МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний автомобільно-дорожній університет

Симбірський Г.Д.

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни “Інформаційні технології”
за напрямом підготовки 6.050702 “Електромеханіка”
(Розділ “Мова програмування *C*++”)

Харьков – 2014

УДК 004.4(075.8)
ББК 22.18я7

Симбірський Г.Д. Конспект лекцій з дисципліни “Інформаційні технології” за напрямом підготовки 6.050702 “Електромеханіка” (розділ “Мова програмування *C*++”). – Харків: ХНАДУ, 2014, - 150 с.

Розглянуті синтаксис, семантика та техніка програмування на мові *C*++ у середовищі *Microsoft Visual Studio* 2010. Проаналізована велика кількість програм, що ілюструють можливості і особливості мови *C*++ та програмного пакету *Microsoft Visual Studio* 2010. Конспект лекцій містить необхідні теоретичні відомості, приклади рішення конкретних задач з текстами програм та схемами алгоритмів і контрольні запитання для самоперевірки.

Оглавление

ВВЕДЕНИЕ

Язык *C* (читается Си) в основе своей был создан в 1972 г. как язык для операционной системы *UNIX*. Автором этого языка считается Деннис Ритчи.

Популярность языка *C* была обусловлена, прежде всего, тем, что на нем были написаны большинство операционных систем. Несколько лет единая политика в стандартизации языка ***C*** отсутствовала. В начале 1980-х в Американском национальном институте стандартов (*ANSI*) началась работа по стандартизации языка ***C***. В 1989 г. работа комитета по языку *C* была ратифицирована, и в 1990 г. был издан первый официальный документ по стандарту языка *C*89.

Изначально язык *C* предназначался для системного программирования при создании операционных систем, системных утилит и встраиваемого программного обеспечения. Он обладает всеми необходимыми для этого свойствами: программы, написанные на нем, эффективны, не требуют специальной среды поддержки и значительного времени для выполнения.

Язык программирования *C*++ был создан в результате дальнейшего развития языка *C* и вначале назывался “язык *C* с классами”. В настоящее время *C*++ широко применяется из-за его стабильности и обширного окружения (стандартные библиотеки, компиляторы и другие инструментальные средства), а также из-за возможности получения программ, выполняющихся с максимальной скоростью на данной аппаратной платформе. Более того, язык *C*++ можно использовать и для создания веб-сайтов через технологию *CGI* (*Common Gateway Interface* – общий шлюзовый интерфейс). Компиляторы, библиотеки и инструменты разработки на языке *C* существуют практически для всех операционных систем. Программы на языке *C*++ отличаются переносимостью между платформами на уровне исходного кода.

Важным аспектом языка *C*++ является его структурированность, что обусловлено использованием блоков. Блок – это набор инструкций, которые логически связаны между собой. Строительными блоками языка *C*++ являются функции.

Данный конспект лекций (далее – конспект) освещает практические приемы программирования на языке *C++* в среде программирования *Microsoft Visual Studio* 2010.

Конспект предназначен для начального изучения языка *C++* в течение одного семестра из расчета 72 аудиторных академических часов, из которых 36 часов – лекционные. Пособие состоит из 18 лекций (тем), каждая из которых содержит теоретическую и практическую части. В теоретической даются основные конструкции языка программирования *C++*, которым посвящена та или иная тема. В практической части приведены примеры, задания, представлены блок-схемы алгоритмов их решения, программные коды и результаты выполнения. После изучения каждой темы необходимо ответить на контрольные вопросы.

Основной своей задачей автор видел создание учебного пособия, пользуясь только которым практически неподготовленный студент мог бы освоить как общие принципы программирования при решении инженерных задач, так и конкретные приемы при использовании языка *C++* в *Microsoft Visual Studio* 2010. Автором был проведен сравнительный анализ достаточного количества учебников и пособий по программированию на *C++* и сделаны выводы об их достоинствах и недостатках.

Наиболее полными и последовательными из этих источников представляются книги американского автора А. Хортона (см. раздел Литература), который рассматривает оба основных стандарта языка *C++* от самых начальных сведений до высокопрофессиональных приемов программирования в среде *Visual C++* 2010. Но для изучения студентами эти пособия являются сложными по характеру изложения и по огромному количеству информации, а примеров решения конкретных задач, особенно инженерных, явно недостаточно.

В представленном Вашему вниманию конспекте автор сделал попытку избежать упомянутых недостатков. Из-за ограниченного объема не всегда удавалось дать комментарии к строкам программных кодов. В полном объеме конспект представлен на образовательном портале ХНАДУ. Там же присутствуют и методические указания к лабораторным работам для данного курса.

Основой для лекций 1, 2, 3, 14, 15, 16 послужили лекции коллектива авторов [3].

ЛЕКЦИЯ 1
ОСНОВНЫЕ ПОНЯТИЯ И СХЕМЫ АЛГОРИТМОВ ПРИ РЕШЕНИИ ИНЖЕНЕРНЫХ ЗАДАЧ

Цель лекции. Изучение основных понятий и схем алгоритмов.

Основные вопросы лекции

1. Основные понятия и определения при алгоритмизации задач.
2. Алгоритмы и схемы простых операций.
3. Алгоритмы и схемы составных операций.
4. Виды алгоритмов.

1.1. Основные понятия и определения при алгоритмизации задач

Решение любой математической задачи на компьютере включает следующие этапы:

1. Разработка **математической** модели задачи, т. е. математическое описание объекта исследований.
2. Разработка метода решения задачи.
3. Составить алгоритм решения задачи, т. е. определить последовательность действий.
4. Разработка программы для решения задачи на одном из языков программирования.
5. Устранение возможных ошибок при выполнении программы.
6. Анализ полученного результата.

Таким образом, **алгоритм решения задачи** - это строгая последовательность действий, которые необходимо выполнить над данными, чтобы получить искомый результат (решить поставленную задачу).

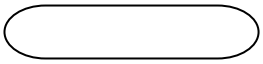
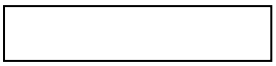
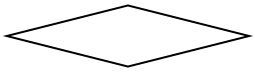
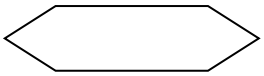
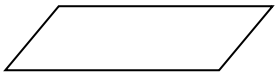
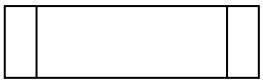
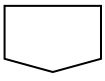
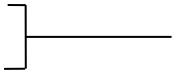
Тогда **программа** – это записанный на языке программирования алгоритм решения задачи.

Каждый язык программирования или алгоритмический язык является набором символов и слов, с помощью которых записываются команды (инструкции) для компьютера,

совокупность которых преобразуется соответствующей программой (компилятором) в машинный код для дальнейшего выполнения компьютером. В программировании алгоритм решения задачи принято изображать в графическом виде. При этом все операции или действия изображаются в виде отдельных блоков. Каждое действие (операция), например, ввод данных, печать документа и пр., имеет свое стандартное условное обозначение (таб. 1.1).

Таблица 1.1

Основные операционные блоки схем алгоритмов

№	Условное обозначение	Наименование операции	Описание операции
1		Начало и завершение	Начало и завершение алгоритма
2		Процесс (присваивание)	Вычислительная операция или их совокупность
3		Решение	Проверка условия и выбор направления процесса
4		Модификация	Заголовок цикла, проверка условий цикла
5		Данные	Ввод и вывод исходных данных и результатов
6		Типовой процесс	Использование ранее созданных алгоритмов, подпрограмм, функций
7		Печать документа	Вывод данных на печать
8		Соединитель внутристраничный	Разрыв линий потока в пределах одной страницы
9		Соединитель межстраничный	Перенос линий потока на другую страницу
10		Узел	Слияние линий потока
11		Комментарии	Описание операционного блока и его особенностей

Конфигурация и размер блоков определяются Государственным стандартом. Последовательность действий, необходимых для решения поставленной задачи, изображенная в виде набора стандартных операционных блоков, называется **блок-схемой алгоритма.**

В блок-схемах алгоритмов операционные блоки соединяются друг с другом линиями потока со стрелками, указывающими направление. Линии потока, направленные вниз и направо могут быть без стрелок.

1.2. Алгоритмы и схемы простых операций

Простое действие – это одна операция. Основные простые действия следующие: - присваивание; - ввод и вывод; выбор.

Присваивание – действие, в результате которого переменная получает определенное значение. Операционный блок присваивания в схемах алгоритмов обозначается символом “процесс” (прямоугольник). Варианты использования блока присваивания приведены на рис. 1.1.

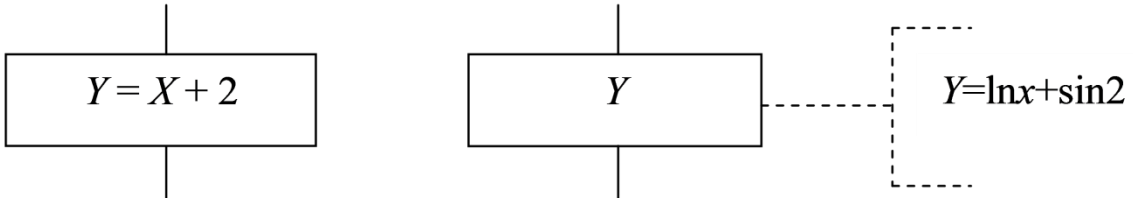


Рис. 1.1. Операционный блок **Присваивание**

В середине прямоугольника записывается команда или имя переменной. Если математическое выражение имеет сложный вид, тогда справа записывается комментарий к блоку (рис. 1.1).

Ввод – действие, в результате которого переменной присваивается начальное значение.

Вывод - действие, в результате которого данные выводятся для отображения.

Операционный блок для обозначения ввода или вывода информации в схемах алгоритмов обозначается символом “данные” (параллелограмм). Варианты оформления блока “данные” представлены на рисунке 1.2.

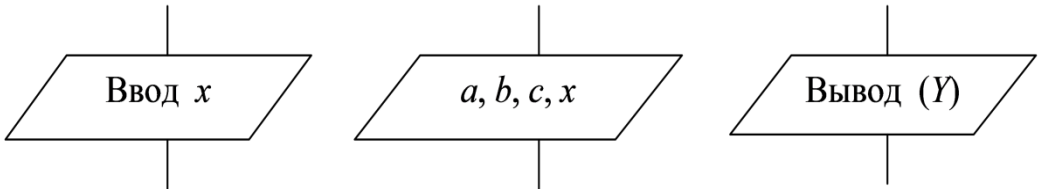


Рис. 1.2. Операционный блок **Ввод** (**Вывод**)

Исследуем процесс разработки алгоритма программы для решения простейшей задачи на следующем примере.

Необходимо вычислить силу тока *I* в новогодней гирлянде, состоящей из *n*=50 электрических лампочек сопротивлением *r*=20 Ом каждая. Используя закон Ома и формулу для расчета суммарного сопротивления последовательной цепи, составим блок-схему алгоритма (рис. 1.3). Перед началом расчетов вводятся значения переменных *r, n, U* (блок "данные"). Затем идет операционный блок, в котором рассчитываются общее сопротивление гирлянды *R* и сила тока *I*. Вычисленное значение *I* необходимо вывести на экран (блок "данные").

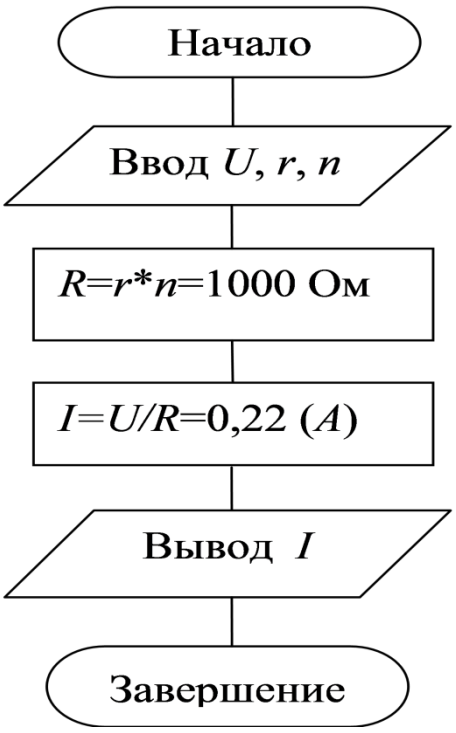


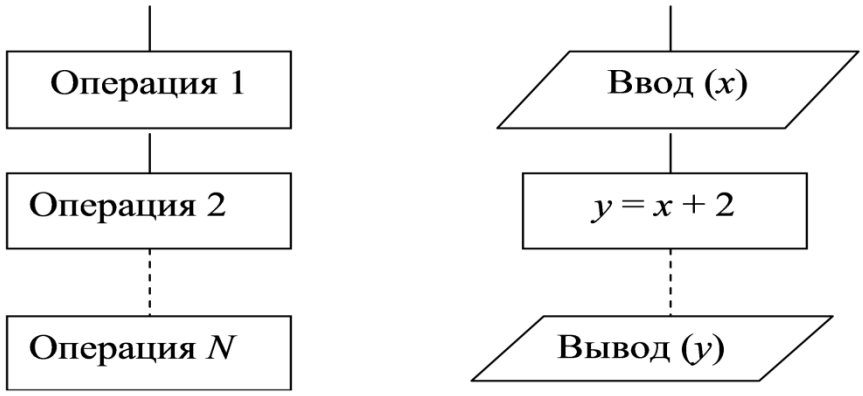
Рис. 1.3. Блок-схема алгоритма определения силы тока в новогодней электрогирлянде

1.3. Алгоритмы и схемы составных операций

Составные операции состоят из простых операций и условий их выполнения и включают:

- прохождение;
- выбор или разветвление;
- повторение или цикл.

Прохождение – последовательность простых операций, выполняемых одна за другой (рис. 1.4).



а) б)

Рис. 1.4. Схема алгоритма составной операции **Прохождение:** а) общий вид; б) пример

Выбор (разветвление) – это прохождение вычислительного процесса по одному из двух возможных направлений алгоритма в зависимости от некоторого условия (рис. 1.5).

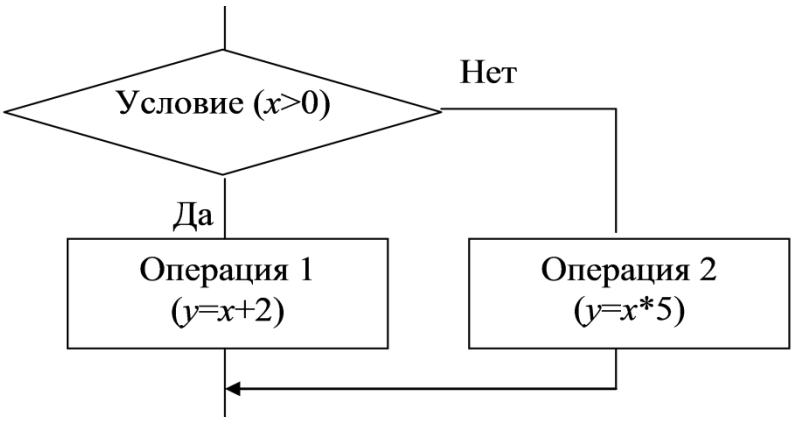


Рис. 1.5. Схема алгоритма составной операции **Выбор**

При выполнении операции **Выбор** проверяется, удовлетворяет ли значение переменной x некоторому условию. Если условие

выполняется, то производится **Операция 1** (направление потока **Да**), в противном случае – **Операция 2** (направление **Нет**). В зависимости от условия задачи блок **Операция 2** может отсутствовать.

Повторение (цикл) – многократное выполнение одной и той же последовательности операций в зависимости от некоторого условия.

Для организации повторения операций используется переменная, называемая **параметр цикла.** Последовательность действий, которая повторяется, называется **телом цикла**. Условие проверяется до или после тела цикла (рис. 1.6).

Цикл выполняется до тех пор, пока параметр цикла удовлетворяет условию. В случае, если параметр цикла не удовлетворяет условию, цикл прекращается.

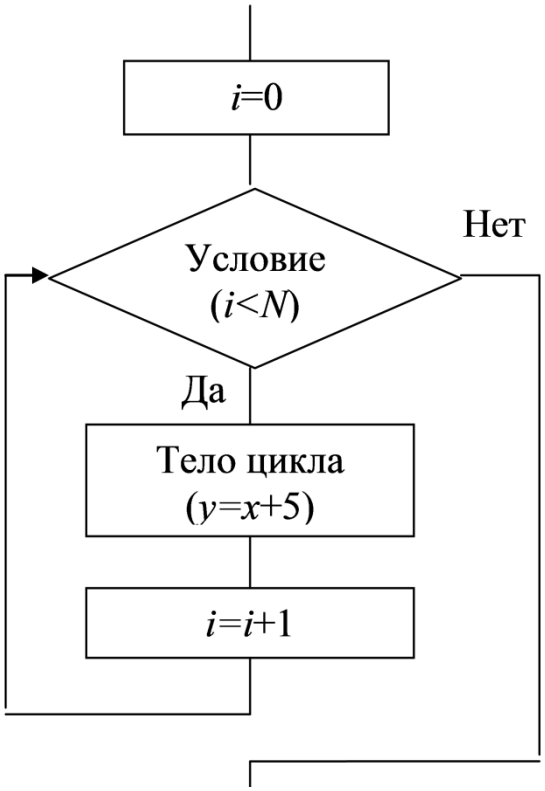


Рис. 1.6. Блок-схема составной операции **Повторение** (цикл)

Начальное значение параметра цикла в языке *C++* принято равным нулю и присваивается перед вхождением в цикл. Параметр цикла принято обозначать символами ***i, j*** и ***k.***

1.4. Виды алгоритмов

Как было отмечено выше, алгоритм – это последовательность определенных действий, выполняемых над исходными данными для решения какой-либо задачи. При этом используются различные виды простых и составных операций (см. раздел 3) или их комбинаций. В зависимости от вида этих операций различают следующие типы алгоритмов:

- линейные;
- разветвленные;
- циклические.

В **линейном алгоритме** действия выполняются последовательно одно за другим. Примером линейного алгоритма является достаточно простая блок-схема решения задачи по определению электрического сопротивления новогодней гирлянды (рис. 1.3).

Разветвленный и **циклический** алгоритмы, их блок-схемы и особенности использования будут нами рассмотрены ниже при изучении соответствующих разделов языка программирования *C++*.

Выводы. Алгоритм решения задачи – это строгая последовательность действий, которые необходимо выполнить над данными, чтобы получить искомый результат (решить поставленную задачу).

Вопросы для самоконтроля

1. Что такое алгоритм?
2. Что обозначает ромб на схеме алгоритма?
3. Какая последовательность решения задачи на ЭВМ?
4. Что обозначает прямоугольник на схеме алгоритма?
5. Как изображается начало алгоритма в схеме?
6. Что обозначает параллелограмм на схеме алгоритма?
7. Какая конструкция используется для изображения разветвления в схемах алгоритмов?
8. Что такое простые операции?
9. Что такое составные операции?
10. Какая конструкция используется для изображения цикла?

ЛЕКЦИЯ 2
ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ *C++*

Цель лекции. Изучить алфавит языка *C++*, типы данных, ключевые слова, идентификаторы, объявление переменных и констант в программе.

Основные вопросы лекции

1. Алфавит языка *C++*.
2. Ключевые слова, идентификаторы, комментарии в *C++*.
3. Типы данных в языке *C++*.
4. Объявление переменных и констант в *C++*.
5. Другие типы данных в *C++*.

2.1. Алфавит языка *C++*

В тексте на любом разговорном языке различают четыре основных группы элементов: символы, слова, словосочетания и предложения. Подобные элементы содержит и любой язык программирования, только слова называют **лексемами** (элементарными конструкциями), словосочетания - **выражениями**, а предложения - **операторами**.

Алфавит языка – это базовый набор символов. Алфавит языка *C++* состоит из латинских букв (прописных и строчных), арабских цифр, специальных и управляющих символов.

1. **Буквы** - *A, B, C, ..., Z, a, b, c, ..., z*.
2. **Цифры** - 0, 1, 2, ..., 9.
3. **Специальные символы** используются для организации процесса вычислений и для передачи компилятору определенных инструкций.

К специальным символам относятся знаки препинания, скобки, знаки пробела, табуляции, перевода строки, возврата каретки, новой страницы и новой строки и т. п.. Эти символы отделяют друг от друга константы и идентификаторы. Последовательность разделительных символов рассматривается компилятором как один символ (последовательность пробелов). Подробно ознакомиться со

специальными символами можно в [1, 2].

4. **Управляющие символы**. В языке *C++* используются управляющие последовательности – комбинации символов, используемые в функциях ввода и вывода информации. Управляющая последовательность строится на основе использования обратной дробной черты (\) (обязательный первый символ) и комбинации латинских букв и цифр (см. таб. 2. 1).

Таблица 2.1

Управляющие последовательности

Управляющая последовательность	Наименование
\b	Возвращение на шаг
\n	Переход на новый ряд
\r	Возвращение каретки
\f	Новая страница
\"	Кавычки
\'	Апостроф
\0	Ноль-символ
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\a	Звуковой сигнал
\ \	Обратная дробная черта
\?	Вопросительный знак

2.2. Идентификаторы. Ключевые слова. Комментарии

Идентификатор – это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются. Первым символом идентификатора может быть буква или знак подчеркивания. Например, ***a***, ***b***, ***x***, ***summa***, ***Summa***, ***Y_2345*** **(**язык *C++* различает прописные и строчные буквы).

Идентификатор создается на этапе объявления переменной, функции, типа и т.п. и не ограничивается по числу символов. После этого его можно использовать в последующих операторах программы. Идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка и начинаться с символа подчеркивания.

Ключевые слова – это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. В таблице 2.2 приведен список ключевых слов *C++* в алфавитном порядке.

Таблица 2.2

Ключевые слова языка *C++*

asm	*do*	*if*	*return*	*typedef*
auto	*double*	*inline*	*short*	*typeid*
bool	*dynamic_cast*	*int*	*signed*	*typename*
break	*else*	*long*	*sizeof*	*union*
case	*enum*	*mutable*	*static*	*unsigned*
catch	*explicit*	*namespace*	*static_cast*	*using*
char	*export*	*new*	*struct*	*virtual*
class	*extern*	*operator*	*switch*	*void*
const	*false*	*private*	*template*	*volatile*
const_cast	*float*	*protected*	*this*	*wchar_t*
continue	*for*	*public*	*throw*	*while*
default	*friend*	*register*	*true*	

Комментарий – это текст, поясняющий значение фрагментов программного кода и предназначенный для пользователя. Комментарии компилятор не воспринимает как часть программы. В *C++* комментарий можно сделать одним из двух способов:

1. Символы “/*” начинают комментарий, заканчивающийся символами “*/”. Вся эта последовательность символов эквивалентна символу пробела.

2. Символы “//“ обозначают комментарий, заканчивающийся в конце данной строки.

2.3. Типы данных в языке *C++*

Основная цель любой программы состоит в обработке данных. Данные, с которыми работают машинные команды, хранятся в оперативной памяти. Компилятору для формирования команд необходимо точно знать, сколько места занимают данные, как они закодированы и какие действия с ними можно выполнять. Каждая константа, переменная, результат вычисления выражения или функции характеризуются **типом данных**, однозначно определяющим допустимые действия над этими данными.

Тип каждой величины, используемой в программе, задается исходя из характеристик соответствующих реальных объектов. Обязательное указание типа позволяет компилятору проверять, правильно ли используется объект в конструкциях языка. От типа величины зависят машинные команды, которые будут использоваться для обработки данных.

Объем памяти, выделяемый для любой константы или переменной, также зависит от ее типа.

Типы данных в языке *C++* делятся на элементарные (базовые) и составные.

Элементарные типы данных являются неделимыми и позволяют описывать целые, вещественные, символьные и логические величины. На основе этих типов программист может конструировать составные типы. К **составным** типам относятся массивы, структуры, объединения, перечисления, ссылки, указатели и классы.

Существует пять базовых типов данных:

– ***bool*** (логический);

– ***char*** (символьный);

– ***int*** (целый);

– ***float*** (действительный);

– ***double*** (действительный с двойной точностью).

Существует четыре ключевых слова, уточняющих внутреннее представление и диапазон значений стандартных типов:

– ***short*** (короткий);
– ***long*** (длинный);
– ***signed*** (знаковый);
– ***unsigned*** (беззнаковый).

Сочетания перечисленных ключевых слов формируют 14 различных арифметических типов. Например, ***char, signed char*** и ***unsigned char*** - это три равноправных различных типа.

Логический тип. Величины логического типа могут принимать только значения ***true*** и ***false*** (истина и ложь), являющиеся ключевыми словами. Величины логического типа могут участвовать в арифметических операциях. При преобразовании к целому типу ***true*** имеет значение 1, ***false*** – 0. Размер логического типа в стандарте не определен и зависит от реализации.

Символьный тип. В стандарте языка *C++* определено три различных символьных типа: ***char***, ***signed char*** и ***unsigned char***. Внутренним представлением символа является его код – целое число. Под величину любого символьного типа отводится одна единица памяти. Размер байта должен быть достаточен, чтобы вместить код любого символа из набора символов реализации для данного компьютера.

Величины символьных типов применяются также для хранения целых чисел, не превышающих границы указанных диапазонов, и могут участвовать в арифметических операциях, поэтому их также относят к целым типам.

Целый тип. В языке *C++* определено 8 типов для хранения целочисленных величин: четыре знаковых (***signed char***, ***short int***, ***int***, ***long int***) и четыре беззнаковых (***unsigned char***, ***unsigned short int***, ***unsigned int***, ***unsigned long int***). По умолчанию все целочисленные типы считаются знаковыми, поэтому спецификатор ***signed*** можно не указывать. Ключевое слово ***unsigned*** позволяет представлять неотрицательные целые числа. Диапазоны целых чисел приведены в таблице 2.4.

Действительные числа записываются в обычной десятичной либо экспоненциальной форме. Например, 527.3, 7.15, 0.43.

Число 527.3 можно записать в экспоненциальной форме как совокупность числа, называемого мантиссою, и порядка – степени при десяти. Например, $527.3*10^0=52.73*10^1=5.273*10^2=5273*10^{-1}$.

Таблица 2.3

Целый тип данных в *C++*

Тип	Размер	Диапазон числа без знака	Диапазон числа со знаком
short int	2 байта	0...65535=2^{16}-1	-2^{15}...2^{15}-1=-327678...32767
int	4 байта	0...4294967295 = 2^{32}-1	-2^{31}...2^{31}-1
long int	4 байта	0...4294967295 = 2^{32}-1	-2^{31}...2^{31}-1

В языке *C++* вместо цифры 10 записывается буква *E*:

$$527.3E0 = 52.73E1 = 5.273E2 = 5273E\text{-}1.$$

Действительный тип – это множество рациональных и иррациональных чисел с десятичной точкой. В языке *C++* используются следующие действительные типы: ***float*** – числа з плавающей точкой одинарной точности; ***double*** – числа з плаваю-щей точкой двойной точности; ***long double*** – для вычислений особой точности. Диапазон чисел действительного типа приведен в таблице 2.4.

Таблица 2.4

Действительный тип данных в C++

Тип	Размер	Диапазон числа
float	4 байта	-2^{31}... 2^{31}-1
double	8 байт	-2^{63}... 2^{63}-1
long double	8 байт	-2^{63}... 2^{63}-1

2.4. Объявление переменных и констант в языке *C++*

Переменная – это фрагмент памяти, в котором хранится элемент данных и к которому можно обращаться по некоторому имени. **Имена переменных** могут включать буквы латинского алфавита **A – z** (в верхнем или нижнем регистре), цифры от 0 до 9 и знак подчеркивания. Имена переменных должны начинаться либо с буквы, либо со знака подчеркивания. В *C++* принято назначать имена переменных с прописных букв, а классов – с заглавных. Компилятор *C++* различает прописные и строчные буквы, например, ***Argument*** и ***argument*** означают разные переменные.

Перед выполнением любой программы необходимо “предупредить” компьютер относительно типа всех переменных, которые предполагается использовать. Этот процесс называется **объявлением** переменной и описывается следующим образом:

ТипПеременной ИмяПеременной;

Например, операция ***int x;*** объявляет целую переменную с именем ***x***.

Имена переменных не должны совпадать с ключевыми словами.

Объявляя переменную, можно сразу **присвоить** ей начальное значение. Например,

int x=0; *int x*(0); *float Y*=42.2; *float Y*(42.2).

Константа – это переменная, не меняющая свое значение в программе. В C++ объявление константы выглядит следующим образом:

const Тип ИмяКонстанты = Значение.

Например, объявить постоянную ***g*** можно следующим образом:

const float g =9.81456783.

2.5. Другие типы данных

Литералы – это данные, находящиеся непосредственно в тексте программы. Литералы могут быть числовые, символьные и строковыми.

Числовые литералы могут быть целыми и действительными. Целый литерал – это целое число, записанное в тексте программы. Число может быть записано в десятичной, восьмеричной или шестнадцатеричной системах. Для того чтобы отличить, в какой системе счисления записано число, перед ним в восьмеричной системе записывают ноль, а в шестнадцатеричной – ноль и “***x***”.

Например:

3729, **134267** – числа в десятичной системе счисления;

054, **0643** – числа в восьмеричной системе счисления;

0*x*82, **0*x*769** – числа в шестнадцатеричной системе счисления.

Символьный литерал (символьная константа) используется для представления одного символа. Это любой символ, заключенный в одинарные кавычки: **‘*Q*’, ‘&’, ‘7’, ‘Ш’**.

Строковый литерал – это любая последовательность символов, заключенная в парные кавычки: “***GAMMA***”, “**Схема**”.

