Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
Амурский государственный университет
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав. кафедрой ИиУС
_____ А.В. Бушманов
«___» _____ 2007 г.

Учебно-методический комплекс дисциплины

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ

для специальности
230201 – Информационные системы и технологии

Составители: Галаган Т.А.
Соловцова Л.А.

2007 г.

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

Алгоритмические языки и программирование для специальности 230201 «Информационные системы и технологии»: учебно-методический комплекс дисциплины. / Галаган Т.А., Соловцова Л.А. – Благовещенск. Изд-во Амурского гос. ун-та, 2007. 87 с.

ОГЛАВЛЕНИЕ

1. РАБОЧАЯ ПРОГРАММА

по дисциплине "Алгоритмические языки и программирование" для специальности 230201 "Информационные системы и технологии"

курс 1 семестр 2

Лекции 36 (час.) Экзамен

Лабораторных занятий 72(час)

самостоятельная работа _56__ (час.)

Всего часов _110___ час.

Составитель: ст.преподаватель Соловцова Л.А..

Факультет Математики и информатики

Кафедра Информационных и управляющих систем

1.1. Цели и задачи дисциплины.

1.1.1. Цели и задачи дисциплины

Целью курса является подготовка студентов к самостоятельной деятельности в области проектирования и сопровождения программных продуктов.

Задачами курса являются освоение алгоритмического языка С++, изучение и использование методов проектирования, отладки и тестирования программных продуктов на языке С++.

1.1.2. Требования к уровню освоения содержания дисциплины

В результате изучения дисциплины студент должен:

- знать принципы построения модульных программ, структурное программирование;
- освоить язык программирования С++;
- знать и уметь работать с основными операторами языка;
- уметь работать со структурными данными, файлами;
- уметь разрабатывать и работать с классами.

1.2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ.

1.2.1. Федеральный компонент.

Дисциплина «Алгоритмические языки» является дисциплиной , входящей в блок общеобразовательных дисциплин федерального компонента для специальности 230201 «Информационные системы и технологии». Государственный стандарт – СД.04

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

Тематический план лекционных занятий

№	Наименование темы	Кол-во часов
1	Введение в С++	2
2	Основные операторы С++	4
3	Функции	4

4	Рекурсивные функции	2
5	Форматный ввод и вывод информации	2
6	Массив как структура данных.	4
7	Обработка строк в С++	2
8	Запись – структурный тип данных С++	2
9	Работа с указателями	6
10	Файлы.	4
11	Классы	4
	Итого	36

Тема 1. Введение в С++. (2 часа)

Структура программы. Идентификаторы. Константы. Объявления переменных. Основные типы данных. Арифметические выражения. Ввод-вывод информации.

Тема 2. Основные операторы алгоритмического языка. (4 часа)

Операторы присваивания. Условный оператор. Операторы цикла.

Тема 3. Функции.(4 часа)

Функция, невозвращающая и возвращающая значение. Параметры функции, передаваемые по значению и по ссылке. Локальные и глобальные переменные.

Тема 4. Рекурсивные функции. (2 часа)

Основные понятия. Механизм работы рекурсивных функций.

Тема 5. Форматный ввод –вывод информации. (2 часа)

Функции для форматного ввода и вывода информации. Описание спецификации.

Тема 6. Массив как структура данных.(4 часа)

Физическое представление массивов. Объявление одномерного и двумерного массивов. Использование массивов как параметров. Инициализация массивов. Методы сортировки массивов.

Тема 7. Обработка строк в С++. (2 часа)

Физическое представление строк. Объявление строк. Функции для их обработки. Инициализация строк.

Тема 8. Запись – структурный тип данных. (2 часа)

Объявление записей. Союзы. Обработка записей.

Тема 9. Работа с указателями (6 часов).

Основные понятия. Инициализация при объявлении. Операции над указателями. Указатели и массивы. Организация динамического массива. Указатели и строки.

Тема 10. Файлы. (2 часа)

Основные понятия. Основные действия при использовании файлов. Создание файла. Обработка файлов. Текстовые файлы. Двоичные файлы.

Тема 11. Классы. (4 часов)

Основные понятия. Создание и использование класса. Конструкторы и деструкторы. Начальная инициализация.

1.2.4. Лабораторные занятия, их содержание и объем в часах.

Тематический план лабораторных занятий.

№	Наименование темы	К-во час.
1	Условные операторы. Использование селективного оператора.	2
2	Функции возвращающие значения	4
3	Функции невозвращающие значения.	2
4	Обработка одномерного массива	2

5	Обработка двумерного массива	2
6	Работа с указателями и массивами	4
7	Обработка строк	4
8	Работа с записями	2
9	Рекурсивные функции	2
10	Обработка двоичного файла	4
11	Обработка текстового файла	4
12	Создание и использование классов	4
	Итого	36

1.2.5. Самостоятельная работа студентов.

Для самостоятельной работы предлагается следующая тема "Изучение динамических структур данных". В рамках этой темы рассмотреть следующие подтемы:

- «Списки»;
- «Стек»;
- «Очередь»;
- «Двусвязный список»;
- «Деревья».

При изучении уделить внимание следующим вопросам:

- физическое и логическое представление в памяти ЭВМ;
- основные приемы работы с данными;
- написание программы для создания и обработки данных.

1.2.6. Вопросы к зачету

1. Данные скалярные типы данных.
2. Стандартные типы данных.
3. Выражения.
4. Операторы ввода-вывода.

5. Оператор присваивания.
6. Составной оператор.
7. Условные операторы.
8. Селективный оператор.
9. Операторы цикла.
10. Основные понятия функций.
11. Параметры подпрограмм.
12. Функции возвращающие значения.
13. Локальные и глобальные переменные.
14. Функции невозвращающие значения
15. Физическое представление массивов.
16. Объявление одномерного и двумерного массивов.
17. Использование массивов как параметров.
18. Методы сортировки массивов.
19. Физическое представление строк.
20. Объявление строк.
21. Функции для обработки строк.
22. Объявление записей.
23. Обработка записей.
24. Основные действия при использовании файлов.
25. Создание файла.
26. Обработка файлов.
27. Замена записей в файле.
28. Текстовые файлы.

1.2.8. Виды контроля

Текущий контроль за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения аудиторных занятий

посредством устного опроса , проведения контрольных работ. Промежуточный контроль осуществляется два раза в семестр на основе анализа аудиторной и самостоятельной работы студента. Итоговый контроль осуществляется после успешного прохождения текущего и промежуточного контроля в виде зачета и устного или письменного экзамена при ответах на два вопроса в билете и дополнительные вопросы экзаменатора.

1.3. Учебно-методические материалы по дисциплине

1.3.1 Перечень обязательной (основной) литературы.

1. Дейл Н.,Уимз Ч., Хедингтон М. Программирование на С++. М.:ДМК, 2000
2. Кнут Д. Искусство программирования. – М.: ИНФРА-М, 2004
3. Турский В. Методология программирования. – М.:Высшая школа, 2001
4. Т.А. Галаган, Л.А.Соловцова. Язык программирования С++ в примерах и задачах.. –Благовещенск: издательство АмГУ, 2005
5. Подбельский. Программирование на языке С++. – СПб.: Питер, 2004.
6. Павловская Т.А. С\С++. Программирование на языке высокого уровня. – СПб.: Питер, 2004.

1.3.1 Перечень дополнительной литературы.

