КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Программное Обеспечение Компьютерных Систем»

СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Для студентов по направлению 710400 «Программная инженерия» всех форм обучения

Бишкек - 2012

РАССМОТРЕНО

На заседании кафедры

«Программное обеспечение

компьютерных систем»

Прот. №14 от 28.02.2012 г.

ОДОБРЕНО

Учебно-методической комиссией

Прот. № от . . 2012 г.

УДК 004.432

Составитель – МАКИЕВА З.Д.

Структурное программирование: методические указания к выполнению лабораторных работ /Кырг. гос. техн. ун-т, Бишкек, 2012, 63 с.

В данных методических указаниях рассматривается структурное программирование на языке С++: создание проекта С++ в Visual Studio 2010, ввод и вывод в С++, управляющие структуры: структуры выбора (операторы if, if/else: switch), структуры повторения (операторы for, while, do/while), объявление и использование одномерных, двумерных и символьных массивов.

Далее рассматривается модульное программирование, указатели, виды памяти, области видимости переменных, локальные и глобальные переменные, работа с файлами в С++ и форматированный вывод. А также дается введение в понятия абстрактных типов данных (структур), передачи структур в функции.

Представлены теоретические сведения языка С++ с примерами разработанных программ, также приведены задания на составление программ для самостоятельной работы. Язык С++ предлагается для изучения по дисциплине “Структурное программирование ”.

Библиогр. 10 названий.

Рецензент канд. ф-м.наук, доцент Кыдыралиев Н.Н.

Введение

Устройство ЭВМ основано на принципах двоичной арифметики, где для представления чисел используются всего две цифры - 0 и 1. Программирование в кодах ЭВМ требует досконального знания системы команд машины и большого внимания, кроме того, процесс программирования в кодах малоэффективен.

Оптимизация программирования в двоичных кодах заключалась в разработке специальной системы кодирования двоичных машинных команд многобуквенными мнемоническими сокращениями. Такое программирование удобнее для программиста, но вместе с тем текст подобной программы становится абсолютно непонятным вычислительной машине и требует специальной программы-переводчика (или компилятора), которая бы заменяла мнемонический код исходной двоичной командой. С момента реализации этой идеи кодирование становится программированием.

Языки, которые требуют предварительного перевода, называются языками высокого уровня. Эти языки более близки к естественному языку. Использование языков высокого уровня значительно повышает эффективность программирования по сравнению с обычным кодированием. В настоящее время существует высокая мотивация к изучению языка программирования высокого уровня C++, так как он реализует передовые принципы программирования и широко используется во всем мире. В 1972 году Денисом Ритчи был создан язык С, который первоначально использовался для разработки ОС Unix. Язык С не зависит от аппаратных средств и позволяет создавать программы переносимые на большинство компьютеров, что привело к его широкому применению. В начале 80-х Бьярн Страуструп разработал расширенный язык С, т.е. C++, который позволяет программировать в стиле языка С и в стиле объектно-ориентированного программирования.

Решение задачи на ЭВМ состоит из следующих этапов:

1) постановка задачи;
2) выбор численного метода решения;
3) разработка алгоритма;
4) программирование алгоритма;
5) тестирование и отладка программы;
6) решение задачи на ЭВМ.

Постановка задачи определяет цель решения задачи, раскрывает её содержание. Задача формируется на уровне профессиональных понятий, должна быть корректной и понятной исполнителю. Ошибка в постановке задачи обнаруживается на последующих этапах и приводит к тому, что работа по подготовке задачи к решению должна будет начаться с самого начала. Для большинства задач на этом этапа разрабатывается математическая модель задачи.

Математическая модель – специальная форма описания задачи использующая язык математики. Математическая модель задаётся в виде уравнений или формул, необходимых для решения задачи. Кроме математической мо-

дели, на этапе постановки определяется перечень исходных данных, перечень результатов, начальные условия, точность вычисления.

Выбор численного метода: одна и та же задача может быть решена с помощью различных численных методов. Выбор метода должен определяться такими факторами, как точность результатов решения, время решения. Для простых задач, данный этап может отсутствовать. Например: Если в задаче требуется вычислить интеграл, может быть выбран для численного интегрирования метод прямоугольников, метод Симпсона или метод трапеции.

Алгоритм – конечная последовательность предписаний (правил), однозначно определяющая процесс преобразования исходных и промежуточных данных в результат решения задачи. При разработке алгоритма математическая модель и выбранный численный метод являются основой для определения последовательности действий.

Алгоритм должен обладать следующими свойствами:

а) массовость –алгоритм должен решать не одну, а целый класс задач;

б) детерминантность – однозначность выполняемых действий, т.е. промежуточные и окончательные результаты разных пользователей должны быть одинаковыми при одинаковых исходных данных;

в) результативность - алгоритм должен обеспечивать получение результата после конечного числа шагов.

Алгоритмы классифицируются на:

— Алгоритмы линейной структуры (последовательные алгоритмы), в которых все действия выполняются последовательно друг за другом.

— Алгоритмы разветвляющейся структуры, в которых в зависимости от выполнения логического условия, процесс пойдет по одной из 2-х ветвей.

— Алгоритмы циклической структуры, содержащие многократно выполняемые участки вычислительного процесса, называемые циклом. Их использование позволяет существенно сократить схему алгоритма.

— Алгоритмы со структурой вложенных циклов, содержащие цикл, внутри которого размещены один или несколько других циклов.

— Алгоритм смешанной структуры, содержащий линейные, разветвляющиеся и циклические структуры.

При создании программ следует применять принципы **структурного программирования.** Структурное программирование позволяет создавать программы более простые для понимания, для проверки, отладки и модификации, чем неструктурированные. При изучении материалов настоящих указаний мы будем учиться создавать структурированные программы.

Рисунок А суммирует сведения по управляющим структурам в C++. Малые окружности использованы, чтобы отметить точки единственного входа и единственного выхода каждой структуры. Произвольное соединение отдельных символов блок-схем может привести к неструктурированным программам. Профессиональное программирование заключается в выборе комбинаций символов, соответствующих ограниченному множеству управляющих структур, и в построении структурированных программ соответствую-

щим комбинированием управляющих структур двумя простыми способами. Для упрощения используются только структуры, имеющие только одну точку входа и одну точку выхода. Это упрощает формирование структурированных программ последовательным соединением управляющих структур, т.е. управляющие структуры просто размещается в программе одна за другой. Такой способ соединения называется пакетированием управляющих структур.

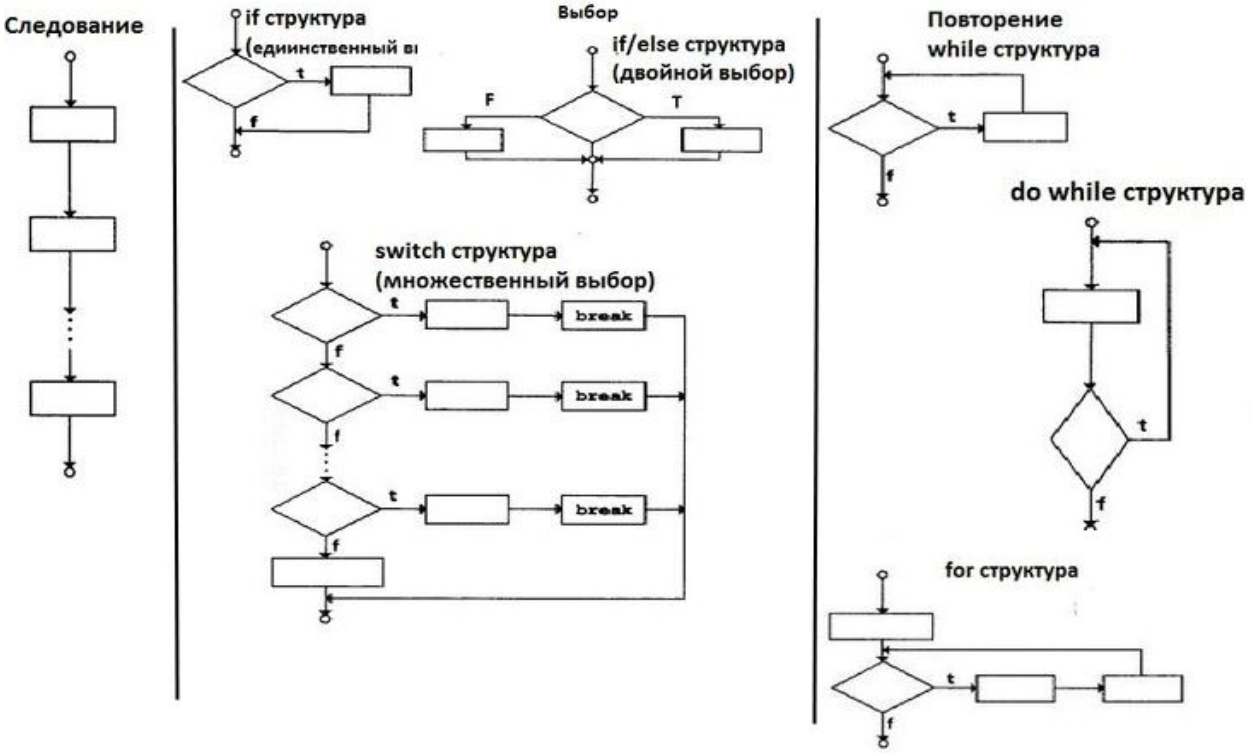


Рис. А. Структуры следования, выбора и повторения с одним входом и одним выходом

На рис. Б приведены правила формирования структурированных программ. Они предполагают, что символ прямоугольника на блок-схеме может использоваться для указания любых действий.