Массивы – структурированные наборы данных.

Указатели – содержат адреса ячеек памяти, в которых содержатся данные.

Строки – особая форма массивов, содержащих символьные данные.

Записи – структуры, связывающие элементы разных типов в один объект.

Файлы – структуры, представленные хранимыми данными.

Выводы. В языке С++ различают четыре основных группы элементов: символы, слова, словосочетания и предложения.

Каждая константа, переменная, результат вычисления выражения или функции характеризуются типом данных, однозначно определяющим допустимые действия над этими данными.

Вопросы для самоконтроля

1. Как правильно записать действительное число на языке *C*++?
2. В результате решения задачи на ПК на екране появилось число 7.45600000*E*+01. Какое это число?
3. Как производится запись символьной константы в языке *C*++?
4. Как записывается идентификатор в *C*++?
5. Как записывается константа в *C*++?
6. Перечислите базовые типы данных в *C*++.
7. Дайте определение константы.
8. Дайте определение переменной.
9. Для чего необходимо объявлять переменные перед действиями с ними?
10. Каков формат объявления переменной?

ЛЕКЦИЯ 3
ОПЕРАЦИИ И ВЫРАЖЕНИЯ В *VISUAL C++* 2010.

Цель лекции. Изучить операции и выражения языка *C++*, порядок и особенности их применения.

Основные вопросы лекции

1. Операции в *C++*.
2. Арифметические операции в *C++*.
3. Операция присваивания в *C++*.
4. Выражения в *C++*.
5. Стандартные функции в *C++*.
6. Операция приведения типа.
7. Операции инкремента и декремента.

3.1. Операции в *C++*

Операция в языке *C++* – это проведение строго определенных знаком операции действий над субъектами операции – **операндами**. Как и в математике, каждая операция в языке *C++* характеризуется приоритетом. **Приоритет** операции – это порядок, в котором проводится вычисление значения выражения с несколькими операциями (см. таб. 3.1).

Каждая операция имеет собственный порядок выполнения – слева направо или справа налево. Знак операции – это символ или сочетание символов, сообщающих компилятору о необходимости проведения определенных арифметических, логических или других действий.

Для каждой операции, определенной в *C++*, определено количество операндов:

– один операнд – **унарная** операция; например, *d*3;*

– два операнда – **бинарная** операция; например, операция сложения *a+d*;

– три операнда – операция **условие**.

Полный список операций языка *C++* приведен в таблице 3.1.

Таблица 3.1

Операции языка программирования *C++*

Операция	Название операции	Приоритет	Порядок выполнения
1	2	3	4
Первичные и постфиксные операции			
[ ]	Індексация массива	16	Слева направо
( )	Вызов функции	16	Слева направо
.	Элемент структуры	16	Слева направо
–>	Элемент указателя	16	Слева направо
+ +	Постфиксный инкремент	15	Слева направо
– –	Постфиксный декремент	15	Слева направо
Одноместные (унарные) операции			
+ +	Префиксный инкремент	14	Справа налево
– –	Префиксный декремент	14	Справа налево
sizeof	Размер в байтах	14	Справа налево
(тип)	Приведение типа	14	Справа налево
~	Поразрядное Нет		
!	Логическое Нет	14	Справа налево
–	Унарный минус	14	Справа налево
&	Взятие адреса	14	Справа налево
*	Разыменование указателя	14	Справа налево
Мультипликативные операции			
*	Умножение	13	Слева направо
/	Деление	13	Слева направо
%	Взятие по модулю	13	Слева направо
Аддитивные операции			
+	Сложение	12	Слева направо
–	Вычитание	12	Слева направо
Операции поразрядного сдвига			
<<	Сдвиг влево	11	Слева направо
>>	Сдвиг вправо	11	Слева направо

Продолжение табл. 3.1

1	2	3	4
Операции отношения (сравнения)			
<	Меньше	10	Слева направо
<=	Меньше или равно	10	Слева направо
>	Больше	10	Слева направо
>=	Больше или равно	10	Слева направо
=	Равно	9	Слева направо
!=	Не равно	9	Слева направо
Поразрядные операции			
&	Поразрядное *AND*	8	Слева направо
^	Поразрядное *XOR*	7	Слева направо
!	Поразрядное *OR*	6	Слева направо
Логические операции			
&&	Логическое *AND*	5	Слева направо
!!	Логическое *OR*	4	Слева направо
Условная операция			
?:	Условная операция	3	Справа налево
Операции присваивания			
=	Присваивание	2	Справа налево
*=	Присваивание умножения	2	Справа налево
/=	Присваивание деления	2	Справа налево
%=	Присваивание модуля	2	Справа налево
+=	Присваивание суммы	2	Справа налево
=	Присваивание разности	2	Справа налево
<<=	Присваивание левого сдвига	2	Справа налево
>>=	Присваивание правого сдвига	2	Справа налево
&=	Присваивание *AND*	2	Справа налево
^=	Присваивание *XOR*	2	Справа налево
!=	Присваивание *OR*	2	Справа налево
,	Запятая	1	Справа налево

3.2. Арифметические операции

В языке *C++* используется пять основных математических операций: сложение, вычитание, умножение, деление и деление по модулю (см. таб. 3.2). Эти операции применяются к данным целого и действительного типов. Операции сложения, вычитания, умножения и деления выполняются слева направо. Если операнды имеют один тип, результат арифметической операции будет иметь тот же тип. Когда эти операции применяются для действительных чисел, это обычные арифметические операции. Но если операция деления применяется к целым операндам, то результат операции будет целым, а остаток от деления отбрасывается. Например, 9/4 будет равно 2, а выражение 2/5 будет равно нулю.

Таблица 3.2.

Арифметические операции в языке *C++*

Операция	Знак в языке *C++*	Порядок выполнения
Сложение	+	Слева направо
Вычитание	-	Слева направо
Умножение	*	Слева направо
Деление	/	Слева направо
Деление по модулю	%	Слева направо

Операция **деление по модулю** используется только с целыми операндами и дает остаток от деления первого операнда на второй. Так, результат выполнения операции 13%4 будет равен 3, а результатом операции 7%7 будет равен 0.

3.3. Операция присваивания

Присваивание – единственная операция, изменяющая значение одного из своих операндов (не считая операций инкремента и декремента, которые будут рассмотрены ниже).

Операция присваивания обозначается знаком “=” и записывается следующим образом:

$$a = b,$$

где *a* –переменная любого базового типа данных языка *C++*; *b* – выражение.

При этом вычисляется значения выражения *b*, стоящего в правой части, а затем вычисленное значение *b* присваивается переменной *a*.

Особенностью операции присваивания в языке *C++* есть возможность записать следующий оператор:

$$a = b = c = x*y.$$

В этом случае присваивание выполняется справа налево: вначале вычисляется значение *x*y*, а потом это значение присваивается *c*, потом *b*, затем *a*.

3.4. Выражения в *C++*

В языке *C++* следующим уровнем представления данных после переменных и констант являются выражения. **Выражение** – это некоторая допустимая комбинация переменных, констант, функций и знаков операций для вычислений в программах. Выражения в языке *C++* записываются в строчку. Например, формула

$$d = \frac{a + b(c + e)}{c(a + b) + t}$$

в языке C++ запишется следующим образом:

$$d = (a + b*(c + e))/(c*(a + b) + t)).$$

В приведенном выше выражении знак “=” обозначает операцию **присваивания**, которая выполняется следующим образом. Вычисляется значение выражения в правой части и присваивается переменной *d*. Для соблюдения необходимой по условию задачи очередности операций используются круглые скобки.

3.5. Стандартные математические функции в *C++*

В языке *C++* в выражения можно вставлять стандартные математические функции, которые вызываются из библиотеки **<*math.h*>**. Перечень математических функций, которые чаще всего встречаются в вычислениях, приведены в таблице 3.3.

Таблица 3.3

Основные стандартные математические функции

Название функции	Что вычисляет	Тип данных функции и аргумента
abs(x)	Абсолютное значение аргумента $\|x\|$	*int abs(int x)*
exp(x)	Экспонента e^x	*double exp(double x)*
log(x)	Натуральный логарифм $\ln x$	*double log(double x)*
log10(x)	Десятичный логарифм $\lg x$	*double log10(double x)*
pow(x,y)	Возведение в степень x^y	*double pow(double x, double y)*
sqrt(x)	Квадратный корень $\sqrt{x}$.	*double sqrt(double x)*
fmod(x,y)	Остаток от деления *x/y*	*double fmod(double x, double y)*
sin(x)	Синус (угол в радианах)	*double sin(double x)*
asin(x)	Арксинус (угол в радианах от –1 до +1)	*double asin(double x)*
cos(x)	Косинус (угол в радианах)	*double cos(double x)*
acos(x)	Арккосинус (угол от –1 до +1)	*double acos(double x)*
tan(x)	Тангенс (угол в радианах)	*double tan(double x)*
atan(x)	Арктангенс (угол в радианах)	*double atan(double x)*

При обращении к стандартным функциям необходимо тщательно учитывать тип данных ее аргументов.

Например, выражение $y = \sin(x)\dfrac{e^x + z^5 - 4.5 \cdot 10^2 \sqrt{x}}{tg(a)(z^x + b)}$ на языке *C++* будет иметь вид:

y=sin(x)(exp(x)+pow(z,5)-4.5*10*10*sqrt(x))/(tan(x)*(pow(z,x)+b)).*

3.6. Операция приведения типа.

В арифметическом выражении могут присутствовать операнды разных типов - как целые, так и действительные. И те, и другие могут иметь разную длину (***chort, long***). В то же время оба операнда любой арифметической операции должны иметь один и тот же тип. В языке *C++* при вычислении значения такого выражения происходит автоматическое приведение типов.

Операция приведения типов состоит в следующем. На каждом шаге вычисления значения выражения выполняется одна операция над одной парой операндов. Если типы операндов не совпадают, то операнд меньшего ранга приводится к типу более высокого ранга. Обычно короткие типы данных приводятся к более длинным, что обеспечивает сохранение значимости и точности:

char, short → int → unsigned int → long →
unsigned long → float → double → long double.

Правила, используемые для автоматического приведения типов данных при вычислении значения арифметического выражения, следующие:

1. Все переменные типа ***char***, ***short int*** преобразуются в ***int.***

2. В любой паре операнды приводятся к одному типу. Например, если один из операндов ***double***, то и другой преобразуется в ***double.***

3. В операторе присваивания конечный результат приводится к типу переменной в левой части оператора. При этом ранг типа может как повышаться, так и понижаться.

3.7. Операции инкремента и декремента

При работе с циклическими программами широко используется прием приращения или убывания значения параметра цикла, например *k=k*+1. В *Visual C++* 2010 предусмотрена сокращенная запись подобных операций, называемых инкремента и декремента.

Унарная операция **инкремента** $i = i + 1$ увеличивает свой операнд (обязательно переменную) на единицу и сокращенно записывается как ***i*++.**

Унарная операция **декремента** $i = i - 1$ уменьшает свой операнд на единицу и сокращенно записывается ***i*- -.**

Эти операции могут использоваться и в выражениях:

$$sum = sum + x * i++.$$

Инкремент и декремент реализуются в двух формах: префиксной, например, ***++i*** и постфиксной, например, ***i*++.** Если знак операции находится слева от параметра цикла, то вначале выполняется инкремент или декремент, а потом вычисляется выражение. В случае, когда знак операции находится справа, то вначале вычисляется выражение, а затем изменяется значение параметра цикла в ходе выполнения операции инкремента или декремента.

Выводы. Операция в языке *C*++ – это проведение строго определенных знаком операции действий над субъектами операции – операндами.

В языке *C*++ следующим уровнем представления данных после переменных и констант являются выражения. **Выражение** – это некоторая допустимая комбинация переменных, констант, функций и знаков операций для вычислений в программах.

Вопросы для самоконтроля

1. Дайте определение операции в языке *C*++.
2. Что такое операнд?
3. Дайте определение приоритета операции и приведите примеры.
4. В чем уникальность операции присваивания?
5. Как записываются математические действия с переменными, константами и функциями в языке *C*++?
6. Каковы правила обращений к математическим функциям в языке *C*++?
7. Что такое выражение?
8. Охарактеризуйте операции инкремента и декремента.

ЛЕКЦИЯ 4
СТРУКТУРА ПРОГРАММ В СРЕДЕ *VISUAL C++* 2010. ОПЕРАТОРЫ ВВОДА И ВЫВОДА

Цель лекции. Изучить операторы ввода и вывода в языкеl *C++*. Изучить структуру программ в среде *Visual C++* 2010

Основные вопросы лекции

1. Основные понятия среды разработки *Visua*l *C++* 2010.
2. Структура программ в среде *Visual C++* 2010.
3. Операторы ввода и вывода в консольном приложении *Win* 32 среды *Visual C++* 2010.
4. Интерфейс среды разработки *Visua*l *C++* 2010.
5. Редактирование и отладка программ в *Visual C++* 2010

4.1. Основные понятия и интерфейс среды разработки *Visual C++* 2010

Данное пособие рассматривает разработку в *Visual C++* 2010 т. н. консольных программ. Эти программы называются в среде *Visual C++* 2010 консольными, т. к. разработчик взаимодействует с ними с помощью клавиатуры и экрана. А. Хортон считает [1], что написание консольных приложений – это лучший способ начать углубленное изучение *C++*, в то же время – это самодостаточный инструмент для решения различных инженерных и математических задач.

Ограниченный объем данного издания обуславливает несколько схематичное раскрытие заявленной темы, чего, однако, достаточно для написания консольных приложений. Более глубокие сведения желающие могут получить в [1].

Рассмотрим необходимые для дальнейшей работы понятия.

В состав *Visual C++* 2010 входит **интегрированная среда разработки** – полностью самодостаточная среда для создания, компиляции, компоновки и проверки программ на языке *C++*. Соответственно список составляющих среды разработки включает в себя редактор, компилятор, компоновщик и библиотеки.

Редактор – это интерактивная среда для создания и редактирования исходного кода *C++*. Редактор автоматически распознает ключевые конструкции языка *C++* и окрашивает их в соответствии с их назначением, что помагает читать код и видеть ошибки при вводе. Редактор создает файл с расширением ***.cpp***.

Компилятор преобразует исходный код в объектный, обнаруживает широкий диапазон ошибок и помещает выходной код в объектные файлы (расширение ***.obj***).

Компоновщик собирает вместе различные модули, созданные компилятором из файлов исходного кода, добавляет необходимые модули из библиотек и сшивает все это в одно исполняемое целое.

Библиотеки – это коллекция предварительно написанных процедур, программ и функций, включаемых в программу для выполнения стандартных или часто встречающихся операций.

4.2. Структура программ в среде *Visual C++* 2010

Программа на языке *C++* состоит из следующих условных частей (см. рис. 4.1):

1. Директивы (указания) компилятору.
2. Объявления и определения переменных, констант, функций.
3. Одна или несколько функций.

Директивы компилятору Директива 1 Директива 2 ..... Директива *N*	1
Объявление и определение (инициализация) переменных, констант, функций	2
Функция 1 Функция 2 …..... Функция *M*	3

Рис. 4.1. Структура программы в среде *Visual C++* 2010

Директивы компилятору предназначены для обработки исходного текста программы перед компиляцией. Любая директива должна начинаться с символа #. На каждой строке может располагаться только одна директива. Например, по директиве ***#include <conio.h>*** в текст программы будет вставлено содержимое заголовочного файла с именем ***conio.h*** (для задержки на экране дисплея окна *DOS*). Заголовочные файлы содержат различную информацию, необходимую для успешной компиляции и работы программы.

Функции описывают совокупность действий, которые должна выполнить программа. Если функций несколько, то среди них выделятся **главная** функция, которая имеет имя ***tmain***. С нее начинается выполнение программы. Если программа содержит единственную функцию, то она обязательно должна иметь имя ***tmain***(…).

В языке *C++* функции являются строительными блоками программы, спроектированными для решения конкретных задач. Тело функции (программа, описывающая ее работу) всегда заключается в фигурные скобки.

Программа может содержать произвольное число директив, указаний, объявлений, определений и функций. Они могут располагаться в любом месте программы, но действовать в программе они начинают только с того места, где расположены.

В исходный текст программы можно вставлять текст из другого файла с помощью директивы **# *include* <имя файла>.**

Если имя файла размещено в угловых скобках, то поиск нужного файла производится только в стандартных каталогах. Если имя файла размещено в кавычках, то поиск нужного файла производится в текущем каталоге. Например, директива ***#include <iostream>*** сообщает компоновщику о необходимости включить в программу файл *iostream,* содержащий функции ввода и вывода данных в консольном приложении *Win* 32 среды *Visual C++* 2010.

После директив и указаний препроцессору в исследуемой программе располагается определение функции ***tmain*(…)**. Любая программа на C++ содержит такую функцию, которая является главной (см. выше). Внутри функции, в свою очередь, должны быть объявлены используемые в ней переменные, константы и

функции. Стандартные математические функции (sin, cos и др.) объявлять не надо.

4.3. Операторы ввода и вывода в консольном приложении *Win* 32 среды *Visual C++* 2010

В приложении *Win*32 среды программирования *Visual C++* 2010 консольный **ввод** данных (с использованием клавиатуры) производится при помощи оператора ***cin***. В C++ этот оператор называется также потоком ввода. Например, для ввода значений трех переменных надо записать:

cin>>a>>b>>c,

где >> - символ операции извлечения данных из потока ввода; *a, b* и *c* – переменные, значения которых будут вводится. Вводимые значения должны разделяться пробелами, а ввод завершается нажатием клавиши **<*Enter*>.**

Потоковый ввод и его операции автоматически распознают переменные и данные любого типа.

Консольный **вывод** данных (на экран дисплея) производится при помощи оператора ***cout***. В C++ этот оператор называется также потоком вывода. Например, для вывода значений трех переменных надо записать:

cout<<a<<b<<c,

где << - символ операции вставки данных в поток вывода; *a, b* и *c* – переменные, значения которых будут выводиться на экран.

Потоковый вывод и его операции автоматически распознают переменные и данные любого типа. Вывод в *Win*32 производится в командную строку окна *DOS*. Помимо данных можно выводить и текстовую строку, заключив ее в кавычки:

cout<<"Summa a+b+c = "<<d.

Стандартные функции ввода и вывода находятся в библиотечном файле ***iostream***. Чтобы связать разрабатываемую

программу со стандартной библиотекой ввода-вывода, необходимо в начале программы указать оператор подключения этой библиотеки ***#include <iostream>***.

Точка с запятой после операторов ***include*** не ставится.

Оператор ***cout*** часто используется с различными опциями (управляющими последовательностями) для расширения его функций (см. таб. 2.1).

В операторе *cout* допускается производить арифметические действия. Например, при выполнении оператора *cout*<<2 + 3 + 4 на экран дисплея будет выведена цифра 9.

4.4. Интерфейс программы *Visual Studio* 2010

Интерфейс *Visual Studio* 2010 нагляден и удобен для создания, редактирования, просмотра и отладки компонентов разрабатываемых программ и приложений – ресурсов, классов, файлов и др. Основной экран разделен на части с различными функциями, размеры которых можно изменять по своему усмотрению. Для облегчения решения стандартных задач есть контекстные меню, доступ к которым осуществляется с помощью нажатия правой кнопки мыши на различных компонентах изображения на экране.

С помощью *Visual C++* 2010 можно работать с разрабатываемым приложением как с проектом.

Проект - это регламентированный (определенный протоколами *Visual Studio* 2010) набор файлов: заглавий, текстов программ, ресурсов, установок, конфигураций и др. Интерфейс *Visual Studio* 2010 дает возможность работать со всеми компонентами проекта одновременно, поэтому экран разделен на несколько зон (окон). Каждая разрабатываемая программа (приложение), даже самая простая, является проектом.

Набор папок и файлов, создаваемый средой *Visual C++* 2010 при разработке проектов, называется **решением.**

При запуске программы *Visual Studio* 2010 открывается окно **Начальная страница – *Microsoft Visual Studio* (Администратор)**, состоящее из трех частей:

1. **Обозреватель решений** – окно в левой части экрана, в котором в дальнейшем будут представлены в виде дерева папки и файлы текущего проекта. В случае необходимости на месте данного окна можно открыть **Окно классов** данного проекта, его **Диспетчер свойств** или **Командный обозреватель;**

2. Окно **Начальная страница** в правой части экрана, в котором предложены на выбор возможные действия пользователя, в частности, **Создать проект** или **Открыть проект**, а также для открытия предложен список последних проектов;

3. Окно **Вывод** в нижней части экрана, в котором при построении решений проектов выводятся сообщения о предупреждениях (***warning***) и ошибках (***error***). На месте этого окна можно открыть **Окно определения кода** или окно **Результаты поиска.**

При создании консольных приложений на языке *C++* работа осуществляется в окне <**Имя проекта> – *Microsoft Visual Studio* (Администратор)**, состоящем из трех частей:

1. **Обозреватель решений** (описание см. выше);

2. Окно для ввода программного кода и для работы с программой в правой части экрана;

3. Окно **Вывод** (описание см. выше).

4.5. Редактирование и отладка программ в *Visual C++* 2010

Код (текст) программы вводится в окне редактора с использованием клавиатуры и основных приемов работы с текстом в операционной системе (ОС) *Windows*. При отображении текста программы используется синтаксическое раскрашивание. По умолчанию текст программы черного цвета с комментариями зеленого цвета и ключевыми и служебными словами синего цвета.

После того, как набран программный код, следует приступить к отладке программе. Для этого следует открыть меню **Построение** и выбрать пункт **Построить решение**. После паузы все сообщения о предупреждениях и ошибках отображаются в окне **Вывод.** Ошибки следует исправлять обязательно, т. к. программа с ошибками выполняться не будет. С предупреждениями программа будет выполняться, однако их необходимо проанализировать и,

если возможно, исправить или учесть. Описание ошибки находится в строке сообщения об ошибке (для получения подробного описания ошибки следует нажать <***F*1>**). Чтобы исправить ошибку необходимо дважды щелкнуть в окне отладчика на сообщении о данной ошибке, после чего в окне редактора появится указатель на строке с ошибкой. После исправления всех ошибок необходимо открыть пункт меню **Отладка**. Программа запускается при выборе пункта меню **Начать отладку** или при нажатии клавиши <***F*5>.**

Применим на практике вышеизложенные сведения. В качестве примера разработаем и запустим на выполнение простую программу для вычисления переменной ***a*** по формуле

$$a = b\frac{c + 2d - cd}{d(5c + 4b)}$$

с использованием операторов консольных ввода и вывода данных. Блок-схема алгоритма решения данной задачи имеет линейный вид и приведена на рис. 4.2.

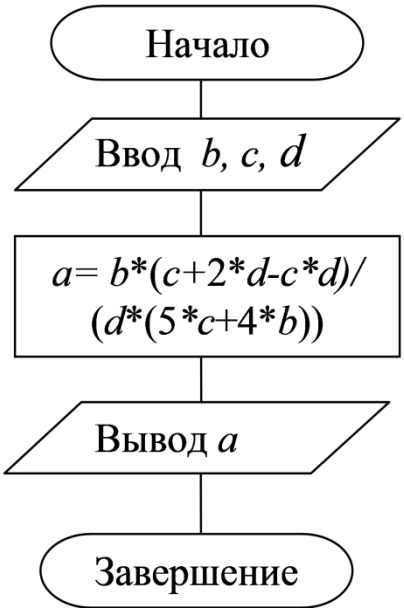


Рис. 4.2. Блок-схема алгоритма вычисления переменной *a*

Для понимания работы рассматриваемых в конспекте программ служат комментарии – текст на русском языке в программном коде, расположенный после символов “//”. Причем,

после нескольких повторений в программах типовых операторов, комментарии к ним будут убираться, чтобы не перегружать программный код.

Программный код (текст программы) с комментариями приведен ниже.

```
1. #include "stdafx.h"
2. #include <conio.h>                    //обеспечивает задержку окна
                                          //DOS на экране дисплея
3. #include <iostream>                   //Директива о подключении
                                          //файла заголовка iostream
4. using namespace std;                  //Подключает все имена из
                                          //пространства имен  std
5. int _tmain(int argc, _TCHAR* argv[ ])  //Объявление главной
функции
6. {                                      //Начало главной функции
7. double a;                              //Объявление переменной  a
8. double b, c=2, d=3;                    //Объявление переменных  b,
c,                                        //d и их инициализация
9. cout<<"vvedite b"<<endl;               //Вывод на экран приглашения
                                          //vvedite b; endl –
перемещение
                                          //курсора на новую строку
10. cin>>b;                               //Ввод значения переменной  b
11. a=b*(c+2*d-c*d)/(d*(5*c+4*b));        //Вычисление переменной  a
                                          //по заданной формуле
12. cout<<"a = "<<a;                      //Вывод на экран значения a
                                          //(информация в кавычках вы-
                                          //водится без изменений)
13. getch();                              //Функция  задержки окна
DOS                                       //на экране
14. return 0;
15. }
```

Директивы препроцессора предназначены для обработки исходного текста программы перед компиляцией. Любая директива должна начинаться с символа #. На каждой строке может располагаться только одна директива. Например, по директиве ***#include <conio.h>*** в текст исследуемой нами программы будет вставлено содержимое так называемого заголовочного файла – в данном примере с именем ***conio.h*** (для задержки на экране дисплея

окна *DOS*). Заголовочные файлы содержат различную информацию, необходимую для успешной компиляции и работы программы. В данной программе это строки 1, 2 и 3.

Программа может содержать произвольное число директив, указаний, объявлений и определений. Они могут располагаться в любом месте программы, но действовать в программе они начинают только с того места, где расположены.

Директива ***#include <iostream>*** сообщает препроцессору о необходимости включить в программу файл ***iostream***, содержащий функции ввода и вывода данных в консольном приложении *Win* 32 среды *Visual C++* 2010.

После директив и указаний препроцессору в исследуемой программе всегда располагается тело **главной функции *tmain*.** Внутри функции, в свою очередь, должны быть объявлены используемые в ней переменные, константы и функции. Стандартные математические функции (sin, cos и др.) объявлять не надо.

Тело функции (программа, описывающая ее работу) всегда заключается в фигурные скобки. В программе это строки 6 и 15.

Подробнее о правилах и особенностях создания функций будет рассказано в последующих лекциях. Здесь же укажем, что в анализируемой программе в строках 7 и 8 объявлены используемые переменные, в строках 9, 10 и 12 организован ввод и вывод переменных, в строке 11 записан оператор для вычисления переменной ***a***. В строку 13 помещена функция ***getch()***, вызываемая из заголовочного файла ***conio.h*** для задержки на экране дисплея окна *DOS*, необходимого для ввода и вывода информации при работе программы.

Оператор ***return*** всегда должен присутствовать в конце программы, описывающей работу функции ***tmain***. Подробно работа этого оператора будет рассмотрена при изучении функций в языке *C++* в следующих лекциях.

Строки 1, 5, 6, 14 и 15 будут автоматически выставляться редактором *Visual Studio* 2010 при создании новых проектов. В дальнейшем оператор ***int _tmain(int argc, _TCHAR* argv[ ])*** будем записывать в виде ***int _tmain(…)*** для компактного размещения текста программ.

Результат выполнения анализируемой программы имеет следующий вид:

vvedite b
10
a = 0.133333

Выводы. *Visual C++* 2010 – полностью самодостаточная среда для создания, компиляции, компоновки и проверки программ на языке *C++*. С помощью *Visual C++* 2010 можно работать с разрабатываемым приложением как с проектом.

Проект - это регламентированный (определенный протоколами *Visual Studio* 2010) набор файлов: заглавий, текстов программ, ресурсов, установок, конфигураций и др. Интерфейс *Visual Studio* 2010 дает возможность работать со всеми компонентами проекта одновременно.

Вопросы для самоконтроля

1. Чем отличаются программы с линейной структурой?
2. Как вывести на экран значение переменной?
3. Как в одном операторе объявить тип переменной и задать ее значение?
4. Перечислите опции оператора *cout* и раскройте их действие.
5. Какие функции выполняет оператор *include*?
6. Какие функции выполняет файл заголовка *iostream*?
7. Для чего служат комментарии в программном коде?
8. Что такое проект в *Visual Studio* 2010?
9. Какие основные части программы в *Visual Studio* 2010?

ЛЕКЦИЯ 5.
РАЗВЕТВЛЯЮЩИЕСЯ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ В *VISUAL C++* 2010. УСЛОВНЫЕ ОПЕРАТОРЫ

Цель лекции: изучение вопросов разработки циклических программ с использованием операторов условного и безусловного переходов, особенностей их конструкции и исполнения.

Основные вопросы лекции

1. Условный оператор *if…else*.
2. Оператор выбора *switch*.
3. Безусловный оператор.

5.1. Условный оператор *if…else*

При решении практически любой задачи, в том числе и при помощи компьютера, приходится принимать те или иные решения. Поскольку вся информация в компьютере представлена в числовом виде, то сравнение числовых значений – это сущность механизма принятия решений в вычислительных процессах на языке C++.

Существует шесть базовых операторов для сравнения значений двух переменных:

1. < (меньше);
2. <= (меньше или равно);
3. > (больше);
4. >= (больше или равно);
5. == (равно);
6. != (не равно).

Любой алгоритм может быть записан на языке программирования *C++*, использующем только три управляющих структуры: последовательное выполнение, ветвление и повторение. При последовательном выполнении операции выполняются в порядке их расположения в программе, с возможным отклонением

для вызова внешнего фрагмента (функции), но с обязательным возвратом в точку вызова.

Ветвление в простейшем случае описывается в языке *C++* с помощью условного оператора.