1. Майер Б., Бодуэн К. Методы программирования. – М.: Мир, 1982
2. Паппас К. Мюррей У. Программирование на С и С++. –Киев.: Издательская группа BHV, 2000.
3. Алферова З.В. Теория алгоритмов. – М.: Статистика, 1989

1.5. УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

Номер недели	Номер темы	Вопросы, изучаемые на лекции	Занятия (номера)		Используемые нагляд. и метод. пособия	Самостоятельная работа студентов		Форма контроля
			(семин.)Практич	Лаборат.		Содержание	часы	
1	2	3	4	5	6	7	8	9
1	1	1-7		1	1,2 (осн) 1.2(доп)	Выбор темы	2	
2	2	1-2		2	1,6(осн) 2(доп)	Изучение литературы		
3	2	3		2	1,2 (осн) 1.2(доп)			К.р.
4	3	1		3	1,6(осн) 2(доп)			
5	3	2-3		4	2,3(осн) 2(доп)	Постановка задачи для реализации на ЭВМ и разработка алгоритма		
6	4	1-2		5	4,6(осн) 3(доп)			К.р.
7	5	1-2		6	1,6(осн) 2(доп)			
8	6	1-3		6	1,6(осн) 2(доп)			
9	6	4-6	5	7	4,6(осн) 2(доп)	Написание программы на языке Си++		
10	7	1-4		7	1,6(осн) 2(доп)			
11	8	1-3	6	8	1,5(осн) 1(доп)			
12	9	1-3		9	1,6(осн) 2(доп)			
13	9	4-5	7	10	1,6(осн) 2(доп)	Отладка и тестирование программы		
14	9	6		10	1,6(осн) 2(доп)			К.р.
15	10	1-3	8	11	1,2(осн) 3(доп)			
16	10	4-6		11	1,6(осн) 2(доп)			
17	11	1-2	9	12	2,5(осн) 1(доп)	Составление документации на программу		
18	11	3-4		12	1,6(осн) 2(доп)			

2. ГРАФИК САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Содержание	Объем в часах	Сроки и форма контроля
3.1. Контрольная работа по теме «Циклы»	2 час.	Собеседование (4 неделя)
3.2. Контрольная работа по теме «Обработка массивов»	2 час.	Собеседование (8 неделя)
3.3 Контрольная работа теме «Работа с файлами»	2 час.	Собеседование (15 неделя)

3. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Для самостоятельной работы предлагается следующая тема "Изучение динамических структур данных". В рамках этой темы рассмотреть следующие подтемы:

- «Списки»;
- «Стек»;
- «Очередь»;
- «Двусвязный список»;
- «Деревья».

При изучении уделить внимание следующим вопросам:

- физическое и логическое представление в памяти ЭВМ;
- основные приемы работы с данными;
- написание программы для создания и обработки данных.

4. ПЕРЕЧЕНЬ УЧЕБНИКОВ, УЧЕБНЫХ ПОСОБИЙ

4.1. Перечень обязательной (основной) литературы.

1. Дейл Н.,Уимз Ч., Хедингтон М. Программирование на С++. М.:ДМК, 2000
2. Кнут Д. Искусство программирования. – М.: ИНФРА-М, 2004
3. Турский В. Методология программирования. – М.:Высшая школа, 2001
4. Т.А. Галаган, Л.А.Соловцова. Язык программирования С++ в примерах и задачах.. –Благовещенск: издательство АмГУ, 2005
5. Подбельский. Программирование на языке С++. – СПб.: Питер, 2004.
6. Павловская Т.А. С\С++. Программирование на языке высокого уровня. – СПб.: Питер, 2004.

4.2. Перечень дополнительной литературы.

4. Майер Б., Бодуэн К. Методы программирования. – М.: Мир, 1982
5. Паппас К. Мюррей У. Программирование на С и С++. –Киев.: Издательская группа BHV, 2000.
6. Алферова З.В. Теория алгоритмов. – М.: Статистика, 1989

5. КОНСПЕКТ ЛЕКЦИЙ

Лекция №1

Введение в C++.

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 С++ 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.*

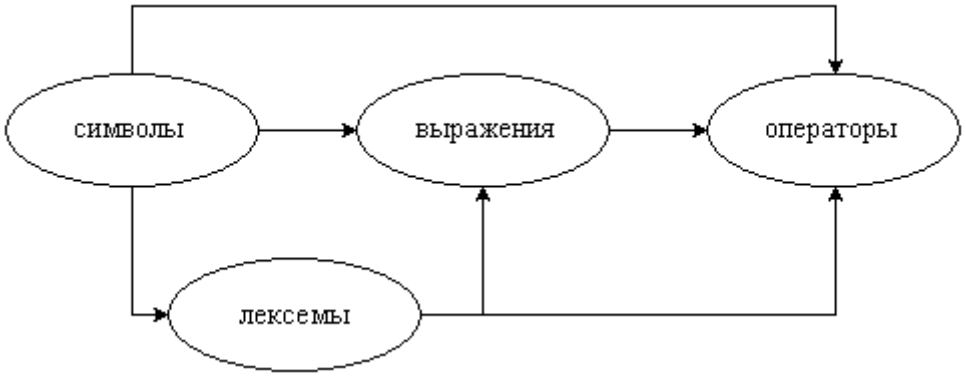


Рисунок 2. Состав алгоритмического языка

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

Для описания стандартных (встроенных) типов в С++ 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.