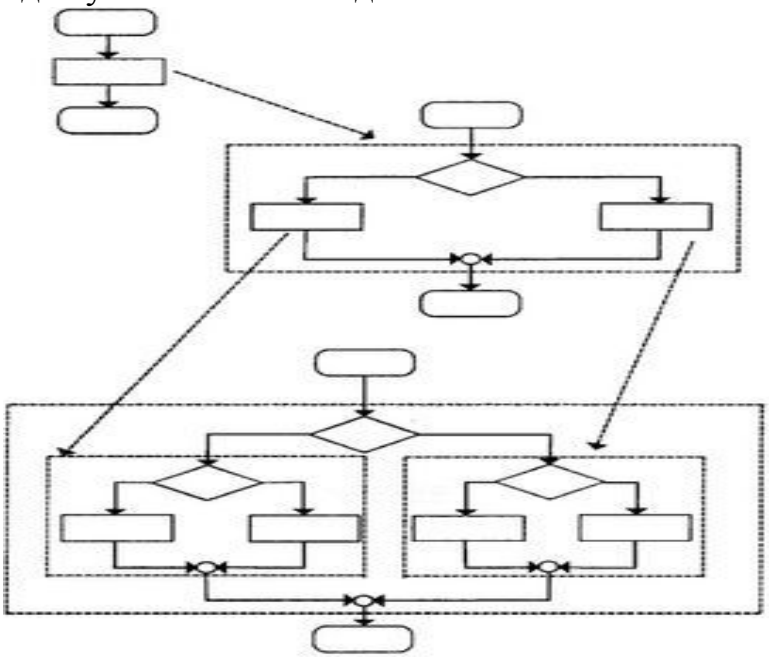


Рис. Б. Правила формирования структурированных программ.

Лабораторная работа №1.

Начало работы в среде «Microsoft Visual Studio 2010»

Выполнить: Зайдите в меню «ПУСК» на панели инструментов и из списка «Все программы» выберите пункт «**Microsoft Visual Studio 2010**» и в следующем выпадающем меню – пункт «**MS Visual Studio 2010**» (см. рис.1). После этого выйдет окно MS Visual Studio 2010, отображенное на рис.2.

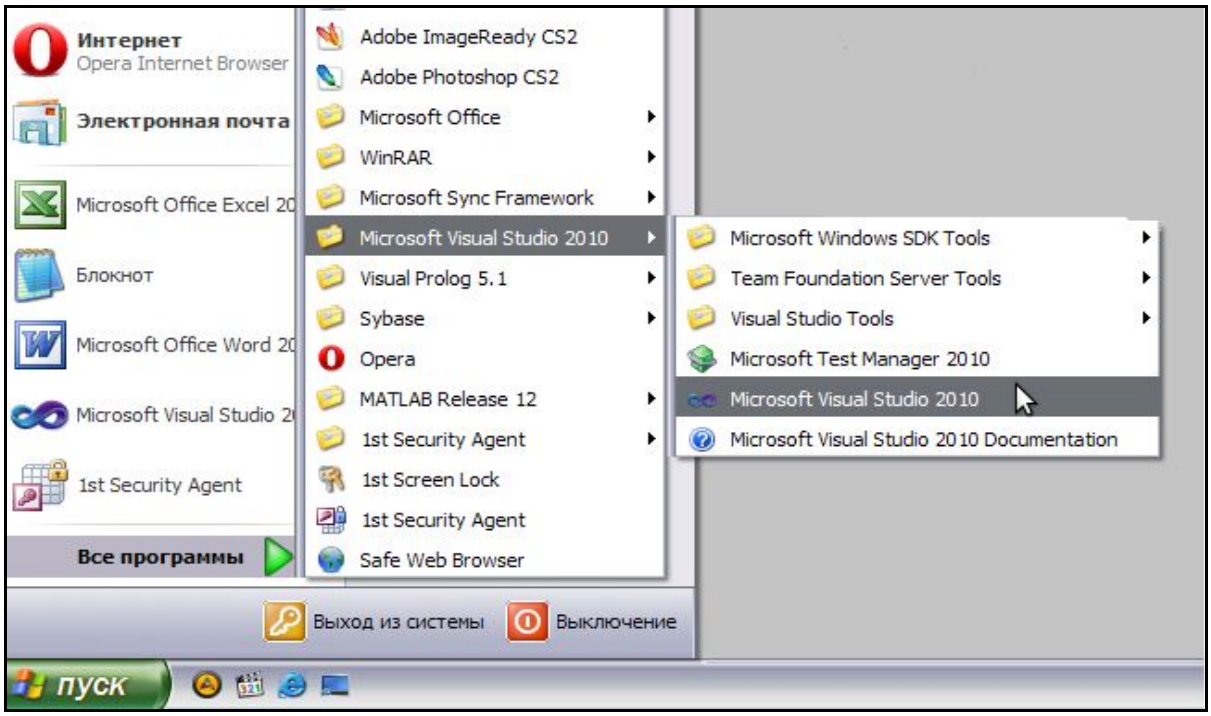


Рис. 1. Открытие программы Microsoft Visual Studio 2010

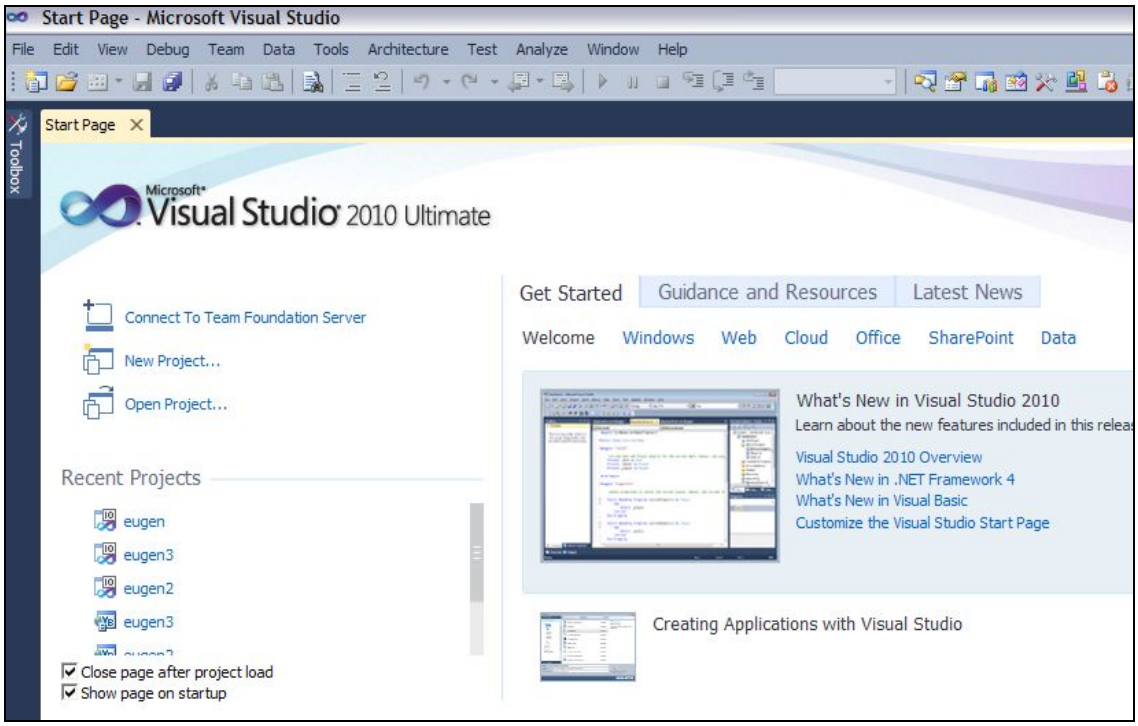


Рис. 2. Окно приложения Microsoft Visual Studio 2010

В открывшемся окне в меню «File» выберите команду «New Project…» (см. рис. 3).

