МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний автомобільно-дорожній університет

Симбірський Г.Д.

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни “Програмування”
(розділ “Мова програмування С++”)

Для студентів напряму підготовки 6.050201 “Системна інженерія”

Харьков – 2015

УДК 004.4(075.8)
ББК 22.18я7

Симбірський Г.Д. Конспект лекцій з дисципліни “ Програмування” за напрямом підготовки 6.050201 “Системна інженерія” (розділ “Мова програмування *C*++”). – Харків: ХНАДУ, 2015, - 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

Лекция 1
Алгоритмизация вычислительных задач

Цель лекции. Изучение основных понятий и конструкций схем алгоритмов.

Вступление

Алгоритмический язык *C++* - это один з основных языков программирования, который называется объектно-ориентированным языком. Язык программирования *C++* будет изучаться нами в составе программного пакета *Microsoft Visual Studio* 2010 как *Visual C++* 2010. Язык *C++* позволяет решать множество задач – от простых школьных заданий до сложнейших задач ядерной физики и космических исследований.

В данном курсе будет изучаться синтаксис, семантика и техника программирования на языке *C++*. Приведено большое количество программ, иллюстрирующих возможности и особенности языка *C++*.

Конспект лекций предназначен для получения студентами в ходе изучения учебной дисциплины «Программирование» теоретических навыков применения языка *C++* при решении прикладных задач на компьютере, а также для использования в качестве вспомогательной литературы при курсовом и дипломном проектировании на старших курсах.

Основные вопросы лекции

1. Основные понятия и определения при алгоритмизации задач.
2. Простые операции и их базовые конструкции.
3. Составные операции и их базовые конструкции
4. Виды алгоритмов.

1. Основные понятия и определения при алгоритмизации задач

Для решения любой задачи на компьютере необходимо выполнить следующие этапы:

1. Разработать математическую модель задачи, т. е. дать математическое описание объекта исследований.
2. Выбрать или разработать **метод решения** задачи.
3. Составить а**лгоритм** решения задачи.
4. На одном из языков программирования разработать **программу** для решения задачи.
5. Добиться выполнения программы, устраняя возможные ошибки.
6. Проанализировать полученный результат и сделать выводы.

В данной лекции остановимся подробнее на составлении алгоритма.

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

Программа – это записанный на языке программирования алгоритм решения задачи.

Каждый язык программирования или алгоритмический язык является набором символов и слов, с помощью которых программист записывает команды (инструкции) для компьютера. Затем программа на языке программирования переводится соответствующей программой (компилятором) в машинный код для дальнейшего выполнения компьютером.

Любой разрабатываемый алгоритм должен удовлетворять следующим требованиям:

1. **Определенность** – алгоритм не должен допускать неоднозначность толкования.

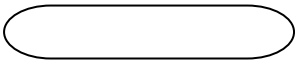
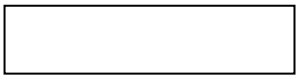
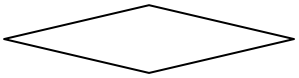
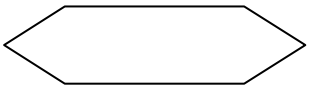
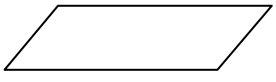
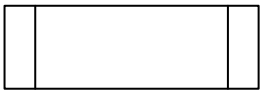
2. **Массовость** – возможность использования алгоритма при решении подобных задач.

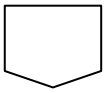
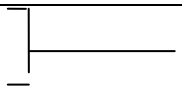
3. **Результативность** – выполнение действий в соответствии с разработанным алгоритмом должно приводить к получению искомого результата.

В программировании принято алгоритм решения задачи изображать в графическом виде. При этом все операции или действия изображаются в виде отдельных блоков. Каждое действие, например, ввод данных, печать документа и др., имеет свое стандартное условное обозначение (см. таб. 1.1). Конфигурация и размер блоков определяются соответствующими стандартами.

Последовательность действий, необходимых для решения поставленной задачи, изображенная в виде набора стандартных операционных блоков, называется **блок-схемой алгоритма**.

Таблица 1.1 Основные операционные блоки схем алгоритмов

№ п/п	Условное обозначение	Наименование	Описание операции
1		Начало, завершение	Начало и завершение алгоритма
2		Процесс	Вычислительная операция или их совокупность
3		Решение	Проверка условия и выбор дальнейшего направления процесса решения
4		Модификация	Заголовок цикла, проверка условий цикла
5		Данные	Ввод исходных данных, вывод данных и результатов
6		Типовой процесс	Использование ранее созданных алгорит-мов, 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

В блок-схемах алгоритмов операционные блоки соединяются друг с другом линиями потока. Линии потока, направленные вниз и направо могут быть без стрелок, указывающих направление. Линии потока, направленные вверх или влево обязательно должны заканчиваться стрелками. Базовые конструкции разделяются на простые и составные.

2. Простые операции и их базовые конструкции

Простое действие – это одна операция. Основные простые действия следующие:

- присваивание;
- ввод;
- вывод.

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

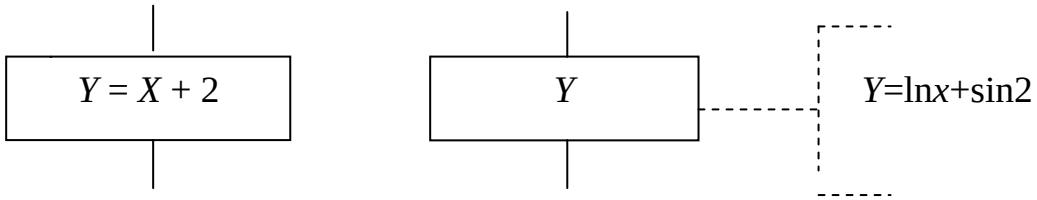


Рис. 1.1. Операционный блок **Присваивание**

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

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

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

Операционный блок ввода или вывода в схемах алгоритмов обозначается символом “данные” (параллелограмм) (рис. 1.2). В середине символа слово **Ввод**

или **Вывод** и в круглых скобках имя переменной или просто перечисляются имена переменных, которые должны быть введены.

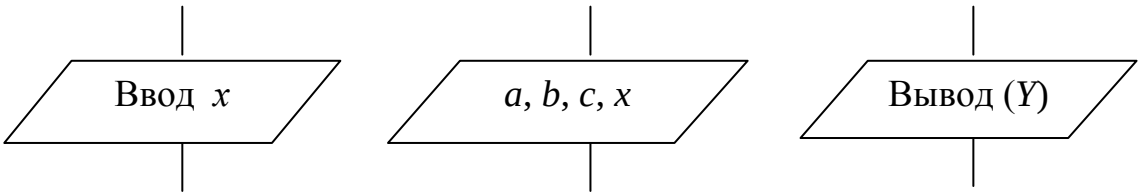


Рис. 1.2. Операционный блок **Ввод** (**Вывод**)

Исследуем процесс разработки алгоритма простейшей программы 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).

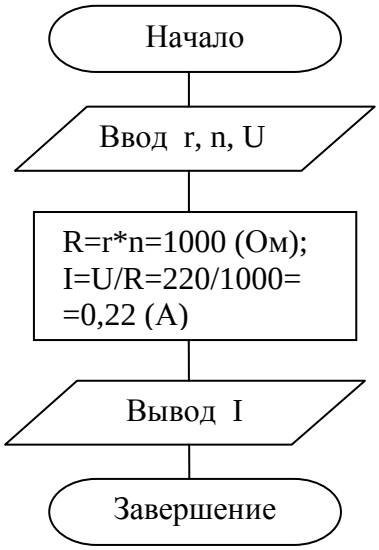


Рис. 1.3. Блок-схема алгоритма определения силы тока
в новогодней электрогирлянде

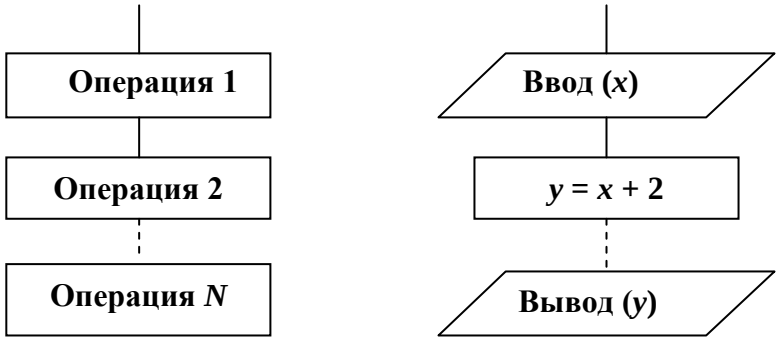
Перед началом расчетов вводятся значения переменных **r, n, U**. Затем идет операционный блок, в котором рассчитываются общее сопротивление гирлянды **R** и сила тока **I**. Значение **I** выводится.

3. Составные операции и их базовые конструкции

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

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

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



а) б)

Рис. 1.4. Блок-схема составной операции **Прохождение:**
а) общий вид; б) пример

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

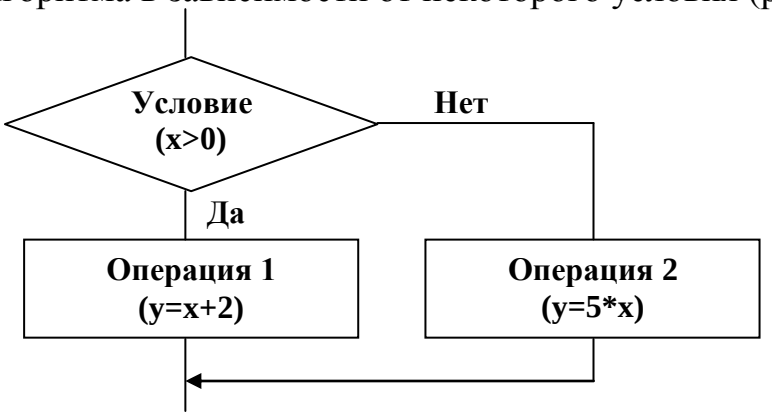


Рис. 1.5. Блок-схема составной операции **Выбор**

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.

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

Подробно данная операция будет рассмотрена в 6-й и 7-й лекциях.

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 операторов языка программирования С++.

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 С++

Цель лекции. Изучить алфавит языка С++, типы данных, ключевые слова, идентификаторы, объявление переменных и констант в программе, структуру программ на языке С++.

Основные вопросы лекции

1. Алфавит языка С++.
2. Ключевые слова, идентификаторы, комментарии в С++.
3. Типы данных в языке С++.
4. Объявление переменных и констант в С++.
5. Литералы и другие типы данных в С++.

1. Алфавит языка С++

В тексте на любом естественном 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 С++ 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 (см. таб. 2.1).

Таблица 2.1 Специальные символы языка С++

Символ	Наименование	Символ	Наименование
,	Запятая	)	Правая круглая скобка
.	Точка	(	Левая круглая скобка
;	Точка с запятой	}	Правая фигурная скобка
:	Двоеточие	{	Левая фигурная скобка
?	Вопросительный знак	<	Меньше
'	Апостроф	>	Больше
!	Восклицательный знак	[	Правая квадратная скобка
|	Вертикальная черта	]	Левая квадратная скобка
/	Дробная черта	#	Номер
\	Обратная дробная черта	%	Процент

~	Тильда	&	Амперсанд
*	Звездочка	^	Логическое Нет
+	Плюс	=	Равно
-	Минус	"	Кавычки

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

4. **Управляющие символы**. В языке С++ 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. 2.2).

Таблица 2.2 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. 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 С++ различает прописные и строчные буквы).

Длина идентификатора по стандарту не ограничена. Идентификатор создается на этапе **объявления переменной**, функции, типа и т.п., после этого его можно использовать в последующих операторах программы. При выборе идентификатора необходимо иметь в виду следующее:

1. Идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка;

2. Не рекомендуется начинать идентификаторы с символа подчеркивания;

3. На идентификаторы, используемые для определения *внешних переменных*, налагаются ограничения компоновщика.

Ключевые слова - это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. В таблице 2.3 приведен список ключевых слов С++ в алфавитном порядке.

Таблица 2.3 Ключевые слова языка С++

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 поясняющий значение фрагментов программного кода текст, предназначенный для пользователя. Комментарии компилятор не воспринимает как часть программы.

В С++ комментарий можно сделать одним из двух способов:

1. Символы «/*» начинают комментарий, заканчивающийся символами «*/». Вся эта последовательность символов эквивалентна символу пробела. Этот способ полезно использовать для многострочных комментариев и изъятия частей программы при редактировании, например:

include <iostream. h> **/*** директива компилятору для включения в программу заголовочного файла **iostream. h** */.

2. Символы «//» предваряют комментарий, который заканчивается в конце данной строки, например:

int a = 17; // инициализация переменной

3.Типы данных в языке С++

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 именно закодированы и какие действия с ними можно выполнять. Все это задается при описании данных с помощью типа. Каждая константа, переменная, результат вычисления выражения или функции должны иметь конкретный тип.

Тип данных однозначно определяет:

1. Множество их возможных значений (связанное с внутренним представлением данных в памяти компьютера);

2. Допустимые действия над данными (операции и функции).

Программист задает тип каждой величины, используемой в программе, исходя из характеристик реальных объектов, которые представляют эти величины. 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 величин, тоже зависит от типа. По стандарту единицей памяти в C++ является байт.

Типы данных в языке C++ делятся на элементарные (базовые или основные) 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.

Базовые типы данных в языке C++ часто называют арифметическими, поскольку их можно использовать в арифметических операциях. 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.

Рассмотрим базовые типы данных языка C++ подробнее.

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 - байт:

sizeof(char) = sizeof(signed char) = sizeof(unsigned char) = 1 .

Размер байта зависит от реализации, однако этот размер должен быть достаточен, чтобы вместить код любого символа из набора символов реализации для данного компьютера. Наличие знака у типа **char** тоже зависит от реализации: он может совпадать либо с **signed char**, либо с **unsigned char**.

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 С++ 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. Диапазон целых чисел в таблице 2.4.

Таблица 2.4 Целый тип данных в С++

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

Действительные числа записываются в обычной десятичной либо экспоненциальной форме. Например, **123.5**, **3.14**, **0.95**.

Число **123.5** можно записать в экспоненциальной форме как совокупность числа, называемого мантиссою, и порядка, который представляет степень 10-ти. Например,

$$\mathbf{123.5 * 10^0 = 12.35 * 10^1 = 1.235 * 10^2 = 1235 * 10^{-1}}.$$

В С++ в экспоненциальной форме вместо цифры 10 записывается буква Е:

123.5E0 = 12.35E1 = 1.235E2 = 1235E-1.

Действительный тип – это множество рациональных и иррациональных чисел с десятичной точкой. В языке С++ используются следующие действительные типы: **float** – числа з плавающей точкой одинарной точности, **double** - числа з плавающей точкой двойной точности, **long double – для** вычислений особой точности. Диапазон чисел действительного типа приведен в таблице 2.5.

Таблица 2.5 Действительный тип данных в С++

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

4. Объявление переменных и констант в языке С++

Основное назначение всех компьютерных программ – операции с данными с целью получения некоторого результата. Условием этого процесса является наличие фрагмента памяти запоминающего устройства компьютера, в котором можно хранить элемент данных и к которому можно обращаться по некоторому имени.

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 С++ принято назначать имена переменных с прописных букв, а классов – с заглавных. Компилятор С++ различает прописные и строчные буквы, например, **Sum** и **sum** означают разные переменные.

Объявление переменной с одновременным заданием типа хранимого под ее именем элемента данных осуществляется следующим образом:

ТипПеременной ИмяПеременной;

Например, строка **int arg;** объявляет целую переменную с именем **arg**.

Имена переменных не должны совпадать с ключевыми словами.

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

Например

int sum=0; или **int sum(0);**

float a=2.7; или **float a(2.7);**

Переменная объявляется **перед** тем местом, где она будет впервые задействована. Подробнее о месте объявления переменных в С++ будет сказано в следующих лекциях.

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

const Тип ИмяКонстанты = Значение;

Например, объявить постоянную π можно следующим образом:

const float pi = 3.14159; .

5. Литералы и другие типы данных

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

Числовые литералы (числа) 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 – еще и латинскую букву «x». Например:

4523, **679134** - числа в десятичной системе счисления;
067, **01497** - числа в восьмеричной системе счисления;
0x73, **0x948** - числа в шестнадцатеричной системе счисления.

Символьный литерал (символьная константа) используется для представления одного символа. Это любой символ, заключенный в одинарные кавычки, например:

‘W’, ‘&’, ‘3’, ‘Ф’.

Строковый литерал – это любая последовательность символов, заключенная в парные кавычки:

“ALPHA”, “ТАБЛИЦЯ”, “Это строка \n”, “123/5”.

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 в совокупности с операциями передачи.

Вопросы для самоконтроля

1. Как правильно записать действительное число на языке *C*++?
2. Результат решения задачи на ПК - число 7.45600000*E*+03. 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 языка программирования С++

Цель лекции. Изучить операции и выражения языка С++, порядок и особенности применения.

Основные вопросы лекции

1. Операции в С++.
2. Арифметические операции в С++.
3. Операция присваивания в С++.
4. Арифметические операции с присваиванием в С++.
5. Выражения в С++.
6. Стандартные математические функции в среде Visual C++ 2010.
7. Операция приведения типа.
8. Операции инкремент и декремент.

1. Операции в С++

Операция в языке С++ - это проведение строго определенных знаком операции действий над субъектами операции - **операндами**. Каждая операция в языке С++ характеризуется своим приоритетом. **Приоритет операции** – это порядок, в котором проводится вычисление значения выражения (таб. 3.1).

Каждая операция имеет свой 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 языка С++ определено количество операндов:

1) один операнд – **унарная** операция; например, **x^2;**
2) два операнда – **бинарная** операция; например, операция сложения **y+z;**
3) три операнда – операция **условие**. Такая операция единственная.

Полный список операций языка С++ приведен в таблице 3.1.

Таблица 3.1 Операции языка программирования С++

Операция	Название операции	Приоритет	Порядок выполнения
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 **NOT**		
!	Логическое **Не**	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
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

2. Арифметические операции

В языке С++ 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, 11/3 будет равно 3, а выражение 1/2 будет равно нулю.

Таблица 3.2. Арифметические операции в языке С++

Операция	**Знак в языке С++**	**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 11%3 будет равен 3, а результатом операции 5%5 будет ноль.

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 = V;

где **A** –переменная любого базового типа данных языка С++; V –выражение.

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

Особенностью операции присваивания в языке С++ 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.**

4. Арифметические операции с присваиванием

Одним из способов сократить объем программного кода являются арифметические операции с присваиванием. Например, операцию

total = total + item;

можно записать как

total += item;

С присваиванием комбинируется не только операция сложения, но и другие арифметические операции: **-=**, ***=**, **/=**, **%=** и т.д.

5. 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.**

Математические действия с переменными, константами и функциями в языке C++ записываются только в строчку, при этом для соблюдения необходимой по условию задачи очередности операций используются круглые скобки.