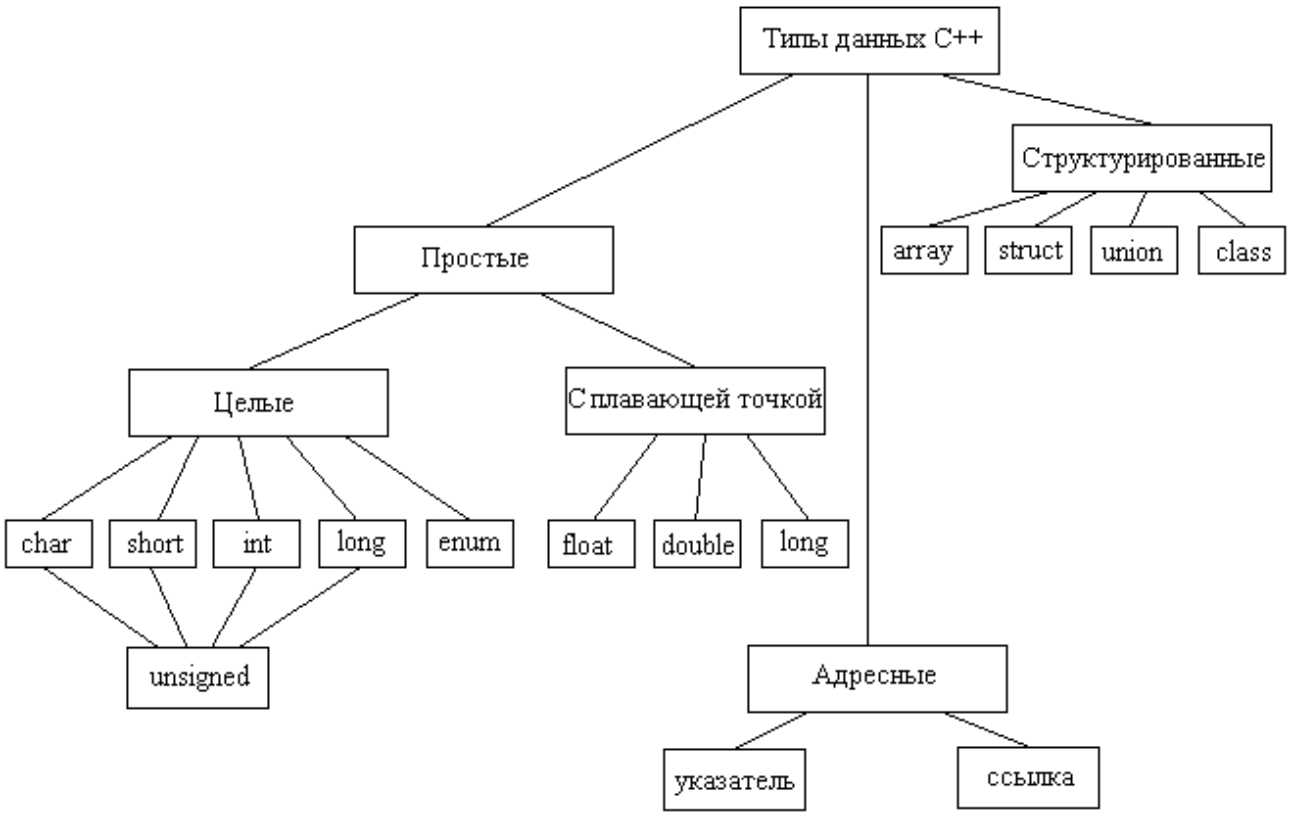


Рисунок 3. Типы данных C++

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;
```

Лекция №2
Основные операторы алгоритмического языка.

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 математическими 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, делением, 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.

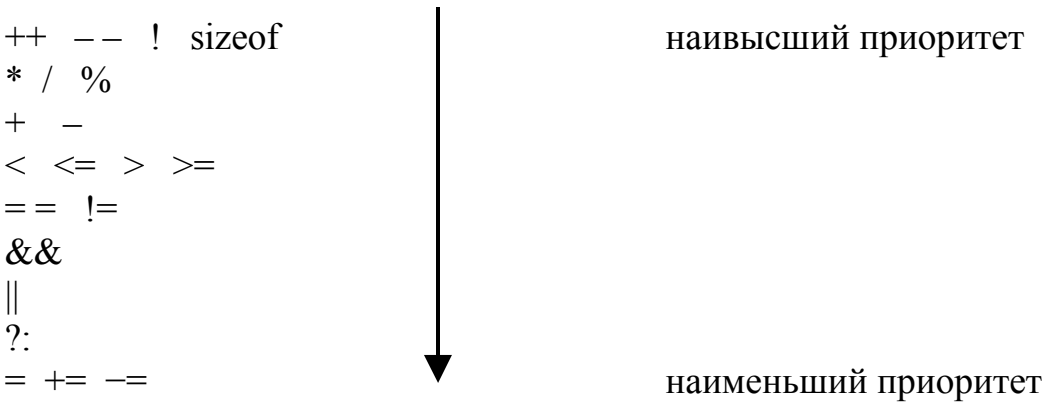


Рисунок 4. Приоритеты операций

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,

```
lnt 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 ближайший 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 C, также доступны и в C++. Наиболее часто используемыми функциями ввода-вывода языка C, 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 C++, включая классы.

Директива #include

Язык C++ 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 C++ (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>
```

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 С 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 С++ для определения встраиваемой функции используется ключевое слово inline. Определение аналогичной функции в программе на С++ 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 С++, 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 -π/2 до π/2
atan(x)	x	Арктангенс x, в диапазоне от -π/2 до π/2
ceil(x)	x	Верхнее значение x (наименьшее целое число ≥ 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 ≤ 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; }
```

Условный оператор if

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

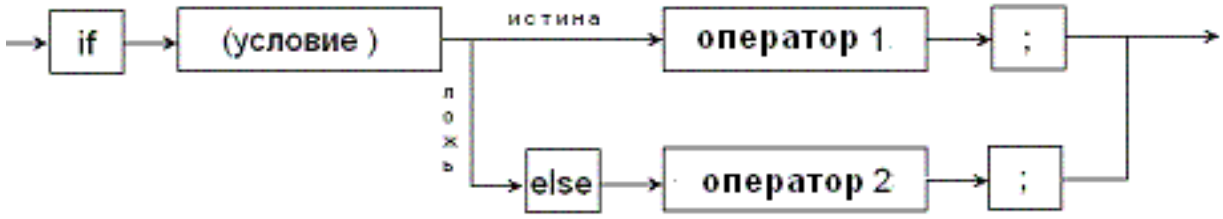


Рисунок 6. Структурная схема условного оператора

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.

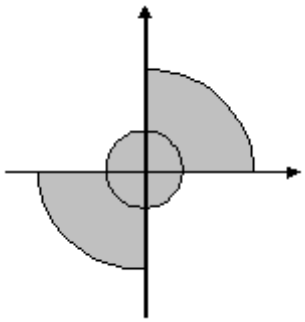
Если требуется проверить несколько условий, их объединяют знакам логических операций.

```
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.

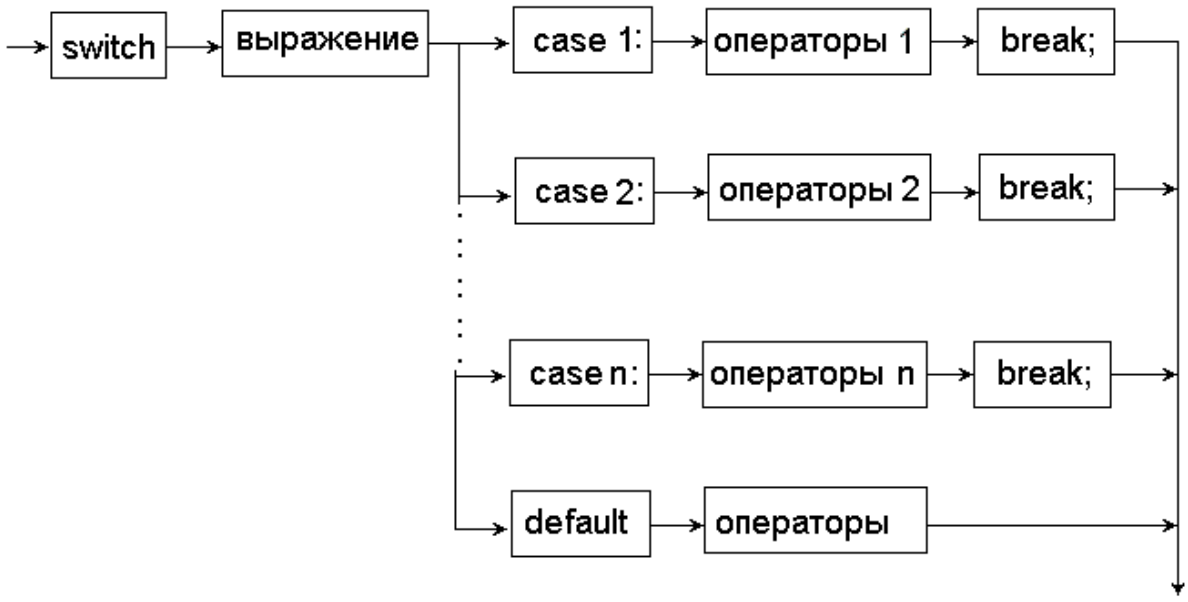


Рисунок 7. Структурная схема оператора switch

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; }
}
```

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 принудительное 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).

