличные от аргументов, описываются внутри тела функции. Процесс использования функции называется вызовом (или запуском) функции. Вызывать функцию можно сколько угодно раз. Функция должна быть определена до того, как начнется ее использование. Для вызова функции, ничего не возвращающей, достаточно написать ее имя, в круглых скобках указать значения фактических параметров – значений аргументов, которые непосредственно участвуют в работе функции. Если функция не имеет аргументов, при ее вызове также ставятся пустые круглые скобки.

Программа содержит пример функции, вычисляющей значение квадратного корня из введенной величины.

```
#include <stdio.h>          //подключение стандартной библиотеки ввода-вывода.
#include <math.h>           //подключение библиотеки математических функций.
void outsqr (void)          //заголовок функции.
{
  int it;
  printf (“Введите целое число”);
  scanf ( “%d”, &it);
  double du;
  du = sqrt(it);
  printf (“Квадратный корень числа %d равен %f \n”, it, du);
}
main ( )
{
 printf (“Эта программа вычисляет квадратный корень целого числа. \n”);
 outsqr ( );
}
```

В начале программы записан ряд директив препроцессора #include, по которой к исходному тексту программы подключаются заголовочные файлы, стоящие в угловых скобках. Заголовочные файлы включают в себя описания функций, которыми оперирует программа. Так, подключение файла stdio.h содержит объявления констант, переменных и функций, необходимых при вводе-выводе. Основные функции библиотеки math.h приведены в приложении 3.

Строка #include обрабатывается не компилятором С++, а специальной программой – препроцессором. Она называется директивой препроцессора и не является выражением языка С++. Точка с запятой после нее не ставится. Директивы препроцессора записываются в отдельной строке.

Рассмотренная функция не принимает значений, т.е. не имеет аргументов, и также не возвращает значений.

В следующем примере рассмотрим функцию, вычисляющую среднее арифметическое значение трех величин и возвращающую его в вызываемую функцию**.** Возвращает значение оператор return, с помощью которого можно передать в вызывающую функцию только одно значение. Для этого возвращаемое значение помещается после указанного оператора. После оператора return может стоять выражение, результат вычисления которого также передается вызывающей функции. Возвращаемое значение (выражение), следующее за return, можно брать в круглые скобки, что не является синтаксическим требованием языка.

```
long Average (long val1, long val2, long val3 )
{
  long sum = val 1 + val 2 + val 3;
  return sum / 3;
}
```

Пусть в программе объявлены некоторые переменные: long a, a1, a2, a3; тогда возможен вызов описанной в примере 2 функции в виде:

a = Average (a1, a2, a3); или a = Average (525, 675, 819);

Необходимо отметить, что при вызове функции аргументы должны стоять в той же последовательности, что и в ее описании.

Использование функций в программе нужно для упрощения структуры программы. Как правило, в функцию выделяют группу операторов, выполняющих законченное действие, в случаях, когда это действие необходимо выполнить неоднократно.

Программа, которая по введенным числам выводит на экран их значения, возведенные в куб.

```
#include <iostream. h>
void square (float x)          //описание функции square
{
  float y=x*x*x;
  cout<<"Квадрат числа  "<<x<<"равен "<<y<<endl;
}

void main( void)               //описание функции main
{
int k;
   float value;
```

```
    cout<<”Введите количество чисел”<<endl;
    cin>>k;
    for (int i=1; i<=k; i++)
     {
          cout<<”Введите число”<<endl;
          cin>>value;
          square (value);
     }
}
```

Нередок случай, когда необходим вызов функция до ее объявления. В этом случае используют прототип функции, который информирует компилятор о существовании данной функции, о типе возвращаемого ею значения, а также о типе и количестве ее аргументов.

Прототип функции имеет следующий вид:

< возвращаемый тип > имя функции (параметры);

Таким образом, прототип функции аналогичен ее заголовку, но после него обязательно ставится точка с запятой, указывающая, что тело функции здесь опущено, и определение функции будет позднее. В списке параметров обычно указываются тип и имя для каждой переменной; элементы списка разделяются запятыми. Указывать имя переменной в прототипе не обязательно, но, как правило, применяется. Так, для функций из рассмотренных выше примеров прототипы выглядят следующим образом:

```
long Average (long val1, long val2, long val3 );
void square (float x);
```

или, если не указывать имена параметров:

```
long Average (long, long , long );
void square (float);
```

Прототипы функций, как правило, располагают после директив препроцессора, до описания основной функции. Рассмотрим пример программы нахождения площадей треугольников с использованием прототипов функций.

```
#include <iostream.h>
#include <math. h>
void print_area (float, float, float);        //объявление прототипа функции print_area
void main (void)                              //описание функции main
{
  int n; float a, b, c;
  cout<<”Введите количество треугольников”<<endl;
  cin>>n;
  for (int i=1; i<=n; i++)
    {
        cout<<”Введите стороны треугольников a, b, c >0”<<endl;
        cout<<”a = “;
        cin>>a;
```

```
            cout<<"b = ";
            cin>>b;
            cout<<"c = ";
            cin>>c;
            print_area (a, b, c);
        }
    }

  void print_area (float x, float y, float z)     //описание функции print_area
        {
        if ( ( x+y>z )&&( x+z>y )&&( y+z>x ) )
        {float p=(x+y+z)/2;
          cout <<"Площадь равна "<<sqrt(p*(p-x)*(p-y)*(p-z))<<endl;
        }
        else cout<<"Треугольник невозможно построить"<<endl;
        }
```

Все функции в языке С++ равноправны: каждая из них (даже main) может быть вызвана любой другой функцией. Функция может также вызывать самое себя (явление рекурсии). Количество рекурсивных вызовов не ограничивается, а определяется лишь возможностями компьютера.

Программа, демонстрирующая применение рекурсии при вычислении факториала.

```
#include <stdio. h>
double factorial (int number);          // объявление прототипа функции factorial
int main ( )
{
   int n;
   double result;
   printf ("Введите число\n");
   scanf ("%d", &n);
   result = factorial(n);
   printf("факториал %d равен %15.0f\", n, result);
   return (0);
}
double factorial (int number)   // описание функции factorial
{
   if (number<=1)              return (1.0);
   else return (number*factorial (number – 1));
}
```

Задание

1. Определить периметры двух треугольников, заданных координатами их вершин. Длину стороны треугольника вычислить в функции.
2. Вычислить среднее арифметическое объемов шаров с радиусами r_1, r_2, r_3. В программе выделить функцию нахождения объема шара.
3. Заданы три конуса (радиусы основания и высота). Определить конус с наибольшим объемом. В программе выделить функцию нахождения объема конуса.
4. Вычислить значение $z = (sign\, x + sign\, y)\, sign\,(x + y)$,

где $sign\, a = \begin{cases} -1, & при \quad a < 0, \\ 0, & при \quad a = 0, \\ 1, & при \quad a > 0. \end{cases}$

5. Составить программу вычисления значений:

$$a = \frac{\sqrt{z^3 + 4z^2 + 7z + 1}}{2 + e^{2z^3 + z - 3}}, \quad b = \frac{t^2 + 13t + 16}{7t^3 + t^2 - 4}.$$

Для определения значений многочленов использовать функцию. Значения величин *z, t* вводить с клавиатуры.

6. Вычислить значение $f = \dfrac{\max(a,b,c) * \min(a,c,d) - \max(b,c,d)}{\min(a,b,c)}$,

где *a, b, c, d* – некоторые значения, введенные с клавиатуры. В программе выделить функции нахождения максимального и минимального значений.

7. Создать функцию вывода на экран таблицы умножения для числа по запросу пользователя.

8. Создать функцию вычисления площади треугольника по трем заданным сторонам.

9. Создать функцию, определяющую, является ли введенное число четным. Если это не так, удвоить его значение.

10. Создать функцию, определяющую, принадлежит ли точка с заданными координатами (x_1 , y_1) уравнению заданной прямой $y = kx + b$. Использовать функцию для определения из набора точек, лежащих на одной прямой.

II. КУРС ЛЕКЦИЙ

Глава 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Этапы решения задач

Программированием называют планирование, распределение во времени или исполнение заданий или событий. Компьютерное программирование – планирование последовательности шагов, которую должен выполнить компьютер.

Компьютерная программа состоит из команд (инструкций), сообщающих компьютеру, что он должен делать и каким образом обрабатывать данные. Задачей большинства программ является распознавание данных, их обработка и вывод на экран в требуемом формате.

Процесс создания программы состоит из двух основных этапов: *решение задачи* и ее *реализация*. Этап решения задачи включает в себя:

- *анализ и спецификацию*, т.е. понимание и определение сути задачи, а также технических и функциональных требований к ее решению;
- *общее решение (алгоритм)* - разработку логической последовательности шагов, приводящих к решению поставленной задачи;
- *проверку*, т.е. подтверждение правильности решения, например, повторением всех его этапов.

Этап реализации содержит:

- *конкретное решение* (*программу*) - перевод алгоритма на язык программирования;
- *тестирование* - запуск программы на компьютере, а затем проверка результата вручную. При обнаружении ошибок проводится анализ программы и алгоритма, нахождения источника ошибок и их исправление.

После выполнения перечисленных этапов начинается этап *сопровождения* включающий:

- *использование* (*эксплуатацию*) программы;
- *поддержку* программы, т.е. изменение в соответствии с требованиями пользователей, а также исправление ошибок, выявленных при ее эксплуатации.

При модифицировании программы, необходимо повторить этапы решения и реализации для тех частей программы, которые были изменены.

Совокупность всех трех этапов – решения, реализации и сопровождения – называется *жизненным циклом программы*.

Процесс программирования начинается с анализа задачи, разработки решения, называемого *алгоритмом*. Компьютер не обладает интеллектом, он не способен самостоятельно решать задачи. Компьютер способен лишь очень быстро и безошибочно повторить процедуру решения, подготовленную программистом. Поэтому понимание и анализ задачи является центральной частью процесса программирования.

Каждая компьютерная программа – это реализация определенного алгоритма. *Алгоритм* – устное или письменное описание логической последовательности действий. Алгоритм отражает последовательность действий, которые могли быть выполнены вручную.

Разработав решение, программист проверяет его, выполняя каждый шаг самостоятельно. Если выясняется, что алгоритм не работает, следует снова анализировать задачу и искать новый алгоритм. После получения правильного алгоритма программист переводит его на язык программирования.

Язык программирования – набор правил, символов и специальных слов, используемых для построения программ.

Любой язык программирования представляет собой сильно упрощенную форму естественного языка (как правило, английского), дополненную математическими символами. За счет ограничения словаря и строгих грамматических правил язык программирования управляет компьютером. Ограниченность и однозначность конструкций языка программирования

вынуждает программистов создавать простые и точные инструкции.

Процесс перевода алгоритма на язык программирования называется *кодированием* (*программированием*). Его результатом является код программы (или просто программа), которая проверяется *запуском* (или *выполнением*) на компьютере. Программа, не выдающая желаемых результатов, нуждается в *отладке*.

Отладка программы поиск причины ее ошибочного поведения и изменение фрагментов программы (или даже самого алгоритма) для устранения этой причины.

Реализацией алгоритма называется его совместное кодирование и тестирование. Если не затрачено достаточное время на обдумывание и разработку алгоритма, отладка и исправление программы может потребовать много дополнительных усилий.

Наряду с решением задачи, реализацией алгоритма и сопровождением программы важной частью процесса программирования является *разработка документации.* Документацией называют текст и комментарии, которые упрощают программу для понимания, использования и изменения.

Языки программирования

Компьютер выполняет инструкции встроенной системы элементарных команд - машинного языка, которой состоит из команд в двоичном представлении.

Языки программирования высокого уровня позволяют создавать компьютерные программы на языке, близком к естественному языку. Их легче использовать, чем машинный код. Примерами языков программирования являются Pascal, C++, Fortran, Java и др.

Специальная программа, называемая *компилятором*, переводит программу, написанную на языке высокого уровня в машинный код.

Программа, написанная на языке высокого уровня, называется *исходной программой* и может быть выполнена на любом компьютере в соответствии с используемым компилятором. Это возможно, поскольку большинство языков высокого уровня стандартизировано.

Компилятор транслирует исходную программу в программу на машинном языке, которая называется *объектной программой*. Если программа содержит ошибки, компилятор создает *листинг* программы – текст с сообщениями об ошибках и другой полезной информацией.

Объектная программа отправляется на *выполнение*. Необходимо разделять понятия компиляции и выполнения программы. Во время компиляции запускается компилятор, создающий объектную программу. При выполнении объектная программа загружается в память компьютера, который выполняет команды в соответствии с исходной программой.

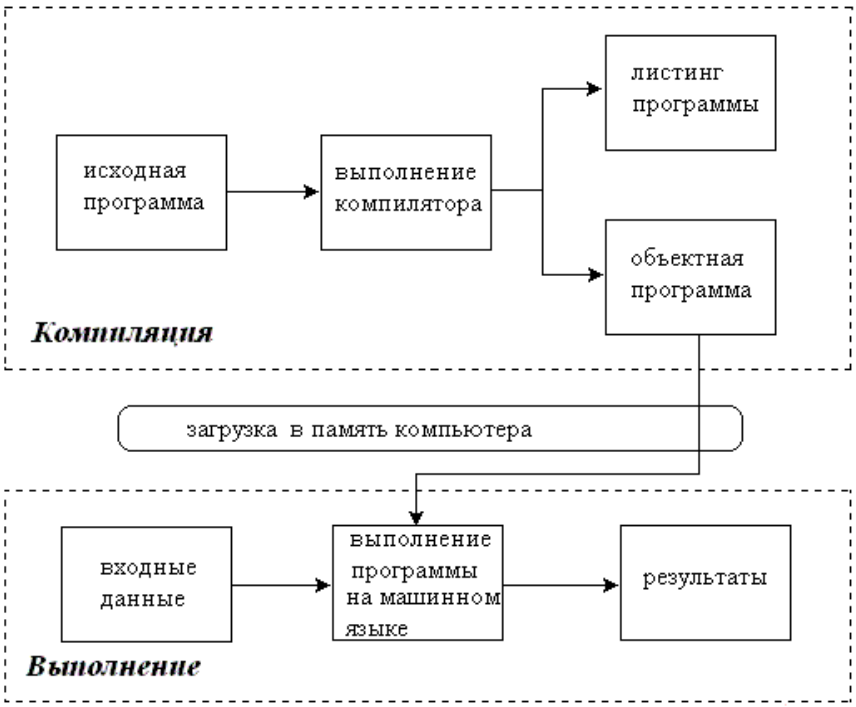


Рисунок 1. Компиляция и выполнение программы

Существуют синтаксические (грамматические) и семантические (смысловые) правила создания программы.

Синтаксис – формальный набор правил, определяющий, какие комбинации цифр и букв могут использоваться в языке программирования. Синтаксис однозначно определяет правила построения конструкций языка. При наличии синтаксических ошибок программа не будет скомпилирована.

Семантика определяет значения команд языка.

Глава 2. БАЗОВЫЕ КОНСТРУКЦИИ ЯЗЫКА С++

Среди современных языков программирования С++ относят к числу наиболее распространенных. В его основу положено значительно меньше синтаксических правил, чем у других языков программирования. Язык менее строго структурирован и предоставляет программисту свободу выбора альтернативных решений одной проблемы.

Характерные для многих языков программирования конструкции языка, напоминающие выражения на английском языке, в С++ встречаются довольно редко. Язык С++ содержит операторы необычного вида и часто использует указатели.

С++ универсален, однако его применение наиболее эффективно в задачах системного программирования – разработке трансляторов, интерфейсов, операционных систем.

Язык С++ поддерживает полный набор структурного и объектно-ориентированного программирования: модульность, блочную структуру программ, раздельную компиляцию, характернее для языков высокого уровня. С другой стороны, он имеет ряд низкоуровневых черт (в частности, операции над битами).

Непосредственным предшественником языка С++ был язык С с классами. В основу языка С было положено значительно меньше синтаксических правил, чем у других языков программирования. В результате для эффективной работы компилятора языка достаточно всего 256 Кб оперативной памяти. Первоначально, в том виде, в каком его создал Деннис Ритчи в 1972, С содержал всего 27 ключевых слов. После чего язык стал бурно развиваться. В 1983 г., при Американском институте национальных стандартов (ANSI) был создан специальный комитет с целью стандартизации языка. Стандарт ANSI языка С включал уже более 50 ключевых слов.

Многие функции, представленные в большинстве других языков, не включены в С++. Например, в С++ нет встроенных функций ввода-вывода, отсутствуют встроенные математические функции. Взамен этого предоставляется доступ к самостоятельным библиотекам, включающим все перечисленные функции и многое другое. Обращение к библиотечным функциям происходят столь часто, что эти функции можно считать составной частью языка. Но в то же время их можно легко переписать, без ущерба для структуры языка. Благодаря небольшому размеру исполняемых модулей программы, написанные на С/С++, отличаются высокой эффективностью и соизмеримы по скорости работы с ассемблерными программами. Большинство компиляторов С++ позволяет обращаться к подпрограммам, написанным на ассемблере.

С начала 90-х появились компиляторы С++ для персональных компьютеров, среди которых, в первую очередь, следует назвать Turbo С++, Borland С++ и Visual С++, С++ Builder. Компилятор языка прост, но он быстр и эффективен.

К слабым сторонам С++ можно отнести слабый контроль за типом данных и выход за границы массива.

К настоящему времени в программировании сформировано несколько направлений:

- процедурное программирование;
- структурное (модульное) программирование;
- объектно-ориентированное программирование.

Язык C++ поддерживает все эти направления.

В *процедурном* программировании основное внимание уделяется алгоритму, а именно его эффективности и компактности. Методы процедурного программирования особенно были важны, когда компьютеры не обладали достаточными быстродействием и объемом памяти. Но нельзя сказать, что они утратили свою актуальность сегодня.

В *структурном* программировании основное внимание уделяется организации данных. Программы делятся на модули таким образом, чтобы данные внутри модулей были скрыты. Применение методов структурного программирования позволяет создавать сложные программные продукты коллективам программистов. Поскольку каждый модуль может быть разработан, скомпилирован и отлажен отдельно. Методы структурного программирования позволяют создавать программы, удовлетворяющие критерию надежности и простые в сопровождении.

В действительности перечисленные направления не исключают, а дополняют друг друга.

Объектно-ориентированное программирование в большей степени, чем структурное программирование, предоставляет возможность создавать программы, обладающие структурированностью, модульностью и абстракциями данных, является современным методом создания сложных программ, в которых недостаточно использования методов структурного программирования.

Состав языка

Любой естественный язык содержит четыре основных элемента: символы, слова, словосочетания, предложения. Алгоритмический язык также включает символы, на основании которых строятся элементарные конструкции (*лексемы*). Из лексем и символов строятся *выражения*, которые в свою очередь образуют *операторы.*

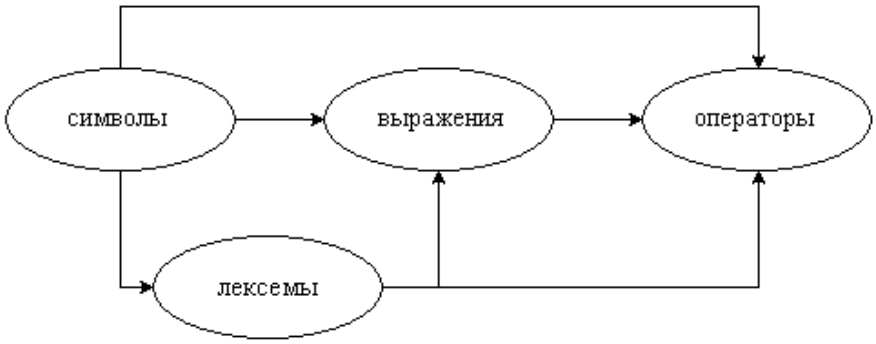


Рисунок 2. Состав алгоритмического языка

Алфавит языка включает в себя основные неделимые знаки, с помощью которых пишутся все тексты программ. Лексема является минимальной единицей языка, имеющей самостоятельный смысл. *Выражение* задает правило вычисления некоторого значения. *Оператор* представляет собой законченное описание некоторого действия. Любое выражение, заканчивающееся точкой с запятой, является оператором, выполнение которого заключается в вычислении выражения.

Операторы бывают исполняемые и неисполняемые. Первые задают действия над данными, а вторые служат для описания данных и называются операторами описания или просто описаниями.

Для описания сложного действия требуется последовательность операторов. Операторы могут объединяться в сложный оператор или блок.

Объединенная единым алгоритмом совокупность описаний и операторов образуют программу на алгоритмическом языке.

Алфавит языка

Алфавит C++ включает в себя:

- прописные и строчные буквы латинские буквы и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки: “, { } , | [ ] ( ) + - / % * . \ ‘ ^ ? < = > ! & # ~ $;
- пробельные символы: пробел, табуляция, символы перехода на новую строку.

Из символов алфавита формируются *лексемы* языка. Лексемы делятся на идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители.

Переменные, идентификаторы

Идентификаторы используются в C++ для именования различных объектов. *Идентификатор* – имя, связанное с данными или функцией программы, которое используется для обращения к этому объекту или функции.

Идентификатор представляет собой последовательность символов произвольной длины, содержащую буквы, цифры и символ подчеркивания, но начинающуюся обязательно с буквы или символа подчеркивания. Прописные и строчные буквы различаются. Пробелы внутри имен не допускаются.

Длина идентификатора по стандарту не ограничена, но некоторые компиляторы и компоновщики накладывают ограничения, например, распознают только первые 31 символ. В именах нельзя использовать термины, являющиеся частью языка C++.

Ключевые слова – зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Язык C++ содержит 63 ключевых слова.

Переменная – именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя (идентификатор) и значение. Имя служит для обращения к области памяти, в которой хранится переменная. Значение переменной может изменяться во время выполнения программы. Прежде чем использовать переменную, ее необходимо определить.

Данные, необходимые для работы программы, хранятся в памяти компьютера. Каждая область памяти имеет однозначно определенный адрес, на который ссылаются, когда необходимо сохранить или прочесть данные.

Идентификаторы используются для обозначения определенной области памяти, а компилятор транслирует имена в соответствующие адреса.

Имя переменной должно отражать смысл хранимой величины, быть легко распознаваемым и не содержать символов, которые можно перепутать друг с другом.

Каждый элемент данных должен принадлежать к определенному типу. Тип данных определяет, в каком виде они представлены в компьютере, а также какие преобразования компьютер может к ним применять.

Типы данных

Тип данных – множество допустимых значений данных и набор операций, применимых к этим значениям.

В C++ определены наиболее часто используемые типы данных. Кроме того, программист может сам определять новые типы.

Для описания стандартных (встроенных) типов в C++ используется набор ключевых слов: int, short, long, signed, unsigned, char, flout, double.

Первые шесть используются для представления целых данных разной длины (по количеству занимаемых битов в памяти компьютера). Они могут появляться в программе по отдельности или в некоторых сочетаниях.

int обозначает основной целый тип, которому соответствует стандартная длина слова, принятая на используемой машине. (На IBM – 16 битов). Диапазон значений, как правило,

зависит от системы. Для многих персональных компьютеров значение типа int меняется от -32768 до +32767.

long или long int может содержать целое значение, не меньшее максимальной величины, допускаемой типом int, или даже больше чем short или short int : максимальное целое число short не больше чем максимальное число типа int, а может и меньше. Обычно числа типа long бывают больше типа short, а тип int реализуется как один из указанных типов, все зависит от конкретной системы. (В IBM для short отводится 16 бит, а для long - 32 бита).

Все эти типы имеют 2 формы: знаковую (signed) и незнаковую (unsigned).

Целые незнаковые константы записываются также как и обычные целые, с тем исключением, что использование знака запрещено. Просто unsigned соответствует unsigned int.

Необходимо помнить, что в С++ число, начинающееся с нуля, является восьмеричным, а не десятичным.

char – самое короткое целое. Значения символьного типа занимают только 1 байт. Наиболее часто этот тип применяется для описания данных, состоящих из отдельного алфавитно-цифрового символа. Их называют символьные переменные. Например, ‘a’, ‘1’, ’+’, ’?’,’з’.

flout и double – числа с плавающей запятой или вещественные, которые могут принимать как положительные так и отрицательные значения. Такие числа имеют целую и дробную части, разделенные точкой. Например, 7.9, 3490.725.

Встроенные типы данных языка С++ подразделят на простые, структурированные и адресные (рисунок 3).

Структурированные и адресные типы данных, а также перечисляемый тип, описываемый с использованием ключевого слова enum, будут рассмотрены позднее.

Кроме перечисленных, к основным типам данных относится тип void, но множество его значений пусто. Он используется для определения функций, которые не возвращают значения и пустых указателей.

Объявить переменную означает задать ее имя и тип. Объявление сообщает компилятору, что данный идентификатор связывается с областью памяти, содержимое которого имеет определенный тип.

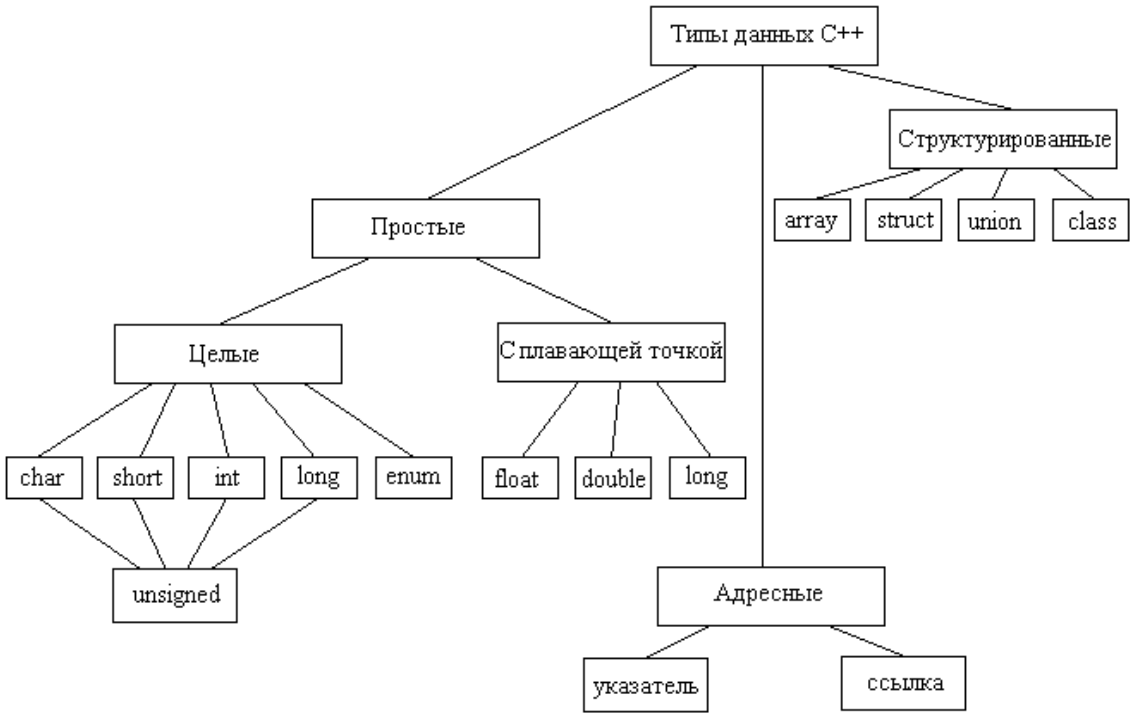


Рисунок 3. Типы данных С++

Синтаксис объявления переменной следующий:

<имя типа> <идентификатор переменной>;

Например,

```
int k;
```

Объявлена переменная с именем k целого типа. Объявление всегда заканчивается точкой с запятой. Таким образом, переменная k может содержать только целое значение. Если компилятор C++ встретит оператор, в котором переменной k будет присваиваться вещественное значение, то он произведет дополнительные действия для преобразования вещественного типа в целое.

Существует возможность объявить сразу несколько переменных одного типа в одном выражении. Для этого имена переменных перечисляются через запятую. Например,

```
int Number, Count;
float cost1, cost2;
unsigned positive;
```

Определяя переменную можно задать также и ее начальное значение, то есть инициализировать переменную. Инициализатор можно записывать в двух формах – со знаком равенства (=) или в круглых скобках.

Пример:

```
int d, c=5, r=4;
short b(8);
int k, l(145),  m;
unsigned s=0.5;
```

Таким образом можно собрать в один оператор описания переменные одного и того же типа, или наоборот, разбить одно описание на несколько операторов – эффект будет одинаков.

При инициализации символьных переменных их значение требуется заключать в апострофы.

```
char ch='z',  f;
```

Операции и выражения

Знак операции это один или более символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов.

Значение переменной можно изменить с помощью операции присваивания =. Например,

```
int summa = 0;
summa = 5;
```

В отличие от алгебраического уравнения в операторе присваивания сначала вычисляется выражение в правой части оператора, а затем оно присваивается отдельной переменной, стоящей слева от знака равенства. Например,

```
int k = 5, m = 1;
m = k;
```

Это означает, что значение переменной m стало равно 5, а не то, что m равно k. Кроме того, выражения m = k и k = m обозначают различные действия. В первом случае обе переменные станут равными пяти, а во втором – единице.

В C++ допускается использование нескольких присваиваний в одном выражении:

```
a = в = c = 0;
```

В любой программе требуется производить некоторые вычисления. Для вычисления значений используются выражения, которые состоят из операндов, знаков операций и скобок. Операнды задают данные для вычислений. Операции задают действия, которые необходимо выполнить.

Основные арифметические операции языка C++ обозначаются стандартными матема-

тическими операциями: сложение +, вычитание −, умножение *, деление /. Эти операции, как и операция присваивания, являются бинарными, так как для каждой из них требуется по два аргумента.

Унарный минус используется для определения отрицательных значений, унарный плюс практически не используется, потому что число без знака считается положительным по определению.

Особое внимание требует деление целых чисел. Необходимо помнить, что при делении целых чисел результат операции также целочисленный. Дробная часть просто отбрасывается. Для получения остатка от деления используют одноименную операцию %.

Так как 8 : 2 равно 4, то, 8 / 2 равно 4, 8 % 2 равно 0; 7 : 2 равно 3 (и 1 в остатке), поэтому 7 / 2 равно 3, а 7 % 2 равно 1.

При делении вещественных чисел получается вещественный результат.

Поскольку выражения могут содержать и переменные, допустимо использовать операции присваивания в следующем виде:

```
a = c + 5;
e = a / 2 - 11;
```

После каждого оператора обязательно ставится точка с запятой. Одна и та же переменная может встречаться с обеих сторон знака присваивания.

```
n=n+ 7;
```

Это означает, что сначала складываются значение, содержащееся в переменной с именем n и семь, а затем полученное значение помещается в переменную n, тем самым, заменяя ее предыдущее значение.

Кроме стандартных арифметических операций в С++ введен ряд специальных операций. Из них наиболее часто используемыми являются операция инкремента (увеличение на единицу) ++, и операция декремента (уменьшение на единицу) - - . Использование операция инкремента и дала название языку – С++.

Эти операции унарные (операции с одной переменной). Они могут использоваться с целым и вещественным аргументом. Использование оператора

```
j++ ;
```

эквивалентно оператору

```
j=j+1;
```

А соответственно j- - эквивалентно j = j – 1.

У этих операций существует особенность: они имеют две формы: префиксную, когда ее можно поместить перед переменной и постфиксную – после переменной.

Записанные в таком виде j+ +; + + j; эти операторы эквивалентны – каждый из них увеличивает значение j на единицу. Поэтому в данном случае выбор одной из форм оператора является делом вкуса. Однако С++ позволяет их использование в середине сложных выражений, и тогда их использование может привести к различным результатам.

Примеры:

```
int bar = 1;
cout << ++ bar;
int bar = 1;
cout << bar ++;
```

В первом случае ++bar увеличивает значение bar на единицу, а затем записывает это значение собственно в bar, то bar будет равен 2, и значение 2 будет выведено на экране.

bar ++ определяет значение bar, а потом выполняет приращение, следовательно, устанавливает bar 2, но выводит на экран 1, поскольку вывод происходит перед выполнением приращение.

В С++ определены операции составного присваивания. Они используются для сокращения записи операторов, содержащих в себе присваивание и арифметическую операцию. Оператор вида <операнд1> += <операнд2> эквивалентен записи <операнд1> = <операнд1> + <операнд2>. Существует составное присваивание со сложением, вычитанием, де-

лением, умножением, с остатком от деления: + = ; - = ; * = , / =, % =.

Пример

```
foo + = 3;        // эквивалентно
foo = foo + 3;
```

C++ содержит операции отношения: больше > , больше или равно > = , меньше < , меньше или равно< = , равно = = , не равно! = , не !.

Также определены логические операции: И && и ИЛИ ||. Результатом логической операции является true (истина) или false (ложь). Результатом операции логическое И является значение true, если оба операнда имеют значение true. Результат логического ИЛИ true, если хотя бы один из операндов имеет значение true. Логические операции выполняются слева направо. Например, результат выражения (k>=1)&&(k<10) будет true, если k=5. И это же выражение есть false, если k=0.

Операция определения размера sizeof( ) предназначена для вычисления размера объекта или типа в байтах, и имеет 2 формы: sizeof (выражение) или sizeof(тип). Так результат выполнения sizeof(int) равен 2, так как для хранения данных этого типа в памяти выделяется 2 байта.