Рис. 3. Создание нового проекта

В появившемся диалоговом окне на панели «Resent templates» выберите пункт «Visual C++», на среднем поле выбрать Win32 Console Application и в поле ввода Name напечатайте имя файла, например «Proba», нажмите клавишу “OK” (см. рис.4). Далее в следующей появившейся форме выбираем подпункт Application Settings и отмечаем радиокнопку Console application и check-box кнопку Empty project (см. рис.5)

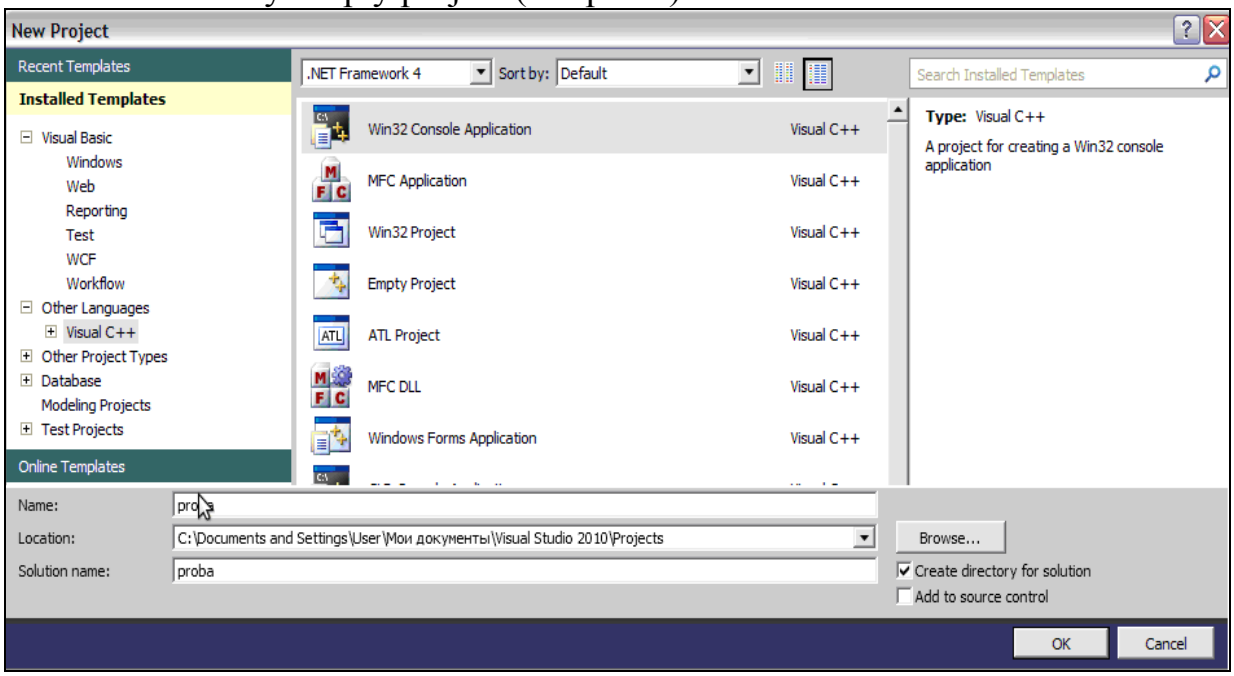


Рис. 4. Окно New Project

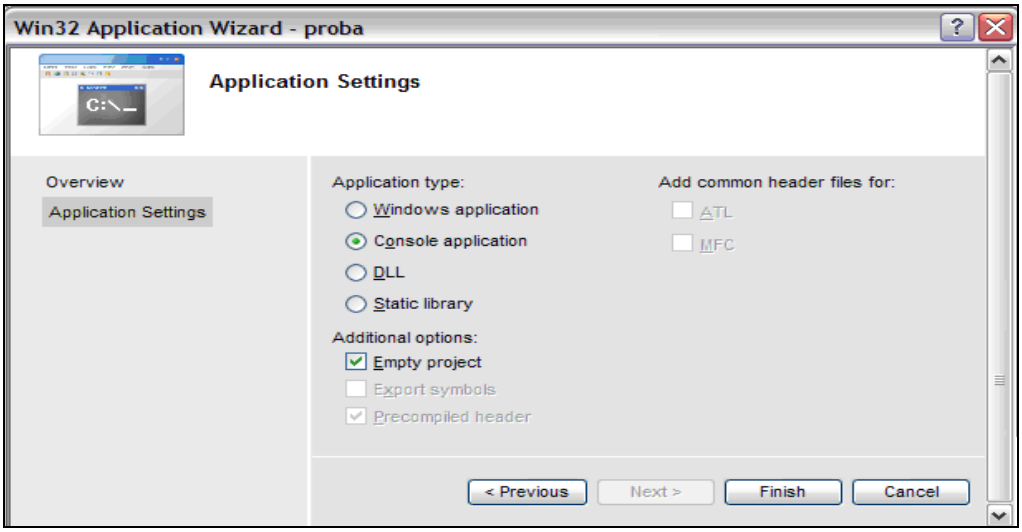


Рис. 5. Настройка приложения

На панели Solution Explorer выделяем левой кнопкой Sourse Files, затем щелкнув правой кнопкой в контекстном меню выбираем команду Add, затем New Item…(см. рис. 6.). В окне Add New Item отмечаем C++ File(.cpp) и заполняем поле Name (см. рис. 7.)

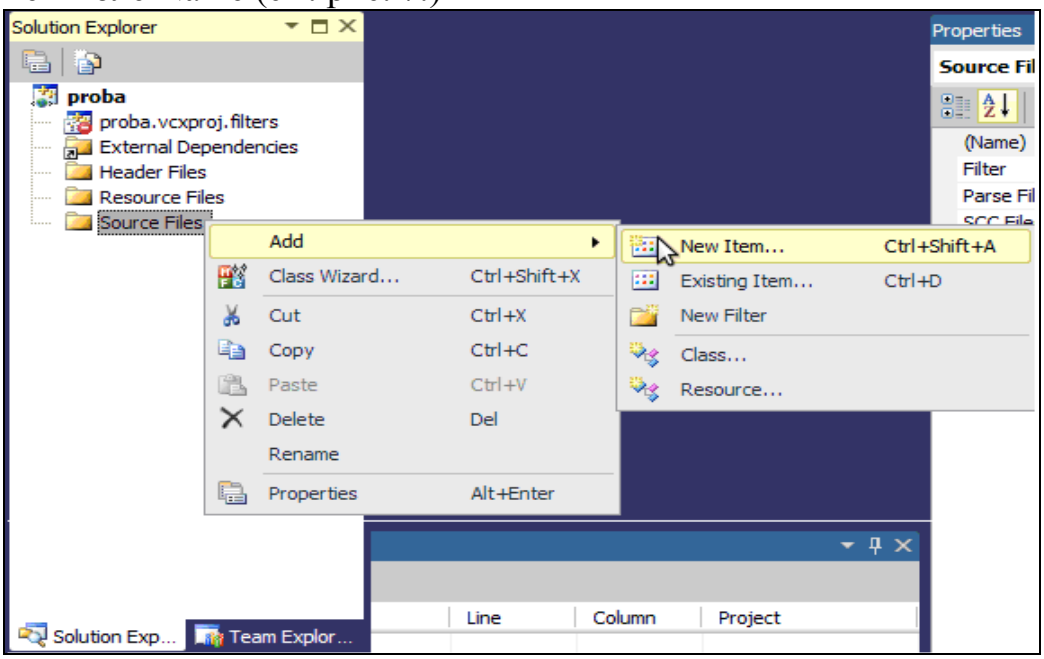


Рис. 6. Панель Solution Explorer.

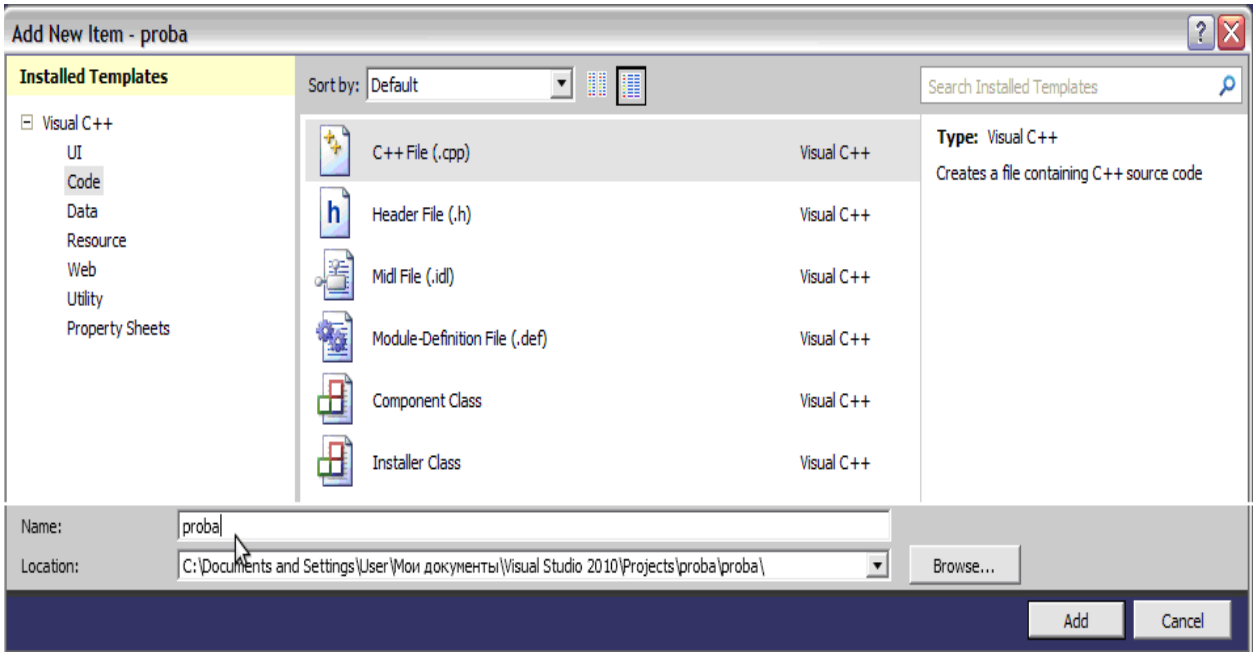


Рис. 7. Окно Add New Item

В появившемся окне редактора печатаем текст программы. Первая программа: сложение двух целых чисел (см. рис. 8).