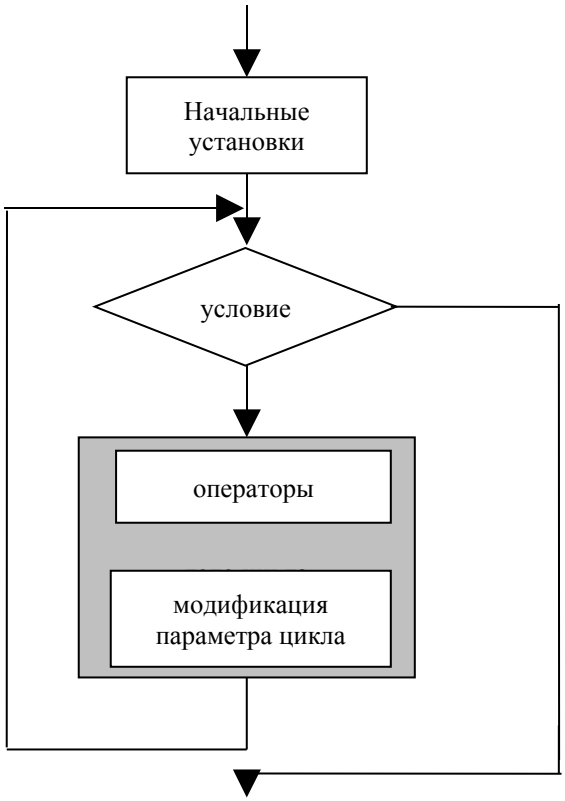


Рисунок 8. Цикл с предусловием

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

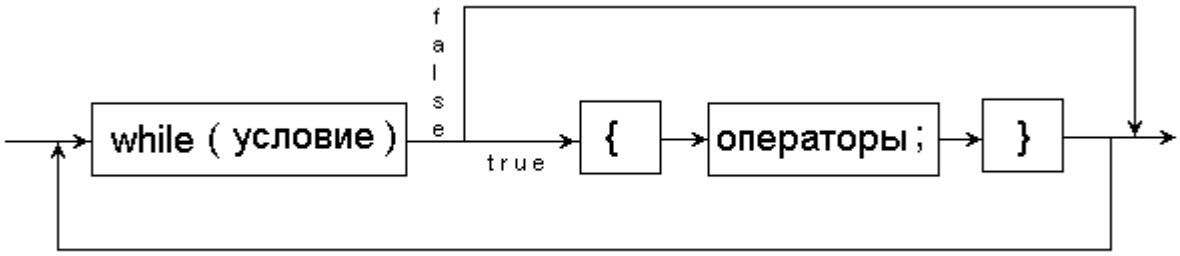


Рисунок 9. Структурная схема оператора цикла while

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:

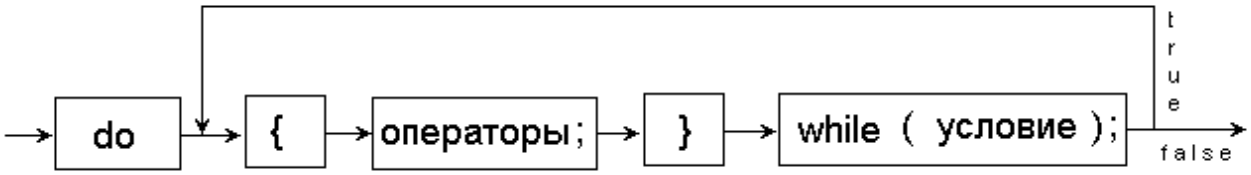


Рисунок 10. Структурная схема оператора цикла do while

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.

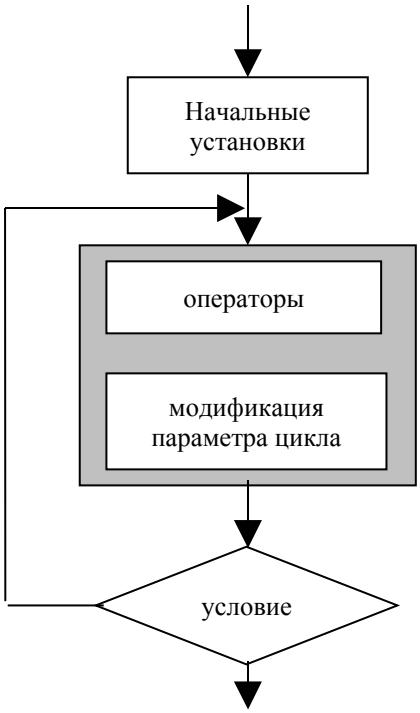


Рисунок 11. Цикл с постусловием

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.

Некоторые структуры вложенных циклов представлены на рисунке 12.

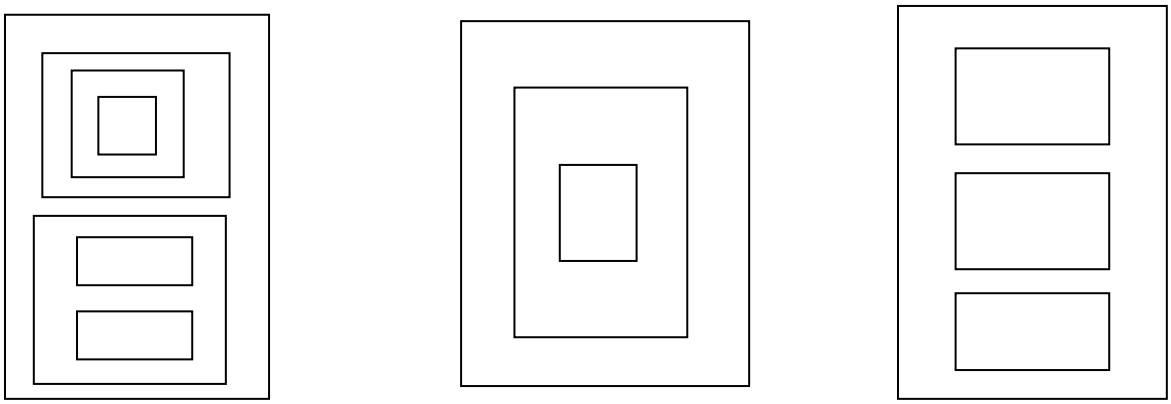


Рисунок 12. Структуры вложенных циклов

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 значение 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.

Лекция №3
Функции.

Ранее говорилось, что любая программа, написанная на языке C++ есть последовательность выполнения функций, причем одна из них обязательно должна называться main( ). Выполнение программы всегда начинается с выполнения ряда операторов этой функции. Все функции в языке C++ 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 ограничения.

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.

Лекция №4
Рекурсивные функции.

Лекция №5
Форматный ввод –вывод информации.

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 цветных сообщений пользуются функцией textcolor(). Закрашивание экрана цветом производится функцией textbackground( ). Цвет задается единственным параметром. Параметрами этих функций могут быть значения:

Цвет	Символьная константа	Числовое значение
Черный	BLACK	0
Синий	BLUE	1
Зеленый	GREEN	2
Голубой	CYAN	3
Красный	RED	4
Пурпурный	MAGENTA	5
Коричневый	BROWN	6
Светло-серый	LIGHTGRAY	7
Темно-серый	DARKGRAY	8
Светло-синий	LIGHTBLUE	9
Светло-зеленый	LIGHTGREEN	10
Светло-голубой	LIGHTCYAN	11
Светло-красный	LIGHTRED	12
Светло-пурпурный	LIGHTMAGENTA	13
Желтый	YELLOW	14
Белый	WHITE	15

Мигание цвета	BLINK	128

В качестве аргумента можно применять как символьные, так и числовые константы. Чтобы задать мигание цвета, следует выполнить, например

textcolor(BLUE+BLINK);

Функция gotoxy( ) – переводит курсор в точку с координатами (x, y).