C++ имеет одну тернарную операцию. Это условная операция (? :). Ее формат:

<условие> ? <выражение1> : <выражение2>;

Данный оператор позволяет создавать простые условные однострочные выражения, в которых выполняется одно из двух действий в зависимости от значения условия. Данный оператор можно использовать вместо инструкции if/else. Рассмотрим пример, в котором определяется модуль числа с помощью условного оператора

```
fvalue = (fvalue>=0.0) ? fvalue : -fvalue;
```

Рассмотрим другой пример применения условной операции. Требуется, чтобы некоторая величина увеличилась на 1, если она не превышает n, иначе принимала бы значение 1:

```
y = (y < n) ? y+1: 1;
```

Операции выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки.

Операции	***Приоритет операций***
++ – – ! sizeof	наивысший приоритет
* / %	
+ –	
< <= > >=	
= = !=	
&&	
||	
?:	
= += –=	наименьший приоритет

Рисунок 4. Приоритеты операций

Выражения состоят из операндов, знаков операций и скобок и используются для вычисления некоторого значения определенного типа. Каждый операнд является, в свою очередь, выражением или одним из его частных случаев – константой или переменной.

Примеры выражений:

```
(d + 8) / 67
x && y || !z
( t + 5*k) / ( f – 56 ) + 470
```

Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операция присваивания выполняются *справа налево*,

остальные – *слева направо*. Например, a=c=y означает a=(c=y), а a+b+c означает (a+b)+c.

Ввод – вывод на экран. Введение в потоки ввода – вывода

Оператор cout (си-аут) позволяет осуществлять вывод данных на экран монитора. Переменная cout зарезервирована для обозначения выходного потока.

Для того чтобы послать значение на си-аут применяют последовательность cout<< (оператор «направить в» или оператор вставки).

Если необходимо напечатать строку символов, требуется взять ее в кавычки. Можно также напечатать несколько значений одновременно, разделяя их оператором вставки.

```
cout << “My name is” <<”Tatyana”;
```

При печати цифр их можно не помещать в кавычки. Возможно объединение текста и цифр.

```
cout <<  “ мой адрес: Институтская“ << 26;
```

В процессе печати можно использовать довольно большое количество специальных символов. Вот некоторые из них:

\n	Начало новой строки
\t	табулятор
\b	возврат назад на один пробел
\f	начало новой строки страницы
\\	печать символа обратный слэш
\’	печать символа’
\”	печать символа ’’

Пример:

```
cout << “ He said:\ ” Hello:\” \ n”;
```

Оператор вставки использует два аргумента. Аргумент слева от << является потоковым выражением (потоковой переменной). Правый аргумент представляет собой строку или выражение, результат которого имеет простой тип. Оператор вставки преобразует правый операнд в последовательность символов и добавляет их выходной поток. Например,

```
cout<<”Результат равен  ” << 5*n + 90;
```

Если n равно 10, то на экране появится:

Результат равен 140

Для перехода на новую строку используют манипулятор endl, который также очищает буфер потока. Например,

```
Int x=17, y=21;
cout << “x= ”<<x<<endl<<”y=”<<y;
```

На экране появится

x=17

y=21

Стандартная библиотека C++ предоставляет пользователю большое количество манипуляторов, позволяющих форматировать ввод-вывод. Манипулятор setw (сокращение от «set width» – «установить ширину») позволяет управлять количеством позиций для вывода следующего за манипулятором элемента данных. Применяется только для форматирования чисел и строк, но не данных типа char. Параметр данного манипулятора – целое выражение, определенное число знаковых позиций для вывода очередного элемента. Данные при выводе выравниваются по правому краю, а свободные позиции слева заполняются пробелами. Например,

```
int ans=33, num=7132;
cout<<setw(4)<<ans<<setw(5)<<num;
```

выведет на экран □□33□7132

```
cout<<setw(1)<<ans<<setw(6)<<num;
```

33□□7132 - поле автоматически расширяется, чтобы вместить двузначное число.

Установка ширины поля является одноразовым действием и влияет только на бли-

жайший элемент вывода.

Контроль числа десятичных позиций при выводе решается с помощью манипулятора setprecision, указывающего количество знаков после запятой. Например,

```
int x=4.856;
cout<<setw(6)<<setprecision(2)<<x;
```

вывод □□□4.85

Для ввода с клавиатуры используют символ cin (си-ин или син) и >> оператор ,,взять из“

```
cin >> Number;
```

Стандартные функции ввода-вывода языка С, также доступны и в С++. Наиболее часто используемыми функциями ввода-вывода языка С, являются printf( ) и scanf( ), которые обеспечивают форматный ввод-вывод и являются достаточно универсальными, особенно при работе с числами, но из-за обилия всевозможных спецификаторов форматирования, становятся иногда громоздкими и трудно читаемыми. Операторы << и >> благодаря понятию перегрузки операторов поддерживают все стандартные типы языка С++, включая классы.

Директива #include

Язык С++ содержит очень небольшое число встроенных функций, но он легко расширяется дополнительными библиотеками.

Операторы cin и cout также являются частью библиотеки. Для их использования необходимо включить в программу соответствующие заголовочные файлы. Это делается с помощью команды #include.

Любая команда, начинающая с решетки называется директивой препроцессора. Она не является выражением языка С++ (поэтому не заканчивается точкой с запятой). Директивы препроцессора могут состоять только из одной строки.

Препроцессор – это программа, действующая как фильтр на этапе компиляции. Так препроцессорная директива #include приказывает компилятору загрузить включаемый файл.

Описания операторов cin и cout находятся в файле с именем iostream.h (input output stream – поток ввода-вывода, h - стандартное расширение заголовочного файла (сокращение от header file).

Синтаксис: имя файла помещается в угловые скобки < >, что указывает препроцессору, что этот файл ищется в стандартном каталоге подключаемых файлов:

```
# include < iostream.h>
```

Компилятор знает, где искать стандартные заголовочные файлы. Если же требуется загрузить заголовочный файл, созданный пользователем, его имя необходимо заключить в кавычки.

```
# include “ myfile.h”
```

Можно также указать полный путь

```
# include “ c \ My \ myfile.h “
```

Для использования манипуляторов при выводе данных также необходимо подключить библиотеку – iomanip.h:

```
# include <iomanip.h>
```

Построение программы

При создании алгоритма решения сложной задачи возможно разбиение решения на решение более простых подзадач. Подпрограммы позволяют сначала записать части программы по отдельности, а затем собрать их в единое работоспособное целое. В языке С++ подпрограммы называются *функциями*, а программа представляет собой набор из одной или более функций. Каждая функция выполняет определенное действие, а все вместе они решают задачу в целом.

Итак, любая программа, написанная на языке С++ есть последовательность выполнения функций, причем одна из них обязательно должна называться main( ). Выполнение про-

граммы всегда начинается с выполнения ряда операторов этой функции. Когда main () хочет, чтобы другая функция выполнила свое задание, она вызывает (активизирует) необходимую функцию.

Все функции в языке С++ равноправны: каждая из них (даже main( )) может быть любой другой функцией. Функция может вызывать саму себя (явление рекурсии). Компилятор не ограничивает число рекурсивных вызовов, но операционная система может наложить чисто практические ограничения.

Описание функции состоит из заголовка и тела, и в общем случае, выглядит следующим образом:

Директива препроцессора #include <iostream.h>

Заголовок <тип результата> имя функции (список аргументов)

Тело функции {

Описание *пять типов операторов*

Присваивание

Функция

Управление

Пустой оператор

}

Предполагается, что любая часть описание функции может не использоваться, кроме имени функции. Пример простейшей программы:

```
# include < iostream.h>
main ( )
{
 cout << “HeLLo!!! ”;
}
```

Данная программа просто выведет на экран монитора HeLLo!!!

Фигурные скобки отмечают начало и конец тела функции. В круглых скобках в общем случае содержится информация, передаваемая этой функции.

Если тип результата не указывается, то предполагается, что функция возвращает значение типа int. Если функция не возвращает результат, указывается слово void. Некоторые компиляторы требуют обязательного указания типа возвращаемого результата перед именем функции.

Наличие списка аргументов и описаний аргументов также не является обязательным. Но круглые скобки всегда должны присутствовать после имени функции. Аргументы функции – это величины, которые необходимо передать из вызывающей функции в вызываемую функцию.

Часто употребляется описание типов аргументов сразу при их объявлении внутри круглых скобок. Тип аргумента может быть любым. Если аргументов несколько, их описания (тип плюс имя) разделяются запятыми. Если функции не передаются величины, то вместо списка аргументов необходимо задавать ключевое слово void. Локальные переменные отличные от аргументов, описываются внутри тела функции.

Возвращает значение оператор return, с помощью которого можно передать в вызывающую функцию только одно значение. Возвращаемое значение можно брать в круглые скобки после ключевого слова return. Круглые скобки не являются обязательными.

Пример

Напишем программу, содержащую три функции: непосредственно main и функции, вычисляющие значения куба и квадрата некоторого целого числа.

```
#include <iostream.h>
int square (int x)
        { return x*x; }
int cube (int y)
```

```
        { return y*y*y; }
  void main ( )
  {     cout<< “Квадрат числа 15 равен”<<square(15)<<endl;
        cout<< “Куб числа 10 равен”<<square(10)<<endl;
  }
```

В каждой из трех функций левая и правая фигурные скобки указывают на начало и окончание тела функции, т.е. начало и окончание исполняемых выражений. Их называют операторные скобки.

В первой строке расположена директива препроцессора, подключающая заголовочный файл iostream.h, необходимый для работы оператора cout.

Далее следует заголовок и тело функции square. Функция имеет один аргумент целого типа, описание которого расположено в круглых скобках после имени функции. Функция square возвращает целочисленный результат в вызываемую функцию, на это указывает ключевое слово int, стоящее перед именем функции. Возвращение результата из функции осуществляет оператор return. Результатом является выражение после указанного оператора. После оператора обязательно ставится точка с запятой (как требует синтаксис языка С++).

Затем в программе расположено описание функции cube, аналогичное предыдущей функции.

Ниже идет описание функции main. Перед именем функции стоит ключевое слово void, сообщающее компилятору, что функция не возвращает значений. Функция не имеет аргументов – в ее заголовке пустые круглые скобки. В скобках также можно указать void,что также будет указывать на отсутствие аргументов. Выполнение любой программы всегда начинается с первого оператора функции main, имеющего в данной программе вид:

cout<< “квадрат числа 15 равен”<<square(15)<<endl;

В этом операторе происходит вызов функции square для аргумента 15, следовательно, выполняется оператор данной функции, т.е. 15 умножается на 15 и результат возвращается в main и при помощи cout выводится на экран в виде:

Квадрат числа 15 равен 225

После этого main продолжает выполнение своих операторов, напечатав сообщение

Куб числа 10 равен 100

На этом программа завершает свое выполнение.

Любая функции в С++ может возвращать не более одного значения, т.е. либо не возвращать ничего, либо возвращать единственное значение. Аргументов же у функции может быть сколько угодно, в том числе ни одного.

Пример. Опишем функцию, возвращающую среднее значение трех величин.

```
// определение функции Average
long Average (long val1, long val2, long val3 )
{
long sum = val 1 + val 2 + val 3;
return sum / 3;
}
```

Такая функция может быть вызвана любой другой функцией. Пусть в программе объявлены некоторые переменные: long a, a1, a2, a3; тогда возможен вызов вида

```
a = Average (a1, a2, a3);
```

или

```
a = Average (525, 675, 819);
```

В случаях, когда вызываемая функция не возвращает значение, ее вызов представляет просто имя функции, а в круглых скобках – фактические значения аргументов функции.

Константы

Иногда требуется, чтобы значение переменной оставалось постоянным в течении всего времени работы программы. Такие переменные называются *константными.*

Для их создания необходимо написать определение для переменной с добавлением ключевого слова const перед типом

```
const int Top = 12;
```

Константные переменные используются в программе также как обычные. Единственное отличие заключается в том, что начальные значения, присвоенные константам при их инициализации, не могут быть изменены в ходе выполнения программы.

В C++ существует и другой способ описания констант, доставшийся в наследство от языка C: с помощью макроопределений использующих директиву препроцессора #define. Например,

```
#define TOP 12
//объявлена константа с именем TOP, равная 12
```

(Точка с запятой после директивы препроцессора не ставится.)

Каждая директива #define позволяет определить одну константу, имя которой следует за директивой. Принято писать имя константы заглавными буквами, что не является требованием компилятора. В отличие от предыдущего способа тип такой константы неизвестен. Процессор при трансляции программы просто заменяет имя константы на определенной с помощью директивы значение.

Данная директива в языке C использовалась также для определения макросов (макроопределение с аргументом). Макросы являются просто текстовыми подстановками и могут не давать компилятору достаточной информации в желательном представлении данной величины. Часто встречаемой ошибкой была, например, такая

```
#define SUMMA(a,b)   a+ b
double result, x=5.2, y=0.8,
result=SUMMA(x,y)*10;
```

После работы препроцессора получим подстановку вида:

```
result=x+y*10;
```

Для корректной подстановки рассмотренная директива должна выглядеть следующим образом:

```
#define SUMMA(a,b) ((a)+(b))
```

В C++ для определения встраиваемой функции используется ключевое слово inline. Определение аналогичной функции в программе на C++ будет выглядеть так:

```
inline double SUMMA(double a, double b)
{ return (a+b); }
```

При определении и использовании встраиваемой функции надо придерживаться следующих правил:

- определение и объявление функции должны располагаться перед первым ее вызовом;
- имеет смысл определять таким способом только маленькие функции;
- компилятор сам решает, является ли данная функция встраиваемой, при этом он руководствуется размером (до 1200 строк), если функция рекурсивна, то встраиваемой является только первый вызов, некоторые компиляторы не допускают использования в встраиваемых функциях операторы цикла и некоторые другие.

Комментарии

Комментариями называются некомпилируемые фрагменты программы. Комментарий используется для пояснения алгоритма или текста программы. Комментарий либо начинается с двух символов «косая черта» («слэш») // и заканчивается переходом на новую строку, либо заключается между символами-скобками /* и */. Внутри комментария можно использовать любые допустимые на компьютере символы, а не только символы алфавита языка C++, так как компилятор комментарии игнорирует.

Библиотечные функции

Некоторые вычисления, такие как извлечение квадратного корня или нахождение модуля, часто используются в программах. Для удобства программиста любой программный комплекс C++ содержит *стандартную библиотеку* (или *библиотеку стандартных функций*) – собрание ранее написанных функций, выполняющих стандартные вычисления.

Библиотека математических функций содержит:

Имя функции	*Параметр*	*Возвращаемое значение*
acos(x)	$-1.0 \le x \le 1.0$	Арккосинус x, в диапазоне от 0.0 до π
asin(x)	$-1.0 \le x \le 1.0$	Арккосинус x, в диапазоне от $-\pi/2$ до $\pi/2$
atan(x)	x	Арктангенс x, в диапазоне от $-\pi/2$ до $\pi/2$
ceil(x)	x	Верхнее значение x (наименьшее целое число $\ge$ x)
cos(x)	x, выраженный в радианах	Тригонометрический косинус x
exp(x)	x	Значение e (2.718…), возведенное в степень x
fabs(x)	x	Модуль x
floor (x)	x	Нижнее значение x (наибольшее целое число $\le$ x)
log(x)	$x > 0.0$	Натуральный логарифм от x
log10(x)	$x > 0.0$	Десятичный логарифм от x
pow(x, y)	если x=0, y>0; иначе y - целое	Значение x, возведенное в степень y
sin(x)	x, выраженный в радианах	Тригонометрический синус x
sqrt(x)	x>0.0	Корень квадратный из x
tan(x)	x, выраженный в радианах	

Параметрами и результатами перечисленных математических функций являются переменные типа float.

Использовать библиотечную функцию несложно. Требуется разместить в начале программы директиву #include с указанием требуемого файла заголовков math.h. Затем можно обращаться к функции в любом месте программы.

Пример:

```
#include <iostream.h>
#include <math.h>
void main ( )
{       float alpfa, beta;
        alpfa = sqrt(169.41 + fabs( pow( beta, 5));
        cout<< alpfa; }
```

Глава 3. МЕТОДЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Блоки или составные выражения

Любое выражение, оканчивающееся точкой с запятой, является оператором. Оператор также может ничего не содержать – так называемый пустой оператор. Пустой оператор обозначается просто точкой с запятой (так как после каждого оператора, том числе и пустого, по требованию синтаксиса языка C++ обязательно ставится точка с запятой). Любой оператор может представлять собой объявление, выполняемый оператор или блок.

Тело функции является примером блока или составного оператора. Блок является последовательностью из нуля или более операторов. Эта последовательность заключена в фигурные скобки. После окончания блока точка с запятой не ставится.

Блок можно использовать везде, где разрешено использование отдельного оператора.

Блоки часто используются в программах, особенно как составная часть других выражений. Пропуск пары фигурных скобок может кардинально поменять смысл алгоритма программы, повлиять на ее выполнение и полученный результат.

В программе операторы блока обычно располагаются с некоторым отступом, что не является требованием компилятора. Но это позволяет выделять блок и контролировать расположение скобок. Это повышает читаемость программы, и следовательно, контроль ошибок.

Внутри блока разрешается чередовать объявления и операторы, т.к. объявления в программе С++ могут располагаться в любом месте. Необходимо только помнить, что все переменные должны быть объявлены перед использованием.

Операторы ветвления

Условный оператор if

Условный оператор if позволяет разветвлять вычислительный процесс на два направления. Структурная схема оператора представлена на рисунке.

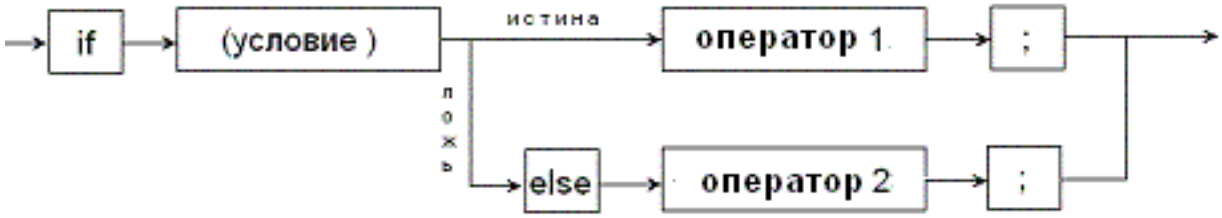


Рисунок 6. Структурная схема условного оператора

Сначала вычисляется выражение, стоящее в условие. Если оно не равно нулю или имеет значение истина, выполняется первый оператор, иначе второй. После этого управление передается следующему оператору. Таким образом, с помощью оператора if можно в ходе выполнения программы задать некоторый вопрос и в зависимости от ответа (да или нет) выполнить те или иные действия.

Синтаксис оператора if:

if (<выражение>) <оператор1>; [else <оператор 2>;]

Ветвь с ключевым словом else не является обязательной. Поэтому она взята в квадратные скобки. (В дальнейшем изложении для обозначения необязательных элементов будут всегда использоваться квадратные скобки.)

Примеры:

```
if ( fvalue>=0.0 )  fvalue = fvalue;  else fvalue = -fvalue;   // вычисляется модуль числа
if ( x<10 ) x+ =10; else x *=2;
if ( f != 0 )  c = 100 / f;
```

Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок (операторные скобки { }), иначе компилятор не сможет определить окончание ветвления.

```
if ( a ) { a++;  v=60*a; }  else v = a;
if ( e==1000 ) { e /=10;  cout << ”e= ” << e; }  else { e =10+y*y;  y++; }
```

При использовании блока в условном операторе точка с запятой после правой фигурной скобки блока не ставится. Точка с запятой применяется для завершения простых операторов.

Блок может содержать любые операторы, в том числе и другие условные.

```
if ( a<b ) { if ( a<c ) m = a;  else m = c; }  else { if ( b<c ) m = b;  else m = c; }
```

Необходимо помнить, что в этом случае else относится к ближайшему из if. Операторные скобки после первого if необязательны, т.к. в него вложен простой оператор if.

Если требуется проверить несколько условий, их объединяют знакам логических операций.

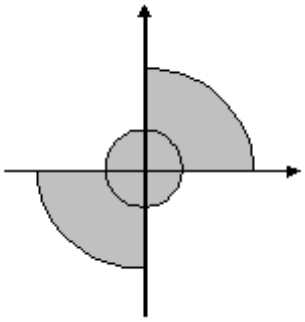
```
 if ( a<b && ( a>d || a==0 ) )  b++; else { b*=a; a=0; }
```

Записанное условие будет истинно в том случае, если выполнится одновременно условие a<b и одно из условий в скобках. Если опустить внутренние скобки, будет выполнено

сначала логическое И, а потом ИЛИ.

Распространенной ошибкой является неверная запись условия проверки переменной на принадлежность диапазону. Например, чтобы проверить условие 0<x<10 нельзя записать его в условном операторе непосредственно, следует писать if (0< x && x>10).

Пример программы. Производится выстрел по мишени, изображенной на рисунке. Радиус внутреннего круга равен единице, а внешнего – тройке. Попадание в меньший круг дает 10 очков, в сегменты большого – 5 очков. Определить количество очков, набранного выстрелом, координаты которого вводятся с клавиатуры.



```
#include <iostream.h>
void main ( )
{
float x, y;
 int score;
 cout << “введите координаты выстрела”<<endl;
 cin >>x>>y;
 if  ( x*x + y*y < 1 ) score = 10;
        else     if  ( x*x + y*y < 9 &&  x*y>0 ) score = 5;
                 else kol = 0;
 cout << “Вы набрали” << kol << “  очков!!!”;
}
```

Выбор из множества альтернативных действий может быть запрограммирован с помощью множества условных операторов. Например, чтобы запрограммировать печать названия месяца по его заданному порядковому номеру, допустимо использовать последовательность условных операторов без вложения:

```
if (m = =1) cout << “Январь”;
if (m = =2) cout << “Февраль”;
if (m = =3) cout << “Март”;
…
if (m = =12) cout << “Декабрь”;
```

Однако эквивалентная вложенная структура более эффективна, поскольку требует меньшее количество сравнений:

```
if (m = =1) cout << “Январь”;
        else     if (m = =2) cout << “Февраль”;
                 else     if (m = =3) cout << “Март”;
                 …
                          else     if (m = =12) cout << “Декабрь”;
```

В первом случае последовательно проверяется все двенадцать условий, а во втором – все сравнения прекращаются после того, как истинное условие обнаружено.

Распространенной ошибкой при записи условного оператора является использование операции присваивания (=) вместо проверки на равенство (= =).

Оператор выбора switch

Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Синтаксис оператора switch:

```
switch ( выражение) {
case <константное выражение 1> : <операторы1>; [ break; ]
case <константное выражение 2> : <операторы2>; [ break; ]
…
case <константное выражение n> : <операторы n>; [ break; ]
[default : операторы];
```

}

Структурная схема оператора приведена на рисунке 7.

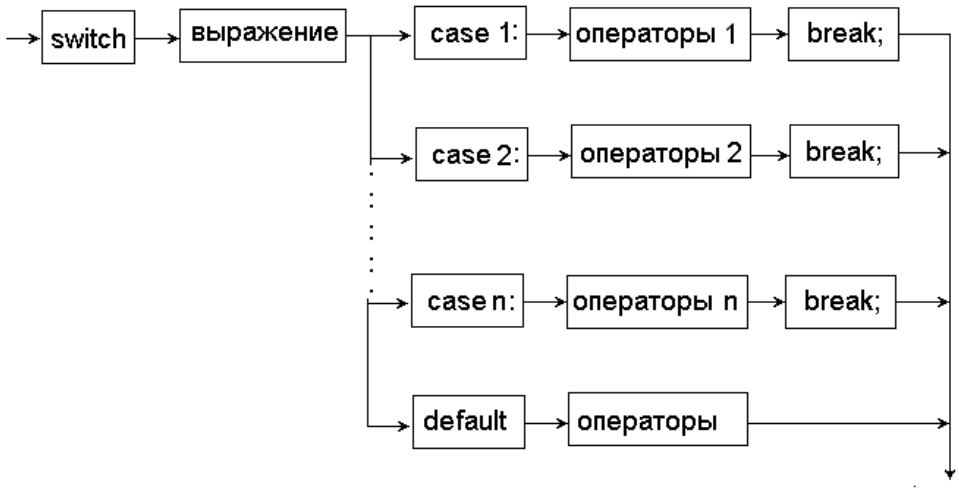


Рисунок 7. Структурная схема оператора switch

Выполнение оператора начинается с вычисления выражения, которое должно быть целочисленным. Затем управление передается первому оператору из списка, помеченному константным выражением, значение которого совпало с вычисленным.

После этого, если выход из переключателя явно не указан (отсутствует break), последовательно выполняются все нижележащие ветви.

Выход из переключателя обычно выполняется с помощью оператора break или return. Оператор break выполняет выход из самого внутреннего из объемлющих его операторов. А оператор return выполняет выход из функции, в которой он описан.

Все константные выражения, расположенные после case, должны быть различны. Если совпадения ни с одним оператором не произошло, выполняются операторы, расположенные после ключевого слова default.

Ветвь default может отсутствовать. В этом случае выполнение программы передается следующему за switch оператору.

Пример предыдущий раздела (печать названия месяца по порядковому номеру) может быть переписан с использованием switch следующим образом:

```
#include <iostream.h>
int main ( )
{ int x;
  cout << “Введите номер месяца” << endl;
  cin >> x;
  switch (x ) {
        case 1 : cout << “Январь”<< endl;              break;
        case 2 : cout << “Февраль”<< endl;    break;
        case 3 : cout << “Март”<< endl;       break;
…
        case 12 : cout << “Декабрь”<< endl;   break;
        default : cout << “Неверный номер” << endl;    }
return 0;
}
```

Несколько меток могут следовать подряд, предполагая выполнение одинаковых действий. Программа вывода времени года по номеру месяца:

```
#include <iostream.h>
void main ( )
{ int x;
```

```
 cout << “Введите номер месяца” << endl;
 cin >> x;
 switch (x ) {
        case 12 :  case 1 :        case 2 :cout << “зима”<< endl;  break;
        case 3 :   case 4 :        case 5 :cout << “весна”<< endl; break;
        case 6 :   case 7 :        case 8 :cout << “лето”<< endl;  break;
        case 9 :   case 10 :       case 11 :        cout << “осень”<< endl; break;
        default : cout << “Неверный номер” << endl; }
}
```

Инструкции перехода

В языке С++ имеется четыре инструкции перехода: goto, break, continue, return.

Оператор безусловного перехода goto имеет формат:

goto <метка>;

Тогда в теле той же функции должна присутствовать ровно одна конструкция вида:

<метка> : оператор;

Метка является обычным идентификатором. Оператор goto передает управление на помеченный оператор.

Наличие конструкции безусловного перехода рассматривается как плохой стиль программирования, и считается, что в четко структурированной и грамотно написанной программе этой конструкции быть не должно.

Использование данного оператора оправдано в случаях:

- принудительного выхода вниз по тексту программы из нескольких вложенных циклов или переключателей;

- перехода из нескольких операторов функции на один.

Инструкция break используется внутри циклов или переключателей и позволяет переход в точку программы, находящуюся непосредственно за оператором, внутри которого находится. Таким образом, break обеспечивает выод из цикла еще до того, как условие цикла станет ложным. По своему действию она напоминает команду goto, только в данном случае не указывается точный адрес перехода. Управление передается первой строке, следующей за телом оператора.

Инструкция continue заставляет программу пропустить все оставшиеся строки цикла, но сам цикл при этом не завершается. Для решения некоторых задач удобно комбинировать инструкции break и continue.

Иногда необходимо прервать выполнение программы задолго до того, как будут выполнены все ее строки. Для этого используют return, которая завершает выполнение той функции, в которой она была вызвана. Если же вызов произошел в функции main( ), то завершается сама программа.

Операторы цикла

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела цикла, т. е. операторов, которые повторяются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

Один повтор выполнения операторов тела цикла называется *итерацией*. Проверка условия выполнения цикла производится на каждой итерации.

Параметрами цикла называются переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения цикла, называются параметрами цикла.

Начальные установки могут явно не присутствовать в программе, их смысл состоит в том, чтобы до входа в цикл задать значения переменным, которые в нем используются.

Цикл завершается, если условие его выполнения не выполняется. Возможно принуди-

тельное завершение, как текущей итерации, так и тела цикла. Для этого используются конструкции перехода.

Для удобства в С++ существуют три разных оператора цикла – while, do while, for.

Цикл с предусловием (while)

В цикле с предусловием проверка условия продолжения цикла выполняется перед телом цикла (рисунок 8).

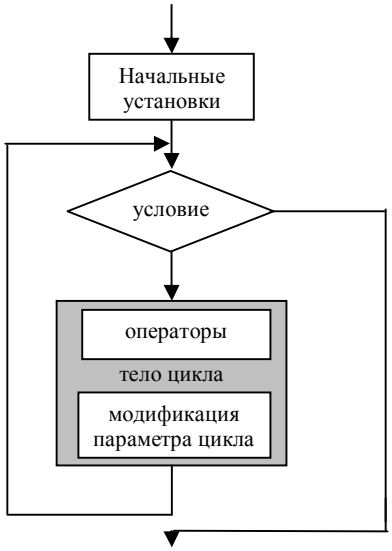


Рисунок 8. Цикл с предусловием

Цикл с предусловием реализован в С++ оператором цикла while, структурная схема которого представлена на рисунке 9.

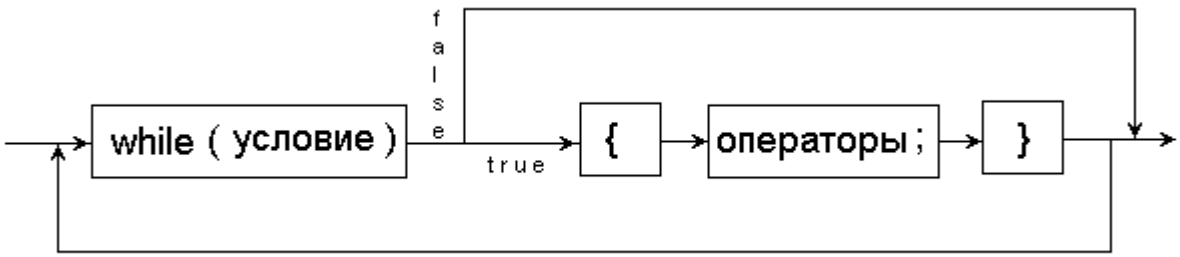


Рисунок 9. Структурная схема оператора цикла while

Выражение, стоящее в круглых скобках, определяет условие повторения тела цикла, представленного простым или составным оператором. Если оператор простой операторные скобки { } не ставятся.

Выполнение оператора цикла начинается с вычисления выражения, стоящего в условии. Если оно истинно, выполняется тело цикла. Если при первой проверке выражение ложно (false), цикл не выполнится ни разу.

Тип выражения должен быть арифметическим или приводимым к нему. Выражение вычисляется перед каждой итерацией цикла.

Пример. Программа печатает таблицу значений функции $y = x^3 - x$ с определенным шагом во вводимом диапазоне.

```
#include <iostream.h>
#include <iomanip.h>
void main ( )
{ float x1, xn, d;
```

```
  cout << " введите диапазон и шаг изменения x"<< endl;
  cin >> x1 >> xn >> d;
  cout << "|          x       |       y      |" << endl;    // шапка таблицы
  float x = x1;
  while ( x <=  xn )
        { cout << "|" << setw(6) << setprecision(3) << x << "|";
        cout << "|" << setw(6) << setprecision(3)<< x*x*x – x << "|" << endl;
        x+=d;
       }
}
```

Цикл while обычно используется в тех случаях, когда число повторений заранее неизвестно.

Цикл с постусловием (do while)

Тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз.

Цикл с постусловием реализован в С++ оператором цикла do … while и имеет вид:

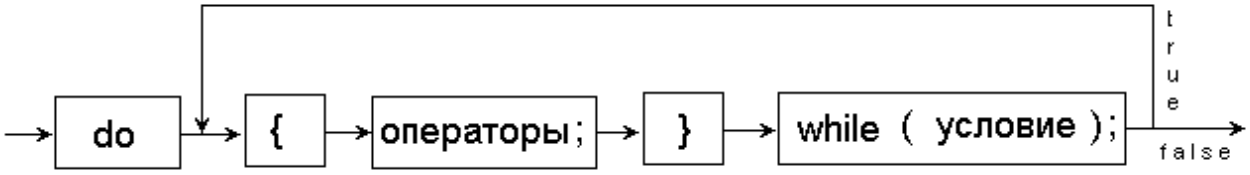


Рисунок 10. Структурная схема оператора цикла do while

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение, составляющее условие выполнение цикла. Даже если условие заведомо ложно, цикл выполнится один раз. Если условие истинно, тело цикло выполнится еще раз. Цикл завершается, когда выражение станет равным false, или в теле цикла будет выполнен оператор передачи управления.