6. Стандартные математические функции в среде Visual C++ 2010

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 (библиотека math.h)

Название функции	**Что вычисляет**	**Тип данных функции и аргумента**
abs(x)	Абсолютное значение (модуль) 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 задается в радианах)	**double sin(double x)**
asin(x)	Арксинус (угол задается в радианах	**double asin(double x)**

Название функции	Что вычисляет	Тип данных функции и аргумента
	от –1 до +1)	
cos(x)	Косинус (угол задается в радианах)	**double cos(double x)**
acos(x)	Арккосинус (угол задается в радианах от –1 до +1)	**double acos(double x)**
tan(x)	Тангенс (угол задается в радианах)	**double tan(double x)**
atan(x)	Арктангенс (угол задается в радианах)	**double atan(double x)**

При обращении к этим функциям необходимо придерживаться следующих правил:

1) **x** и **y** должны быть типа **double**;
2) углы (аргументы) в тригонометрических функциях задаются в радианах.
3) вычисляемые функциями данные имеют тип **double**.

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)) .
```

7. 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 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 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.

Возможно принудительное приведение типа, выполняемое с помощью операции приведения типа, имеющей следующую структуру:

Тип Выражение; .

где **тип** – один из стандартных (встроенных) типов данных в *C++*.

Операция может применяться к любому операнду в выражении. Например, чтобы результат деления на 2 переменной **x** типа **int** имел тип **float**, надо записать

float x / 2; .

8. Операции инкремента и декремента

Унарная операция **инкремента** увеличивает свой операнд (**обязательно переменную**) на единицу и записывается **i ++** , что эквивалентно записи **i = i + 1** .

Унарная операция **декремента** уменьшает свой операнд на единицу и записывается **i –** , что эквивалентно записи **i = i – 1;**

Эти операции могут использоваться и в выражениях:

sum = sum + x *i++;

Инкремент и декремент реализуются в двух формах: префиксной **++i** и постфиксной **i++.**

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-вывода и структуру программ в среде Visual C++ 2010

Основные вопросы лекции

1. Структура программ на языке C++ в среде Visual C++ 2010.
2. Операторы ввода и вывода в консольном приложении Win 32 среды Visual C++ 2010

1. Структура программ на языке C++ в среде Visual C++ 2010

Программа на языке C++ состоит из следующих элементов (см. рис. 4.1):

1.Директивы препроцессора.
2. Указания компилятору.
3. Объявления и определения.
4. Одна или несколько функций.

директива препроцессора 1 # директива прероцессору 2 ..... # директива прероцессору N
указание компилятору 1 # указание компилятору 2 ..... # указание компилятору M
Объявление и определение (инициализация) переменных, констант, функций
Функция 1 Функция 2 ..... Функция N

Рис. 4.1. Структура программы на языке C++

Рассмотрим структуру программ на языке C++ для консольного приложения Win 32 среды Visual C++ 2010 на примере одной простой программы. Это программа 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. 4.1. Код (текст) программы с комментариями приведен ниже.

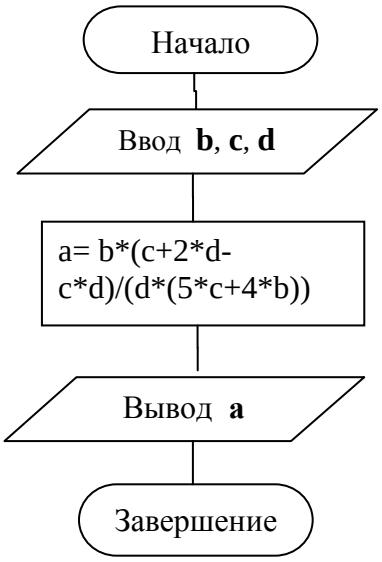


Рис. 4.2. Блок-схема алгоритма вычисления переменной **a**

Для реализации данной блок-схемы разработан следующий программный код:

```
1. #include "stdafx.h"
2. #include <conio.h>                    //Файл conio.h обеспечивает задержку
                                          //окна DOS на экране дисплея
#include <iostream>                       //Директива  подключения файла
                                          //заголовка   iostream
using namespace std;                      //Подключение всех имен из
                                          //пространства имен  std
int _tmain(int argc, _TCHAR* argv[ ])     //Объявление главной функции
{                                         //Начало главной функции
      double a;                           //Объявление переменной   a
      double b, c=2, d=3;                 //Объявление переменных   b, c, d и их
                                                //инициализация
      cout<<"vvedite b"<<endl;            //Вывод на экран надписи для ввода b .
                                          /      /Здесь endl – признак перемещения
курсора на                                      //новую  строку
            cin>>b;                       //Ввод значения переменной  b
            a=b*(c+2*d-c*d)/(d*(5*c+4*b));      //Вычисление  a по заданной
формуле
            cout<<"a = "<<a;              //Вывод на экран значения a
            getch();                      //Функция  задержки окна DOS на экране
return 0;
}
```

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.

Указания компилятору осуществляются с помощью директивы препроцессора **pragma**, служащей для установки различных параметров компилятора. Директива имеет вид:

#pragma директива .

Например, директива **#pragma argsused** означает, что компилятору не нужно выдавать предупреждающие сообщения о том, что параметры функции не используются внутри нее.

В анализируемой программе такие указания компилятору отсутствуют.

Функции описывают совокупность тех действий, которые должна выполнить программа.

Если функций несколько, то среди них выделятся одна - **главная**, 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. В нашей программе это строки 6 и 15.

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.

В исходный текст программы можно вставлять текст из другого файла с помощью директив препроцессора **#include** (включать):

 # 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** - поиск нужного файла производится в текущем каталоге. Например, 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 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.

Подробнее о правилах и особенностях создания функций в программе мы остановимся 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 С++ в следующих лекциях.

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

В Консольном приложении Win32 среды программирования Visual C++ 2010, в котором консольный (с использованием клавиатуры) ввод данных производится при помощи оператора **cin**. В С++ 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; **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 (на экран дисплея) вывод данных производится при помощи оператора **cout**. В С++ 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; **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 Win32 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. раздел 1 лекции 2).

В операторе **cout** допускается производить арифметические действия. Например, при выполнении оператора

cout <<2 + 3 + 4

на экран дисплея будет выведена результат – цифра 9.

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 разветвляющихся программ с использованием операторов условного и безусловного переходов, особенностей их конструкции и исполнения.

Основные вопросы лекции

1. Разветвляющиеся вычислительные процессы.
2. Условный оператор **if…else**. Синтаксис и использование в программах.
3. Оператор выбора **switch**.
4. Безусловный оператор.

1. Разветвляющиеся вычислительные процессы

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:

< меньше ;
<= меньше или равно;
> больше ;
>= больше или равно;
== равно;
!= не равно.

Любой конкретный алгоритм может быть записан на языке программирования, использующем только три управляющий структуры: последовательное выполнение, ветвление и повторение.

Последовательность операторов выполняется в порядке их естественного расположения в программе, с возможным отклонением для вызова внешнего фрагмента (функции), но с обязательным возвратом в точку вызова.

2. Условный оператор if…else

Любой алгоритм может быть записан на языке программирования с использованием только трех управляющих структур: последовательное выполнение, ветвление и повторение. Последовательное выполнение реализуется в линейных алгоритмах, исследованных в предыдущей лабораторной работе. В данной работе будут исследованы разветвленные алгоритмы, реализующие алгоритмы ветвления.

На рис. 2.1 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**.

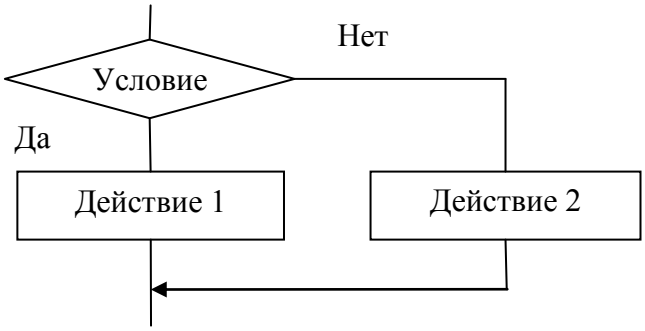


Рис.5.1. Структура ветвления

Структура ветвления описывается в языке С++ 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 - **{ }**.

В**ыражение** в скобках представляет собой условие, заданное с помощью операций отношений и логических операций.

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. Например,

```
{
a=b+c;          // сложный оператор
d=a+b;
}

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

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

if – если;
then – тогда;
else – иначе;
case – случай;

switch – переключатель;
break – прерывать;
default – не выполнять.

Пример 1. Исследовать и выполнить программу для определения переменной **a** по следующему условию:

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

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

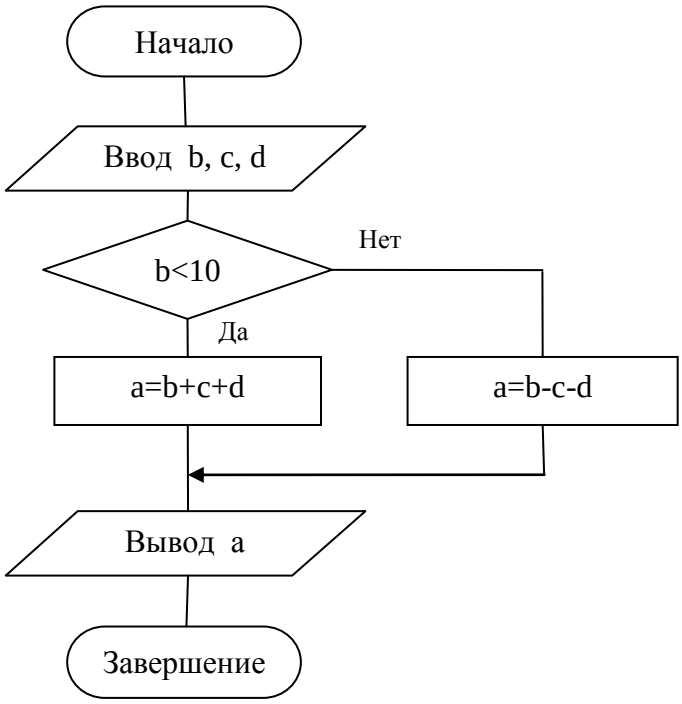


Рис. 5.2. Блок-схема алгоритма определения переменной **a** (простые операторы)

Программный код, реализующий алгоритм, имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>                  //Файл conio.h обеспечивает задержку окна DOS на
                                    //экране дисплея
#include <iostream>                 //Директива include подключает файл ввода–
вывода   using namespace std;       //Подключает все имена из пространства имен  std

int _tmain(int argc, _TCHAR* argv[])      //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 (1-я часть)
    a=b+c+d;                     //Вычисление переменной a при выполнении
условия
    else                         //Условный оператор if …else (2-я часть)
    a=b-c-d;                     //Вычисление  a при невыполнении условия
cout<<"a = "<<a;                 //Вывод на экран значения переменной  a
getch();                         //Функция задержки окна DOS на экране
return 0;
}
```

Пример 2. Исследовать и выполнить программу для определения переменной **a** по условию **Примера 1**. Отличие 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. Объясните их.

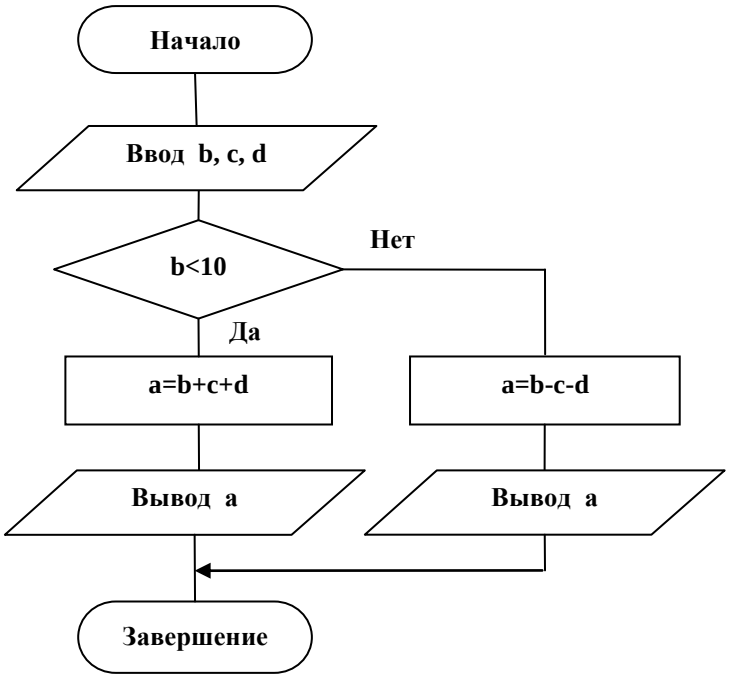


Рис. 5.3. Блок-схема алгоритма определения переменной **a** (сложные операторы)

Введите и выполните программный код, реализующий алгоритм. Обратите внимание на сдвиг фигурных скобок, ограничивающих сложные операторы, относительно скобок, охватывающих тело главной функции.

```
#include "stdafx.h"
#include <conio.h>               //Файл conio.h обеспечивает задержку окна
                                 //DOS на экране дисплея
#include <iostream>              // include подключает файл ввода–вывода
using namespace std;             //Подключает все имена из пространства имен  std
```

```
int _tmain(int argc, _TCHAR* argv[])      //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 (1-я часть)
      {                                   //Начало сложного оператора
      a=b+c+d;                            //Вычисление перем. a при выполнении условия
      cout<<"a=b+c+d= "<<a;               //Вывод на экран значения переменной  a
      }                                   //Конец сложного оператора
      else                                //Условный оператор if …else (2-я часть)
      {                                   //Начало сложного оператора
      a=b-c-d;                            //Вычисление  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;
}
```

3. Использование условного оператора if…else для 3-х интервалов значений переменой

Выше исследовались 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).

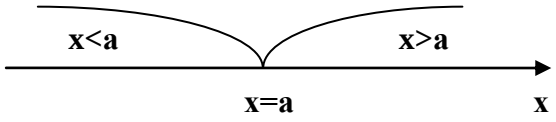


Рис. 5.4. Графическая интерпретация логического выражения для двух интервалов значений **x**

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. 2.5). В этом случае условие запишется следующим образом:

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

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

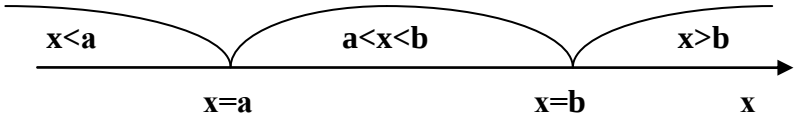


Рис. 5.5. Графическая интерпретация логического выражения для трех интервалов значений **x**

4. Исследование вложенного условного 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. 2.1).

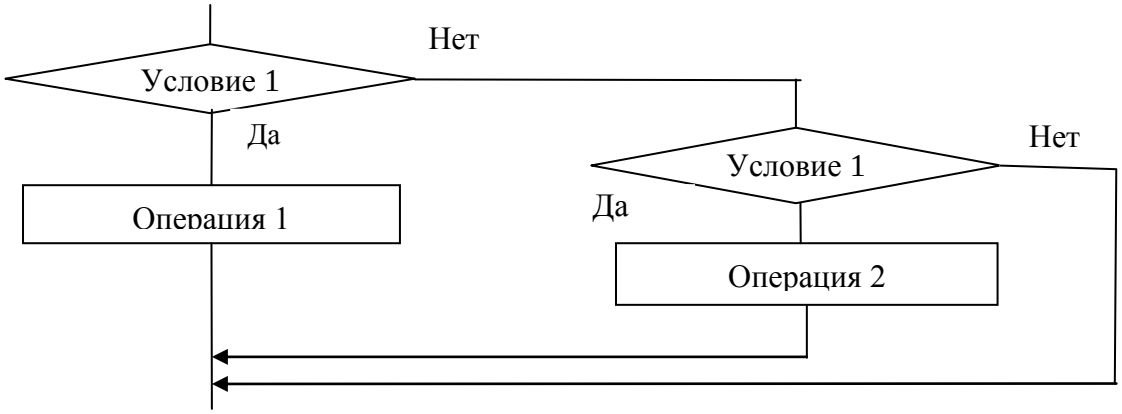


Рис.5.6. Структура вложенного условного оператора **if…else**

3. 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-схема алгоритма, реализующего работу оператора выбора **switch**, приведена на рис. 5.4.

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).

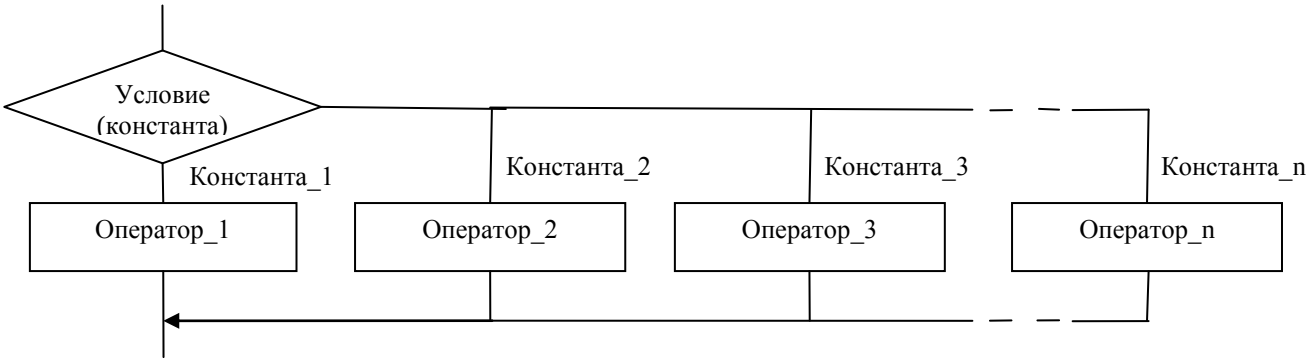


Рис. 5.4. Блок-схема алгоритма работы оператора **switch**

Пример 3. Проанализируем 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 знаками, то на экран выводится сообщение о неизвестной операции.