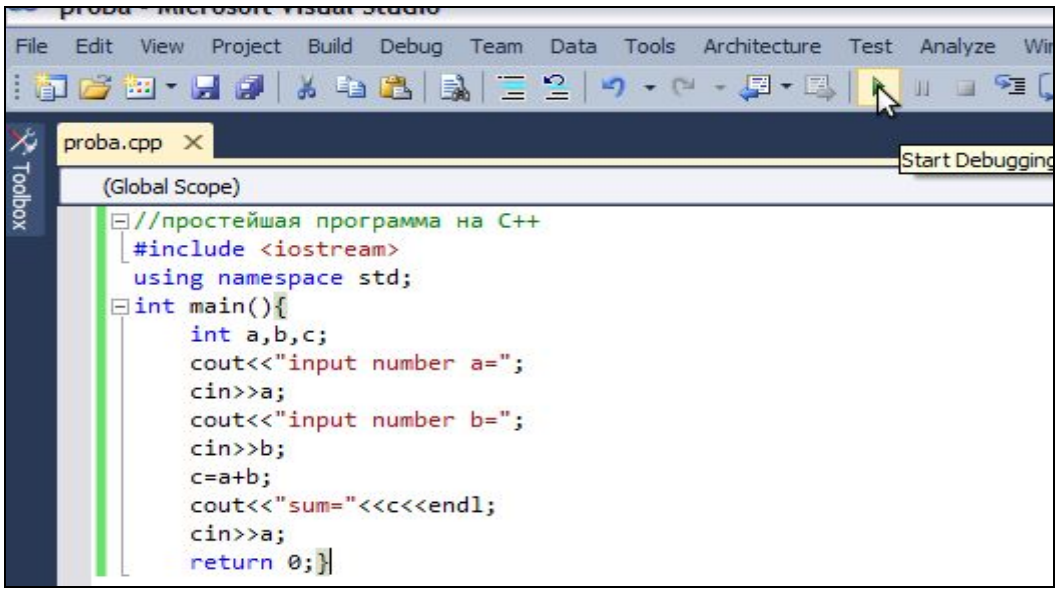


Рис 8. Программа вычисления суммы двух целых чисел

Во время набора внизу в окне «Errors list» появляется список сообщений об ошибках, рядом с ошибкой пояснение на английском языке, какого рода ошибка (см. рис. 9).

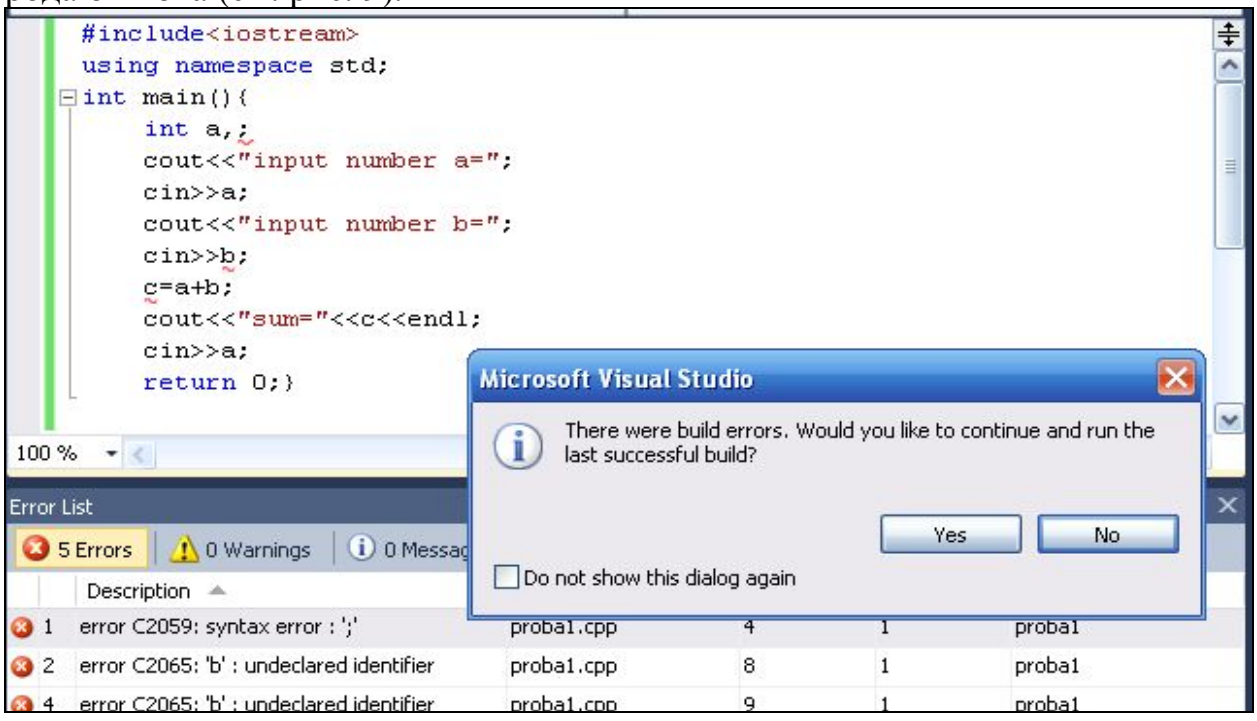


Рис. 9. Сообщения об ошибках компиляции

Существует 2 вида ошибок: ошибки «errors» и предупреждения «warnings» (см. рис. 9). Ошибки типа «errors» указывают на явные ошибки, без исправления которых не будет создан исполняемый файл. Ошибки типа «warnings» не препятствуют созданию исполняемого файла, но также должны быть внимательно изучены, так как могут указывать на логические ошибки программиста. Если в программе есть ошибки, подведите курсор мыши к строке с сообщением об ошибке и дважды щелкните левую кнопку мыши, курсор укажет на строку, содержащую эту ошибку.

После того как набрали текст программы в окне редактора отправляем программу на компиляцию. Войдя в меню «Debug», запустите пункт «Start debugging F5» или щелкните левой кнопкой мыши на кнопке  на панели инструментов. После устранения всех ошибок откроется консольное окно, которое предложит ввести исходные данные ***input number a=*** и ***input number b=*** после ввода которых, в окне появится результат (рис.10).

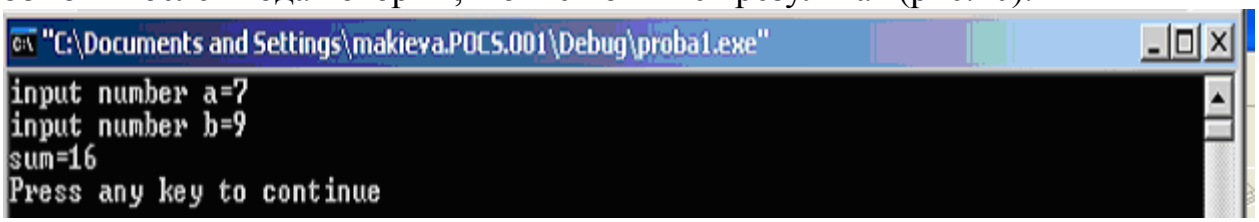


Рис. 10. Консольное окно вывода результатов вычислений

Рассмотрим набранную в окне программу. Символы «//» открывают комментарий на одну строку. Если комментарий состоит из нескольких строк, можно расположить его между символами «/*» и «*/». Комментарии не обрабатываются компилятором и служат для пояснений пользователю.

Каждая программа должна содержать функцию main. Левая фигурная скобка отмечает начало тела main. В строке int a, b, c; происходит ***объявление переменных***. Символы **a, b** и **c** являются ***именами переменных***.