Функция clrscr( ) очищает экран.

Пример.

```
#include <conio.h>
#include<stdio.h>
void main ( ) {
clrscr( );
textcolor(3+128);
gotoxy(35, 50);
printf("Доброе утро!");
clrscr( );
        }
```

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);
```

Лекция №6
Массив как структура данных

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
```

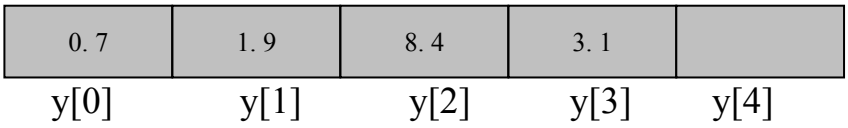
Для доступа к элементу массива после его имени указывается номер элемента (индекс), заключенный в квадратные скобки. Нумерация элементов массива начинается с нуля. Следовательно, диапазон значений для объявленного в примере массива x лежит в пределах от 0 до 2. Следовательно,

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:



В С++ 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.

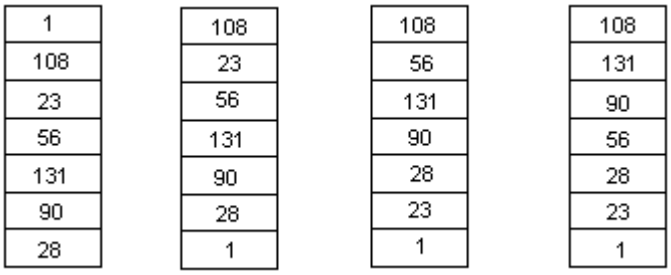


Рисунок 13. Сортировка «пузырьком»

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.

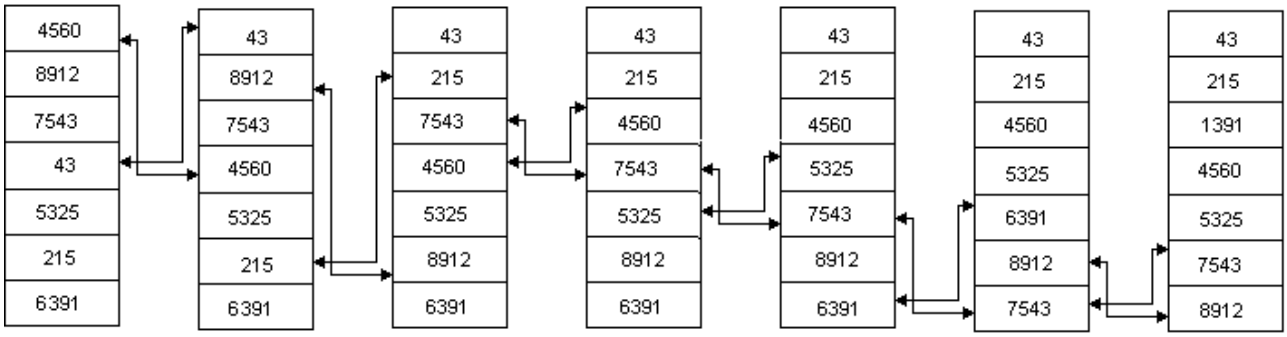


Рисунок 14. Сортировка методом прямого выбора

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]
столбцы		
[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

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];
```

Лекция №7
Обработка строк в С++.

Часто встречаются ситуации, когда требуется обратиться к элементам массива в определенном 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;
                }
        }
```

Язык C++ 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.

Лекция №8
Запись – структурный тип данных.

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 ; }
}
```

Лекция №9
Работа с указателями

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.

[2222] [4250] [6420] [7080]

В результате вычитания целого значения указатель будет ссылаться на элемент, смещенный на указанную величину ячеек влево по отношению к текущей ячейке. Соответственно использование 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);
```

```
}
```

Лекция №10
Файлы.

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 дописывает 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 C + +

Библиотека C++ 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( );
}
```

Лекция №11
Классы.

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 C++. 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 C++ 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

C++ 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 по-разному.

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( ) }
```

6. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1

Условные операторы. Использование селективного оператора

Задание

1. Выполнить табулирование функции. Задать область значений x, y, z и шаги табулирования по x, y, z.

Вариант 1

$$F(x,y,z)=\begin{cases} \ln(x/2.5)+y^2/(z-1) & \text{если } 0<=x<7,\ -5<=y<0,\ z=1 \\ \sqrt{x/y}+z & \text{если } x<0,\ y<-5,\ z<1 \\ (x+5.8*y)/(z-3) & \text{иначе} \end{cases}$$

Вариант 2.

$$F(x,y,z)=\begin{cases} \exp(x/z)+y^2*(x-1.5) & \text{если } 1<=x<5,\ -5<=y<1,\ z<1 \\ x/y+\sqrt{z} & \text{если } x<0,\ y<-5,\ 2>z>1 \\ (1/x-5.8*y)/\sqrt{(z-3)} & \text{иначе} \end{cases}$$

Вариант3

$$F(x,y,z)=\begin{cases} \sin(x*y)/(z-1) & \text{если } -1<=x<3,\ -5<=y<5,\ z>=3 \\ (x+y)/z & \text{если } x<-2,\ y<-5,\ z<1 \\ \ln(z-3/(x-1)) & \text{иначе} \end{cases}$$

Вариант 4

$$F(x,y,z)=\begin{cases} \ln(1/x)+\sin(1/(z-y)) & \text{если } 0<=x<7,\ -5<=y<0,\ z=1 \\ \sqrt{z}+1/(x*y) & \text{если } x<0,\ y<-5,\ z<1 \end{cases}$$

x/(z-y) иначе

Вариант 5

$$F(x,y,z)=\begin{cases} 1/\underline{x+x/\sin(z-y)} & \text{если } 0<=x,\ -5<=y<0,\ z=1 \\ \sqrt{z+4.5*x}+1/(x*y) & \text{если } x<-3.5,\ y<-5,\ z<1 \\ x/\mathrm{Ln}(z-y) & \text{иначе} \end{cases}$$

Вариант 6

$$F(x,y,z)=\begin{cases} z/\underline{x+(2-y)/\sin(z/y)} & \text{если } 0<=x,\ -3<=y<0,\ z>1 \\ \sqrt{z+4.5*x}+1/(x*y) & \text{если } x<-1,\ y<-5,\ z<5 \\ x/\mathrm{Ln}(z-y) & \text{иначе} \end{cases}$$

Вариант 7

$$F(x,y,z)=\begin{cases} 1/\underline{(x-y)/x/\sin}(z-y) & \text{если } -2<=x,\ -1<=y<0,\ z<1 \\ \sqrt{z-5*x}+1/(x*y) & \text{если } x<0,\ -5<y<-1,\ z>5 \\ x/(z-y) & \text{иначе} \end{cases}$$

Вариант 8

$$F(x,y,z)=\begin{cases} \sqrt{1/x+x/\sin(z-y)} & \text{если } 0<=x,\ -3<=y<-1,\ z>1.5 \\ z/(4.5-x)+1/(z*x-y) & \text{если } x<-1,\ y<-5,\ z<1 \\ (x-1)/\sin(z/y) & \text{иначе} \end{cases}$$

Вариант 9

$$F(x,y,z)=\begin{cases} 1/(x-1/y)+x/(z-y) & \text{если } -1<=x,\ -5<=y<1,\ z>3 \\ \sqrt{\sin(z+4.5*x*y)} & \text{если } x<-7,\ y<-5,\ z<1 \\ \mathrm{Ln}(x/(z-y)) & \text{иначе} \end{cases}$$

Вариант 10

$$F(x,y,z)=\begin{cases} \mathrm{Exp}(1/x)+x/(1-y) & \text{если } 0<=x,\ 0<=y,\ z>3 \\ \sqrt{z+4.5/x}+1/\mathrm{Ln}(x*y) & \text{если } -3<x<-1,\ y<-5,\ z<1 \\ \sqrt{x/(z-y)} & \text{иначе} \end{cases}$$

Лабораторная работа №2

Функции невозвращающие значения.