На рис. 5.1 приведен пример структуры ветвления. Операционный блок **Условие** состоит из выражения, содержащего логическое отношение, т. е. условие. Если условие выполняется, то реализуется **Действие 1** алгоритма, а в случае невыполнения - **Действие 2**.

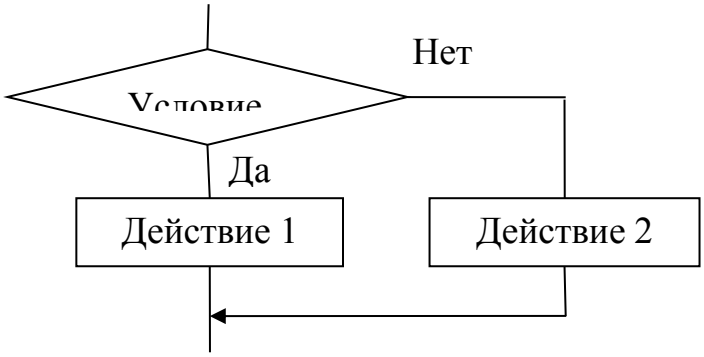


Рис.5.1. Структура ветвления

Структура ветвления описывается в языке *C++* с помощью **условного оператора**:

***if* (выражение)**
оператор_1;
else
оператор_2;

где часть ***else* оператор_2;** может отсутствовать.

Вначале вычисляется выражение в скобках. Если оно истинно, то выполняется **оператор_1**. Если выражение ложно, то **оператор_1** пропускается и выполняется **оператор_2**. Вместо единичных операторов могут использоваться группы из нескольких операторов (сложные операторы). При этом они заключаются в фигурные скобки - **{ }**.

Выражение в скобках представляет собой условие, заданное с помощью операций отношений и логических операций. В программировании постоянно приходится сравнивать числовые значения переменных, т. к. на этом построен процесс принятия

решений при работе над различными проектами.

Сложные операторы. К сложным операторам относят собственно сложные операторы и блоки. В обоих случаях это последовательность операторов, помещенная в фигурные скобки.

Блок отличается от сложного оператора наличием **объявлений переменных** в теле блока. Так, последовательность

```
{
a=b+c;
d=a+b;
}
```

является сложным оператором, а последовательность операторов

```
{
int a,b,c,d;
a=b+c;
d=a+b;
}
```

является блоком оператором.

Для лучшего понимания действия операторов в языке *C++* следует помнить перевод следующих английских слов:

if – если;
then – тогда;
else – иначе;
case – случай;
switch – переключатель;
break – прерывать;
default – не выполнять.

Используя полученные сведения, разработаем программу для определения переменной *a* по следующему условию:

$$a = \begin{cases} b + c + d, & \text{если} \quad b > 10; \\ b - c - d, & \text{если} \quad b \le 10. \end{cases}$$

Рассмотрим программу, производящую арифметические

действия с тремя переменными *b, c* и *d* с использованием условного оператора ***if…else*** и операторов консольных ввода и вывода данных.

Блок-схема алгоритма решения задания 1 представлена на рис.5.2.

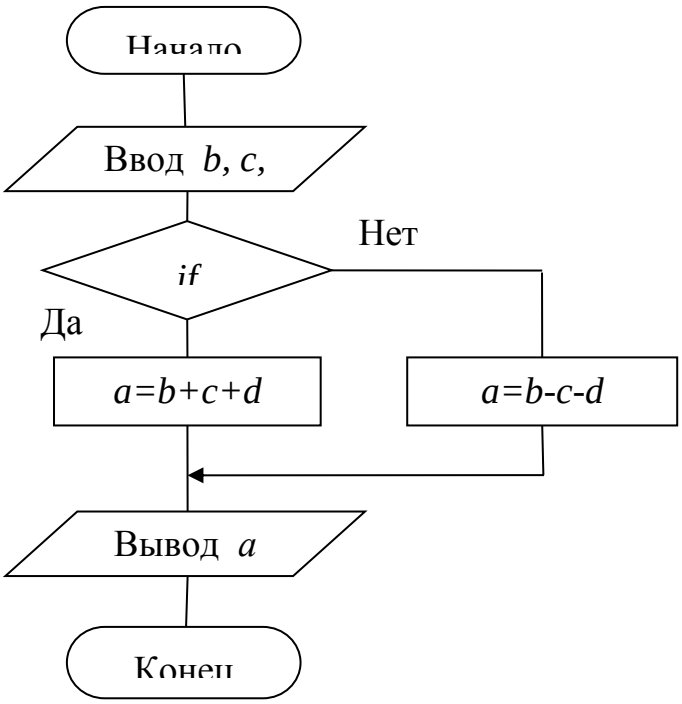


Рис. 5.2. Блок-схема алгоритма определения переменной **a** (использование простых операторов)

Ниже приведен код (текст) программы с комментариями.

```
#include "stdafx.h"
#include <conio.h>                  //Файл conio.h обеспечивает
                                    //Задержку  окна DOS на экране
#include <iostream>                 //Директива include подключает файл
                                    //ввода–вывода  iostream
using namespace std;                //Подключает все имена из пространства
                                    //имен  std
int _tmain(…)                       //Объявление главной функции _tmain
{                                   //Начало главной функции
int a, b;                           //Объявление переменных целого типа
int c=2, d=3;                       //Объявление переменных целого типа  и
                                    //их инициализация
cout<<"Vvvedite b"<<endl;           //Вывод на экран надписи-приглашения
                                    //Vvedite b
```

```
cin>>b;                          //Ввод значения переменной  b
if (b<10)                        //Условный оператор if …else
        a=b+c+d;                 //Вычисление переменной a при
                                 //выполнении условия
        else                     //Условный оператор if …else (2-я часть)
        a=b-c-d;                 //Вычисление переменной a при
                                 //невыполнении условия
cout<<"a = "<<a;                 //Вывод на экран значения a
getch();                         //Функция задержки окна DOS на экране
return 0;
}
```

Исследуем программу для определения переменной ***a*** по условию предыдущей задачи. Отличие будет состоять в использовании сложных операторов в условном операторе ***if…else***.

Блок-схема алгоритма решения данного задания представлена на рис. 5.3. Обратите внимание на отличия данной блок-схемы от схемы, приведенной на рис. 5.2.

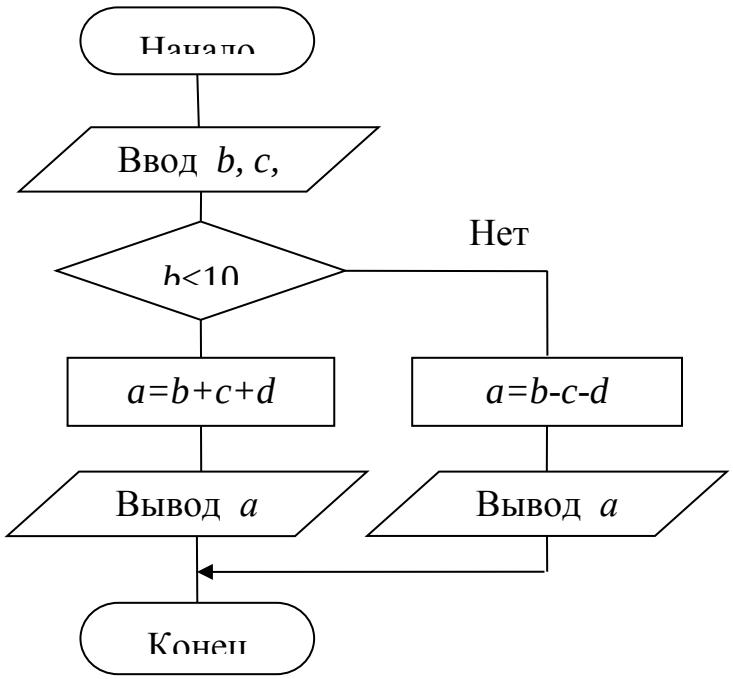


Рис. 5.3. Блок-схема алгоритма определения переменной **a** (использование сложных операторов)

Ниже приведен программный код, реализующий данный алгоритм. Обратите внимание на сдвиг фигурных скобок, ограничивающих сложные операторы, относительно скобок,

охватывающих тело главной функции (удобства чтения и анализа).

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>                 //Директива include подключает
                                    //файл ввода–вывода iostream
using namespace std;                //Подключает все имена из
                                    //пространства имен  std
int _tmain(…)                       //Объявление главной функции _tmain
{                                   //Начало главной функции
int a, b;                           //Объявление переменных целого
//типа
int c=2, d=3;                       //Объявление переменных целого
типа и их    инициализация
cout<<"Vvvedite b"<<endl;           //Вывод на экран приглашения  Vvedite b
cin>>b;                             //Ввод значения переменной  b
if (b<10)                           //Условный оператор if…else
        {                           //Начало сложного оператора
        a=b+c+d;                    //Вычисление переменной a при
                                    //выполнении условия
        cout<<"a=b+c+d= "<<a;       //Вывод на экран значения переменной  a
        }                           //Конец сложного оператора
        else                        //Условный оператор if …else)
        {                           //Начало сложного оператора
        a=b-c-d;                    //Вычисление переменной a при
                                    //невыполнении условия
        cout<<"a=b-c-d= "<<a;       //Вывод на экран значения переменной  a
        }                           //Конец сложного оператора
getch();                            //Функция задержки окна DOS на экране
return 0;
}
```

Условный оператор *if…else* для 3-х интервалов значений переменой. Выше рассматривались случаи применения условного оператора *if...else* для 2-х интервалов значений переменой, т. е. рассматривались простые условия (логические выражения) типа ***x<a***. При этом число ***a*** делит числовую ось ***x*** на два интервала (рис. 5.4).

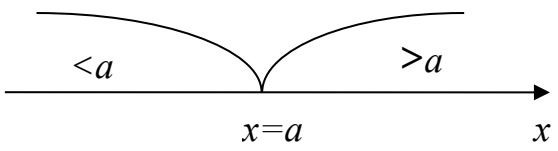


Рис. 5.4. Графическая интерпретация логического выражения для двух интервалов значений **x**

Язык *C++* и среда *Visual C++*2010 позволяют в случае необходимости применять в условном операторе *if...else* и более сложные логические выражения с использованием различных логических операций.

Рассмотрим случай, когда в условии оператора *if...else* указан некоторый интервал значений, с которым сравнивается переменная. Например, переменная **x** должна находиться в интервале значений от ***a*** до ***b*** (***a<b***) (рис. 5.5). В этом случае условие запишется следующим образом:

***if* (*x*>=*a* && *x*<=*b*) ,**

где **&&** - операция логического **И.**

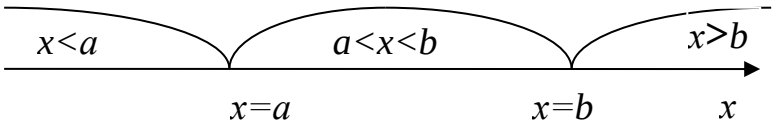


Рис. 5.5. Графическая интерпретация логического выражения для трех интервалов значений **x**

Вложенный условный оператора *if…else*. Во многих случаях использование простого (одинарного) оператора *if...else* не обеспечивает решение поставленной задачи. Очевидно, что данный оператор, например, не позволяет вычислительному процессу принять решение в случае, когда для какой-то переменной существует четыре и более интервала ее значений. В этом случае применяют вложенный условный оператор *if...else*.

На рис. 5.6 приведен пример структуры вложенного условного оператора *if...else* (в качестве базовой использована структура, изображенная на рис. 5.1).

Таких вложений можетбыть несколько в зависимости от конкретной задачи. Чтобы не ошибиться в процессе отладки задачи, необходимо в программном коде каждый вложенный условный оператор (вместе с выполняемыми в нем операторами) смещать табулятором вправо. Это значительно облегчает чтение программного кода.

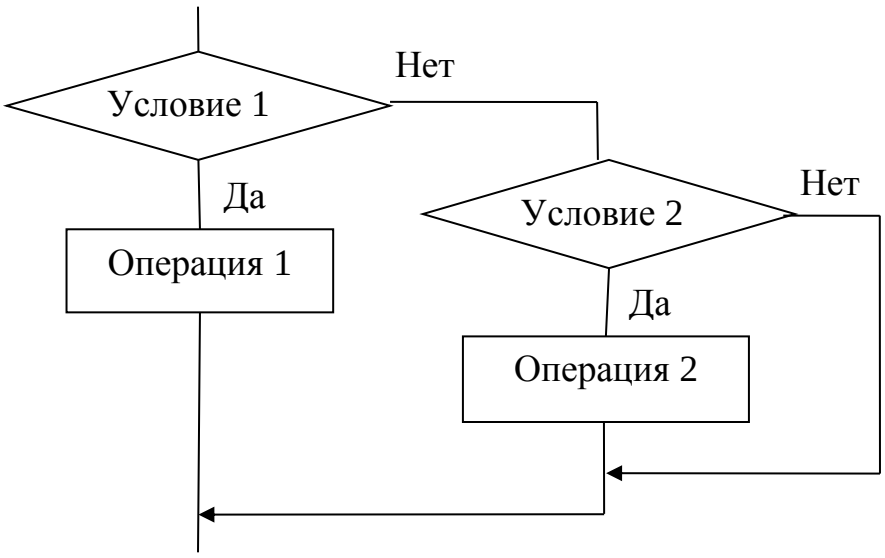


Рис.5.6. Структура вложенного условного оператора ***if…else***

5.2. Оператор выбора *switch*

Оператор ***switch*** (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Он обеспечивает выбор из нескольких вариантов на основании некоторого фиксированного параметра. Формат оператора следующий:

switch **(выражение)**
{
case **константа_1: оператор_1;**
case **константа_2: оператор_2;**
case **константа_3: оператор_3;**
.....
case **константа_*n*: оператор_n;**
[*default*: оператор;]
}

Блок-сема алгоритма, реализующего работу оператора выбора *switch*, приведена на рис. 5.7.

Выполнение оператора *switch* начинается с вычисления выражения в скобках (оно должно быть целочисленным). Его значение последовательно сравнивается с константами, которые записаны следом за каждым оператором ***case***. Затем управление

передается тому оператору, который помечен константным выражением (меткой), значение которого совпало с вычисленным выражением в условии. После этого последовательно выполняются все остальные ветви, если выход из переключателя явно не указан.

Все константные выражения должны иметь разные значения, но быть одного и того же целочисленного типа. Несколько меток могут следовать подряд. Если совпадения не произошло, выполняются операторы, расположенные после слова ***default*** (а при его отсутствии управление передается следующему за *switch* оператору).

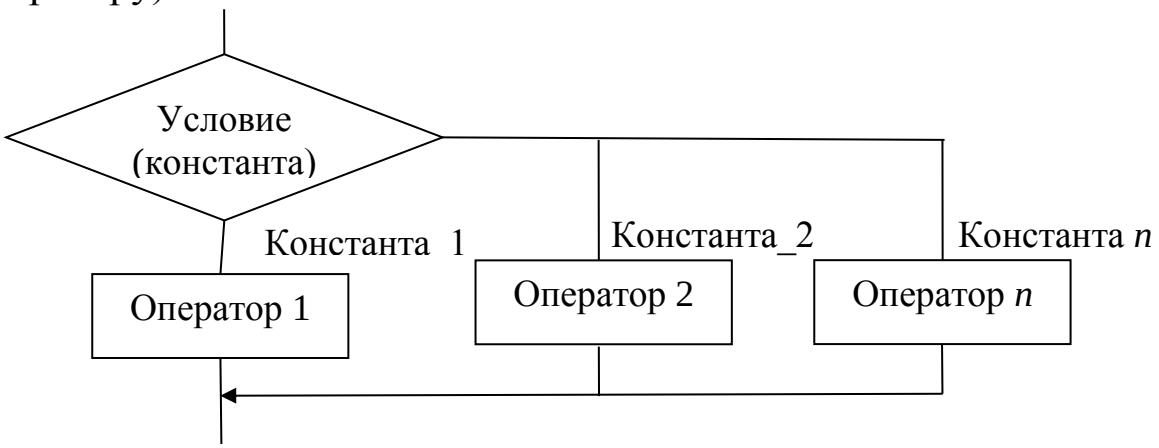


Рис. 5.7. Блок-схема алгоритма работы оператора *switch*

Проиллюстрируем работу оператора *switch* на примере программы, реализующей простейший калькулятор на 4 действия. Здесь консольно вводятся две переменные ***a*** и ***b*** и знак операции, которую необходимо совершить с этими переменными. Оператор *switch* сравнивает введенный знак операции со знаком, содержащемся в четырех метках ***case***, и производит необходимую арифметическую операцию. Результат выводится на экран. Если знак операции не совпадает с содержащимися в метках, то на экран выводится сообщение о неизвестной операции. Рассмотрим код:

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;          //Подключает все имена из пространства  std
int _tmain(…)                 //Объявление главной функции
```

```
{                                   //Начало главной функции
double a, b, res;                   //Объявление переменных
char operation;                     //Символьная переменная
cout << "Vvedite a : "<<endl;
cin >> a;                           //Ввод значения переменной  a
cout <<"Vvedite
operation: "<<endl;
cin >> operation;                   //Ввод знака операции
cout << "Vvedite b : "<<endl;       //Вывод на экран надписи-приглашения
                                    //Vvedite b :
cin >> b;                           //Ввод значения переменной b
switch (operation)                  //Условие в операторе  switch
{                                   //Начало оператора  switch
case '+': res = a + b; break;       //Метка ”+” и вычисление переменной res
                                    //для этой метки
case '-': res = a - b; break;       //Метка ”-” и вычисление переменной res
                                    //для этой метки
case '*': res = a * b; break;       //Метка ”*” и вычисление переменной res
                                    //для этой метки
case '/': res = a / b; break;       //Метка ”/” и вычисление переменной res
                                    //для этой метки
default : cout <<\                  //Сообщение о неизвестной операции
"Unknown operation"<<endl;
}                                   //Конец сложного оператора
cout << "Result : " << res;         //Вывод на экран результата
getch();                            //Функция задержки окна DOS
return 0;
}                                   //Конец главной функции
```

5.3. Безусловный оператор *goto*

Переход к любому оператору программы без условия (директивный переход) в среде *Visual C*++2010 осуществляется с помощью оператора безусловного перехода ***goto***. В отличие от оператора *if…else* этот оператор осуществляет переход в нужное место программы без выполнения каких-либо условий. Оператор имеет следующий вид:

***goto* Метка;**

Меткой обозначают какой-либо оператор, на который должен быть осуществлен переход из места установки оператора *goto*. В качестве метки выступает идентификатор с расположенным за ним символом двоеточия:

Метка: оператор;

Как только выполнение программы достигает оператора *goto*, управление передается оператору, помеченному меткой.

Проиллюстрируем работу оператора *goto* на примере программы из подраздела 5.1 (рис. 5.3). Расположим оператор *goto* с меткой *A* перед условным оператором, а саму метку *A* – перед концом программы. Теперь условный оператор не выполнится, т. к. компьютер выполнит оператор ***goto A*** и перейдет сразу к выводу надписи ***"Perehod po metke";***

```
#include "stdafx.h"
#include <conio.h>                  //Файл conio.h обеспечивает задержку
                                    //окна DOS на экране дисплея
#include <iostream>                 //Директива include подключает файл
                                    //ввода–вывода iostream
using namespace std;
int _tmain(…)                       //Объявление главной функции _tmain
        {                           //Начало главной функции
        int a, b;                   //Объявление переменных целого типа
        int c=2, d=3;               //Объявление переменных целого типа  и
                                    //их инициализация
        cout<<"Vvvedite b"<<endl;
        cin>>b;                     //Ввод значения переменной  b
        goto A;                     //оператор goto
        if (b<10)
        {
        a=b+c+d;
        cout<<"a=b+c+d= "<<a;
        }
        else
        {
        a=b-c-d;
        cout<<"a=b-c-d= "<<a;
        }
A: cout<<"Perehod po metke ";       //Метка A и оператор вывода
getch();
```

```
return 0;
}
```

Выводы. Поскольку вся информация в компьютере представлена в числовом виде, то сравнение числовых значений – это сущность механизма принятия решений в вычислительных процессах на языке С++.

Структура ветвления описывается в языке *С++* с помощью условного оператора.

Вопросы для самоконтроля

1. Запишите в общем виде оператор условного перехода.
2. Какие базовые операторы существуют для сравнения значений двух переменных?
3. Записан оператор ***if*(условие) *S*;**. Если условие не выполняется, то какой оператор будет выполнен?
4. Оператор условного перехода имеет вид ***if*(условие) *s*1; *else* *s*2;**. Если условие не выполняется, то какой оператор будет выполнен?
5. Запишите в общем виде оператор безусловного перехода.
6. Записан оператор: ***if*(*X*>0) {*X*=*X*–1; *Y*=0; } *else* *Y*=*Y*+1;**. Сколько ошибок в приведенном операторе?
7. Укажите на возможность такой записи оператора ***if*(1<*X*<3) *S*1; *else* *S*2;**
8. Какое значение будет иметь переменная ***Z*** после выполнения операторов ***X*=1;*Y*=1;*Z*=0; *if*(*X*>0) *if*(*Y*>0)*Z*=1;*else* *Z*=2;**?
9. В каких скобках записывается составной оператор в языке *С++*?
10. Как записать в операторе ***if*(условие)** проверку условия на равенство переменной нулю?
11. Сколько ошибок сделано при записи оператора ***if*(*a*>*b*) *x*=a *else* *x*=*b***?
12. Текущий оператор выполняется какое-то число раз, пока не будет выполнено завершающее его условие. Какая это вычислительная структура?
13. В операторе ***if*** проверяется условие ***if*((*x*>=*a*) &&(*x*<=*b*))**. Что означает выполнение этого условия?

ЛЕКЦИЯ 6
ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ В *VISUAL C++* 2010

Цель работы: изучение циклических программ с использованием операторов цикла ***while*** **и** ***do...while*****.**

Вопросы лекции:

1. Циклические вычислительные процессы.
2. Оператор цикла ***while*** в среде Visual *C++* 2010.
3. Исследование оператора цикла ***do...while*** в среде *Visual C++*.
4. Вложенные циклы в *Visual C++* 2010.

6.1. Циклические вычислительные процессы

Возможность повторно выполнять некоторые действия очень важна при разработке любых программ и программных приложений. Вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, называется **циклическим.**

Цикл выполняет оператор или группу операторов до тех пор, пока истинно (или ложно) определенное условие относительно некоторой переменной, называемой **параметром цикла.**

Многократно повторяющиеся части такого процесса составляют **тело цикла.**

Алгоритм циклических структур должен содержать (рис. 6.1):

1. **Подготовку к циклу** – присваивание начального значения параметру цикла.

2. **Проверку условия** выполнения тела цикла.

3. **Тело цикла** – действия, которые выполняются в циклической программе для разных значений параметра цикла.

4. **Изменение (модификация)** значений параметра цикла.

На рис. 6.1 изображена блок-схема алгоритма циклического вычислительного процесса, где помимо характеристик операционных блоков в качестве примера в скобках приведены реальные операторы.

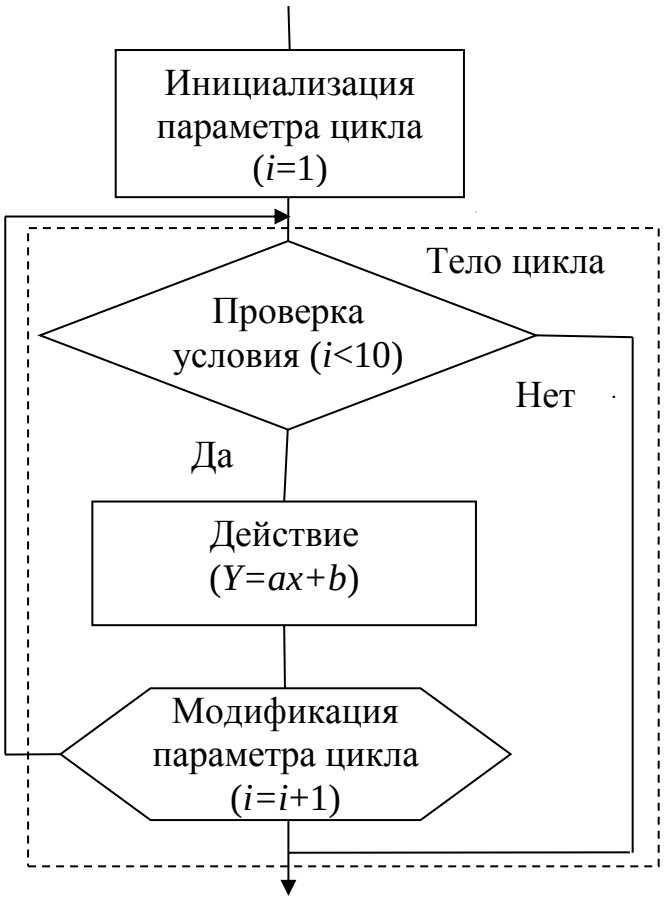


Рис. 6.1. Блок-схема алгоритма циклического вычислительного процесса

В среде *Visual C++* 2010 циклические вычислительные процессы реализуются с помощью операторов ***while***, ***do...while*** и ***for***, анализ которых будет проведен ниже.

Для лучшего понимания действия операторов в настоящей работе следует знать перевод следующих английских слов:

do – делать, выполнять;
while – пока;
for – для.

6.2. Оператор цикла *while* в среде *Visual C++* 2010

Оператор цикла ***while*** реализует вычислительную структуру **цикл с предусловием** и имеет следующий вид:

***while* (логическое выражение)**
оператор; .

В качестве **логического выражения** используется отношение или логическое выражение относительно параметра цикла в виде какого-то условия. Если оно истинно, то тело цикла (**оператор**) выполняется до тех пор, пока логическое выражение не перестанет выполняться, т. е. не станет ложным. **Оператор** может быть простым или составным (из нескольких операторов, заключенных в фигурные скобки).

Блок-схема алгоритма циклического вычислительного процесса с предусловием, реализуемого оператором цикла *while*, приведена на рис. 6.1.

Проанализируем простейшую программу с оператором цикла *while*, вычисляющую значения функции: $\boldsymbol{Y=ax + sin(\pi x)}$, если значение x изменяется от x_0 до x_k с шагом Δx.

Блок-схема алгоритма решения данного задания представлена на рис.6.2.

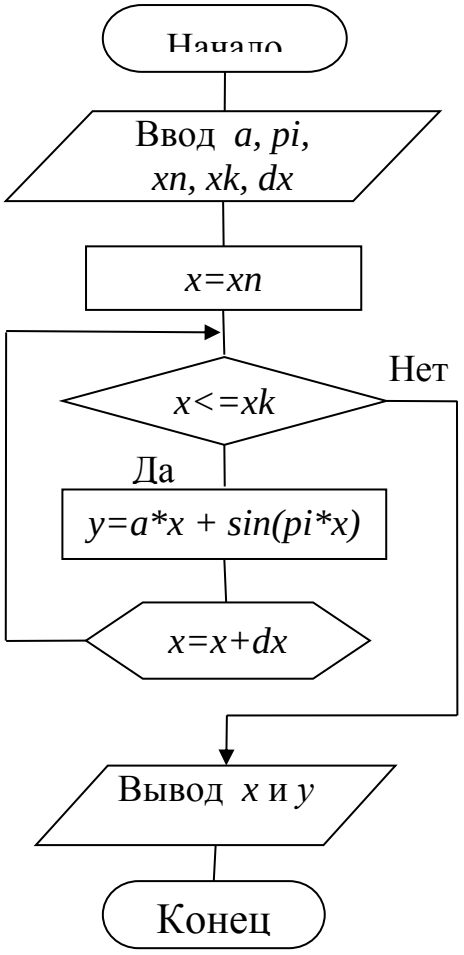


Рис. 6.2. Блок-схема алгоритма программы с оператором *while*

Определим типы и значения исходных данных, которые понадобятся для решения задачи:

a – константа, инициализируем ее, как *a* = 2 и определим ее тип как *int* (целое);

π – константа, ее инициализируем как *pi* = 3.14, тип *double* (действительное двойной точности);

x_0 – переменная, обозначим ее как *xn*, тип *double*;

x_k – переменная, обозначим ее как *xk*, тип *double*;

Δx – переменная, обозначим ее как *dx*, тип *double*;

x – переменная, обозначим ее как *x*, тип *double*;

y – переменная, обозначим ее как *Y*, тип *double*.

Перед написанием программ рекомендуется всегда составлять такой список переменных и идентификаторов, используемых в конкретной задаче.

Ниже приведен программный код, реализующий разработанный алгоритм. Обратите внимание на сдвиг фигурных скобок, ограничивающих сложные операторы, относительно скобок, охватывающих тело главной функции.