Переменная – это область в памяти компьютера, где может храниться некоторое значение для использования его в программе. Данное объявление определяет, что переменные a, b, c имеют тип данных **int**; это значит, что они будут содержать ***целые значения***, т.е. целые числа, такие как, например, 7, -11, 0. Переменные должны объявляться с указанием имени и типа данных в любом месте программы до их использования. Для объявления переменных используются следующие типы данных:

Тип	Размер в байтах	Диапазон значений
char	**1**	**От –128 до 127 – символьный тип**
unsigned char	**1**	**От 0 до 256 – беззнаковый символьный тип**
short	**2**	**От –32768 до 32767 короткое целое число**
unsigned short	**2**	**От 0 до 65535 беззнаковое короткое целое число**
int	**4**	**От –2147483648 до 2147483647 – целое число**
unsigned int	**4**	**От 0 до 4294967295 - беззнаковое целое число**
long	**4**	**Совпадает с int – целое число**
unsigned long	**4**	**Совпадает с unsigned int – беззнаковое целое число**
float	**4**	**От 1,2E-38 до 3,4E+38 – вещественное число одинарной точности**
unsigned float	**4**	**От 0 до 7E+38 - беззнаковое вещ. число одинарной точности**
double	**8**	**От 2,2E-308 до 1,8E+308 –вещ. число двойной точности**
unsigned double	**8**	**От 0 до 4E+308 - беззнаковое вещ. число двойной точности**
bool	**1**	**true или false – логический тип**
void		**Пустой тип**

Строка #include <iostream> дает директиву препроцессору подключить библиотечный файл iostream.h, содержащий различные функции ввода и вывода. В нашей программе это объекты ***cin*** и ***cout.***

Объекты потоков **cout** и **cin** вызывают взаимодействие между пользователем и компьютером. Оператор ***cout<<"input number a=";*** печатает на экране сообщение ***input number a=***. Оператор ***cin>>a;*** получает от пользователя значение переменной ***a***. Оператор присваивания ***c=a+b;*** рассчитывает сумму переменных **a** и **b** и присваивает результат переменной **c,** используя операцию присваивания =. Оператор читается так: **«c *получает*** значение, равное **a + b».** Операция поместить в поток (<<) «знает», как выводить каждую единицу данных. Многократное использование операции поместить в поток в одном операторе называется ***сцепленной операцией поместить в поток***. В строке cout <<"sum=" << c << endl; текст, заключенный в кавычки (sum=), печатается без изменения, а вместо символа **c** печатается числовое значение переменной **c**, **endl** - это константа, которая содержит управляющий символ, переводящий курсор на новую строку. В операторах вывода можно также выполнять вычисления. Мы могли объединить два предыдущих оператора в один: cout << "sum=" << a + b << endl;

Функция main обязательно должна вернуть значение, что делает оператор return 0;. Значение 0 говорит об успешном завершении программы. Правая фигурная скобка (}) в данном случае информирует компьютер о том, что функция **main** окончена.

В программах могут выполняться арифметические вычисления. Язык C++ содержит арифметические операции сложение (+), вычитание (-), деление (/), умножение (*) и вычисления остатка от деления (%). Приоритет выполнения операций такой же, как и в обычных алгебраических выражениях, т.е. сначала выполняются умножение, деление и вычисление остатка от деления, а затем сложение и вычитание. Приоритет можно изменить, как и в обычной алгебре, с помощью круглых скобок. Арифметические операции используются как обычно, кроме двух особенностей:

1. целочисленное деление дает целый результат, т.е. если мы делим целое число на целое результат будет целым; например, выражение 7/4 равно 1, а выражение 17/5 равно 3. Заметим, что любая десятичная часть при целочисленном делении просто отбрасывается (т.е. усекается) — округление не производится. Если мы хотим получить в результате число с дробной частью, то после одного из чисел достаточно поставить знак точку; например, записать 7/4. и получить результат 1.75 или результат от 17./5 будет равен 3.4.

2. операция вычисления остатка % используется только для целых чисел, для вещественных чисел используется функция fmod, о которой вы узнаете ниже. Выражение x % y дает остаток от деления x на y. Таким образом, 7 % 4 равно 3; 17 % 5 равно 2. Эта операция используется для определения кратности, чётности, нечётности.

Для использования различных математических функций следует подключить библиотеку встроенных математических функций #include <cmath>

Функция	Действие
sqrt(x)	$\sqrt{x}$
log(x)	ln x
pow(x,y)	x^{y}
fmod(x,y)	остаток от деления “x” на “y” (для вещественных чисел)
fabs(x)	|x|
log10(x)	lgx
ceil(x)	округляет “x” до ближайшего целого, не меньшего“x”, ceil(9.2)=10.0
floor(x)	округляет “x” до ближайшего целого, не превышающего “x”, floor(9.2)=9.0
exp(x)	e^{x}
sin(x), cos(x), tan(x), atan(x)	тригонометрические функции sinx, cosx, tgx, arctgx

Необходимо выполнить нижеследующие задания для самостоятельной работы на ЭВМ. Результаты выполнения заданий всех лабораторных работ оформите в тетради в следующем порядке:
1) Лабораторная работа №, 2) условие задачи, 3) исходные данные, 4) блок-схемы (обозначения элементов блок-схем представлены в прил. 1.), 5) код программы, 6) результаты вычислений.

Задания для самостоятельной работы

1. Дан радиус окружности. Вычислить длину окружности и площадь круга.
2. Даны 2 катета прямоугольного треугольника. Найти гипотенузу и площадь треугольника.
3. Найти площадь кольца, внутренний радиус которого равен 20, а внешний – заданному числу r (r>20).
4. Даны x, y, z. Вычислить a, b, если

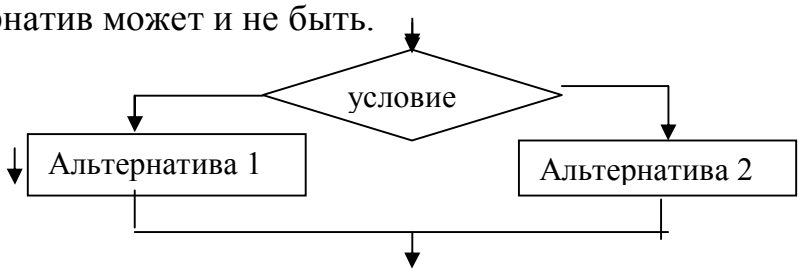
$$a = \frac{\sqrt{|x-1|} - \sqrt[3]{|y|}}{1 + \frac{x^2}{2} + \frac{y^2}{4}}, \qquad b = x\,(arctg\ \ z + e^{-(x+3)})$$

5. Найти площадь равнобочной трапеции с основаниями a и b и углом α при большем основании a. (Примечание: углы компилятор принимает в радианах).
6. Известны длины трех сторон треугольника. Вычислить его площадь.
7. Даны два целых числа. Найти среднее арифметическое этих чисел и среднее геометрическое их модулей.
8. Вычислить расстояние между двумя точками X1,Y1 и X2,Y2.
9. Дано четырехзначное целое число X.

Определить цифры числа.
Ответ выдать в виде, например:
7 – thousands; 3 – hundreds; 4 – tens; 6 - ones

Лабораторная работа № 2

В первой лабораторной работе мы рассматривали линейные (последовательные) алгоритмы, где действия идут последовательно один за другим. Если же перед нами стоит выбор выполнения различных действий, используются разветвляющиеся алгоритмы. Программа в зависимости от выполнения логического условия обязательно должна пойти по одной из ветвей. Одной из альтернатив может и не быть.



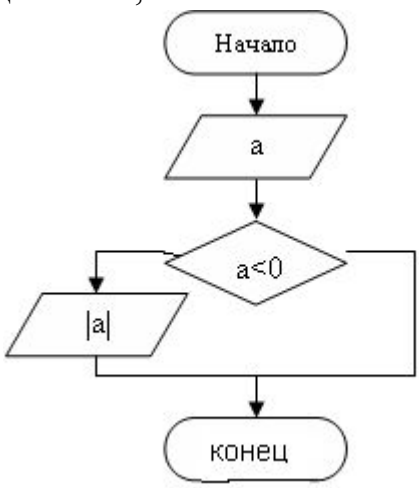
Для реализации разветвляющихся алгоритмов используется условный оператор “if”. Оператор “if” имеет две формы записи:

1) if (<условие>) <оператор>;
2) if (<условие>) <оператор1>; else <оператор2>;

Рассмотрим первый вид if (<условие>) <оператор>;

Проверяется условие, если оно истинно, то выполняется “оператор”, иначе программа идет далее вниз по тексту.

Пример: Дано вещественное число. Если оно отрицательно, то вывести его абсолютное значение.



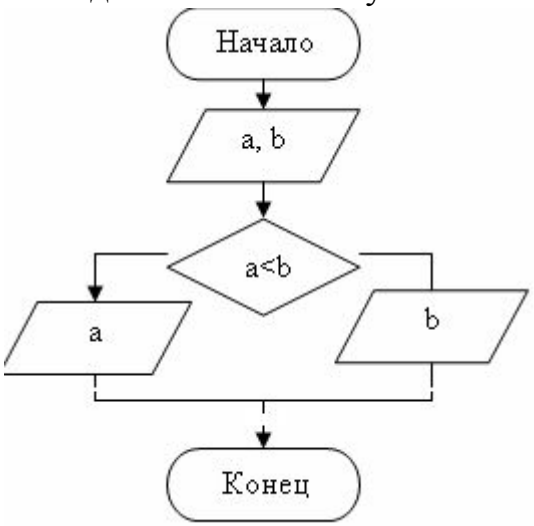

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double a;
    cin>>a;
    if (a<0)
      cout<<fabs(a)<<endl;
    return 0;}
```

Теперь рассмотрим второй вид

if (<условие>) <оператор1>; else <оператор2>;

Проверяется условие, если оно истинно, то выполняется “оператор1”, иначе выполняется «оператор2», далее программа идет вниз по тексту.

Пример: Даны два целых числа. Вывести наименьшее из них.