```
#include "stdafx.h"
#include <conio.h>                    //Файл conio.h обеспечивает задержку
                                      //окна DOS на экране дисплея
#include <iostream>                   //Подключение файл ввода–вывода
iostream
using namespace std;                  //Подключает имена из пространства  std
int _tmain(int argc, _TCHAR* argv[])  //Объявление главной функции _tmain
{                                     //Начало главной функции
double a, b, res;                     //Объявление вещественных переменных
char operation;                       //Объявление символьной  переменной
cout << "Vvedite a : "<<endl;         //Вывод на экран надписи  Vvedite a :
```

```
cin >> a;                                   //Ввод значения переменной  a
cout << "Vvedite operation : "<<endl;       //Вывод на экран надписи  Vvedite
operation :
cin >> operation;                           //Ввод знака операции
cout << "Vvedite b : "<<endl;               //Вывод на экран надписи  Vvedite b :
cin >> b;                                   //Ввод значения переменной  b
      switch (operation)                    //Условие (сравнение) в операторе  switch
      {                                     //Начало оператора  switch
      case '+': res = a + b; break;         //Метка ”+” и вычисл-е res для этой метки
      case '-': res = a - b; break;         //Метка ”-” и вычисл-е  res для этой метки
      case '*': res = a * b; break;         //Метка ”*” и вычисл-е res для этой метки
      case '/': res = a / b; break;         //Метка ”/” и вычисл-е  res для этой метки
      default : cout <<"Unknown operation"<<endl;      //Вывод сообщения о
                                                       //неизвестной операции
      }                                     //Конец сложного оператора
cout << "Result : " << res;                 //Вывод на экран результата вычислений
getch();                                    //Функция задержки окна DOS на экране
return 0;
}                                           //Конец главной функции
```

4. Безусловный оператор goto

Переход к любому оператору программы без условия (директивный переход) в среде Visual C++2010 осуществляется с помощью оператора безусловного перехода **go to**. 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 A.

Пример 4. Проанализируем работу оператора **goto** на примере программы из задания 2. Расположим оператор **goto** с меткой **A** перед условным оператором, а 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>                 //Подключение файла ввода–вывода iostream
using namespace std;                //Подключает все имена из пространства  std
int _tmain(int argc, _TCHAR* argv[])//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  Vvedite b
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();                               //Функция задержки окна DOS на экране
return 0;
}
```

Вопросы для самоконтроля.

1. Как записать в операторе **if** проверку условия на равенство переменной нулю?
2. Записан оператор **if(a>b) x=a else x=b.** Сколько ошибок сделано при записи оператора?
3. В операторе **if** проверяется условие **if ((x>=a) &&(x<=b)).** Что означает это условие?
4. Укажите правильную запись оператора условного перехода:
   1. **if(x>1) then y=x; else y:=0;**
   2. **if(x>1)y=x else y=0;**
   3. **if(x>1)y=x; else y=0;**
5. В общем виде оператор условного перехода имеет вид **if(умова) S1; else S2;** . Что такое **S1** и **S2**?
6. Укажите правильную запись оператора безусловного перехода:
   1. **goto A1;**
   2. **goto A1,A2,A3**;
   3. **goto 1;**
7. Записан оператор: **if(X>0) { X = X-1; Y= 0; } else Y= Y+1 .** Сколько ошибок в приведенном операторе?
8. Укажите на возможность такой записи оператора **if(1< X < 3) S1; else S2;**
9. 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;
10. В операторе **if** необходимо записать условие, что "X не принадлежит отрезку [A, B]". Укажите на правильную запись этого условия.
    1. **if(A > X > B)** 2. **if((X>A)&&(X>B))** 3. **if((X<A)and(X>B))**

Лекция 6

Циклические вычислительные процессы в Visual C++ 2010. Операторы while и do...while.

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++ 2010
4. Вложенные циклы в Visual C++ 2010

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 приведены реальные операторы.

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.

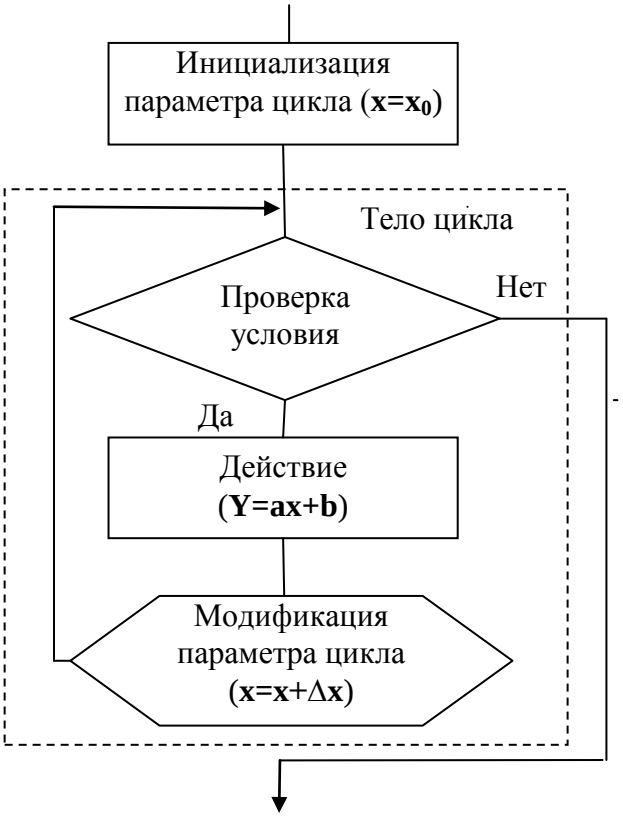


Рис. 6.1. Блок-схема алгоритма циклического вычислительного процесса

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.

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 **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.

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

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

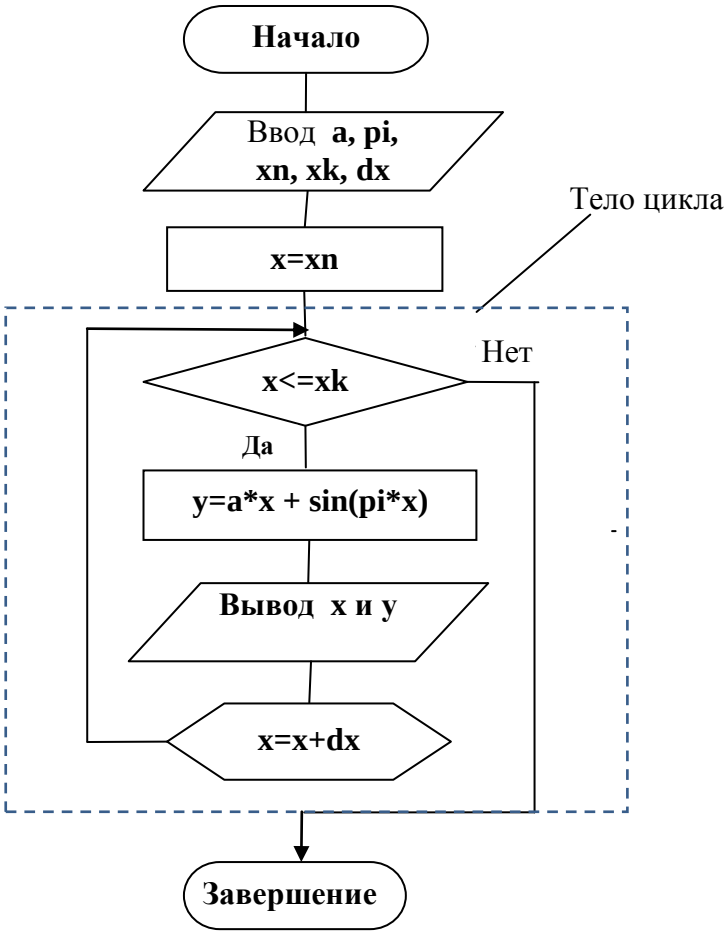


Рис. 6.2. Блок-схема алгоритма программы с оператором **while**

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);

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

$\mathbf{x_k}$ – 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**.

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

```
int _tmain(int argc, _TCHAR* argv[])//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;            //Инициализация 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 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 переменной y
    x=x+dx;                         //Вычисление  следующего значения  x
    }                               //Конец составного оператора
getch();                            //Функция задержки окна DOS на экране
return 0;
}                                   //Конец главной функции
```

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.

Блок-схему вычислительного процесса, реализующего оператор цикла **do…while**, проанализируем на примере решения программы для вычисления значения функции $Y = \sin^3(x-a) - \cos^2(x+a)^3$ для значений аргумента **x** от x_0=0 до x_k=1 с шагом Δx=0,2. Используем оператор **do…while**.

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);

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**.

Блок-схема алгоритма решения данной задачи приведена на рис. 6.3.

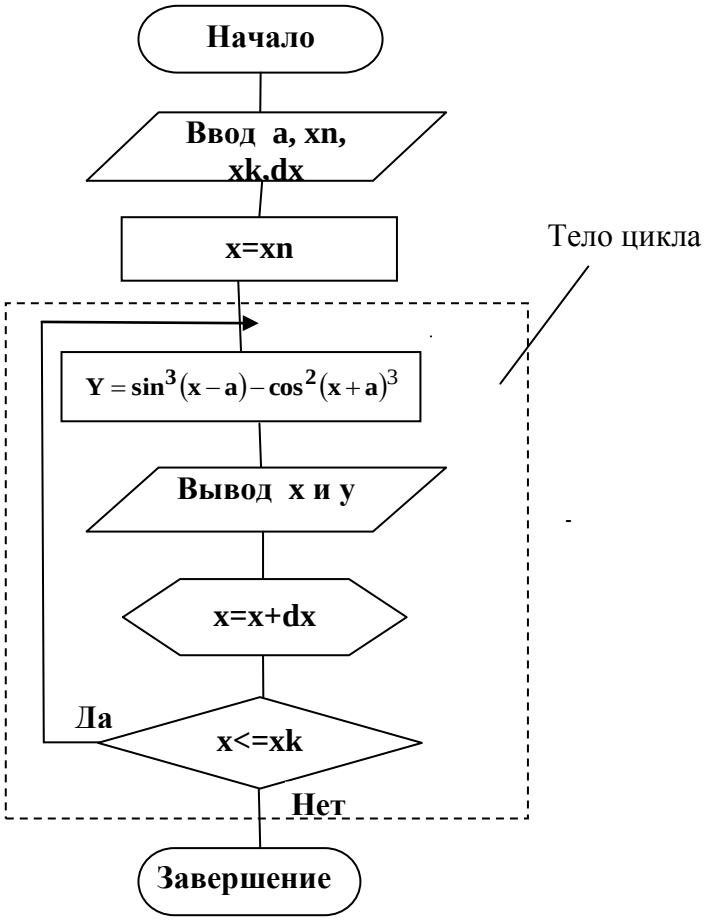


Рис. 6.3. Блок-схема алгоритма программы с оператором **do...while**

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

```
int _tmain(int argc, _TCHAR* argv[])      //Объявление главной функции _tmain
{
double a=0.3;                             //Инициализация константы a
double x, y, xn, xk, dx;                  //Объявление переменных и параметра цикла
cout<<" Vvedite xn, xk, dx: "<<endl; //Надпись для ввода  параметров цикла
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  x
cout<<" X   "<<"   Y  "<<endl;            //Вывод на экран заголовков "X" и "Y"
     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  x
     }                                    //Конец составного оператора
     while (x<=xk);                       //Оператор while (условие цикла x ≤ xk)
```

```
getch();                                  //Функция задержки окна DOS на экране
return 0;
}                                         //Конец главной функции
```

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 С++ 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.

Проанализируем работу программы в Visual C++ 2010, использующей такую вычислительную конструкцию, как вложенный цикл, или цикл в цикле на примере программы для вычисления значения функции $s = \sum_{n=1}^{4} \sum_{k=1}^{5} \sin^3(knx - a)$ для некоторой переменной **x**. Используем вложенный цикл с оператором **while**.

Блок-схема алгоритма решения данной задачи приведена на рис. 6.4. Необходимо ввести код программы, построить решение и произвести вычисления.

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

a – константа, инициализируем ее, как **a** = 1,4 и определим ее тип как **double** (действительное двойной точности);

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

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

s – переменная, обозначим ее как **S**, тип **double.**

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

```
int _tmain(int argc, _TCHAR* argv[])      //Объявление главной функции _tmain
{
const double a=1.4;                       //Объявление константы a
double x, y, n, k, s;                     //Объявление  переменных и параметров
cout<<" Vvedite x "<<endl;                //Вопрос для ввода  x
cin>>x;                                   //Ввод  переменной x
cout<<"x= "<<x<<"  a= "<<a<<endl;         //Вывод исходных данных
n=1;                                      //Подготовка к циклу – предоставление
```

```
                                    //начального значения параметру цикла n
s=0;                                //Начальное значение переменной s
    while (n<=4)                    //Оператор while (условие цикла n ≤ 4)
    {
    y=0;                            //Начальное значение переменной y
    k=1;                            //Подготовка к циклу – предоставление
                                    //начального значения параметру цикла k
        while (k<=5)                //Вложенный оператор while
        {
        y=y+ pow(sin(k*n*x-a),3);   //Суммирование y на k-ом шаге
        cout<<k<<"   y =  "<<y<<endl;  //Вывод на экран  y на k-ом шаге
        k=k+1;                      //Вычисление  следующего значения  k
        }
        s=s+y;                      //Суммирование s на n-ом шаге
        cout<<n<<"   s =  "<<s<<endl;  //Вывод переменной s на k-ом шаге
        n=n+1;                      //Вычисление  следующего значения n
    }                               //Конец составного оператора
getch();                            //Функция задержки окна DOS на экране
return 0;
}                                   //Конец главной функции
```

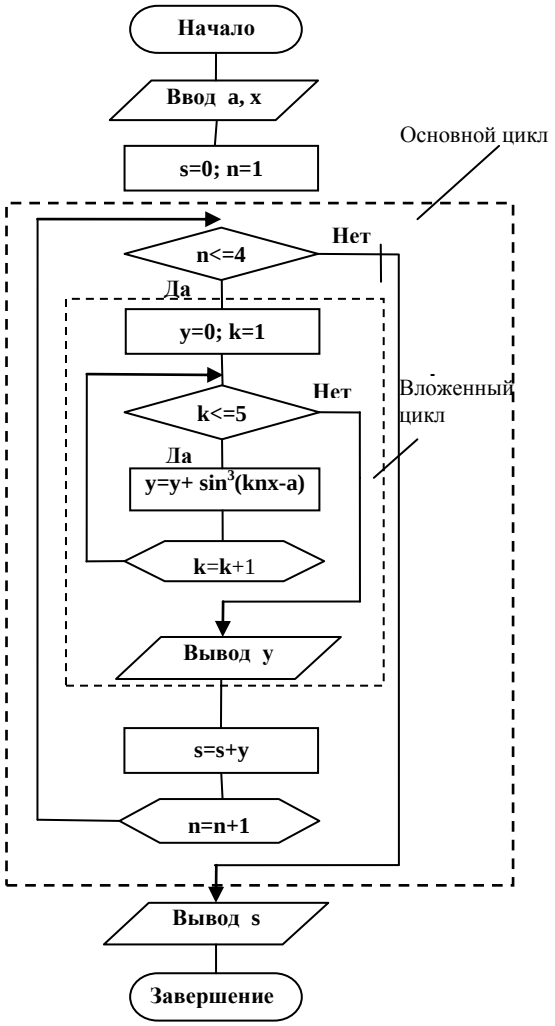


Рис. 6.4. Блок-схема алгоритма программы с использованием вложенного цикла

Вопросы для самоконтроля.

1. С помощью каких действий реализуются циклические процессы в C++?
2. Запишите в общем виде оператор **while** и опишите, как он выполняется.
3. Запишите в общем виде оператор **do... while** и опишите, как он выполняется.
4. Перечислите, что должно быть в конструкции цикла.
5. В языке C++ оператор **do… while** – это…
    1. Оператор цикла с предпосылкой.
    2. Оператор цикла с послеусловием.
    3. Оператор цикла с параметром.
6. В языке C++ в операторе **while** последовательность операторов (блок)
    1. Нужно заключать в фигурные скобки.
    2. Не нужно заключать в фигурные скобки.
    3. Заключать в фигурные скобки на усмотрение автора программы.
7. Сколько раз выполнится оператор цикла **int i=1; while(i>3) i=i+1;**?
    1. Один раз.
    2. Ни одного.
    3. Некоторое число раз.
8. Сколько раз выполнится оператор цикла **int x=1; do x=x+1; while(x>3);**?
    1. Один раз.
    2. Ни одного.
    3. Некоторое число раз.
9. Сколько раз выполнится оператор цикла **int i=1; while(i<3) i=i+1;**?
    1. 1.
    2. 2.
    3. 3.
10. Сколько раз выполнится оператор цикла **int x=3; while(x>1) x=x+1;**?
    1. Один раз.
    2. Ни одного.
    3. Бесчисленное число раз.
11. Сколько раз выполнится оператор цикла **int x=1; while(x<=3) x=x+1;**?
    1. 1.
    2. 2.
    3. 3.
12. Чему будет равняться **x** после выхода из цикла

int x=1; do x=x+1; while(x<3);?
    1. x=1.
    2. x=2.
    3. x=3.

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. Разработка программ с использованием оператора цикла **for**.
5. Вложенные циклы **for** в Visual C++ 2010.

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**.

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

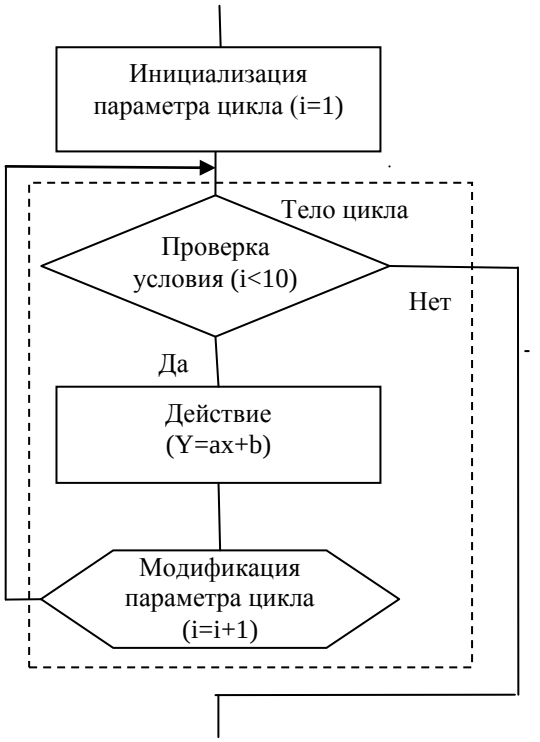


Рис. 7.1. Блок-схема алгоритма циклического вычислительного процесса

Алгоритм циклической вычислительной структуры содержит:

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.**

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

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 этот оператор 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):

- блок инициализации, т. е. присвоения параметру цикла начального значения (**i=1**);
- блок проверки условия (**i<=n**);
- 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):

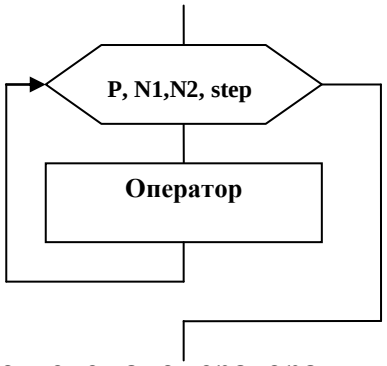


Рис. 7.2. Блок-схема оператора цикла **for**

На схеме алгоритма приведены следующие обозначения:

- P – параметр цикла;

- N1, N2 – границы изменения параметра цикла;

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

Параметр цикла **P**, границы изменения **N1**, **N2** и шаг **step** должны иметь один и тот же тип переменной.