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;

int _tmain(…)                    //Объявление главной функции _tmain
{
const int a=3;                   //Инициализация целой константы a
const  double  pi=3.14;          //Инициализация действительной
константы  π
double x, y, xn, xk, dx;         //переменных и параметра цикла: Y, x, x0,
xk  и Δx
cout<<" Vvedite xn, xk, dx: "<<endl;      //Вопрос для ввода  параметров
цикла
cin>>xn>>xk>>dx;                 //Ввод  исходных данных
cout<<"xn= "<<xn<<"  xk=
"<<xk<<"  dx= "<<dx<<endl;       //Вывод на экран исходных данных
x=xn;                            //Подготовка к циклу – предоставление
                                 //начального значения параметру цикла x
cout<<" X   "<<"   Y  "<<endl;   //Вывод на экран заголовков "X" и "Y"
      while (x<=xk)              //Оператор while (условие цикла x ≤ xk)
```

```
        {                                   //Начало составного оператора
        y=a*x + sin(pi*x);                  //Вычисление переменной y на данном
шаге
        cout<<x<<"   "<<y<<endl;           //Вывод на экран действительной
переменной y
        x=x+dx;                             //Вычисление следующего значения x
        }                                   //Конец составного оператора
getch();                                    //Функция задержки окна DOS на экране
return 0;
}                                           //Конец главной функции
```

Обратите внимание, что оператор *while (x<=xk)* и другие операторы, входящие в тело цикла, сдвинуты относительно остальных операторов главной функции и программы в целом. С такими сдвижками принято записывать программный код, чтобы он легко читался программистом. Это не влияет на выполнение программы и особенно полезно в случае, когда программы достатточно велики. В данном конспекте большее число таких сдвижек не удалось разместить из-за ограниченного формата.

6.3. Оператор цикла *do...while* в среде *Visual C++* 2010

Оператор цикла ***do...while*** реализует вычислительную структуру **цикл с послеусловием** и имеет следующий вид:

do
оператор;
while **(логическое выражение);**

В качестве **логического выражения** используется отношение или логическое выражение относительно параметра цикла в виде какого-то условия. Если оно истинно, то тело цикла (**оператор**) выполняется до тех пор, пока **выражение** не перестанет выполняться, т. е. не станет ложным. Оператор может быть простым или составным (из нескольких операторов, заключенных в фигурные скобки).

Основное отличие оператора цикла *do…while* от оператора *while* состоит в том, что здесь условие относительно параметра цикла проверяется после выполнения тела цикла. Поэтому оператор *do…while* выполняется как минимум один раз.

Вычислительный процесс, реализующий оператор цикла *do…while*, проанализируем на примере решения задачи по вычислению функции $Y = \sin^3(x-a) - \cos^2(x+a)^3$ для значений аргумента x от x_0=0 до $\boldsymbol{x_k}$=1 с шагом $\boldsymbol{\Delta x}$=0,2. Блок-схема алгоритма решения данной задачи приведена на рис. 6.3.

Определим типы и значения исходных данных, которые понадобятся для решения задачи:

a – константа, инициализируем ее, как a = 0,3 и определим ее тип как *double* (действительное двойной точности);

$\boldsymbol{x_0}$ – переменная, обозначим ее как ***xn***, тип *double*;

$\boldsymbol{x_k}$ – переменная, обозначим ее как ***xk***, тип *double*;

$\boldsymbol{\Delta x}$ – переменная, обозначим ее как ***dx***, тип *double*;

x – переменная, обозначим ее как ***x***, тип *double*;

y – переменная, обозначим ее как ***Y***, тип *double*.

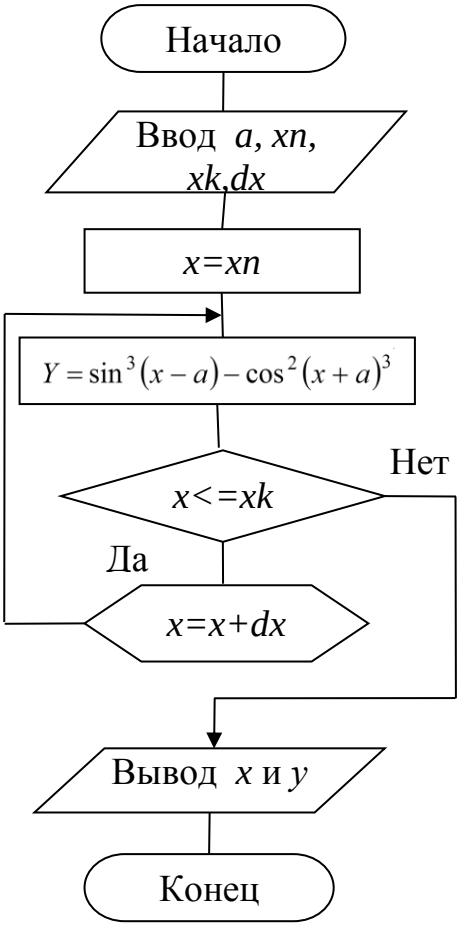


Рис. 6.3. Блок-схема алгоритма программы с оператором *do...while*

Ниже приведен программный код, реализующий данный алгоритм. Обратите внимание на сдвиг оператора цикла и фигурных скобок, ограничивающих сложные операторы, относительно операторов, входящих в тело главной функции.

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;
int _tmain(…)                    //Объявление главной функции _tmain
{
double a=0.3;                    //Инициализация константы a
double x, y, xn, xk, dx;         //Переменные и параметр цикла: Y,
                                 //x, x0,  xk  и Δx
cout<<" Vvedite xn, xk, dx: "\
<<endl;
cin>>xn>>xk>>dx;                 //Ввод  исходных данных
cout<<"xn= "<<xn<<"  xk= "\
<<xk<<"  dx= "<<dx<<endl;       //Вывод исходных данных
x=xn;                            //Подготовка к циклу – предоставление
                                 //начального значения параметру
                                 //цикла – переменной x
cout<<" X   "<<"   Y  "<<endl;   //Вывод на экран заголовков
      do                         //Начало оператора do…while
      {                          //Начало составного оператора
      y=pow(sin(x-a),3)-\
      pow(pow(cos(x+a),3),2);    //Вычисление   y на данном шаге
      cout<<x<<"   "<<y<<endl;  //Вывод на экран действительной
переменной y
      x=x+dx;                    //Вычисление следующего значения
параметра x
      }                          //Конец составного оператора
      while (x<=xk);             //Оператор while (условие x ≤ xk)
getch();
return 0;
}
```

Обратите внимание на знаки “\”, поставленные в операторах *cout* и в середине оператора, рассчитывающего функцию *y*. Это знак переноса части оператора на следующую строку в случае нехватки места. В данном случае на экране места бы хватило.

6.4. Вложенные циклы в *Visual C++* 2010

Циклы можно вкладывать друг в друга. Это следует из записи в общем виде, например, оператора цикла ***while***:

while (логическое выражение)
оператор;

где **оператор** – любой оператор, в том числе и сложный, состоящий из нескольких операторов. Т. к. исключений из этого правила в языке *C++* нет, то в качестве оператора может быть использован любой из операторов цикла.

Тогда вложенный цикл с оператором *while* будет иметь следующий вид:

***while* (логическое выражение)**
{
    ***while* (логическое выражение)**
    оператор;
},

где **оператор** – любой оператор, в том числе и сложный.

В электронной версии конспекта на сайте ХНАДУ можно ознакомится с программами, использующей вложенные циклы.

Выводы. Вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, называется циклическим.

Цикл выполняет оператор или группу операторов до тех пор, пока истинно (или ложно) определенное условие относительно некоторой переменной, называемой параметром цикла. В среде *Visual C++* 2010 циклические вычислительные процессы реализуются с помощью операторов *while*, *do...while* и *for*.

В операторе цикла в качестве логического выражения используется отношение или логическое выражение относительно параметра цикла в виде какого-то условия. Если оно истинно, то тело цикла (оператор) выполняется до тех пор, пока логическое выражение не перестанет выполняться, т. е. не станет ложным.

ЛЕКЦИЯ 7
ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ В *VISUAL C++* 2010. ОПЕРАТОР ЦИКЛА *FOR*

Цель работы: изучение вопросов разработки циклических программ с использованием оператора цикла *for*.

Вопросы лекции

1. Циклические вычислительные процессы.
2. Оператор цикла *for* в среде *Visual C++* 2010.
3. Операции инкремента и декремента.
4. Использование оператора цикла *for*.
5. Вложенные циклы *for* в *Visual C++* 2010.

7.1. Циклические вычислительные процессы

В предыдущей лекции были рассмотрены циклические вычислительные процессы в Visual C++ 2010 и реализующие их операторы *while* и *do...while*. В данной лекции будет изучена разработка циклических программ с использованием оператора цикла **for**. Кратко напомним основные понятия циклических вычислительных процессов.

Циклическим называется вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям.

Цикл выполняет оператор или группу операторов до тех пор, пока истинно (или ложно) определенное условие относительно некоторой переменной, называемой **параметром цикла**.

Многократно повторяющиеся части такого процесса составляют **тело цикла**.

Алгоритм циклических структур должен содержать (рис. 7.1):

1. **Подготовку к циклу** – присваивание начального значения параметру цикла.
2. **Проверку условия** выполнения тела цикла.
3. **Тело цикла** – действия, которые выполняются в циклической программе для разных значений параметра цикла.

4. **Изменение (модификация)** значений параметра цикла.

На рис. 7.1 изображена блок-схема алгоритма циклического вычислительного процесса, где помимо характеристик операционных блоков в качестве примера приведены реальные операторы.

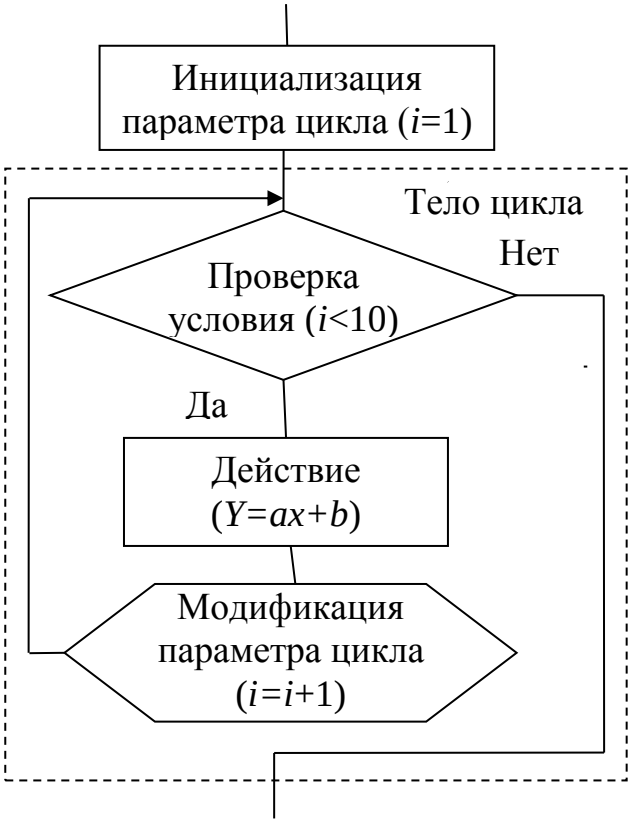


Рис. 7.1. Блок-схема алгоритма циклического вычислительного процесса

В среде *Visual C++* 2010 циклические вычислительные процессы реализуются с помощью операторов *while, do...while* и *for*. Ранее были исследованы операторы *while* и *do...while*. В настоящей работе будут исследованы программы с использованием оператора *for*.

7.2. Оператор цикла *for*

Оператор цикла ***for*** используется, когда количество повторений тела цикла **заранее известно**. Форма записи оператора цикла *for* следующая:

***for* ([выражение инициализации]; [выражение проверки (условие)]; [выражение модификации])**
оператор внутри цикла;

Квадратные скобки показывают, что данная секция в операторе может быть опущена.

На практике это выглядит, например, следующим образом:

```
for(i=1; i<=n; i=i+1)
Y =A*i;
```

где ***i*** – параметр цикла.

Анализ данной записи показывает, что оператор *for* объединяет в себе три операционных блока из блок-схемы циклического вычислительного процесса (рис. 7.1):

1. Блок инициализации, т. е. присвоения параметру цикла начального значения (*i*=1);
2. Блок проверки условия (*i*<=*n*);
3. Блок модификации параметра цикла (*i*=*i*+1).

Это свойство оператора цикла *for* позволяет существенно упростить вычислительные процессы и программные коды при решении различных задач в *Visual C++* 2010.

В схемах алгоритмов оператор цикла *for* отражается символом **модификация** (рис. 7.2):

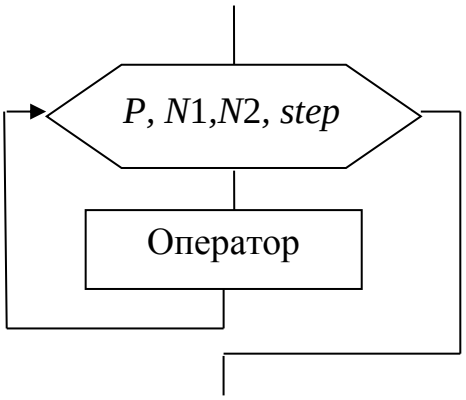


Рис. 7.2. Блок-схема оператора цикла **for**

На схеме алгоритма приведены следующие обозначения:

- ***P*** – параметр цикла;
- ***N*1, *N*2** – границы изменения параметра цикла;

- *step* – шаг изменения параметра цикла (если шаг не указан, то он равняется 1).

Параметр цикла *P*, границы изменения *N*1, *N*2 и шаг *step* должны иметь один и тот же тип.

Возможности оператора цикла *for* очень широки. Например, вместо любого из трех выражений в записи общей формы можно записать два и более выражения, разделенных запятыми:

```
for(i=1, j=1, z=1; i<=n; i=i+1, j=j+1, z=z+1)
 Y =A*i*j*z;
```

Особенностью оператора цикла *for* является то, что параметр цикла *P*, границы изменения *N*1, *N*2 и шаг *step* могут отсутствовать в записи оператора (знаки “**;**” должны присутствовать).

Например, если пропущено условие, то цикл будет выполняться бесконечно. Приведем три примера бесконечных циклов:

```
for(i=0; ; i++) cout<<” Бесконечный цикл ”<<endl;
for(i=1; 1; i++) cout<<” Бесконечный цикл ”<<endl;
for(; ; ) cout<<” Бесконечный цикл ”<<endl;
```

7.3. Операции инкремента и декремента

В языке *C*++ принято операцию приращения цикла ***i=i*+1** записывать как ***i*++**, например:

```
for(i=1; i<=n; i++)
Y =A*i;
```

Такая запись называется операцией инкремента. Если ***i=i*-1**, то записывают ***i*--** и называют операцией декремента.

Эти операции свойственны только языку *C*++. Унарные (одиночные) операции инкремента (декремента) увеличивают (уменьшают) свой операнд (обязательно переменную) на единицу. Они изменяют значение самой переменной, то есть являются

скрытой формой операции присваивания. Иногда эти операции применяют как самостоятельный оператор:

***i*++; *i*--;**

что эквивалентно записи

***i* = *i* +1; *i* = *i* - 1;**

Но эти операции могут использоваться и в выражениях, например,

***sum* = *sum* +*x* *++*i* ;**

Инкремент и декремент реализуются в двух формах: префиксной и постфиксной. Особенности выполнения этих форм следующие.

При префиксной форме ***++i*** (***--i***) - переменная ***i*** сначала увеличивается (уменьшается) на единицу , а затем ее значение используется в выражении.

При постфиксной форме ***i++*** (***i--***) – значение переменной ***i*** сначала используется в выражении и только после вычисления выражения переменная увеличивается (уменьшается) на единицу.

7.4. Использование оператора цикла *for*

Рассмотрим простейший пример оператора цикла *for*. Необходимо разработать программу для вывода на экран в столбец чисел от 1 до заданного числа *n*=15 с шагом 1. Использовать оператор *for*. Блок-схема алгоритма выполнения такого задания приведена на рис. 7.3.

Ниже приведен программный код, реализующий данный алгоритм:

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;
        int _tmain(…)
        {
        int i, n;
        cout<<"Vvedite n "<<endl;
        cin>>n;                                  //Ввод значения переменной  n
                for ( i=1; i< n; i++ )           //Оператор цикла  for
                cout << i <<endl;                //Тело цикла – вывод параметра  i
```

```
getch();
return 0;
}
```

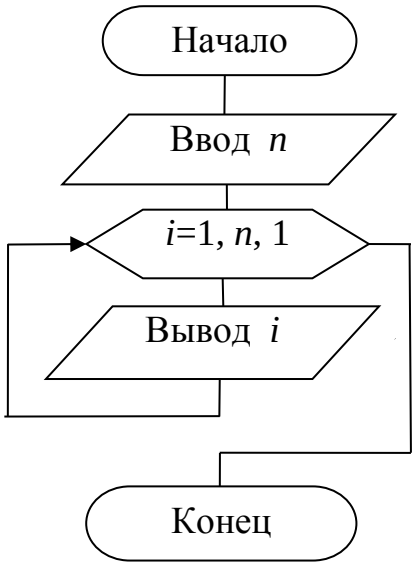


Рис. 7.3. Блок-схема алгоритма для печати в столбец *n* чисел

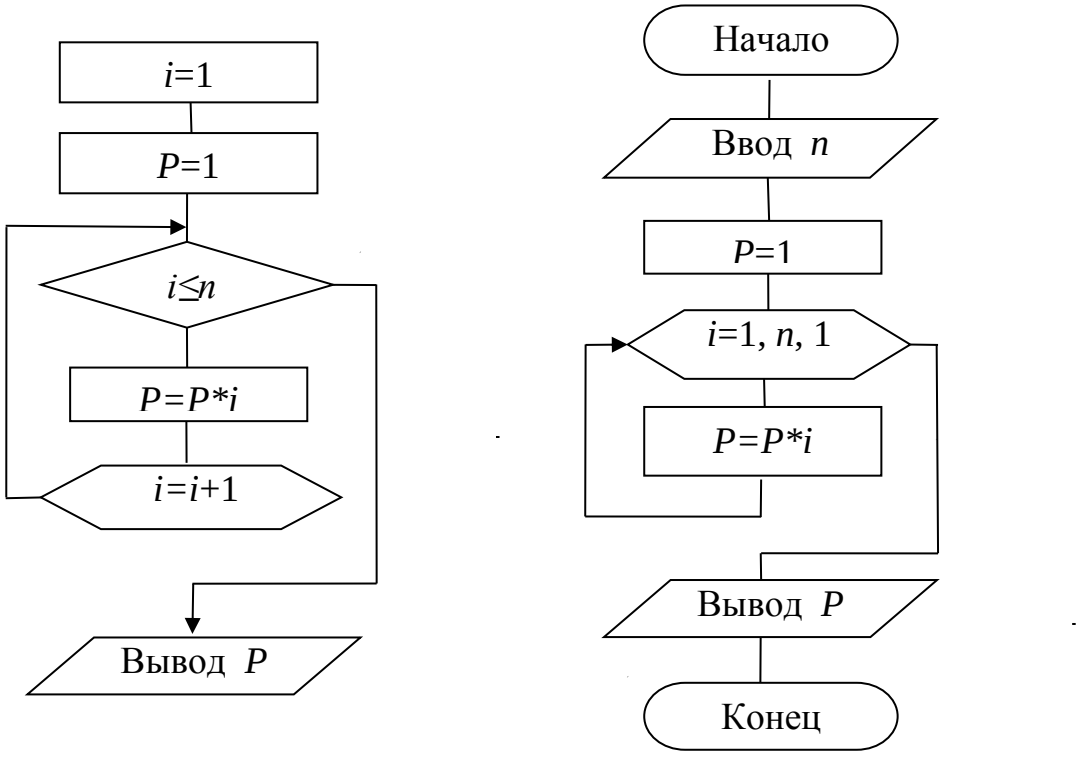
В результате выполнения программы для *n*=15 будет напечатан столбец положительных чисел от 1 до 15. Анализ результатов показывает, что работа оператора цикла *for* происходит в полном соответствии со схемой вычислительного процесса на рис. 7.1. В данном примере параметром цикла *for* является переменная *i* (она же – счетчик). В начале цикла счетчик (переменная *i*) инициализируется значением 1. Потом выполняется тело цикла (*cout*<<*i*<<*endl*;) и проверяется, не достиг ли счетчик значения *n*=15. После каждого выполнения тела цикла счетчик (*i*) увеличивается на единицу. Как только *i* станет равным 15, тело цикла пропускается и управление передается следующему оператору программы.

Разработаем программу вычисления факториала целого числа *n* с использованием оператора цикла *for*. **Факториал** – это произведение целых чисел от 1 до *n*. Необходимо проанализировать блок-схему вычислительного процесса и алгоритм решения задачи при помощи оператора цикла *for*, ввести код программы, построить решение и произвести вычисления для n=10.

Факториал числа *n* вычисляется по следующей формуле:

$$n! = 1*2*3*4* \ldots *n = \prod_{i=1}^{n} i$$

При вычислении факториала начальному значению произведения П необходимо присвоить 1. При каждом выполнении тела цикла $П_{i-1}$ будем умножать на *i*. Примем для обозначения произведения идентификатор *P*. Блок-схемы вычислительного процесса и алгоритма решения задания при помощи оператора цикла *for* имеют следующий вид:



а б

Рис. 7.4. Блок-схемы вычисления факториала целого числа *n*:
а) вычислительный процесс; б) алгоритм с использованием оператора *for*

Учитывая блок-схемы на рис. 7.4, программу вычисления факториала можно записать в следующем виде:

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;
int _tmain(…)
```

```
{
int i, n, p;
cout<<"Vvedite N"<<endl;
cin>>n;                          //Ввод значения переменной  n
p =1;                            //Начальное значение 1 переменной
                                 //P для произведения
      for (i=1; i<=n; i++)       //Оператор цикла  for
      p =p*i;                    //Тело цикла – произведение P и  i
      cout<<"faktorial " <<n\
      <<"   =  "<<p<<endl;      //Печать результата
getch();
return 0;
}
```

После запуска программы на экране появится результат:

Vvedite celoe chislo: 5
factorial 5   = 120

Обратите внимание на необходимость присваивания начального значения (единица) переменной *P* для вычисления последующих произведений. Причем сделано это должно быть до начала цикла.

Исследуем программу для вычисления суммы *n* четных чисел, начиная от двух. Формула для вычисления суммы *n* четных чисел, начиная от двух, имеет следующий вид:

$$s = \sum_{i=1}^{n} (2 * i),$$

где *i* – параметр цикла (номер четного числа на данном шаге).

Блок-схема алгоритма решения такой задачи приведена на рис. 7.5. Обратите внимание на необходимость присваивания начального значения (единица) переменной *S* для вычисления последующих сумм. Причем сделано это должно быть до начала цикла.

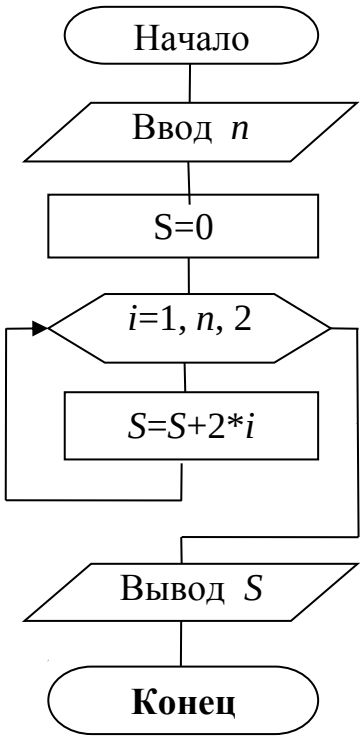


Рис. 7.5. Блок-схема алгоритма для вычисления суммы *n* четных чисел

Программа вычисления суммы *n* четных чисел имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i, n, S;
cout<<"Vvedite N"<<endl;
cin>>n;                          //Ввод значения переменной  n
S =0;                            //Начальное значение 0 переменной
                                 //S для суммирования
      for ( i=1; i<=n; i++ )     //Оператор цикла  for
      S=S+2*i;                   //Тело цикла – расчет суммы
      cout<<"Summa  " <<2*n\
      <<"   =  "<<S<<endl;      //Печать результата
getch();
return 0;
}
```

После запуска программы на экране появится результат:

Vvedite celoe chislo: 5
Summa 10 = 30

Обратите внимание на необходимость присваивания начального значения (ноль) переменной **s** для вычисления последующихсумм.

Выводы. Оператор цикла *for* используется, когда количество повторений тела цикла заранее известно. Анализ показывает, что оператор *for* объединяет в себе три операционных блока из блок-схемы циклического вычислительного процесса:

1. Блок инициализации, т. е. присвоения параметру цикла начального значения (i=1);
2. Блок проверки условия ($i<=n$);
3. Блок модификации параметра цикла ($i=i$+1).

Вопросы для самоконтроля

1. Перечислите, что должно быть в конструкции цикла.
2. С помощью каких действий реализуются циклические вычислительные процессы в языке *C*++?
3. Запишите в общем виде оператор цикла ***for*** и опишите, как он выполняется.
4. Чем оператор цикла ***for*** отличается от операторов цикла ***while*** и ***do...while***?
5. В каких случаях используется оператор цикла ***for***?
6. Запишите и дайте краткую характеристику операции инкремента?
7. Сколько раз выполнится оператор цикла ***int*** **N=3;** ***for*** **(*int i*=1; *i*<=*N*; *i*++)** ***N*=*N*-1;**?
8. Сколько раз выполнится в программе оператор цикла ***for*****(*i*=0; *i*<1; *i*++)** ***cout*** **<<i;**?
9. Записан оператор ***for*****(*i*=2; *i*<10; *i*+=2)** **cout<<*i* ;**. Что будет выведено на экран дисплея?

ЛЕКЦИЯ 8
ОДНОМЕРНЫЕ МАССИВЫ И ИХ ОБРАБОТКА В *VISUAL C++* 2010

Цель лекции: изучить особенности обработки одномерных массивов в среде *Visual C++* 2010.

Основные вопросы лекции

1. Одномерные массивы данных в среде *Visual C++* 2010.
2. Ввод и вывод одномерных массивов в *Visual C++* 2010.
3. Присваивание и копирование одномерных массивов.
4. Поиск элемента в одномерных массивах в *Visual C++* 2010.
5. Сортировка элементов в одномерных массивах.

8.1. Одномерные массивы данных в Visual C++ 2010

Массив – это конечная именованная последовательность однотипных величин.

Массив в информатике – это некоторое множество мест в памяти компьютера, называемых **элементами массива,** к которым можно обратиться по одному имени переменной. Каждый из **элементов** хранит единицу данных определенного типа (тип данных одинаков для всех элементов массива).

Каждый элемент массива определяется именем массива и его порядковым номером в массиве (индексом). Индекс элемента массива – всегда целое число.

Массивы бывают одномерными и многомерными. В данной лекции рассмотрим работу с **одномерными массивами** в *Visual C++* 2010. Одномерные массивы еще называют **векторами.**

Для использования массива в программе его необходимо **объявить**, т. е. зарезервировать под массив определенное количество ячеек памяти.

При объявлении массива указывается тип элементов массива, имя массива и его размер:

Тип Имя массива[размер]; .

Например, оператор ***int A*[8];** описывает целый одномерный массив по имени ***A*** из 8-и целых чисел. В памяти будет зарезервировано место для 8-и целочисленных элементов массива (таб. 8.1).

Таблица 8.1

Значения и расположение в памяти элементов одномерного массива *A*[8]

Элемент массива	*A*[0]	*A*[1]	*A*[2]	*A*[3]	*A*[4]	*A*[5]	*A*[6]	*A*[7]
Индекс элемента (*i*)	0	1	2	3	4	5	6	7
Значение элемента	15	22	34	57	11	29	89	47

Обращение к элементам массива осуществляется по имени массива с указанием индекса (номера элемента массива) в квадратных скобках:

***A*[7]= 47;**
***x* = *A*[3];**
***y* = *A*[5];** .

Индексация элементов в массиве начинается с ноля. Например, если объявлен массив ***int B*[100]**, то элементы массива будут иметь следующие индексы:

***B*[0], B[1], … *B*[99]**.

Тогда оператор

***x*=*B*[13];**

означает, что переменной *x* будет присвоено значение 14-го элемента массива *B*.

Массив, как и переменную, можно инициализировать при объявлении. Значения для последовательных элементов массива отделяются один от другого запятыми и помещаются в фигурные скобки. Например,

***int C*[6]= {2, 4, 7, 11, 12, 13};** .

Если в списке инициализации значений элементов указано меньше, чем размер массива, то имеет место частичная инициализация. При таком объявлении в выражении инициализации после последнего значения для наглядности ставят запятую:

***int C*[6]= {2, 4, 7,};** .

При этом элементам ***C*[0]**, ***C*[1]** и ***C*[2]** будут присвоены значения 2, 4 и 7, а оставшиеся элементы массива инициализации не получат.

Кроме этого, инициализация массивов возможна в процессе выполнения программы – путем записи данных в отведенные для массивов ячейки памяти.

При работе с массивами, в т. ч. одномерными, целесообразно использовать оператор цикла *for*, т.к. известен размер обрабатываемого массива (число элементов массива), т. е. число повторений цикла.

В языке *C++* не проверяется выход индекса за пределы массива. Если имеем целочисленный массив ***int m*[100]**, а в программе указано ***x*=*m*[200]**, то сообщения об ошибке не будет, а переменной **x** будет присвоено произвольное значение.

При обработке массивов в *Visual C++* 2010 все действия в программе выполняются над элементами массива (!), а не над массивом в целом. При этом индекс элемента может быть задан либо его значением, либо выражением:

$A[4],\ F[i+k+1];$.

Над массивами можно выполнять следующие действия:

1. Вводить массивы в память компьютера.
2. Выводить массивы на экран дисплея, на другое устройство или в файл.
3. Присваивать определенные значения элементам массивов.
4. Копировать массивы.
5. Переставлять элементы массивов.

6. Сортировать элементы массивов.

8.2. Ввод и вывод одномерных массивов в *Visual C++* 2010

Т. к. все действия необходимо выполнять над элементами массива, то для ввода массива в память компьютера необходимо организовать его поэлементный ввод посредством оператора цикла *for* (т. к. известен размер массива).

Рассмотрим способы консольного ввода и вывода массива ***A*** из *N* целых чисел. Например, необходимо ввести с клавиатуры массив ***A*** в память, определить его размер и вывести эту информацию на дисплей.

Блок-схема алгоритма решения такой задачи приведена на рис. 8.1.

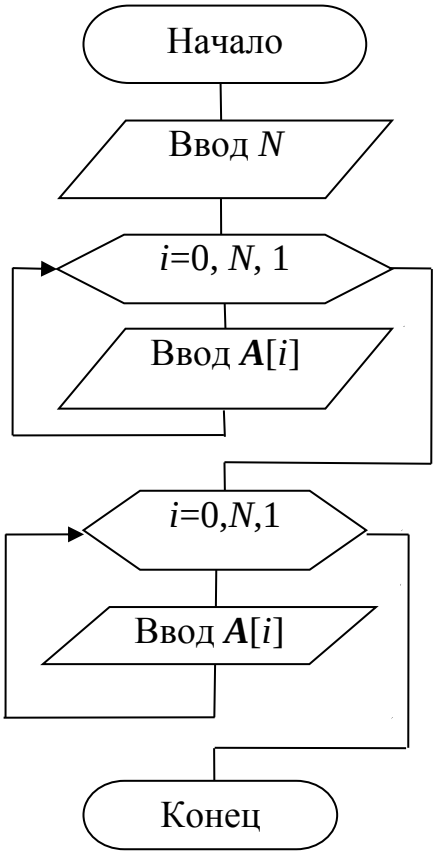


Рис. 8.1. Блок-схема алгоритма ввода и вывода элементов вектора с использованием цикла *for*

Реализующая данный алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
int const N=8;                          //Инициализация размерности массива
int i;                                  //Объявление параметра цикла i
int A[N];                               //Объявление одномерного массива  A
cout<<endl<<"Vvedite massiv:"\
<<endl;
      for(i=0; i<N; i++)                //Цикл по i для ввода массива  A
      cin>>A[i];                        //ввод i-го элемента массива  A
      cout<<endl;
      for (i=0; i<N; i++)               //Цикл по i для вывода A на экран
      cout<<"  A["<<i<<"]= "<<A[i];     //Вывод i-го элемента массива  A
      cout<<endl<<"size= "<<i<<endl;    //Вывод размерности массива  A
getch();
return 0;
}
```

После запуска программы и ввода элементов массива **A** результат на экране будет следующий:

```
Vvedite massiv:
1 2 3 4 5 6 7 8
A[0]=1  A[1]=2  A[2]=3  A[3]=4  A[4]=5  A[5]=6  A[6]=7  A[7]=8
size= 8
```

8.3. Присваивание и копирование одномерных массивов

Элементам массива могут быть присвоены значения выражений. Элементы массива и значения выражений должны иметь один и тот же тип.