```
#include <iostream>
using namespace std;
int main() { int a, b;
cin>>a>>b;
if(a <b) cout<<”minimum=”<<a<<endl;
else cout<<”minimum=”<<b<<endl;
 return 0;}
```

Большинство операторов языка С++ в соответствии с синтаксисом могут выполнить только один оператор. В случае если по условию задачи требуется выполнить несколько операторов, они должны быть заключены в фигурные скобки {}. Операторы идущие после блока if, заключённые в фигурные скобки называются ”телом” оператора if.

Имеется более короткий способ записи условного оператора:

(<условие>)?<оператор1>:<оператор2>;

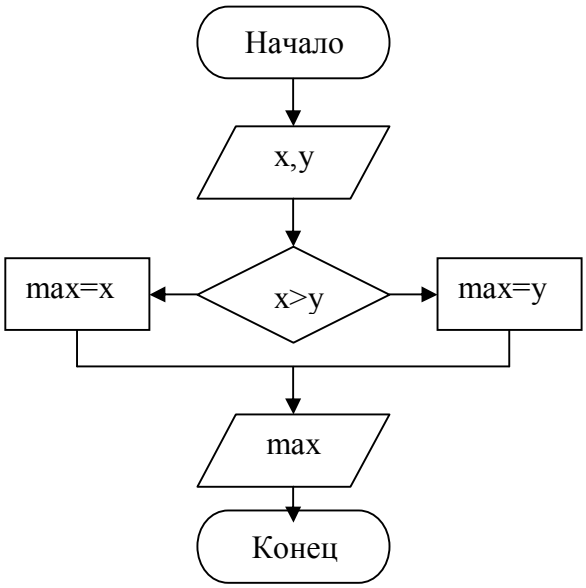
Проверяется условие, если оно истинно, то выполняется “оператор1”, иначе “оператор2”. Применение данного оператора приводит к получению более компактного машинного кода.

Пример:

Даны 2 действительных числа.

Найти наибольшее из этих 2-х чисел.

```
#include <iostream>
using namespace std;
int main() {
    double x, y, max;
    cin>>x>>y;
    max=(x>y)?x:y;
    cout<<”max=”<<max<<endl;
    return 0;
}
```



Для записи условия используют логические операции: >(больше), <(меньше), >=(больше или равно), <=(меньше или равно), = =(равно), !=(не равно) и логические операторы && (логическое и), || (логическое или), ! (отрицание). Условие может принимать 2 значения: true(истина) или false(ложь).

Для определения истинности сложного логического условия (выражения), записанного с помощью логических операций и операторов, применяется таблица истинности, где 1 – истина (true), а 0 – ложь (false).

Таблица истинности				
x	y	x&&y	x||y	!x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Логический оператор “&&” (и) даёт истинное значение логического выражения только в том случае, когда оба операнда этого оператора являются истинными. Например: if ((x==5)&&(y==5)) – это выражение буден истинным только в том случае, когда x=5 и y=5.

Логический оператор “||” (или) даёт истинное значение логического выражения в том случае, когда хотя бы один из операндов этого оператора

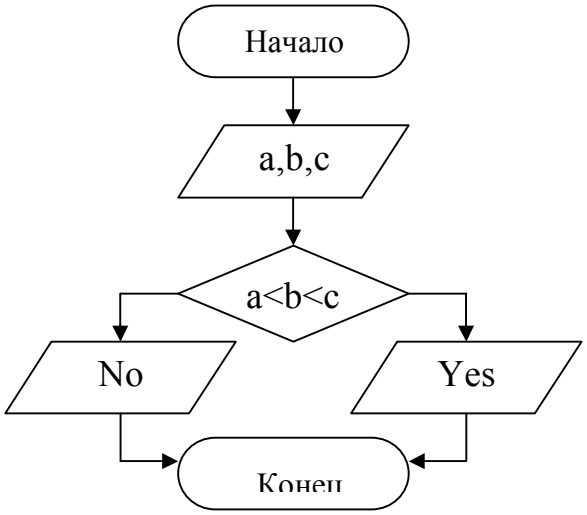
является истинным. Например: if ((x==5)||(y==5)) – это выражение буден истинным в том случае, когда или x=5 или y=5.

Логические операции так же как и арифметические операции имеют приоритет выполнения. Логические операции (>, <, <=, >=) имеют больший приоритет, чем логические операторы (! , || , &&).

Пример использования логических операторов:

Даны 3 действительных числа. Проверить правильно ли выполняется следующее соотношение: a<b<c;

```
#include <iostream>
using namespace std;
int main() {
   double a, b, c;
    cin>>a>>b>>c;
       if ((a<b)&&(b<c))
  cout<<"Yes";
       else
  cout<<"No";
    return 0;}
```



Оператор множественного выбора “switch”

В некоторых случаях использование оператора “if” может привести к возникновению конструкций с большим количеством вложений, усложняющих восприятие программы. Для решения этой проблемы предусмотрен оператор “switch”, который позволяет рассматривать сразу несколько условий.