Возможности оператора цикла **for** очень велики. 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 **N1**, **N2** 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;
```

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, одноместные) 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.

4. Разработка программ с использованием оператора цикла for

Рассмотрим простейший пример оператора цикла **for**. Необходимо исследовать программу для печати 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.

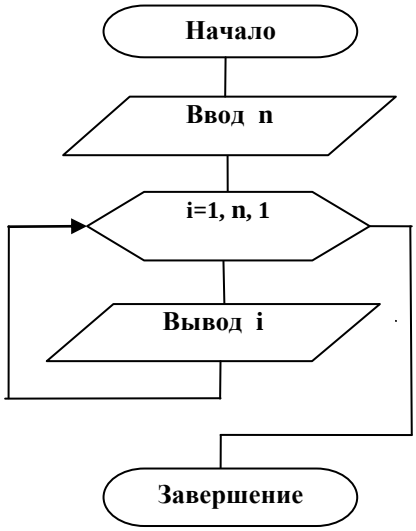


Рис. 7.3. Блок-схема алгоритма для печати в столбец **n** чисел

Ниже приведен программный код, реализующий данный алгоритм:

```
int _tmain(int argc, _TCHAR* argv[])
{
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 цикла i
getch();
return 0;
}
```

В результате выполнения программы 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.

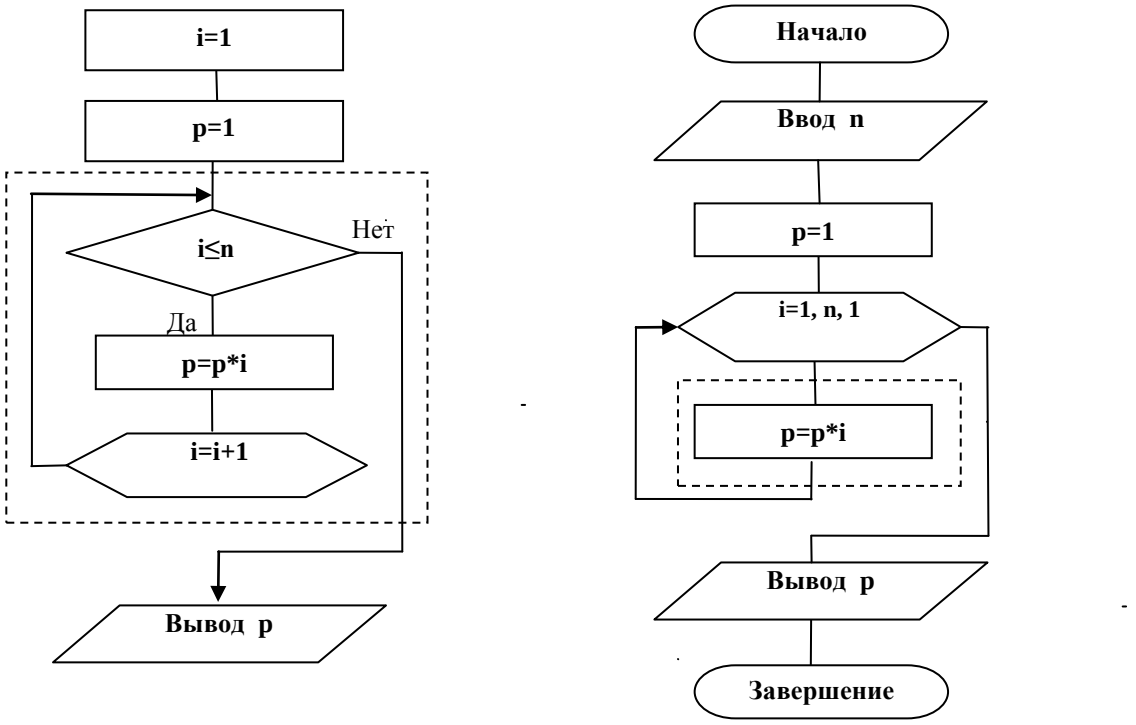
Пример 1. Исследуем программу вычисления факториала целого числа **n** с использованием оператора **for. Факториал** – это произведение целых чисел от 1 до **n**. Необходимо проанализировать блок-схемы вычислительного процесса и

алгоритма решения задания при помощи оператора цикла **for**, ввести код программы, построить решение и произвести вычисления для n=10.

Факториал числа **n** вычисляется по следующей формуле:

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

При вычислении факториала начальному значению произведения П необходимо присвоить 1. При каждом выполнении тела цикла $П_{i-1}$ будем умножать на номер шага **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) вычислительного процесса; б) алгоритма с использованием оператора **for**

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

```
int _tmain(int argc, _TCHAR* argv[])
{
int i, n, p;
cout<<"Vvedite N"<<endl;
cin>>n;                                  //Ввод значения переменной  n
p =1;                                    //Начальное значение  p для
                                         //вычисления 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.

Пример 2. 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 (ноль) переменной **p** для вычисления последующих сумм.

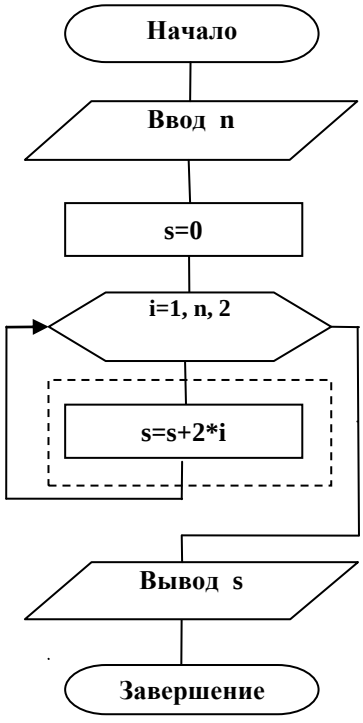


Рис. 7.5. Блок-схема алгоритма для вычисления суммы **n** четных чисел

Программный код для вычисления суммы **n** четных чисел имеет следующий вид:

```
int _tmain(int argc, _TCHAR* argv[])
{
int i, n, s;
```

```
cout<<"Vvedite N"<<endl;
cin>>n;                                  //Ввод значения переменной  n
s =0;                                    //Начальное значение  s для
                                         //вычисления 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.

Пример 3. Исследуем программу вычисления функции

$$Y = \sum_{k=1}^{6}\left(\cos^3 kx + \sin\frac{0{,}5x}{\sqrt[3]{k^2+x^2+1}}\right)$$

для **x**=1,32. Блок-схема алгоритма для вычисления данной функции аналогична блок-схеме алгоритма для вычисления суммы **n** четных чисел на рис. 7.5. Необходимо ввести код программы, построить решение и произвести вычисления.

Для решения задачи используется оператор цикла **for**, т. к. известно количество повторений тела цикла (k=10). Введем такие идентификаторы: **x, Y, k** и **S**. Формула для расчета функции **Y** на каждом шаге вычислений на языке С++ имеет следующий вид:

Y=pow(cos(k*x),3)+sin(0.5*x/pow((k*k+x*x+1),1/3)).

Тогда программа для расчета функции **Y** примет вид:

```
int _tmain(int argc, _TCHAR* argv[])
{
int i, k=10;
double x=1.32, Y, S;
Y=0;                                     //Начальное значение  Y для суммирования
cout<<"x= "<<x<<endl;                    //Вывод x
     for (i=1; i<k; i++)                 //Оператор цикла  for
     {                                   //Начало тела цикла
     S=pow(cos(k*x),3)+sin(0.5*x/pow((k*k+x*x+1),1/3));      //Расчет функции на
                                                            //каждом шаге
     Y=Y+S;
     cout<<"k= "<<i<<"    Y = "<<Y<<endl;   /Вывод  Y на каждом шаге
     }                                   //Конец тела цикла
getch();
```

```
return 0;
}
```

Так как тело цикла содержит более одного оператора, то эти операторы охвачены фигурными скобками.

После запуска программы на экране появится результат:

```
x= 1.32
k= 1    Y = 1.1365
k= 2    Y = 2.27299
k= 3    Y = 3.40949
k= 4    Y = 4.54599
k= 5    Y = 5.68249
k= 6    Y = 6.81898
k= 7    Y = 7.95548
k= 8    Y = 9.09198
k= 9    Y = 10.2285
```

Рис.7.6. Результаты вычислений для Примера 3.

Обратите внимание на то, что результат расчета функции **Y** выводится на каждом шаге вычислений. Это сделано с целью контроля при отладке программы. После устранения возможных ошибок можно выводить только итоговый результат.

Пример 4. Исследовать программу вычисления значения функции

$$Y = \sum_{n=0}^{4} n\left(\sum_{k=1}^{5} \sin^3(knx - a) \right)$$

для **x=**1,4 и **a**=2,3 (использовать вложенный цикл **for**). Необходимо ввести код программы, построить решение и произвести вычисления.

Определим исходные данные, которые понадобятся для решения задачи:

a – константа, инициализируем ее, как **a** = 2,3 и определим ее тип как **double** (действительное двойной точности);

x – константа, инициализируем ее, как **x**=1,4, тип **double**;

Y – переменная, тип **double**;

s – промежуточная переменная, обозначим ее как **S**, тип **double**.

Формула для расчета промежуточной суммы (внутри цикла по **k**) $s = \sin^3(knx - a)$ на k-м шаге вычислений на языке C++ имеет следующий вид:

s=pow(sin(k*n*x-a),3) .

Блок-схема алгоритма решения данной задачи приведена на рис. 7.7.

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

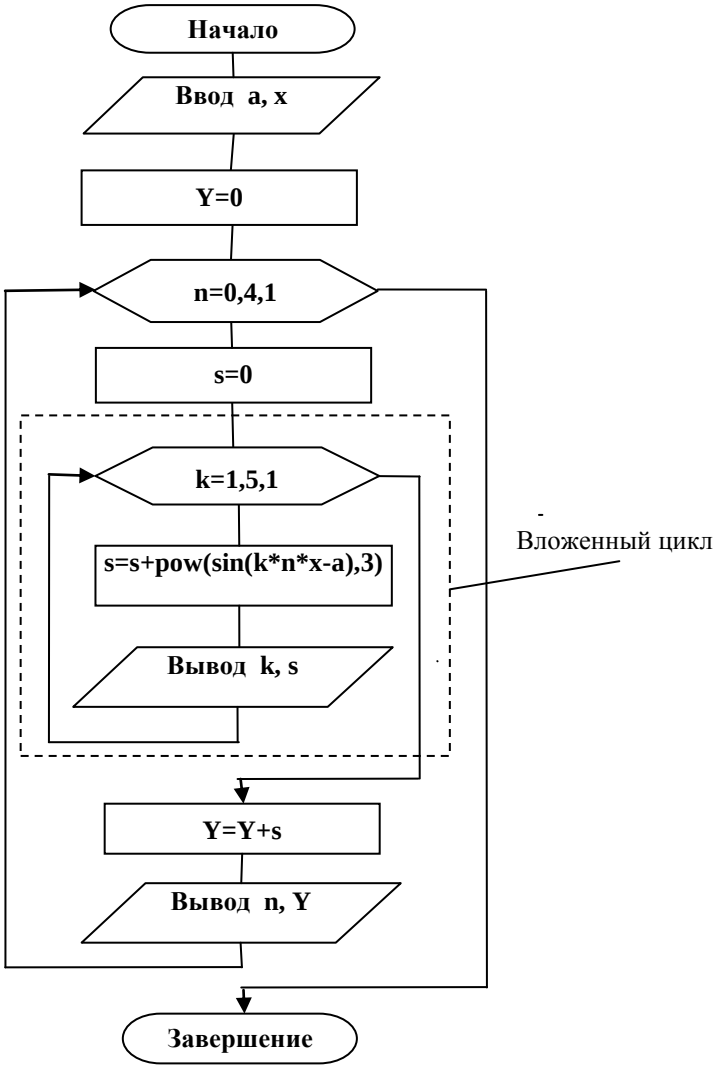


Рис. 7.7. Блок-схема алгоритма решения задачи с использованием вложенного цикла **for**

```
int _tmain(int argc, _TCHAR* argv[])      //Объявление главной функции _tmain
{
const double x=1.4, a=2.3;                //Инициализация констант a и x
double Y, n, k, s;                        //Объявление переменных
cout<<"x= "<<x<<"  a= "<<a<<endl;         //Вывод исходных данных
Y=0;                                      //Начальное значение s для суммирования
    for(n=0;n<=4;n++)                     //Внешний оператор for (условие  n ≤ 4)
    {                                     //Начало составного оператора
    s=0;                                  //Начальное значение y для суммирования
        for(k=1;n<=5;k++)                 //Вложенный оператор for
        {
        s=s+ pow(sin(k*n*x-a),3);         //Суммирование y на k-ом шаге
        cout<<k<<"   s =  "<<s<<endl;     //Вывод  переменной y на k-ом шаге
        }
    Y=Y+n*s;                              //Суммирование s на n-ом шаге
    cout<<n<<"  Y =  "<<Y<<endl;          //Вывод переменной s на k-ом шаге
    }                                     //Конец составного оператора
```

```
getch();                    //Функция задержки окна DOS на экране
return 0;
}                           //Конец главной функции
```

Результаты вычислений приведены на рис. 7.8. Промежуточные результаты вычислений выводятся на экран для дополнительного контроля правильности работы программы (тестирования программы).

```
x= 1.4  a= 2.3
  k = 1    s =  -0.414669
  k = 2    s =  -0.829338
  k = 3    s =  -1.24401
  k = 4    s =  -1.65868
  k = 5    s =  -2.07334

     n = 0  Y =  0
  k = 1    s =  -0.48065
  k = 2    s =  -0.370455
  k = 3    s =  0.476942
  k = 4    s =  0.473016
  k = 5    s =  -0.526753

     n = 1  Y =  -0.526753
  k = 1    s =  0.110195
  k = 2    s =  0.10627
  k = 3    s =  0.100225
  k = 4    s =  0.225993
  k = 5    s =  -0.21643

     n = 2  Y =  -0.959613
  k = 1    s =  0.847396
  k = 2    s =  0.841352
  k = 3    s =  0.388923
  k = 4    s =  1.20605
  k = 5    s =  1.20274

     n = 3  Y =  2.64861
  k = 1    s =  -0.0039253
  k = 2    s =  0.121842
  k = 3    s =  0.938967
  k = 4    s =  1.79397
  k = 5    s =  1.94911

     n = 4  Y =  10.445
```

Рис. 7.8. Результаты вычислений для Примера 4

Вопросы для самоконтроля:

1. Перечислите, что должно быть в конструкции цикла.
2. С помощью каких действий реализуются циклические вычислительные процессы в языке С++?
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?

Лекции 8-9
Одномерные массивы и их обработка в Visual C++ 2010

Цель лекции: изучение особенностей обработки одномерных массивов в Visual C++ 2010

Основные вопросы лекции

1. Одномерные массивы данных и их инициализация в среде Visual C++ 2010.
2. Консольный ввод и вывод одномерных массивов в среде Visual C++ 2010.
3. Присваивание и копирование одномерных массивов в Visual C++ 2010.
4. Поиск элемента в одномерных массивах в Visual C++ 2010.
5. Перестановка и сортировка элементов в одномерных массивах.

1. Одномерные массивы данных и их инициализация 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 исследуем 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. **Р**асположение 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];
в = A[5];
```

Индексация элементов в массиве **всегда** 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 массив **m[100]** целочисленный массив:

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 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.

2. Консольный ввод и вывод одномерных массивов в среде 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).

Пример 1. Исследовать способы консольного ввода и вывода массива **A** из **N** целых чисел. 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.

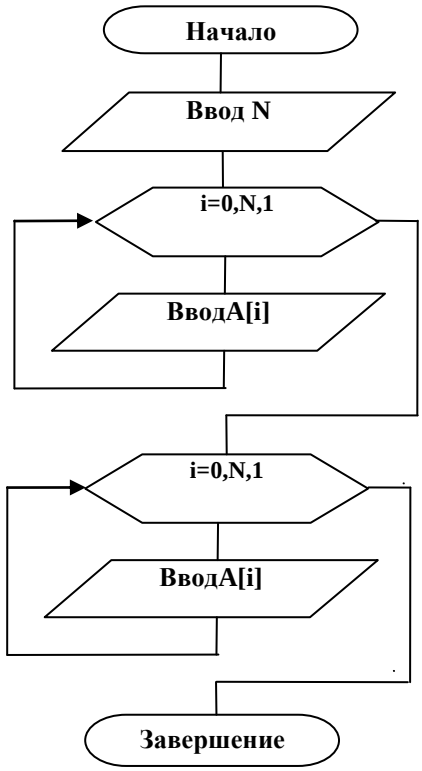


Рис. 8.1. Блок-схема алгоритма ввода и вывода элементов вектора с использованием цикла **for**

Реализующая данный алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
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** вид экрана 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

3. Присваивание и копирование одномерных массивов в Visual C++ 2010

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

A[0]= 3.5;
A[1]= 0;
A[2]= 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 массив **B** будет иметь вид:

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

Пример 2. Задать массив **X** действительных чисел из 8 элементов непосредственным присваиванием в программном коде. Вычислить массив **Y** по заданной формуле:

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

Массив **Y** вывести на экран.

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

Реализующая данный алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i;
double X[8]={1.2, 3.4, 12, 5.12, 23.1, 3, 0, 35.2};  //Инициализация массива X[8]
double Y [8];
    for (i=0; i<8; i++)                    //Цикл для вычисления и вывода
```

```
                                        //элемента  Y[i]
    {                                   //Начало составного оператора
    Y [i]=2*X[i]+pow(X[i], 5/3);        //Вычисл-е  элементов массива 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
```

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.

Пример 3. Исследуем программу для поиска элемента одномерного массива по образцу. 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.

Приступая к решению задачи, определим типы и имена переменных, 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);

Бл**о**к-схема алгоритма поиска элементов одномерного массива приведена на рис.8.2.

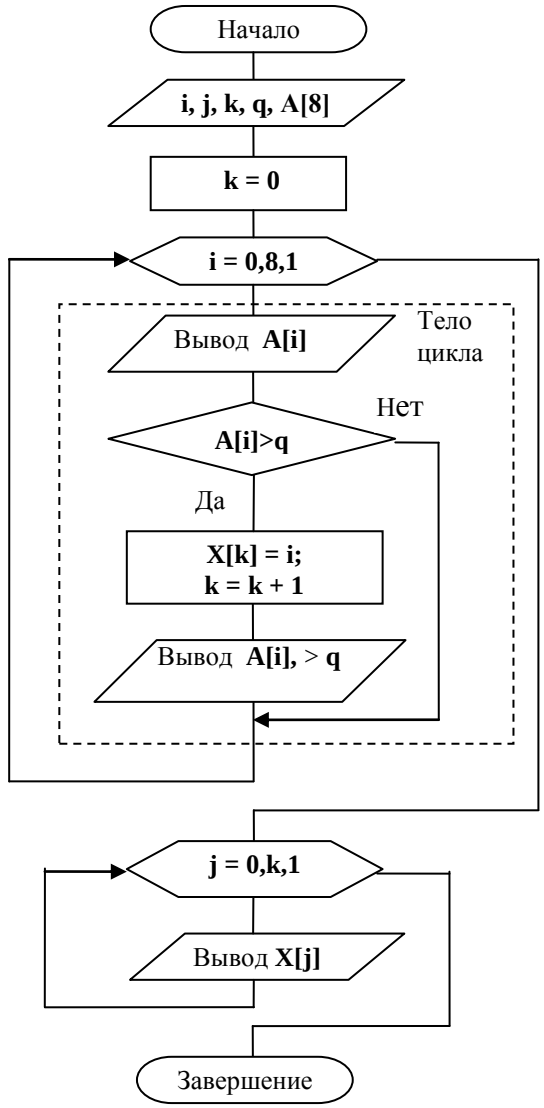


Рис. 8.2. Блок-схема алгоритма поиска элементов одномерного массива

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i, j, k;
double q=3.7;                          //Объявление перменной  q
double A[8]={1,  2,  3,  4,  5,  6,  7,  8};   //Инициализация элементов  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 элем-в массива A[8]
      {                                //Начало составного оператора в цикле
      cout<<"  A["<<i<<"] = "<<A[i];   //Вывод всех элементов массива  A[8]
            if(A[i]>q)                 //Условный оператор для сравнения
                                       //элементов массива A[8] с q
            {                          //Начало составного оператора в условном
```

```
            X[k]=i;                          //Записываются номера элементов,
                                             //больших q,  в массив X[k]
            k=k+1;
            cout<<"  >q"] <<endl;            //Сообщение, что A[i] больше  q
            }                                //Конец составного оператора в условном
            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, больших, чем q
      cout<<X[j]<<" ";                       //Вывод номеров таких элементов
getch();
return 0;
}
```

В результате выполнения программы получим:

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

Исследуем программу для поиска элемента одномерного массива по правилу.