Например, объявлен массив *float A*[3], тогда возможна запись

$$A[0]= 3.5;\ A[1]= 0;\ A[2]= a*x + b;\ .$$

Копирование массивов – это присваивание значений элементов одного массива элементам другого массива. При копировании оба массива должны иметь одинаковый размер и тип элементов. Оператор цикла, копирующий массив **A** в массив **B** имеет вид:

***for*(*i*=0; *i*<*N*; *i*++) *B*[*i*]=*A*[*i*]; .**

Используем этот оператор для того, чтобы задать массив ***X*** действительных чисел из 8 элементов непосредственным присваиванием в программном коде. Массив ***Y*** вычислим по заданной формуле:

$$Y_i = 2X_i + \sqrt[3]{X_i^{\,5}}$$

и выведем на экран.

Блок-схема алгоритма решения этой задачи аналогична блок-схеме, приведенной на рис. 7.1.

Реализующая этот алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
int i;
double X[8]={1.2,  3.4,  12,\                  //Инициализация массива X[8]
5.12,  23.1,  3,  0,  35.2};
double Y [8];
      for (i=0; i<8; i++)                      //Цикл для вычисления  Y[i]
      {                                        //Начало составного оператора
      Y [i]=2*X[i]+pow(X[i], 5/3);             //Вычисление элементов Y[8]
      cout<<"  X["<<i<<"]="<<X[i];             //Вывод элементов массива X[8]
      cout<<"  Y["<<i<<"]="<<Y[i];             //Вывод элементов массива Y[8]
      cout<<endl;                              //Переход на новую строку
      }                                        //Конец составного оператора
getch();
return 0;
}
```

В результате выполнения программы получим:

X[0]=1.2 Y[0]=3.6
X[1]=3.4 Y[1]=10.2
X[2]=12 Y[2]=36
X[3]=5.12 Y[3]=15.36
X[4]=23.1 Y[4]=69.3
X[5]=3 Y[5]=9
X[6]=0 Y[6]=0
X[7]=35.2 Y[7]=105.6

8.4. Поиск элемента в одномерных массивах

Поиск элемента в массиве заключается в выделении из массива отдельных его элементов. Поиск может проводиться по образцу или по правилу.

Поиск по образцу заключается в следующем. Задается значение некоторой переменной (образец) и все элементы массива (или часть элементов) сравниваются со значением этой переменной (образцом).

Поиск по правилу проводится на основе проверки некоторых условий, которым должны отвечать либо элемент массива, либо группа элементов.

Рассмотрим пример поиска элемента по образцу.

Пускай задан массив ***A*** из 8 произвольных чисел. Необходимо определить количество элементов, которые больше заданного числа *q* и их порядковые номера. На экран вывести исходный массив, элементы, большие *q* и их порядковые номера.

Для начала определим типы и структуры данных, которые будут использоваться в программе:

q – число, по которому осуществляется поиск элементов исходного массива ***A*;**

i – номер элемента исходного массива ***A*** (параметр цикла);

***A*[8]** – массив из *N* произвольных чисел;

X*[8]** – массив для хранения номеров элементов исходного массива ***A , больших *q***.**

j – номер элемента массива ***X*** для хранения номеров (параметр цикла).

Блок-схема алгоритма поиска элементов одномерного массива приведена на рис.8.2.

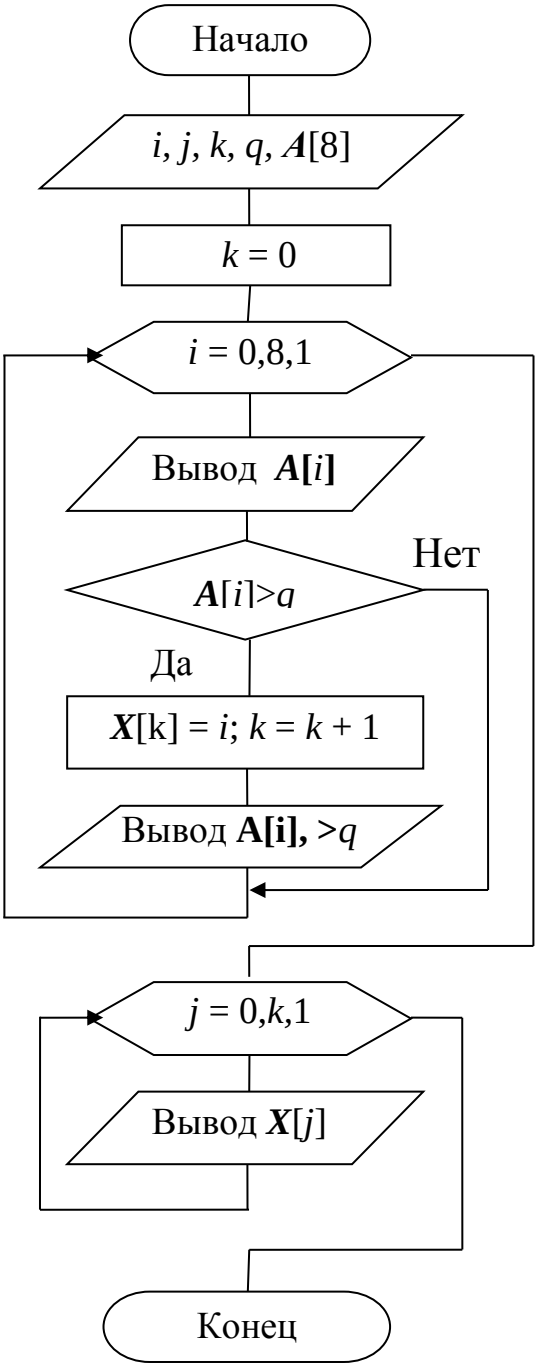


Рис. 8.2. Блок-схема алгоритма поиска элементов одномерного массива

Реализующая алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
int i, j, k;
double q=3.7;                          //Объявление переменной  q
double A[8]={1, 2, 3, 4, 5, 6, 7, 8};  //Инициализация элементов исходного
массива A[8]
int X[8];                              //Объявление массива X[k] номеров
                                       //элементов массива  A[8]
k=0;
      for (i=0; i<8; i++)              //Цикл для перебора элементов A[8]
      {                                //Начало составного оператора в цикле
      cout<<"  A["<<i<<"]\            //Вывод всех элементов массива  A
       = "<<A[i];
      if(A[i]>q)                       //Условный оператор для сравнения
                                       //элементов массива A[8] с q
      {                                //Начало составного оператора
      X[k]=i;                          //Записываются номера элементов,
                                       //больших q,  в массив X[k]
      k=k+1;
      cout<<"  >q:  A["<<i<<"]\       //Вывод  элементов массива A[8],  > q
            = "<<A[i]<<endl;
                        }              //Конец составного оператора
      else
      cout<<endl;
      }                                //Конец составного оператора в цикле
      cout<<endl\
<<"Nomera elementov  >q: ";
      for (j=0;j<k; j++)               //Цикл для вывода номеров элементов
                                       //массива A[8],  > q
      cout<<X[j]<<" ";                 //Вывод номеров элементов массива A[8],
                                       //> q, на экран
getch();
return 0;
}
```

В результате выполнения программы получим:

A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4 >q: A[3] = 4
A[4] = 5 >q: A[4] = 5
A[5] = 6 >q: A[5] = 6
A[6] = 7 >q: A[6] = 7
A[7] = 8 >q: A[7] = 8
Nomera elementov >q: 3 4 5 6 7

8.5. Сортировка элементов одномерных массивов

Сортировка массива – это упорядочивание элементов массива по определенным признакам. Например, необходимо упорядочить массив по возрастанию значений его элементов.

Рассмотрим сортировку массива методом парной перестановки элементов массива.

Элементы массива $x[0]$, $x[1]$, $x[2]$, $x[3]\ldots x[n-1]$ рассматриваются слева направо. В образовавшихся парах элементы по очереди сравниваются:

$$x[0] \text{ и } x[1],\ x[1] \text{ и } x[2],\ x[2] \text{ и } x[3]\ldots x[n-2] \text{ и } x[n-1].$$

Если $x[i-1]>x[i]$, то делается перестановка этих элементов. После того, как просмотрен весь массив, максимальный элемент будет расположен на последнем месте, то есть он будет иметь индекс $n-1$. При втором просмотре массива эта процедура повторяется, но только уже для массива $x[0], x[1], x[2], x[3]\ldots x[n-2]$. При третьем просмотре – для массива $x[0], x[1], x[2], x[3]\ldots x[n-3]$.

Рассмотрим способ парной перестановки соседних элементов одномерного массива. Эта задача решается с использованием вспомогательной переменной, играющей роль буфера.

Например, необходимо поменять местами элементы $A[i]$ и $A[i+1]$. Введем дополнительную переменную C того же типа, что и элементы массива.

Алгоритм перестановки будет следующим (рис. 8.4):

1. Элемент $A[i]$ помещаем в C;

2. Элемент $A[i+1]$ пересылаем на место $A[i]$;

3. Элемент $A[i]$, который находится в C, пересылаем на место $A[i+1]$.

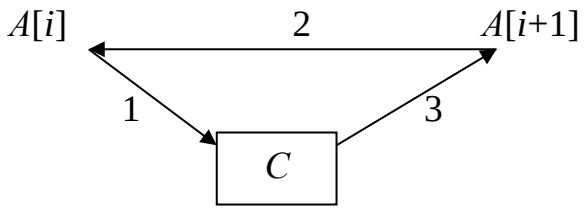


Рис.8.4. Схема перестановки элементов одномерных массивов

Фрагмент программы перестановки двух элементов массива будет иметь следующий вид:

```
C = A[i];
A[i]= A[i+1];
A[i+1]= C;
```

Разработаем на этих принципах программу для сортировки одномерного массива по возрастанию. Например, задан массив ***A*** из 15 произвольных чисел. Отсортировать элементы массива ***A*** по возрастанию. На экран вывести отсортированный массив ***A***.

Вначале определим идентификаторы и типы данных, которые будут использоваться в программе:

i – номер элемента исходного массива ***A***;

A*[15]** – исходный массив из 15 чисел (для удобства отладки программы составим массив ***A из чисел от 0 до14);

B – вспомогательная переменная для хранения максимального элемента массива для i-го шага;

Реализующая алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
int i,j;                          //Объявление индексов i и j
double B;                         //Объявление переменной B
```

```
double A[15]={13,2,0,4,12,\          //Инициализация исходного массива A[15]
6,10,14,9,7,11,5,1,8,3};
      for (j=0; j<15-1; j++)         //Внешний цикл для перебора элементов
                                     //массива A[15]
            for (i=0; i<15-1; i++)   //Вложенный цикл для перебора
элементов
                                     //массива A[15]
            if(A[i]>A[i+1])          //Условный оператор для сравнения
                                     //элементов массива A[15]
            {
            B=A[i];                  //Присваивание B значения i-го элемента
            A[i]=A[i+1];             //Перемена местами i-ого и i+1-ого
                                     //элементов
            A[i+1]=B;
            }
      for (i=0; i<15; i++)           //Цикл для вывода элементов отсортиро-
                                     //ванного массива A[15]
cout<<"  A["<<i<<"] = "<<A[i];
getch();
return 0;
}
```

В результате выполнения программы на экран в строку будет поэлементно выведена отсортированная по возрастанию матрица A:

```
A[0] = 0  A[1] = 1  A[2] = 2  X[3] = 3  A[4] = 4  A[5] = 5  A[6] = 6  A[7] = 7
A[8] = 8  A[9] = 9  A[10] = 10  A[11] = 11  A[12] = 12  A[13] = 13  A[14] = 14
```

Данная программа не является оптимальной. Видно, что исходный массив был отсортирован уже при j=10. При необходимости для устранения избыточных операций можно ввести некоторые критерии.

Выводы. Под массивом в языке *C++* понимают набор данных одного и того же типа, собранных под одним именем. Отдельная единица таких данных, что входит в массив, называется элементом массива. В качестве элемента массива могут выступать данные любого типа (один тип данных для всех элементов массива). Каждый элемент массива определяется именем массива и его порядковым номером в массиве (индексом). Индекс элемента массива – всегда целое число.

При обработке массива все действия в программе выполняются над элементами массива, а не над массивом в целом.

Над массивом можно выполнять следующие действия:

1. Введение массива в память.
2. Выведение массива на экран дисплея.
3. Присвоение определенных значений элементам массива.
4. Копирование массива.
5. Перестановка элементов массива.
6. Поиск элемента в массиве.
7. Сортировка массива.

Вопросы для самоконтроля

1. Что такое массив в информатике?
2. Чем однозначно определяется каждый элемент массива?
3. Какой индекс у 3-его элемента массива ***int B*[100]**?
4. Можно ли объявить массив следующим образом: ***int C*[6]= {2, 4, 7, 11, 12, 13}**?
5. Как объявляется массив в языке *C*++?
6. Какое значение будет присвоено переменной ***X*** в **программе** на *C*++ оператором ***X = m* [13]**?
7. Какие ошибки сделаны при записи оператора ***if* (*A*>7) *B*=*A*;** если обрабатывается массив ***A***?
8. Каким образом в программе на *C*++ выполняются все действия при обработке массива?
9. Каким оператором можно ввести с клавиатуры **n** элементов массива ***X***?
10. Как в программе на *C*++ будет выведен на экран массив ***A*[*k*]**оператором ***for*(*i*=0; *i*<*k*; *i*++) *cout*<<*A*[*i*]<<*endl*;**?
11. Задан массив ***X* = {*X*1, *X*2 ..., *XN*}**. Каким оператором можно вывести этот массив на экран?
12. Возможна ли следующая инициализация массива ***B*** программе на *C*++: ***int B*[6]={2, 4, 7}**?
13. Массивы ***A*** и ***B*** одинакового размера и их элементы имеют тип **int**. Можно ли копировать массив ***A*** в ***B*** операцией ***B*=*A***?
14. При каком условии можно копировать массив ***A*** в массив ***B***?

15. Какой оператор копирует массив ***A*[*N*]** в массив ***B*[*N*]**?

ЛЕКЦИЯ 9
ОПЕРАЦИИ С МНОГОМЕРНЫМИ МАССИВАМИ В *VISUAL C*++ 2010

Цель лекции. Изучить способы и особенности работы с двухмерными массивами в *Visual C*++ 2010.

Основные вопросы лекции

1.Двухмерные массивы данных в среде *Visual C*++ 2010.
2. Консольный ввод и вывод двухмерных массивов.
3. Присваивание и копирование двухмерных массивов.

9.1. Двухмерные массивы данных в *Visual C*++ 2010

Под размерностью массива понимают число индексов, которое необходимо указать для получения доступа к отдельному элементу массива. Массивы, рассмотренные в лабораторной работе № 5, например, были одномерными и требовали только одного индекса. Массивы с более чем одной размерностью, называются многомерными.

Самым простым многомерным массивом является двухмерный массив (матрица).

При объявлении массива указывается тип элементов массива, имя массива и его размер:

Тип ИмяМассива [Размер 1] [Размер 2];

Например, оператор ***int A*[3][8]** описывает целый двухмерный массив по имени **A** из 24-х целых чисел. В памяти будет зарезервировано место для 24-х целочисленных элементов массива (рис. 9.1), которые располагаются в **непрерывном** блоке памяти.

Массивы хранятся в памяти компьютера построчно, начиная с 1-й строки.

Рис. 9.1. Значения и расположение в памяти элементов массива A[3][8]

Возможна инициализации массива непосредственно в программном коде. В этом случае в фигурных скобках приводятся значения элементов массива в следующем виде, например:

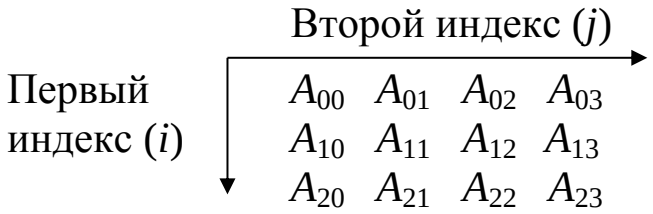
***int B*[2][3]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};**

или

***int B*[2][3]= {{1, 2, 3, 4} {5, 6, 7, 8} {9, 10, 11, 12}};**

Количество инициализаторов не обязано совпадать с количеством элементов массива. Если инициализаторов меньше, то оставшиеся элементы не определены.

Двухмерный массив, например, **int A[3][4]** можно представить в виде следующей матрицы:



Первый индекс – это номер строки в массиве, второй индекс – номер столбца.

В памяти компьютера элементы такой матрицы разместятся в таком порядке:

$A_{00}, A_{01}, A_{02}, A_{03}, A_{10}, A_{11}, A_{12}, A_{13}, A_{20}, A_{21}, A_{22}, A_{23}$.

Кроме этого, инициализация массивов возможна в процессе выполнения программы путем записи данных в отведенные для массивов ячейки памяти.

При работе с массивами, в т. ч. двухмерными, целесообразно использовать оператор цикла ***for***, т.к. известен размер обрабатываемого массива (число элементов массива), т. е. число повторений цикла.

В языке *C++* не проверяется выход индекса за пределы массива. Если массив имеется целочисленный масив ***int m*[100]** , а в программе указано ***x*=*m*[200]**, то сообщение об ошибке не будет, а переменной ***x*** будет присвоено произвольное значение.

При обработке массивов в *Visual C++* 2010 все действия в программе выполняются над элементами массива (!), а не над массивом в целом. При этом индекс элемента может быть задан либо его значением, либо выражением:

$$A[4],\ F[i+k+1]\ .$$

Над многомерными массивами можно выполнять следующие действия:

1. Вводить массивы в память компьютера.
2. Выводить массивы на экран дисплея, на другое устройство или в файл.
3. Присваивать определенные значения элементам массивов.
4. Копировать массивы.
5. Переставлять элементы массивов.
6. Сортировать элементы массивов.

Изучим проведение действий над многомерными массивами на примере двухмерных массивах. Прежде всего разберемся, как производить ввод с клавиатуры и вывод на экран двухмерных массивов.

9.2. Консольный ввод и вывод двухмерных массивов

Для ввода двухмерного массива в память компьютера необходимо организовать его поэлементный ввод посредством вложенного оператора цикла ***for*** (т. к. известен размер массива).

Для консольного вывода двухмерного массива также надо с помощью оператора цикла ***for*** организовать поэлементный вывод исследуемого массива.

Изучим способы консольного ввода и вывода двухмерных массивов. Например, необходимо ввести с клавиатуры матрицу ***A*** размерностью $M \times N$ в память компьютера и вывести эту информацию на дисплей.

Блок-схема алгоритма решения данной задачи приведена на рис. 9.2. Так как

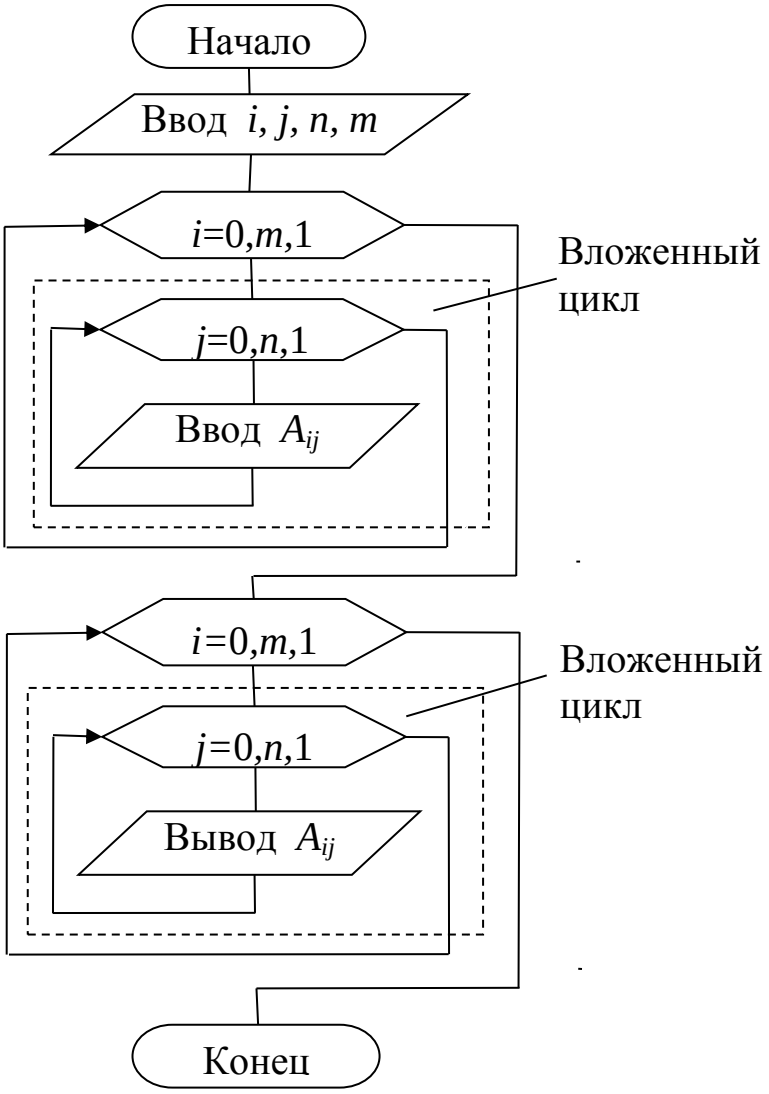


Рис. 9.2. Блок-схема алгоритма поэлементного ввода

и вывода матрицы с использованием цикла **for**

Для придания программе большей универсальности зададим размерность исходной матрицы с запасом, а реальная размерность будет вводиться в каждом конкретном случае. Число строк обозначим ***m***, а столбцов **- *n*.**

Поэлементный ввод матрицы ***A*** организован при помощи двух операторов цикла ***for*** – внешнего и внутреннего (вложенного). Внешний цикл организует перебор элементов матрицы ***A*** по строкам (изменяя номер строки *i*), а вложенный – по столбцам при фиксированном индексе *i*, т. е. в рамках одной строки по номеру столбца *j*.

Вывод исходной матрицы на экран будет производиться аналогичным образом.

Реализующая алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
const int M=10,N=10;              //Число строк и столбцов матрицы A
int i, j, m, n, A[M][N];          //Объявление переменных и матрицы A
cout<<"Vvedite chislo strok i \
stolbcov matricu:"<<endl;
cin>>m>>n;                        //Ввод числа строк и столбцов исходной
                                  //матрицы A
cout<<" Vvedite postrochno \
elementu matricu:"<<endl;
      for(i=0; i<m; i++)          //Внешний цикл ввода элементов A
            for(j=0; j<n; j++)    //Вложенный цикл ввода элементов
                                  //матрицы A
            cin>>A[i][j];         //Ввод очередного элемента матрицы A
cout<<endl;
cout<<"Matrica A"<<endl;
      for(i=0; i<m; i++)          //Внешний цикл вывода элементов
                                  //матрицы A
      {                           //Начало составного оператора для
```

```
                                        //внешнего цикла for
            for(j=0; j<n; j++)          //Вложенный цикл для вывода элементов
                                        //матрицы A
            cout<<A[i][j]<<"   ";       //Вывод элемента Aij матрицы A
cout<<endl;
                    }
                                        //Конец составного оператора для
внешнего цикла for
getch();
return 0;
}
```

После выполнения программы на экран будет выведено:

```
Vvedite chislo strok i stolbcov matricu:
3 4
Vvedite postrochno elementu matricu:
1 2 3 4 5 6 7 8 9 10 11 12
Matrica A
1   2   3    4
5   6   7    8
9   10  11   12
```

9.3. Присваивание и копирование двухмерных массивов

Элементам массива могут быть **присвоены** значения выражений. При этом элементы массива и значения выражений должны иметь один и тот же тип.

Например, объявлен массив ***double A*[3][4]**, тогда возможна запись

$$A[0][0] = 3.5;$$
$$A[1][3] = 0;$$
$$A[2][1] = a*x + b;$$

Копирование – это присваивание значений элементов одного массива элементам другого массива. При копировании оба массива должны иметь одинаковый размер и тип элементов. Копирование массива ***A*** в ***B*** будет иметь вид:

***for*(*i*=0; *i*<N; *i*++) *B*[*i*][*j*]= *A*[*i*][*j*].**

Рассмотрим основные приемы проведения различных операций с матрицами.

Допустим, задана матрица целых чисел ***A*** размерностью 3×4 с элементами от 1 до 12. Необходимо транспонировать матрицу ***A***, т. е. поменять местами ее строки и столбцы, а затем рассчитать матрицу ***B***, элементы которой определяются по формуле

$$B_{ij} = \sqrt[3]{A_{ij}^2} + \sin^2(A_{ij}^3) \ .$$

Элементы исходной матрицы ***A*** введем непосредственно в программном коде. Исходную матрицу ***A***, транспонированную ***AT*** и полученную матрицу ***B*** выведем на экран.

Блок-схема алгоритма решения данной задачи аналогична схеме, приведенной на рис. 9.2. Вместо двух конструкций цикл в цикле применим три:

1. Цикл в цикле для вывода матрицы ***A*** на экран;
2. Цикл в цикле для транспонирования элементов матрицы ***A***, т. е. проведения поэлементной операции $A_{ij}=A_{ji.}$
3. Цикл в цикле для вычисления элементов матрицы ***B*** по заданной формуле и вывода матрицы ***B*** на экран.

Реализующая данный алгоритм программа приведена ниже. При самостоятельной работе на компьютере необходимо учитывать, что все переносы в программном коде обусловлены форматом данного издания. В окне редактора этого делать не придется.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i, j;
double AT[4][3], B[4][3];          //Объявление переменных и матриц A и B
double A[3][4]=\                   //Инициализация исходной матрицы A
{1,2,3,4,5,6,7,8,9,10,11,12};
cout<<"Matrica A"<<endl;           //Вывод названия матрицы

    for(i=0; i<3; i++)             //Внешний цикл по выводу элементов  A
```

```
        {
                for(j=0; j<4; j++)          //Вложенный цикл по выводу элементов A
                cout<<A[i][j]<<"   ";       //Вывод элементов исходной матрицы A
cout<<endl;
        }
//Операция транспонирования матрицы A
        for(i=0; i<3; i++)                  //Внешний цикл для транспонирования
                for(j=0; j<4; j++)          // Вложенный цикл для транспонирования
                AT[j][i]=A[i][j];           //Операция транспонирования матрицы A
cout<<"Matrica AT = \                       //Вывод названия
A transponirovannaya"<<endl;
        for(i=0; i<4; i++)                  //Начало цикла
        {
                for(j=0; j<3; j++)          //Вложенный цикл для вывода AT
                cout<<AT[i][j]<<"  ";       //Вывод элементов матрицы AT
cout<<endl;
        }
cout<<"Matrica B = f(AT)"<<endl; //Вывод названия
                //Вычисление матрицы B
        for(i=0; i<4; i++)                  //Внешний цикл для поэлементного
                                            //вычисление матрицы B
        {
                for(j=0; j<3; j++)          // Вложенный цикл для поэлементного
                                            //вычисление матрицы B
                {
B[i][j]=5*pow(sin(pow(AT[i][j],3)),2)\
+pow((AT[i][j]*AT[i][j]),1./3.);            //Вычисление  Bij = f(ATji)
                cout<<B[i][j]<<"  ";        //Вывод элементов матрицы B
                }
        cout<<endl;
        }
getch();
return 0;
}
```

Результат будет выведен на экран дисплея:

```
Matrica A
1    2    3    4
5    6    7    8
9    10   11   12
Matrica AT = A transponirovannaya
1   5   9
2   6   10
3   7   11
4   8   12
Matrica B = f(AT)
4.54037   4.82155   4.43915
6.48155   5.72441   8.06024
6.65336   5.09899   8.64416
6.75208   4.03162   5.31802
```

Выводы. Под размерностью массива понимают число индексов, которое необходимо указать для получения доступа к отдельному элементу массива. Массивы с более чем одной размерностью, называются многомерными. Самым простым многомерным массивом является двухмерный массив (матрица).

При объявлении массива указывается тип элементов массива, имя массива и его размер:

Вопросы для самоконтроля

1. Каким образом в программе на *C++* объявляется двухмерный массив?

2. Каким образом двухмерный массив располагается в памяти компьютера?

3. Возможно ли следующее объявление массивов в *C++*: ***int i, j, k, m ; int A[i][j], B[k][m]***?

4. Как в программе на *C++* будет выведен на экран двухмерный массив ***A[k][k]***?