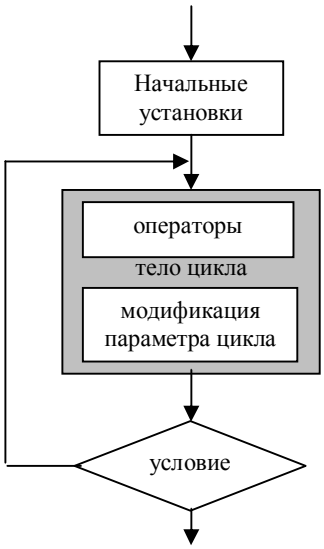


Рисунок 11. Цикл с постусловием

Пример. Программа угадывания загаданного числа.

```
#include <iostream.h>
#include <stdlib.h>
void main ( )
{  float x, y;
```

```
  randomize( );            // инициализация счетчика случайных чисел
  y = random (10);                  // выбор числа в диапазоне от 0 до 10
  do
  {  cout << " введите произвольное число меньшее 10" << endl;
     cin >> x;
     if ( x = = y) { cout <<" вы угадали!"; break; }
     else          if  ( x< y ) cout<< "введите меньшее"<<endl;
                   else      cout<< "введите большее"<<endl;
    }  while (x!=y);
}
```

В приведенном примере для задания числа используется генератор случайных чисел. Оператор randomize( ); инициализирует счетчик случайных чисел. Оператор y = random (10); присваивает переменной y случайно выбранное число из диапазона от 0 до 10. Для использования счетчика случайных чисел требуется подключить библиотеку stdlib.h.

Необходимо отметить, что счетчик случайных чисел генерирует только целые положительные значения из указанного диапазона.

Если требуется, чтобы переменная принимала случайным образом как положительные, так и отрицательные значения, можно воспользоваться конструкцией вида y = random (10) - 5;

Цикл с параметром (for)

Цикл for называют также циклом с заданным числом повторений. Он имеет следующий формат:

for (инициализация; выражение (условие) ; модификации) оператор;

Инициализация используется для объявления и присвоения начальных значений величинам, используемым в цикле. Инициализация выполняется один раз перед выполнением тела цикла.

Цикл с параметром реализуется как цикл предусловием. Выражение определяет условие выполнения цикла: если его результат равен истине, то цикл выполняется. Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметра цикла.

Тело цикла представляет собой простой или составной оператор.

Любое из трех выражений, указанных в скобках, является необязательным. Точки с запятой должны всегда оставаться на своих местах, даже в случае, если все три выражения отсутствуют.

Примеры

```
for ( ; ; );        // пример бесконечного цикла
for (y=2; y<20; y++)  r+=y;
```

В инициализации и модификации параметра можно писать несколько операторов, разделенных запятой.

```
for (i = 1, s=1; i<11; i++) s*=i;        // вычисления факториала 10
```

Цикл for обычно используется в тех случаях, когда можно точно определить необходимое число повторов.

Допускается объявление переменной прямо в строке инициализации цикла for. Значение счетчика цикла может не только увеличиваться, но и уменьшаться, причем на произвольный шаг, который может быть не только целым, но и числом с плавающей точкой.

```
for (float k = 11.5; k>0.5; k - = 0. 5) s+=k;
```

В качестве параметра цикла можно использовать и символьную переменную.

```
for (char ch='a'; ch< 'z'; ch++) …
```

Если тело цикла содержит более одной команды, следует использовать фигурные скобки и руководствоваться определенными правилами оформления, чтобы сделать текст более наглядным.

```
#include<iostream.h>
#include<math.h>
#include<iomanip.h>
void main( )                                        // табулирование функции
{ cout << "+----+-----------+ \ n" ;
  cout << "|  T  |       Y       | \n";
  cout<< "+-----+-----------+ \n";
  int n=10, t;
  float a=0.3, y;
  for ( t=1; t <=10;+ +t )                          // тело цикла
          {  if ( sin( ( t * t + 1 ) / n )>0 )   y=a*sin( ( t * t+1) / n );
             else  if (sin( ( t * t + 1 ) / n ) < 0 )  y=cos( t+ 1/n );
             cout<< "|  " << setw(2)<< t << " |  "<< setw(5) << setprecision(2) << y << "    | \n";
          }
 cout << "+ -----+ -----------+ \n";
}
```

Любой цикл while может быть приведен к эквивалентному ему циклу for и наоборот. Операторы цикла взаимозаменяемы, но можно привести рекомендации по выбору наилучшего в каждом конкретном случае.

Оператор do while обычно используют, когда цикл требует обязательно выполнить хотя бы раз (например, если в цикле производится ввод данных).

Оператором while удобнее пользоваться в случаях, когда число итераций заранее неизвестно, нет очевидных параметров цикла или модификацию параметров удобнее записывать не в конце тела цикла.

В большинстве остальных случаев предпочтительнее использование оператора for.

Часто встречающиеся ошибки при проектировании циклов - использование в теле цикла неинициализированных переменных или неверная запись условия выполнения цикла. Во избежание ошибок рекомендуется:

- проверить, всем ли переменным, встречающимся в правой части операторов присваивания в теле цикла, присвоены до этого начальные значения (а также возможно ли выполнение других операторов);

- проверить, изменяется ли в цикле хотя бы одна переменная, входящая в условие выхода из цикла;

- предусмотреть аварийный выход из цикла по достижению некоторого количества итераций;

- не забывать о том, что если в теле цикла требуется выполнить более одного оператора, их нужно заключать в фигурные скобки.

Если телом цикла является циклическая структура, то такие циклы называются вложенными или сложными.

При работе с вложенными циклами необходимо обращать внимание на правильную расстановку фигурных скобок, чтобы четко разделить границы циклов.

Цикл, содержащий в себе другой цикл, называется внешним. Цикл, содержащий в теле другого цикла, называется внутренним.

Внутренний и внешний циклы могут быть любыми из трех рассмотренных видов: циклом с параметрами, циклом с предусловием, циклом с постусловием. Правила организации как внешнего, так и внутреннего цикла, такие же, как и для простого цикла каждого из этих видов. Однако при построении вложенных циклов необходимо соблюдать следующее дополнительное условие: все операторы внутреннего цикла должны полностью лежать в теле внешнего цикла.

Параметры циклов разных уровней не изменяются одновременно. Сначала все свои значения изменит параметр цикла низшего уровня вложенности при фиксированных (начальных значениях) параметров циклов с высшим уровнем. Затем изменяется на один шаг значе-

ние параметра цикла следующего уровня, и снова полностью выполняется самый внутренний цикл, и т.д. до тех пор, пока параметры циклов не примут все требуемые значения.

Массивы

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой переменных одного типа требуется выполнить различные действия, им дают одно имя и отличают по порядковому номеру. Это позволяет записывать множество действий и операций над этими элементами через циклы.

Конечная именованная последовательность однотипных величин называется массивом.

Для создания массива компилятору необходимо знать тип данных, количество элементов в массиве и требуемый класс памяти. Массивы могут иметь те же типы данных, что и простые переменные.

Синтаксис объявления массива:

<Тип данных> <имя массива> [размерность массива];

Примеры:

```
int array[45];                  //массив из 45 элементов целого типа с именем array
double rt[12];                  // массив rt, состоящий из 12 элементов типа double
const int ARRAY_SIZE=90;
float alp[ARRAY_SIZE];          // вещественный массив alp из 90 элементов
```

Размерность массива предпочтительнее задавать с помощью именованных констант, как это сделано в примере. При таком подходе для изменения во всей программе достаточно изменить значение константы в ее объявлении.

Возможна также инициализация массива при его объявлении. Для этого значения элементов массива перечисляются через запятую в фигурных скобках после имени массива.

```
int a[5]={4, 90, 71, 45, 3};
```

Количество элементов должно соответствовать размеру массива. Если их окажется меньше, то недостающие элементы будут инициализированы нулями (или мусором, хранящимся в памяти). Если же элементов окажется больше – компилятор выдаст ошибку.

При инициализации допускается использование пустых скобок в объявлении массива, и тогда компилятор сам определяет размерность массива. Например,

```
float x[ ]= {4, 8, 19};       // размерность x равна 3
```

Для доступа к элементу массива после его имени указывается номер элемента (индекс), заключенный в квадратные скобки. Нумерация элементов массива начинается с нуля. Следовательно, диапазон значений для объявленного в примере массива x лежит в пределах от 0 до 2. Следовательно,

x[0]=4 x[1]=8 x[2]=19

В памяти компьютера элементы массива располагаются последовательно друг за другом. Место в памяти для хранения элементов массива выделяется компилятором сразу после обработки его объявления.

Так объявление вида

```
float y[5]= {0.7, 1.9, 8.4, 3.1 };
```

выделит место для расположения пяти элементов типа float. Так как выполнена инициализация четырех первых элементов, они сразу займут свое место в памяти компьютера, а ячейки памяти для пятого элемента пока останутся свободными:

0. 7	1. 9	8. 4	3. 1	
y[0]	y[1]	y[2]	y[3]	y[4]

В C++ не осуществляется проверка значений индексов на выход за пределы массива, ни в процессе компиляции, ни в процессе выполнения программы. Так если, в тексте программы напишем выражение

```
y[5]=2;
```
компьютер сохранит значение 2 в ячейке памяти, следующей за ячейкой, соответствующей последнему элементу массива. При этом старое значение данной ячейки будет затерто. Поэтому проверка значения индекса – обязанность программиста.

К элементам массива можно применять все операции, допустимые для обычной переменной типа, соответствующего типу элементов массива. Можно присваивать ему значения

```
y[0]=56;
```
применять арифметические операции
```
y[2]=7*y[0]+3;
```
считывать или выводить значения
```
cin >> y[0];       cout << y[1];
```
передавать его в качестве параметров функций
```
x = sqrt (y[1]);
```
Пример: Программа подсчета суммы элементов массива.
```
#include<iostream.h>
void main ( ) {
  const int n=10;
  int i, m[n], s=0;
  cout << " введите" << n << "чисел" <<endl;
                    // ввод элементов массива с клавиатуры
  for ( i=0; i < n; i++) cin >> m[ i];
  for ( i=0; i < n; i++) s+=m[ i ] ;
  cout << "сумма введенных элементов равна "<<s<<endl;
}
```
Для улучшения эффективности алгоритма ввод элементов массива и подсчет их суммы можно объединить в один цикл:
```
  for ( i=0; i<10; i++) { s+=m[i];    cin >> m[i]; }
```
Пример: Программа выводит на экран количество дней в месяце для невисокосного года.
```
include< iostream. h>
int days[ ]={31,28,31,30,31, 30, 31, 31, 30, 31, 30, 31};
void main ( )
{
for (int index=0; index<sizeof(days)/(sizeof(int)); index++)
          cout << "месяц" << index+1 << " имеет" << days[index] << "дней \n";
}
```
Размерность массива определяется компилятором, а в цикле определение размерности массива происходит с использованием операции sizeof.

Работа с массивами

Классическими задачами при работе с одномерными массивами являются нахождение суммы и произведения элементов массива, определение минимального и максимального элементов массива, сортировка (упорядочивание) элементов.

Пример нахождения суммы элементов массива рассмотрен в предыдущем разделе. Суммирование элементов происходит в цикле, но необходимо помнить, что значение переменной, предназначенной для хранения значения суммы, должно обязательно равно нулю до начала цикла. При подсчете произведения элементов массива значение произведения до цикла должно быть равно единице.
```
#include<iostream.h>
void main ( ) {
  const int n=15;
  int i, m[n], p=1;
```

```
 cout << " введите" << n << "чисел" <<endl;
                    // ввод элементов массива с клавиатуры
 for ( i=0; i < n; i++)
          { cin >> m[ i];
            if ( m[ i] > 0 ) p*= m[ i ] ; }
 cout << "произведение положительных элементов равно "<<p<<endl;
}
```

Стандартный алгоритм нахождения минимального значения заключается в следующем: предполагаем, что первый элемент является минимальным. Далее в цикле каждый следующий элемент сравниваем с минимальным. Если он меньше минимального (обнаруженного на данный момент), то минимальным становится текущий элемент.

Для реализации данного алгоритма нужно не забыть ввести вспомогательную переменную для хранения значения минимального элемента.

Пример. Нахождение минимального элемента в массиве, состоящем не более чем из 50 элементов.

```
#include<iostream.h>
void main ( )
{const int n=50;
 int i, k,  m[n], min;
 cout << " сколько элементов будете вводить?" <<endl;
                    // задаем количество элементов массива с клавиатуры
 cin >> k;
                    // ввод элементов массива с клавиатуры
 for ( i=0; i< k; i++)
          { cout << " введите m[ " << i << " ] ";  cin >> m[ i ];}
 min = m [ 0];
 for ( i=1; i< k; i++)  if ( m[ i ] < min )   min=m [ i ];
 cout << "минимальный из введенных элементов равен "<< m[ i] << endl;
}
```

Наиболее часто встречающая ошибка при создании такой программы заключается в том, что студенты забывают, что записи вида min = m [i]; и m[i] = min; не эквивалентны. В первой записи переменной min присваивается значение m[i] при этом само m[i] остается без изменения. Во втором случае наоборот.

Для нахождения максимального элемента нужно в условном операторе знак меньше поменять на знак больше. Рассмотрим пример нахождения максимального значения и его порядкового номера.

```
#include<iostream.h>
void main ( )
{
const int k=10;
 int i,   m[ k ], max, nmax;
 for ( i=0; i< k; i++)
          { cout << " введите m[ " << i << " ] ";  cin >> m[ i ]; }
 max = m [ 0]; nmax=0;
 for ( i=1; i< k; i++)  if ( m[ i ] > max )  { max=m [ i ]; nmax=i };
 cout << "максимальный элемент равен "<< m[ i] << "его номер" << nmax;
}
```

При разработке программного обеспечения очень распространенной операцией является сортировка значений, т.е. расположение списка в некотором порядке (например, слова по алфавиту или числа – в возрастающем или убывающем порядке). Один из способов сортировки – пузырьковый. В нем попарно сравниваются рядом стоящие элементы, и если второй больше первого (при сортировке по убыванию) элементы меняются местами.

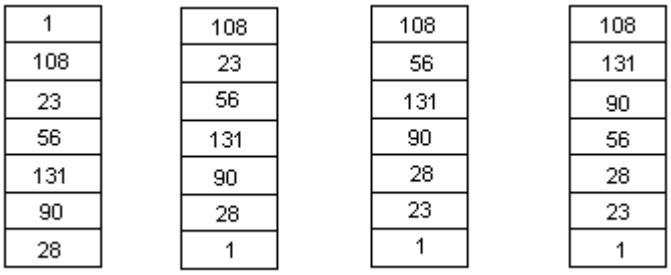


Рисунок 13. Сортировка «пузырьком»

Хотя в предложенном примере сортировка выполнилась за четыре шага, проверка элементов будет продолжаться еще две итерации, т.к. полное число итераций на единицу меньше размерности массива. Сортировка пузырьковым методом является неэффективным методом, в следствие большого числа сравнений.

Более эффективным является метод прямого выбора, при котором последовательно происходит просмотр всего списка значений и выбор из него минимального или максимального (в зависимости от порядка сортировки), далее расположение его на нужном месте обменом местами с элементом, стоявшим там ранее. Этот алгоритм представлен на рисунке.14.

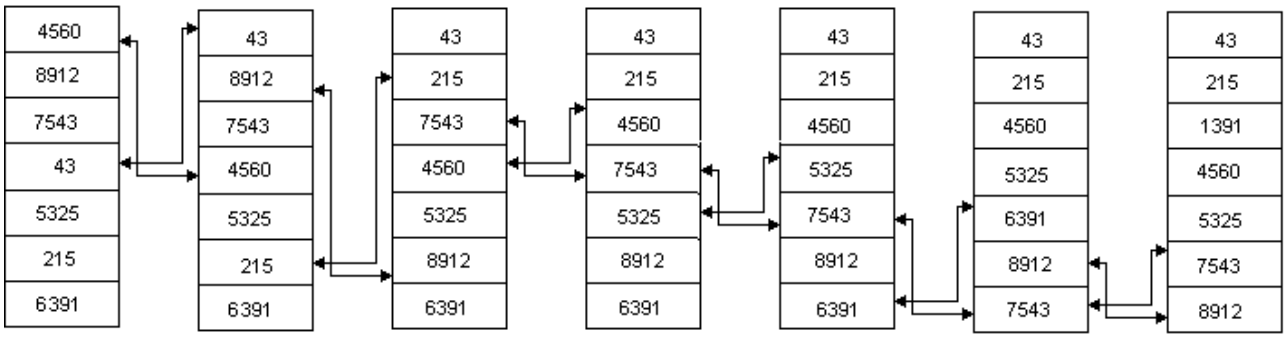


Рисунок 14. Сортировка методом прямого выбора

Для программной реализации этого алгоритма необходимо введение некоторой временной переменной, чтобы при обмене значения переменных не терялись. Функция, с помощью которой реализуется сортировка методом прямого выбора, приведена ниже.

```
void Sort ( )
// сортируем элементы массива по возрастанию
{const int n=20;
 int list [n];
 int temp;                    //переменная для временного хранения
 int i;                       //переменная управления циклом
 int place;                   //переменная управления циклом
 int minIndex;                //индекс минимального значения
    for ( i=0; i < n –1; i++ )
   {
        minIndex = i;
// ищем индекс минимального элемента среди значений list [ i … n -1]
        for (place = i+1; place< n; place++)
              if ( list[place] < list[ minIndex ] )  minIndex = place;
//меняем местами list[ minIndex ] и list[ i ]
        temp = list[ minIndex ];
        list[ minIndex ]= list[ i ];
        list[ i ] = temp;
    }
}
```

Следует отметить, что поиск минимального значения во внутреннем цикле происходит в оставшейся части списка.

Чтобы произвести сортировку по убыванию, нужно на каждом шаге вместо минимального значения находить максимальное значение.

Многомерные массивы

Для объявления многомерного массива необходимо после его имени задать несколько размеров, заключенных в квадратные скобки. Размерность массивов в С++ не ограничивается. Она может зависеть от возможностей компьютера и особенностей конкретного компилятора. Как правило, на практике используются двумерные массивы.

Двумерный массив в С++ представляет собой массив одномерных массивов.

```
float dam[4] [5];
```

В этом случае массив будет содержать 4 строки и 5 столбцов. Инициализация двумерного массива происходит следующим образом:

```
float art [5] [2]={  { 1. 2, 1. 5 },
                     { -4. 0, 3. 6 },
                     { 2. 3,  -6.1 },
                     { 7. 3,  0. 4 },
                     { 0. 0, -2. 7 } };
```

При этом внутренние фигурные скобки могут быть опущены. Например,

```
float art [5][2] = {1. 2, 1. 5, -4. 0, 3. 6, 2. 3, -6. 1, 7. 3, 0. 4, 0. 0, -2. 7} ;
```

Все сказанное может быть распространено на трехмерный массив.

```
int rum [3][7][2]; // массив размерности три, каждый элемент которого двумерный массив.
```

Если одномерный массив используется для представления списка, то двумерный массив – для представления таблицы, содержащей строки и столбцы. При этом подразумевается, что все элементы принадлежат одному типу данных. При обращении к отдельному элементу двумерного массива нужно указать его позицию в строке и столбце. Нумерация строк и столбцов начинается с нуля. Массив из примера можно представить как таблицу на рисунке 15. Выделенный на рисунке элемент есть art [1][2].

	[0]	[1]
[0]	1. 2	1. 5
[1]	-4. 0	3.6
[2]	2. 3	- 6. 1
[3]	7. 3	0. 4
[4]	0. 0	- 2. 7

столбцы

Строки

Рисунок 15. Двумерный массив

Обработка данных в двумерных массивах означает обращение к массиву одним из четырех способов: случайным образом, по строкам, по столбцам, по всему массиву. Каждый из этих способов может включать обработку части массива.

Самым простым способом получения значения элемента массива является точное указание местоположения элемента. Такой процесс называется случайным доступом, потому что пользователь может ввести любую случайную комбинацию координат, например

```
cout<< art [0][4];
```

Работа со строками

Часто встречаются ситуации, когда требуется обратиться к элементам массива в опре-

деленном порядке (например, найти максимальный элемент в каждой строке матрицы).

Пусть задан двумерный массив, содержащий 5 строк и 6 столбцов:

```
int A[5] [6];
```

Предположим, что нужно сложить все элементы строки с номером 3 (четвертая строка) в массиве А и вывести результат. Это можно легко сделать с помощью цикла for:

```
total = 0;
for ( int col = 0; col < 6; col++ )    total+= A[3][col];
        cout << "результат равен" << total << endl;
```

Этот цикл проходит по всем столбцам массива А, но номер строки всегда остается равным 3. Каждое значение из этой строки прибавляется к переменной total.

Если нужно получить сумму в двух строках – номер 1 и 2, то конечно можно дважды привести предыдущий фрагмент дважды: в первый раз указывать индекс 1, а во второй – 2.

```
total = 0;
for ( int col = 0; col < 6; col++ )    total += A[1] [col];
cout << "результат равен" << total << endl;
total = 0;
for ( int col = 0; col < 6; col++ )    total += A[2] [col];
cout << "результат равен" << total << endl;
```

Но правильнее создать вложенные циклы, а индекс строки сделать переменной - параметром внешнего цикла.

```
for ( int row = 1; row < 3; row++)
        { total = 0;
          for ( int col = 0; col < 6; col++ )  total +=A [row] [col];
           cout << "результат" << row << " строки равен" << total << endl;
        }
```

Этот способ короче и его значительным преимуществом является несложная его модификация для случая обработки любого диапазона строк.

Внешний цикл управляет изменением номера строки, а внутренний – номером столбца. Для каждого значения переменной row обрабатываются все столбцы, затем внешний цикл переходит к следующей строке.

Таким образом, обращение к элементам массива производится в следующем порядке:

A[1][0] A[1][1] A[1][2] A[1][3] A[1][4] A[1][5]

На второй итерации внешнего цикла переменная row увеличится на единицу и становится равной 2, а индекс столбца изменяется от 0 до 5:

A[2][0] A[2][1] A[2][2] A[2][3] A[2][4] A[2][5].

Таким способом можно обрабатывать все элементы в таблице.

В программах для обозначения строк очень часто используется переменная i, а для столбцов – j.

Пример. В матрице размером 4x7 определить максимальные элементы каждой строки и записать их в одномерный массив и вывести его на печать.

```
#include <iostream. h>
int main ( )
{ const int I =4;  // количество строк
  const int J =7;  // количество столбцов
   int m[I][J]={ 4, 7, 0, 9, 6, 2, 3, 68, 23, 0, -6, 3, 8, 5, 7, 9, -4, 3, 1, 3, 91, 3, 7, 6, 0, 4, 6, 1};
   int mmax [ I ], max, i, j;
   for (  i = 0; i < I; i++)
        {          max= m [ i ][ 0 ];
                   for (  j = 1; j < J; j++ )
                            if  ( m [ i ][ j ] > max )      max = m [ i ][ j ];
                    mmax[ i] = max;
        }
```

```
for (  i = 0; i < I; i++)    cout << “максимумы  строк равны” <<mmax << “  “;
}
```

Работа по столбцам

При работе со столбцами внешний цикл организуют по второму индексу массива, а внутренний по первому. Так для нахождения всех положительных элементов в каждом столбца и вывода полученных результатов можно написать:

```
for ( int  j = 0; j < J; j++)
        { summa = 0;
          for ( int i = 0; i < I; i++ ) if ( m[ i ][ j ] > 0)  m += m [ i ][ j ];
          cout << “результат” << j << “ столбца равен” << summa << endl;
        }
```

Сначала складываются все положительные элементы первого столбца, выводится результат, и только потом индекс внешнего цикла меняется и происходит обращение к элементам следующего столбца.

Инициализация таблицы

Как и в случае одномерных массивов, двумерные массивы можно инициализировать при объявлении. Это непрактично, если таблица имеет большую размерность. Для ввода значений с клавиатуры также можно использовать вложенные циклы.

```
for ( int  j = 0; j < J; j++)
for ( int i = 0; i < I; i++ )    cin >> m [ i ][ j ];
```

Здесь запись производится построчно, для ее выполнения по столбцам внутренний и внешний циклы нужно поменять местами.

При задании массива произвольным образом более эффективно использование элементов счетчика случайных чисел.

Вывод таблицы

Еще одним случаем обработки таблицы является задача вывода ее содержимого на экран в общепринятом виде (таблицей значений). Как правило, это подразумевает ее построчный вывод:

```
for ( int i = 0; i < I; i++ )
{         for ( int  j = 0; j < J; j++)   cout << m [ i ][ j ]<< “  “;
          cout << endl;
}
```

Внешний цикл позволяет выполнить вывод всех элементов строки, изменением второго индекса во внутреннем цикле, и переход на новую строчку перед каждым изменением параметра внешнего цикла.

Примеры программ обработки двумерных массивов

Пример. Определение минимального значения среди элементов матрицы M(6, 6), расположенных над главной диагональю.

```
#include <iostream. h>
#include <stdlib.h>
void main ( )
{
const int n = 6;
int M[n][n], i, j, min;
randomize( );
for (  i = 0; i < n; i++ )
        for (   j = 0; j < n; j++)
        M [ i ][ j ]=random(20) - 10;                  // задание значений в диапазоне от -10 до 10
```

```
for (  i = 0; i < n; i++ )                    // вывод на экран полученной матрицы
        {       for ( int  j = 0; j < n; j++)  cout << M [ i ][ j ]<< "  ";
                cout << endl;
        }
min = M[0] [1];
for (  i = 0; i < n; i++ )
        for (   j =i+1; j < n; j++)           / * условие расположения элементов
                                              над      главной диагональю */
        if ( M[ i ][ j ]<min)  min = M[ i ][ j];
 cout<< "значение минимума "<<min;
        }
```

Если требуется преобразовать элементы, расположенные в матрице под главной диагональю и на ней, используют вложенные циклы вида

```
for (  i = 0; i < n; i++ )
        for (   j =i; j < n; j++)
```

В случаях работы с элементами матрицы, лежащими на главной диагонали, достаточно использовать один цикл.

Пример. Поменять местами максимальный и минимальный элементы главной диагонали матрицы T(7, 7).

```
#include <iostream. h>
#include <stdlib.h>
void main ( )
{
const int n = 7;
int T[n][n], i, j, min, max, nmin, nmax;
randomize( );
for (  i = 0; i < n; i++ )
        for (   j = 0; j < n; j++)
        T [ i ][ j ]=random(30) - 15;         //задание значений в диапазоне от -15 до 15

for (  i = 0; i < n; i++ )                    // вывод на экран полученной матрицы
        {       for ( int  j = 0; j < n; j++)  cout << T [ i ][ j ]<< "  ";
                cout << endl;
        }
min = max=T[0] [0];
nmin=nmax=0;
for (  i = 1; i < n; i++ )
                {
                if ( min > T[ i ][ i ] )        { min = T[ i ][ i ];    nmin = i; }
                else if ( max < T[ i ][ i ] ) { max = T[ i ][ i ];    nmax = i; }
                }
int t = T[nmin][nmin];                        // вспомогательная переменная для обмена
T[nmin][nmin] =T[nmax][nmax];
T[nmax][nmax] =t;
        for (  i = 0; i < n; i++ )                      // вывод на экран измененной матрицы
        {       for ( int  j = 0; j < n; j++)  cout << T [ i ][ j ]<< "  ";
                cout << endl;
        }
}
```

Библиотека stdio.h

Аббревиатура stdio.h означает standard input output header, т.е. заголовок стандартного ввода-вывода. Данная библиотека содержит прототипы функций стандартного ввода вывода.

Она широко использовалась в языке С и поддерживается в С++.

Функция printf( ) – форматный вывод. Синтаксис обращения к printf( ):

printf(“управляющая строка”[, параметр1 [, параметр2 [, . . ., параметрn]]]);

где параметр1, параметр2, . . ., параметрn – выводимые на экран значения, которые могут быть переменными, константами или выражениями, вычисляемые перед выводом.

Управляющая строка задает формат вывода значений на экран. Управляющей строкой служит фраза, помещенная в кавычки и содержащая информацию двух видов: тест, непосредственно выводимый на экран, и инструкции, зависящие от типа выводимых значений (форматы вывода) и размещенные на месте предполагаемого вывода значений.

Формат	*Тип переменной*
%d	Десятичное целое число
%c	Один символ
%s	Строка символов
%e	Экспоненциальная запись числа с плавающей точкой
%f	Десятичная запись числа с плавающей точкой
%u	Десятичное число без знака
%o	Восьмеричное целое без знака
%x	Шестнадцатеричное целое без знака

Таким образом, управляющая строка показывает, в каком виде будет осуществлен вывод информации на экран.

Например, запись вида

```
const float pi= 3.14;
printf (“значение числа pi равно %f \n”, pi);
```

выведет на экран:

значение числа pi равно 3.14

после чего осуществиться переход на новую строку.

Каждому аргументу из списка, следующему за управляющей строкой, должна соответствовать отдельная инструкция:

```
int x=125, y;
y=x*x;
printf (“Квадрат числа %d равен %d \n”, x, y);
```

Это приведет к выводу

Квадрат числа 125 равен 15625

Указанием в инструкции числа (спецификатора точности) определяет ширину поля и точность. Например,

```
int k = 12;
float x = 5.12, y = 4.275, z;
z=x*y;
printf (“значение числа k равно %2d \n”, k);
printf (“x = %5.2f, y = %6.3, z = %7.3f  \n”, x, y, z);
```

Первая цифра означает количество знаков для вывода всего числа, а вторая количество знаков после запятой. Если цифр в числе меньше указанного формата лишние цифры просто отбрасываются.

Если спецификатор точности не задан, точность будет установлена по умолчанию: 1 – для форматов d, o, u, x; 6 – для формата e.

scanf( ) – функция форматного ввода. В ней также указывается управляющая строка и список аргументов. Основное отличие от printf( ) – в управляющей строке указывается только инструкции формата, а в списке параметров перед именем переменных обязательно располагается &. Таким образом, функция scanf() использует адреса расположения переменных в памяти (указатели). Особенным является использование формата строковой переменной %s, перед именем переменной знак & не указывается. Например,

```
printf (“укажите Ваше имя, возраст и состояние:”);
scanf( “%s  %d  %f ”, name, &age, &st );
printf ( “%s  -  %d лет, %f тыс. руб. \n”, name, age, st);
```

Ввод-вывод одного символа осуществляется с использованием функций getchar( ) и putchar( ). Функция getchar( ) – получает один символ, поступающий с клавиатуры и возвращает его, putchar( ) выводит один символ на экран.

```
char ch=getchar( );
putchar( ‘s’ );
putchar( ch);
putchar(getchar( ));
```

Библиотека stdio.h не ограничивается перечисленным и содержит множество других полезных функций.

Для вывода на экран цветных сообщений пользуются библиотекой conio.h. Для вывода

Символьные строки

Символьная строка – последовательность одного и более символов, заключенная в двойные кавычки. Для формирования символьных строк, занимающих несколько строк программы, используется комбинация символов \ и \n. Кавычки не являются частью строки, они служат для обозначения ее начала и конца. Строки представляются в виде массива элементов типа char. Число элементов символьного массива на единицу больше числа символов в строке. Это требуется для нуль-символа (/0), который автоматически добавляется в качестве последнего байта в памяти для того, чтобы отмечать конец строки.

Строка может быть строковой константой или строковой переменной (символьным массивом). В обоих случаях можно выводить элементы строки с помощью оператора cout<< до тех пор, пока не встретится нулевой символ.

Например:

```
cout<<”results are:”;
char msg[ ]=”welcome”;
cout<<msg;
```

Инициализировать строку можно всю целиком или посимвольно:

```
char message [10]= {‘c’, ‘o’, ‘o’, ‘б’, ‘щ’, ’e’, ‘н’, ‘и’, ‘e’};
```

Для ввода строк существует несколько возможностей. Первая – использование оператора cin. При чтении вводимых данных этот оператор будет пропускать все предшествующие непечатаемые символы: пробелы и символы перевода строки. Он также автоматически добавляет нуль-символ в конец строки.

Например, если объявлены две строки:

```
char firstStr[31], secondStr[31];          //2 строки из 30 символов и место для ‘/0’
```

а поток ввода выглядит как □□abc□□defgh□□15, тогда с помощью кода

```
cin>> firstStr >> secondStr;
```

символы ‘a’,’b’,’c’, ‘/0’ поместятся в элементы массива с firstStr[0] по firstStr[3], а символы ‘d’, ‘e’, ‘f’, ‘g’, ‘/0’ – в элементы массива с secondStr[0] по secondStr[4].

Таким образом, данный оператор нельзя использовать для ввода строк, содержащих пробелы. Второй его недостаток в том, что если строковая переменная недостаточно велика, чтобы вместить в ней последовательность вводимых символов и нуль-символ, то данные из потока ввода будут записываться в память за пределами массива.

Для вывода строк также можно использовать функцию printf( ). Для ввода – scanf( ) и get( ). Функция get ( ) в этом имеет два параметра: строковую переменную и целое типа int, отвечающее за количество вводимых символов в строке плюс один (для нуль-символа).

Например:

```
char line[51];
cin.get(line, 51);
```

В этом случае непечатные символы не пропускаются. Функция считывает и сохраняет

полностью всю строку ввода (длиной не более чем указано вторым параметром). Чтобы правильно считать две последовательные строки, нужно не забывать о символе передачи строки:

```
char dummy;
cin.get(line1, 51);
cin.get(dymmy);          //убрать '/n' из потока ввода перед использованием get
cin.get(line2, 51);
```

Язык С++ предлагает большой ассортимент полезных функций для работы со строками. Прототипы всех функций работы со строками содержатся в файле string.h. Рассмотрим несколько из них.