Задание

Вариант 1. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Определить является ли число простым. Формальные параметры – число и признак, который =1, если число простое и 0 – иначе.

Вариант 2. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти наименьшее общее кратное двух заданных чисел.

Формальные параметры – три числа, первое и второе заданные числа, третье – их наименьшее общее кратное.

Вариант 3. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Заданы три положительных числа. Определяют ли эти числа стороны треугольника и если да, то какого равностороннего, равнобедренного или простого.

Формальные параметры – три положительных числа – стороны треугольника и признак, который равен 0 – если числа не определяют треугольник, 1 – простой треугольник, 2 – равнобедренный, 3 – равносторонний.

Вариант 4. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Определить являются ли два целых числа взаимно простыми. (Два числа называются взаимно простыми, если они не имеют общих делителей).

Формальные параметры – два заданных целых числа и признак, который равен 1 – если числа взаимно простые, 0 – иначе.

Вариант 5. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти наибольший общий делитель двух заданных чисел.

Формальные параметры – три числа, первое и второе заданные числа, третье – их наибольший общий делитель.

Вариант 6. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Определить является ли число совершенным. Совершенным называется такое натуральное число, которое равно сумме всех своих делителей, за исключением самого числа, например: 28=1+2+4+4+14.

Формальные параметры – заданное для проверки число и признак, который равен 1 если число совершенно, 0 – в противоположном случае.

Вариант 7. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.

Решить квадратное уравнение.

Формальные параметры – три числа, определяющие коэффициенты квадратного уравнения, два – корни этого уравнения.

Вариант 8. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.

Заданы четыре положительных числа. Могут ли они определять четырехугольник, квадрат, прямоугольник, ромб, параллелограмм.

Формальные параметры – 4 числа, определяющие стороны четырехугольника и признак, принимающий значение 0 – если числа не определяют четырехугольник, 1 – параллелограмм или прямоугольник, 2– ромб или квадрат.

Вариант 9. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.

Задать номер года и определить является ли го високосным (високосные года – это те года , которые делятся на 400, и те, у которых номер делится на 4, но не делится на 100).

Формальные параметры – № года, признак, равный 1 если год високосный, 0– иначе.

Вариант 10. Составить функцию типа void для решения представленной ниже задачи, написать фрагмент программы с ее использованием.

Найти сумму всех простых делителей для заданного числа.

Формальные параметры – заданное число и сумма простых делителей числа.

Лабораторная работа №3

Функции возвращающие значения.

Задание

Вариант 1.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.

Найти с заданной точностью значение выражения

$Y= 1+ 1/x +1/x^2 +1/x^3 + \ldots. \varepsilon <= 10^{-4}$ $x>1$

Вариант 2.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y= 1+ \cos(x) + \cos^2(x) + \cos^3(x) + \ldots\ldots \quad \varepsilon <= 10^{-4}$

Вариант 3.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y= 1+ \sin^2(x) + \sin^4(x) + \sin^6(x) + \ldots\ldots \quad \varepsilon <= 10^{-4}$

Вариант 4.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$$Y= \sum_{n=1}^{\infty}(1/n!) \qquad \varepsilon <= 10^{-4}$$

Вариант 5.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$$Y= \sum_{n=1}^{\infty}(x^n/n) \qquad \varepsilon <= 10^{-4} \quad , \quad |x| < 1$$

Вариант 6.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y=x + x^3/3 + x^5/5 + x^7/7 +\ldots.. \quad \varepsilon <= 10^{-4} \ |x|<1$

Вариант 7.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y=2(1/x + 1/3x^3 + 1/5x^5 + 1/7x^7 +\ldots..) \quad \varepsilon <= 10^{-4} \ |x|>1$

Вариант 8.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y=x - x^3/3 + x^5/5 - x^7/7 +\ldots.$ $\varepsilon <= 10^{-4}$ $|x|<1$

Вариант 9.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y=1 + x^3/3! + x^5\backslash 5! + x^7\backslash 7! +\ldots.$ $\varepsilon <= 10^{-4}$ $|x|<1$

Вариант 10.Составить функцию возвращающую значение для решения представленной ниже задачи, написать фрагмент программы с ее использованием.
Найти с заданной точностью значение выражения

$Y=1 + x^2/2! + x^4\backslash 4! + x^6\backslash 6! +\ldots.$ $\varepsilon <= 10^{-4}$ $|x|<1$

Лабораторная работа №4

Обработка одномерного массива

Задание

Вариант 1. З1.Написать функцию для нахождения самой длинной последовательности подряд идущих элементов массива, равных нулю. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти сумму n элементов отсортированного массива, начиная с k-того.

Вариант 2. З1.Написать функцию для нахождения количества различных чисел в массиве. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти сумму n наибольших значений элементов отсортированного массива.

Вариант 3. З1.Написать функцию для нахождения самой длинной последовательности подряд идущих равных между собой элементов массива. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти сумму n наименьших элементов отсортированного массива.

Вариант 4. З1.Написать функцию для нахождения среднего значения максимального и минимального элементов массива и замены максимальных и минимальных элементов найденным средним. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти количество максимальных элементов.

Вариант 5. З1.Написать функцию для нахождения количества элементов массива равных максимальному и минимальному элементу и замены этих элементов нулевыми значениями. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти количество минимальных элементов.

Вариант 6. З1.Написать функцию для нахождения среднего значения всех элементов массива, начиная с первого минимального и до конца массива и замены этим средним всех минимальных элементов массива. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции выполнить замену всех наименьших значений нулевыми значениями.

Вариант 7. З1.Написать функцию для нахождения среднего значения всех элементов массива, начиная с первого до первого максимального и замены всех максимальных этим средним значением. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей

функции выполнить замену всех наибольших значений средним значением максимального и минимального.

Вариант 8. З1.Написать функцию для нахождения самой длинной последовательности подряд идущих элементов массива, равных какому либо заданному. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти элемент, который расположен в середине массива и определить количество таких элементов в массиве.

Вариант 9. З1.Написать функцию для нахождения среднего значения элементов массива, расположенных между максимальным и минимальным элементом массива и замены всех нулевых значений на это среднее. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти сумму n элементов отсортированного массива, начиная с k-того.

Вариант 10. З1.Написать функцию для нахождения среднего значения элементов массива, расположенных между первым максимальным и последним минимальным элементом массива, и замены элементов равных 1 найденным средним. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию для выполнения сортировки элементов массива методом прямого выбора (массив – параметр функции). В вызывающей функции найти количество наименьших и наибольших элементов массива.