```
for(i=0; i<k; i++)
        for(j=0; j<k; j++)
        cout<<A[i][j]<<endl;
```

5. Что нужно изменить в фрагменте программы чтобы массив ***A[k][k]*** был выведен на экран в виде строки?

```
for(i=0; i<k; i++)
        for(j=0; j<k; j++)
        cout<<A[i][j]<<endl;
```

6. Как будет выведен на экран двухмерный массив в фрагменте программы на *C++*?

```
int  i, j, k, m;
        for(i=0; i<k; i++)
```

for (j=0; j<m; j++)
cout<<A[i][j]<<" ";

7. Почему при работе с массивами в *C++* целесообразно использовать оператор цикла ***for***?

8. Какое действие выполняет оператор ***for* (*i=0*; *i*<*N*; *i*++) *B*[*i*][*j*]=*A*[*i*][*j*];**?

ЛЕКЦИЯ 10
ОПЕРАЦИИ С МНОГОМЕРНЫМИ МАССИВАМИ В *VISUAL C++* 2010

Цель лекции. Изучить способы и особенности работы с двухмерными массивами в *Visual C++* 2010.

Основные вопросы лекции

1. Поиск элемента в двухмерных массивах.
2. Сортировка элементов в двухмерных массивах.

10.1. Поиск элемента в двухмерных массивах

Поиск и сортировка двухмерных массивов в среде *Visual C++* 2010 осуществляется таким же образом, как и одномерных. Основной принцип – это сравнение элементов двухмерного массива с некоторой переменной. В качестве этой переменной может выступать либо очередной элемент двухмерного массива (матрицы), либо какая-то константа, относительно которой производится сортировка массива. Таким образом, в подобных задачах приходится использовать операторы цикла и условные операторы, с применением которых мы уже ознакомились в предыдущих лекциях.

Для двухмерных массивов справедливы все рассуждения и выкладки, приведенные в п. 8.5. Единственное отличие состоит в использовании не одного, а двух циклов для перебора всех элементов двухмерного массива.

Проиллюстрируем вышесказанное на следующем примере. Допустим, что задана матрица ***A*** из 20 произвольных чисел размера

4×5. Необходимо найти максимальный элемент матрицы ***A*** и номера его строки и столбца (индексы элемента матрицы). Элементы матрицы ***A*** введем в программе прямым образом. На экран выведем исходную матрицу ***A***, максимальный элемент и его индексы.

Блок-схема алгоритма решения данной задачи приведена на рис. 10.1.

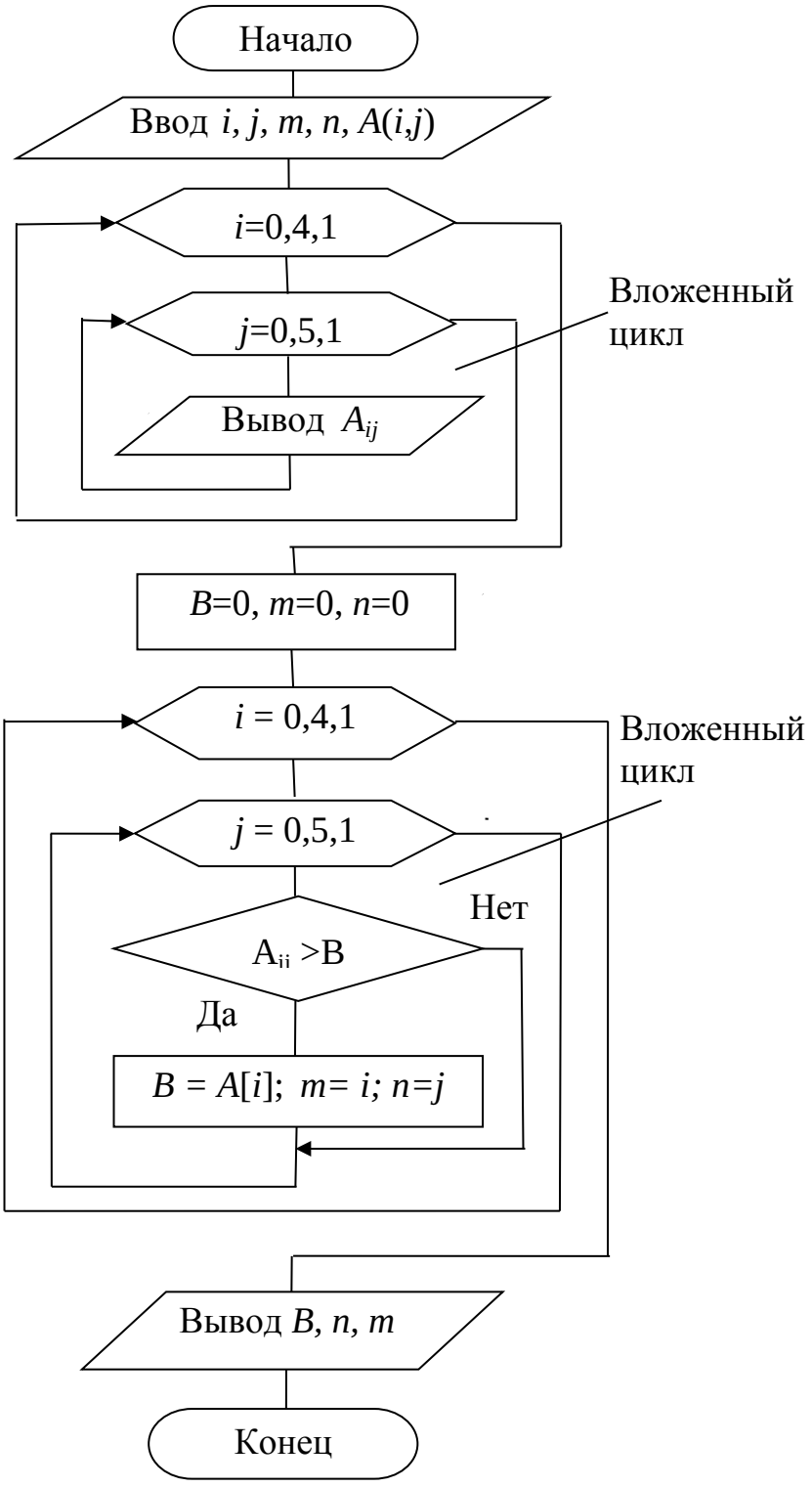


Рис. 10.1. Блок-схема алгоритма поиска максимального элемента матрицы

Поэлементный ввод матрицы ***A*** организован при помощи двух операторов цикла ***for*** – внешнего и внутреннего (вложенного). Внешний цикл организует перебор элементов матрицы ***A*** по строкам (изменяя номер строки *i*), а вложенный – по столбцам при фиксированном индексе *i*, т. е. в рамках одной строки по номеру столбца *j*.

По такому же принципу организован перебор элементов матрицы ***A*** для их сравнения с переменной *B*. Однако здесь в двойной цикл добавлен условный оператор для сравнения очередного элемента с переменной *B*, т. е. максимальным на этом шаге элементом.

Для реализации данного алгоритма разработан программный код, приведенный ниже.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;ij
int _tmain(int argc, _TCHAR* argv[])
{
int i,  j, m, n;
                                        // Объявление переменных
int A[4][5]={1,2,18,3,20,4,5,6,7,\      //Инициализация матрицы A
8,9,10,19,11,12,13,14,15,16,17};
int B;
cout<<"Matrica A"<<endl;
      for(i=0; i<4; i++)                //Внешний цикл вывода матрицы A
      {
            for(j=0; j<5; j++)          //Внутренний цикл вывода матрицы
            cout<<A[i][j]<<"\t";        //Вывод  элемента  матрицы A
      cout<<endl;
      }
B=A[0][0];                              //Подготовка к поиску максималь-
m=0; n=0;                               //ного элемента матрицы  A
      for(i=0; i<4; i++)                //Внешний  цикл по перебору
                                        //элементов матрицы A
            for(j=0; j<5; j++)          //Внутренний  цикл по перебору
                                        //элементов матрицы A
            if(A[i][j]>B)               //Выбор максимального элемента
            {
            B=A[i][j];                  //Запоминание  в B большего
```

```
                                    //элемента Aij
            n=i;                    //Запоминание в  n номера строки i
            m=j;                    //Запоминание в m номера столбца j
            }
cout<<endl<<"Max= "<<B;             //Вывод  максимального элемента
cout<<"   Stroka  "<<n+1;           //Вывод  номера строки
cout<<"   Stolbec  "<<m+1<<endl;    //Вывод   номера столбца
getch();
return 0;
}
```

Экран после выполнения программы должен иметь следующий вид:

```
Matrica A
1     2     18     3     20
4     5      6     7      8
9     10    19    11    12
13    14    15    16    17
Max= 20     Stroka  1     Stolbec  5
```

10.2. Сортировка элементов в двухмерных массивах

Применим теоретический материал, с которым вы ознакомились в подразделах 8.5 и 10.1 для сортировки двухмерных массивов.

Разработаем программу для поиска минимального элемента каждой строки некоторой заданной матрицы **A**, составления из них вектор **R** (одномерный массив) и определения суммы его четных и нечетных элементов. Элементы исходной матрицы **A** размера 4×5, состоящей из 20 произвольных чисел, введем с клавиатуры. На экран выведем исходную матрицу **A,** вектор **R** и вычисленные суммы.

Блок-схема алгоритма решения данной задачи не приводится, т. к. она аналогична блок-схеме на рис. 10.1.

Для решения задачи разработан следующий программный код:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
```

```
{
const int M=10, N=10;                  //Максимально возможное число строк и
                                       //столбцов матрицы
int A[M][N], R[M];                     //Объявление  матрицы A и вектора R
int i, j, k, m, S1, S2;                //Объявление  индексов и
                                       //вспомогательных переменных
cout<<"Vvedite chislo strok  i\
 stolbcov matricu  A :"<<endl;
cin>>m>>k;                             //Ввод реальных размеров исходной
                                       //матрицы A
cout<<"Vvedite postrochno \
elementu matricu  A :"<<endl;
      for(i=0; i<m; i++)               //Внешний  цикл вывода элементов  A
            for(j=0; j<k; j++)
            cin>>A[i][j];              //Ввод элементов исходной матрицы A
      //Поиск минимального элемента строки исходной матрицы A
      for(i=0; i<m; i++)
      {                                //Начало составного оператора в цикле
      R[i]=A[i][0];                    //Задание начального значения
                                       //минимального элемента строки
            for(j=0; j<k; j++)         //Определение минимального элемента
                  if(A[i][j]<R[i])     //каждой строки исходной матрицы A
                  R[i]=A[i][j];        //Запись его в  массив R
      }                                //Конец составного оператора в цикле
S1=0;                                  //Задание начальных значений сумм
S2=0;                                  // четных и нечетных элементов вектора R
      //Определение суммы четных и нечетных элементов массива R
            for(i=0; i<m; i++)
            {                          //Начало составного оператора в цикле
                  if(R[i]%2==0)        //Условие – есть ли остаток от деления на
                                       //2 (четное или нет)
                  S2=S2+R[i];          //Определение суммы четных элементов R
                  else S1=S1+R[i];     //Определение суммы нечетных элементов
            }                          //Конец составного оператора в цикле
cout<<"Matrica A"<<endl;
      for(i=0; i<m; i++)               //Внешний цикл вывода элементов A
      {                                //Начало составного оператора в цикле
            for(j=0; j<k; j++)
            cout<<A[i][j]<<"\t";       //Вывод элементов матрицы A
      cout<<endl;
      }                                //Конец составного оператора в цикле
cout<<endl<<"Vektor R"<<endl;
      for(i=0; i<m; i++)
```

```
        cout<<R[i]<<"   ";              //Вывод вектора R
cout<<endl;
cout<<"S1= "<<S1<<endl;                //Вывод суммы S1 четных элементов
cout<<"S2= "<<S2<<endl;                //Вывод суммы S2 нечетных элементов
getch();
return 0;
}
```

В результате выполнения вышеприведенной программы получим следующие результаты:

```
Matrica A
1   2   3   4
5   6   7   8
9   10   11   12
Matrica AT = A transponirovannaya
1  5  9
2  6  10
3  7  11
4  8  12
Matrica B = f(AT)
4.54037  4.82155  4.43915
6.48155  5.72441  8.06024
6.65336  5.09899  8.64416
6.75208  4.03162  5.31802
```

Выводы. Поиск и сортировка двухмерных массивов в среде *Visual C++* 2010 осуществляется таким же образом, как и одномерных. Основной принцип – это сравнение элементов двухмерного массива с некоторой переменной. В качестве этой переменной может выступать либо очередной элемент двухмерного массива (матрицы), либо какая-то константа, относительно которой производится сортировка массива.

Вопросы для самоконтроля

1. Какие основные принципы поиска элементов в двухмерных массивах?
2. Какие основные принципы сортировки элементов в двухмерных массивах?
3. Как будет выведен на экран двухмерный массив в фрагменте программы на *C++*?

```
int  i, j, k, m
        for(i=0; i<k; i++)
        {
                for ( j=0; j<m; j++)
                cout<<A[i][j];
                cout<<endl;
        }
```

4. Что будет выведено на экран в фрагменте программы на *C*++?

```
for(i=0; i<n; i++)
        for (j=0; j<n; j++)
        cout<<A[0][j];
```

5. Что нужно изменить в фрагменте программы для вывода массива ***A*[*k*][*m*]** на экран в виде матрицы?

```
int  i, j, k, m;
        for(i=0; i<k; i++)
                for ( j=0; j<m; j++)
                cout<<A[i][j]<<” “;
```

6. Почему при работе с массивами в *C*++ целесообразно использовать оператор цикла ***for***?

7. Какое действие выполняет оператор ***for*(*i*=0; *i*<N; *i*++) *B*[*i*][*j*]=*A*[*i*][*j*];**?

ЛЕКЦИЯ 11
ФУНКЦИИ В СРЕДЕ *VISUAL C*++ 2010. НАЗНАЧЕНИЕ, ОСНОВНЫЕ ПОНЯТИЯ И ВЫЗОВ

Цель лекции. Изучить назначение и особенности разработки и использования функций в среде *Visual C*++ 2010

Основные вопросы лекции:

1. Основные понятия, назначение и объявление функций в *Visual C*++ 2010.
2. Описание функций в *Visual C*++ 2010.
3. Вызов функций в *Visual C*++ 2010.
4. Передача аргументов функции в *Visual C*++ 2010.

11.1. Основные понятия, назначение и объявление функций в *Visual C*++ 2010

Основную часть программного кода в *C*++ составляют функции. **Функция** – это самостоятельная единица программы для решения конкретной задачи. Функции позволяют разбивать программу на отдельные автономные блоки. Любая программа содержит, по крайней мере, одну функцию (главную) - ***tmain*(…)**.

Для создания правильного кода компилятору необходимо сообщить в начале программы имя функции, тип возвращаемого результата, а также количество и типы аргументов. Для этой цели в *C*++ используется так называемый **прототип функции**. Прототип функции задается следующим образом:

ТипРезультата ИмяФункции(ТипПараметра1 [ИмяПараметра1], …);

Использование прототипа функции является **объявлением функции**. Чаще всего прототип функции совпадает с заглавием функции. В отличие от заглавия функции прототип заканчивается **точкой с запятой**.

Имена формальных параметров функции при ее объявлении не играют роли. Поэтому прототип функции может выглядеть следующим образом:

***int function*(*int a, float b, float c*);**

или

***int function*(*int, float, float*);**

Два этих объявления функции ***function*** равносильны.

11.2. Описание функций в *Visual C++* 2010

Основная форма описания или программный код функции имеет следующий вид:

Тип ИмяФункции(ТипПараметра1 ИмяПараметра1, …)
{
Тело функции
}

Описание функции состоит из заглавия функции и тела функции. Все исследованные нами выше программы имели по умолчанию такое описание главной функции:

***int _tmain*(…)**
{
Тело функции
}

В описании функции **Тип** перед ее именем определяет тип значения, которое возвращает функция. Если тип не указан, то по умолчанию предусматривается, что функция возвращает целое значение (тип *int*).

Список параметров состоит из перечня типов параметров и их имен, разделенных запятыми. Функция может не иметь параметров, но круглые скобки необходимы всегда.

В списке параметров для каждого параметра должен быть указан тип. Например, в описании *function*(*int x, int в, float z*)

правильный список параметров, а в описании ***function*(*int x,в,float z*)** – неправильный.

В теле функции обязательно должен присутствовать оператор ***return*** (возвратить) с параметром того же типа, что и возвращаемое значение. Мы постоянно сталкивались с этим оператором в конце главных функций.

Оператор *return* имеет два варианта использования:

1. Вызывает немедленный выход из функции и возвращение в программу, которая ее вызвала.

2. Используется для возвращения вычисленного значения функции.

Если возвращаемое значение не используется в дальнейшем в программе, то оператор *return* следует без параметра или вообще может быть опущен. В этом случае возвращение в программу осуществляется после достижения закрывающейся скобки "**}**".

В случае, когда оператора *return* в теле функции нет или за ним нет значения, то значение, возвращаемое функцией, неизвестно (не определено). Если функция должна возвращать значение, но не делает этого, компилятор выдает предупреждение. Все функции, которые возвращают значение, могут использоваться в выражениях языка *C*++.

Функция может вызывать другие функции (одну или несколько). А те, в свою очередь, проводить вызов третьих и т.д. Кроме того, функция может вызывать саму себя. Это явление в программировании называется **рекурсией.**

Любая программа в среде *Visual C*++ 2010 обязательно включает главную функцию ***tmain*()**. С этой функции начинается выполнение любой программы.

11.3. Вызов функций в *Visual C*++ 2010

Для того чтобы функция выполняла определенные действия в программе, она должна быть вызвана. Функция выполняется только при обращении к ней. По окончании работы функция возвращает в основную программу в качестве результата значение некоторой переменной.

Вызов функции осуществляется путем указания в программе ее имени (идентификатора), за которым в круглых скобках следует список аргументов, разделенных запятыми:

ИмяФункции(аргумент 1, аргумент 2, … аргумент *N*).

Каждый аргумент функции является переменной, выражением или константой. Они передаются в тело функции для последующего использования в вычислительном процессе. Список аргументов может быть пустым.

На рис. 11.1 схематично показано, как взаимодействует основная программа и функции в *Visual C++ 2010*.

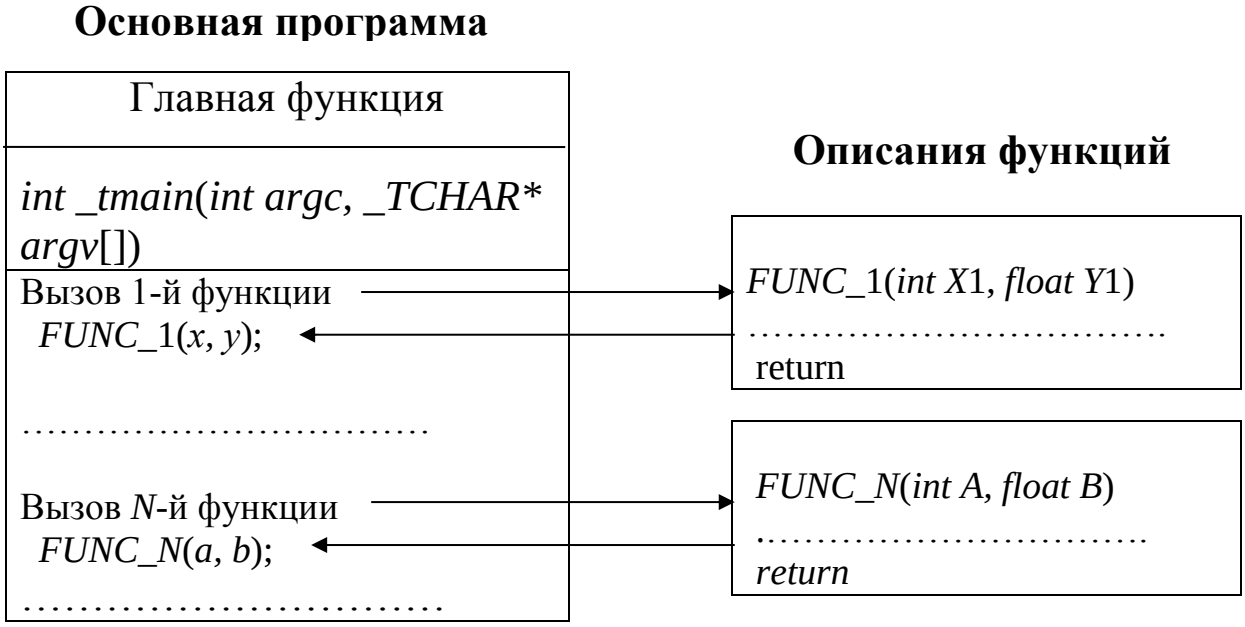


Рис. 11.1. Порядок вызова функций из основной программы

Аргументы, которые указаны в заглавии функции, носят название **формальных**, например, в ***FUNC*_1(*int X*1, *float Y*1)** формальные параметры – ***X*1** и ***Y*1.**

Аргументы, которые указаны в имени функции при ее вызове, называются фактическими. Например, при вызове ***FUNC*_1(*X*, *Y*)** фактические параметры – ***X*** и *Y*. Фактические параметры принимают конкретные значения, передающиеся формальным параметрам.

11.4. Передача аргументов функции в *Visual C++* 2010

Существует два способа передачи аргументов функции в *C++*: по значению и по ссылке.

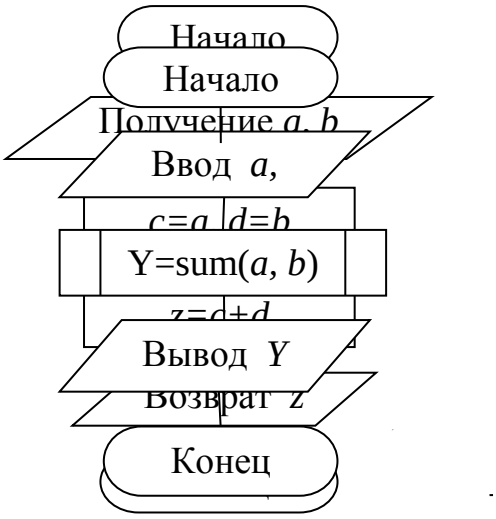
Когда происходит передача аргумента по значению, в функции создается локальная переменная с именем аргумента, в которую записывается его значение. Внутри функции может измениться значение этой переменной, но не самого аргумента.

Тип каждого фактического параметра (константы или переменной) в инструкции вызова функции должен совпадать с типом соответствующего формального параметра, указанного в объявлении функции.

Если параметр функции используется для возвращения результата, то в объявлении функции этот параметр должен быть ссылкой, а в инструкции вызова функции как фактический параметр должен быть указан адрес переменной (передача аргументов по ссылке).

Применим полученные сведения на практике. Разработаем программу, вычисляющую ***Y*** по формуле ***Y=a+b***, в которой вычисление ***Y*** реализовано в виде функции.

Блок-схема алгоритма решения данной задачи приведена на рис. 11.3.



а б

Рис. 11.3. Блок-схема алгоритма вычисления ***Y*** по формуле ***Y=a+b*** *с* использованием функции: а) основная программа; б) функция ***sum***

Программа, использующая функцию для вычисления ***Y*** по формуле ***Y=a+b***, будет иметь следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
double sum(double c, double d);          //Объявление (прототип) ф-ии sum
int _tmain(…)
{
double Y, a, b;
cout<<"Vvedite a,  b :"<<endl;
cin>>a>>b;                               //Ввод чисел a и b
Y=sum(a, b);                             //Вызов функции, вычисляющей
                                         //сумму двух чисел
cout<<endl<<"Y = "<<Y<<endl;             //Вывод значения Y
getch();
return 0;
}

double sum(double c, double d)           //Описание функции sum
{
double z;
z=c+d;                                   //Вычисление  z=c+d
return z;                                //Возврат значения  z  в основную
                                         //программу
}
```

Результат выполнения программы следующий:

```
Vvedite a,  b :
5.4 4.3
Y=9.8
```

Выводы. Основную часть программного кода в *C++* составляют функции. Функция – это самостоятельная единица программы для решения конкретной задачи. Функции позволяют разбивать программу на отдельные автономные блоки. Любая программа содержит, по крайней мере, одну функцию (главную).

Для создания правильного кода компилятору необходимо сообщить в начале программы имя функции, тип возвращаемого

результата, а также количество и типы аргументов. Для этой цели в *C++* используется так называемый прототип функции.

Вызов функции осуществляется путем указания в программе ее имени (идентификатора), за которым в круглых скобках следует список аргументов, разделенных запятыми.

Вопросы для самоконтроля

1. В чем должны совпадать при обращении к функции реальные и формальные параметры?
2. Как осуществляется обращение к функции?
3. Могут ли быть функции без параметров?
4. В программе используются две независимые функции. В каком порядке они должны быть описаны в разделе прототипов?
5. В массиве ***A*** необходимо определить максимальный элемент и его номер. Возможно ли этот блок оформить функцией?
6. В программе описано две функции. Возможно ли в них использовать одинаковые переменные?
7. В программе описана функция ***int f(int x)*** . Возможно ли так обратится к этой функции: ***Y = f(n)/(f(m)*f(n-m));***?
8. Укажите правильный прототип функции ***Fun***, использующей в качестве аргумента целый массив ***D[n]***.
   1) ***double Fun (double B[], int n);***
   2) ***double Fun (int D[], int n);***
   3) ***Fun (double B[], int n);***
9. Укажите правильное обращение к функции ***Fun***, использующей в качестве аргумента целый массив ***D[n]***.
   1) ***func(int D[], n);***
   2) ***Fun (D, n);***
   3) ***func (int V[][]);***
10. Когда функция не возвращает никакого значения, можно ли ее использовать в выражениях языка *C++*?
11. Где могут быть объявлены переменные, выступающие параметрами функции?
12. Верна ли запись заглавия функции ***Fun*** **(x, int в, float z)**?

ЛЕКЦИЯ 12
ФУНКЦИИ В СРЕДЕ *VISUAL C++* 2010. ОБЛАСТИ ДЕЙСТВИЯ ПЕРЕМЕННЫХ

Цель лекции. Изучить области действия и видимости переменных в программах с функциями. Изучить особенности использования функций в работе с массивами в *Visual C++* 2010

Основные вопросы лекции:

1. Области действия и видимости переменных в программах в среде *Visual C++* 2010.
2. Функции и массивы в *Visual C++* 2010

12.1. Области действия и видимости переменных в программах в среде *Visual C++* 2010

Область действия переменной – это часть или части программного кода, в которых данные переменные **определены** (доступны для действий с ними в данном месте программы).

С точки зрения области действия переменных различают три типа переменных:

– локальные;

– глобальные;

– формальные.

Локальные переменные – это переменные, объявленные в середине блока, в частности, внутри описания функции. Локальная переменная доступна в середине блока или внутри функции, в которых она объявлена. Область действия локальной переменной – данный блок или функция.

Формальные переменные (параметры) – это переменные, объявленные при описании функции как ее аргументы. Формальные параметры используются в теле функции, как локальные переменные. Область действия формальных параметров – тело функции.