Функция strlen() вычисляет длину строки без нуль-символа. Функция имеет один аргумент – имя строки.

```
#include <string.h>
#include <iostream.h>
void main ( )
{
char str[ ]= "Hello, my friend!";
cout<<strlen(str);                    // результат 17
}
```

Функция strcmp( ) сравнивает две строки. Она имеет два аргумента и возвращает целое. Значение этой функции равно 0, если строки равны. Значение strcmp(str1, str2) – целое число меньше 0, если str1<str2 и целое >0, если str1>str2. Сравнение строк происходит в лексикографическом порядке, т.е. в порядке их расположения в словаре.

Пример. Функция, проверяющая правильность введенного пароля с трех попыток.

```
#include <string.h>
#include <iostream.h>
int password ( )
{
char s[5], pass[ ]="лето";
int i_true=0;
for (int i=0; i<3; i++)
        { cin>>s;
        if (strcmp (s, pass)= =0)  {  i_true=1; break; }
        }
if (i_true = = 0) {cout<<"пароль неверен"; return 1; }
else {cout<<"пароль верен"; return 0; }
}
```

Функция strcpy( ) копирует содержимое второй, из указанных в качестве параметра, строки в первую, замещая прежние данные, включая '/0'. При этом первая указанная строка должна иметь достаточный размер.

```
char mystr[100];
strcpy(mystr, "abcdefgh");
```

Работа с символьным массивом аналогична работе с числовым массивом.

Пример. Определение количества вхождений каждого символа в заданную строку.

```
#include <stdio.h>
#include <string.h>
void main ( )
{ int k;
  char *str;
  printf ("Введите любую последовательность символов /n");
  scanf (" %s ", str);
  for (char ch='a'; ch<='z'; ch++)
        { k=0;
```

```
            for ( int i=0; i < strlen(str); i++)
             if (str [ i ] ) == ch) k++;
            if (!k)   printf ("Количество символа %c равно %d /n", ch, k);
          }
}
```

Программа, сортирующая заданной строки в алфавитном порядке.

```
#include<iostream.h>
#include<string.h>
#include <conio.h>
void main( )
{ clrscr( );
char m[ ]=" ночевала тучка    золотая на груди            утеса-великана ";
int i;
for (char c='a'; c<'я'; c++)
        {  i=0;
          if ( m[0]==c)     { cout<<m[0];
                                  while (m[i+1]!='  ')              { cout<<m[i+1];  i++; }
                                  cout<<endl;
                                }
           for  ( i=i; i<strlen(m); i++ )
                 if (m[ i ]== '  ' && m[i+1]==c)              { while (m[i+1]!=' ') { cout<<m[i+1];  i++; }
                                                                  cout<<endl;
                                                               }
           }
}
```

В примере первый цикл организован для пропуска лишних пробелов между словами.

Массив символьных строк выглядит как двумерный массив. К его строкам применимы все функции для работы со строками. Можно работать и непосредственно с отдельными его элементами. Так инициализировать символьный массив, содержащий четыре строки можно следующим образом:

static char fruit[4][ ]={“Слива”, “Персик”, “Яблоко”, “Апельсин”};

Теперь компилятор автоматически определит количество элементов в строке – девять, т.к. самое длинное слово содержит восемь символов плюс один для размещения нуль-символа.

Функции

Функция – это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие. Функция может принимать параметры и возвращать значение.

Фигурные скобки отмечают начало и конец тела функции. В круглых скобках после ее имени, в общем случае, содержится информация, передаваемая этой функции – описание параметров (тип и имя).

Наличие списка параметров и их описаний не является обязательным. Но круглые скобки всегда должны присутствовать после имени функции. Тип параметра может быть любым. Если параметров несколько, их описания разделяются запятыми. Если функции не передаются величины, то вместо списка аргументов можно задавать ключевое слово void.

Некоторые компиляторы требуют обязательного указания типа возвращаемого результата перед именем функции. Если тип результата не указывается, то предполагается, что функция возвращает значение типа int. Если функция не возвращает результат, указывается слово void.

Возвращает значение оператор return, с помощью которого можно передать в вызывающую функцию только одно значение. Возвращаемое значение берется в круглые скобки после ключевого слова return. Круглые скобки не являются обязательными, но считаются

правилом хорошего тона среди программистов.

Переменные, используемые в функции, отличные от параметров, должны описываются внутри тела функции. Они называются локальными.

Пример. Функция, возвращающая минимальное значение из трех величин.

```
        // определение функции min
float min (float val1, float  val2, float val3 ) {
        if ( val1< val2 &&  val1<val3) return val1;
                else if ( val2 < val1 &&  val2<val3) return val2;
                        else if ( val3 < val2 &&  val3<val1) return val3;
}
```

Эта функция может быть вызвана любой другой функцией. Вызов может выглядеть, например, так

```
 a = min (525, 675, 819);
```

Если в программе объявлены некоторые переменные: float a, a1, a2, a3; тогда возможен вызов

```
a = min (a1, a2, a3);
```

Параметры (аргументы) функции, используемые в ее описании, называются *формальными*, параметры, содержащиеся в вызове функции и располагаемые на их месте, называются *фактическими*.

Так в описании функции min формальными параметрами являются val1, val2, val3, а фактическими параметрами в ее вызове становятся 525, 675, 819 и a1, a2, a3.

Количество и типы формальных и фактических параметров должны совпадать. Иначе компилятор выдаст ошибку.

В случаях, когда вызываемая функция не возвращает значение, ее вызов представляет собой просто имя функции, за которым в круглых скобках указываются фактические параметры, если таковые имеются.

Функции могут располагаться в тексте программы в любом порядке, но следует помнить, что они должны быть определены до своего вызова.

Так пример программы, содержащей функцию min( ) может выглядеть так:

```
 #include <iostream.h>
 // определение функции min
 float min (float val1, float  val2, float val3 )
{                                                                   // начало функции min
        if ( val1< val2 &&  val1<val3) return val1;
                else if ( val2 < val1 &&  val2<val3) return val2;
                        else if ( val3 < val2 &&  val3<val1) return val3;
}                                                                   // окончание функции min
// определение функции main
void main ( )
{                                                                   // начало функции main
  int x, y, z;
  cout<< " Введите три числа";
  cin>> x >> y >> z;
  cout<< "минимум из этих чисел"<< min (x, y, z);   //вызов функции min
}                                                                   // окончание функции main
```

Необходимо отметить, что приведенный пример является "неграмотным" с точки зрения эффективности программы. Поскольку функции в программе должны создаваться с целью упрощения алгоритма программы, структуризации сложной программы, ее наглядности и читаемости. В данной программе нецелесообразно выделять отдельную функцию нахождения минимума, так как это только усложняет ее текст. Функции используются, как правило, когда требуется выполнить одинаковые действия с данными, имеющими различными значениями одинакового типа.

Разделение программы на функции позволяет избежать избыточности кода, поскольку определение функции записывается один раз, а вызывать ее можно многократно из разных точек программы.

Тело функции похоже на любой другой фрагмент кода, за исключением того, что оно содержится в отдельном блоке внутри программы. При разработке функции нужно формально описать ее поведение и механизм взаимодействия с ней, т. е. входные и выходные значения.

Например, с помощью программы, решили вывести сообщение «С Днем рождения!!!», ограничив его сверху двумя строками звездочек, а снизу четырьмя строками. Можно определить функцию вывода на печать двух строк и воспользоваться ею трижды. Данная функция не будет возвращать значение, а в качестве входного параметра можно использовать цвет, которым будут выводиться символы.

```
#include <iostream.h>
#include <conio.h>
void print_str( int x)                    //определение функции print_str
        { textcolor(x );
          cout<< " ******************** "<<endl;
          cout<< " ******************** "<<endl;
        }
void main ( )              // определение функции main
        {
        clrscr( );         // очистка экрана
        textbackground(14 );
        print_str( int 4);
        textcolor(3+128);
        cout<<"С Днем рождения!!!"<<endl;
        print_str( int 10);
        print_str( int 12);
        clrscr( );
        }
```

Нередким является случай, когда функция вызывается до того, как будет объявлена. В этом случае используется прототип функции. Прототип функции имеет следующий вид:

< возвращаемый тип > имя функции (параметры);

Таким образом, прототипом является заголовок функции, оканчивающийся точкой с запятой. В списке параметров обычно указывается тип и имя для каждой переменной; элементы списка разделяются запятыми. Указание имени параметров в прототипе не обязательно, но, как правило, применяется.

Прототип информирует компилятор о существовании функции, о типе возвращаемого значения, а также о типе и количестве аргументов, которые ей передаются.

Так для функции из вышеприведенного примера прототип выглядит следующим образом:

```
        void print_str( int x);
```

или

```
        void print_str( int );
```

Прототипы функций, как правило, располагают после директив препроцессора, до описания основной функции. Рассмотрим пример программы нахождения площадей треугольников с использованием прототипов функций.

```
#include <iostream.h>

#include <math. h>

void print_area (float, float, float);        //объявление прототипа функции print_area

void main (void)                                    //описание функции main
```

```
{
  int n; float a, b, c;
  cout<<"Введите количество треугольников"<<endl;
  cin>>n;
  for (int i=1; i<=n; i++)
        { cout<< "Введите стороны треугольников a, b, c >0" << endl;
         cout<<"a = ";  cin>>a;
        cout<<"b = ";  cin>>b;
        cout<<"c = ";  cin>>c;
        print_area (a, b, c);
        }
}
void print_area (float x, float y, float z)    //описание функции print_area
{
        if ( ( x+ y > z )&&( x+ z > y )&&( y+ z > x ) )
        { float p=( x + y+ z) / 2;
          cout <<"Площадь равна "<<sqrt(p*(p-x)*(p-y)*(p-z))<<endl;
        }
        else cout<<"Треугольник невозможно построить"<<endl;
}
```

Нужно разделять понятия «определение» и «объявление» функции. Объявление функции это заголовок или прототип. Определение функции, кроме объявления, содержит тело функции.

Все функции в программе, созданной на языке С++, равноправны, т.е. любая функция (включая main) может вызвать другую, в том числе и саму себя. Вызов функцией самой себя называется рекурсией. В языке С++ количество рекурсивных вызовов не ограничивается, а определяется лишь возможностями компьютера.

Пример: Программа, демонстрирующая применение рекурсии при вычислении факториала.

```
#include <stdio.h>
double factorial (int number);      // объявление прототипа функции
int main( )
{
int n;
double result;
printf ("Введите число\n");
scanf ("%d",&n);
result=factorial(n);
printf("факториал %d равен %15.0f\", n, result);
return (0);
}
double factorial (int number)   // объявление функции
{
if (number<=1) return (1.0);
else return(number*factorial(number-1);
}
```

Для корректной работы рекурсивной функции, она должна содержать хотя бы одну

нерекурсивную ветвь алгоритма, заканчивающуюся оператором возврата. Данная рекурсия называется прямой. Существует еще косвенная рекурсия, когда две или более функций вызывают друг друга.

Рекурсивные функции чаще применяются для реализации рекурсивных алгоритмов. Любую рекурсивную функцию можно реализовать и без применения рекурсии.

Достоинством рекурсии является компактная запись, а недостатком - расход на повторные вызовы функции и передачу ей копий параметров, а также опасность переполнения стека, в котором хранятся копии значений параметров функции.

Область действия и время жизни переменных

Как уже говорилось, переменные в языке C++ могут быть объявлены в любом месте программы. Локальные переменные – это переменные, объявленные внутри блока, например, такого как тело функции или условный оператор. К локальным переменным нельзя обращаться вне блока, который их содержит.

```
if (k>0)
{ int n;
  cin>>n;
  k=k+n;
}
```

Поскольку в данном примере переменная n – локальная, то к ней нельзя обращаться в любом выражении вне блока. То же правило доступа относится и к локальным именованным константам.

Но переменные могут быть объявлены и вне блока. Такие переменные называются глобальными. Глобальные переменные видны во всех функциях программы, где не описаны локальными переменные с теми же именами. Использовать их для передачи между функциями очень легко. Тем не менее, это делать не рекомендуется, поскольку затрудняется отладка программы. Использование глобальных переменных считается плохим стилем программирования.

Если перечислить все фрагменты программы, из которых к идентификатору можно обращаться, то получится описание области видимости идентификатора или его *области действия*. C++ определяет три категории области действия для любого идентификатора:

- Область действия класса. Этот термин относится к типу данных, называемому классом.
- Локальная область действия. Область действия идентификатора, объявленного внутри блока, простирается от точки объявления до конца этого блока.
- Глобальная (файловая) область действия. Область действия идентификатора, объявленного снаружи всех классов и функций, простирается от точки объявления до конца всего файла.

Когда функция объявляет локальный идентификатор с тем же самым именем, что и у глобального идентификатора, локальный идентификатор имеет приоритет внутри тела функции. Другими словами, область действия идентификатора не включает вложенные блоки, которые содержат локально объявленный идентификатор с точно таким же именем.

Имена функций C++ имеют глобальную область действия, и не существует такого понятия, как локальная функция – то есть запрещается вкладывать описание одной функции в другую.

Глобальные переменные встречаются достаточно редко. Имеются отрицательные аспекты их использования. Область действия глобальной переменной или константы простирается от точки ее объявления до конца файла.

Понятие, и связанное и отдельное от области действия переменной – это *время жизни* – период времени в процессе выполнения программы, когда идентификатор фактически занимает место в памяти. Память для локальных переменных выделяется в момент перехода управления на функцию. Затем переменные «оживают» на время работы функции, и память

освобождается при выходе из функции. Время жизни глобальной переменной – это время работы всей программы. Память для глобальных переменных выделяется только один раз, когда программа начинает выполняться, и освобождается только при завершении программы.

Необходимо отметить, что понятие области действия связано с этапом компиляции, а время жизни – с этапом выполнения программы.

В C++ каждая переменная имеет класс памяти, который определяет время жизни. Существует четыре спецификатора класса памяти: auto, register, static, extern.

Спецификатор класса памяти может предшествовать объявлению переменных и функций, указывая компилятору, как следует хранить переменные в памяти и как получать доступ к переменным или функциям. Переменные, объявленные со спецификаторами auto и register, являются локальными, а со спецификаторами static и extern – глобальными. Смысл спецификатора класса памяти несколько различается в зависимости от места объявления переменной или функции.

Переменная, объявленная на внешнем уровне (вне любой функции), по умолчанию имеет класс памяти extern. Область ее действия распространяется до конца файла. С помощью спецификатора extern можно объявить переменную, которая будет доступна из любого места программы. Это может быть ссылка на переменную, описанную в другом файле или ниже в том же файле.

Если в одном из файлов создана переменная с классом памяти static, то она может быть объявлена с тем же именем и с тем же спецификатором static в любом другом исходном файле. Так как статические переменные доступны только в пределах своего файла, конфликтов имен не возникает.

При объявлении переменной на внутреннем уровне (в теле функции) можно использовать любой из четырех спецификаторов памяти (по умолчанию устанавливается класс auto). Переменные, имеющие класс памяти auto, имеют область видимости ограниченную блоком, в котором они объявлены.

Спецификатор register указывает компилятору, что данную переменную необходимо сохранить в регистре процессора, если это возможно. В результате сокращается время доступа к данным и упрощается программный код. Область действия регистровых переменных такая же, как и у автоматических. В случае отсутствия свободных регистров переменной присваивается класс auto, и она сохраняется в памяти.

Переменная, объявленная на внутреннем уровне со спецификатором static, будет глобальной, но доступ к ней получить можно только внутри блока. По умолчанию статической переменной присваивается нулевое значение.

Спецификатор extern на уровне блока используется для создания ссылки на переменную с те же именем, объявленную на внешнем уровне в любом исходном файле программы. Если в блоке определяется переменная с тем же именем, но без спецификатора extern, то внешняя переменная становится недоступной в данном блоке.

Пример:

```
//исходный файл 1
extern int i;       //объявление, ссылающееся на данное ниже определение  i
void main( ) {
i=i+1;
cout<<i;//значение i равно 4
next( );
}
int i=3;            // определение i

next( ) {
i=i+1;
cout<<i;// значение i равно 5
```

```
other( );
}
//исходный файл 2
extern int i;       // объявление i, ссылающееся на определение i в первом исходном файле
other( ) {
i=i+1;
cout<<i;// значение i равно 6
}
```

В примере два исходных файла в совокупности содержат три внешних объявления переменной i. Только в одном содержится ее инициализация. Самое первое объявление extern в первом исходном файле делает глобальную переменную доступной прежде ее определения в файле. Объявление переменной во втором исходном файле делает глобальную переменную доступной во втором файле.

Если бы переменная не была инициализирована ни в одном из объявлений, она бы была неявно инициализирована нулевым значением при компиляции. В этом случае программа напечатала бы значения 1, 2, 3.

Указатели

При обработке объявлений переменных компилятором в памяти выделяется место в памяти в соответствии с ее типом. После чего все обращения в программе к переменной по ее имени заменяются компилятором на начальный адрес области памяти, которая выделена под ее хранение. В ней располагается значение переменной.

Программист может сам определять переменные для хранения адресов памяти. Указатель является переменной, которая содержит адрес другой переменной или функции. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим типом.

Описание указателя определяет тип данных, на которые указатель ссылается. Синтаксис объявления указателя:

<тип указываемых данных> * <имя указателя>;

Примеры:

```
int *int_ptr;                   // указатель на целое
double *d_ptr;          // указатель на тип double
char *c;                // указатель на символьную переменную
int *array[10];                 // объявление массива указателей, каждый из которых
                                // указывает на  значение int
int (*pointer)[10];             // объявлен указатель с именем pointer, который
                                 // указывает на  массив из 10 элементов
```

Возможно также использование указателя на тип void, который может указывать на объект любого типа. Его обычно называют пустым указателем. При выполнении операций над пустым указателем, либо над объектом, на который он указывает, необходимо явно привести тип указателя к типу отличному от void.

Как и любые другие переменные, указатели можно инициализировать при их объявлении. Например, в следующем фрагменте создаются две именованные ячейки памяти result и p_result.

```
int result;
int *p_result=&result;
```

Идентификатор result представляет собой обычную целочисленную переменную, а p_result – указатель на переменную типа int. Одновременно с объявлением указателя p_result происходит его инициализация адресом переменной result. Сама переменная result остается неинициализированной.

При определении указателей надо стремится выполнить их инициализацию. Непреднамеренное использование неинициализированных указателей – распространенный вид ошибок в программах.

Можно также записывать инициализатор после имени указателя в круглых скобках:

```
int result;
int *p_result (&result);
```

или явно присваивать указателю адрес области памяти:

```
float* r=  (float*)0xB8000000;
```

где 0xB8000000 – шестнадцатеричная константа.

Для присваивания указателю пустого значения используют 0 или константу NULL, определенную в заголовочных файлах C++.

Операции с указателями

Для указателей, как и для переменных других типов, существует ряд операций: присваивание, сложение с константой, вычитание, инкремент, декремент, сравнение, приведение типа, косвенная адресация или разадресация (разыменовывание) – * и операция получения адреса – &.

Результатом операция косвенной адресации является величина, помещенная в ячейку с указанным адресом. Ее можно использовать для получения и изменения значения некоторой величины.

Операция получения адреса (&) можно применять далеко не с каждым выражением. Когда за этим знаком следует имя переменной, результатом операции является номер ячейки памяти, в которой она хранится.

Недопустимо попытка получения адреса в случаях:

- константного выражения, например ptr=&57;
- в выражениях с арифметическими операторами

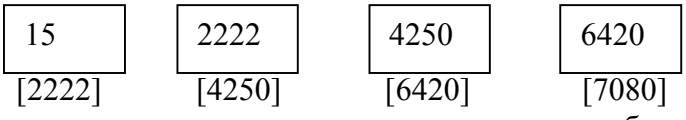
```
int rezult=0;
ptr=&(result+35);
```

- с переменными класса памяти register.

В C++ можно создавать указатели на другие указатели, которые, в свою очередь, содержат имена реальных переменных. Чтобы объявить в программе указатель, который в свою очередь будет хранить адрес другого указателя, нужно просто удвоить число звездочек в объявлении. Количество указателей в цепочке, задающее уровень косвенной адресации, соответствует числу звездочек перед именем указателя. Уровень косвенной адресации определяет, сколько раз следует выполнить операцию раскрытия указателя, чтобы получить значение конечной переменной. В следующем фрагменте создается ряд указателей с различными уровнями косвенной адресации.

```
int value=15;
int *ip;
int **ipp;
int ***ippp;
ip=&value;
ipp=&ip;
ippp=&ipp;
```

В четырех первых строчках объявлены четыре переменные: value типа int, указатель ip на переменную типа int (первый уровень косвенной адресации), указатель ipp (второй уровень адресации) и указатель ippp (третий уровень адресации). Можно создать указатель любого уровня. В пятой строке указателю первого уровня ip присваивается адрес переменной value. Теперь значение переменной value (15) может быть получено с помощью выражения *ip и т.д.



В результате вычитания целого значения указатель будет ссылаться на элемент, смещенный на указанную величину ячеек влево по отношению к текущей ячейке. Соответственно ис-

пользование операций инкремента и декремента к указателю будет производить сдвиг на количество ячеек памяти соответствующие типу переменной, на которую ссылается указатель. Предполагается, что оба указателя одного типа и связаны с одним и тем же массивом. Иначе результат невозможно предсказать.

Результатом разности указателей будет целое число, соответствующее числу элементов между ячейками, на которые они ссылались.

При записи выражений с указателями нужно помнить о приоритетах операций. Например, *ptr++=25;

Операции разадресации и инкремента имеют одинаковый приоритет и выполняются справа налево. Но инкремент используется в постфиксной форме, он выполняется после операции присваивания. Следовательно, сначала по адресу, записанному в ptr, поместится 25, а затем увеличится значение указателя, т.е. аналогично *ptr=25; ptr++;

Указатели и массивы

При использовании указателей происходит работа с адресом ячейки памяти и получение лишь косвенного доступа к ее содержимому.

Часто указатели используются при работе с массивами. Указатели и массивы логически связаны друг с другом. Имя массива является константой, содержащей адрес первого элемента массива. Вследствие чего значение имени массива не может быть изменено оператором присваивания или каким-либо другим оператором. Так, если объявлен массив

```
float temp[10];
```

то `temp==&temp[0]`

Используя операции сложения, вычитания инкремента и декремента к имени массива (или к указателю) можно передвигаться по элементам массива. Т.е. temp++ будет указывать на второй элемент массива, т.е. элемент с индексом 1. Таким образом, можно работать с массивами, не используя их стандартного обращения к собственным элементам. Используя операцию косвенной адресации можно обращаться к значениям элементов массива. Выражение *(temp+1) эквивалентно temp[1].

Это верно и для многомерных массивов. Предположим, есть описание:

```
int zippo [4][2];
```

Напомним, что первое число означает количество строк, второе – количество столбцов. Тогда zippo = = &zippo[0][0], а zippo+ 5 на zippo[2][1].

Обращаться к элементу массива можно и следующим способом:

*(zippo[i] + j) или *(* (zippo+ i) + j)

Двумерный массив является одномерным массивом, элементы которого - массивы, следовательно, имя его первой строки zippo[0], а имя четвертой строки - zippo[3]. Однако имя массива является также указателем на этот массив в том смысле, что оно ссылается на его первый элемент. Следовательно,

```
zippo[0]= =&zippo[0][0]
zippo[2]= =&zippo[2][0].
```

Передача массива в функцию выполняется следующим механизмом: в описании функции в списке формальных параметров указывается тип элементов массива, его имя, а после пустые квадратные скобки, которые указывают, что данный аргумент является массивом. При вызове функции в списке фактических параметров указывается имя массива. Например,

```
int sum (int n, int array[ ]) { … }     // объявление функции
```

И тогда, в действительности, функция получает адрес первого элемента массива, и следовательно, она может быть вызвана и следующим образом:

```
int c=sum(15, &arr[0]);   // где int arr[15];
```

или более простой конструкцией:

```
int c=sum(15, arr);
```

В случае передачи в функцию в качестве параметра двухмерного массива в первой

паре квадратных скобок размерность не указывается, а во второй паре – скобок должна быть указана обязательно. Например,

int sort(int ard[ ][7]);

В этом случае фактическим параметром также может служить имя массива.

При передаче в функцию, как одномерного массива, так и двумерного, можно в качестве формальных параметров использовать напрямую указатели, но при этом требуется следить, чтобы не выйти за пределы массива.

Пример.

Программа, вычисляющая значение $z = (a,b) - (b,c)(d,e)$, где a, b, c – векторы размерности 10; a, d и e векторы размерности 12. Запись вида $(x,\ y)$ означает скалярное произведение векторов. Координаты векторов хранятся в одномерных массивах.

```
#include <iostream.h>
#include <stdlib.h>
int scalar (int* x1,  int* x2, int n);
// прототип функции для подсчета скалярного произведения
// два первых параметра – указатели на целый тип,  для передачи массивов
// третий параметр – размерности массивов
void main ( )
{ int a[10], b[10], c[10], d[12], e[12];
int i;
randomize( );
for ( i=0; i< 10; i++)
 {        a[ i ] = random (20) -10;
          b[ i ] = random (20) -10;
          c[ i ] = random (20) -10;
}
for ( i=0; i< 12; i++)
 {        d[ i ] = random (20) -10;
          e[ i ] = random (20) -10;
}
int z = scalar(a, b, 10) – scalar(b, c, 10) * scalar(d, e, 12);
cout << “z=”<<z;
}
int scalar (int* x1,  int* x2, int n)   // описание функции
{ int sc=0;
for ( int k=0; k<n; n++)
sc+= *( x1+ k )* *(x2 + k );
return sc;
}
```

Поскольку одномерный массив является массивом, состоящим из массивов, о чем уже говорилось, это позволяет использовать функцию, предназначенную для работы с одномерным массивом, для работы со строками двумерного массива.

Пусть описана функция нахождения среднего арифметического массива целых чисел:

```
float mean(int array[ ], int n)
{ int index,  sum=0;
  if (n>0)
                    {          for (index=0,sum=0; index<n; index++)  sum+ = *(array + index);
                    return (float)sum / n;      // явное приведение типа
                    }
  else              { cout<<”Нет массива”<<endl;        return 0;       }
}
```

Параметром функции является целочисленный массив. Поэтому сумма элементов

также будет целой. Но в C++ при делении целого на целое результат также целый. Поэтому используется явное приведение типа. Заметим, что к типу float, приводим только первый операнд, и тогда при делении вещественного числа на целое получим вещественный результат. Если привести к типу float все выражение, т.е. (float)(sum / n); - то дробная часть будет равна нулю.

Данную функцию можно использовать для нахождения среднего арифметического строки матрицы:

```
void main ( )
{ int nk[ 3] [4] = { {2,4,6,9}, {10, 20, 40,10}, {3, 7, 0, 9} };
for ( int line=0; line<3; line++)
printf (“Среднее арифметическое %d строки равно %d \n”,line+1,mean(nk[line], 4);
}
```

Динамическая память

Во время компиляции программ на языках C/C++ память компьютера делится на четыре области: программного кода, глобальных данных, стек и динамическую область. Последняя отводится для хранения временных данных и управляется функциями распределения памяти, такими как malloc( ) и free( ). Функция malloc( ) резервирует непрерывный блок ячеек для хранения указанного объекта и возвращает указатель на первую ячейку этого блока. Функция free( ) освобождает ранее резервированный блок и возвращает эти ячейки в динамическую область для последующего резервирования.

В качестве аргумента в функцию malloc( ) передается целочисленное значение, указывающее количество байтов, которое необходимо зарезервировать. Если резервирование прошло успешно, то malloc( ) возвращает переменную типа void*, которую можно привести к любому необходимому типу указателя.

В следующем примере резервируется память для 300 чисел типа float:

```
float *pf;
int number=300;
pf=(float*) malloc (number*sizeof(float));
```

Выделенный блок надежно отделен от других блоков и не перекрывается ими. Предугадать, где именно он будет размещен невозможно. Все блоки надежным образом помечаются, чтобы система могла легко их обнаружить и определять их размер.

Если блок ячеек больше не нужен, его можно освободить следующей строкой:

```
free ( (void*) pf);
```

Прототипы данных функций содержатся в файле stdlib.h.

В программах на языках C++ резервирование и освобождение блоков памяти в динамической области может происходить в любой момент времени. Необходимо помнить об освобождение зарезервированной памяти, как только отпадет необходимость в соответствующем объекте. Если, не заботясь об освобождении неиспользуемой памяти, создавать все новые и новые динамические объекты, то программа может столкнуться с ее недостатком. При нехватке динамической памяти компилятор располагает переменные в области программного кода, не сообщая об этом программисту.

Создатели языка C++ посчитали оперирование свободной памяти столь важной задачей для работы программы, что добавили дополнительные операторы new и delete, аналогичные выше рассмотренным функциям. Аргументом оператора new служит выражение, возвращающее количество байтов, необходимых для резервирования. Этот оператор возвращает указатель на начало выделенного блока памяти. Аргументом оператора delete выступает адрес первой ячейки освобождаемого блока. Операторы new и delete являются встроенными компонентами языка, и подключение дополнительных библиотек не требуется.

```
#include <iostream.h>
#define SIZE 512
void main ( )
```

```
{
int *pr_buffer;
pr_buffer=new int[SIZE];
if (pr_buffer ==NULL)    cout << «недостаточно памяти»;
        else             cout  << «память зарезервирована»;
delete (pr_buffer);
}
```

В отличие от функции malloc( ) оператор new автоматически выполняет определение точного размера блока памяти, основываясь на заданном типе объекта. В примере резервируется 512 ячеек типа int.

В качестве аргумента можно использовать как стандартные типы, так и типы, определяемые пользователем.

В случае выделения памяти для массивов оператор new возвращает адрес первого элемента массива:

```
float &i;
i=new float[100];
```

или

```
int n=100;
float *p=new float[100];
```

В этих случаях в динамической памяти отводится непрерывная область, достаточная для размещения 100 элементов вещественного типа, и адрес ее начала записывается в указатель p или ссылку i. Динамические массивы нельзя при создании инициализировать, и они не обнуляются автоматически. Доступ к элементам динамического массива происходит обычным способом.

Преимущество динамических массивов заключается в том, что размерность может быть переменной, т.е. объем памяти, выделяемой под массив, определяется на этапе выполнения программы. При выделении памяти для многомерного массива все размерности, кроме первой, должны быть константами (константными выражениями). Первая размерность может быть задана переменной, значение которой будет известно к моменту использования оператора new. В этом случае указатель в левой части должен иметь правильный тип, например

```
int_ptr = new int[j][4][5];
int (*p)[ ] = new int[2][3];
```

Ссылки

Ссылка это еще одно название для указателя, который не требует разыменовывания при использовании. Разница между указателем и ссылкой заключается в том, что программист может использовать ссылку как обычный объект, несмотря на то, что к объекту будет производиться косвенный доступ, в то время как к указателю необходимо присвоить явно значение адреса объекта.

```
int i = 123;
int *p=&i;
```

Выражение &i является ссылкой на объект с именем i. Через адрес оно ссылается на объект с именем i.

C++ позволяет объявлять ссылочные переменные. Например, объявление &j=i; создает переменную j - ссылку(второе имя) для i .

```
void main(void) {
int i=3;
int
j=2;
…….
}
```

После выполнения присваивания переменная i, также как и j, будет иметь значение 2.

Ссылки безопаснее указателей, поскольку адреса ссылок невозможно переприсваивать. Необходимо помнить что, однажды инициализировав ссылку, ей уже нельзя присвоить другое значение. В отличие от указателей ссылки всегда связаны с объектом. Они используются в качестве переменных, параметров и результатов функций.

Ссылки полезны в функциях, которые возвращают несколько значений. Ведь с помощью оператора return можно возвращать только одно значение. Например, рассмотрим функцию, меняющую между собой значения двух переменных.

```
#include <iostream.h>
void exchange(int &a, int&b)
{ int c=a;
  a=b;
  b=c;
}
void main()
{ int a=100, b=10;
 cout<<"До обмена: a=" <<a<<", b="<<b<<endl;
exchange (a,b);
cout<<" После обмена: a=" <<a<<", b="<<b<<endl;
}
```

Здесь доступ к переменной осуществляется как через ее имя, так и через имя-синоним (ссылку) в вызываемой функции. После завершения вызываемой программы (функции) имя-синоним уничтожается, однако измененное значение переменной в вызываемой функции сохраняется. Таким образом, обращаясь к переменным через ссылки, появляется возможность работать с локальными переменными, как с глобальными. Это позволяет возвращать из функции любое количество значений.

Глава 4. Типы данных, определяемые пользователем

Переименование типов

В C++ можно создавать новые типы данных только на основе уже существующих. Можно приписывать типу данных новое имя, используя ключевое слово typedef. Синтаксис такой процедуры следующий:

typedef <тип><новое имя типа>;

Например,

```
typedef char* string;
```

Введенное таким образом имя можно использовать, как и имя стандартного типа. Тогда объявление вида

```
string name, sign;
```

аналогично объявлению

```
char *name, *sign;
```

На практике задание нового типа используется для облегчения понимания программы.