Лабораторная работа №5

Обработка двумерного массива

Задание

Вариант 1. З1. Написать функцию для удаления из двумерной матрицы всех строк, начинающихся 0.
З2. Написать функцию нахождения среднеарифметического минимальных элементов столбцов матрицы.

Вариант 2. З1. Написать функцию преобразования двумерной матрицы в одномерный столбец.

З2. Написать функцию нахождения произведение максимальных элементов строк матрицы.

Вариант 3. З1. Написать функцию удаления из двумерного массива всех строк , в которых находятся максимальные элементы.
З2. Написать функцию нахождения максимума среди минимальных элементов столбцов матрицы.

Вариант 4. З1. Написать функцию преобразования одномерного массива в двумерный. (количество строк задается пользователем, при нехватке элементов одномерного массива – дополнить одномерный массив нулями)
З2. Написать функцию нахождения минимума среди максимальных элементов столбцов матрицы.

Вариант 5. З1. Написать функцию дополнения двумерного массива строкой, элементы которой максимальные элемента массива.
З2. Написать функцию нахождения максимума среди минимальных элементов строк матрицы.

Вариант 6. З1. Написать функцию удаления из двумерного массива всех столбцов , в которых находятся максимальные элементы.
З2. Написать функцию нахождения минимума среди максимальных элементов строк матрицы.

Вариант 7. З1. Написать функцию удаления из двумерного массива среднего столбца и средней строки.
З2. Написать функцию нахождения среднеарифметического всех максимальных значений столбцов матрицы.

Вариант 8. З1. Написать функцию удаления из двумерного массива всех строк с минимальными и максимальными элементами
З2. Написать функцию нахождения среднеарифметического всех минимальных значений столбцов матрицы.

Вариант 9. З1. Написать функцию сравнения двух двумерных массивов и среди отличных элементов двух матриц найти максимальный.
З2. Написать функцию нахождения среднеарифметического всех максимальных значений столбцов и максимальных значений строк матрицы.

Вариант 10. З1. Написать функцию удаления из двумерного массива всех столбцов , в которых находятся минимальные элементы.
З2. Написать функцию нахождения среднеарифметического всех максимальных значений столбцов и минимальных значений строк матрицы.

Лабораторная работа №6

Работа с указателями и массивами

Задание

Вариант 1. 31.Написать функцию для замены всех элементов массива значение которых превышает среднее значение максимального и минимального элемента максимальным элементом , а все значения, меньше среднего – минимальным элементом. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

32. Написать функцию для определения равны ли две матрицы.

Вариант 2. 31.Написать функцию для нахождения среднего значения всех минимальных и максимальных элементов массива и замены им максимальных элементов. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

32. Написать функцию для определения является ли матрица единичной.

Вариант 3. Написать функцию для нахождения средней длины последовательностей подряд идущих элементов массива, равных нулю (последовательность, длина которой равна 1 не учитывать). Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

32. Написать функцию определения является ли матрица симметричной относительно главной диагонали.

Вариант 4. . 31.Написать функцию для нахождения среднего значения элементов массива, расположенных между первым максимальным и последним минимальным элементом массива, и замены элементов равных 1 найденным средним. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

32. Написать функцию нахождения среднеарифметического всех максимальных значений столбцов и минимальных значений строк матрицы.

Вариант 5. .Написать функцию для нахождения среднего значения элементов массива, расположенных между максимальным и минимальным элементом массива и замены всех нулевых значений на это среднее. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

32. Написать функцию нахождения среднеарифметического всех максимальных значений столбцов и максимальных значений строк матрицы.

Вариант 6. . З1.Написать функцию для нахождения самой длинной последовательности подряд идущих элементов массива, равных какому либо заданному. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию нахождения среднеарифметического всех минимальных значений столбцов матрицы.

Вариант 7 . З1.Написать функцию для нахождения среднего значения всех элементов массива, начиная с первого до первого максимального и замены всех максимальных этим средним значением. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2.Написать функцию нахождения среднеарифметического всех максимальных значений столбцов матрицы.

Вариант 8. . З1.Написать функцию для нахождения среднего значения всех элементов массива, начиная с первого минимального и до конца массива и замены этим средним всех минимальных элементов массива. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию нахождения минимума среди максимальных элементов строк матрицы.

Вариант 9. З1.Написать функцию для нахождения количества элементов массива равных максимальному и минимальному элементу и замены этих элементов нулевыми значениями. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

З2. Написать функцию нахождения максимума среди минимальных элементов строк матрицы.

Вариант10. З1.Написать функцию для нахождения среднего значения максимального и минимального элементов массива и замены максимальных и минимальных элементов найденным средним. Массив должен передаваться в функцию в качестве параметра. Выполнить инициализацию массива при его объявлении.

.З2 Написать функцию нахождения минимума среди максимальных элементов столбцов матрицы.

Лабораторная работа №7

Обработка строк

Задание

Вариант 1. Написать функцию для удаления из массива строк заданной подстроки.

Вариант 2. Написать функцию для подсчета в массиве строк гласных букв.

Вариант 3. Написать функцию для подсчета в массиве строк количества слов, разделителем слов считать пробел и ;.

Вариант 4. Написать функцию для определения местоположения различных символов массива строк.

Вариант 5. Написать функцию для удаления из массива строк одинаковых слов, разделителем слов является пробел.

Вариант 6. Написать функцию для определения количества вхождений заданной подстроки в массив строк.

Вариант 7. Написать функцию для выделения и вывода на экран предложений из массива строк, ограничитель предложения – точка справа.

Вариант 8. Написать функцию для выделения всех слов из массива строк и сортировки этих слов по алфавиту.

Вариант 9. Написать функцию для выделения всех слов из массива строк, начинающихся с заданной подстроки.

Вариант 10. Написать функцию для выделения всех слов из массива строк, заканчивающихся заданным символом.

Лабораторная работа №8

Работа с записями

Задание

Вариант 1. Написать функцию для обработки массива записей, содержащей следующие поля:

- название музыкального альбома;
- год выпуска;
- жанр;
- страна-изготовитель;

– цена.

Выдать на экран все записи отсортированные по первому полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 2. Написать функцию для обработки массива записей, содержащей следующие поля:

– название газеты или журнала;
– вид СМИ (газета или журнал);
– город, где издается СМИ;
– тираж;
– цена.

Выдать на экран все записи отсортированные по третьему полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 3. Написать функцию для обработки массива записей, содержащей следующие поля:

– наименование товара;
– производитель;
– год выпуска;
– партия;
– количество;
– цена.

Выдать на экран все записи отсортированные по второму полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 4. Написать функцию для обработки массива записей, содержащей следующие поля:

– название фильма;
– год выпуска;
– киностудия-производитель;
– жанр фильма;
– режиссер.

Выдать на экран все записи отсортированные по третьему полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 5. Написать функцию для обработки массива записей, содержащей следующие поля:

– марка автомобиля;
– страна-изготовитель;

– вид двигателя;
– год выпуска;
– цена.

Выдать на экран все записи отсортированные по пятому полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 6. Написать функцию для обработки массива записей, содержащей следующие поля:

– название книги;
– автор;
– год издания;
– тираж;
– цена.

Выдать на экран все записи отсортированные по второму полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 7. Написать функцию для обработки массива записей, содержащей следующие поля:

– код специальности;
– название специальности;
– название факультета;
– время обучения;
– размер платы за обучение.