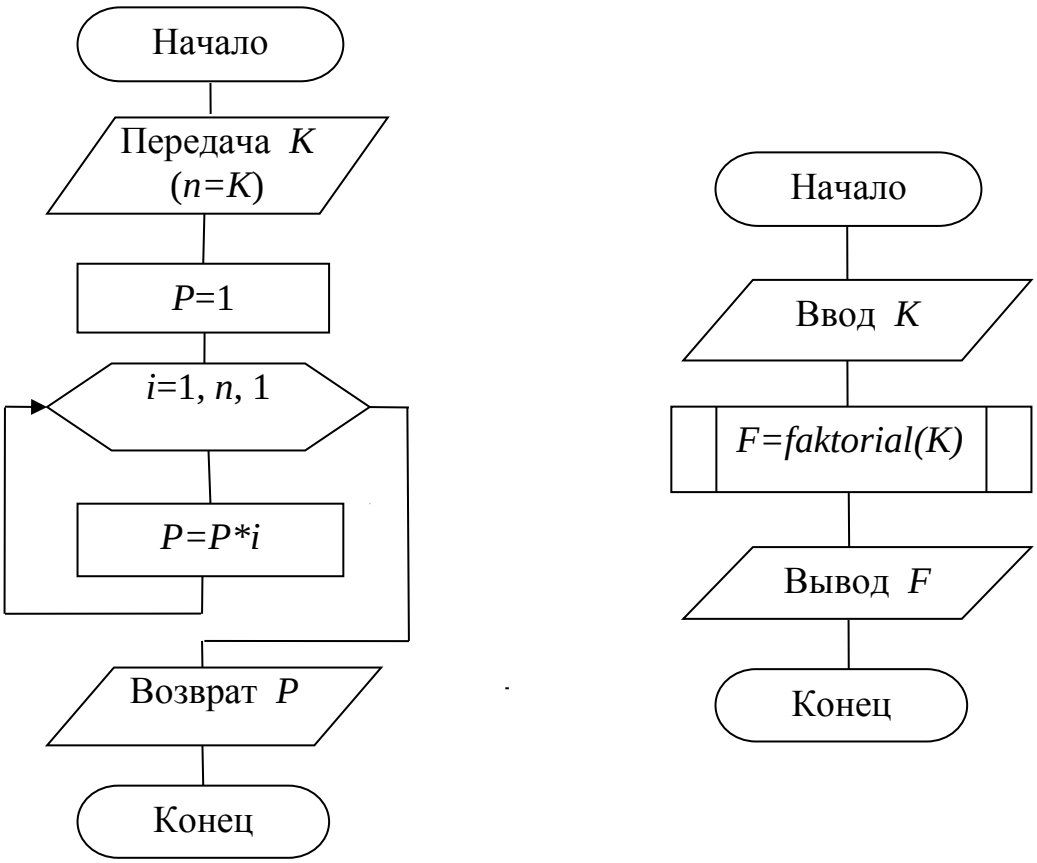
Глобальные переменные – это переменные, объявленные в основной программе вне какой-либо функции. Они могут быть

использованы в любом месте программы. Область действия глобальной переменной – вся программа.

В качестве примера использования при решенни инженерных задач в среде *Visual C++* 2010 функций вообще и функций с локальными параметрами, в частности, разработаем программу, содержащую функцию вычисления факториала числа. Факториал целого числа ***n*** рассчитывается по формуле

$$F = n! = 1*2*3*4* \ldots *n = \prod_{i=1}^{n} i;$$

Блок-схема алгоритма решения данной задачи приведена на рис. 12.1.



а б

Рис. 12.1. Блок-схема алгоритма вычисления факториала числа с использованием функции:
а) функция ***faktorial*(*int n*)**; б) основная программа

Процедуру расчета факториала числа *n* оформим в виде функции, в которой будет вычисляться произведение по

вышеприведенной формуле. Внутри этой функции будут использоваться локальные переменные ***i*** и ***P***, которые вместе с переменной ***n*** позволят рассчитать факториал $P = \prod_{i=1}^{n} i$; Переменная ***n*** будет передаваться в функцию из основной программы как формальный параметр при вызове последней. Назад в основную программу будет передаваться вычисленное значение факториала в виде переменной ***P***.

Программа для функции, вычисляющей факториал числа **n**, будет иметь следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int faktorial(int n);                  //Объявление (прототип) функции
                                       //вычисления факториала
int _tmain(…)
{                                      //Начало  основной программы
int K;
cout<<"Vvedite K :"<<endl;
cin>>K;                                //Ввод числа K
F=faktorial(K);                        //Вызов функции, вычисляющей
                                       //факториал числа
cout<<endl<<"Faktorial K= "<<K \       //Вывод  факториала числа K
<<endl;
getch();
return 0;
}                                      //Конец основной программы
int faktorial(int  n)                  //Описание функции факториала
{                                      //Начало описания функции
int i, P;                              //Объявление локальных переменн.
P=1;
for(i=1; i<=n; i++)                    //Цикл для вычисления  факториала
P=P*i;
return P;                              //Возврат P в основную программу
}                                      //Конец описания функции
```

Результат выполнения программы следующий:

Vvedite K :
5
Faktorial K= 120

В разработанной программе все переменные локальные, т. е., например, переменные *i* и *P* из функции *faktorial* для основной программы остаются невидимыми.

12.2. Функции и массивы в *Visual C*++ 2010

Если в качестве аргумента функции используется массив, то необходимо указать адрес начала массива и его размер.

Заглавие функции, обрабатывающей массив, необходимо записать следующим образом:

***float function*(*float A*[*n*]);**

или

***float function*(*float A*[ ], *int n*),**

где ***A*[*n*]** – массив размерностью ***n***.

В этом случае вызов функции ***function*** из основной программы запишется, например, следующим образом:

***function*(B, k)**,

где ***B*[*k*]** – массив размерностью ***k***.

Сучетом вышесказанного разработаем программу для решения следующей задачи.

Задан массив ***A*** из ***n*** произвольных чисел. Необходимо сформировать новый массив ***B***, каждый элемент которого равняется частному от деления соответствующего элемента массива ***A*** на его максимальный элемент. На экран вывести начальный массив ***A***, его максимальный элемент и массив ***B***. Поиск максимального элемента массива ***A*** оформить функцией.

Запрограммируем функцию нахождения максимального элемента массива и его номера и используем ее в разрабатываемой программе следующим образом:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
double Max(double B[],int n);        //Объявление (прототип)  функции
                                     //поиска максимального  элемента
int _tmain(…)
{
const int N=50;                      //Максимальный размер исходного
массива
int i, k;                            //Объявление параметра  цикла и
реального размера массива
double A[N], M;                      //Объявление массива A и переменной  M
cout<<"Vvedite razmer \
massiva :"<<endl;
cin>>k;                              //Ввод реального размера массива
cout<<" Vvedite massiv:"<<endl;
        for(i=0; i<k; i++)
        cin>>A[i];                   //Ввод исходного массива A
        cout<<endl;
        M=Max(A, k);                 //Обращение к функции поиска  Max()
cout<<" Ishodnui massiv "<<endl;
        for(i=0; i<k; i++)           //Цикл для печати  исходного массива A
        cout<<A[i]<<' ' ;
cout<<endl<<"Max= "<<M<<endl;//Печать максимального  элемента
                                     //исходного  массива
cout<<" Novui massiv "<<endl;
        for(i=0; i<k; i++)           //Цикл для расчета и печати элементов
нового  массива
        cout<<A[i]/M<<' ';
        cout<<endl;
        getch();
        return 0;
}
double Max(double B[],int n)         //Описание функции поиска максималь-
                                     //ного элемента массива
{
int j;
double C=B[0];                       //Присваивание переменной C начального
                                     //значения (1-го элемента)
        for(j=0; j<n; j++)           //Цикл для перебора элементов массива и
                                     //сравнения их с C
        if (B[j]>C) C=B[j];
```

```
return C;                    //Возвращение в основную программу знач-я  C
}
```

После выполнения программы экран будет иметь следующий вид:

```
Vvedite razmer massiva :
7
 Vvedite massiv:
1 6 0 3 7 9 5

 Ishodnui massiv
1 6 0 3 7 9 5
Max= 9
 Novui massiv
0.111111 0.666667 0 0.333333 0.777778 1 0.555556
```

Вывод. Область действия переменной – это часть или части программного кода, в которых данные переменные **определены** (доступны для действий с ними в данном месте программы).

С точки зрения области действия переменных различают три типа переменных:

– локальные;

– глобальные;

– формальные.

Локальные переменные – это переменные, объявленные в середине блока, в частности, внутри описания функции. Локальная переменная доступна в середине блока или внутри функции, в которых она объявлена. Область действия локальной переменной – данный блок или функция.

Формальные переменные (параметры) – это переменные, объявленные при описании функции как ее аргументы. Формальные параметры используются в теле функции, как локальные переменные. Область действия формальных параметров – тело функции.

Глобальные переменные – это переменные, объявленные в основной программе вне какой-либо функции.

Вопросы для самоконтроля

1. Какова роль оператора ***return*** **в** функции?

2. Как осуществляется выход из функции, когда в теле функции отсутствует оператор ***return***?

3. Что такое локальная переменная?

4. В чем должны совпадать при обращении к функции реальные и формальные параметры?

5. Как осуществляется обращение к функции?

6. В массиве ***A*** необходимо определить максимальный элемент и его номер. Возможно ли этот блок оформить функцией?

7. В программе описано две функции. Возможно ли в них использовать одинаковые переменные?

8. В программе описана функция ***int f(int x)*** . Возможно ли так обратится к этой функции: ***r = f(n)/(f(m)*f(n-m));***?

9. Укажите правильный прототип функции ***FF***, использующей в качестве аргумента целый массив ***W[n]***.

1) ***double FF(double B[], int n);***
2) ***double FF(int W[], int n);***
3) ***FF(double B[], int n);***

10. Укажите правильное обращение к функции ***FF***, использующей в качестве аргумента целый массив ***W[n]***.

1) ***func(int W[], n);***
2) ***FF(W, n);***
3) ***func(int W[][]);***

11. Когда функция не возвращает никакого значения, можно ли ее использовать в выражениях языка *C++*?

12. Где могут быть объявлены переменные, выступающие параметрами функции?

13. Верна ли запись заглавия функции ***func(x, int в, float z)***?

ЛЕКЦИЯ 13
АДРЕСАЦИЯ ПЕРЕМЕННЫХ И УКАЗАТЕЛИ В СРЕДЕ *VISUAL C++* 2010

Цель лекции. Изучить адресацию памяти и исследовать особенности применения указателей в *Visual C++* 2010.

Основные вопросы лекции.

1. Понятие адреса переменной в *Visual C++* 2010.
2. Понятие указателя в *Visual C++* 2010.
3. Разыменование указателей в *Visual C++* 2010.
4. Операции с указателями в *Visual C++* 2010.
5. Указатели и массивы в *Visual C++* 2010.

13.1. Понятие адреса переменной в *Visual C++* 2010

Переменная занимает в памяти компьютера определенную область (набор ячеек). Расположение переменной в памяти, т. е. данный набор ячеек, определяется адресом. При объявлении переменной для нее резервируется место в памяти. Размер зарезервированной памяти зависит от типа данной переменной.

Для доступа к содержимому выделенной памяти служит его **имя** (идентификатор). Для того чтобы узнать адрес конкретной переменной, применяется операция **взятия адреса**. Синтаксис операции следующий:

&ИмяПеременной,

т. е. перед именем переменной ставится знак **&**.

Рассмотрим программу, в которой используется операция взятия адреса для двух переменных ***A*** и ***B***:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
```

```
{
        double A=23.1;                          //Инициализация переменных
        double B=57.88;                         //A и B
        cout<<"Znachenie A= "<<A<<endl;         //Вывод на экран значения A
        cout<<"Adres A= "<<&A<<endl;            //Вывод на экран адреса A
        cout<<"Znachenie B= "<<B<<endl;         //Вывод на экран значения B
        cout<<"Adres B= "<<&B<<endl;            //Вывод на экран адреса B
getch();
return 0;
}
```

В результате выполнения такой программы получим следующие результаты:

```
Znachenie A= 23.1
Adres A= 002EF800
Znachenie B= 57.88
Adres B= 002EF7F0
```

При анализе программ по этой теме следует учитывать, что адреса переменных на разных компьютерах будут отличаться.

Адреса переменных записаны в шестнадцатиричной системе счисления. Признаком 16-ричного числа являются символы **0*x*.**

13.2. Понятие указателя в *Visual C++* 2010

В языке *C++* есть возможность осуществлять непосредственный доступ к ячейкам памяти. Для этого предусмотрен специальный тип переменных – указатели.

Указатель – это переменная, содержащая адрес некоторого объекта. Объектом может быть переменная или функция. В общем случае указатель – это целое число.

Если переменная будет указателем, то она должна быть объявлена в программе. Указатель в программе объявляется следующим образом:

ТипОбъекта *Идентификатор.*

Здесь **ТипОбъекта** определяет тип данных, на которые ссылается указатель с именем **Идентификатор**. Символ “*”

(звездочка) означает, что следующая за ней переменная является указателем. При объявлении указателя под него резервируется 4 байта.

Примеры объявления указателей:

Char *A
int *temp, i, *z;
double f, *ptr;

Здесь объявлены указатели ***ch***, ***temp***, ***z***, ***ptr*** и переменные ***i*** и ***f***.

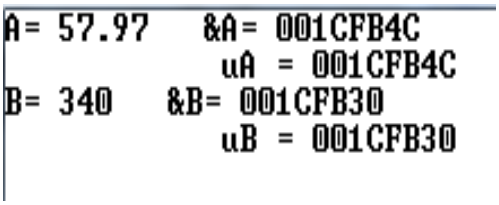
Поскольку указатель является ссылкой на некоторую область памяти, ему может быть присвоенj только значение **адреса переменной**, а не значение самой переменной.

Рассмотрим пример объявления и инициализации указателя.

Разработаем программу, в которой инициализируются указатели ***uA*** и ***uB*** переменных ***A*** и ***B*** с присвоением им значений адресов этих переменных.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
double A=57.97;
double *uA=&A;                    //Инициализация указателя uA и
                                  //присваивание ему адреса A
double B=340;
double *uB;                       //Инициализация указателя uB
uB =&B;                           //Присваивание указателю uB
                                  //значения адреса B
cout<<"A= "<<A<<\                 //Вывод переменной A и адреса A
"   &A= "<<&A<<endl;
cout<<"        uA = "<<uA<<endl;  //Вывод указателя uA
cout<<"B= "<<B<<\                 //Вывод переменной B и адреса B
"   &B= "<<&B<<endl;
cout<<"        uB = "<< uB <<endl;//Вывод указателя uB
getch();
return 0;
}
```

Результат выполнения программы следующий:



Очевидно, что значение указателя совпадает со значением адреса соответствующей переменной.

13.3. Разыменование указателей

Указатели помогают осуществлять непосредственный доступ к памяти. Для того чтобы получить (прочитать) значение, записанное по адресу, который находится в указателе, используют операцию непрямого обращения или **разыменования** (*). Для этого используется имя указателя со звездочкой перед ним.

Проверим это в программе, в которой разыменуются указатели ***uA*** и ***uB***, т. е. определяются значения переменных ***A*** и ***B*** по значениям указателей на эти переменные:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
double A=57.97;
double *uA=&A;                    //Инициализация указателя uA и
                                  //присваивание ему адреса A

double B=340;
double *uB;                       //Объявление указателя uB
uB =&B;                           //Присваивание указателю uB
                                  //значения адреса B

cout<<"A= "<<A<<\
"   &A= "<<&A<<endl;
cout<<"        uA = "<< uA <<endl;//Вывод указателя uA
cout<<"* uA =  "<<* uA <<endl;    //Разыменование  указателя uA и
вывод результата
cout<<"B= "<<B<<"   &B= "<<&B<<endl;
cout<<"        uB = "<< uB <<endl;//Вывод указателя uB
```

```
cout<<"* uB = "<<* uB <<endl;          //Разыменование указателя uB и
                                       //вывод результата
getch();
return 0;
}
```

После выполнения программы на экран будет выведена следующая информация:

```
A= 57.97   &A= 0026FD68
           uA = 0026FD68
* uA =  57.97
B= 340   &B= 0026FD4C
           uB = 0026FD4C
* uB = 340
```

Очевидно, что в результате разыменования указателей можно получить значения переменных, адреса которых совпадают со значениями самих указателей.

13.4. Операции с указателями

Язык *C++* позволяет работать с указателями также, как и с переменными стандартных типов. Однако операции над указателями отличаются некоторыми особенностями.

С указателями можно выполнять следующие операции: разадресация (*), присваивание, сложение с константой, вычитание,инкремент (++), декремент (- -), сравнение, приведение типов. При работе с указателями часто используется операция получения адреса (&).

13.4.1. Операция присваивания указателей

Указатели одного и того же типа могут использоваться в операциях присваивания, как и другие любые переменные.

Ниже приведена программа, в которой применяются операции присваивания указателей ***px*** и ***g***, а ***g*** затем разыменуется для проверки, совпадает ли значение ****g*** со значением переменной ***x***.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int x=10;
int *px, *g;
px=&x;
g=px;
cout<<"px= "<<px<<endl;
cout<<"g= "<<g<<endl;
cout<<"x= "<<x<<endl <<  " *g= "<<*g<<endl;
getch();
return 0;
}
```

После выполнения программы на экран будет выдана следующая информация:

```
px= 002FFB74
g= 002FFB74
x= 10
 *g= 10
```

Очевидно, что операция присваивания между указателями и последующее разыменование указателя ***g*** не внесло погрешностей и значение разыменованного указателя ***g*** совпадает со значением переменной ***x***.

13.4.2. Операция суммирования указателей

Как и над другими типами данных, над указателями можно выполнять арифметические операции: сложение и вычитание. Но арифметические операции над указателями имеют свои особенности. К таким особенностям выполнения операций относится следующее.

Общая формула для вычисления значения указателя после выполнения операции ***P = P + n*** будет иметь вид:

$$P = P + n*k;$$

где k – количество байт памяти базового типа указателя.

Другой операцией является операция вычитания

$$P2 - P1 .$$

Это вычисление числа элементов между указателями.

Операции инкремента (++) и декремента (--) являются частным случаем сложения и вычитания.

Разработаем программу для иллюстраци этих положений:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int *p;
int x;
p=&x;
cout<<”    p= “<<p<<endl;
cout<<”++p= “<<++p<<endl;
}
```

Вид экрана после выполнения программы:

```
    p= 0x0012FF78
++p= 0x0012FF7C
```

После выполнения этой программы мы видим, что при операции ***++p*** значение указателя ***p*** увеличилось не на 1, а на 4. Это верно, поскольку новое значение указателя должно указывать не следующий адрес памяти, а адрес следующего целого.

Другие арифметические операции над указателями невозможны.

13.4.3. Операция сравнения указателей

Указатели можно сравнивать, применяя все 6 операций сравнение.

Сравнение ***p<g***, например, означает, что адрес, находящийся в ***p***, меньше адреса, находящегося в ***g***.

13.5. Указатели и массивы

В языке *C++* принято, что имя массива – это адрес ячейки памяти, начиная с которой располагается массив. Другими словами, имя массива – это адрес первого (нулевого) элемента массива.

Если объявлен массив

***int A*[12];**

то **Mas** является указателем на массив, точнее на первый элемент массива.

Таким образом**,**

***pA = A*[0];**

Для того чтобы получить значение 8-го элемента массива **A**, необходимо задать

***A*[8]=*(*A*+7).**

В *C++* **n**-й элемент массива определяется как

***A*[*n*] =*(*A*[0]+*n*).**

Выводы

В языке *C++* есть возможность осуществлять непосредственный доступ к памяти. Для этого предусмотрен специальный тип переменных – указатели. Указатель – это переменная, содержащая адрес некоторого объекта.

Поскольку указатель является ссылкой на некоторую область памяти, ему может быть присвоен только адрес переменной, а не ее значение.

Указатели помогают осуществлять непосредственный доступ к памяти. Для того чтобы получить (прочитать) значение, записанное по адресу, который находится в указателе, используют операцию

непрямого обращения или разыменования. Для этого используется имя указателя со звездочкой перед ним.

Вопросы для самоконтроля

1. Что будет выведено на экран после выполнения программы ***int A*=300; *cout*<<&*A*;**?
2. Возможна ли следующая инициализация указателя?

```
char A=”yes”;
char *pA=&A;
```

3. Что означает оператор ***double *pA=&A;***?
4. Что такое указатель в языке *C*++?
5. Как объявляется указатель с именем ***pA*** в языке *C*++?
6. Как обозначается операция взятия адреса переменной ***A*** в языке *C*++?
7. Как обозначается операция разыменования указателя ***pA*** в языке *C*++?
8. Что означает операция разыменования указателя в языке *C*++?
9. Какое значение примет ***Y*** в программе?

```
double  X=10.1;
double *pA;
pA=&X;
Y=*pA;
```

10. Можно ли в языке *C*++ выполнять арифметические операции над указателями?
11. Что будет выведено на экран дисплея после выполнения программы?

```
int *pA;
for (i=0; i<100; i++)
out<<*(pA+i)<<” “;
```

12. Возможно ли так объявить указатель ***int **ppA;***?
13. На сколько байтов изменится значение ***pC*** в программе?

```
int C[20];
int *pC=&C;
pC++;
```

ЛЕКЦИЯ 14
МАССИВЫ СИМВОЛЬНЫХ ПЕРЕМЕННЫХ (СТРОКИ) В *VISUAL C++* 2010

Цель лекции. Исследовать особенности обработки массивов символьных переменных **(**строк) в среде *Visual C++* 2010.

Основные вопросы лекции

1. Описание строк в среде *Visual C++* 2010.
2. Ввод и вывод строк в среде *Visual C++* 2010.
3. Функции обработки строк в среде *Visual C++* 2010.
4. Строки в среде *Visual C++* 2010.

14.1. Описание строк в среде *Visual C++* 2010

В языке *C++* массив символьных переменных или символьная строка - это одномерный массив, состоящий из символов и обозначающийся типом ***char***:

***char A*[11].**

Символьная строка – это последовательность символов, дополненная специальным символом-ограничителем, указывающим конец строки. Ограничивающий символ записывается управляющей последовательностью **“\0”**. Для такой символьной строки применяют название “строка *C*” (была предложена разработчиком языка). В других ветвях языка *C++* существуют другие представления символьных строк.

Каждый символ в строке занимает один байт.

Символьная константа **“\0”** , ограничивающая символьную строку, называется нулевым байтом. Ее следует учитывать при определении соответствующего массива символов: если строка должна содержать *N* символов, то в определении массива следует указать *N*+1 элемент.

Например, определение

***char A*[11];**

означает, что строка содержит 10 элементов типа ***char*** (символов), а последний байт зарезервирован для нулевого байта.

В качестве символов могут использоваться.

1. Прописные буквы латинского и российского алфавитов.
2. Строчные буквы латинского и российского алфавитов.
3. Цифры от 0 до 9.
4. Символы пунктуации.
5. Символьные константы.
6. Управляющие символы.
7. Пробел.
8. Шестнадцатеричные цифры.

Символьные массивы при их определении могут инициализироваться как обычный массив:

***char A*[13]={'*K*','*h*','*a*','*r*','*k*','*o*','*v*','-','2','0','1','4'}.**

а могут – как символьная строка. Символьная строка – это последовательность символов, заключенных в кавычки:

***char A*[13]="Kharkov-2014".**

Отличие этих двух способов заключается в том, что во втором случае автоматически будет прибавлен еще и нулевой байт.

Для выделения места в памяти под символьный массив произвольного размера необходимо указать количество символов в строке (если оно известно) или задать явно больший размер массива:

***char B*[80]= "Это инициализация массива символов".**

В данном случае указан размер массива 80 (с запасом), хотя для размещения этой строки можно было указать 35 (с учетом нулевого байта).

Инициализировать символьный массив можно и без указания его размера:

***char B*[ ]= “Это инициализация массива символов”.**

В этом случае компилятор *Visual C++* 2010 сам определит необходимый размер памяти под этот массив.

14.2. Ввод и вывод строк в среде *Visual C++* 2010

При вводе символов с пробелами, последние игнорируются операторами ввода “*cin*>>” и вывода “*cout*<<”. Поэтому при работе со строками вместо этих операторов целесообразней использовать функцию ***getline***:

***getline*(ИмяСимвольнойПеременной,**
РазмерСимвольнойПеременной),

где **ИмяПеременной** указывает на строку, в которую осуществляется ввод; **РазмерПеременной** – число символов, подлежащих вводу.

В качестве примера использования операторов потоковых ввода и вывода символьных массивов разработаем следующую программу:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
char stroka[30], A[50];                  //Объявление символьных
                                         //переменных
cout<<"Vvedite ctroku  <\                //Вывод на экран приглашения
30  simvolov:"<<endl;
cin>>stroka;                             //Ввод символьной переменной
                                         //stroka
cout<<"Vu vveli stroku:"<<endl;
cout<<stroka<<endl;                      //Вывод символьной
                                         //переменной  stroka
```

```
cout<<"Vvedite novuyu ctroku\
< 30  simvolov::"<<endl;
cin>>stroka;                          //Ввод новой символьной
                                      //переменной  stroka
cout<<"Vu vveli novuyu stroku:"\
<<stroka<<endl;                       //Вывод символьной
                                      //переменной  stroka
cout<<"Vvedite novuyu ctroku\
< 50  simvolov::"<<endl;
cin>>A;                              //Ввод символьной переменной A
cout<<"Vu vveli novuyu \
stroku: "<<A<<endl;                   //Вывод символьной
                                      //переменной  A
cout<<"A0= "<<A[0]<<endl;             //Вывод 0-го элемента
                                      //переменной  A
cout<<"A8= "<<A[8]<<endl;             //Вывод 8-го элемента
                                      переменной  A
getch();
return 0;
}
```

В приведенной программе с клавиатуры вводятся две символьные переменные – ***stroka*** и ***A***, после чего эти символьные переменные выводятся на экран как целиком, так и в виде отдельных элементов. После выполнения программы экран будет иметь следующий вид:

```
Vvedite ctroku  <  30  simvolov:
wwwwwwwwwwwqqqqqqqqq
Vu vveli stroku:
wwwwwwwwwwwqqqqqqqqq
Vvedite novuyu ctroku  <  30  simvolov::
aaaaaddddddffff
Vu vveli novuyu stroku: aaaaaddddddffff
Vvedite novuyu ctroku  <  50  simvolov::
sssssddddgggg
Vu vveli novuyu stroku: sssssddddgggg
A0= s
A8= g
```

Если при вводе символьных массивов ***stroka*** и ***A*** вводить символы с пробелами, то можно убедиться, что используемый оператор ввода *cin*>> не видит этих пробелов. Поэтому при работе со строками вместо этого оператора целесообразно использовать функцию ***getline***.

Реализуем предыдущую программу с применением оператора ***getline***:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
char stroka[70], A[50];          //Объявление символьных переменных
cout<<"Vvedite ctroku \
< 30 simvolov:"<<endl;
cin.getline(stroka,20);          //Ввод символьной переменной stroka
cout<<"Vu vveli stroku:"<<endl;

cout<<stroka<<endl;              //Вывод символьной переменной stroka
cout<<"Vvedite novuyu \
ctroku < 30 simvolov:"<<endl;
cin.getline(A,20);               //Ввод символьной переменной A
cout<<"Vu vveli novuyu \
stroku: "<<endl<<A;              //Вывод символьной переменной A
getch();
return 0;
}
```

После выполнения программы экран будет иметь вид:

```
Vvedite ctroku  <  30  simvolov:
www sss ttttt
Vu vveli stroku:
www sss ttttt
Vvedite novuyu ctroku  <  30  simvolov:
qqq yyyyyyy ppppppppp
Vu vveli novuyu stroku:
qqq yyyyyyy ppppppppp
```

Очевидно, что функция ***getline*** обеспечивает ввод и выводсимвольных переменных с пробелами.

При использовании функции ***getline*()** **РазмерПеременной** должен быть меньше или равен размеру объявленной символьной строки.

Объявленная в вышеприведенной программе строка ***stroka*** может принять 70 символов. Например, если в функции ***getline*(*stroka*, 20)** указано число 20, то при вводе строки с 37 символами введется строка из 20 символов. Остальные символы будут отброшены.

14.3. Функции обработки строк в среде *Visual C*++ 2010

Для работы со строками существуют специальные функции, описание которых находится в заглавном файле ***string.h***, который необходимо включать в программу оператором *include*:

***#include*<*string.h*>.**

Рассмотрим функции, которые используются наиболее часто.

14.3.1. Определение длины строки

Часто при работе со строками необходимо знать, сколько символов содержит строка. Для получения информации о длине строки используется функция ***strlen*()**. Вызов функции имеет вид:

***strlen*(ИмяСимвольнойПеременной);**

Функция возвращает значение на единицу меньшее, чем отводится под массив (без учета нулевого байта).

Разработаем программу, в которой будет использоваться функция ***strlen*().** Эта программа приведена ниже.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
char A[80];
int k;
cout<<"Vvedite ctroku  <\
30  simvolov:"<<endl;
cin.getline(A,30);                    //Вызов функции getline() для ввода
                                      //массива A
cout<<"Vu vveli stroku: "<<endl<<A;   //Вывод символьной переменной  A
k=strlen(A);                          //Вызов функции strlen(A)
cout<<endl<<"k= "<<k<<endl;           //Вывод переменной k (количество
                                      //символов в A)
getch();
return 0;
}
```

Вид экрана после работы программы:

```
Vvedite ctroku  <  30  simvolov:
wwwwwwwww  RRRRRRRRR  xxxxxxxxxx
Vu vveli stroku:
wwwwwwwww  RRRRRRRRR  xxxxxxx
k= 29
```

Анализ результатов подтверждает, что функция ***strlen*** определяет количество символов в символьной строке. Следует учитывать, что пробелы также входят в это количество.

14.3.2. Копирование строк

Значения строк могут копироваться из одной строки в другую. Копирование осуществляется с помощью следующих функций.

Функция ***strcpy*(*S*1,*S*2)** используется для побайтного копирования строки *S*2 в строку *S*1. Копирование прекращается при достижении нулевого байта. Поэтому длина строки *S*1 должна быть достаточно большой, чтобы в нее поместилась строка *S*2.

Разработаем программу, в которой будет использоваться функция ***strcpy*()**. Эта программа приведена ниже.

```
#include <string.h>                    //Добавление библиотечного
                                       //файла для работы со строками
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
char A[80];
int k;
cout<<"Vvedite ctroku\
< 0  simvolov:"<<endl;
cin.getline(A,30);                     //Ввод символьной переменной  A
cout<<"Vu vveli stroku: "<<endl<<A;
strcpy(A, "Proverka kopirovaniya");    //Вызов функции strcpy(A) для
                                       //копирования строки в строку
cout<<endl<<"Novaya stroka: "<<A;      //Вывод новой символьной
                                       //переменной  A
getch();
return 0;
}
```

Вид экрана после работы программы:

```
Vvedite ctroku  <  30  simvolov:
aaaaaaaaaaaaaaa ddddddd
Vu vveli stroku:
aaaaaaaaaaaaaaa ddddddd
Novaya stroka: Proverka kopirovaniya_
```

Очевидно, что функция ***strcpy*** скопировала строку "***Proverka kopirovaniya***" в исходную строку ***A***.