Перечисляемые типы

Перечисляемый тип представляет собой набор целых чисел, определенный с помощью ключевого слова enum. Каждому числу набора приписывается имя. Синтаксис:

enum <имя типа> { <имя константы>[=<целое значение>], …};

Константы должны быть целыми и могут инициализироваться обычным способом. Указание значения необязательно. По-умолчанию первой в списке константе присваивается значение нуль. Последующие константы принимают значение на единицу большее предыдущего.

Имя типа также необязательно. Оно задается в том случае, если в программе требует-

ся определить переменную данного типа. Тогда компилятор обеспечивает, чтобы эти переменные принимали значение только из списка констант.

Примеры

```
enum Computer_Drives { floppy = 1, harddrivers = 2, cd_room = 4 };
enum Com_Number {first=1, second};    // здесь second равно 2
enum Seasson {Fall, Winter, Spring, Summer};
enum  {two = 2, tree, four, ten = 10, eleven, fifty= ten+ 40 }; // tree равно 3,  four- 4, fifty 50
```

С помощью первых трех объявлений можно задавать переменные. В последнем случае просто перечисляется набор констант.

При использовании арифметических операций перечисления преобразуются к целому типу.

Структуры

Структуры позволяют определять новые типы данных путем логического группирования переменных различных типов. В отличие от массива, все элементы которого одного типа, структура может содержать элементы разных типов. Элементы структуры являются полями. Описание типов полей содержится в структурном шаблоне. Описание структурного шаблона выглядит следующим образом:

```
struct [<имя структуры>] {
                        <имя типа1> <идентификатор1>;
                        <имя типа2> <идентификатор2>;
                        ……..
                    } [список переменных];
```

Например,

```
struct book{
        char title[40];
        char author[30];
        float value;
    };                  //описание структурного шаблона
```

В примере представлено описание структурного шаблона. Структурный шаблон задает тип. После его определения можно смело объявлять переменные, вновь созданного типа. Правила объявления структурной переменной аналогичны правилам объявления простой переменной, т.е. указывается имя типа, за которым следуют имена переменных, разделяемые запятыми.

Например

```
struct book library;            // описание структурной переменной libry типа book
struct book lib1, lib2,*ptb;    //описание двух структурных переменных и указа-
                                 //теля на структурную переменную
struct book library[100]; //объявлен массив из 10 элементов типа book
```

Теперь каждая из объявленных переменных имеет поля title, author и value. При объявлении структурной переменной ключевое слово struct не является обязательным, но очень широко используется.

Можно совмещать в одном объявлении определение структурного шаблона и структурной переменной.

```
struct book{
        char title[40];
        char author[30];
        float value;
    } b1, b2;
```

Объявлены переменные b1 и b2 типа book.

Имя типа структуры можно не задавать. Как правило, это в случае, если предполагается использование структурного шаблона всего один раз. Тогда происходит объединение в

один этап объявления структурного шаблона и структурной переменной. Например,

```
struct { float x, y;
} complex;
```

Элемент структуры может иметь базовый тип, либо быть массивом, указателем или, в свою очередь, структурой. Элемент структуры не может быть типа той структуры, в которую он входит, но он может быть объявлен указателем типа структуры, в которой он содержится.

Идентификаторы полей структуры должны различаться между собой, но идентификаторы различных структур могут совпадать.

Доступ к полям структурной переменной осуществляется через операцию выбора (точка). Например,

```
scanf ( "%f ", &libry. value);
printf( "%s", libry[0]. autor);
```

Как и любую другую переменную, структурную переменную можно инициализировать при объявлении. Тогда значения ее полей перечисляются в фигурных скобках в порядке их описания в шаблоне, через запятую.

```
struct book libry = {"Просто и ясно о Borland C++", "Бруно Бабэ", 400};
```

При инициализации массива структур следует заключать в фигурные скобки каждый элемент массива.

Как и к другим типам, структурам можно давать другое имя с помощью ключа typedef. Например,

```
typedef struct
                    { char name[80];
                    long anct_Num;
                    } custInfo;
```

Объявление переменной этого типа выглядит следующим образом:

```
custInfo newCust;
```

Для переменных одного и того же структурного типа определена операция присваивания. При ее использовании происходит поэлементное копирование.

Пример программы.

Создать структурный шаблон, хранящий сведения о некотором человеке: фамилию, пол, возраст. Создать массив, содержащий сведения о 15 человек. Выбрать из списка фамилии мужчин в возрасте от 20 до 35 лет, причем их фамилия должна начинаться с введенной с клавиатуры буквы.

```
#include <iostream.h>
void main( )
{const int n=15;
 struct man {          //описание структурного шаблона
                    char family[20];
                    char sex;
                    int age;
                 };
 struct man people [n];                    //массив структурных переменных
 int i;
 char symbol;
 for ( i=0; i < n; i+ + )
        {cout<< "введите фамилию: " ;
        cin>> peole[i].family;
        cout<< "введите возраст: ";
        cin>> peole[i].age;
        cout<< "введите пол: м – мужской или ж – женский: ";
        cin>> peole[i].sex;
```

```
        }
    cout<< " введите первую букву фамилии: ";
    cin << symbol;
    for ( i=0; i < n; i++)
        { if  ( peole[i].family[0]= = symbol)
            if (peole[i].sex = = 'м')
                if ( peole[i].age > 19)&&( peole[i].age < 36 )
                    cout<< peole[i].family <<endl;
        }
    }
```

Полем структуры может быть другая структура. В этом случае структура называется иерархической.

```
struct DateType
{
    int month;
    int day;
    int year;
};
struct Statistic
{
    DateType lastServiced;
    int gurantee;
};
struct CarRec
{
    char  descript[50];
    int  number;
    float  cost;
    DateType  dataSale;
    Statistic  history;
};
CarRec Car, Cars[100];
```

Инструкция для обращения к полям внутренних структур строятся слева направо, начиная с имени структурной переменной. Например, Car.dataSale.day – поле day структуры типа DateType, содержащейся в переменной Car типа CarRec.

Объединения

Объединение позволяет запоминать данные различных типов в одном и том же месте памяти. В каждый момент времени объединение может хранить значение только одного типа из набора. Память, которая выделяется переменной типа объединение, определяется размером наиболее длинного из элементов объединения. Все элементы объединения размещаются в одной и той же области памяти с одного и того же адреса. Значение текущего элемента теряется, когда другому элементу объединения присваивается значение.

Синтаксис определения объединения аналогичен определению структуры, с использованием ключевого слова union.

```
union sign { int svar;
             unsigned uvar;
           }number;          //Знаковое или беззнаковое целое

union { char *a, b;     float f[20];
    } jack;
```

Память, выделенная для хранения переменой jack, равна памяти необходимой массиву из 20 элементов типа float.

Объединения используются для экономии памяти в тех случаях, когда известно, что больше одного значения поля одновременно не требуется. Объединения часто используются в качестве поля структуры, при этом в структуру удобно включить дополнительное поле, определяющее, какой именно элемент объединения используется в каждый момент. Тогда имя объединения можно не указывать, что позволяет обращаться непосредственно к его полям. Например,

```
#include <iostream.h>
void main ( ) {
                enum ptype {Card, Check};
                struct {
                        ptype type;
                        union {
                                char card [25];
                                long check;
                                };
                        } info;
// присваивание значения info
switch ( info.type ) {
        case Card: cout<< “оплата по карте”<< info.card; break ;
        case Check: cout<< “оплата чеком”<< info.check; break ; }
}
```

Указатели на структуры

Возможно объявление указателей на структуру или объединение, и тогда для получения доступа к отдельным элементам структуры применяется указатель и оператор –>. Рассмотрим данные возможности на примере.

```
#include <stdio.h>
struct boat {                                   //объявление структурного шаблона
                char model[20];
                char nomer[8];
                int year;
                float price;
                };
int main ( )
{ int i, number;
struct boat boats[30], *ptr_boat;
ptr_boat = &boats[0];
printf («информацию о скольких лодках будем вводить?»);
scanf (“%d”, &number) ;
for ( i=0; i < number; i+ + )
{
printf (“введите модель судна”);
gets(ptr_boat –> model);  // ввод строки
printf (“введите регистрационный номер судна”);
gets(ptr_boat–>nomer);
printf (“введите модель судна”);
scanf(“%d”,&ptr_boat–>year);
printf (“введите стоимость судна”);
scanf(“%f”,&ptr_boat–>price);
ptr_boat+ +;
```

```
}
...
return 0;
}
```

Передача структур в качестве аргументов в функцию

Структуры передаются в функцию по значению, т. е. в функции модифицируются лишь копии исходных данных. В прототипе функции следует указать в качестве параметра структурную переменную. Так при объявлении переменной с использованием структурного шаблона из предыдущего примера, возможно объявление прототипа функции в виде:

```
void print (struct boat boat1);
```

Указывать ключевое слово struct необязательно. Если же вместо самой структуры передавать указатель на нее, то это может значительно ускорить выполнение программы.

```
void print (struct boat *boats);
```

Таким же способом происходит и передача массива структур в функцию.

Глава 5. РАБОТА С ФАЙЛАМИ

Во всех примерах рассмотренных ранее предполагалось, что ввод в программу осуществляется с клавиатуры, а вывод направляется на экран монитора.

В случаях использования большого объема вводимых и выводимых данных используют файлы.

Файлом называется именованная область внешней памяти, в которой содержится некоторая информация. Хранение информации в файле позволяет не вводить заново данные при повторном запуске программы, а также просматривать данные сколько угодно раз. Содержимое файла может быть просто просмотрено на экране или напечатано на принтере.

Язык С++ позволяет работать с файлами несколькими способами.

Текстовые и бинарные файлы

Первый из предлагаемых способов применялся в языке С и поддерживается в С++. Язык С выделяет 2 вида файлов: текстовые и двоичные (бинарные). Текстовым считается файл, в котором информация запоминается в виде символов кода ASCII (или аналогичном). Он отличается от бинарного файла, который обычно используется для запоминания кодов машинного языка. Таким образом, содержимое текстового файла можно просмотреть и изменить в любом текстовом редакторе. Для чтения бинарного файла необходима специальная программа.

Для работы с любым видом файла описываем указатель на переменную типа file (иногда используют термин файловая переменная):

```
file *in;
```

Открытие файла происходит с помощью функции fopen( ). Данной функцией управляют три основных параметра.

Имя файла, который следует открыть, является первым аргументом.

Второй аргумент, указывающий режим использования файла, может принимать три значения: “r” – файл используется для чтения, “a” – для дополнения, “w” – для записи (при применении “r” используется существующий файл, для двух других, если файл не существует, то он будет создан; если используется “w” для уже существующего файла, старая версия файла затирается);

Третий параметр является указателем на файл, и это значение возвращается функцией. Например,

```
file *in;
in = fopen( “test”, ”r” );
```

Теперь in является указателем на файл, хранящийся на диске под именем test. С этого

момента программа будет работать с файлом через in, а не по имени test. Если fopen( ) не способна открыть требуемый файл, то она возвращает значение 'NULL' (определенное в файле stdio.h как 0).

Рекомендуется использовать проверять существование файл перед его открытием, например, строкой вида

if ((in = fopen("test", "r")) != NULL)

Если файл создать невозможно, то указателю in присваивается значение NULL, и условие становится ложным. В этом случае нужно вывести на экран предупреждающее сообщение и завершить программу.

Закрыть файл проще. Для этого используют функцию fclose( ), которая имеет только один аргумент – указатель на файл.

fclose(in) ;

Ввод текстового файла осуществляется с помощью функций getc( ) или getch( ), работающих с одним символом. Различие этих функций в том, что первая работает с целым типом, а вторая – с типом char. Функции putc( ) и putch( ), соответственно записывают один символ. Прототипы эти функций содержатся в заголовочном файле stdio.h.

```
//Программа печатает содержимое файла на экран и определяет  количество точек
#include < stdio.h >
void main( )
{
file *in;  // описание указателя на файл
char ch, point = ' . ';
int count=0;
if ( ( in = fopen( "test", "rt" )) != NULL )
        {  while ( (ch = getch(in) ) != EOF)          // получает символ из in
              { putch(ch, stdout); //посылает на stdout (стандартный вывод - экран)
                if (ch = = point) count++;
              }
        fclose(in);
        }
else printf("файл не открывается /n");
}
```

При работе с файлом используются понятия начало и конец файла. Когда файл открывается, система помещает файловый указатель на начало файла – его первый элемент. При чтении информации в систему передается первый элемент, и происходит сдвиг указателя на следующий элемент. Таким образом, работа с файлом походит на работу с массивом, размер которого заранее не известен. Указатель перемещается до тех пор, пока не будет достигнут конец файла. Конец файла помечается константой EOF (End of File), которая автоматически формируется при закрытии файла, после записи в него информации. Данная константа возвращается функцией feof( ) в случае достижения конца файла. Ее параметром служит указатель на файл. Следовательно, задание цикла чтения до конца файла может быть записано как:

while (!feof(in)) { … }

Для указания, что файл открыт в двоичном режиме необходимо добавить букву b во второй аргумент функции fopen() (для текстового добавляется t , но это не является обязательным).

Для работы с бинарными файлами существует ряд функций: fread (buffer, size, count,stream) содержит 4 аргумента.Данная функция читает count элементов длины size из входного потока (файла) stream (FILE * stream) и помещает в заданный массив buffer. При этом значение указателя файла увеличивается на число действительно прочитанных байтов.

fwrite ( ) позволяет записывать в файл и имеет такие же аргументы. Функция дописы-

вает count записей по size байтов каждый из области buffer в выходной поток stream.

fseek (stream, offset, origin) перемещает внутренний указатель файла, связанный с потоком stream, на новое место в файле, которое вычисляется по смещению offset и указанию направления отсчета origin. После использования fseek( ) следующая операция ввода/вывода с указанным потоком stream будет выполнена, начиная с той позиции, на которую произведено перемещение. Аргумент offset должен быть типа long, а origin может принимать значение одной из следующих целочисленных констант:

SEEK_SET (значение 0) – начало файла,

SEEK_CUR (значение 1) – текущая позиция указателя файла,

SEEK_END (значение 2) – конец файла.

Функция fseek( ) возвращает целое значение.

Для текстового режима работы с файлом можно также использовать fseek(), но значение аргумента offset должно быть получено с помощью функции ftell() или равняться нулю.

long ftell (stream);

file *stream;

Функция позволяет получить текущую позицию указателя файла, связанного с потоком stream. Позиция задается как смещение относительно начала файла.

Запись и чтения файла осуществляется поэлементно. Элементом файла может быть символ, или целое число, или даже структурная переменная.

Если текстовый файл можно создать с помощью текстового редактора, например WordPad или среды Borland C++, то бинарный файл нужно создавать программно. Так для хранения координат точек на плоскости можно создать файл следующим образом:

```
#include<stdio.h>
#include<stdlib.h>
void main ( )
{
struct point                    //шаблон и переменная для хранения координаты
{       int x, y;
        }z;
FILE *f;
f = fopen( “point .bbb”, ”wb”);
randomize( );
int k=1;
while ( !k )        { printf (“Задать координаты точки? Если да - 1, иначе 0 \n”);
                      z. x= random(100) – 50;
                      z. y = random(100) – 50;
                      fwrite (&z, sizeof(point), 1, f);
        }
fclose(f);
}
```

Таким образом, на диске в текущей директории создан файл с именем point.bbb, и теперь обрабатывать хранящиеся в нем данные. Требуется отметить, что в отличие от массива при работе с файлом нужно помнить, что количество элементов файла неизвестно. Например, в заданном файле определим координат точек, принадлежащих прямой, заданной уравнением $y = kx + b$. Значения констант k, b задаются с клавиатуры.

```
#include<stdio.h>
void main ( )
{
struct point
{       int x, y;
        }z;
int k, b, count=0;
```

```
printf (“Введите параметры уравнения”);
scanf ( “%d   %d  ”, &k, &b);
FILE *f;
f = fopen( “point .bbb” ,”rb” );
while ( !feof(f) ) {
                    fread (&z, sizeof(point), 1, f);
                    if ( z.y = = k*z.x + b )   { count++;
                                                 printf (“точка (%d, %d) лежит на прямой, z.x, z.y);
                                                }
                   }
fclose(f);
printf ( “количество найденных точек  %d, count);
}
```

Файловый ввод/вывод с применением потоков С + +

Библиотека С++ содержит три класса с помощью которых можно управлять файловым вводом-выводом:

ifstream	подключает к программе файл, предназначенный для ввода данных (входной файловый поток)
ofstream	подключает к программе файл, предназначенный для вывода данных (выходной файловый поток)
fstream	подключает к программе файл, предназначенный как для ввода, так и для вывода

Для подключения данных классов необходимо подключить файл ifstream.h. Для создания объекта класса ifstream и связи с ним файла, находящегося в текущем каталоге, требуется записать следующее:

```
ifstream f (“text.txt”, ios::in);
```

Итак, создан объект с именем ifsin класса ifstream. С ним связан файл text.txt. Если файл, с которым связан объект, находится не в текущем каталоге необходимо полностью указать путь его расположения. Вторым аргументом указывается один из следующих флагов:

Флаг	***Назначение***
ios :: in	Файл открывается для чтения, его содержимое не открывается
ios :: out	Файл открывается для записи
ios :: ate	После создания объекта маркер текущей позиции устанавливается в конец файла
ios :: app	Все выводимые данные добавляются в конец файла
ios :: trunc	Если файл существует, его содержимое очищается автоматически

```
/* Программа, демонстрирующая создание потоков ifstream и ofstream для обмена  данными с файлом*/
#include <fstream. h>
#include <iostream. h>
void main ( )
{
char ch;
ifstream fin (“text.txt”, ios :: in);
if   (!fin) cout<<” Невозможно открыть файл”<<endl;
ofstream fout (“text1.txt”, ios :: out);
if   (!fout ) cout<<” Невозможно открыть файл”<<endl;
while (fout && fin.get(ch) );
fout.put(ch);
fin. close( );
```

```
fout. close( );
}
```

/* Программа считывает содержимое текстового файла и выделяет после нажатия клавиши пробела десятое слово и третье предложение */

```
#include <fstream. h>
#include <iostream. h>
#include <conio.h>
void main ( )
{
char ch, probel=32, tochka=46, escape=27, dir[20];
int j, k=0, t=0;
ifstream ffile (“my_file.txt”, ios :: in);
while (ffile)
        { textcolor (WHITE);
              ffile.get(ch);            // посимвольный вывод из файла
               cout<<ch;
            }
probel = getch( );
  clscr ( );
  ifstream ffile (“my_file. txt”, ios :: in);
  while (ffile)
        {     file1.get(ch);          // посимвольный вывод из файла
              if (ch= = probel) k++;
            if (ch= = tochka) t++;
                  if ( t= =2)|| (k= =9)
                        {if ( t= =2) textcolor (RED);
                          if ( k= =9) textcolor (GREEN); }
                  else textcolor (WHITE);
                  cout<<ch;
            }
  escape = getch( );
  }
```

Глава 6. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

В основе объектно-ориентированного программирования (ООП) лежит идея разбиения задачи на группу объектов. Объекты содержат в себе данные и подпрограммы для их обработки. Данные и стандартные подпрограммы, обрабатывающие данные, объединяются в один блок (тип), называемый *классом*. При необходимости можно получить доступ к данным класса, но при этом используются только подпрограмма данного класса.

Достоинства объектно-ориентированных программ:

- хорошая структурированность, что облегчает понимание алгоритма программы;
- возможность их разбиения на небольшие компоненты и тестирования отдельных компонент;
- легкость расширения.

Три ключевых принципа определяют объектно-ориентированное программирование:

- инкапсуляция;
- наследование;
- полиморфизм.

Инкапсуляция (сокрытие информации) – определение пользователем новых типов данных. Каждый такой тип содержит определение набора значений и операций, которые

могут быть выполнены над этими значениями, и образует так называемый абстрактный тип данных. При этом никакие другие функции, кроме набора определенных в классе операций, не должны иметь доступа к значениям этого типа. В языке С++ инкапсуляция данных реализуется с помощью механизма классов. Переменные класса являются объектами, из которых строится программа.

Наследование – механизм, позволяющий строить иерархию типов. Это предполагает определение базового типа (или прародителя), а затем использование этого типа для построения производных типов (потомков). Причем каждый производный тип наследует все свойства базового типа, включая как данные, так и набор операций (функции). Новые типы данных могут также иметь свои дополнительные свойства. Таким образом, они не дублируют свойства базового класса, а только имеют возможность их использовать.

Полиморфизм в переводе с греческого означает «много форм», заключается в обозначении общего действия одним именем (функцией), которое используется во всей иерархии типов. Каждый тип в этой иерархии реализует это действие своим собственным, пригодным для него способом. В языке С++ полиморфизм имеет две формы:

- перегрузка операций и функций,
- использование виртуальных функций.

Понятие «класс»

Класс является типом данных, определяемый пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними. Класс есть расширение понятие структуры языка С++. Каждый представитель класса называется объектом.

Класс является типом, определяемым пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с этими данными. Объединение данных и функций, которые обрабатывают объект определенного типа, называется инкапсуляцией.

Данные класса называются полями или данными-членами класса. Функции класса – методами или функциями-членами класса. Поля и методы называются элементами класса. Описание класса выглядит примерно так:

```
class <имя> {
[ private: ]
< описание скрытых элементов>
[ public: ]
< описание доступных элементов>
[ protected: ]
< описание защищенных элементов>
};
```

Описанию элементов предшествуют ключевые слова, являющиеся спецификаторами доступа: private (закрытый), public(открытый) и protected (защищенный).

Открытые элементы (public) доступны снаружи класса. Они, как правило, отвечают за внешний интерфейс класса и являются методами класса.

Закрытые элементы (private) предназначены только для внутреннего использования в классе. Их используют для полей класса, после чего они становятся доступным только для методов класса, в состав которого они входят.

Защищенные элементы (protected**)** доступны для методов данного класса и классов, производных от него.

При создании класса программист сам должен решать, какие из членов класса будут общедоступными, а какие закрытыми. Однако, большинство классов имеет типичную схему: закрытая часть содержит данные, а общедоступная – функции для работы с этими данными.

В классе могут присутствовать многочисленные открытые и закрытые секции. Можно повторять в объявлении класса ключевые слова public и protected столько раз, сколько

понадобится и в любом порядке.

Спецификатор доступа не является обязательным, и если он не используется, по умолчанию элементы класса становятся закрытыми.

Для полей класса справедливы следующие правила:

- они могут быть любого типа, кроме типа того же класса (но могут быть ссылками или указателями на класс, в котором объявлены);
- поля могут быть объявлены с модификаторами const, но при этом они принимают значение только один раз (с помощью конструктора) и не могут изменяться;
- они могут быть описаны с модификаторами static, но не auto, extern или register;
- инициализация полей при объявлении не допускается.

Классы, определяемые программистом, похожи на встроенные типы. Объект класса это переменная типа «класс». Можно объявлять сколько угодно объектов класса, причем синтаксис их объявления аналогичен объявлению переменной любого другого типа.

[class] <идентификатор класса> <идентификатор переменной>;

Можно передавать объекты в качестве параметров функций или возвращать их как значение функции; объявлять массивы, состоящие из объектов класса.

Как и любая переменная, объект класса может быть динамическим (т.е. создаваться каждый раз, когда управление достигает его объявления, и уничтожаться, когда управление выходит из данного блока) или статическим (т.е. создаваться один раз и уничтожаться по завершению программы).

С другой стороны, язык C++ обрабатывает классы иначе, чем встроенные типы. Большинство встроенных операций не могут применяться к классам. Нельзя использовать операцию сложения – «+» для двух объектов, или оператор = = для их сравнения. Но все эти операции применимы для полей классов. Для самих объектов справедливы следующие встроенные операции: выбор элемента (.), присваивание (=), получение адреса (&), косвенная адресация (*), и операция ->.

Рассмотрим класс, созданный для хранения даты и времени в классе.

```
class TTime {
        private:
                int year,  month, day, hour, minute;
        public:
                  void Dispay( );
                  void SetTime(int d, int m, int y, int hr, int min);
};
```

В рассмотренном примере класс TTime имеет семь элементов класса: пять полей – year, math, day, hour, minute и два метода класса - Dispay( ) и SetTime( ).

Методы класса объявлены прототипами функций. Предположительно, они выполняют какие-то действия над полями. Их тела будут описаны позднее. Тогда их описанию должны обязательно предшествовать имя класса и операция разрешения видимости (::).

```
// Определение функций-членов
void TTime::SetTime(int d, int m, int y, int hr, int min)  {
          day = d;
          month = m;
          year = y;
          minute = min;
          hour = hr;
};
void TTime::Display( ) {
 char s[40];
printf ( "Дата: %2d. %2d. %4d  Время: %2d:%2d ",  day, month, year, hour, minute);
};
```

Тело метода может также содержаться внутри определения класса. В этом случае

функция называется встроенной (inline) функцией-элементом.

Выше рассмотренный пример показывает применение невстроенных методов.

Можно определить встроенную функцию-элемент и вне тела класса, указав в заголовке определения ключевое слово inline.

Существуют отличия между элементами класса и обычными функциями.

1. При описании имени метода предшествует имя класса и оператор разрешения области видимости **::.** Имя класса однозначно определяет имя метода, поэтому в программе могут быть другие функции с такими же именами.
2. Внутри метода операторы имеют прямой доступ к членам класса.
3. Функция main( ) использует класс как и любой другой тип. Для использования метода, требуется создать объект его типа, чтобы использовать метод класса.

Доступ к элементам объекта аналогичен доступу к полям структуры. Для этого используются операции точка (выбора) при обращении к элементу через имя объекта и операция -> при обращении через указатель. Обратиться таким образом можно только к элементам со спецификатором доступа public. Получить или изменить значения элементов со спецификатором private можно только через обращение к соответствующим методам.

Так для применение использование методов класса TTime возможно в функции main( ) следующим образом.

```
  void main ( )
{
    TTime day;                    //объявление объекта day класса TTime
    day.SetTime( 24, 2, 1996, 4, 20);
    day.Display( );
};
```

Возможно определение указателя на функцию-элемент класса.

Синтаксис его определения:

возвращаемый_тип (имя_класса : : *имя_указателя)(параметры)

Методы класса могут вызывать другие функции-элементы того же класса, используя имя функции. Обычно функции или функции-элементы других классов могут вызывать элементы класса с помощью операций . и ->, применяемых представителю или указателю на представитель класса.

Пример использования указателей при работе с объектами класса.

```
class Coord {
                int x, y;
public:
                void SetCoord(int _x, int _y) {x = _x; y = _y};
                void GetCoord(int & _x, int & _y) {_x = x; _y = y};
};
int main(void)
{
        Coord org;                          // локальный объект
        Coord *orgPtr = &org;    //создать указатель на объект
        org.x = 0;
        orgPtr->y = 0;
        org.SetCoord(10, 10);
        int col, row;
        orgPtr->GetCoord(col, row);
        return 0;
       };
```

В качестве аргументов функций можно использовать указатели на функции, например:

```
void CallMemeberPtr( void ( A : : *funcPtr )( ) );
```

//Функция, вызывающая функцию-элемент, адрес которой передается как параметр

В С++ структура, класс и объединение рассматриваются как типы данных. Структура и класс похожи друг на друга, но в структуре и объединении элементы имеют по умолчанию доступ public, а в классе private.

Конструктор

С++ имеет две встроенных особенности – конструкторы и деструкторы – помогающие не забывать про инициализацию объектов и про очистку памяти после завершения работы с объектом.

Конструктор – функция, которая автоматически вызывается при создании объекта (инициализации класса).

Внутрь конструкторов могут быть помещены процедуры инициализации для установки необходимых значений полей до использования объекта. Может быть объявлено сразу несколько конструкторов в одном классе для того, чтобы инициализировать объекты поразному.

По завершению работы с объектом автоматически вызывается функция, называемая деструктором.

Использование конструкторов и деструкторов необходимо для избежания массы досадных ошибок.

Конструктор является методом класса с тем же именем, что и сам класс. Он вызывается автоматически компилятором при создании представителя класса.

Если конструктор в программе не определен, то компилятор автоматически генерирует конструктор без параметров, т.е. конструктор умолчанию.

Для конструкторов выполняется следующие правила:

конструктор не возвращает значение, поэтому в его описании возвращаемый тип не указывается, причем даже void;

возможно объявление несколько конструкторов с разными наборами параметров для разных видов инициализации;

конструктор, вызываемый без параметров, называется конструктором по умолчанию;

конструктор не наследуется;

конструктор не может быть объявлен как const, volatiler, virtual или static.

Синтаксис объявления конструктора аналогичен синтаксису объявления любого другого метода класса. Например,

```
class XYValue {
                int x, y;
        public:
                XYValue (int X = 100, int Y = 10) { x=X;  y=Y; }
. . .
 };
```

Примерами вызова конструкторов для объектов данного класса могут служить следующие конструкции:

```
XYValue S( 200, 300), M(50), Z;
// Создаются три объекта. Значения не указанных параметров устанавливаются по умолчанию
```

Таким образом, конструктор вызывается, если в программе встретилась какая-либо из следующих конструкций:

```
 <имя класса>  <имя объекта> [(список параметров)];
// список параметров не должен быть пустым
<имя класса>  (список параметров);
// создается объект без имени (список может быть пустым)
<имя класса>  <имя объекта> = <выражение>;
// создается объект без имени и копируется
```

Пример класса с несколькими конструкторами.

```
enum color {white, black, auburn, spotted};
class kitten {
        int age;
        color skin;
        char *name;
        public:
        kitten ( int a=1);  //конструктор по умолчанию
        kitten ( color sk);
        kitten (char *nam);
        int get_ammo ( ) {return ammo};
        };
        …
        kitten :: kitten ( int  a)
        { age=a; skin=black; name=0;}

        kitten :: kitten  ( color sk)
{ switch (sk) {
                case black :      age=10; skin=black;  strcpy (name, “черныш” );   break;
                case white :      age=2; skin=white; strcpy (name, “снежок” ); break;
                case auburn :     age=3;  skin=auburn; strcpy (name, “рыжик” );  break;
                case spotted : age=5; skin=spotted; strcpy (name, “ ” );  break;
              }
 }
kitten:: kitten ( char *nam)
{ name = new char [strlen(nam) +1];
// к длине строки добавляется 1 для хранения нуль-символа
// strlen( ) – определяет длину строки
strcpy (name, nam); // копирует содержимое одной строки в другую
cout<< “введите возраст:”;
cin>> a;
cout<< “окрас”;
cin>>skin; }
…
```

При использовании нескольких конструкторов в одном классе необходимо помнить, что они должны отличаться набором параметров, каждый из них должен иметь уникальный набор типов аргументов, иными словами, каждый из них должен иметь уникальную сигнатуру. Кроме того, если определение конструктора содержит список инициализации элементов, то список может определяться от заголовка определения функции двоеточием. В этом случае поля перечисляются через запятую. Для каждого поля в скобках указывается инициализирующее значение, которое может быть выражением. Без этого способа не обойтись при инициализации полей-констант, полей-ссылок, полей-объектов. Пример использования инициализации полей в конструкторе.

```
kitten :: kitten ( color c) :  age(1), skin(c),  name(0) { };
```

Деструктор

Деструктор является дополнением конструктора. Он имеет то же имя, что и класс, но ему предшествует префикс-тильда (~). Деструктор вызывается всякий раз, когда уничтожается представитель класса. Для деструктора выполняются те же правила, что и для конструктора. Выделение динамической памяти одна из важных функций конструктора. Память, выделенная для динамического объекта в конструкторе, освобождается при удалении объекта - в деструкторе.

Примером деструктора для рассмотренного класса kitten является

```
kitten :: ~ kitten( ) {delete [ ] name;}
```
Пример простейшего деструктора:
```
# include <iostream.h>
 class Pair{
                    int first, second;
  public:
         Pair ( int one, int two): first(one), second (two)
                    {cout<< " объект создан "<< endl;}
                     ~Pair ( ) { cout <<" объект удален "<< endl;}
                    void out ( ) { cout<< first<< " << second; }
  }
void main( )  {
 Pair num(2,3); num. out( );
 Pair num(4,5); num. out( ) }
```

Указатель this

Иногда внутри метода требуется обратиться к указателю на объект. Это можно сделать через ключевое слово this. У каждого объекта указатель this свой. Он содержит в себе адрес объекта, полем которого он является.

Указатель this называют неявным указателем. Он автоматически создается компилятором. Каждый объект класса имеет свою копию полей класса. Методы же класса существуют только в одном экземпляре.

При вызове метода ему передается неявный аргумент, который обозначает конкретный объект класса. Неявный указатель this можно использовать и явно для работы с полями объекта, как и любой другой указатель.

```
class Simple{
                    int a;
public:
                    Simple( );
                    void Greet( ) { printf("Hello!\n"); }
        };
  Simple::Simple( ) {
                    this->a=15;
                    Greet( );
                    (*this).Greet;              //оба оператора вызывают функцию Greet
  };
```

Конструктор копии

По числу параметров и их назначению конструкторы делятся на 4 группы: конструкторы по умолчанию,
конструкторы с параметрами,
конструкторы копирования,
конструкторы преобразования типов.

Первые две группы рассмотрены ранее: конструкторы по умолчанию не имеют параметров и в них можно выполнять только самые простые действия, например задание начальных значений переменных и выделение динамической памяти. Конструкторы с параметрами применяются в случаях, когда необходимо вручную установить начальные параметры полей объекта.