```
switch (<выражение>) {
case <первое_значение>:       <оператор>; break;
case <второе_значение>:       <оператор>; break;
case <n-ое_значение>:  <оператор>; break;
default: <оператор>; break; }
```

Выражение, заданное в скобках оператора “switch” сравнивается со значениями, указанными за операторами “case”. В случае совпадения значений выражения, выполняется оператор в строке соответствующего оператора “case”. Будут выполняться все строки программы после выбранного оператора “case” до тех пор, пока не закончится тело блока оператора “switch” или не повстречается оператор “break”. Выполнение оператора “break” приводит к выходу из оператора “switch”. Если “break” отсутствует, то управление передаётся следующему оператору “case” или оператору “default”. Если ни одно из значений операторов “case” не совпадает с выражением, то выполняются строки программы, стоящие после оператора “default”. Наличие оператора “default” не обязательно. В случае его отсутствия и несовпадения выражения ни с одним из значений, будет выполнен оператор стоящий после блока оператора “switch”.

Примечание: выражение оператора switch может быть только целочисленного типа или типа char.

Пример: написать программу, которая по введённой оценке определяет статус учащегося. Решим задачу 2-мя способами для сравнения.

Программа с использованием вложенных конструкций if/else	Программа с использованием оператора switch

#include <iostream>
using namespace std;
int main() { int mark;
cout<<"Enter mark – ";
cin>>mark;
if (mark= =5)
cout<<"Excellent\n";
else
if (mark==4) cout<<"Good\n";
else
if (mark==3) cout<<"Satisfy\n";
else cout<<"Bad\n";
return 0; }
	#include <iostream>
	using namespace std;
	int main() {
	int mark;
	cout<<"Enter mark - ";
	cin>>mark;
	switch (mark) {
	case 5: cout<<"Excellent\n"; break;
	case 4: cout<<"Good\n"; break;
	case 3: cout<<"Satisfy\n"; break;
	default: cout<<"Bad\n";
	}
	return 0;}

Выполнить на ЭВМ и оформить в тетради задания для самостоятельной работы.

Задания для самостоятельной работы

1. Дано действительное число a. Вычислить f(a), если

$$f(x)=\begin{cases}0 & при \quad x \le 0,\\ x^2-x & при \quad 0<x\le 1,\\ x^2-\sin \pi x^2 & в \quad остальных \quad случаях\end{cases}$$

2. Даны действительные числа a, b, c. Найти корни квадратного уравнения $ax^2+bx+c=0$, в противном случае ответом должно служить сообщение, что корней нет.

3. Даны действительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x, y, z.

4. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны.

5. Даны действительные числа. Выбрать из них, те которые принадлежат интервалу [1,3].

6. Даны действительные числа x и y (x!=y). Меньшее из них заменить полусуммой чисел, а большее – удвоенным произведением.

7. Если сумма 3 действительных чисел x,y,z <1, то наименьшее из этих чисел заменить полусуммой двух других, в противном случае заменить меньшее из x и y полусуммой двух оставшихся значений.

8. Программа запрашивает у пользователя «Сколько Вам лет?» . Вы вводите положительное число до 100. Программа должна выдать полный ответ « Вам X (или лет или года или год).

9. Даны действительные числа a, b, c.Удвоить эти числа, если a≥b≥c, и заменить их абсолютными значениями, если это не так.

10. Дано натуральное число n (n≥9999). Является ли это число палиндромом (перевертышем) с учетом четырех цифр, как , например, числа 2222, 6116, 0440 и т.д.

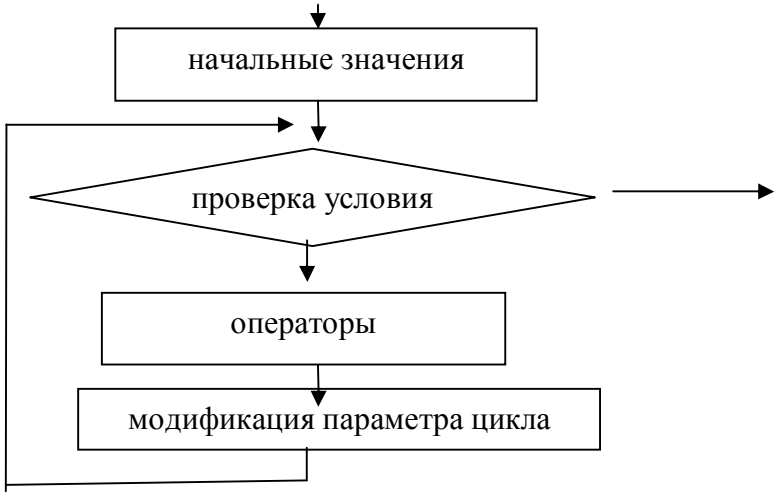
Лабораторная работа № 3
Операторы цикла

Операторы цикла используется для организации многократно повторяющихся действий. Любой цикл состоит из:

- тела цикла, т.е. операторов повторяющихся несколько раз;
- начальных значений;
- модификации (изменения) параметра цикла
- проверки условия продолжения цикла.

Один проход цикла называется итерацией. В С++ для удобства, а не по необходимости существует три типа операторов цикла. Это оператор цикла с предусловием while, оператор цикла с постусловием do/while и оператор цикла for. Поставленную задачу можно решить любым из них. Рассмотрим их подробнее.

Оператор цикла с предусловием "while" определяет действие, которое должно повторяться, пока условие в скобках остается истинным. Если условие при первом же шаге ложно, тело цикла не выполнится ни разу.



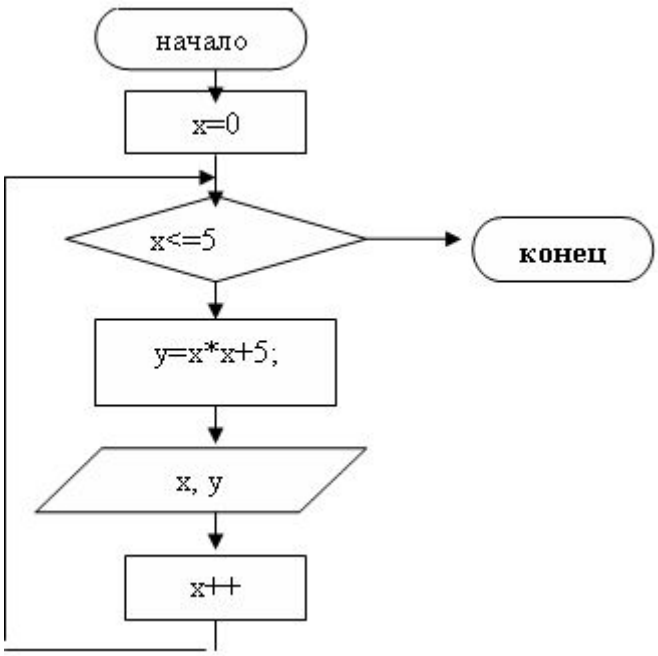
Форма записи оператора:

while (< условие>) { <операторы>};

Пример: Вычислить значения функции $y=x^2+5$, если $x \in [0,5]$ с шагом 1.

```
#include <iostream>
using namespace std;
int main() {
  int  x, y;
  x=0;
  while (x<=5)
{
y=x*x+5;
cout<<"x="<<x<<"y="<<y <<endl;
        x++;
}

 return 0; }
```



При использовании **оператора цикла с постусловием “do/while”** условие проверяется после выполнения тела цикла, это гарантирует выполнение цикла хотя бы один раз, даже если условие всегда ложно.

Форма записи оператора цикла с постусловием “do/while”:

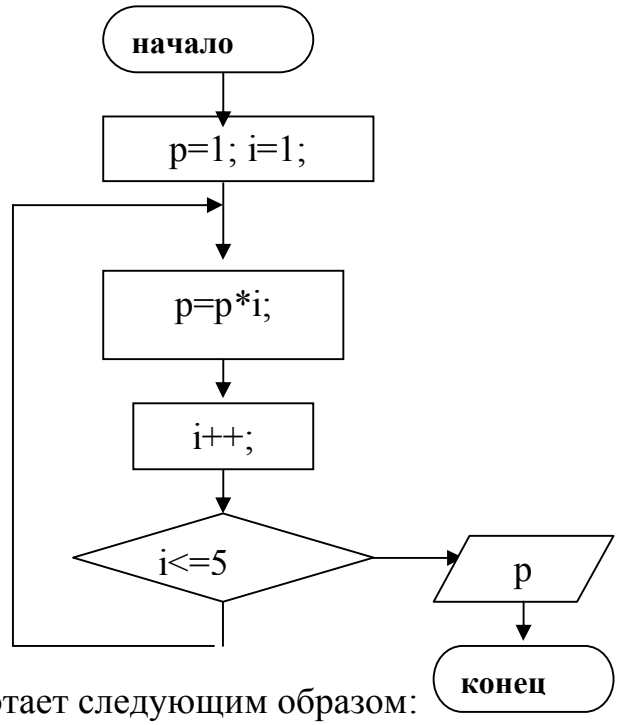
do

{<операторы>}

while(<условие>);

Пример: Вычислить произведение целых чисел от 1 до 5.

```
#include <iostream>
using namespace std;
int main() {
  int p, i;
  p=1; i=1;
{
   p*=i;
        i++;
  }   while (i<=5);
  cout<<p<<endl;
 return 0;
}
```



Оператор цикл for работает следующим образом:

1) Присваивается начальное значение счётчику цикла, т.е инициализируется счётчик цикла.

2) Выполняется проверка значения счётчика. Если результат проверки равен значению “true”, то сначала выполняется тело цикла, а за тем операция над счётчиком (приращение, уменьшение и т.д.).

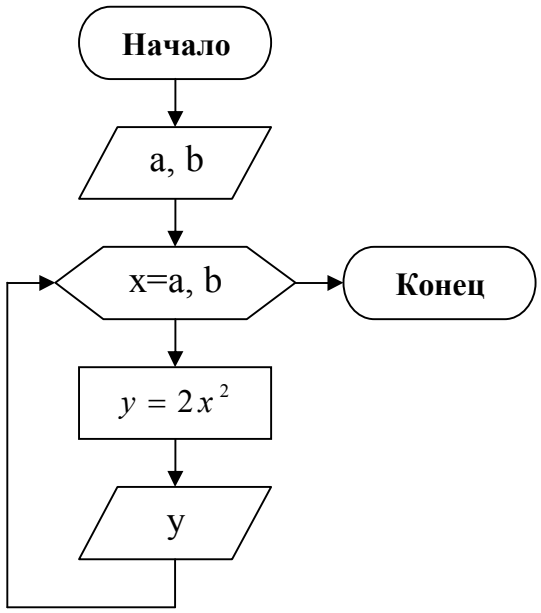
Форма записи оператора цикла

for (<инициализация_счётчика>;<проверка>;<операция_над_счётчиком>) операторы;

Пример:

Вычислить значение функции $y = 2x^2$ для всех значений "x" в диапазоне x∈ [a;b], где a- нижняя граница диапазона, b- верхняя граница диапазона.

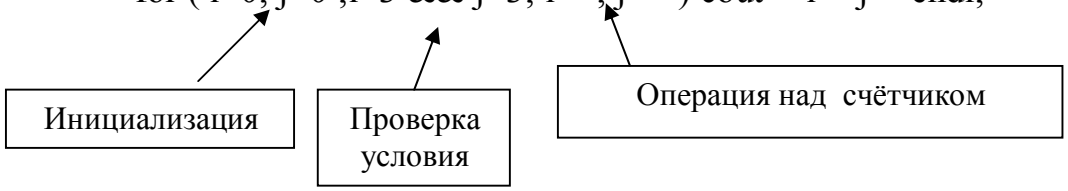
```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  int a,b,x;
  cin>>a>>b;
  double y;
  for (x=a; x<=b; x++)
{
    y=2 * pow ( x, 2);
    cout<<"y="<<y<<endl;
  }
  return 0;
}
```



Множественная инициализация счётчика цикла for

Синтаксис оператора цикла for позволяет инициализировать несколько переменных счётчиков, проверять сложное условие продолжения цикла и последовательно выполнять несколько операций над счётчиками цикла. Если присваиваются значения нескольким счётчикам или выполняются операции с несколькими счётчиками, то они записываются последовательно, и разделяются запятыми.

```
for ( i=0, j=0 ;i<3 && j<3; i++, j++ ) cout<<i<<j<<endl;
```



Результатом этого фрагмента программы будет вывод чисел в виде:

00
11
22

Вложенные циклы "for"

Цикл, организованный внутри тела другого цикла, называется вложенным циклом. В этом случае внутренний цикл полностью выполняется на каждой итерации (шаге) внешнего цикла.