Пример 4. Задан массив **A** из 8 произвольных чисел. Найти максимальный элемент массива **A** и его номер. На экран вывести исходный массив **A**, максимальный элемент массива **A** и его номер.

Решение. Определим типы и структуры данных, которые будут использоваться в программе:

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

A[8] – массив из 8 произвольных чисел;

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

C – вспомогательная переменная для хранения номера (индекса) максимального элемента массива.

Блок-схема алгоритма поиска максимального элемента одномерного массива приведена на рис. 8. 3

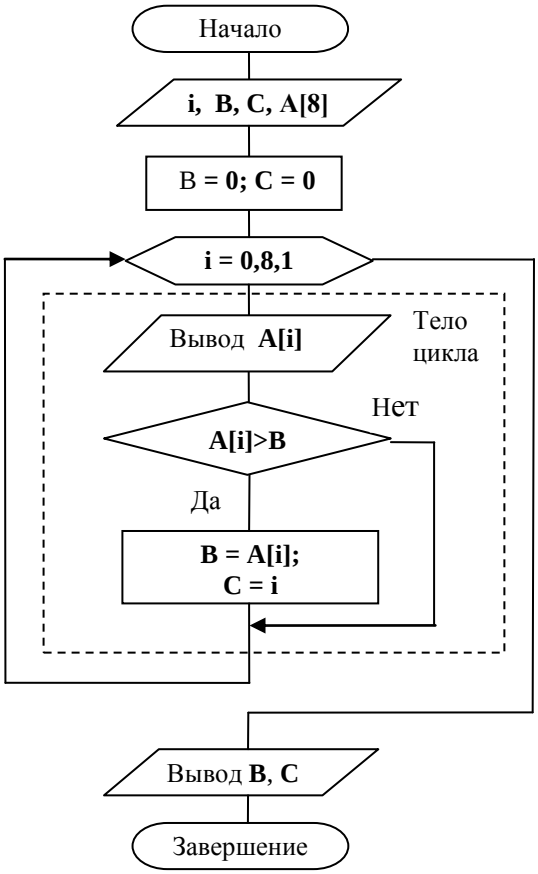


Рис. 8.3. Блок-схема алгоритма поиска максимального элемента одномерного массива

```
int _tmain(int argc, _TCHAR* argv[])
{
int i, C;
double B;
double A[8]={1, 2, 3, 4, 5, 6, 7, 1};      //Инициализация элементов массива A
B=A[0];
C=0;
    for (i=0; i<8; i++)                    //Цикл для перебора элем-в массива A[8]
    {                                      //Начало составного оператора в цикле
    cout<<"A["<<i<<"]="<<A[i];             //Вывод всех элементов массива  A[8]
        if(A[i]>B)                         //Условный оператор для сравнения
                                           //очередного элемента  с B
        {                                  //Начало составного оператора в условном
        B=A[i];                            //Присваив. B значения макс. элемента
        C=i;                               //Присваивание переменной C номера
                                           //максимального элемента
        }                                  //Конец составного оператора в условном
    }                                      //Конец составного оператора в цикле
cout<<endl<<»Max element: «<<B;
cout<<endl<<"N max elementa: "<<C;
getch();
```

return 0;
}

В результате выполнения программы получим:

A[0] = 1 A[1] = 2 A[2] = 3 A[3] = 4 A[4] = 5 A[5] = 6 A[6] = 7 A[7] = 1

Max element: 7

N max elementa: 6

5. Перестановка и сортировка элементов одномерных массивов в среде Visual C++ 2010

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

Эта задача решается путем многократного поиска максимального элемента в исходном массиве и пересылки его в новый упорядоченный массив.

Рассмотрим сортировку массива методом парной перестановки элементов массива.

Элементы массива **x[0], x[1], x[2], x[3]…x[n-1]** рассматриваются слева направо и по очереди сравниваются пары элементов:

x[0] и **x[1], x[1]** и **x[2], x[2]** и **x[3]…x[n-2]** 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 массива дело, то есть он будет иметь индекс **n-1**. При втором просмотре массива эта процедура повторяется только для массива **x[0], x[1], x[2], x[3]…x[n-2]**. При третьем просмотре – для массива **x[0], x[1], x[2], x[3]…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. 7.1).

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]**.

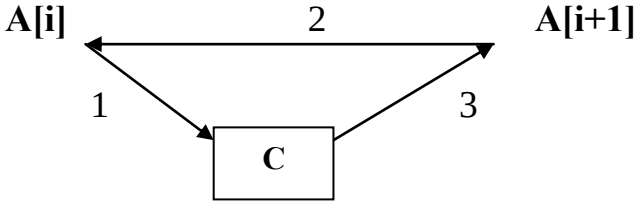


Рис.8.4. Схема перестановки элементов одномерных массивов

Фрагмент программы перестановки двух элементов массива будет иметь следующий вид.

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

Исследуем программу для сортировки одномерного массива по возрастанию.

Пример 5. Задан массив **A** из 15 произвольных чисел. Отсортировать элементы массива **A** по возрастанию. На экран вывести отсортированный массив **A**.

Решение. Определим идентификаторы и типы данных, которые будут использоваться в программе:

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

A[15] – исходный массив из 15 произвольных 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.

Программный код для решения задачи имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i,j;                                    //Объявление переменных  i и j
double B;                                   //Объявление вспомогательной перем. B
double A[15]={13,2,0,4,12,6,10,14,9,7,11,5,1,8,3};      //Инициализация исходного
массива A[15]
    for (j=0; j<15-1; j++)                  //Внешний цикл для перебора элементов
        for (i=0; i<15-1; i++)              //Вложенный цикл перебора элементов
            if(A[i]>A[i+1])                 //Условный оператор для сравнения
                                            //элементов массива A[15]
            {
            B=A[i];                         //Присваивание B значения i-ого элемента
            A[i]=A[i+1];                    //Перемена местами  i-го и i+1-го элем-в
            A[i+1]=B;
            }
    for (i=0; i<15; i++)                    //Цикл для вывода элементов отсорти-
                                            //рованного массива A[15]
    cout<<"  A["<<i<<"] = "<<A[i];
getch();
return 0;
}
```

В результате выполнения программы получим:

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

После отладки программы можно оставить только итоговый (на последнем шаге) вывод массива A.

Данная программа не является оптимальной. Видно, что исходный массив был отсортирован уже при **j**=10. Для устранения избыточных операций можно ввести некоторые критерии, но это усложнило бы понимание студентами рассматриваемых в данной работе алгоритмов.

Вопросы для самоконтроля:

1. Массив в информатике – это…
   1. Некоторое множество мест в памяти компьютера.
   2. Некоторое множество мест в памяти компьютера, называемых элементами массива, к которым обращаются по одному имени переменной.
   3. Некоторое множество мест в памяти компьютера, называемых элементами массива, к каждому из которых обращаются по своему имени.
2. Чем однозначно определяется каждый элемент массива?
   1. Именем массива.
   2. Именем массива и его порядковым номером в массиве (индексом).
   3. Индексом элемента.
3. Какой индекс у 3-его элемента массива **int B[100]**?
   1. 3.
   2. 2.
   3. 0.
4. Можно объявить массив следующим образом: **int C[6]= {2, 4, 7, 11, 12, 13}**?
   1. Да.
   2. Нет.
   3. В некоторых случаях.
5. Как объявляется массив в языке C++?
   1. **int A[1..20**].
   2. **mas:** A[1..**20] int**.
   3. **int A[20]**.
6. Какое значение будет присвоено переменной **X** в программе на C++ оператором **X = m [13]**?
   1. 12-го элемента.
   2. 13-го элемента.
   3. 14-го элемента.
7. Сколько ошибок сделано при записи оператора **if (A>7) B=A;** если обрабатывается массив **A**?
   1. 3.
   2. 2.
   3. 0.
8. В программе на C++ при обработке массива все действия выполняются…
   1. Над массивом в целом.
   2. Над элементами массива .

3. Только над индексами элементов массива.

9. Каким оператором можно ввести с клавиатуры **n** элементов массива **X**?

1. **for (i=0; i<=n; i++) cin>>X[i]**.
2. **for (i=0; i< n; i++) cin>>X[i]**.
3. **for (i=1; i<=n; i++) cin>>X[i]**.

10. Как в программе на С++ будет выведен на экран массив **A[k]**оператором **for(i=0; i<k; i++) cout<<A[i]<<endl;**?

1. В виде строки.
2. В виде столбца.
3. В виде матрицы.

11. Любой массив в языке С++…

1. Имеет имя.
2. Не имеет имени.
3. Имена имеют только элементы массива.

12. Задан массив **X = {X1, X2 ..., XN}**. Каким оператором можно вывести этот массив на экран?

1. **cout<<X;**
2. **for (i=0; i<n; i++) cout<<X;**
3. **for (i=0; i<n; i++) cout<<X[i];**

13. В программе на С++ объявлен массив **int B[10];** Массив **B** содержит…

1. 9 элементов.
2. 10 элементов.
3. 11 элементов.

14. Возможна ли следующая инициализация массива **B**: **int B[6]={2, 4, 7};**?

1. Возможно.
2. Не возможно.
3. Зависит от программиста.

15. В программе на С++ при обработке массива индекс элемента может быть задан…

1. Значением.
2. Выражением.
3. Не важно, как.

16. Есть массивы **A** и **B** одинакового размера и их элементы имеют тип **int**. Возможно ли в языке С++ копирование массива **A** в **B** операцией **B = A**?

1. Возможно.
2. Невозможно.
3. В некоторых случаях.

17. При каком условии можно копировать массив **A** в массив **B**?

1. Массивы **A** и **B** разного размера и их элементы имеют одинаковый тип.
2. Массивы **A** и **B** одинакового размера и их элементы имеют разный тип.
3. Массивы **A** и **B** одинакового размера и их элементы имеют одинаковый тип.

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

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

Лекции 10-11
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 и их инициализация в среде Visual C++ 2010.
2. Консольный ввод и вывод двухмерных массивов в среде Visual C++ 2010.
3. Присваивание и копирование двухмерных массивов в Visual C++ 2010.
4. Поиск элемента в двухмерных массивах в Visual C++ 2010.
5. Перестановка и сортировка элементов в двухмерных массивах.

1. Двухмерные массивы данных и их инициализация в Visual C++ 2010

Под размерностью массива понимают число индексов, которое необходимо указать для получения доступа к отдельному элементу массива. Массивы, рассмотренные нами выше, 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).

В памяти компьютера двухмерный массив располагается непрерывно по строкам, то есть

A[0][0], A[0][1], A[0][2], … A[0][8], A[1][0], A[1][1], A[1][2], A[1][3] … A[1][8].

На рис. 10.1 приведена схема размещения элементов массива из 24-х целых чисел по имени **A** размерностью 3×8. В памяти будет зарезервировано место для 24-х целочисленных элементов массива, которые располагаются в **непрерывном (!!!)** блоке памяти.

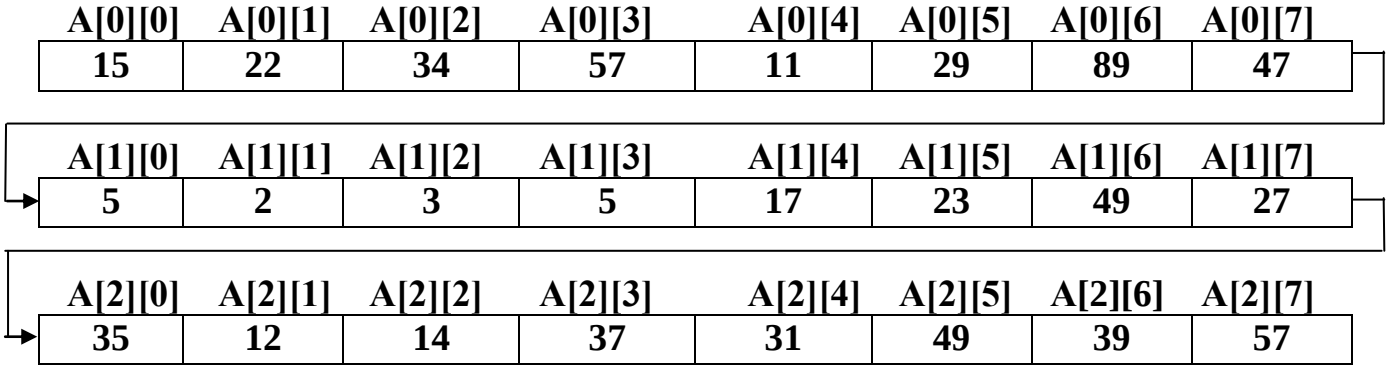


Рис. 10.1. Значения и расположение в памяти элементов массива **A[3][8]**

Массивы хранятся в памяти компьютера так, что самый правый индекс измеряется быстрее всего.

Возможна инициализации массива непосредственно в программном коде. В этом случае в фигурных скобках приводятся значения элементов массива (рис. 9.1) в следующем виде:

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

или

int A[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:

	Второй индекс (j)			
Первый индекс (i)	A_{00}	A_{01}	A_{02}	A_{03}
	A_{10}	A_{11}	A_{12}	A_{13}
	A_{20}	A_{21}	A_{22}	A_{23}

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 **m[100]** целочисленный массив:

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 двухмерными 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.

2. Консольный ввод и вывод двухмерных массивов в среде Visual C++ 2010

Т. к. все действия необходимо выполнять над элементами двухмерных массивов, то для ввода двухмерного массива в память компьютера необходимо организовать его поэлементный ввод посредством 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.

Пример 1. Исследовать способы консольного ввода и вывода двухмерных массивов. Необходимо ввести с клавиатуры матрицу **A** размерностью **M**×**N** в память компьютера и вывести эту информацию на дисплей.

Решение. 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, а вложенный – по столбцам.

Вывод исходной матрицы на экран будет производиться аналогичным образом.

Блок-схема алгоритма решения данной задачи приведена на рис. 10.2.

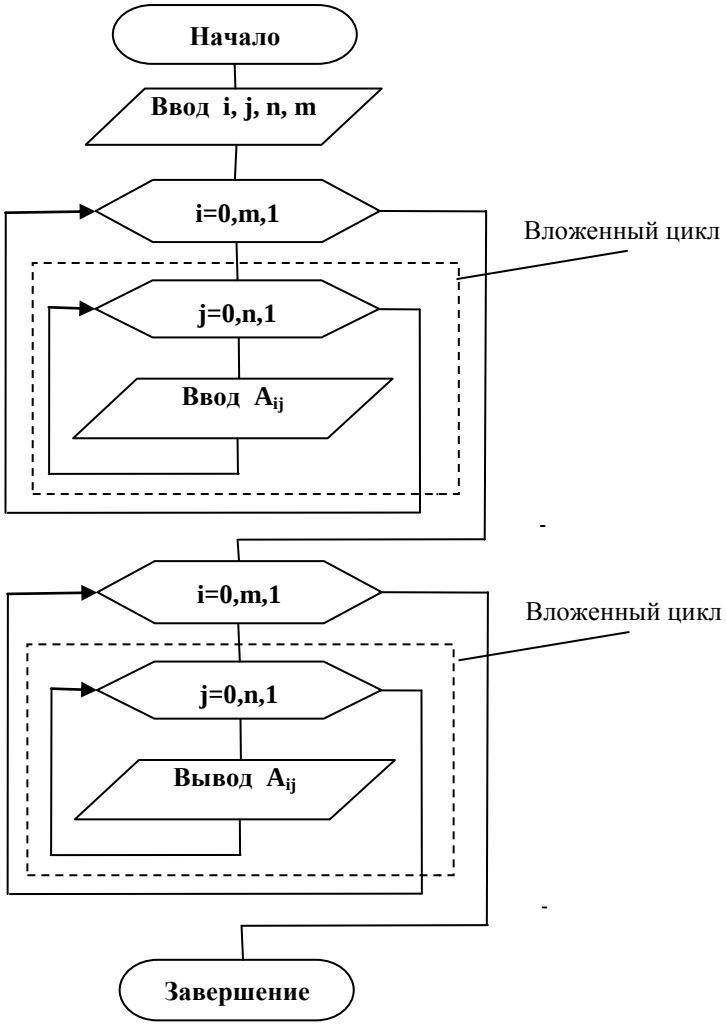


Рис. 10.2. Блок-схема алгоритма поэлементного ввода и вывода матрицы с использованием цикла **for**

Программный код, реализующий данный алгоритм, выглядит следующим образом:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
const int M=10,N=10;                  //Число строк и столбцов матрицы A с запасом
int i, j, A[3][4];                    //Объявление переменных и матрицы A
cout<<" Vvedite postrochno elementu matricu:"<<endl;
   for(i=0; i<3; i++)                 //Внешний цикл ввода элементов матрицы A
        for(j=0; j<4; j++)            //Вложенный цикл ввода элементов матрицы A
        cin>>A[i][j];                 //Ввод элемента матрицы A
cout<<endl;
cout<<"Matrica A"<<endl;
   for(i=0; i<3; i++)                 //Внешний цикл вывода элементов матрицы A
   {                                  //Начало составного оператора для внешнего
                                      //цикла for
        for(j=0; j<4; j++)            //Вложенный цикл для вывода элементов
                                      //матрицы A
        cout<<A[i][j]<<"   ";         //Вывод элемента матрицы A
cout<<endl;
   }                                  //Конец составного оператора для внешнего
                                      //цикла for
getch();
return 0;
}
```

После выполнения программы получим:

```
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
```

3. Исследование различных операций в двухмерных массивах в среде Visual C++ 2010

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.