Конструктор копии воспринимает в качестве аргумента только один параметр, тип которого константная ссылка на объект класса (const тип_класса &) или простую ссылку на объект (тип_имени &). Например:

```
class Coord {
                    int x,y;
```

```
public:
                Coord(const Coord & scr);              // Конструктор копии
  };

Coord::Coord(const Coord &scr) {
        x = scr.x;        y = scr.y;
};
```

Ссылка передается каждый раз, когда новый объект инициализируется значениями уже существующего объекта.

Существует операция присваивания, которая присваивает объекту значение другого объекта. Операция присваивания является функцией-членом с именем operator=, который воспринимает единственный аргумент типа const тип_класса & или тип_класа &.

```
class Coord {
                int x,y;
      public:
              Coord& operator = (const Coord &scr);
               };
Coord& Coord::operator = (const Coord &scr) {
        x = scr.x;
        y = scr.y;
        return *this;
        };
```

Конструктор преобразования типов – конструктор с параметрами, которому передается только один параметр, не совпадающий с типом класса, к которому принадлежит конструктор.

Наследование

Простое наследование описывает родство между двумя класса. Класс, являющийся прародителем называется базовым классом (родителем). Класс, созданный из базового класса называется производным классом (потомком). Производный класс наследует все элементы базового класса и может обладать новыми элементами, свойственными только ему.

Из одного базового класса могут выводиться многие классы. Производный класс сам может в свою очередь быть базовым, свойства которого наследуют другие классы. Таким способом создается иерархия класса.

Синтаксис механизма наследования следующий:

class <идентификатор>: [ключ доступа] <идентификатор базового класса>
{тело класса};

Разница между использованием различных ключей доступа при объявлении наследования представлена в таблице:

Ключ доступа	Спецификатор в базовом классе	Ключ доступа в производном классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

Таким образом, закрытые элементы базового класса в производном классе недоступны вне зависимости от ключа. Обращение к ним осуществляется только через методы базового класса.

Элементы protected при наследовании с ключом private становятся в производном классе private, а в остальных случаях доступ к ним не изменяется.

Доступ к открытым элементам при наследовании становится соответственным ключу доступа. Указание ключа доступа необязательно, по умолчанию для классов используется ключ доступа private.

Производный класс наследует из базового класса поля и методы, а также деструктор, но не конструкторы и операции присваивания.

В общем случае производный класс наследует все данные и функции своих предков, например:

```
class Base {
private:
        int count;                              // поле класса
public:
Base( ) { count = 0; }                  // определение конструктора
void SetCount(int n) { count = n; }     // определение метода
int GetCount( ) { return count; }  // определение метода
};
```

Если требуется класс, имеющий все свойства Base, но, дополнительно, обладающий способностью изменения значения поля объекта на заданную величину. Тогда можно объявить производный класс в виде:

```
class TDerived: public Base {
public:
Derived( ):Base( ){  };
void ChangeCount(int n) {SetCount( GetCount( ) + n ) };
};
```

Порядок вызова конструктора в производном классе определяется следующими правилами. Если в конструкторе производного класса явный вызов конструктора базового класса отсутствует, автоматически вызывается конструктор базового класса по умолчанию (то есть без параметров). Если конструктор базового класса требует указания параметров, он должен быть вызван явным образом в конструкторе производного класса в списке инициализации.

Для иерархии классов, состоящей из нескольких уровней, конструкторы базовых классов вызываются, начиная с самого верхнего уровня. После выполняются конструкторы элементов классов, которые являются объектами, в порядке их объявления в классе, а потом - конструктор класса.

В случаях нескольких базовых классов их конструкторы вызываются в порядке объявления.

Вызов функций базового класса предпочтительнее копирования фрагментов кода из функций базового класса в функции производного. Кроме сокращения объема кода, этим достигается упрощение модификации программы.

Правила вызова деструкторов в производном классе:

Если в производном классе не объявлен деструктор, он формируется по умолчанию и вызывает деструкторы всех базовых классов.
В отличие от конструкторов, при написании деструктора в производном классе не требуется явного вызова деструкторов базовых классов.

Для иерархии классов, состоящей из нескольких уровней, деструкторы вызываются в порядке, строго обратном вызову конструкторов: сначала вызывается деструктор класса, затем – деструкторы элементов класса, а потом деструктор базового класса. Например,

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main( ) {
 /* Класс прямоугольник, содержащий конструктор, деструктор, функцию
нахождения площади */
 class pryamoug          {
  protected:       float x, y;
  public:
                   pryamoug( ) {                                    // конструктор по умолчанию
                   cout<< “Введите ширину и длину прямоугольника”;
                   cin<<x<<y;
                   }
                   pryamoug ( float xx, float yy) { x=xx; y=yy; }   //конструктор с параметрами
                   ~pryamoug( ) { cout<< “треугольник уничтожен”; }       //деструктор
                   float S( ){ return (x*y); }                      // метод
};
  /* производный класс пирамида, определяемый основанием и высотой и содержащий функции нахождения площади поверхности и объема пирамиды*/
 class piramida : public pryamoug {
              float h, s;
  public:   piramida( float xx, float yy, float hh ) : pryamoug ( xx, yy )          //конструктор
              { h = hh; s = pryamoug :: S( ); }
              float pover( )   {                          // нахождение площади поверхности
              float p, a;
              p=2*( x + y );
             a=sqrt ( h*h + sqrt( ( x / 2 )*( x / 2 )+( y / 2 )*( y / 2 ) ) );
             return ( p*a / 2 );
            }
             float V( ) { return ( s*h / 3); }                  //нахождение объема
};
float  x1, y1, h1;
cout<<"Введите ширину основания";
cin>>x1;
cout<<"Введите длину основания";
cin>>y1;
cout<<"Введите высоту пирамиды";
cin>>h1;
pryamoug A(x1, y1);
piramida B(x1, y1, h1);
printf("Площадь основания пирамиды %4d \n",A.S( ));
printf("Площадь поверхности пирамиды %4.2f \n", B.pover( ) );
printf("Объем пирамиды %3.0f \n",B.V());
getch( );
```

```
}
```

При желании можно изменить поведение унаследованного объекта. То есть переопределить метод базового класса в производном классе. Это называется *замещением.*

Если элемент производного класса имеет то же имя, что и элемент базового класса, для объекта производного класса используется представитель производного класса, а для объекта базового класса – метод базового. Например,

```
# include<iostream.h>
# include<conio.h>
class Base {                          // базовый класс
 protected:       int Price;
public:           void Print Me( );
};
void Base:: Print Me( )   { cout<<" Base"<<Price<<"\n";}

class Derived: public Base        // Derived унаследован от Base
{  public:    void Print Me( );       //Print Me( ) другая ( замещенная )
};
 void Derived:: Print Me( )
{  cout<<"Derived"<<Price<<"\n";
    Base::Print Me( );                 // вызываем метод базового класса
}
void main( ) {
   Base Bclass;
   Derived CopyClass;
   Bclass. Price=1;
   CopyClass. Price=7;
   Bclass. Print Me( );
   CopyClass. Print Me( );
   cout<<" \Press any key to end ";
   while (!kbhit( ));
}
```

Множественное наследование

Множественное наследование позволяет создание нового класса из нескольких базовых. При этом в объявлении производного класса после его имени следует перечислить через запятую имена всех базовых классов с соответствующими ключами доступа. Например:

```
class D: public A, public B, public C {… };
```

Можно использовать спецификаторы доступа и вперемешку. Множественное наследование отличается от простого лишь тем, что производный класс наследует все свойства всех перечисленных классов.

Также как и при простом наследовании, при множественном наследовании необходимо инициализировать переменные базовых классов. Для этого в конструкторе производного класса необходимо инициировать конструкторы базовых классов их перечислением после двоеточия через запятую.

```
D( ): A( ), B( ), C( ){. . .};
```

В случае конфликта имен элементов базовых классов следует использовать оператор видимости. То есть, если в базовых классах есть одноименные элементы, при этом может произойти конфликт идентификаторов, который устраняется с помощью операции доступа к области видимости:

```
class A { public : int print( );
. . .
};
```

```
class B { public : int print( );
. . .
};
class C: public A, public B {
. . .
};
void main ( )
{ C object;
object. A::print( );
object. B::print( );
}
```
Использование в этом случае обычного вызова метода класса object. print( ); приведет к ошибке.
```
# include<iostream.h>
#include<iomanip.h>
# include<conio.h>
class Person {                          // базовый класс
protected:          char name[20];
                    int age;
                    char sex;
public:  void Get_info( );
                    void Print( );
};
class Academics {
protected:          char course_name[20];
                    int semester;
public:              int Get_info( );
                    void Print( );
};
class Student: public Person, public Academics {
private:  float amount;
public:              int Get_info( );
                     void Print( );
};
void Person :: Get_info( ) {
cout<< "Имя?  "; cin>>name;
cout<< "Возраст?"; cin>>age;
cout<< "Пол ( М / Ж) ?; cin>>sex;
}
void Person :: Print ( ) {
cout<< name<< "/ t " age<< "/ t "<< sex << "/ t ";
}
void  Academics :: Get_info( ) {
cout<< "Шифр специальности ( АСОИУ/ ИС / ПМ …)?  ";
cin>>course_name;
cout<< "Семестр(1 / 2/ 3 … )?";
cin>>semester;
}
void Academics :: Print ( ) {
cout<< course_name<< "/ t " << semester << "/ t ";
}
void Student :: Get_info( ) {
```

```
Person :: Get_info( );
Academics :: Get_info( );
cout<< “Оплата ( тыс. руб.) ?;   cin>>amount;
}
void Person :: Print ( ) {
Person :: Print ( );
Academics :: Print ( );
 cout<< setw(4)<< amount<<endl;
}
void main( )
{
clrscr( );
const n=10;
Student object[n];
cout << “Введите следующую информацию: ”;
for ( int k=0; k<n; k++)
{
        object[ k ]. Get_info( );
        object[ k ]. Print( );
}
```

Перегрузка функций и операций

В языке C++ допускается использование перегруженных функций. Под перегрузкой функций (не обязательно методов класса) понимается создание нескольких прототипов функции, имеющих одинаковое имя. Компилятор различает их по набору аргументов. Перегруженные функции оказываются весьма полезными, когда одну и ту же операцию необходимо выполнить над аргументами различных типов. Какую именно функцию требуется вызвать, компилятор определяет по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого значения в разрешении перегрузки не участвует.

Например, пусть необходимо создать функцию для подсчета суммы элементов массивов различного типа int и float

```
#include <iostream.h>
int summa (int iarray[ ] );                    // прототип функции для массива типа int
float summa (float farray[ ] );      // прототип функции для массива типа float
int main ( ) {
int iarray[5] = {1, 6, 0, 4, 3 };
float farray[5] = { 6.7, 8.0, 3.9, 7.6, 0.4 };
int isum;
float fsum;
isum=summa(iarray);
cout<<”сумма массива целых чисел равна”<<isum<<endl;
fsum=summa(farray);
cout<<”сумма массива дробных чисел равна”<<fsum<<endl;
return 0;
}

int summa (int array[ ])
{ int i, s=0;
for (i=0; i<5;i++)
s+=iarray[i];
return s;
```

```
}
float summa (float array[])
{ int i;
float s=0;
for (i=0; i<5;i++) s+=iarray[i];
return s;
}
```

При использовании перегруженных функций необходимо помнить, что если функции отличаются только типом возвращаемого значения, а не типами аргументов, такие функции не могут носить одинаковое имя. Функции не могут быть перегружены, если их параметром является ссылка.

Разрешается осуществлять не только перегрузку функций, но и операторов. В большинстве языков программирования реализована концепция перегрузки операторов, пусть и в неявном виде. Так, например, оператор суммирования позволяет складывать значения разных типов.

В создаваемом классе можно изменить свойства большинства стандартных операторов, таких как +, -, *, /, заставив их работать не только с данными базовых типов, но и также с объектами.

Перегружать явным образом можно большинство операций, за исключением: . ?: # ## sizeof.

Перегрузка осуществляется с помощью методов специального типа (функций-операций). Синтаксис функции-операции:

<тип> operator <операция> (список параметров) {тело функции}

Например,

```
class angle {
                int degree, minites, seconds;
public:  angle_value (char *);
                int operator > ( angle &a)                    // перегруженный оператор
   { if (3600*degree + 60*minites + seconds>3600*a.degree + 60*a.minites + a.seconds)
                return 1;
   else return 0;
  }
};
```

На перегрузку операторов накладываются следующие ограничения:

- сохраняются количество аргументов, приоритет оператора и порядок группировки его операндов, используемые в стандартных типах данных;
- невозможно изменить синтаксис оператора;
- смысл стандартного оператора применительно к базовым классам переопределять нельзя;
- невозможно создавать новые операторы,
- функции-операции не наследуются;
- функции-операции не могут быть объявлены как static.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
const size = 80;
class Phrase
{
private:
        char str[size];
public:
        Phrase ( ) { strcpy (str, " "); }
```

```
        Phrase (char *s) { strcpy (str, s); }
        void display ( ) { cout<< str<<endl;}
        Phrase operator + = (Phrase a)            // перегрузка оператора + =
        {       if ( strlen(str) + strlen(a. str) < size)      strcat (str, a. str);
                else cout<< “строка слишком длинная”;
                return (*this);
        }
};
void main ( )
{ clrscr( );
Phrase b;
Phrase c ( “ тест”);
b+=c;
b. display( );
Phrase d ( “еще “);
d + = c+ =d;
d. display( );
}
```

Виртуальные функции

Виртуальная функция – это функция, которая определяется в базовом классе, а в любом классе, порожденном из базового, может переопределяться. Виртуальная функция вызывается только через указатель или ссылку на базовый класс. Определение того, какой экземпляр виртуальной функции вызывается, зависит от класса объекта, адресуемого указателем или ссылкой, и осуществляется во время выполнения.

Метод определяется виртуальным заданием ключевого слова virtual при объявлении функции в базовом классе.

Например,

virtual void Set (int x, int y);

Правила объявления виртуальных функций:

Если в базовом классе метод определен как виртуальный, то метод, определенный в производном классе с тем же именем и набором параметров, автоматически становится виртуальным, а отличающимися параметрами - обычным.

Виртуальные методы наследуются, т.е. переопределять их в производном классе требуется только при необходимости задания различающихся действий.

Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции разрешения видимости.

Виртуальный метод не может быть объявлен как static, но может быть дружественным.

Если в классе вводится описание виртуального метода, он должен быть определен хотя бы как чисто виртуальный.

Виртуальная функция, объявленная в базовом классе иерархии порождения, часто никогда не используется для объектов этого класса, т.е. не имеет определения. Чтобы подчеркнуть это, используется следующая запись:

virtual int init(void) = 0;

Такие функции называются *чисто виртуальными функциями*.

Класс с одной или с большим количеством чисто виртуальных функций называется *абстрактным классом*. Абстрактный класс может быть использован только как базовый класс для последующих порождений новых классов. Следовательно, нельзя создавать объекты абстрактного класса и нельзя использовать его в качестве типа значения, возвращаемого функцией, и типа параметров функции.

Пример

```
class Shapes {
protected:      int x, y;
public:         virtual void draw ( ) = 0;
                virtual void rotate ( int ) = 0;
};
class Circle : public Shapes {
private: radius;
public:         circle ( int r, int xx, int yy ) { r=radius; x=xx; y=yy;}
                void draw ( ) { …}
                void rotate ( int u ) { …}
};
```

Теперь невозможно создать объекта класса Shapes, поскольку он содержит чисто виртуальные функции. Объявлять объекты класса Circle возможно. Так как виртуальные функции draw ( ) и rotate ( ) в нем переопределены.

Возникновение систем программирования

Первоначально компиляторы разрабатывались и поставлялись вне связи с другими техническими средствами. С ними поставлялись только библиотеки стандартных функций языка программирования. Поэтому задачами разработчиков компиляторов являлись:

подготовка текста исходной программы на входном языке компилятора;

подача данных на вход компилятора;

получение результатов работы компилятора в виде набора объектных файлов;

получение от компоновщика исполняемого файла и подготовка его с помощью загрузчика;

проверка с помощью отладчика правильности выполнения программы и отправка ее на выполнение.

Все эти действия выполнялись последовательностью команд с набором ключей в командной строке операционной системы.

В последствии пользователям стали предоставлять все программные модули в комплекте с компилятором в сопровождении технических средств. Форматы объектных файлов и файлов библиотек были унифицированы, что позволило сочетать библиотеки и объектные файлы различных компиляторов и разработчиков.

Для написания командных файлов компиляции был предложен специальный язык – Makefile, что стало первым шагом по созданию систем программирования. Этот язык позволил в достаточно удобной форме описывать весь процесс создания программы, предоставляя входные модули, библиотеки подпрограмм, порождаемые объектные файлы, ключи, правила обработки для каждого типа файлов. Язык Makefile стал стандартным средством, единым для всех компилятором и требующим от пользователя высокой степени профессиональной подготовки.

Следующим шагом в развитие систем программирования стала интегрированная среда разработки, объединившая в себе возможности текстового редактора и командного языка компиляции. Команды, библиотеки и ключи задавались в виде интерфейсных форм. Главное преимущество - все действия пользователя выполнялись непосредственно в окне редактирования исходного текста программы. Это стало возможным благодаря бурному развитию персональных компьютеров. Первой удачной такой средой считается интегрированная среда программирования Turbo Pascal.

Дальнейшее развитие средств разработки тесно связано с распространением развитых средств графического интерфейса пользователя (GUI - Graphical User Interface), ставшего неотъемлемой частью многих современных операционных систем. Для описания гра-

фических элементов программ потребовались соответствующие языки. На основе, которых сложилось понятие ресурсов пользовательского интерфейса, которым называют множество данных, обеспечивающих внешний вид интерфейса пользователя результирующей программы не связанных напрямую с логикой ее выполнения.

Современная система программирования – комплекс программных средств, предназначенных для кодирования, тестирования, отладки программного обеспечения.

В качестве основных тенденций современного этапа развития систем программирования следует указать внедрение в них средств разработки на основе языков четвертого поколения 4GL (Four Generation Languages) и поддержки систем быстрой разработки программного обеспечения – RAD (Rapid Application Development).

Критерии качества программы

Основные критерии оценки качества программы для ЭВМ. Известно, что один и тот же алгоритм может быть реализован на ЭВМ различными способами, т.е. может быть составлено несколько различных программ, решающих одну и ту же задачу.

Таким образом, нужно иметь некоторые критерии оценки программы, с помощью которых можно судить насколько одна программа лучше другой. Анализ и оценка программы носят преимущественно качественный характер.

1. Программа работает и решает поставленную задачу. Понятно, что эта характеристика программы является самой важной.

В связи с этим каждая программа должна быть устроена так, чтобы можно было проверить правильность полученных результатов. Такая проверка проводится в процессе отладки программы, на определенных наборах входных данных, для которых заранее известен ответ. Но отладка может доказать лишь наличие ошибок в программе, но не может доказать правильности программы для всех возможных вычислений, реализуемых с ее помощью. В связи с этим необходима разработка методов аналитической верификации программы. Для аналитического доказательства правильности программы требуется, чтобы программа легко анализировалась. Это означает, что программа должна быть устроена так, чтобы можно было понять, каким образом с ее помощью получается данный ответ.

2. Минимальное время, затрачиваемое на тестирование и отладку программы. Тестирование и отладка программы – необходимый этап в процессе решения задачи на ЭВМ. Он занимает от трети до половины всего времени разработки программы, поэтому очень важно уменьшить время, затрачиваемое на тестирование и отладку. Тестирование и отладка программы облегчается, если программа просто анализируется и снабжена необходимыми комментариями, облегчающими ее понимание. Хорошие комментарии могут ускорить процесс отладки. Понимание и отладка программы облегчается, если она имеет простую и ясную структуру, в частности, если ограничено использование операторов передачи управления (GOTO). Перегруженность программы этими операторами приводит к хаотической структуре и затрудняет отладку. Еще один важный принцип – использование мнемонических обозначений для переменных. Языки программирования представляют здесь вполне достаточные возможности. Для лучшего понимания программы необходимо использовать мнемонику, отражающую физический (математический, экономический и т.д.) смысл переменной (например, SPEED - скорость).

3. Уменьшение затрат на сопровождение. Разработанная и отлаженная программа предназначена для многократного использования, и ее эксплуатацией, как правило, занимаются не разработчики, а другие программисты, входящие в так называемую группу сопровождения. Программистам, сопровождающим программу, часто приходится продолжать отладку программы и производить ее модернизацию, в связи с изменением технического задания, введением новых средств программного обеспечения или выявлением новых ошибок и недоработок в программе. Для уменьшения затрат на сопровождение необходимо, чтобы каждый разработчик учитывал сложность сопровождения. Следует разрабатывать, отлаживать

и оформлять программу с учетом того, что ее будут использовать и сопровождать другие программисты.

4. Гибкость программы. Разработанная программа обычно находится в эксплуатации длительное время. За это время могут измениться требования к решаемой задаче, техническое задание, требования к программе. Появляется необходимость внести определенные изменения в программу, что в некоторых случаях бывает трудно сделать, т.к. разработчиком не предусмотрена такая возможность. "Хорошая" программа должна допускать модификацию.

5. Уменьшение затрат на разработку. Программирование является коллективным трудом. Состав группы программистов, работающих над решением данной задачи, может по каким-либо причинам измениться. Поэтому проектирование и разработка программы должны вестись таким образом, чтобы было возможно при необходимости передать ее завершение другому программисту. Несоблюдение этого требования часто приводит к срыву сроков сдачи программ в эксплуатацию. 6. Простота и эффективность. Программа должна быть просто организована. Это может проявляться и в структуре программы, и в использование простых и наиболее естественных средств языка программирования, и в предпочтении простых структур данных и т.п. Эффективность программы считается одной из главных ее характеристик. Поэтому часто в ущерб другим качествам программы разработчики прибегают к сложным ухищрениям, чтобы уменьшить объем используемой памяти или сократить время выполнения программы. Во многих случаях затрачиваемые на это усилия не оправдывают себя. Разумный подход к повышению эффективности программы состоит в том, чтобы выявить наиболее "узкие" места и постараться их улучшить.

Главное требование, которому должна удовлетворять программа, — работать в полном соответствии со спецификацией и адекватно реагировать на любые действия пользователя. Кроме этого, программа должна быть выпущена точно к заявленному сроку и допускать оперативное внесение необходимых изменений и дополнений. Объем занимаемой памяти и эффективность алгоритмов при этом, к сожалению, отходят на второй план. Иными словами, современные критерии качества программы — это, прежде всего, надежность, а также возможность точно планировать производство программы и ее сопровождение. Для достижения этих целей программа должна иметь простую структуру, быть хорошо читаемой и легко модифицируемой.

III. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ВЫПОЛНЕНИЮ ДОМАШНИХ ЗАДАНИЙ И КОНТРОЛЬНЫХ РАБОТ

При выполнении домашних заданий необходимо повторить материал лекций и других видов занятий. Изучить соответствующие разделы учебно-методических пособий.

При подготовке к контрольной работе повторить изученный материал, выполнить задания для самостоятельной проверки и ответить на контрольные вопросы из соответствующих разделов методических пособий.

IV. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ПРОВЕДЕНИЮ И ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторные работы проводятся по подгруппам в компьютерном классе. Каждый студент получает индивидуальное задание в соответствии с вариантом.
Выполняя задание, студент пользуется материалом, изученным в лекционном курсе, а также методическими пособиями:

Галаган Т.А., Соловцова Л.А. Язык программирование С++ в примерах и задачах. Учебно-методическое пособие. Благовещенск: Изд-во АмГУ, 2002. 40 с.

Галаган Т.А., Соловцова Л.А. Язык программирование С++ в примерах и задачах. Практикум. Благовещенск: Изд-во АмГУ, 2005. 104 с.

Перед созданием любой программы требуется точно продумать алгоритм. Записать его блок-схемой или словесно. Надо четко определить, что в нее требуется ввести и что получить в результате, в какой последовательности выполнять действия. В случае необходимости выделить циклические структуры и подпрограммы. В циклах четко определить параметры, задать их начальные значения, определить условия повторения и завершения цикла. В функциях определить количество передаваемых и возвращаемых значений.

При кодировании программы нужно определить тип используемых данных в зависимости от возможного диапазона принимаемых значений. При вводе величины не забывать осведомить об этом пользователя, а иногда сообщить и о типе, диапазоне или порядке ввода значений. Такое сообщение должно быть информативно и коротко. Вывод данных лучше сопровождать текстом и форматированием. Формат вывода можно уточнить при помощи модификаторов.

В именах переменных необходимо отражать их назначение, что повышает читаемость и понимание программы.

При записи сложных выражений нужно обращать внимание на приоритет операций. Текст программы лучше сопровождать краткими и информативными комментариями, что облегчает как понимание программы, так и ее отладку.

Объявление локальных переменных предпочтительнее по сравнению с глобальными.

Для отладки программы нужно запустить ее на выполнение несколько раз, задавая различные значения вводимых величин. Перед запуском необходимо иметь заранее подготовленные тестовые примеры, содержащие исходные данные и ожидаемые результаты. Их количество зависит от алгоритма. проверьте реакцию программы на заведомо неверные исходные данные.

Для быстрого поиска ошибки в алгоритме рекомендуется выводить промежуточные данные.

При сдаче лабораторной работы студент должен продемонстрировать преподавателю созданную языке С++, правильно работающую, отлаженную программу на основе эффективного алгоритма.

Преподаватель, принимая лабораторную работу, тестирует программу студента и задает ему вопросы по конструкциям, используемым в программе и теоретическим основам программирования.

V. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа №1
Тема **Линейные программы** (2 часа)

Задание. Написать программу для расчета двух формул. Подготовить тестовые задания и проверить результат с помощью калькулятора. Результаты вычислений по первой и второй формуле должны совпадать.

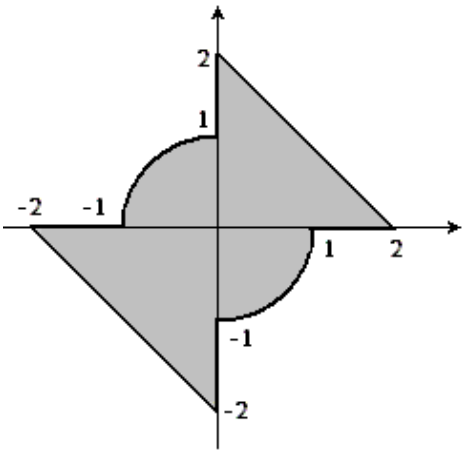
Вариант 1 $z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$ $z_2 = \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$	Вариант 8 $z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$ $z_2 = \sin(x+y)\sin(y-x)$
Вариант 2 $z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$ $z_2 = 2\sqrt{2}\cos\alpha\sin\left(\frac{\pi}{4} + 2\alpha\right)$	Вариант 9 $z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$ $z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cos(\alpha+\beta)$
Вариант 3 $z_1 = \frac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$ $z_2 = \frac{1 - tg\alpha}{1 + tg\alpha}$	Вариант 10 $z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$ $z_2 = ctg\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$
Вариант 4 $z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha}$ $z_2 = tg\,3\alpha$	Вариант 11 $z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$ $z_2 = 2\sin\alpha$
Вариант 5 $z_1 = \frac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$ $z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$	Вариант 12 $z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha}$ $z_2 = ctg\left(\frac{3}{2}\pi - \alpha\right)$
Вариант 6 $z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$ $z_2 = 4\cos\frac{\alpha}{2}\cos\frac{5}{2}\alpha\cos 4\alpha$	Вариант 13 $z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$ $z_2 = \cos^2\alpha + \cos^4\alpha$
Вариант $z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$ $z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$	Вариант 14 $z_1 = \frac{\cos\alpha + \sin\alpha}{\cos\alpha - \sin\alpha}$ $z_2 = \frac{\cos 2\alpha}{1 - \sin 2\alpha}$

Лабораторная работа № 2

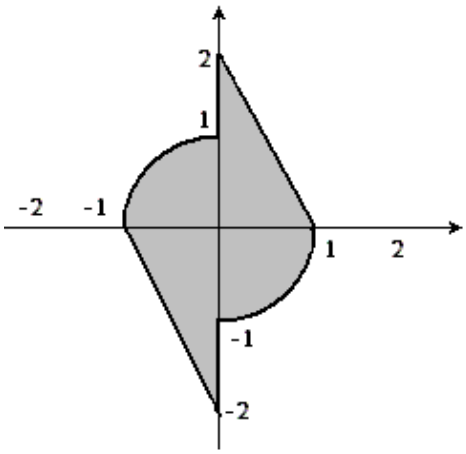
Тема: **Программирование разветвляющей структуры** (2 часа)

Задание. Дана «мишень» в виде закрашенной области, изображенной на рисунке. Создать алгоритм для определения попадания точки с координатами (x, y) в мишень. Значение координат точки вводить с клавиатуры. Написать программу с использованием условного оператора if.

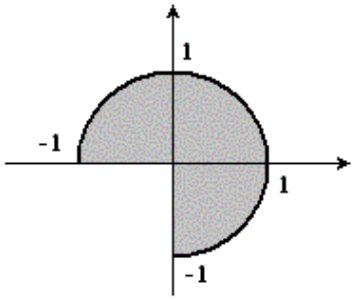
1.



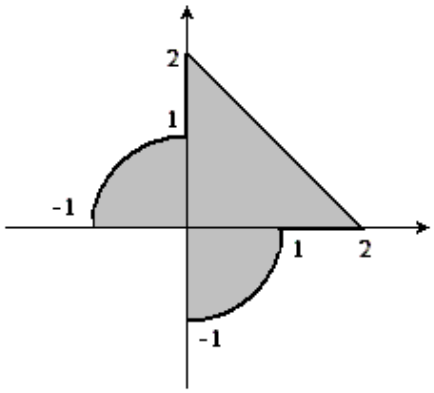
2.



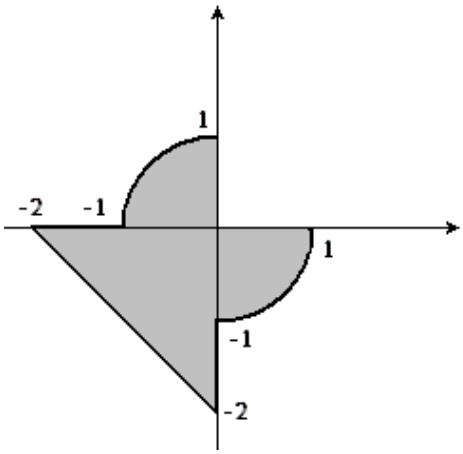
3.



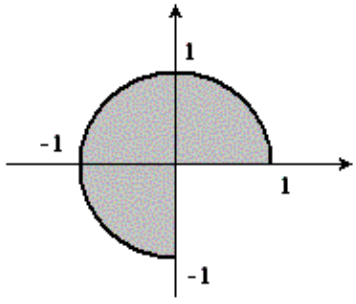
4.



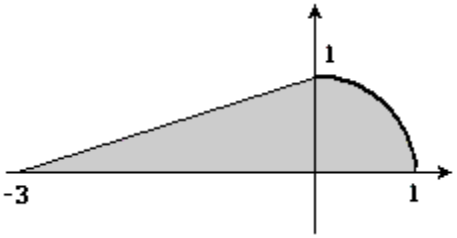
5.



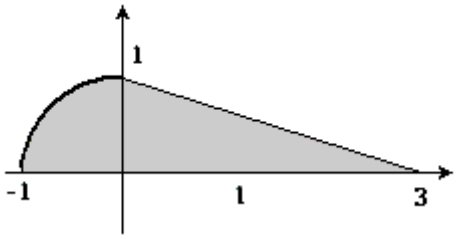
6.



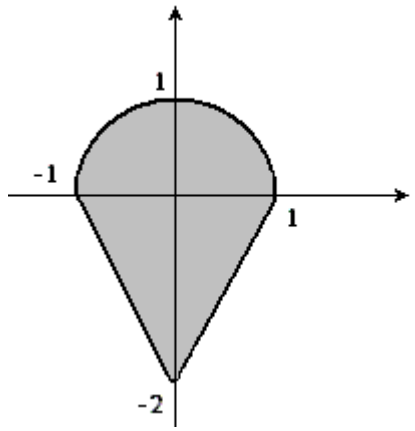
7.



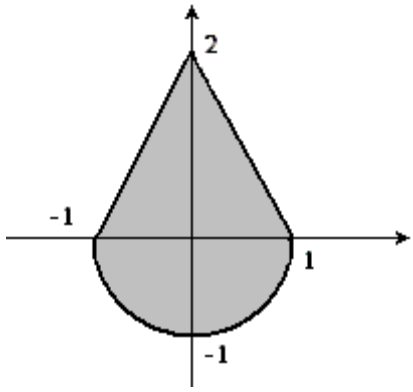
8.



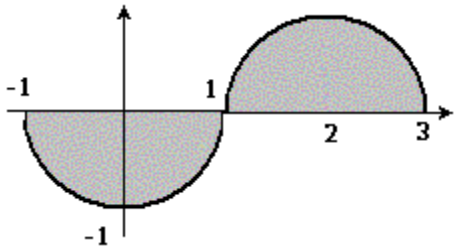
9.



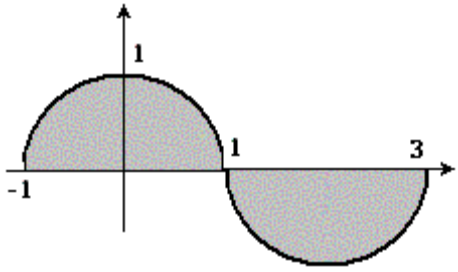
10.