Функция ***ctrncpy*()** отличается от функции ***ctrcpy*()** тем, что включает еще один параметр, указывающий количество символов, которые необходимо копировать из строки *S*2 в строку *S*1. Функция имеет вид:

***ctrncpy*(*S*1, *S*2, *n*)**,

где *n* – количество символов (целое без знака).

Если длина *S*1 меньше длины *S*2, то при копировании происходит урезание символов строки *S*1.

Пранализируем использование функции *strncpy*():

```
#include <string.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      char A[]="0123456789";          //Ввод символьной переменной  A
      char B[]="qwertyuiop";          //Ввод символьной переменной  B
      cout<<"S2= "<<A<<endl;          //Вывод символьной переменной  A
      cout<<"S1= "<<B<<endl;          //Вывод символьной переменной  B
      strncpy(B,A,4);
      cout<<"S2new= "<<B<<endl;       //Вывод новой символьной
переменной  B
getch();
return 0;
}
```

Вид экрана после работы программы:

```
S1= 0123456789
S2= qwertyuiop
S1new= 0123456789
S2new= 0123tyuiop
```

То есть, из строки *S*2 в строку *S*1 будут скопированы 4 первых символа и размещены в начале короткой строки *S*2.

14.3.3. Присоединение строк

Присоединение (**конкатенация**) строк используется для образования новой строки символов из двух и более исходных строк. Для этой цели используются функции

***strcat*(*S*1, *S*2)** и ***strncat*(*S*1, *S*2, *n*)**.

Функция ***strcat*(*S*1, *S*2)** присоединяет строку *S*2 к строке *S*1 и помещает ее в массив, где находилась строка *S*1. Строка *S*2 не изменяется. Вновь полученная строка *S*1 автоматически завершается нулевым байтом.

Разработаем программу для использования функции *strcat*():

```
#include <string.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;

int _tmain(…)
{
char A[30], B[30];
strcpy(A, "Hello, ");                  //Копирование строки "Hello, " в строку A
strcpy(B, "World!");                   //Копирование строки "World!" в стр. B
cout<<"A= "<<A<<endl;
cout<<"B= "<<B<<endl;
strcat(A, B);                          //Присоединение строки B к строке A
cout<<endl<<"A= "<<A<<endl;
cout<<"B= "<<B<<endl;
getch();
return 0;
}
```

Результат выполнения программы следующий:

```
A= Hello,
B= World!

A= Hello, World!
B= World!
```

Функция ***strncat*(*S*1, *S*2, *n*)** также осуществляет присоединение строк, однако присоединяет лишь указанное в третьем параметре количество символов, например:

```
char S1[80]="Dlya prodolgeniya ";
char S2[80]="nagat knopku OK !";
strncat(S1,S2,7);
cout<<S1<<endl;
```

В результате на экран будет выведена строка:

Dlya prodolgeniya nagat knopku OK !

14.3.4. Сравнение строк

В библиотеке функций *string.h* есть функции, выполняющие посимвольное сравнение двух строк.

Функция *strcmp*(*S*1, S2) сравнивает строки *S*1 и *S*2. После сравнения строк данная функция возвращает одно из следующих значений:

<0 – если строка *S*1 меньше чем *S*2;
=0 – если строки эквивалентные;
>0 – если *S*1 больше, чем *S*2.

Эта функция проводит сравнение строк, различая прописные и строчные буквы, например:

```
char S1[]="ааааааааавв";
char S2[]="БББББББББГГ";
int к;
k= strcmp(S1,S2);
cout<<"k= "<<k<<endl;
```

Переменной *k* будет присвоено негативное значение (-1), несмотря на равное количество символов в строках. Строка *S*1 меньше строки *S*2 по той причине, что прописные буквы имеют код символов меньше чем те же строчные буквы.

На экран будет выведено:

k= -1

Функция ***stricmp*(*S*1, S2)** сравнивает строки *S*1 и *S*2**,** не различая регистра символов.

Функция ***strnimp*(*S1, S2, n*)** проводит сравнение определенного числа (***n***) первых символов двух строк. Регистр символов при этом учитывается.

Кроме рассмотренных функций есть большое количество других функций обработки строк.

14.4. Строчные переменные в *Visual C*++ 2010

Язык *C*++ позволяет объявлять и применять строчные переменные, т.е. переменные, которые содержат строки:

***string A*.**

В данном случае строчная переменная ***A*** инициализирована пустой строкой.

Строчную переменную можно инициализировать строчным литералом, используя оператор

***string A*= "*Kharkov*-2014".**

Впоследствии, переменной **A** можно присвоить другую строку, используя оператор присвоения:

***A* = "*Visual C*++ 2010".**

Эта строка складывается из 16 символов. В этом случае говорят, что длина строки ***A*** равна 16. Для вычисления текущей длины строки можно применять или функцию length(), или функцию *size*().

Ссылаться на отдельные символы строки можно с помощью индексов, как будто строка является массивом. Таким образом, в предыдущем примере ячейка *A*[0] содержит символ **“*V*”**, а ячейка *A*[10] - символ **“+”**.

Строки можно сравнивать, используя обычные операторы сравнения. Причем, можно проверять не только равенство, но и какая из строк предшествует другой.

Строки можно **конкатенировать** (присоединять), используя оператор «+». В итоге образуется новая строка, состоящая из двух частей: первой и второй строк, записанных последовательно.

Например, если в программе поместить объявление

***string B* = "*Com*";**

то операторы ***string C=B+"puter";*** и **B+= "puter"** присвоят переменным *C* и *B* строку "*Computer*".

Выводы

В языке *C++* строка представляется как одномерный массив, элементы которого имеют тип ***char***, заканчивающийся нулевым байтом. Со строками можно производить различные действия – копировать, присоединять, определять их длину и т. д.

Язык *C++* позволяет объявлять и применять строчные переменные, т.е. переменные, которые содержат строки. При этом строки можно сравнивать, используя обычные операторы сравнения. Причем, можно проверять не только равенство, но и какая из строк предшествует другой. Строки также можно конкатенировать (присоединять), используя оператор «+».

Вопросы для самоконтроля

1. Сколько символов содержит массив ***char A*[11] в** программе на *C++*?
2. Какая функция в языке *C++* используется для копирования строк?
3. Что значит в языке *C++* конкатенация строк?
4. Что такое символьная строка?
5. Что такое нулевой байт?
6. Как инициализируется символьный массив?
7. Как объявляется символьный массив?
8. Для чего применяется функция ***getline*()**?
9. Какие аргументы используются в функции ***getline*()**?
10. Для чего применяется функция ***strcpy*()**?
11. Какие аргументы используются в функции ***strcpy*()**?
12. Для чего применяется функция ***strcat*()**?
13. Какие аргументы используются в функции ***strcat*()**?
14. Для чего применяется функция ***strncat*()**?
15. Какие аргументы используются в функции ***strncat*()**?

ЛЕКЦИЯ 15
ИСПОЛЬЗОВАНИЕ ФАЙЛОВ ДЛЯ ВВОДА И ВЫВОДА ДАННЫХ В *VISUAL C++* 2010

Цель лекции. Изучить особенности использования файлов для ввода, хранения и вывода данных в *Visual C++*.

Основные вопросы лекции

1. Файлы и потоки ввода и вывода данных в *Visual C++* 2010.
2. Создание, открытие и закрытие файлов в *Visual C++* 2010.

15.1. Файлы и потоки ввода и вывода данных в Visual C++

При решении большинства задач на любом языке программирования возникает необходимость записывать, хранить и получать информацию, используя файлы.

Файл (*file*) – это именованный объект, хранящий данные (программа или любая другая информация) на каком-либо носителе (винчестер, флешь-память, *CD* и др.).

Поток (*stream*) - это виртуальное логическое устройство, связывающее программу с физическим устройством ввода-вывода (терминалом, дисководом и др.). Поскольку потоки не зависят от физических устройств, то одна и та же функция может записывать информацию на диск или на другое устройство.

Поток связывают с определенным файлом, выполняя операцию **открытия**. Как только файл открыт, можно проводить обмен информацией между ним и программой.

Файл отсоединяется от определенного потока (т.е. разрывается связь между файлом и потоком) с помощью операции **закрытия.** При закрытии файла, открытого с целью вывода, содержимое (если оно есть) связанного с ним потока записывается на внешнее устройство. Этот процесс, который обычно называют дозаписью потока, гарантирует, что никакая информация случайно не останется в буфере диска. Если программа завершает работу нормально, то все файлы закрываются автоматически. В случае аварийного завершения программы, файлы не закрываются.

В языке *C++* существует два типа потоков:

- текстовый (*text*);
- двоичный (*binary*).

Текстовый поток – это последовательность символов. В ***C++*** считается, что текстовый поток организован в виде строк, каждая из которых заканчивается символом новой строки. Среди символов в потоке может быть символ возврата каретки, перехода на новую строку и др.

Двоичный поток – это последовательность байтов, однозначно соответствующих информации, находящейся на внешнем носителе. Причем никакого преобразования символов не происходит.

Доступ к информации в потоках и в файлах неодинаков. Например, из файла на диске можно выбрать 3-ю запись или заменить 6-ю запись. В то же время в файл, связанный с печатью, информация может передаваться только последовательно. Это иллюстрирует самое главное отличие между потоками и файлами.

Потоки, использующиеся в программах, делятся на:

- входные, из которых читается информация;
- выходные, в которые вводится информация;
- двунаправленные, допускающие как чтение, так и запись.

В соответствии с особенностями «устройства», к которому «присоединен» поток, потоки принято разделять на:

- стандартные;
- консольные;
- строчные;
- файловые.

Стандартные и консольные потоки соответствуют передаче данных от клавиатуры до дисплея.

Если символы потока в совокупности образуют символьный массив в основной памяти, то это **строчный поток.**

Если информация размещается на внешнем носителе данных, то это **файловый поток** или просто **файл.**

До сих пор во всех программах курса выполнялся обмен со стандартными потоками:

cin – стандартный входной поток, связанный с клавиатурой;

cout – стандартный выходной поток, связанный с экраном дисплея.

Используя операции включения (записи) данных в поток “<<” и извлечения данных из потока “>>” выполнялся обмен данными с дисплеем и с клавиатурой ПК. Для этих целей в программу необходимо было включить заголовочный файл *iostream.h*.

Рассмотрим особенности обмена данными с файлами.

15.2. Создание, открытие и закрытие файлов в *Visual C++*

Библиотека ввода и вывода данных в файлы подключается в заголовочном файле *stdio.h* и включает средства для работы с последовательными файлами. **Последовательный файл** можно представить как именованную цепочку (ленту, строку) байтов, имеющую начало и конец. Последовательный файл отличается от файла с другой организацией тем свойством, что чтение из файла (или запись в него) ведется байт за байтом от начала до конца.

В каждый момент времени позиция в файле, из которого выполняется чтение (или запись), определяется значениями **указателя позиции** записи и чтения файла.

Установка указателя записи (чтения) на нужные байты выполняется либо автоматически, либо за счет управления их положением. В библиотеке ввода-вывода есть соответствующие средства.

При работе с файлами используются следующие операции:

- создание файла;
- удаление файла;
- поиск файла на внешнем носителе;
- открытие файла;
- чтение из файла или запись данных в файл;
- позиционирование файла;
- закрытие файла.

Все перечисленные действия могут быть выполнены с помощью средств библиотеки ввода-вывода.

Операция **открытия файла** связывает поток с определенным файлом. Операция **закрытия** файла разрывает эту связь.

Запись или чтение из файла осуществляются с помощью указателя файла. **Указатель файла** – это указатель на структуру

типа ***File***. Для объявления переменной–указателя файла, например, ****fp***, используется следующий оператор:

File *fp.

Указатель файла указывает на структуру, содержащую различные сведения о файле, его имя, статус и указатель текущей позициив начало файла

При обработке данных, хранящихся в файле, программе необходимо иметь доступ к данному файлу. С помощью переменной ***fp** ведется в дальнейшем вся работа с файлом в программе.

***File *F*1;**

В файле **stdio.h** определены функции для работы с файлами. Основные из них приведены в таблице 15.1.

Таблица 15.1

Функции для работы с файлами в *Visual C*++ 2010

Функция	Описание функции
fopen()	Открыть файл
fclose()	Закрыть файл
fseek()	Переместить (установить) указатель позиции файла
feof()	Возвращает значение «истина», если достигнут конец файла
ferror()	Возвращает значение «ложь», если найдена ошибка
fread()	Читает блок данных из потока.
fwrite()	Записывает блок данных в поток
rewind()	Устанавливает указатель позиции файла на начало
remove()	Удаляет файл

При открытии файла функция *fopen*() выполняет два действия:
- открывает файл и связывает его с потоком;

- возвращает указатель, ассоциируемый с этим файлом.

Прототип функции *fopen*() имеет следующий вид:

File *fopen* (*char *filename, char mode*),

где ***char *filename*** – строка, содержащая полное имя файла на диске; ***char *mode*** – строка, определяющая режим открываемого файла.

Возможны следующие режимы открытия файла:

“***r***” – открыть файл для чтения;

“***w***” – создать файл для записи;

“***a***” – открыть для добавления в существующий файл;

“***rb***” – открыть двоичный файл для чтения;

“***wt***” – создать текстовый файл для записи;

“***at***” – открыть текстовый файл для добавления;

“***r+t***” – открыть текстовый файл для чтения и записи;

“***w+t***” – создать текстовый файл для чтения и записи;

“***a+t***” – открыть текстовый файл для добавления.

Например, для создания файла для записи данных с именем ***C*:*Inform.dat*** необходимо записать:

```
File *F1;
F1=fopen(“C:\\Inform.dat”, “w”).
```

При открытии файла всегда необходимо провести проверку открытия файла.

Применим полученные сведения для решения реальной задачи.

Разработаем программу для записи переменной ***A*** в открываемый файл, чтения значения ***A*** из этого файла и вывода значения ***A*** на печать.

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
```

```
{
double A, B, S;
double *pS;                          //Указатель на буфер обмена
pS=&S;                               //Присваивание адреса S указателю pS
cout<<"Vvedite A"<<endl;
cin>>A;                              //Ввод A
cout<<"A= "<<A<<endl;
File*F1;                             //Объявление файла для записи
if((F1=fopen("D:\\Inform.DAT", \     //Стандартная процедура создания  и
"w"))==NULL)                         //открытия файла для записи данных  с
{                                    //одновременной проверкой результата
cout<<" No open file "<<endl;//этих действий и сообщением в случае
exit(1);                             //ошибки
S=A;                                 //Запись A в буфер
fwrite(pS,4,1,F1);                   //Запись из буфера в открытый файл
fclose(F1);                          //Обязательное закрытие файла
fread(pS,4,1,F1);                    //Чтение переменной pS из файла
B=S;                                 //Запись из буфера в переменную B
cout<<"B= "<<B;
getch();
return 0;
}
```

В данной программе используется метод определения ошибки при открытии создаваемого файла. Неоткрытие файла приравнивается к константе *NULL*, которая определена в библиотеке *stdio.h*. Функция *exit*(1) определена в файле *stdlib.h* и прекращает выполнение программы, а единицу возвращает в операционную систему. Перед прекращением программы она закрывает все открытые файлы, освобождает буферы и выводит все необходимые сообщения на экран.

Если файл открывается для записи, то существующий файл удаляется и создается новый.

При открытии файла для чтения, требуется, чтобы он существовал.

В случае открытия файла для чтения и записи существующий файл не уничтожается, однако создается, если он не существовал ранее.

После чтения данных из файла (или записи данных в файл) он должен быть закрыт следующим образом:

***fclose*(*F*1).**

Выводы

При решении большинства задач на любом языке программирования возникает необходимость записывать, хранить и получать информацию, используя файлы.

Файл – это именованный объект, хранящий данные (программа или любая другая информация) на каком-либо носителе (винчестер, флешь-память, *CD* и др.).

Поток - это виртуальное логическое устройство, связывающее программу с физическим устройством ввода-вывода (терминалом, дисководом и др.). Поскольку потоки не зависят от физических устройств, то одна и та же функция может записывать информацию на диск или на другое устройство.

Поток связывают с определенным файлом, выполняя операцию открытия. Как только файл открыт, можно проводить обмен информацией между ним и программой.

Вопросы для самоконтроля

1. Как объявляется ввод данных в файл?
2. Как объявляется операция вывода данных из файла?
3. Какой функцией определяется конец файла при чтении данных из файла?
4. Когда функция ***eof*()** возвращает значение 0?
5. Для определения конца файла записан оператор ***while*(!*name_file.eof*())**. Сколько шагов будет выполняться цикл?
6. Какой функцией закрывается файл при завершении работы с ним?
7. Какая функция используется при чтении массивов или структур из файла?
8. Из какого файла будет прочитана информация при выполнении оператора ***ifstream datbook*(“*datbook.dat*”);**?
9. Для чего используется функция ***write*()**?
10. Для чего используется функция ***rewind*()**?

ЛЕКЦИЯ 16
ИСПОЛЬЗОВАНИЕ ФАЙЛОВ ДЛЯ ВВОДА И ВЫВОДА ДАННЫХ В *VISUAL C++* 2010

Цель лекции. Изучить особенности использования файлов для ввода, хранения и вывода данных в *Visual C++*.

Основные вопросы лекции

1. Запись данных в файл и чтение из файла в *Visual C++* 2010.
2. Позиционирование файла в *Visual C++* 2010.

16.1. Запись (чтение) данных в файл в *Visual C++* 2010

Запись данных в поток проводится функцией ***fwrite*()** с прототипом

***unsigned fread*(*void *buf, int bytes, int c, File *fptr*)**,

где ***buf*** – указатель на буфер памяти, откуда будет происходить обмен с файлом:

bytes – длина каждой единицы записи (чтение) в байтах;

c – количество единиц записи, которое будет прочитано (записано);

fptr – указатель на соответствующий файл.

Чтение данных в поток проводится функцией ***fread*()** с прототипом с такими же атрибутами:

***unsigned fwrite*(*void *buf, int bytes, int c, File *fptr*)**.

Разработаем программу для записи в файл ***D*:*Inform.dat*** массив из 12 произвольных чисел. Также необходимо прочитать этот массив из файла и вывести его на экран дисплея.

Программа записи массива в файл, чтение массива из файла и отображения его на экране дисплея будет иметь следующий вид:

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(…)
{
float S;                          //Объявление буфера обмена
float *pS;                        //Объявление указателя на буфер
float A[12];                      //Объявление массива
int i;
pS=&S;                            //Присваивание адреса буфера
                                  //указателю на буфер
cout<<"Vvedite   A:"<<endl;
for(i=0; i<12; i++)
cin>> A[i];                       //Ввод массива
File *F1;                         //Объявление указателя  файла F1
      if((F1=fopen("C:\\Inform.dat",\
      "w+b"))==NULL)
      {
      cout<<"No open file"<<endl;
      exit(1);
      }
      for(i=0; i<12; i++)         //Цикл для  чтения данных из файла
      {
      S=A[i];                     //Запись элемента массива в буфер
      fwrite (pS, 4, 1, F1);      //Запись порции из 4-х байтов в файл
      }
fclose(F1);                       //Закрытие файла F1
int l;
float A[12];                      //Объявление массива A
File *F2;                         //Объявление указателя файла F2
      if((F2=fopen("C:\\Inform.dat",\  //Открытие файла C:\\Inform.dat в
      "r+b"))==NULL)              //режиме  чтения и записи
      {
      cout<<" No open file "<<endl;
      exit(1);
      }
i=0;
      while(i<12)                 //Цикл для чтения данных из файла
      {
      fread(pS, 4, 1, F2);        //Чтение одной порции из 4-х байтов
```

```
                                        //в буфер S
        A[i]=S;                         //Запись из буфера S  в массив
        i+=1;                           //Подсчет  количества  прочитанных
чисел из файла
        }
l=i;                                    //Определение количества прочитан-
                                        //ных чисел
fclose(F2);                             //Закрытие файла F1
cout<<endl<<"From file:";
for(i=0; i<l; i++) cout<< A[i]<<'  ';   //Вывод массива на экран
cout<<endl;
getch();
return 0;
}
```

Вид экрана после выполнения программы:

```
Vvedite   A:
1 2 3 4 5 6 7 8 9 10 11 12
From file:
1 2 3 4 5 6 7 8 9 10 11 12
```

При чтении данных из файла размер файла часто неизвестен. Для определения конца файла служит функция *feof*(). Она имеет следующий прототип:

***int feof*(*File *fptr*)**.

Функция возвращает значение «истина», если достигнут конец файла и «ложь» - если нет.

Программа чтения данных из файла *C*:*Inform.dat* с использованием функции *feof*() имеет следующий вид:

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
float S;                                //Объявление буфера обмена
```

```
float *pS;                          //Объявление указателя буфера
float A[9];                         //Объявление массива
int i;
pS=&S;                              //Присвоение адреса буфера
                                    //указателю буфера

cout<<"Vvedite  A:"<<endl;
for(i=0; i<9; i++) cin>>A[i];       //Ввод массива
        File *F1;                   //Объявление указателя на файл F1
        if((F1=fopen("C:\\Inform.dat", \
        "w+b"))==NULL)              //Открытие файла C:\\Inform.dat в
                                    //режиме  записи

        {
        cout<<" No open file "<<endl;
        exit(1);
        }
        for(i=0; i<9; i++)
        {
        S= A[i];                    //Запись элемента массива в буфер
        fwrite(pS, 4, 1, F1);       //Запись порции из 4-х байтов в файл
        }
fclose(F1);                         //Закрытие файла F1
int l;
float A[9];                         //Объявление массива A
File *F2;                           //Объявление указателя файла F2
        if((F2=fopen("C:\\Inform.dat",\
        "r+b"))==NULL)              //Открытие файла C:\\Inform.dat в
                                    //режиме  чтения и записи

        {
cout<<" No open file "<<endl;
exit(1);
        }
i=0;
        while(!feof(F2))            //чтение данных из файла пока не
                                    //наступит конец файла

        {
        fread(pS, 4, 1, F2);        //чтение одной порции из 4-х байтов
                                    //в буфер S
        Mas1[i]=S;                  //запись из буфера S в массив
        i+=1;                       //счет количества прочитанных
                                    //чисел из файла

        }
l=i-1;                              //определение количества прочитан-
                                    //ных чисел
```

```
fclose(F2);                          //закрытие файла F2
cout<<endl;
for(i=0; i<l; i++) cout<<A[i]<<' ';  //вывод массива на экран
cout<<endl;
getch();
return 0;
}
```

Вид экрана после выполнения программы:

```
Vvedite  A :
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
```

16.2. Позиционирование файла

Функция ***rewind*()** устанавливает указатель позиции файла на начало файла. Прототип этой функции имеет следующий вид:

***void rewind*(*FILE *fptr*)**.

Обращение к функции имеет вид:

***rewind* (*F*2)**.

Чтение и запись в файл не обязательно делать последовательно. Можно получить доступ непосредственно к нужному байту. Для этих целей используется функция ***fseek*()**, устанавливающая указатель позиции в нужное место.

Прототип этой функции имеет вид:

***int fseek*(*FILE *fptr, long num, int mode*)**,

где *fptr* – указатель на соответствующий файл; *num* – количество байт от точки отсчета для установки текущей позиции указателя файла; *mode* – определяет точку отсчета (см. таб. 16.1).

Таблица 16.2

Значения *mode* для различных точек отсчета

Точка отсчета	Значение *mode*
Начало файла	0
Текущая позиция	1
Конец файла	2

Применим теорию позиционирования файлов на практике. Разработаем программу для решения следующей задачи. В файле *C*:*Inform.dat* записано 9 чисел: 1, 2, 3, 4, 5, 6, 7, 8, 9. Необходимо прочитать данные из этого файла, начиная с 3-го числа, определить сумму прочитанных чисел и добавить ее в файл *C*:*Inform.dat*. Затем прочитать полученный массив из файла и вывести его на экран дисплея.

Программа будет иметь вид:

```
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
const int n=30;                    //Максимальний размер массива
float S *pS;                       //Объявление буфера обмена и
                                   //Указателя на буфер
float A[n], Sum;                   //Объявление массива и суммы
int i,l;
pS=&S;                             //Присваивание адреса буфера
                                   //указателю буфера
FILE *F2;                          //Объявление указателя файла F2

if((F2=fopen("C:\\Inform.dat", \
"r+b"))==NULL)                     //Открытие файла C:\\Inform.dat в
                                   //режиме чтения и записи
{
      cout<<" No open file "<<endl;
      exit(1);
}
```

```
i=0;
fseek(F2, 8, 0);                    //Установка указателя позиции
                                    //файла через 8 байт с начала файла
while(!feof(F2))                    //Чтение данных из файла, пока не
                                    //наступит конец файла
{
        fread(pS,  4, 1, F2);       //Чтение одной порции из 4-х байтов
                                    //в буфер S
        A[i]=S;                     //Запись из буфера S в массив
        i+=1;                      //Счет количества прочитанных чисел
}
l=i-1;                              //Определение количества
                                    //прочитанных чисел
fclose(F2);                         // Закрытие файла
cout<<endl;
for(i=0; i<l; i++) cout<<A[i]<<'  ';    //Вывод массива на экран
cout<<endl;
Sum=0;
for(i=0; i<l; i++) Sum=Sum+A[i];    //Вычисление суммы
cout<<"Sum= "<<Sum<<endl;           //Отображение суммы на экране
File *F3;                           //Объявление указателя файла  F3
F3=fopen("C:\\Inform.dat", "a");    //Открытие файла C:\\Inform.dat в
                                    //режиме добавления
S=Sum;                              //Пересылаем сумму в буфер
fwrite(pS, 4, 1, F3);               //Запись суммы из буфера в файл
fclose(F3);                         //Закрытие файла F3
FILE *F4;                           //Объявление указателя файла  F4
if((F4=fopen("C:\ \Inform.dat", \   //Открытие файла C:\\Inform.dat в
"r+b"))==NULL)                      //Режиме чтения и записи
{
cout<<" No open file "<<endl;
exit(1);
}
i=0;
while(!feof(F4))                    //Чтение данных из файла, пока не
                                    //наступит конец файла
{
        fread(pS,  4, 1, F4);       //Чтение одной порции из 4-х байтов
                                    //в буфер S
        A[i]=S;                     //Запись из буфера S в массив
        i+=1;                       //Счет количества прочитанных
                                    //чисел из файла
}
```

```
l=i-1;                                  //Счет количества прочитанных
                                        //чисел из файла
fclose(F4);                             //Закрытие файла F4
cout<<endl;
for(i=0; i<l; i++) cout<<A[i]<<'  ';    //Вывод массива на экран
cout<<endl;
getch();
return 0;
}
```

Вид экрана после выполнения программы:

3 4 5 6 7 8 9

Sum= 42

1 2 3 4 5 6 7 8 9 42

Выводы

При усложнении программ возникает необходимость хранить и получать информацию, используя файлы.

Операции ввода-вывода в языке *C++* организованы с помощью библиотечных функций.

При работе с файлами используются следующие процедуры:

- создание файлов;
- удаление файлов;
- поиск файлов на внешнем носителе;
- открытие файла;
- чтение из файла или запись данных в файл;
- позиционирование файла;
- закрытие файла.

Все перечисленные действия могут быть выполнены с помощью средств библиотеки ввода-вывода.

Вопросы для самоконтроля

1. Как объявляется ввод данных в файл?
2. Как объявляется операция вывода данных из файла?
3. Какой функцией определяется конец файла при чтении данных из файла?

4. Когда функция ***eof*()** возвращает значение 0?

5. Для определения конца файла записан оператор ***while*(!*name_file.eof*())**. Сколько шагов будет выполняться цикл?

6. Какой функцией закрывается файл при завершении работы с ним?

7. Какая функция используется при чтении массивов или структур из файла?

8. Из какого файла будет прочитана информация при выполнении оператора ***ifstream datbook*(“*datbook.dat*”);**?

9. Для чего используется функция ***write*()**?

10. Для чего используется функция ***rewind*()**?

Література

1. Хортон А. *Visual C++* 2010. Полный курс. Киев: Диалектика, 2011. – 1205 с.
2. Пахомов Б.И. *C/C++* и *Ms Visual C++* 2010. – СпБ.: БХВ-Петербург, 2011. – 722 с.
3. Онуфрей Ю.Є., Подоляка О.О., Тімонін В.О. Обчислювальна техніка та програмування. Розділ 2. Мова програмування *C* (*C++*). Конспект лекцій. – Харків: ХНАДУ, 2010. – 163 с.

Навчальне видання

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни “Інформаційні технології”

за напрямом підготовки 6.050702 “Електромеханіка”

(Розділ “Мова програмування С++”)

Укладач: Симбірський Г. Д.

Відповідальний за випуск *О. Я. Ніконов*

Підписано до друку
Формат 60×84 1/16. Папір офсетний. Гарнітура Times New Roman.
Друк RISO. Умовн. друк. Арк. . Обл.-вид. арк. .
Замовлення № . Тираж прим. Ціна договірна.

Видавництво ХНАДУ, 61002, м. Харків-МСП, вул. Петровського, 25

Свідоцтво державного комітету інформаційної політики, телебачення та радіомовлення України про внесення суб’єкта видавничої справи до державного реєстру видавництв, виготівників і розповсюджувачів видавничої продукції, серія ДК №897 від 17.04.2002 р.