Пример 2. Исследуем операцию присваивания в двухмерных массивах, а также операцию транспонирования матрицы. Задана матрица целых чисел **A** размерностью 3×4. Необходимо транспонировать матрицу **A**, т. е. поменять местами ее строки и столбцы, а затем рассчитать матрицу **B**, элементы которой определяются по формуле

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

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

$$A(3,4) = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix} .$$

Блок-схема алгоритма решения данной задачи приведена на рис. 10.3.

Реализующая данный алгоритм программа имеет следующий вид:

```
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]={1,2,3,4,5,6,7,8,9,10,11,12};//Инициализация исходной матрицы A
cout<<"Matrica A"<<endl;           //Вывод названия
    for(i=0; i<3; i++)             //Внешний цикл вывода элементов матрицы A
    {
        for(j=0; j<4; j++)         //Вложенный цикл для вывода элементов A
        cout<<A[i][j]<<"   ";      //Вывод элементов исходной матрицы A
        cout<<endl;
    }
    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 = A transponirovannaya"<<endl;
    for(i=0; i<4; i++)             //Начало цикла
    {
        for(j=0; j<3; j++)         //Вложенный цикл для вывода матрицы AT
```

```
            cout<<AT[i][j]<<"  ";          //Вывод элементов матрицы AT
            cout<<endl;
      }
cout<<"Matrica B = f(AT)"<<endl;    //Вывод названия
      for(i=0; i<4; i++)                //Внешний цикл для поэлементного вычисле-
                                        //ния матрицы B
      {
            for(j=0; j<3; j++)          // Вложенный цикл для вычисления матрицы B
            {
            B[i][j]=5*pow(sin(pow(AT[i][j],3)),2)+pow((AT[i][j]*AT[i][j]),1./3.);
            cout<<B[i][j]<<"  ";          //Вывод элементов матрицы B
            }
      cout<<endl;
      }
getch();
return 0;
}
```

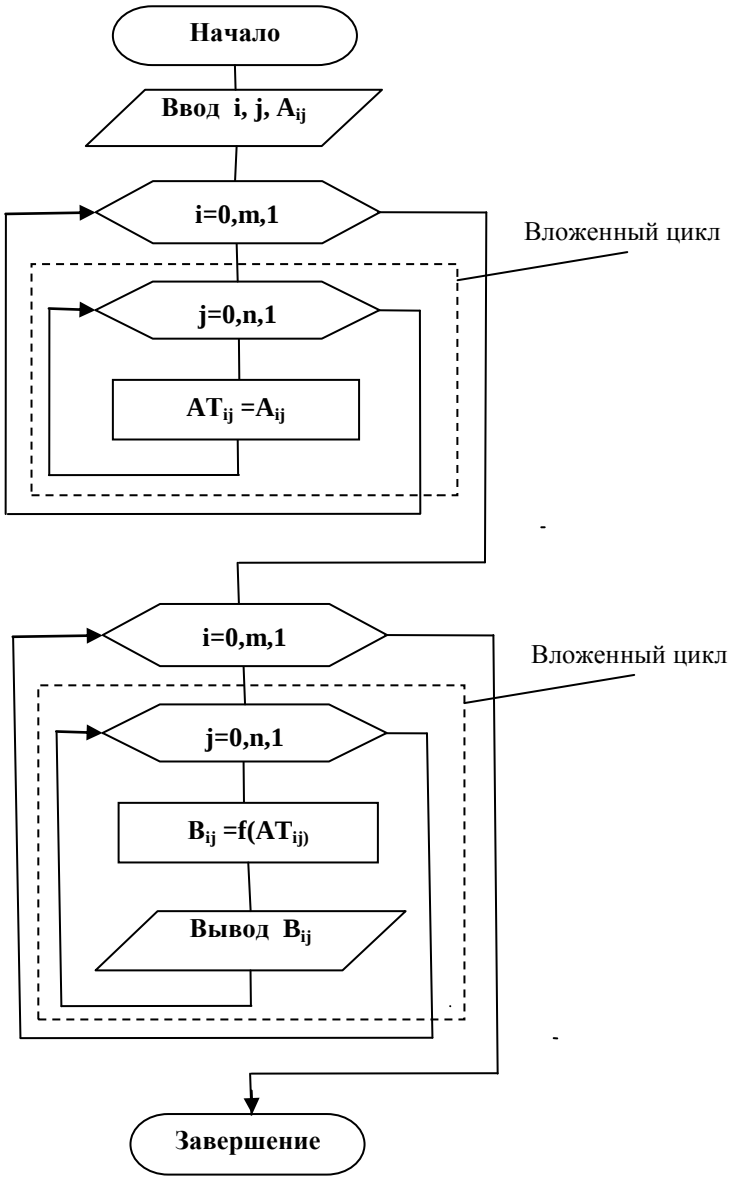
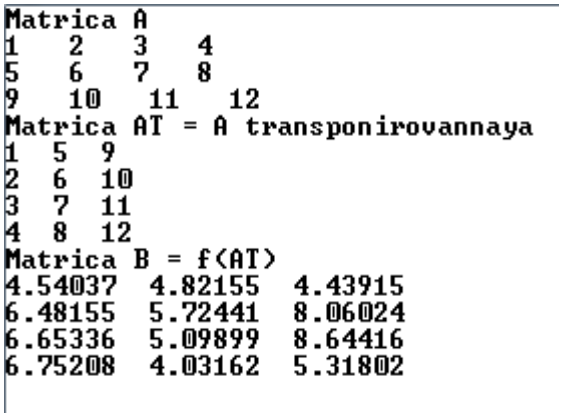


Рис. 10.3. Блок-схема алгоритма транспонирования и присваивания матрицы

После запуска программы на выполнение экран дисплея должен иметь вид:



Пример 3. Исследуем операцию поиска максимального элемента матрицы. 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. 10.4.

Для решения задачи используем следующий программный код:

```
#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]={1,2,18,3,20,4,5,6,7,8,9,10,19,11,12,13,14,15,16,17};
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 A
        cout<<A[i][j]<<"\t";            //Вывод  элемента  матрицы A
        cout<<endl;
    }
B=A[0][0];                              //Подготовка к поиску макс. элемента
m=0; n=0;
    for(i=0; i<4; i++)                  //Внешний  цикл перебора эл-тов матрицы A
        for(j=0; j<5; j++)              //Внутренний  цикл перебора элементов  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 элемента Aij
            m=j;                        //Запоминание в m номера столбца элем. Aij
            }
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;
}
```

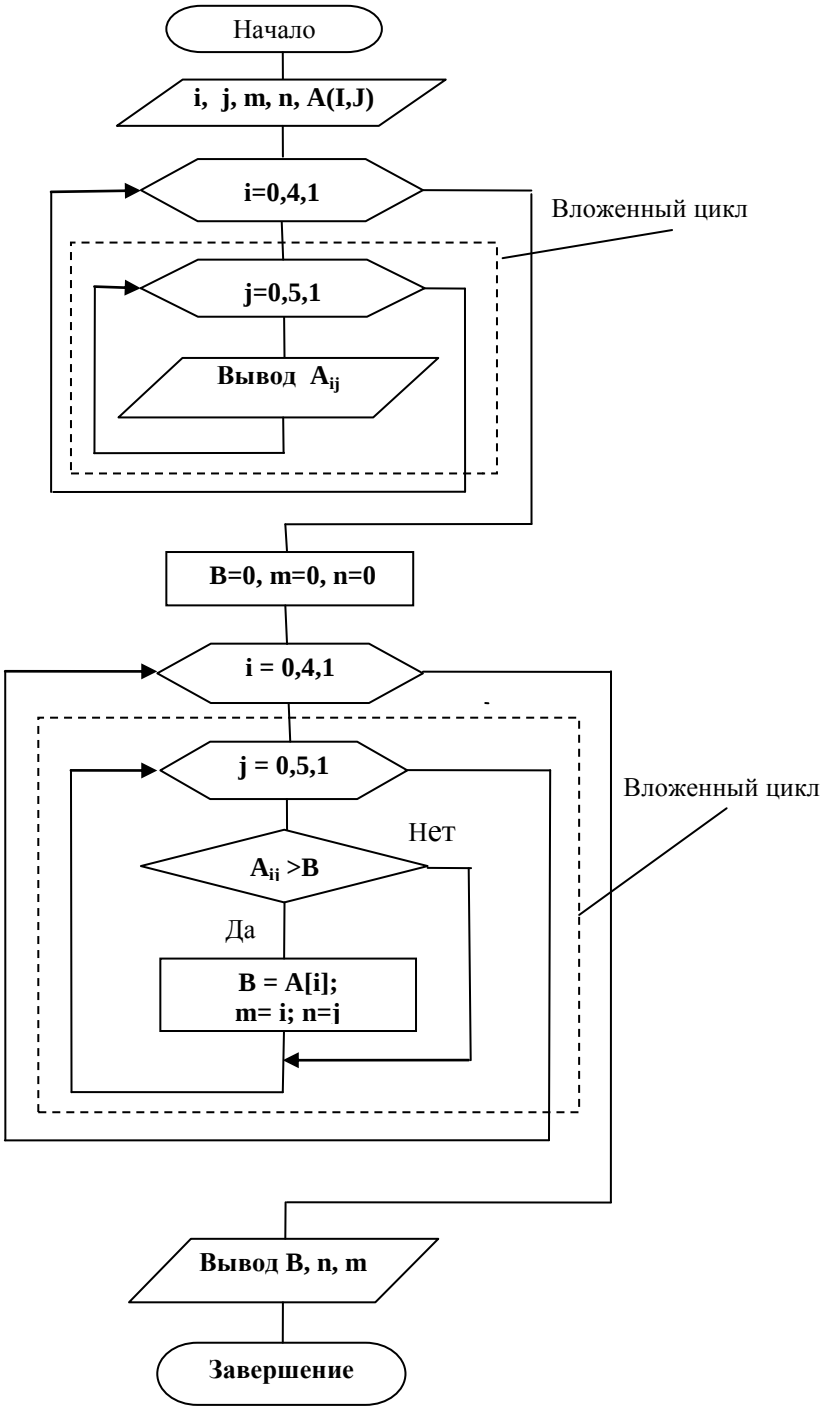


Рис. 10.4. Блок-схема алгоритма поиска максимального элемента матрицы

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

Пример 4. Исследуем способы ввода и вывода двухмерных массивов, а также следующие действия с матрицей: найти минимальный элемент каждой строки матрицы **A**, составить из них вектор **R** (одномерный массив) и определить суммы его четных и нечетных элементов. Задана матрица из 20 произвольных чисел размера 4×5. Элементы исходной матрицы **A** 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. 10.3.

Для решения задачи предлагается следующий программный код:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
const int M=10, N=10;                  //Максимальное число строк и столбцов
матрицы
int A[M][N], R[M];                     //Декларирование матрицы A и вектора R
int i, j, k, m, S1, S2;                // Декларирование индексов и переменных
cout<<"Vvedite chislo strok i stolbcov matricu A :"<<endl;
cin>>m>>k;                             //Ввод реальных размеров исходной матрицы
cout<<"Vvedite postrochno elementu matricu A :"<<endl;
    for(i=0; i<m; i++)                 //Внешний цикл вывода элементов матрицы A
        for(j=0; j<k; j++)
        cin>>A[i][j];                  //Ввод элементов исходной матрицы A
        //Поиск минимального элемента каждой 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 четных
S2=0;                                  //и нечетных элементов вектора R
    //Операции по определению 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 элем-в вектора R
        else S1=S1+R[i];                //Определение суммы нечетных элементов
                                        //вектора R
    }                                   //Конец составного оператора в цикле
cout<<"Matrica A"<<endl;
    for(i=0; i<m; i++)                  //Внешний  цикл вывода элементов матрицы 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
```

Вопросы для самоконтроля.

1. Каким образом в программе на С++ объявляется двухмерный массив?
    1. **double A[0..2][0..3];**
    2. **double A[2, 3];**
    3. **double A[2][3];**
2. В памяти компьютера двухмерный массив располагается…
    1. непрерывно строками;
    2. непрерывно столбцами;,

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

1. Возможно;
2. Не возможно;
3. если **i>j**, а **k>m**.

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

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

1. В виде строки;
2. В виде столбца;
3. В виде матрицы.

5. Что нужно изменить в программе

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

чтобы массив **A[k][k]** был выведен на экран в виде строки?

1. Вместо **i** подставить **k**;
2. Убрать **<<endl**;
3. Вместо **j** подставить **k**.

6. Как будет выведен на экран двухмерный массив в программе на C++?

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

1. В виде строки;
2. В виде столбца;
3. В виде матрицы.

7. Как будет выведен на экран двухмерный массив в программе 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;
}
```

1. В виде строки;
2. В виде столбца;
3. В виде матрицы.

8. 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];
```

1. Первая строка матрицы, повторенная **n** раз;
2. Первый столбец матрицы, повторенный **n** раз;

3. Полная матрица.

9. Что нужно изменить в программе

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

для вывода массива **A[k][m]** на экран в виде матрицы?

1. Вместо **i** подставить **k**;
2. Добавить **cout<<endl;**
3. Вместо **j** подставить **m**.

10. Почему при работе с массивами в С++ целесообразно использовать оператор цикла **for**?

1. Оператор цикла **for** безошибочен;
2. Известен размер массива, т. е. число повторений цикла;
3. Оператор цикла **for** быстродействующий.

11. Какое действие выполняет оператор

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

?

1. Выводит на экран матрицу **B[i][j]**;
2. Выводит на экран матрицу **A[i][j]**;
3. Копирует матрицу **A[i][j]** в матрицу **B[i][j]**.

Лекция 12
Функции и их применение в среде Visual C++ 2010

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

Основные вопросы лекции:

1. Понятие, назначение и объявление функций в среде Visual C++ 2010.
2. Описание (программный код) функций в Visual C++ 2010.
3. Вызов функций в Visual C++ 2010.
4. Передача аргументов функции в Visual C++ 2010.
5. Области действия и видимости переменных в программах в среде Visual C++ 2010.
6. Функции и массивы в Visual C++ 2010

1. Понятие, назначение и объявление функций в среде 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, например, **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.

2. Описание (программный код) 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(int argc, _TCHAR* argv[])
{
Тело функции
}
```

В заглавии **Тип** перед именем функции 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 и имен параметров, разделенных запятыми. Функция может не иметь параметров, но круглые скобки необходимы всегда.

В списке параметров для каждого параметра должен быть указан тип. Например,

function (int x, int в, float z) - правильный список параметров;

function (int x, в, float z) - неправильный список параметров.

В теле функции обязательно должен присутствовать оператор **return** (возвратить) с параметром того же типа, что и возвращаемое значение.

Оператор **return** имеет два варианта использования.

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

2. Используется для возвращения 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 фигурной 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 программы.

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 и т. п.

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

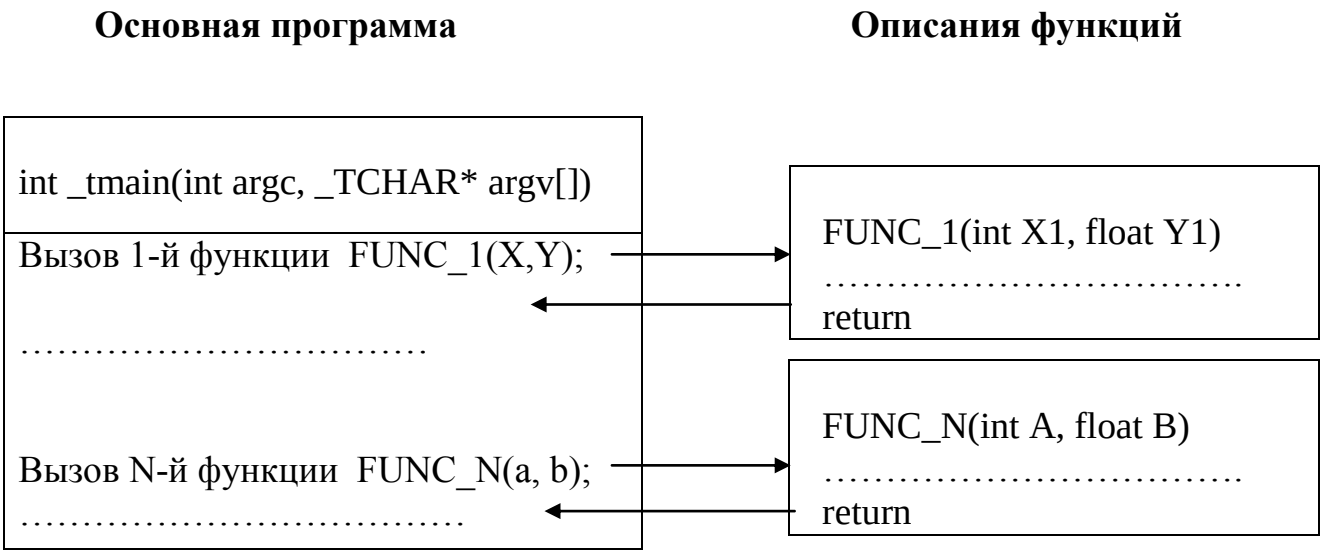


Рис. 12.1. Порядок вызова функций

Аргументы, которые указаны в заглавии функции, носят название **формальных**, например, в **FUNC(int X1, float Y1)** формальные параметры – **X1** и **Y1**.

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

В языке C++ есть особенность – все аргументы функции передаются по значению.

Например, при трансляции функции **float func(float x, float в)** в стеке выделяется место для ее формальных параметров. В это выделенное место заносятся значения фактических параметров, то есть значения параметров при вызове функции. Далее функция использует эти параметры (см. рис. 12.2) .

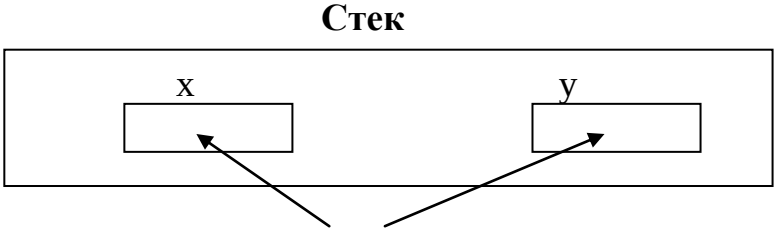


Рис. 12.2. Вызов функции **func (a,b)**

4. Передача аргументов функции в Visual C++ 2010

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

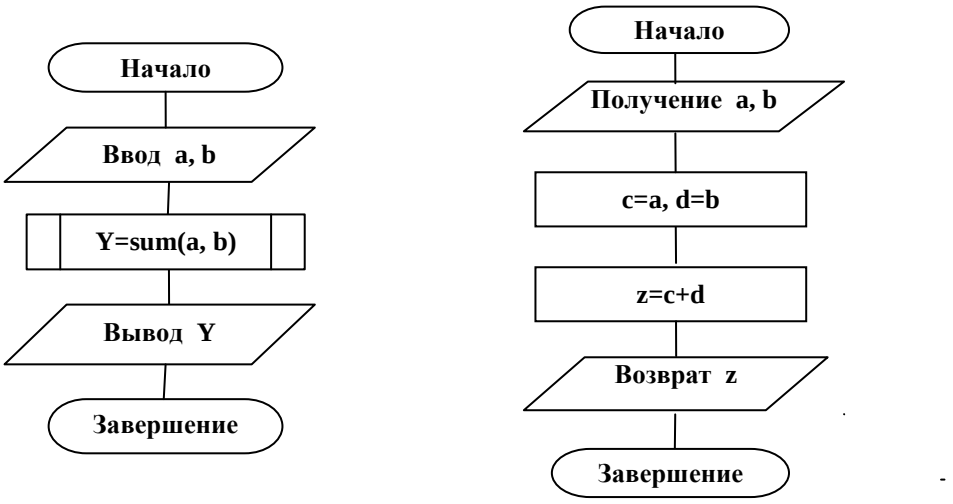
Когда происходит передача переменной-аргумента по значению, в функции создается локальная переменная с именем аргумента, в которую записывается его значение. Внутри функции может измениться значение этой переменной, но не самого аргумента.