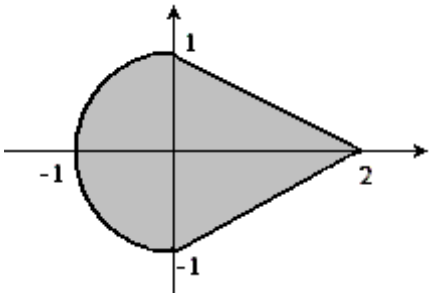
11.



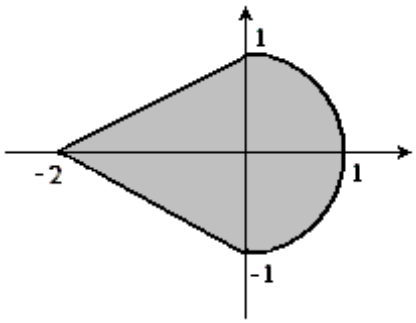
12.



13.



14.



Лабораторная работа №3

Тема **Разветвляющиеся программы. Оператор switch.** (2 часа)

Задание. Написать программу с использованием оператора множественного выбора switch.

Варианты

1. Создать программу, вычисляющую площадь фигуры на плоскости.

$$S = \begin{cases} \pi R^2, & при \quad k = 1 \\ a\sqrt{3}/4, & при \quad k = 3 \\ a^2, & при \quad k = 4 \\ 3\sqrt{3}a/2, & при \quad k = 6 \end{cases}$$

Значение k вводить с клавиатуры. Предусмотреть вывод сообщения при вводе отсутствующего значения k.

2. С клавиатуры ввести число y, обозначающее день недели. В зависимости от введенного значения на экран вывести название дня недели. Предусмотреть вывод об ошибке вво-

да, и также напоминание о выходных днях.

3. С клавиатуры ввести число x, обозначающее номер месяца. В зависимости от введенного значения на экран вывести название сезона (лето, весна, осень, зима). Предусмотреть вывод об ошибке ввода, если введенного число не соответствуют возможному значению номера месяца.

4. Создать программу, содержащую справочную информацию о значениях тригонометрических функций. По введенной с клавиатуры величине угла x, выраженной в градусах, программа должна выдавать соответствующие значения cos x и sin x.

градусы	0	30	45	60	90
sin x	0	1/2	$\sqrt{2}/2$	$\sqrt{3}/2$	1
cos x	1	$\sqrt{3}/2$	$\sqrt{2}/2$	1/2	0

Предусмотреть вывод об ошибке ввода, если введенного число не соответствуют возможному значению.

5. С клавиатуры ввести число m, обозначающее номер месяца. В зависимости от введенного значения программа должна выводить на экран количество дней в месяце (без учета високосных года). Предусмотреть вывод об ошибке ввода, если введенного число не соответствуют возможному значению номера месяца.

6. С клавиатуры ввести число m, обозначающее номер месяца. В зависимости от значения введенного числа программа должна выводить на знак по гороскопу: (1 - козерог, 2 - водолей, 3 – рыбы, 4 – овен, 5 – телец, 6 – близнецы, 7 – рак, 8 – лев, 9 – дева, 10 – весы, 11- скорпион, 12 – стрелец). Предусмотреть вывод об ошибке ввода, если введенного число не соответствуют возможному значению номера месяца.

7. С клавиатуры ввести число y, обозначающее количество углов в правильном многоугольнике. В зависимости от введенного значения на экран вывести величину угла, выраженного в градусах. Предусмотреть вывод об ошибке ввода.

8. Вычислить значение переменной A, в зависимости от величины параметра B.

$$A=\begin{cases} 5x & B=0 \\ 25x/15 & B=1 \\ 100e^{x} & B=2 \\ x^{5} & B=3 \end{cases}$$

Значение x вводить с клавиатуры. Предусмотреть вывод сообщения при вводе несуществующего значения B.

9. Вычислить значение переменной A, в зависимости от величины параметра B.

$$A=\begin{cases} 34x & B=10 \\ 25.x-x^{2} & B=20 \\ 10x-45/37 & B=30 \\ 98(x-1) & B=40 \end{cases}$$

Значение x вводить с клавиатуры. Предусмотреть вывод сообщения при вводе несуществующего значения B.

10. С клавиатуры ввести целое число (3, 4, 5, 6, 7, 8) в зависимости от значения которого на экран вывести название многоугольника, содержащего указанное число сторон, например 3 – треугольник. Предусмотреть вывод сообщения об ошибке, если введенное число не соответствует заданному диапазону.

11. С клавиатуры ввести целое число, означающее номер класса в школе, в зависимости от его значения на экран вывести следующие сообщения 1- 4 – младший класс, 5 - 8 – средняя школа, 9 -11 – старший класс. Предусмотреть сообщение для случая неверно введенного значения.

12. С клавиатуры ввести целое число, означающее время суток в часах. В зависимости от введенного значения на экран вывести (утро, день, полдень, вечер, ночь). Предусмотреть

обработку ошибочно введенного значения.

13. С клавиатуры ввести целое число, означающее величину угла, выраженную в градусах (30, 45, 90, 135, 180, 225, 270, 315, 3 60). По введенному значению программа должна выводить величину угла, выраженную в радианах ($\pi/6, \pi/4$ и т.д.). Предусмотреть обработку ошибочно введенного значения.

14. С клавиатуры ввести средний бал студента, по результатам сессии. В зависимости от введенного значения вывести на экран следующие сообщения:

От 4.5 до 5	-	Вам начислена повышенная стипендия
От 4 до 4.5	-	Вам начислена стипендия
От 3 до 4	-	Стипендии нет
Меньше 3	-	Вы отчислены

Предусмотреть вывод об ошибке ввода, если введенного число меньше 2 или больше 5.

Лабораторная работа № 4

Тема: Программирование разветвляющей и циклической структуры **(2часа)**

Задание. Вычислить и вывести на экран в виде таблицы все значения функции y на заданном интервале с шагом Δt. В каждой строке таблицы должно содержаться значение аргумента и значение функции. Таблица должна содержать заголовок и шапку. Для выравнивания границ таблицы при выводе на экран использовать библиотеку iomanip.h. Значение констант задать в программе. Для задания цикла в программе использовать оператор while. Составить блок-схему алгоритма.

Варианты

1.

$$y=\begin{cases} at^2\ln t, & 1\le t\le 2 \\ 1, & t<1 \\ e^{at}\cos bt, & t>2 \end{cases}$$ a = -0.5, b = 2, $t\in[0;3]$, $\Delta t = 0.15$

2.

$$y=\begin{cases} \pi t^2 - 7/t^2, & t<1.3 \\ at^3+7\sqrt{t}, & t=1 \\ \lg(t+7\sqrt{t}), & t>1.3 \end{cases}$$ a=1.5, $t\in[0.8;2]$, $\Delta t = 0.1$

3.

$$y=\begin{cases} at^2+bt+c, & t<1.4 \\ a/t+\sqrt{t^2+1}, & t=1.4 \\ (a+bt)/\sqrt{t^2+1}, & t>1.4 \end{cases}$$ a=2.8, b=-0.2, c=4 $t\in[1;2]$, $\Delta t = 0.05$

4.

$$y=\begin{cases} \pi t^2 - 7t^2, & t<1.4 \\ at^3+7\sqrt{t}, & t=1.4 \\ \ln(t+7\sqrt{|t+a|}, & t>1.4 \end{cases}$$ a=1.65, $t\in[0.7;2]$, $\Delta t = 0.1$

5.

$$y=\begin{cases}1.5\cos^2 t, & t<2\\ 1.8at, & t=2\\ (2-t)+3tgt, & t>2\end{cases}$$ a=2.3, $t\in[0.2;2.8]$, $\Delta t=0.2$

6.

$$y=\begin{cases}t\sqrt{t-a}, & t>a\\ t\sin at, & t=a\\ e^{-at}\cos at & t<a\end{cases}$$ a=2.5, $t\in[1;5]$, $\Delta t=0.5$

7.

$$y=\begin{cases}bt-\lg bt, & bt<1\\ 1, & bt=1\\ bt+\lg bt, & bt>1\end{cases}$$ b=1.5 $t\in[0.1;1]$, $\Delta t=0.1$

8.

$$y=\begin{cases}\sin at^2, & t>3.5\\ \lg t+\sqrt{t^2+1}, & t=3.5\\ \cos^2 t, & t<3.5\end{cases}$$ a=2.8 $t\in[2;5]$, $\Delta t=0.25$

9.

$$y=\begin{cases}\lg(t+c), & t>1\\ a^3, & t=1\\ \sin^2\sqrt{|at|}, & t<1\end{cases}$$ a=20.8, c=4

$t\in[0.5;2]$, $\Delta t=0.1$

10.

$$y=\begin{cases}(\ln^3 t+t^2)/\sqrt{t+c}, & t<0.5\\ 1/t+\sqrt{t+c}, & t=0.5\\ \cos t+t\sin^2 t, & t>0.5\end{cases}$$ c=2.2

$t\in[0.2;2]$, $\Delta t=0.2$

11.

$$y=\begin{cases}(a+b)/(e^t+\cos t), & t<2.8\\ (a+t)/(t+1), & t=2.8\\ e^t+\sin t, & t>2.8\end{cases}$$ a=2.6, b=-0.29 $t\in[0;7]$, $\Delta t=0.5$

12.

$$y=\begin{cases}a\lg t+\sqrt{|t|}, & t>1\\ a^2, & t=1\\ 2a\cos t+3t^2, & t<1\end{cases}$$ a=0.9, $t\in[0.8;2]$, $\Delta t=0.1$

13.

$$y=\begin{cases} a/t+bt^2+c, & t<4 \\ t, & 4\le t\le 6 \\ at+bt^2, & t>6 \end{cases} \qquad \text{a=2.1, b=1.8, c=-20.4}$$

$t\in[0;12], \qquad \Delta t=1$

14.

$$y=\begin{cases} \sqrt{at^2+b\sin t+1}, & t<0.1 \\ at+b, & t=0.1 \\ \sqrt{at^2+b\cos t+1}, & t>0.1 \end{cases} \qquad \text{a=2.5, b=0.4} \qquad t\in[-1;1], \qquad \Delta t=0.2$$

Лабораторная работа № 5

Тема: **Вычисление суммы ряда с заданной точностью** (2 часа)

Задание. Вычислить значение суммы ряда с заданной точностью ε . На экран вывести значение суммы ряда, количество слагаемых в сумме, контрольное значение. Для организации цикла в программе использовать оператор с заданным числом повторений do … while. Эффективный алгоритм может быть получен выражением каждого следующего слагаемого в сумме через предыдущее.

Варианты

№	*Функция*	*Ограничение*	*Точность*	*Значение*
1	$\sum_{n=0}^{\infty}\frac{(-1)^n x^n}{n!}$	$\lvert x\rvert<\infty$	10^{-5}	e^{-x}
2	$\sum_{n=0}^{\infty}\frac{(-1)^n x^{n+1}}{n+1}$	$\lvert x\rvert<1$	10^{-4}	*ln(x+1)*
3	$\frac{\pi}{2}+\sum_{m=0}^{\infty}\frac{(-1)^{m+1} x^{2m+1}}{2m+1}$	$\lvert x\rvert<1$	10^{-5}	*arctg x*
4	$\frac{\pi}{2}+\sum_{n=0}^{\infty}\frac{(-1)^{n+1}}{(2n+1)x^{2n+1}}$	$\lvert x\rvert>1$	$0.5*10^{-4}$	*arctg x*
5	$-\frac{\pi}{2}+\sum_{m=0}^{\infty}\frac{(-1)^{m+1}}{(2m+1)x^{2m+1}}$	$x<-1$	10^{-4}	*arctg x*
6	$\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!}$	$\lvert x\rvert<\infty$	10^{-4}	*cos x*
7	$\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n+1)!}$	$\lvert x\rvert<\infty$	$0.5*10^{-5}$	$\frac{\sin x}{x}$
8	$\sum_{m=0}^{\infty}\frac{(-1)^m x^{2m}}{m!}$	$\lvert x\rvert<\infty$	10^{-6}	e^{-x^2}
9	$\frac{\pi}{3}+\sum_{n=1}^{\infty}(-1)^n\frac{(\pi/3)^{2n+1}}{(2n+1)!}$	$\lvert x\rvert<\infty$	$0.5*10^{-4}$	$\sin(\pi/3)$
10	$\sum_{n=1}^{\infty}(-1)^n\frac{(\pi/6)^{2n}}{(2n)!}$	$\lvert x\rvert<\infty$	$0.5*10^{-4}$	$\cos(\pi/6)$

11	$\sum_{m=0}^{\infty}\frac{(x-1)^{2m+1}}{(2m+1)(x+1)^{2m+1}}$	$x>0$	10^{-4}	$\frac{\ln x}{2}$
12	$\sum_{m=0}^{\infty}\frac{(-1)^m(x-1)^{m+1}}{(m+1)}$	$0<x\le 2$	$0.5*10^{-5}$	$ln\ x$
13	$\sum_{m=0}^{\infty}\frac{(x-1)^{m+1}}{(m+1)(x+1)^{m+1}}$	$x>\frac{1}{2}$	$0.5*10^{-4}$	$ln\ x$
14	$2\sum_{m=0}^{\infty}\frac{x^{2m+1}}{2m+1}$	$\lvert x\rvert<1$	10^{-4}	$\ln\frac{1+x}{1-x}$

Лабораторная работа №6

Тема: **Одномерные массивы** (2 часа)

Задание. Составить блок-схему алгоритма решения задачи. Написать программу на языке С++. Элементы массива ввести с клавиатуры.

Варианты

1. В одномерном массиве A, содержащем 15 элементов, определить сумму положительных элементов, расположенных после минимального элемента данного массива.
2. В одномерном массиве B, содержащем 12 элементов, определить среднее арифметическое отрицательных элементов, расположенных после минимального элемента данного массива.
3. В одномерном массиве C, содержащем 14 элементов, определить произведение положительных элементов, расположенных до максимального элемента этого массива.
4. В одномерном массиве A, содержащем 14 элементов, определить сумму положительных элементов, расположенных между минимальным и максимальным элементами данного массива.
5. В одномерном массиве B, содержащем 16 элементов, определить среднее геометрическое отрицательных элементов, расположенных между минимальным и максимальным элементами данного массива.
6. В одномерном массиве C, содержащем 13 элементов, определить сумму минимального и максимального элементов данного массива.
7. В одномерном массиве H, содержащем 14 элементов, определить сумму положительных элементов, расположенных после минимального элемента данного массива.
8. В одномерном массиве X, содержащем 17 элементов, определить минимальный элемент среди положительных элементов данного массива.
9. В одномерном массиве X, содержащем 14 элементов, определить максимальный элемент среди отрицательных элементов данного массива.
10. В одномерном массиве A, содержащем 15 элементов, определить произведение положительных элементов, расположенных после минимального элемента данного массива.
11. В одномерном массиве B, содержащем 16 элементов, определить произведение отрицательных элементов, расположенных после максимального элемента данного массива.
12. В одномерном массиве D, содержащем 14 элементов, определить среднее арифметическое элементов, расположенных после минимального элемента данного массива.
13. В одномерном массиве D, содержащем 14 элементов, определить разность между минимальным и максимальным элементами данного массива.
14. В одномерном массиве Y, содержащем 14 элементов, заменить минимальный элемент нулем, а максимальный увеличить в десять раз.

Лабораторная работа №7

Тема: **Сортировка элементов одномерного массива**(2 часа)

Задание. Элементы массива задавать с помощью счетчика случайных чисел. Для сортировки элементов массива использовать один из известных способов: пузырьковая сортировка или сортировка методом прямого выбора.

Варианты

1. Элементы массива A (20) отсортировать следующим образом: сначала расположить все отрицательные элементы в порядке возрастания, затем нулевые, а следом положительные в порядке убывания.
2. Элементы вектора B (28) отсортировать следующим образом: сначала расположить все отрицательные в порядке убывания, затем положительные в порядке возрастания, а следом нулевые.
3. Первые десять элементов вектора M (30) отсортировать в порядке возрастания, а остальные в порядке убывания.
4. Отсортировать по убыванию элементы массива P (32), расположенные до максимального элемента данного массива.
5. Отсортировать по возрастанию элементы массива X (29), расположенные за минимальным элементом данного массива.
6. Отсортировать по возрастанию элементы массива O (25), расположенные между минимальным и максимальным элементами данного массива.
7. Если положительных элементов вектора K (24) больше чем отрицательных, то отсортированные положительные элементы расположить в начале вектора. Иначе в начале вектора расположить отсортированные отрицательные элементы.
8. Вывести на экран сначала нулевые элементы вектора C (39), а за ними отсортированные по убыванию ненулевые элементы данного вектора.
9. Первые пять элементов массива T (23) оставить в неизменном порядке, следующие 10 отсортировать в порядке возрастания, за ними три элемента оставить без изменения, а оставшиеся отсортировать в порядке возрастания.
10. Если номер максимального элемента вектора K (25) больше 15, то отсортировать по убыванию элементы вектора, расположенные до максимального элемента, иначе отсортировать элементы данного вектора, расположенные за максимальным элементом.
11. Если номер минимального элемента вектора H(28) меньше 18 , отсортировать по возрастанию элементы, расположенные после него, иначе отсортировать элементы с 8 по 18 номер.
12. Расположить элементы вектора P(20) следующим образом: сначала минимальный элемент, затем максимальный, а далее оставшиеся элементы вектора, отсортированные по возрастанию.
13. Вывести на экран сначала отсортированные по возрастанию элементы вектора Y(25), стоящие на четных местах, а затем отсортированные элементы, стоящие на нечетных местах.
14. Дан вектор U(20). Вывести на экран элементы данного вектора следующим образом: 1-ый, 5-ый, 15-ый и 20-ый элементы оставить без изменения, а элементы, расположенные между ними отсортировать по убыванию.

Лабораторная работа № 8

Тема Символьные массивы (2 часа)

Задание. Символьная строка является массивом типа int. Для работы с символьными строками воспользуйтесь функциями библиотеки string.h

Варианты

1. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сравнить данные строки по длине. Если они не равны, присоединить к меньшей строке - большую. Определить количество слов в полученной строке. Результат вывести на экран.

2. Ввести две строки с клавиатуры. Посчитать в каждой из них количество гласных букв. В строке, содержащей большее число гласных, удалить все согласные буквы.

3. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Соединить их содержимое. Определить длину полученной строки. Вывести на экран первую половину полученной строки.

4. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Если их содержимое одинаково, оставить строки без изменения, иначе соединить содержимое строк. При этом первыми должны быть символы той строки, в которой первый символ по алфавиту раньше.

5. Одну строку инициализировать в программе, другую – ввести с клавиатуры. В каждой из строк исключить поместить первый символ в конец строки, после чего соединить строки, расположив вначале большую по алфавиту.

6. В строке символов найти самое длинное и самое короткое слово. Сформировать новую строку, расположив в ее начале самое короткое слово, затем самое длинное, а потом все остальные слова в алфавитном порядке.

7. Определить в строке количество предложений. Если предложений более одного, копировать второе предложение в отдельную строку и вывести ее содержимое на экран.

8. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сформировать из них новую строку, чередуя в ней слова из заданных строк.

9. Две строки инициализировать в программе, третью – ввести с клавиатуры. Сформировать из них новую строку по следующему алгоритму: сначала соединить строки в порядке возрастания их длины, затем исключить из нее первое и среднее слова.

10. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сравнить строки в лексикографическом порядке, и если они не равны соединить их. Отредактировать вновь полученную строку, исключив множественные пробелы.

11. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Если строки не тождественны, соединить их, добавив к большей по длине меньшую. Отредактировать вновь полученную строку, удалив из нее самое короткое слово.

12. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Присоединить к меньшей по длине строке большую. Отредактировать вновь полученную строку исключив из нее слова , содержащие меньше двух букв.

13. Одну строку инициализировать в программе, другую – ввести с клавиатуры. В строке меньшей длины добавить символы ‘а’, чтобы строки сравнялись по длине. В строке большей длины исключить все символы ‘о’.

14. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сравнить строки на тождественность и если это не так, соединить их содержимое. Во вновь полученной строке определить слово наименьшей длины.

Лабораторная работа № 9

Тема: ***Двумерные массивы*** (4 часа)

Задание. Составить блок-схему алгоритма решения задачи. Написать программу на языке С++. Значения элементов массива вводить с клавиатуры. Осуществить вывод элементов первоначальной и измененной матрицы в общепринятом виде (в виде таблицы).

Варианты

1. В матрице размерности 8 на 6 определить номер первого из столбцов, содержащих хотя бы один нулевой элемент. В каждой второй строке матрицы заменить максимальный элемент нулем. Найти сумму положительных элементов матрицы.

2. В матрице размерности 5 на 10 определить количество столбцов, содержащих хотя бы один нулевой элемент, вывести на экран их номера. В каждой строке матрицы, поменять местами максимальный и минимальный элементы. Заменить положительные элементы последний строки нулями.

3. В матрице размерности 8 на 8 определить сумму элементов в столбцах, не содержащих отрицательные элементы. Найти минимум среди сумм модулей элементов диагоналей, параллельных главной диагонали матрицы.

4. В матрице размером 8 на 8 найти такие k, что k-ая строка матрицы совпадает с k-ым столбцом. Увеличить минимальный элемент матрицы в десять раз.

Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

5. В матрице размером 6 на 9 определить сумму модулей его отрицательных нечетных элементов каждой строки. Найти сумму элементов в тех столбцах, которые содержат хотя бы один нулевой элемент. Заменить положительные элементы второго и третьего единицами.

6. В матрице размером 10 на 10 найти сумму элементов в тех столбцах, которые содержат хотя бы один нулевой элемент. Заменить отрицательные элементы матрицы их модулями. В измененной матрице найти произведение элементов, расположенных ниже главной диагонали.

7. В матрице размером 8 на 10 определить номера минимумов в каждой строке. Посчитать произведение отрицательных элементов, расположенных выше главной диагонали. Элементы побочной диагонали заменить нулями.

8. В матрице размерности 12 на 8 найти номер первого из столбцов, не содержащих ни одного из отрицательных элементов. В каждой второй строке матрицы заменить минимальный элемент нулем, а максимальный сотней.

9. В матрице размерности 8 на 9 определить произведение элементов в каждом втором столбце. Найти минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы. В последнем столбце матрицы заменить положительные элементы единицами, а отрицательные – нулями.

10. В матрице размером 9 на 9 определить количество строк, содержащих хотя бы один положительный элемент. Определить номер столбца, в котором содержится минимальный элемент матрицы. На главной диагонали матрицы заменить положительные элементы нулями.

11. В матрице размерности 8 на 8 определить сумму минимального и максимального элементов, из расположенных на главной диагонали. В каждой второй строке заменить положительные элементы нулями. Во вновь полученной матрице определить среднее арифметическое каждого столбца.

12. В матрице размерности 10 на 10 поменять местами минимальный и максимальный элементы. Определить произведение положительных элементов пятого и десятого столбцов (отдельно). В каждой строке подсчитать сумму положительных элементов.

13. В матрице размерности 6 на 7 посчитать произведение сумм положительных элементов строк. В каждом втором столбце заменить минимальный элемент нулем.

14. В матрице размерности 8 на 8 посчитать произведение элементов главной диагонали. Найденным значением заменить минимальный элемент матрицы. Посчитать сумму отрицательных элементов каждого столбца.

Лабораторная работа №10

Тема: **Функции** (2 часа)

Задание. Составить пользовательскую функцию. Продемонстрировать ее использование в программе.

Варианты

1. Определить среднее арифметическое сторон двух треугольников, заданных координатами их вершин. Длину стороны треугольника вычислять в функции.

2. Вычислить среднее арифметическое объемов шаров с радиусами r_1, r_2, r_3. Для нахож-

дения объема шара использовать функцию.

3. Заданы три конуса (радиусы основания и высота). Определить конус с наибольшим объемом. Для нахождения объема конуса использовать функцию.

4. Вычислить значение $z = (sign\, x + sign\, y)\, sign\,(x + y)$,

где $sign\, a = \begin{cases} -1, & при \quad a < 0, \\ 0, & при \quad a = 0, \\ 1, & при \quad a > 0. \end{cases}$

5. Составить программу вычисления значений:

$$a = \frac{\sqrt{z^3 + 4z^2 + 7z + 1}}{2 + e^{2z^3 + z - 3}}, \quad b = \frac{t^2 + 13t + 16}{7t^3 + t^2 - 4}.$$

Для определения значений многочленов использовать функцию. Значения величин *z, t* вводить с клавиатуры.

6. Вычислить значение $f = \dfrac{\max(a,b,c) * \min(a,c,d) - \max(b,c,d)}{\min(a,b,c)}$,

где *a, b, c, d* – некоторые значения, введенные с клавиатуры. Для нахождения максимального и минимального значений использовать функции.

7.Создать функцию вывода на экран таблицы умножения для числа по запросу пользователя. Продемонстрировать работу функции в программе.

8. Создать функцию определения дня недели по введенной дате. Считать, что 1 января 1 года 1 века было понедельником.

9. Создать функцию нахождения скалярного произведения двух векторов в трехмерном пространстве..

10. Создать функцию вычисления площади треугольника по трем заданным сторонам. Продемонстрировать работу функции.

11. Создать функцию, определяющую принадлежит ли точка с заданными координатами (x_1 , y_1) уравнению прямой $y = kx + b$. Использовать функцию для определения из набора точек, лежащих на одной прямой.

12. Создать функцию нахождения длины медианы треугольника по заданным длинам сторон.

13. Создать функцию нахождения факториала произвольного числа.

14. Создать функцию вычисления определителя для матрицы размерности 3 на 3.

ЛАБОРАТОРНАЯ РАБОТА № 11

Тема: **Массивы и указатели** (4 часа)

Задание. В каждом задании требуется создать функцию, параметром которой является одномерный массив. Продемонстрируйте вызов функции. Тело функции задайте двумя способами: обращение к элементам через указатели и обычным способом. Продемонстрируйте возможность применения созданной функции к строкам двумерного массива.

Варианты

1. Создайте функцию, меняющую местами минимальный и максимальный элементы одномерного массива. Воспользуйтесь функцией для каждой строки матрицы B (5, 4).

2. Создайте функцию, вычисляющую среднее арифметическое отрицательных элементов одномерного массива. Воспользуйтесь данной функцией для каждой строки матрицы A (4, 8).

3. Создайте функцию, вычисляющую сумму отрицательных элементов одномерного массива в случае, если их более 3, и возвращающую нуль иначе. Воспользуйтесь данной функцией для каждой строки матрицы X (4, 7).

4. Создайте функцию, вычисляющую сумму значений элементов одномерного массива, расположенных между его максимальным и минимальным элементами. Воспользуйтесь функцией для каждой строки матрицы O (3, 8).

5. Создайте функцию, подсчитывающую количество отрицательных элементов одномерного массива, порядковый номер которых меньше номера минимального элемента. Воспользуйтесь функцией для каждой строки с четным номером матрицы A (8, 4).

6. Создайте функцию, заменяющую положительные элементы массива нулями, а отрицательные их абсолютными величинами, увеличенными в 10 раз. Воспользуйтесь функцией для каждой нечетной строки матрицы C (5, 6).

7. Создайте функцию, подсчитывающую количество положительных элементов одномерного массива, порядковый номер которых больше номера максимального элемента. Воспользуйтесь функцией для каждой строки матрицы A (7, 6), с номером кратным 3.

8. Создайте функцию, параметром которой будет одномерный массив. Поменяйте местами первый элемент массива с последним, второй – с предпоследним, и т.д. Воспользуйтесь функцией для каждой строки матрицы H (4, 8)

9. Создайте функцию, входным параметром которой будет одномерный массив. Посчитайте среднее арифметическое элементов, стоящих на нечетных местах. Воспользуйтесь функцией для каждой строки матрицы M (4, 6).

10. Создайте функцию, параметром которой будет одномерный массив. Перепишите элементы, лежащие между 3-им и 8-ым элементом в обратном порядке. Воспользуйтесь функцией для каждой строки матрицы T (3, 10).

11. Создайте функцию, находящую среднее арифметическое значение из максимального и минимального элементов одномерного массива. Воспользуйтесь данной функцией для каждой строки матрицы E (5, 6).

12. Создайте функцию, вычисляющую разность между количеством отрицательных и количеством положительных элементов одномерного массива. Воспользуйтесь функцией для каждой строки матрицы P (2, 14).

13. Создайте функцию, находящую произведение максимального и минимального элементов одномерного массива. Воспользуйтесь функцией для каждой строки матрицы K (4, 8)

14. Создайте функцию, вычисляющую сумму произведений первого элемента одномерного массива с последним, второго – с предпоследним и т. д. При работе с элементами массива используйте указатели. Воспользуйтесь функцией для каждой строки матрицы T (5, 6).

ЛАБОРАТОРНАЯ РАБОТА № 12

Тема: **Ссылки** (2 часа)

Задание. В алгоритме решения задачи выделите необходимые функции (как правило, повторяющаяся последовательность действий). Функция должна возвращать несколько значений, для этого требуется использование ссылок. В программе продемонстрируйте вызов функции.

Варианты

1. Расположите в порядке возрастания корни квадратных уравнений $x^2 + bx - 4 = 0$ и $ax^2 - 8 + c = 0$ (где ***a, b, c*** – значения, вводимые с клавиатуры). В случае мнимых корней считать их равными нулю.

2. Даны два одномерных массива вещественных чисел C(12) и B(10). Вычислить значение $z = \dfrac{S_A^+ * S_B^-}{S_B^+ + S_A^-}$, где S_A^+ – сумма положительных элементов вектора A. Сумму положительных и отрицательных элементов для одного вектора находить в одной функции.

3. Даны массивы K(14) и C(8). Вычислить значение $z = \frac{S_A^+ * S_C^+}{K_C^+ + K_A^+}$, где S_A^+, K_A^+ – сумма и количество положительных элементов вектора A. Сумму и количество положительных элементов для одного вектора находить в одной функции.

4. Создайте функцию, подсчитывающую по заданным сторонам треугольника величины его углов. Выведите значения полученных углов из главной функции.

5. Создайте функцию, получающую в качестве аргументов некоторый вес, выраженный в килограммах и граммах, а также величину увеличения данного веса в граммах. Функция должна возвращать новый вес, выраженный в граммах и килограммах.

6. Создайте функцию, получающую в качестве аргументов длины сторон треугольника и подсчитывающую его периметр и площадь. Продемонстрируйте работу данной функции на примере трех треугольников, и расположите величины их площади по возрастанию.

7. Создайте функцию, вычисляющую номер максимального элемента матрицы. Продемонстрируйте работу данной функции для матриц A(3, 4), B(2, 3), C(4, 2).

8. Создайте функцию, вычисляющую максимальный и минимальный элементы вектора. Продемонстрируйте ее работу на примере векторов X(10), H(15).

9. Вычислите значение $T = \frac{\max(A) * \min(A+B) - \max(B)}{\max(A+B) + \min(A) * \min(B)}$, где A, B, C – массивы размерности 10. Значение максимума и минимума одного вектора вычислять в одной функции.

10. Создайте функцию нахождения площади поверхности и объема шара по заданному радиусу. Продемонстрируйте ее работу для трех значений радиуса, выведите на печать результаты работы функции из основной программы.

11. Определите функции нахождения суммы и разности векторов в трехмерном пространстве. Для векторов $\vec{a}, \vec{b}, \vec{c}, \vec{d}$, координаты которых заданы с клавиатуры, вычислить $\vec{a} - \vec{c} + \vec{b} - \vec{d}$.

12. Определите функцию нахождения векторного произведения векторов трехмерного пространства. Продемонстрируйте работу функции.

13. Определите функции нахождения суммы векторов на плоскости и умножения вектора на скаляр. Задайте случайным образом координаты векторов $\vec{a}, \vec{b}, \vec{c}, \vec{d}$ и посчитайте $2\vec{a} + 3\vec{c} + \vec{d} + \vec{b}$.

14. По заданным сторонам треугольника определить длины биссектрисы, высоты и медианы для большей из строн.

ЛАБОРАТОРНАЯ РАБОТА № 13

Тема **Структуры. Динамическое распределение памяти**. (6 часов)

Задание. В каждом из вариантов задания для организации данных использовать массив структур. При вводе данных применять методы динамического распределения памяти.

Организовать интерфейс пользователя на основе меню, обязательными пунктами которого будут: ввод данных, выход из программы. Остальные пункты меню согласно варианту. При вводе данных обеспечьте проверку правильности их ввода.

Варианты

1. Составьте программу обработки итогов сессии. Структурный шаблон должен содержать фамилию, имя студента; номер группы; пять оценок, полученных на экзаменах с указанием названия предметов.

По запросу пользователя предусмотреть вывод следующих сведений:

- средний балл определенной группы студентов;
- список неуспевающих студентов с указанием предметов, по которым получены двойки;
- список отличников и их процент от общего числа студентов.

2. Составьте программу, которая реализует компьютерный вариант кассового аппарата. В структурном шаблоне должны содержаться дата, список покупок с указанием их цены, платежная сумма, вносимая покупателем и сумма сдачи, которая должна высчитываться программно. Программа по запросу пользователя предусмотреть вывод всех чеков и процедуру закрытия кассы, т.е. подсчет общей выручки за один день.