Выдать на экран все записи отсортированные по пятому полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 8. Написать функцию для обработки массива записей, содержащей следующие поля:

– номер группы;
– специальность;
– факультет;
– плановое количество студентов;
– количество студентов, обучающихся в настоящее время.

Выдать на экран все записи отсортированные по третьему полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 9. Написать функцию для обработки массива записей, содержащей следующие поля:

– название телепередачи;

– вид телепередачи;
– автор;
– время показа;
– рейтинг.

Выдать на экран все записи отсортированные по второму полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Вариант 10. Написать функцию для обработки массива записей, содержащей следующие поля:

– № заявки на учебную дисциплину;
– название дисциплины;
– специальность;
– курс;
– количество часов.

Выдать на экран все записи отсортированные по третьему полю в виде таблицы. Массив должен содержать не менее 10 записей. Выполнить инициализацию полей записей.

Лабораторная работа №9

Рекурсивные функции

Задание

Вариант 1. З1. Написать функцию, возвращающую значение и реализующую рекурсивную формулу

*F(N) = F(N-1)*1.7/3.1*

С граничным условием F(1)=1.

З2. Написать рекурсивную функцию для вывода на печать всех элементов массива A(N x N).

Вариант 2. З1. Написать функцию, возвращающую значение и реализующую рекурсивную формулу

F(N) = F(N-2) + 3.5

С граничным условием F(1)=1, F(0)=0.

З2. Написать рекурсивную функцию для нахождения суммы всех элементов массива A(N).

Вариант 3. З1. Написать функцию, возвращающую значение и реализующую рекурсивную формулу

F(N) = F(N-1) + F(N-2)/3

С граничным условием F(0)=1, F(1)=1.

З2. Написать рекурсивную функцию для вычисления X^n.

Вариант 4. З1. Написать функцию, возвращающую значение и реализующую рекурсивную формулу

F(N) = F(N-1) + F(N-2) – F(N-3)

С граничным условием F(2)=2; F(1)=1; F(0)=1.

З2. Написать рекурсивную функцию для нахождения произведения всех элементов массива A(N).

Лабораторная работа №10

Обработка файлов

Задание

Вариант №1

1. В файле целых чисел заменить все нулевые значения максимальным значением.
2. В текстовом файле посчитать количество заданных букв.

Вариант №2

1. В файле целых чисел среднеарифметическое.
2. В текстовом файле посчитать количество.

Вариант №3

1. В файле целых чисел заменить все нулевые значения среднеарифметическим всех значений из файла.
2. В текстовом файле найти самое длинное предложение.

Вариант №4

1. В файле целых чисел найти максимальное значение.
2. В текстовом файле посчитать количество предложений.

Вариант №5

1. В файле целых чисел найти сумму положительных и отрицательных значений.
2. В текстовом файле посчитать количество вхождений подстроки.

Вариант №6

3. В файле целых чисел заменить все значения равные максимальному нулевыми значениями.
4. Из текстовом файле удалить заданную подстроку.

Вариант №7

1. В файле целых чисел посчитать количество нулевых значения.

2. В текстовом файле посчитать количество предложений.

Вариант №8

1. В файле целых чисел заменить все нулевые значения максимальным значением.
2. В текстовом файле посчитать количество заданных букв.

Вариант №9

1. Из файла целых чисел удалить все нулевые значения.
2. В текстовом файле посчитать процентное отношение гласных и согласных букв.

Вариант №10

1.В файле целых чисел найти сумму четных и нечетных чисел.
2.В текстовом файле удалить все гласные буквы.

Лабораторная работа №11

Обработка текстового файла

Задание

Вариант 1. Определить количество предложений в тексте; по нажатию клавиши выделяет цветом третье предложение.

Вариант 2. Определить количество слов в тексте; по нажатию клавиши выделяет цветом десятое слово.

Вариант 3. Определить количество слов в тексте, начинающихся с гласной буквы; выделяет шестое слово в тексте по нажатию произвольной клавиши.

Вариант 4. Определить количество слов в тексте, у которых первый и последний символы совпадают; выделяет по нажатию клавиши первое из найденных слов.

Вариант 5. Определить количество предложений, начинающихся с гласной буквы; по нажатию клавиши выделяет первое из найденных предложений.

Вариант 6. Определить количество символов в самом маленьком предложении; по нажатию любой клавиши выделяет цветом найденное слово.

Вариант 7. Определить Определяет в тексте количество символа, введенного с клавиатуры; по нажатию произвольной клавиши выделяет цветом второе и пятое вхождение данного символа.

Вариант 8. Редактировать текст, удаляя лишние символы пробелов между словами; по нажатию произвольной клавиши выделяет цветом первое предложение.

Вариант 9. Редактировать текст, заменяя буквы «о» на «а»; по нажатию произвольной клавиши выделяет цветом первое исправленное слово.

Вариант 10. Редактировать текст, заменяя двойную букву «н» на одинарную; по нажатию произвольной клавиши выделяет цветом первое исправленное слово.

Лабораторная работа №12

Создание и использование классов

Задание

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.

7. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

1. Межсессионная аттестация студентов проводится дважды в семестр на 7 и 13 неделях 2-го семестра семестра.

2. Аттестационная оценка складывается из оценок, полученных за контрольные работы по результатам промежуточного тестирования.

3. Организация аттестации студентов, проводится в соответствии с положением АмГУ о курсовых экзаменах и зачетах.

8. ЗАДАНИЯ ДЛЯ ПРОВЕДЕНИЯ ЗАЧЕТА

Задание 1
Пусть даны значения логических переменных x, y, z:
X=true, y=false, z=true
Вычислить следующие логические выражения:

1. x && y || x && z
2. (x || !y) && (!x || z)
3. x || y && z
4. !(x || y) && z

Задание 2.
Даны переменные I, j, m, n типа int:
Каким будет результата выполнения следующего фрагмента?

```
cout << "Madam";
if  (i < j)
   if (m != n)
      cout << "How";
   else
      cout << "Now";
cout << "I'm";
if (i  >= m )
  cout << "Cow";
else
  cout << "Adam";
```

Задание 3.
Что выводит следующий цикл?

```
number = 1;
 while (number < 11)
   {
    number++;
    cout << number << endl;
   }
```

Задание 4.
Предполагается, что следующий код должен напечатать четные числа в диапазоне от 1 до 15, но в нем есть две ошибки.

```
n=2;
while (n!=15)
{  n=n+2;
  cout << n << ' ';
 }
```

Здание 5.

Указать в приведенной ниже программе следующие элементы: прототип функции, заголовок функции, фактические параметры, локальные переменные, описание функции, формальные параметры, вызов функции, тело функции.

```
void Test ( int, int, int);
int main ()
{ int a, b, c;
…….
   Test(a, c, b);
   Test(b, a, c);
…….
}
void Test ( int d, int e, int f)
{ int g;…….}}
```

9. КАРТА КАДРОВОЙ ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ

Лектор – доцент Галаган Татьяна Алексеевна

Руководитель лабораторных работ – старший преподаватель Соловцова Любовь Александровна

Татьяна Алексеевна Галаган,
доцент кафедры ИиУС АмГУ
Любовь Александровна Соловцова
Старший преподаватель кафедры ИиУС АмГУ

Учебно-методический комплекс по дисциплине «Алгоритмические языки и программирование»

Изд-во АмГУ. Подписано к печати Формат 60х84/16. Усл. печ. л.
Тираж Заказ