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).

Пример 1. Разработать программу, вычисляющую **Y** по формуле **Y=a+b**, в которой расчет **Y** оформлен в виде функции.

Блок-схема алгоритма решения данной задачи приведена на рис. 12.3.



а) б)

Рис. 12.3. Блок-схема алгоритма вычисления **Y** по формуле **Y=a+b** с использованием функции:
а) основная программа; б) функция **sum(double c, double d)**

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

```
double sum(double c, double d);          //Объявление (прототип) функции
                                         //вычисления суммы
int _tmain(int argc, _TCHAR* argv[])
{
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
```

```
    cout<<endl<<"Y = "<<Y<<endl;        //Вывод значения Y
getch();
return 0;
}

double sum(double c, double d)          //Описание функции вычисления z=c+d
{
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=9.7

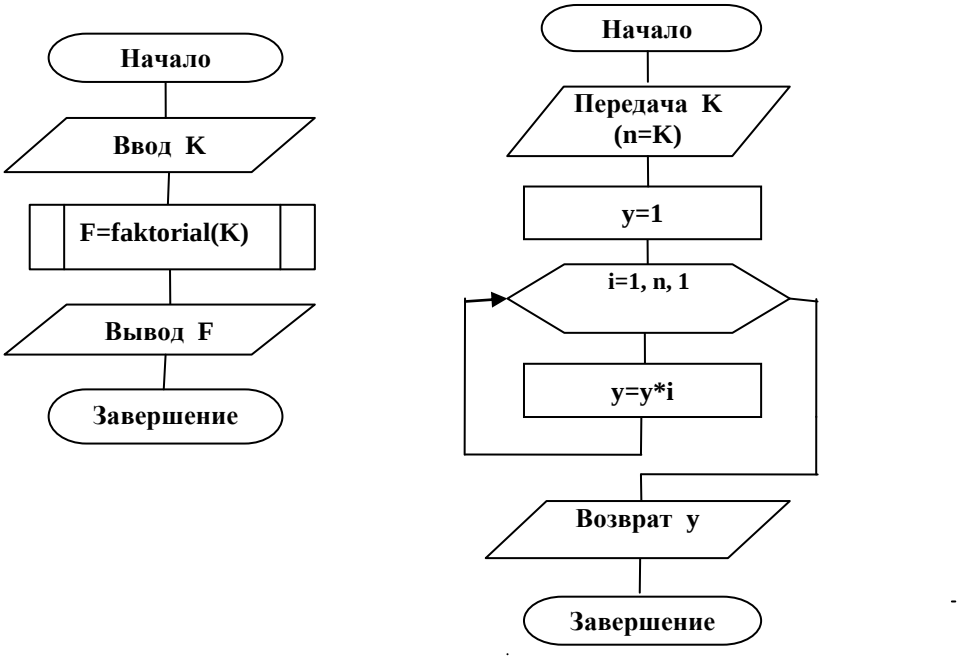
Пример 2. Разработать программу, использующую функцию вычисления факториала числа:

$$\mathbf{F = n!}\ .$$

Для составления программы данное выражение запишем следующим образом:

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

Блок-схема алгоритма решения данной задачи приведена на рис. 12.4.



а) б)

Рис. 12.4. Блок-схема алгоритма вычисления факториала числа
с использованием функции:
а) основная программа; б) функция **faktorial(int n)**

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

```
int faktorial(int  n);                    //Объявление (прототип) функции
                                          //вычисления факториала
int _tmain(int argc, _TCHAR* argv[])
{
int K;
cout<<"Vvedite K:"<<endl;
cin>>K;                                   //Ввод числа K
     F=faktorial(K);                      //Вызов функции, вычисляющей
факториал числа
     cout<<endl<<"Faktorial K= "<<K<<endl; //Вывод значения факториала числа
K
getch();
return 0;
}

int faktorial(int  n)                     //Описание функции вычисления
факториала числа
{
     int j, y;
     y=1;
          for(j=1; j<=n; j++)             //Цикл для вычисления  факториала числа
          y=y*j;
     return y;                            //Возврат значения  y  в основную
}
```

Результат выполнения программы следующий:

Vvedite K:

5

Faktorial K= 120

Пример 3. Разработать программу для вычисления числа соединений **R** из **N** элементов по **M** по формуле

$$\mathbf{R} = C_N^M = \frac{N!}{M!(N-M)!};$$

где расчет **R** производится с использованием функции.

Воспользуемся созданной в **Примере 2** функцией **factorial(int n)** для вычисления факториала числа и будем обращаться к этой функции непосредственно в формуле для расчета **R**:

```
int faktorial(int  n);                    //Объявление (прототип)  функции
                                          //вычисления факториала
int _tmain(int argc, _TCHAR* argv[])
{
int N,M,R;
cout<<"Vvedite N i M (N>M):"<<endl;
cin>>N>>M;                                //Ввод значений  N и M
     R=faktorial(N)/(faktorial(M)*faktorial(N-M));  //Формула для R (с трехкратным
```

```
                                        //вызовом функции faktorial)
    cout<<endl<<"R= "<<R<<endl;         //Вывод значения R
getch();
return 0;
}

int faktorial(int n)                    //Описание функции вычисления
                                        //факториала числа
{
int j, y;
y=1;
    for(j=1; j<=n; j++)
    y=y*j;
return y;                               //Возвращение в программу значения  y
}
```

Результат выполнения программы следующий:

Vvedite N и M:
8 4
R= 70

5. 5. 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, в котором она объявлена. Блок открывается и закрывается фигурными скобками. 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.

6. 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);

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

function (B, k);

Пример 4. Задан массив **A** из **N** произвольных чисел. 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:

```
double Max(double B[],int n);          //Объявление (прототип) функции поиска
макс. элемента
int _tmain(int argc, _TCHAR* argv[])
{
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 вывода исходного массива A
    cout<<A[i]<<' ' ;
cout<<endl<<"Max= "<<M<<endl;          //Вывод макс. элемента исходного массива
cout<<" Novui massiv "<<endl;
    for(i=0; i<k; i++)                 //Цикл для расчета и вывода элементов нового
                                       //массива
    cout<<A[i]/M<<' ';
cout<<endl;
getch();
return 0;
}
double Max(double B[],int n)           //Описание функции поиска максимального
```

```
                                        //элемента массива
{
    int j;
    double C=B[0];                      //Присваивание переменной C начального
знач-я (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 осн. программу значения  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
```

Пример 5. Задан массив **A** из **N** произвольных чисел. Разработать программу для нахождения максимального элемента этого массива и его номера, которые вывести на экран. Поиск максимального элемента массива и его номера оформить функцией, а сами переменные использовать как глобальные, т. е. “видимые” и в основной программе, и в описаниях функций.

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

```
double Max(double B[], int n);          //Объявление (прототип) функции Max()
double C;                               //Объявление глобальной переменной C
int k;                                  //Объявление глобальной  переменной k
int _tmain(int argc, _TCHAR* argv[])
{
const int N=50;                         //Объявление максимального размера
                                        //исходного массива
int i, m;                               //Объявление параметра цикла и
                                        //реального размера массива
double A[N];                            //Объявление исходного массива  A[N]
cout<<"Vvedite razmer massiva:"<<endl;
cin>>m;                                 //Ввод реального размера массива
cout<<endl<<"Vvedite massiv :"<<endl;
    for(i=0; i<m; i++) cin>>A[i];       //Ввод элементов исходного массива A[i]

    cout<<endl;
Max(A, m);                              //Обращение (вызов) к функции Max()
```

```
cout<<"Max= "<<C<<endl;          //Вывод макс. элемента массива A
                                 //(глобальная переменная)
cout<<" Nomer max "<<k<<endl;    //Вывод номера макс. элемента
                                 //(глобальная переменная)
getch();
return 0;
}

double Max(double B[], int n)    //Заголовок описания (кода) функции  Max
{
int j;                           //Объявление параметра цикла
C=B[0];                          //Глобальной переменной C присваиваем
                                 //первый элемент
    for(j=0; j<n;j++)            //Цикл для поиска максимального элемента
        if(B[j]>C)
        {
        C=B[j];                  //Определяем максимальный элемент
        k=j;                     //Номер макс. элемента заносим в глобальную
                                 //переменную k
        }
return 0;                        //Функция не возвращает никаких значений
}
```

После выполнения программы экран будет иметь следующий вид:

```
Uvedite razmer massiva :
8

Uvedite massiv :
11 23 15 7 8 12 4 9

Max= 23
 Nomer max 1
```

Вопросы для самоконтроля

1. При обращении к функции реальные и формальные параметры должны совпадать…
    1. По количеству и месту;
    2. По количеству, типу и месту;
    3. По количеству и типу.
2. Обращение к функции осуществляется…
    1. По заглавию функции;
    2. По имени функции;
    3. С помощью специального оператора.
3. Функции могут быть…
    1. Только с параметрами;
    2. Только без параметров;
    3. С параметрами и без параметров.

4. В программе используются две независимые функции **F1** и **F2**. В каком порядке они должны быть описаны в разделе прототипов?

1. Сначала **F1**, а затем **F2;**
2. Сначала **F2**, а затем **F1;**
3. В любом порядке.

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

1. Возможно;
2. Возможно в отдельных случаях;
3. Невозможно.

6**.** В программе описано две функции **F1** и **F2**. Возможно ли в них использовать одинаковые переменные?

1. Возможно;
2. Не возможно;
3. Возможно в отдельных случаях.

7. Отдельный блок программы целесообразно оформить функцией, если в нем есть…

1. Один результат;
2. Два и более результата;
3. Все равно сколько результатов.

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

1. Возможно;
2. Невозможно;
3. Возможно в отдельных случаях

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 С++?

1. Можно;
2. Нельзя;
3. Можно, когда выражение принимает определенное значение.

12. Параметры функции могут быть переменные, которые…

1. Объявлены в функции;
2. Объявлены вне функции;
3. Все равно где.

13. Укажите правильную запись заглавия функции:

1. **func(int x, в, float z)**;

2. **func(x, int в, float z);**
3. **func(int x, int в, float z);**

14. Оператор **return**…
    1. Вызывает выход из функции;
    2. Используется для возвращения значения функции;
    3. И первое, и второе.
15. В теле функции оператор **return**…
    1. Может отсутствовать;
    2. Должен быть обязательно;
    3. Все зависит от назначения функции.
16. Когда в теле функции отсутствует оператор **return**, выход из функции осуществляется…
    1. Когда выполнится последний оператор функции;
    2. В данном случае возникнет зависание программы ;
    3. Оператор **return** должен быть в теле функции обязательно.
17. Если функция не имеет аргументов, то в прототипе такой функции необходимо…
    1. Ничего не писать в скобках функции;
    2. В скобках записать служебное слово int;
    3. В скобках записать служебное слово void.
18. Локальная переменная – это переменная, которая объявлена…
    1. В теле данной функции;
    2. Вне данной функции;
    3. В качестве параметра в заглавии функции.
19. В заглавии функции записываются…
    1. Формальные параметры;
    2. Фактические параметры;
    3. Глобальные переменные.
20. При обращении к функции реальные параметры должны совпадать с формальными…
    1. По месту;
    2. По месту и типу;
    3. По месту, типу и количеству.

Лекции 13-14
Символьный массив или строка в среде Visual C++ 2010

Цель лекции. Исследовать особенности обработки строк (массивов символьных переменных) 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.
5. Функции превращения типов.

1. Описание строк в среде Visual C++ 2010

В языке C++ массив типа **char** - это одномерный массив, состоящий из символов:

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 С” (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: **. ;** и т. п.
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 двойные 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, хотя для размещения этой строки необходимо 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 сам определит необходимый размер памяти под этот массив.

2. Ввод и вывод строк в среде Visual C++ 2010

В языке C++ при работе со строками, как и при работе с числовыми переменными, можно использовать операторы ввода в поток >> и вывода из потока <<. Но оператор ввода >> игнорирует пробелы, которые вводятся.

Пример 1. В качестве примера использования операторов потоковых ввода и вывода символьных массивов исследуем следующую программу:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
char stroka[30], A[50];                    //Объявление символьных переменных
cout<<"Vvedite ctroku  <  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;
}
```

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::
ssssddddgggg
Vu vveli novuyu stroku: ssssddddgggg
A0= s
A8= g
```

При вводе символов с пробелами, последние игнорируются операторами ввода >> и вывода <<. Поэтому при работе со строками вместо этих операторов целесообразней использовать следующую функцию:

getline(ИмяСимвольнойПеременной, РазмерСимвольнойПеременной);

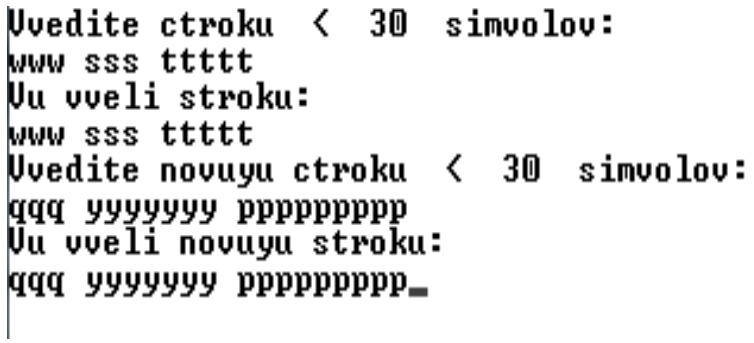
где **ИмяПеременной** указывает на строку, в которую осуществляется ввод; **РазмерПеременной** – число символов, подлежащих вводу.

Реализация применения этого оператора описана в нижеследующей программе.

Пример 2. Исследуем использование функции **getline()**.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
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 следующий вид:



При использовании функции **getline() РазмерПеременной** 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 27 символами введется строка из 20 символов. Остальные символы будут отброшены.

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.

3.1. Определение длины строки

Очень 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).

Пример 3. Исследуем использование функции **strlen()**.

```
#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) для
```

```
                                        //определения количества символов в
                                        //массиве A
cout<<endl<<"k= "<<k<<endl;             //Вывод переменной k (кол.
символов)
                                        //в массиве A)
getch();
return 0;
}
```

Вид экрана после работы программы:

```
Vvedite ctroku  <  30  simvolov:
wwwwwwww  RRRRRRRR  xxxxxxxxx
Vu vveli stroku:
wwwwwwww  RRRRRRRR  xxxxxxx
k= 29
```

3.2. Копирование строк

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

Функция **strcpy(S1,S2)** используется для побайтного копирования строки **S2** в строку **S1**. Копирование прекращается при достижении нулевого байта. Поэтому длина строки **S1** должна быть достаточно большой, чтобы поместилась строка **S2.**

Пример 4. Исследуем использование функции **strcpy()**:

```
#include <string.h>                     //Добавление библиотеч. файла для
                                        //работы со строками
#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);                      //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_
```

Функция **ctrncpy()** отличается от функции **ctrcpy()** тем, что включает еще один параметр. Он указывает количество символов, которые необходимо копировать из строки **S2** в строку **S1**. Функция имеет вид:

ctrncpy (S1, S2, n);

где **n** – количество символов (целое без знака).

Если длина **S1** меньше длины **S2**, то происходит урезание символов.

Пример 5. Исследуем 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<<"S1= "<<A<<endl;                  //Вывод символьной переменной  A
cout<<"S2= "<<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 **S2** в строку **S1** будут скопированы 4 первых символа и размещены в начале короткой строки **S2**.

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 (S1, S2) и strncat (S1, S2, n);

Функция **strcat (S1, S2)** присоединяет строку **S2** к строке **S1** и помещает ее в массив, где находилась строка **S1**. Строка **S2** не изменяется. Вновь полученная строка S1 автоматически завершается нулевым байтом.

Пример 6. Исследуем использование функции **strcat()**:

```
#include <string.h>
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
char A[30], B[30];
strcpy(A, "Hello, ");                  //Копирование строки "Hello, " в строку A
strcpy(B, "World!");                   //Копирование строки "World!" в строку 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 (S1, S2, 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 k

3.3. Сравнение строк

В библиотеке функций **string.h** есть функции, выполняющие посимвольное сравнение двух строк.

Функция **strcmp (S1, S2)** сравнивает строки **S1** и **S2**. После сравнения строк данная функция возвращает одно из следующих значений:

<0 – если строка **S1** меньше чем **S2**;

=0 – если строки эквивалентные;

>0 – если **S1** больше, чем **S2**.