3. Создайте компьютерный вариант записной книжки. В структурном шаблоне должны содержаться следующие сведения: Фамилия, Имя, дата рождения, номер телефона. По требованию пользователя предусмотреть вывод

- списка фамилий в алфавитном порядке с указанием номеров телефонов;
- списка лиц, которых необходимо поздравить с Днем рождения в заданном месяце (с указанием числа) с указанием ближайшего именинника.

4. Составьте программу справочной службы аэропорта. Структурный шаблон должен содержать пункт назначения, номер рейса, дату и время вылета, время в полете, стоимость билета, наличие билетов кассе. Время прибытия в пункт назначения высчитывается программно.

По запросу пользователя по названию города предусмотреть вывод информации о рейсах до заданного пункта назначения с указанием времени вылета, времени прибытия, номере рейса, наличие билетов и стоимости билета. Список отсортировать по времени вылета.

5. Составьте программу об ассортименте обуви в магазине. Структурный шаблон должен содержать: наименование; название модели (начинается с буквы M для мужской обуви, с W – для женской); размеры; цвет; количества пар.

По запросу пользователя предусмотреть вывод полного ассортимента обуви в алфавитном порядке, и выдачу информации по неполным данным (по названию и размеру, названию и цвету, отдельно женской и мужской обуви указанного размера).

6. Создайте два массива, содержащие сведения о пяти нападающих каждой из хоккейных команд «СПАРТАК» и «ДИНАМО». Структурный шаблон должен содержать: фамилию; число заброшенных шайб; число, сделанных голевых передач; штрафное время.

Программа по данным из этих массивов, создает новый массив, содержащий фамилию; команду; сумму очков (голы + передачи) для шести лучших игроков обеих команд, и выводит его содержимое на экран.

7. Создайте программу справочной службы железнодорожного вокзала. Структурный шаблон должен содержать: номер поезда, станцию назначения, название остановок (ограничить шестью), время отправления, наличие билетов.

По запросу пользователя предусмотреть

- вывод информации об отправление поездов в указанный пункт назначения (пунктом назначения может служить как конечная станция, так и промежуточная) в указанный временной интервал,
- наличие билетов на поезд с указанным номером.

8. Создайте структурный шаблон содержащий: фамилию, имя, пол, рост, дату рождения человека. По запросу пользователя предусмотреть вывод:

- всех данных в алфавитном порядке фамилий;
- средний рост мужчин,
- фамилии и имени самой молодой женщины, рост которой превосходит средний рост мужчин.

9. Структурный шаблон должен содержать сведения о работниках предприятия: фамилия, имя, год рождения, должность, название отдела, стаж работы, оклад.

Создайте программу, которая позволяет получить список сотрудников пенсионного

возраста в алфавитном порядке с указанием стажа работы и должности, а также средний заработок работающих в указанном отделе.

10. Структурный шаблон должен содержать сведения об автомобилях:
марка, страна-производитель, год выпуска, стоимость.

По запросу пользователя организовать вывод на экран полного ассортимента автомобилей российского производства, отсортированных в алфавитном порядке их марок, вывод по неполным данным (по дате выпуска, марке, марке и дате, стоимости).

11. Структурный шаблон должен содержать сведения о коллекции книголюба: шифр книги, автор, название, год издания.

По требованию пользователя организовать вывод на экран списка книг в алфавитном порядке с указанием автора; число книг издания указанного пользователем года; список книг указанного автора.

12. Структурный шаблон должен содержать данные о пациентах скорой помощи: фамилия, пол, год рождения, домашний адрес, диагноз, информацию о госпитализации (да, нет).

По требованию пользователя организовать вывод на экран списка госпитализированных больных в алфавитном порядке с указанием диагноза; а также список детей с указанным диагнозом, список не госпитализированных больных с указанием домашнего адреса.

13. Структурный шаблон должен содержать сведения об игрушках магазина: название, стоимость, возрастные границы, для которых предназначена игрушка.

По требованию пользователя организовать вывод на экран названий игрушек, цена которых не превосходит указанной пользователем, и которые подходят детям определенного возраста (список игрушек отсортировать в порядке возрастания их цены); вывод перечня игрушек по названию; название игрушек, которые подходят как детям 4 лет, так и детям 10 лет.

14. Структурный шаблон должен содержать сведения багажной квитанции: фамилия, количество вещей, указание веса каждой вещи отдельно и всего багажа в целом (общий вес высчитывать программно).

По запросу пользователя организовать вывод на экран общего веса багажа каждого пассажира; фамилии владельца багажа (в алфавином порядке, средний вес одной вещи в котором отличается не более чем на 0,3 кг от общего среднего веса вещей; определите количество пассажиров, имеющих более трех вещей в багаже.

ЛАБОРАТОРНАЯ РАБОТА № 14

Тема: **Текстовые файлы. Функции библиотеки для работы со строками и символами** (2 часа)

Задание. С помощью текстового редактора создать файл, содержащий текст, длина которого не превышает 1000 символов (длина строки не превышает 70 символов). Имя файла должно иметь расширение TXT. Написать программу, которая выводит текст из файла на экран и выполняет действия в соответствии с вариантом.

Варианты

1. Определяет количество предложений в тексте; по нажатию клавиши выделяет цветом третье предложение.
2. Определяет количество слов в тексте; по нажатию клавиши выделяет цветом десятое слово.
3. Определяет количество слов в тексте, начинающихся с гласной буквы; выделяет шестое слово в тексте по нажатию произвольной клавиши.
4. Определяет количество слов в тексте, у которых первый и последний символы совпадают; выделяет по нажатию клавиши первое из найденных слов.
5. Определят количество предложений, начинающихся с гласной буквы; по нажатию клавиши выделяет первое из найденных предложений.

6. Определяет количество символов в самом маленьком предложении; по нажатию любой клавиши выделяет цветом найденное слово.

7. Определяет в тексте количество символа, введенного с клавиатуры; по нажатию произвольной клавиши выделяет цветом второе и пятое вхождение данного символа.

8. Редактирует текст, удаляя лишние символы пробелов между словами; по нажатию произвольной клавиши выделяет цветом первое предложение.

9. Редактирует текст, заменяя буквы «о» на «а»; по нажатию произвольной клавиши выделяет цветом первое исправленное слово.

10. Редактирует текст, заменяя двойную букву «н» на одинарную; по нажатию произвольной клавиши выделяет цветом первое исправленное слово.

11. Определяет количество восклицательных предложений; выделяет цветом по нажатию произвольной клавиши первое найденное предложение.

12. Подсчитывает количество слов в самом длинном предложении; выделяет цветом по нажатию произвольной клавиши найденное предложение.

13. Определяет количество слов в файле, совпадающих с введенным с клавиатуры словом. По нажатию произвольной клавиши выделяет цветом последнее из найденных слов.

14. Редактирует файл, заменяя в нем повествовательные предложения восклицательными, а восклицательные – вопросительными. По нажатию произвольной клавиши выделяет цветом последнее отредактированное предложение.

ЛАБОРАТОРНАЯ РАБОТА №15
Тема: Бинарные файлы (2 часа)

Задание. Составьте программу, формирующую бинарный файл, т.е. файл, в котором информация хранится в машинных кодах. Считывая данные из файлы, выполните действия в соответствии с заданием.

Варианты

1. В файле содержится некоторое количество чисел. Сформировать из них матрицу, содержащую 4 столбца. Недостающие элементы последней строки обнулить. Вывести на экран матрицу в общепринятом виде, ее размерность и суммы элементов главной диагонали и побочной диагонали.

2. В файле содержатся числа. Сформировать из них матрицу, содержащую пять элементов в строке. Лишние числа отбросить. Вывести на печать матрицу в общепринятом виде и посчитать сумму элементов, содержащихся в предпоследней строке.

3. В файле содержатся числа. Сформировать квадратную матрицу из 25 чисел, читая элементы с конца файла, и посчитать сумму элементов над главной диагональю полученной матрицы.

4. Из цифр, содержащихся в файле сформировать новый файл следующим образом, сначала расположить трехзначные цифры, затем двузначные. Найти максимальное и минимальное число в полученном наборе.

5. В файле содержатся числа, сформировать из них матрицу 4X4, взяв первые 8 чисел сначала файла, а остальные - с конца файла. Посчитать произведение элементов под главной диагональю матрицы.

6. В файле содержатся числа. Сформировать из них матрицу, содержащую пять элементов в строке. Первые десять элементов взять из начала файла, последние пять с его конца. Вывести на печать матрицу в общепринятом виде и посчитать сумму элементов, содержащихся в последней строке.

7. В файле заданы множество точек A и точка d вне его. Найти все пары точек, лежащих с точкой d на одной прямой.

8. В файле задано множество точек на плоскости. Найти все точки, лежащие внутри некоторого треугольника. Треугольник задавать с клавиатуры координатами вершин.

9. В файле задано множество точек на плоскости. Подсчитать количество точек, ле-

жащих внутри указанной окружности. Окружность задавать с клавиатуры координатами центра и радиусом. Вывести на экран координаты найденных точек.

10. В файле задано множество точек на плоскости. Найти из данного множества равноудаленные точки от заданной с клавиатуры точки.

11. В файле задано множество точек в пространстве. Определить множество точек, лежащих в одной плоскости.

12. В файле задано множество точек в пространстве. Подсчитать количество точек, лежащих внутри указанной сферы. Сферу задавать с клавиатуры координатами центра и радиусом. Вывести на экран координаты найденных точек.

13. Описать структуру с именем ZNAK, содержащую следующие поля:
фамилия, имя; знак зодиака; день рождения (массив из трех чисел).

Написать программу, записывающую данные в бинарный файл и отображающую на экран информацию о людях, родившихся под знаком, название которого введено с клавиатуры.

14. Описать структурный шаблон, содержащий поля: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута.

Написать программу, записывающую данные в бинарный файл и отображающую на экран информацию о маршрутах, которые начинаются в пункте, название которого введено с клавиатуры.

ЛАБОРАТОРНАЯ РАБОТА № 16

Тема: ***Объектно-ориентированное программирование. Классы.*** (4 часа)

Задание. В соответствии с вариантом составить класс. Класс должен обязательно содержать явное описание двух видов конструктора – по умолчанию и с параметрами. Объявить объекты созданного класса и продемонстрировать работу всех методов.

Варианты

1. Организовать класс *матрица,* содержащий конструктор, деструктор, функцию вывода матрицы в общепринятом виде, функцию нахождения транспонированной матрицы и определителя матрицы. Продемонстрировать в программе работу всех функций.

2. Организовать класс *треугольник,* определенный по длинам трех сторонам содержащий конструктор, деструктор, функции нахождения периметра и площади (по формуле Герона). Продемонстрировать в программе работу всех функций

3. Организовать класс *треугольник,* определенный по координатам вершин и содержащий конструктор, деструктор, функции нахождения периметра, длин сторон и высоты на большую сторону, площади треугольника. Продемонстрировать в программе работу всех функций.

4. Организовать класс *параллелограмм*, определяемый длиной сторон и меньшим углом и содержащий конструктор, деструктор, функции нахождения периметра и площади треугольника, длин диагоналей. Продемонстрировать в программе работу всех функций.

5. Организовать класс *окружность,* определяемый координатой центра и длиной радиуса. Класс должен содержать конструктор, деструктор, функцию вычисления площади круга и длины окружности. Продемонстрировать в программе работу всех функций

6. Организовать класс *дробь,* содержащий конструктор, деструктор, функцию вывода дроби в общепринятом виде и функцию выделения целой части. Продемонстрировать в программе работу всех функций

7. Организовать класс *дробь,* содержащий конструктор, деструктор, функцию вывода дроби в общепринятом виде и функцию приведения дроби к несократимому виду. Продемонстрировать в программе работу всех функций

8. Опишите класс *Дата,* содержащий данные – число, месяц, год. Опишите конструктор и функцию, проверяющую правильность введенной даты, функцию вывода даты на экран в формате 12.03.2005. Продемонстрировать в программе работу всех функций.

9. Описать класс *Треугольник*, содержащий координаты вершин, конструктор,

функцию, определяющую правильность введения данных, т.е. проверяющую возможность построения треугольника по заданным вершинам, и функцию, рисующую треугольник на экране. Продемонстрировать в программе работу всех функций.

10. Описать класс *Окно* (прямоугольная рамка), с функцией перемещением окна. Продемонстрировать работу всех функций в программе.

11. Описать класс *Вектор*, заданный координатами начала и конца вектора на плоскости. Создать конструктор, деструктор, рисования вектора на экране, нахождения длины вектора. Продемонстрировать работу всех функций в программе.

12. Описать класс *Многочлен*, с полями степень и коэффициенты. Создать конструктор, деструктор, вычисления значения многочлена от аргумента.

13. Описать класс *Студент*, с полями имя, фамилия, год поступления, номер группы, оценки сессии. Создать конструктор, деструктор, методы определения номера курса и среднего балла по последней сессии. Продемонстрировать работу всех функций в программе

14. Описать класс *Множество*, позволяющий добавлять и удалять элементы из множества, пересечение, объединения и разность множеств. Продемонстрировать работу всех функций в программе.

ЛАБОРАТОРНАЯ РАБОТА № 17

Тема: ***Наследование*** (4 часа)

Задание. В соответствии с вариантом создать производный класс, используя методику простого наследования. В производном классе предусмотреть конструктор с параметром, содержащий вызов конструктора базового класса. Создать объекты производного и базового класса, продемонстрировать работу всех методов для этого создать в программе пользовательское меню.

Варианты

1. Организовать класс *треугольник,* определенный по трем сторонам содержащий конструктор, деструктор, функцию нахождения периметра и площади (по формуле Гейрона). Организовать производный класс, содержащий дополнительно функцию нахождения углов треугольника, высоты и новой функции нахождения площади по основанию и высоте. Продемонстрировать в программе работу всех методов класса.

2. Организовать класс *окружность,* определяемый координатой центра и радиуса. Класс должен содержать конструктор, деструктор, функцию вычисления площади круга. Организовать производный класс *конус*, определенный по радиусу основания и координатой вершины, и содержащий функции нахождения площади поверхности и объема конуса. Продемонстрировать в программе работу всех функций

3. Организовать класс *треугольник,* определяемый по величине углов. Заданный класс должен содержать конструктор, деструктор, функцию вычисления площади треугольника. Организовать производный класс, дополнительно содержащий сведения о длине сторон треугольника и вычисляющий его периметр и площадь (причем функция нахождения площади будет иметь другой вид). Продемонстрировать в программе работу всех функций

4. Организовать класс *прямоугольник,* содержащий конструктор, деструктор, функцию нахождения площади. Организовать производный класс *пирамида*, определенный основанием и высотой и содержащий функции нахождения площади поверхности и объема пирамиды. Продемонстрировать в программе работу всех функций

5. Описать класс *Позиция*, определяющий координаты на экране. Описать класс С*трока,* содержащий конструктор и функцию вывода на экран. Описать производный класс – *Позиция* + *Строка,* содержащую функцию вывода строки с нужной позиции

6. Опишите класс *Дата,* содержащий данные – число, месяц, год. Опишите конструктор и функцию, проверяющую правильность введенной даты. Опишите производный класс, включив в дату день недели. Опишите функцию, для определения дня недели, на ко-

торый приходится введенная дата, если считать, что 1-е января 1-го года нашей эры – понедельник, функцию вывода даты на экран.

7. Описать класс *Карта* (масть и достоинство), содержащий конструктор и функцию вывода на экран. Описать производный класс, содержащий 2 карты и козырь, и функцию, проверяющую бъет ли первая карта вторую с учетом козыря. Продемонстрируйте работу функций.

8. Описать класс *Треугольник*, содержащий координаты вершин, конструктор, функцию, определяющую правильность введения данных, т.е. проверяющую возможность построения треугольника по заданным вершинам, и функцию, рисующую треугольник на экране. Описать производный класс *треугольник + цвет,* дополнительно содержащую цвет и функцию, закрашивающую треугольник в заданный цвет. Продемонстрировать в программе работу всех методов класса.

9. Описать класс *Окно* (прямоугольная рамка), с функцией перемещением окна. Описать производный класс *Окно с заголовком,* содержащий конструктор и наследующий все свойства базового класса *Окно* и функцию изменения цвета. Продемонстрировать работу всех функций в программе.

10. Создать класс *Точка*. На его основе создать классы *Цветная точка* и Ц*ветная линия*. Все классы должны иметь методы для установки и получения значений всех координат, а также изменения цвета и получения текущего цвета. Продемонстрировать в программе работу всех методов класса.

ЛАБОРАТОРНАЯ РАБОТА № 18

Тема **Полиморфизм: виртуальные функции, перегрузка операций** (2 часа)

Варианты

1. Выполнить соответствующий вариант задания из предыдущего раздела, но для сложения, деления и умножения комплексных чисел использовать перегрузку соответствующих операций.

2.Выполнить соответствующий вариант задания из предыдущего раздела, но для сложения и произведения матриц использовать перегрузку соответствующих операций.

3.Выполнить соответствующий вариант задания из предыдущего раздела, определив виртуальную функцию для нахождения площади таким образом, чтобы в базовом и производном классе тела функций определялись по-разному.

4.Описать абстрактный класс *Животное*. На его основе реализовать классы Млекопитающее, Рыба, Птица. Классы должны содержать функции вывода значений на экран. Классы должны содержать характеристики животных: название, вид, местообитание. Отдельными характеристиками являются для млекопитающих – травоядное, хищник или всеядное; для рыб – морская или пресноводная; для птиц – перелетная или нет.

5.Создать абстрактный класс средства передвижения. На его основе реализовать класс самолет, машина, корабль. Классы должны иметь параметры средств передвижения: скорость, расход топлива, производителя, год выпуска. Для самолета указать высоту полета, для машины – объем двигателя, для самолета и корабля – количество посадочных мест, для корабля – водоизмещение.

6.Выполнить соответствующий вариант задания из предыдущего раздела, но для вычитания и сложения дробей использовать соответствующие перегруженные операции.

7.Выполнить соответствующий вариант задания из предыдущего раздела, но для умножения и деления дробей использовать соответствующие перегруженные операции.

8.Описать абстрактный класс фигура на плоскости. На его базе создать классы круг, треугольник, прямоугольник. Предусмотреть методы создания объектов, вычисление площади фигур, периметра для треугольника и прямоугольника, длины окружности – для круга.

9.Выполнить соответствующий вариант задания из предыдущего раздела. Функцию, определяющую бьет ли первая карта вторую, определить в базовом классе как виртуальную

и переопределить ее в производном классе

10. Выполнить соответствующий вариант задания из предыдущего раздела. Функцию, рисующую треугольник на экране, определить в базовом классе как виртуальную и переопределить ее в производном классе с закрашиванием треугольника.

11. Выполнить соответствующий вариант задания из предыдущего раздела. Функцию перемещения окна на экране, определить в базовом классе как виртуальную и переопределить ее в производном классе с возможность перемещения вместе с заголовком.

12. Создать абстрактный класс правильный многоугольник. На его основе создать классы треугольник, квадрат, пятиугольник. Предусмотреть методы создания объектов, вычисления их периметра, величины угла.

13. Описать абстрактный класс фигура на плоскости. На его базе создать классы круг, треугольник, прямоугольник. Предусмотреть методы создания объектов, изображения и перемещения их на экране.

14. Выполнить соответствующий вариант №15 задания из предыдущего раздела. Функцию определения текущего времени определить в базовом классе как виртуальную и переопределить ее в производном классе.

Лабораторная работа № 19

Тема: **Графические возможности C++**(4 часа)

Задание. Используя функции графической библиотеки языка C++ выполнить задание в соответствии с вариантом.

Варианты

1.Начертите на экране произвольную прямую линию. Коэффициенты уравнения прямой задайте случайным образом. Также случайным образом задайте координаты 20 точек. Изобразите их на экране, выделив разными цветами точки, лежащие в разных полуплоскостях относительно прямой, а третьим цветом точки, лежащие на прямой. Повторите процесс по желанию пользователя программы.

2. Изобразите 7 концентрических окружностей на экране монитора. Создайте эффект движения «круги на воде», циклически изменяя цвет окружностей от меньшей окружности к большей. Процесс смены цвета окружности повторить 10 раз.

3. Изобразите на экране небольшой круг. Изобразите движение шара на экране по прямой, заданной уравнением $y = kx + b$. Коэффициенты уравнения должны быть заданы счетчиком случайных чисел. При достижении границы экрана происходит изменение траектории движения шара с учетом физического закона о том, что угол падения равен углу отражения

4. Изобразите в правом верхнем углу экрана облако (набором пересекающихся окружностей). Циклически перерисовывая облако, сдвигая изображения по всем координатам, переместите облако в центр экрана, изобразите дождь, падающий из облака.

5. Создать мультфильм «полет НЛО». НЛО изображать в виде отдельных символов:

```
  \ /
=======
```

. Очередное положение НЛО на экране определяется датчиком случайных чисел. Результатом обращения к датчику должны быть номера строки и столбца экрана, с которыми следует совместить левый верхний угол прямоугольника, объемлющего НЛО. Каждый раз, когда номер полученного столбца окажется кратным 5, НЛО должен помять цвет.

```
  / \
```

6. В рисованных мультфильмах иллюзия движения создается последовательной сменой кадров, каждый из которых фиксирует положение объекта. Используя этот принцип получить мультфильм, показывающий человечка выполняющего приседания. Человечка изображать прямыми линиями и кругом.

7. Воспользовавшись технологией предыдущего задания, изобразите мультфильм о гребле, для рисунков воспользуйтесь олимпийской символикой.

8. Из геометрических фигур составьте кораблик. Перерисовыванием в цикле изображение кораблика, создайте иллюзию движения.

9. Изобразите плывущую подводную лодку.

10. Из геометрических фигур составьте домик с окном, по нажатию клавиши в окне должен загораться свет.

11. Изобразить на экране небольшой шар, движущийся по окружности с постоянной скоростью и на каждом обороте, меняющий случайным образом цвет.

12. Создайте программу «восход солнца»

13. Нарисуйте из пересекающихся окружностей облако. По нажатию клавиши из облака «льется дождик».

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

Межсессионный контроль осуществляется на основе выполнения домашних заданий, контрольных работ, лабораторных заданий.

По итогам их выполнения и контроля самостоятельно изученных теоретическим вопросов в сроки, установленные деканатом (как правило, на 6-ой и 12-ой неделе семестра) преподавателем выставляется аттестационная оценка.

VII. КОМПЛЕКТЫ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ

1 семестр

Экзаменационный билет № 1

1. Этапы создания программы
2. Обработка двумерных массивов по строкам
3. Составить блок-схему алгоритма решения задачи: если вещественное число, введенное с клавиатуры положительное вычислить значение его квадратного корня, если отрицательное, возвести в квадрат

Экзаменационный билет № 2

1. Этапы компиляции и выполнения программы
2. Обработка двумерных массивов по столбцам
3. Составить блок-схему алгоритма вычисления значения величины:

$$r(k)=\begin{cases}(k+25)/(k-25), & k>25\\ k+25, & 0\le k<25\\ (k-25)/(k+25) & -25<k<0\end{cases}$$

Экзаменационный билет № 3

1. Линейные алгоритмы
2. Методы сортировок
3. Написать функцию, вычисляющую площадь треугольника по известным длинам сторон треугольника. Длины сторон должны быть параметрами функции

Экзаменационный билет № 4

1. Понятие алгоритма. Способы записи алгоритмов.
2. Оператор цикла с предусловием.
3. В матрице размером 4x5 посчитать сумму положительных элементов каждого столбца матрицы. Составить программу на языке C++.

Экзаменационный билет № 5

1. Разветвляющиеся алгоритмы

2. Функции ввода-вывода (printf( ), scanf( ))
3. В матрице размерностью 8x6 определить номера столбцов, содержащих отрицательные элементы. Составить программу на языке С++.

Экзаменационный билет № 6

1. Циклические алгоритмы
2. Инструкции перехода
3. По заданным длинам катетов трех прямоугольных треугольников определить длину наибольшей гипотенузы. Составить программу на языке С++. Для вычисления гипотенузы использовать функцию.

Экзаменационный билет №7

1. Состав языка С++
2. Двумерные массивы
3. Составить блок-схему алгоритма нахождения номера минимального элемента.

Экзаменационный билет №8

1. Структура программы языка С++
2. Множественный выбор: оператор switch
3. В матрице размерностью 6x6 определить сумму элементов, расположенных над главной диагональю. Составить программу на языке С++

Экзаменационный билет №9

1. Переменные, идентификаторы
2. Директивы препроцессора #define
3. В матрице размерностью 5x8 определить местоположение минимального элемента. Составить программу на языке С++.

Экзаменационный билет №10

1. Типы данных языка С++
2. Определение функций в С++
3. Составить блок-схему алгоритма решения задачи: в одномерном массиве определить среднее арифметическое отрицательных элементов.

Экзаменационный билет №11

1. Основные операции языка С++
2. Оператор цикла с постусловием
3. Составить программу на языке С++. Определить среднее арифметическое объемов цилиндров, заданных радиусом основания и высотой. В программе определить функцию для нахождения объема цилиндра.

Экзаменационный билет №12

1. Описание констант и переменных. Инициализация переменных
2. Разветвляющиеся алгоритмы
3. В матрице размерностью 6x6 определить среднее арифметическое значений каждого столбца. Составить программу на языке С++.

Экзаменационный билет №13

1. Директивы препроцессора #include
2. Строки
3. Составить блок-схему алгоритма решения задачи: в матрице размерностью 5 на 9 определить отрицательных элементов каждого столбца.

Экзаменационный билет №14

1. Библиотека математических функций языка C++
2. Технология создания программы
3. Составить программу на языке C++, выдающую на экран название дня недели по введенному номеру. Предусмотреть обработку неправильно введенного номера.

Экзаменационный билет №15

1. Ввод-вывод с использованием библиотеки iostream (cin>>, cout<<)
2. Область действия и время жизни переменных
3. Составить программу на языке C++: в матрице размерность 8x8 определить произведение элементов, расположенных под главной диагональю.

Экзаменационный билет №16

1. Условный оператор if
2. Циклические алгоритмы
3. Составить функцию нахождения суммы минимального и максимального элементов, найденных из трех значений, заданных в качестве параметров функции.

Экзаменационный билет №17

1. Оператор цикла с заданным числом повторений
2. Понятие алгоритма. Способы записи алгоритмов.
3. Составить программу на языке C++, выдающую по введенному с клавиатуры номеру месяца, время года, к которому он относится. Предусмотреть обработку неверно введенного номера месяца.

Экзаменационный билет №18

1. Одномерные массивы
2. Линейные алгоритмы
3. Определить функцию, находящую значение

$$r(k)=\begin{cases}(k+25)/(k-25), & k>25\\ k+25, & 0\le k<25\\ (k-25)/(k+25) & -25<k<0\end{cases}$$

где k – входной параметр функции.

Экзаменационный билет №19

1. Функции библиотеки conio.h
2. Объявление и определение функций
3. Составить программу на языке C++ табулирования функции $y(t)=\begin{cases}t^2-2t & при\ t<0\\ 2t^4 & при\ t\ge 0\end{cases}$

$t\in[-2;\ 3]$ $\Delta t=0.2$

Экзаменационный билет №20

1. Понятие прототипа функции
2. Оператор выбора switch
3. Написать программу вычисления функции $f(x)=\sum_{n=1}^{45}\left(x^3-13\right)$,

x – ввести с клавиатуры.

2 семестр

Экзаменационный билет № 1

1. Конструкторы
2. Перечисляемый тип
3. Задача. Создайте функцию нахождения количества элементов одномерного массива по модулю превосходящих 10. Для работы с массивами используйте указатели. Продемонстрируйте работу функции, применив ее к целочисленным массивам A(8), B(10)

Экзаменационный билет № 2

1. Связь указателей с массивами
2. Деструкторы
3. Задача. Подсчитайте в текстовом файле количество вопросительных предложений, начинающихся с гласной буквы

Экзаменационный билет № 3

1. Ссылки. Передача аргументов функции по ссылке
2. Виртуальные функции
3. Задача. Опишите понятие *вершина* в виде структуры с элементами: название горной вершины и ее высота. Опишите и инициализируйте массив из пяти вершин. Выведите на экран названия вершин в порядке убывания их высот.

Экзаменационный билет № 4

1. Объявления typedef
2. Перегрузка функций и операций
3. Задача. Создайте функцию нахождения количества элементов одномерного массива по модулю превосходящих 10. Для работы с массивами используйте указатели. Продемонстрируйте работу функции, применив ее к целочисленным массивам A(8), B(10)

Экзаменационный билет № 5

1. Структуры
2. Функции работы с файлами fopen( ), fclose( )
3. Задача. Создайте функцию, меняющую местами в одномерном массиве 1-ый элемент с последним, 2-ой с предпоследним и т.д. Для работы с элементами массива используйте указатели. Примените данную функцию к строкам матрицы B (8, 4).

Экзаменационный билет № 6

1. Объединения
2. Ссылки
3. Задача Создайте класс *Человек*, содержащий данные имя, фамилия, дата рождения, пол. Опишите конструктор и функцию, подсчитывающую количество полных лет. Объявите 6 объектов данного класса и выведите на экран фамилии женщин старше 30 лет

Экзаменационный билет № 7

1. Указатели. Операции с указателями
2. Спецификаторы доступа
3. Задача. Задайте структурный шаблон *собака,* содержащий ее породу, кличку, возраст, окрас. Объявите массив из 6 элементов данного типа. Выведите на печать данные о собаках в алфавитном порядке их пород.

Экзаменационный билет № 8

1. Текстовые и бинарные файлы в C++
2. Простое и множественное наследование

3. Задача. В матрице A(5, 5) найдите номер минимального элемента, расположенного над главной диагональю. Поменяйте его местами с элементом, номер которого вводится с клавиатуры

Экзаменационный билет № 9

1. Функции работы с файлами fwrite( ), fread( ),fseek( )
2. Связь массивов и указателей
3. Задача. Определите структурный шаблон дата: число, месяц, день недели. Создайте массив из 10 элементов, содержащий даты. Выведите на экран только летние рабочие дни, содержащиеся в массиве.

Экзаменационный билет № 10

1. Чтение и запись текстовых файлов
2. Рекурсия
3. Задача. Создайте функцию, которая берет две переменные типа int и заменяет большую из них на произведение переменных, а меньшую на их сумму, деленную на 10 . В программе продемонстрируйте работу данной функции.

Экзаменационный билет № 11

1. Определение класса
2. Функции открытия и закрытия файлов
3. Задача. Вычислите значение $z = \frac{x_1^2 * y_1^2}{\sqrt{x_2 * y_2}}$, где x_i – кони уравнения $x^2 - 5x+c=0$; y_i – корни уравнения $ay^2 +3y – 8=0$. Создайте функцию нахождения корней квадратного уравнения. Значение констант a и c ввести с клавиатуры.

Экзаменационный билет № 12

1. Ключевые принципы объектно-ориентированного программирования
2. Объединения
3. Задача. Заданы радиусы двух окружностей.

Вычислить значение $z = \frac{s_1 + s_2}{l_1 + l_2}$,

где s_i – площадь i-ой окружности, а l_i – ее длина. Значения площади и длины окружности находить в одной функции

Экзаменационный билет № 13

1. Работа со структурами с использованием указателей
2. Функции работы с файлами fwrite( ), fread( ),fseek( )

Задача. Опишите класс *треугольник*, определяемый тремя сторонами и содержащий конструктор и функцию нахождения периметра и площади треугольника. Создайте 3 объекта данного класса, найдите среднее арифметическое площадей треугольников.

Экзаменационный билет № 14

1. Динамическое распределение памяти
2. Конструкторы
3. Задача. В текстовом файле удвойте одиночные пробелы

Экзаменационный билет № 15

1. Понятие класс
2. Ссылки
3. Задача. Создайте программу, которая получает дату (день, месяц), а возвращает общее количество дней вплоть до этой даты. Для задания данных используйте структурный

шаблон.

Экзаменационный билет № 16

1. Простое и множественное наследование
2. Перечисляемый тип
3. Задача. В текстовом файле подсчитайте количество вопросительных предложений.

Экзаменационный билет № 17

1 . Технология создания программы

2. Бинарные файлы

3. Задача. Опишите класс *окружность*, содержащий координаты центра окружности, ее радиус, конструктор и функцию, вычисляющую длину окружности. Создайте три объекта данного класса и выведите на печать координаты центра окружности с наибольшим радиусом и значение ее длины.

Экзаменационный билет № 18

1 . Эволюция систем программирования

2. Бинарные файлы

3. Задача. Опишите класс *матрица*, содержащий размерность матрицы, ее значения, конструктор и функцию, определяющую ее максимальный элемент. Определите два объекта данного класса и посчитайте среднее арифметическое их максимумов

Экзаменационный билет № 19

1 . Критерии качества программы

2. Структуры

3. Задача. В текстовом файле определите количество символов, совпадающих с символом, введенным с клавиатуры

Экзаменационный билет № 20

1 . Дружественные функции и классы

2. Понятие рекурсии

3. Задача. Создайте структурный шаблон погода – день, месяц, температура воздуха, наличие осадков. Создайте программу определения из 20 наблюдений самого холодного летнего дня.

СОДЕРАНИЕ