Эта функция проводит сравнение строк, различая прописные и строчные буквы, например:

```
char S1[]="ааааааавв";
char S2[]="БББББББГГ";
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 **S1** меньше строки **S2** по той причине, что прописные буквы имеют код символов меньше чем те же строчные буквы.

На экран будет выведено: **k= -1** .

Функция **stricmp (S1, S2)** сравнивает строки **S1** и **S2,** 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:

- функции превращения строк (превращение элементов символьной строки из одного регистра в другой);
- функции обращения строк (меняет порядок прохождения символов в строке на обратную);
- функции поиска символов (одного или группы символов в строке).

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";
B+="puter";

присвоят переменным **C** и **B** строку "**Computer**".

Аналогично, к строке можно приписать отдельный символ: **D+="p";**

5. Функции превращения типов

Функции превращения типа используются для превращения чисел, введенных в виде символьных строк, в числовое представление, выполнение определенных математических операций над ними и обратное превращение в строку символов.

Эти функции размещаются в заглавном файле **stdlib.h**.

Функции **atof()**, **atoi()**, **atol()** – превращают строку символов соответственно в число типа **float, int, long**.

Функции **ecvt(), fcvt(), gcvt()** – превращают число с плавающей точкой типа **double** в строку символов.

При использовании функции **ecvt()** десятичная точка и знак числа не включаются в полученную строку.

При использовании функции **fcvt()** округляет приобретенное значение к заданному числу цифр.

При использовании функции **gcvt()** включает символ десятичной точки.

Функции **itoa(), ltoa(), ultoa()** превращают числа типа **int, long и unsigned long** в строку символов.

Функции **strtod() и strtol()** превращают строку символов соответственно в число типа **double** и **long**.

Рассмотрим использование некоторых из этих функций.

Функции

int atoi(const char *ptr);
long atoll(const char *ptr);

превращают строку символов, на которую указывает указатель **ptr** в число типа **int**.

Функция **gcvt** имеет прототип

char *gcvt(double val, int sig, char *buf);

и превращает число **val** типа **double** в строку с помещением ее в буфер **buf** (**int sig** – число цифр, которые подлежат превращению).

Если число цифр, которые подлежат превращению, меньше числа, указанного в **sig**, то в преобразованном числе указывается знак и десятичная точка. Младшие разряды дробной части отбрасываются. Иначе - число превращается в экспоненциальную форму.

Выводы.

В языке С++ строка представляется как одномерный массив, элементы которого имеют тип **char**.

Следовательно, символьная строка – это одномерный массив типа **char**, заканчивающийся нулевым байтом.

Вопросы для самоконтроля.

1. В программе на С++ определен массив **char A[11]**. Это означает, что строка содержит:
    1. 10 символов;
    2. 11 символов;
    3. 12 символов.
2. В языке С++ для копирования строк используется функция:
    1. **strlen( )**;
    2. **strcpy( )**;
    3. **strcat( )**.

3 В языке С++ конкатенация строк – это:
    1. Копирование одной строки в другой;
    2. Сравнение двух строк;
    3. Присоединение одной строки к другому.

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
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 языке 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.

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**:

Пример 1. Рассмотрим задачу определения адреса двух переменных **A** и **B**.

Проанализируйте и выполните программный код, приведенный ниже.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
double A=23.1;
double B=57.88;
cout<<"Znachenie A= "<<A<<endl;
cout<<"Adres A= "<<&A<<endl;
cout<<"Znachenie B= "<<B<<endl;
cout<<"Adres B= "<<&B<<endl;
getch();
return 0;
}
```

В результате должен быть получен следующий результат:

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

Очевидно, что оператор **&B** дает возможность определить адрес переменной.

При исследовании программ по этой теме следует учитывать, что адреса переменных будут отличаться для различных компьютеров.

Адреса переменных записаны в шестнадцатеричной системе счисления.

Адреса локальных переменных размещаются в стеке. Поэтому их адреса следуют в обратном порядке (стек растет в направлении младших адресов). Разница в адресах **A** и **B** всегда будет одинаковая и при 4-х байтовом представлении чисел типа **int** составит 4 байта.

2. Понятие указателя в Visual C++ 2010

В языке C++ есть возможность осуществлять непосредственный доступ к 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.

На рис. 15.1 показана взаимосвязь между адресом переменной и указателем на этот адрес.

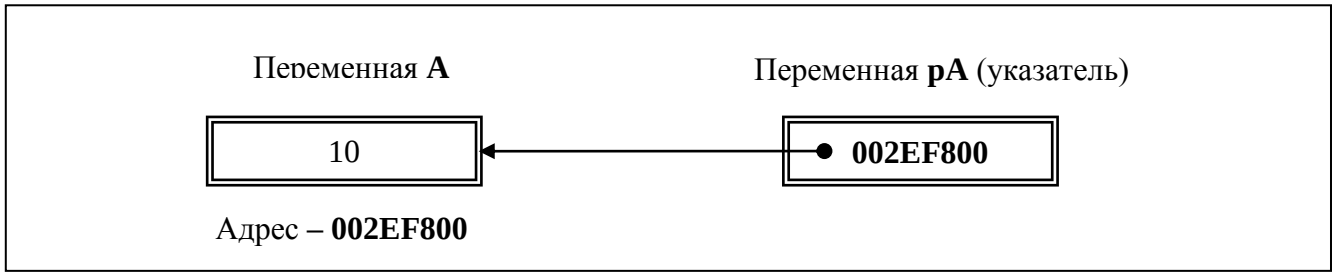


Рис. 15.1. Взаимосвязь между адресом переменной и указателем на этот адрес.

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 присвоен только значение **адреса переменной**, а не значение самой переменной.

Рассмотрим пример объявления и инициализации указателя.

Пример 2. Ниже приведена программа, 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(int argc, _TCHAR* argv[])
{
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<<"   &A= "<<&A<<endl;    //Вывод переменной A и адреса A
cout<<"     uA = "<<uA<<endl;           //Вывод указателя uA
cout<<"B= "<<B<<"   &B= "<<&B<<endl;    //Вывод переменной B и адреса B
cout<<"     uB = "<< uB <<endl;         //Вывод указателя uB
getch();
return 0;
}
```

Результат выполнения программы следующий:

```
A= 57.97   &A= 001CFB4C
             uA = 001CFB4C
B= 340   &B= 001CFB30
             uB = 001CFB30
```

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

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.

Пример 3. Программа, 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(int argc, _TCHAR* argv[])
{
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 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 выдана следующая информация:

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

3. Операции с указателями.

Язык С++ 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 (&).

Операция присваивания.

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

Пример 4. Программа, в которой применяется операциях 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 внесли погрешностей и значение разыменованного указателя **g** совпадает со значением переменной **x.**

Сравнение указателей.

Указатели можно сравнивать, применяя все 6 операций сравнения.

P1 = = P2 – сравнение на равенство;

P1 != P2 – сравнение на неравенство;

P1 < P2 – меньше;

P1 < = P2 – меньше или равно;

P1 > P2 – больше;

P1 > = P2 – больше или равно.

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

Операция sizeof

Как и к любой переменной или типу данных, к указателям можно применять операцию определения размера – **sizeof**. В современных ПК указатель имеет размер 4 байта (32 двоичных разряда) и может адресовать 2^{32} **= 4 Гбайт** памяти.

К указателям можно применять не только операцию **sizeof** но и одноименную функцию. Исследуем этот прием в нижеследующей программе.

Пример 5. Рассмотрим случай применения операции определения размера **sizeof**.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
```

```
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
 {
     unsigned long A=546213;
     bool T=false;
     unsigned long *pA=&A;
     bool *pT=&T;
     cout<<sizeof pA<<endl;
     cout<<sizeof (pA)<<endl;
     cout<<sizeof pT<<endl;
     cout<<sizeof (pT)<<endl;
getch();
return 0;
}
```

Вид экрана после выполнения программы:

```
4
4
4
4
```

4. Указатели и массивы

В языке С++ 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 С++ **n**-й элемент массива определяется как

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

Пример 7. Исследуем программу, которая вводит в память в клавиатуры целочисленный массив, вычисляет сумму его элементов и выводит на экран эту информацию с использованием указателей.

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
 {
     const int M=50;
     int i, m, S, Mas[M];
     int *pMas;
     pMas=Mas;
```

```
        cout<<"Vvedite razmer massiva m:"<<endl;
        cin>>m;
        cout<<"Vvedite massiv:"<<endl;
        for (i=0; i<m; i++, pMas++)
              cin>>*pMas;
        cout<<endl;
        S=0;
        pMas=pMas-m;
        for (i=0; i<m; i++, pMas++)
              S=S+*pMas;
              pMas=pMas-m;
        for (i=0; i<m; i++, pMas++)
              cout<<*pMas<<' ';
        cout<<endl<<"S= "<<S<<endl;
  getch();
  return 0;
  }
```

Вид экрана после выполнения программы:

```
Vvedite razmer massiva m:
7
Vvedite massiv:
1 2 3 4 5 6 7

1 2 3 4 5 6 7
S= 28
```

Вывод. В языке С++ 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;**?
   1. Значение **A**, то есть 300;
   2. Адреса ячеек, в которых записано значение **A;**
   3. Сообщение об ошибке.
2. Укажите правильное объявление указателя в С++.
   1. ***double pA;**
   2. **double *pA;**

3. **double pA*;**

3. Возможна ли следующая инициализация указателя: **char A=”yes”; char *pA=&A;**?

1. Возможна;
2. Невозможна;
3. Такой конструкции в С++ нет.

4. Что означает оператор **double *pA=&A;**?

1. Инициализация переменной **A;**
2. Объявление указателя **pA**;
3. Инициализация указателя **pA** и присваивание ему адреса переменной **A**.

5. Что такое указатель в языке С++?

1. Адрес некоторой переменной или функции;
2. Переменная, содержащая адрес некоторой переменной или функции;
3. Стандартная функция.

6. Как объявляется указатель с именем **pA** в языке С++?

1. **int** ***pA**; 2. **int** «**pA**; 3. **int &A**.

7. Как обозначается операция взятия адреса переменной **A** в языке С++?

1. ***pA**; 2. «**pA**; 3. **&A**.

8. Как обозначается операция разыменования указателя **pA** в языке С++?

1. ***pA**; 2. «**pA**; 3. **&pA**.

9. Что означает операция разыменования указателя в языке С++?

1. Получение адреса переменной в указателе;
2. Получение значения переменной, записанной по адресу, который находится в указателе;
3. Запись переменной по адресу, который находится в указателе.

10. Какое значение примет **Y** в программе **double X=10.1; double *pA; pA=&X; Y=*pA;**

1. **Y**=10.1;
2. В переменную **Y** запишется адрес, по которому находится значение **X**;
3. Будет выдано сообщение об ошибке с указанием на последнюю строку.

11. Можно ли в языке С++ выполнять арифметические операции над указателями?

1. Можно; 2. Нельзя; 3. Только операцию присваивания.

12. Что будет выведено на экран дисплея после выполнения программы **int *pA; for (i=0; i<100; i++) cout<<*(pA+i)<<” “;**?

1. Значения элементов какого-то массива;
2. Значения указателей;
3. Такую конструкцию в языке С++ использовать нельзя.

13. Укажите на возможность такого объявления указателя **int **ppA;**

1. Возможно;
2. Невозможно;
3. Все зависит от содержания программы.

14. На сколько байтов изменится значение **pC** в программе **int C[20]; int *pC=&C; pC++;**?

1. Не изменится; 2. Увеличится на 4 байта**;** 3. Уменьшится на 4 байта

Лекции 16
Ввод и вывод данных с использованием специальных файлов 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.
3. Запись данных в файл и чтение из файла в Visual C++ 2010.
4. Позиционирование файла в Visual C++ 2010.

1. Файлы и потоки ввода и вывода данных в Visual C++ 2010

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, используемые 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.

2. Создание, открытие и закрытие файлов в Visual C++ 2010

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 *F1;

В файле **stdio.h** определены функции для работы с файлами. Основные из них приведены в таблице 16.1.

Таблица 16.1. 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» при достижении конца 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 открываемогофайла.

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.

Пример 1. Исследовать программу для записи переменной **A** в открываемый файл, чтения значения переменной A из этого файла и вывода значения **A** на печать.

```
#include <stdio.h>                  //Подключение библиотеки программ
                                    //файлового ввода-вывода
#include <stdlib.h>                 //Подключение библиотеки стандартных операций
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
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", "w"))==NULL)   //Стандартная процедура
                                                  //создания  и открытия
    {                               //файла для записи данных  с одно-
    cout<<"Ne mogu otkrit F1"<<endl; //временной проверкой результата этих
    exit(1);                        //действий и сообщением в случае 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. Не открытие 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 (F1);

Отлаженная программа дает следующий результат:

Таким образом, значение переменной было верно записано в специально открытый файл **F1** и прочитано из этого файла.

3. Запись (чтение) данных в файл

Запись данных в поток проводится функцией **fwrite()**, а чтение - функцией **fread()**. Эти функции имеют следующие прототипы:

unsigned fread (void *buf, int bytes, int c, FILE *fptr);

и **unsigned fwrite (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.

В Примере 1 уже были применены эти операторы для ввода в файл и вывода из него одной переменной, для чего используется буфер памяти компьютера. Исследуем их применение

Пример 2. В памяти ПК есть массив из 12 произвольных чисел. Записать этот массив в файл **D:\\Inform.dat**. 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(int argc, _TCHAR* argv[])
{
```

```
const int N=30;                          //Максимальный размер массива
int A[N];                                //Объявление массива A[N]
int i,n;                                 //Параметр цикла i и размер массива n
int S;                                   //Объявление переменной S в качестве
                                         //буфера обмена
int *pS;                                 //Указатель pS на буфер обмена S
pS=&S;                                   //Присваивание адреса переменной S
                                         //указателю pS
cout<<"Vvedite razmer n:"<<endl;
cin>>n;                                  //Ввод реального размера массива A[N]
cout<<"Vvedite massiv A:"<<endl;
      for(i=0;i<n;i++)                   //Цикл для ввода массива A[N]
      cin>>A[i];                         //Ввод i-го элемента массива A[N]
cout<<" Massiv A: "<<endl;
      for(i=0;i<n;i++)                   //Цикл для вывода массива A[N]
      cout<<A[i]<<" ";                   //Вывод i-го элемента массива A[N]
FILE*F1;                                 //Объявление переменной–указателя файла
if((F1=fopen("D:\\Inform.dat", "w"))==NULL)      //Открытие файла для записи с
                                                 //условием вывода на экран
{                                        // предупреждения в случае сбоя при открытии
cout<<"Ne mogu otkrit F1"<<endl;
exit(1);
}
      for(i=0;i<n;i++)                   //Цикл для записи в открытый файл массива
A[N]
      {
      S=A[i];                            //Запись i-го элемента массива A[N] в буфер  S
      fwrite(pS,4,1,F1);                 //Запись  A[i] из буфера в файл
      }
fclose(F1);                              //Закрытие файла
int B[N];                                // Объявление нового массива B[N]
FILE*F2;                                 //Объявление переменной–указателя файла
if((F2=fopen("D:\\Inform.dat","r"))==NULL)       //Открытие файла для чтения с
                                         //условием вывода на экран предупреждения
{                                        //в случае сбоя при открытии
cout<<"Ne mogu otkrit F2"<<endl;
exit(1);
}
      for(i=0;i<n;i++)                   //Цикл для чтения из файла элем-в массива B
      {
      fread(pS,4,1,F2);
      B[i]=S;                            //Запись единицы данных из буфера S в i-й
                                         //элемент массива  B
      }
fclose(F2);                              //Закрытие файла
cout<<endl<<" New massiv B:"<<endl;
```

```
    for(i=0;i<n;i++)                    //Цикл для вывода массива B[N]
    cout<<B[i]<<" ";                    //Вывод на экран i-го элемента массива  B
cout<<endl;
getch();
return 0;
}
```

Вид экрана после выполнения программы:

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.

Пример 3. Исследуем программу чтения данных из файла **C:\\Inform.dat** с использованием функции **feof()**.

Такая программа 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
cout<<"Vvedite  massiv 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<<"Ne mogu otkrit file1!"<<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 k;
float B[9];                             //Объявление массива B
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<<"Ne mogu otkrit file2!"<<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
      B[i]=S;                                    //Запись из буфера S в массив
      i+=1;                                      //Счет количества прочит. чисел из файла
      }
k=i-1;                                           //Определение количества прочит. чисел
fclose(F2);                                      //Закрытие файла F2
cout<<"Massiv B:"<<endl;
for(i=0; i<k; i++) cout<<B[i]<<' ';              //Вывод массива на экран
cout<<endl;
}
```

Вид экрана после выполнения программы:

```
Vvedite  massiv A:
1 2 3 4 5 6 7 8 9
Massiv B:
1 2 3 4 5 6 7 8 9
```

4. 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 (F2);
```

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.

Точка отсчета	Значение **mode**
1. Начало файла	0
2. Текущая позиция	1
3. Конец файла	2

Пример 4. В файле **C:\\Inform.dat** записано 9 чисел: 1, 2, 3, 4, 5, 6, 7, 8 и 9. Необходимо прочитать данные из этого файла, начиная с 3-го числа (то есть с 3), 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 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[])
{
const int n=30;                          //Максимальний размер массива
float S *pS;                             //Объявление буфера обмена и указателя
                                         //на буфер
float Mas1[n], Sum;                      //Объявление массива и суммы
int i,k;
pS=&S;                                   //Присваивание адреса буфера указателю
                ё                        //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<<"Ne mogu otkrit file!"<<endl;
     exit(1);
     }
i=0;
fseek(F2, 8, 0);                         //Установка указателя позиции файла
                                         //через 8 байтов с начала файла
     while(!feof(F2))                    //Чтение данных из файла, пока не
                                         //наступит конец файла
     {
     fread(pS,  4, 1, F2);               //Чтение порции из 4-х байтов в буфер S
     Mas1[i]=S;                          //Запись из буфера S в массив
     i+=1;                               //счет количества прочитанных чисел
     }
k=i-1;                                   //Определение количества прочитанных
                                         //чисел
fclose(F2);                              //Закрытие файла
```

```
cout<<endl;
for(i=0; i<k; i++) cout<<Mas1[i]<<'  ';      //Вывод массива на экран
cout<<endl;
Sum=0;
for(i=0; i<k; i++) Sum=Sum+Mas1[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;                                       //Пересылка суммы 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", "r+b"))==NULL)  //Открытие файла C:\\Inform.dat
                                             //в режиме чтения и записи
      {
      cout<<" Ne mogu otkrit file 2!"<<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
      Mas1[i]=S;                             //Запись из буфера S в массив
      i+=1;                                  //Счет количества прочит. чисел из файла
      }
k=i-1;                                       //Счет количества прочит. чисел из файла
fclose(F4);                                  //Закрытие файла F4
cout<<endl;
for(i=0; i<k; i++) cout<<Mas1[i]<<'  ';      //Вывод массива на экран
cout<<endl;
}
```

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 С++ 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. Для ввода данных в файл необходимо следующее объявление:
    1. **ofstream name_file**;
    2. **ofstream (“filename.txt”)**;
    3. **ofstream name_file (“filename.txt”)**;
2. Чтобы выполнить операцию вывода данных из файла необходимо следующее объявление:
    1. **ifstream name_file (“filename.dat”);**
    2. **iostream name_file (“filename.dat”);**
    3. **fstream name_file (“filename.dat”);**
3. При чтении данных из файла конец файла определяется функцией:
    1. **eol()**; 2. **eof()**; 3. **eok()**.
4. Функция **eof()** возвращает значение 0, когда…
    1. Конец файла еще не наступил;
    2. Конец файла наступил;
    3. Эта функция ничего не возвращает.
5. Для определения конца файла записан оператор **while (! name_file.eof()).** Цикл будет выполняться…
    1. Пока функция **eof()** возвращает ложь (0);
    2. Когда функция **eof()** возвращает истину (1);
    3. Такую конструкцию использовать нельзя.
6. При завершении работы с файлом его нужно закрыть функцией:
    1. **close();**
    2. **close.file_name;**
    3. **file_name.close().**
7. При чтении массивов или структур из файла используется функция:
    1. **read;**
    2. **fread;**
    3. **ifread.**
8. Из какого файла будет прочитана информация при выполнении оператора **ifstream koord (“koord1.dat”);**?
    1. **koord**;
    2. **koord1**;
    3. Здесь имя файла не указано.
9. Функция **write** используется для…
    1. Ввода массива (структуры) в файл;
    2. Вывода массива (структуры) из файла;
    3. Вывода массива (структуры) на экран.