```
for ( i=0; i < 2; i++)
  for ( j=0; j < 2; j++)
    cout << i << j << endl;
```

Результатом этого фрагмента программы будет вывод чисел в виде:

00
01
10
11
20
21

Счётчик цикла может быть как целого, так и вещественного типа. В качестве выражения изменения счётчика цикла может быть использовано сложное арифметическое выражение.

for (i=0.5; i<100; i/5+10)

Синтаксис цикла “for” позволяет определять счётчик цикла одновременно с его инициализацией.

for (int i=0; i<5; i++)

Выполнить на ЭВМ и оформить в тетради задания для самостоятельной работы.

Задания для самостоятельной работы

1.Вычислить значение функции $y = 2x^2 + 15$ при x изменяющемся от 1 до 20 с шагом 0.5.

2. Вычислить значение функции $y = 0.8x - \sin\sqrt{x} - 0.1$ при x изменяющемся от 0 до 16 с шагом 2.

3. Вычислить значение функции

$$t = \begin{cases} a\sin\left(\left(i^2+1\right)/n\right), & при \quad \sin\left(\left(i^2+1\right)/n\right) > 0 \\ \cos\left(i+1/n\right), & при \quad \sin\left(\left(i^2+1\right)/n\right) \le 0 \end{cases}$$

a=0.3, n=10, i изменяется от 1 до 10 с шагом 1.

4. Вычислить значение функции

$$S = \begin{cases} a + \dfrac{b}{e^x} + \cos x, & при \quad x < 2 \\ \dfrac{a+b}{x+1}, & при \quad x \ge 6 \\ e^x + \sin x, & при \quad 2 \le x < 6 \end{cases}$$

a=2.6, b=5, x изменяется от 0 до 10 с шагом 0.5

5. Найти сумму и произведение последовательности 10 целых чисел от 1 до 10. Использовать оператор **do…while**

6. Вычислить $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{100^2}$.

Использовать оператор **while**

7. Дано действительное число *a*. Найти среди чисел $1, 1+\frac{1}{2}, 1+\frac{1}{2}+\frac{1}{3}, \ldots$ первое большее *a*. Использовать оператор **do…while**

8. Даны натуральное n и действительное x. Вычислить $\sin x + \sin^2 x + \ldots + \sin^n x$. Использовать оператор **for.**

9. Дано натуральное n. Вычислить:

a. $(1+\frac{1}{1^2}) * (1+\frac{1}{2^2}) .* \ldots * (1+\frac{1}{n^2})$;

b. $\frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \ldots + \frac{1}{\sin 1 + \sin 2 + \ldots + \sin n}$;

c. $\sqrt{2+\sqrt{2+\ldots+\sqrt{2}}}$; n -корней

d. $\frac{\cos 1}{\sin 1} * \frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} * \ldots * \frac{\cos 1 + \cos 2 + \ldots + \cos n}{\sin 1 + \sin 2 + \ldots + \sin n}$

10. Дано натуральное *n*. Найти $\sum_{i=1}^{n} \frac{x_i}{1 + |y_i|}$, где $x_1=y_1=1$; $x_i=0.3x_{i-1}$; $y_i=x_{i-1}+y_{i-1}$, i=2,3,… Использовать оператор do…while

11. (Песня «Двенадцать дней рождества») Напишите программу, использующую повторение и структуры switch для печати текста песни «Двенадцать дней рождества». Одна структура switch должна использоваться для печати дня («Первый», «Второй» и т.д.). Другая структура switch должна использоваться для печати остальной части каждого куплета (возможно проще иметь дело с более знакомой песней того же типа «Двенадцать негритят»*).*

12. Рассчитайте значение π на основании бесконечного ряда
π =4-4/3+4/5-4/7+4/9-4/11+...
Сколько членов ряда потребовалось для получения значения 3.14? 3.141? 3.1415? 3.14159?

13. Напишите программу, которая читает пять чисел (каждое между 1 и 30). Для каждого просчитанного числа ваша программа должна напечатать строку, содержащую соответствующее число смежных звездочек. Например, если ваша программа прочла число 7, она должна напечатать *******.

14. Напишите программу, которая напечатает следующий ромб. Вы можете использовать операторы вывода, которые печатают или одну звездочку (*), или один пробел. Максимально используйте повторение (с вложенными структурами for) и минимизируйте число операторов вывода.

```
   *
  ***
 *****
*******
*********
*******
 *****
  ***
   *
```

Лабораторная работа №4
Одномерные массивы

Массив – набор однотипных объектов имеющих общее имя и отличающихся местоположением в этом наборе. Элементы массива занимают один непрерывный участок памяти компьютера и располагаются последовательно друг за другом. Нумерация элементов в массиве начинается с 0 и заканчивается n-1 элементом, где n-количество элементов массива.

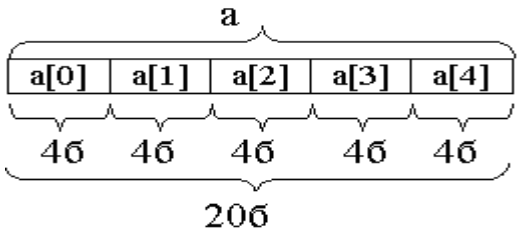
C++ в отличие от других языков не запрещает запись в любой участок памяти, поэтому программисту самому надо следить за тем, чтобы не произошел выход за границы массива, так как это может привести к прекращению работы операционной системы.

Синтаксис описания одномерного массива:

<Тип элементов массива> <имя_массива>[<количество элементов>];

Например: int a[5];

Массив из 5 целочисленных элементов, имя массива – a, занимает непрерывный участок памяти размером 20 байтов, нумерация начинается с 0 и заканчивается 4.



Одновременно с описанием массива можно инициализировать его элементы.

1) int a[5]={ 5, 8, 1, 3, 2 };
2) int a[]={ 5, 8, 1, 3, 2};

Во втором случае компилятор сам определит количество элементов массива по списку значений и выделит соответствующее количество памяти.

Также элементы массива можно инициализировать следующим образом: a[0]=5; a[1]=8; и т.д.

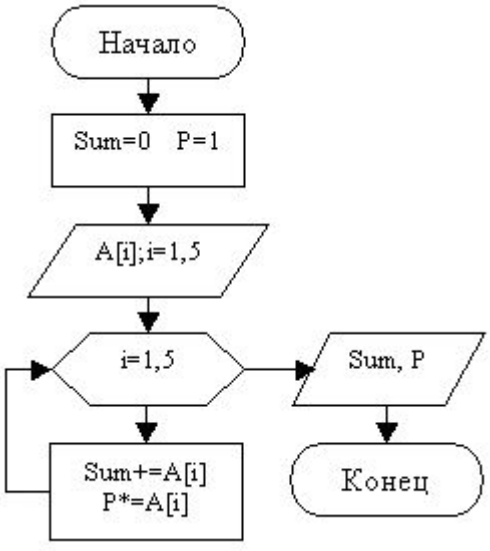
Для работы с массивами используются операторы цикла.

```
for ( i=0; i<5; i++)    cin>>a[i];        //Ввод массива
for ( i=0; i<5; i++)    cout<<a[i];       //Вывод массива
```

Стандартные алгоритмы обработки одномерных массивов

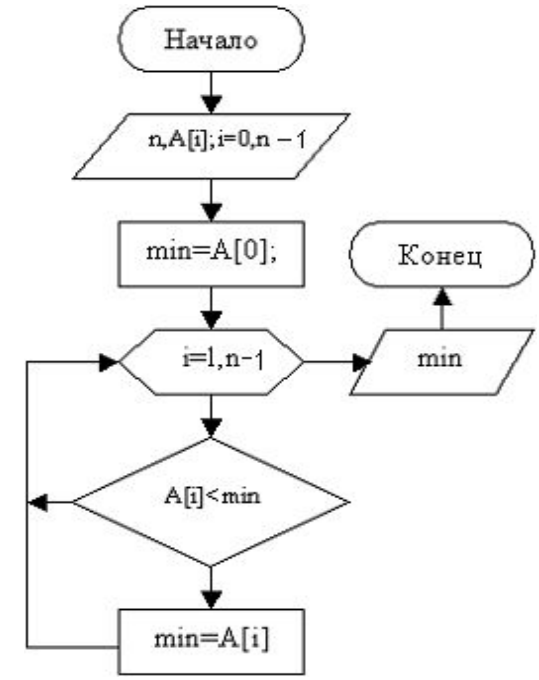
1. Поиск суммы и произведения массива:

```
#include <iostream>
using namespace std;
int main() {   int A[5], Sum=0, P=1,i;
  for (i=0;i<5;i++) cin>>A[i];
  for (i=0;i<5;i++) {
Sum+=A[i];  P*=A[i];   }
  cout<<"Sum="<<Sum<<"\nP="<<P<<endl;
  return 0; }
```



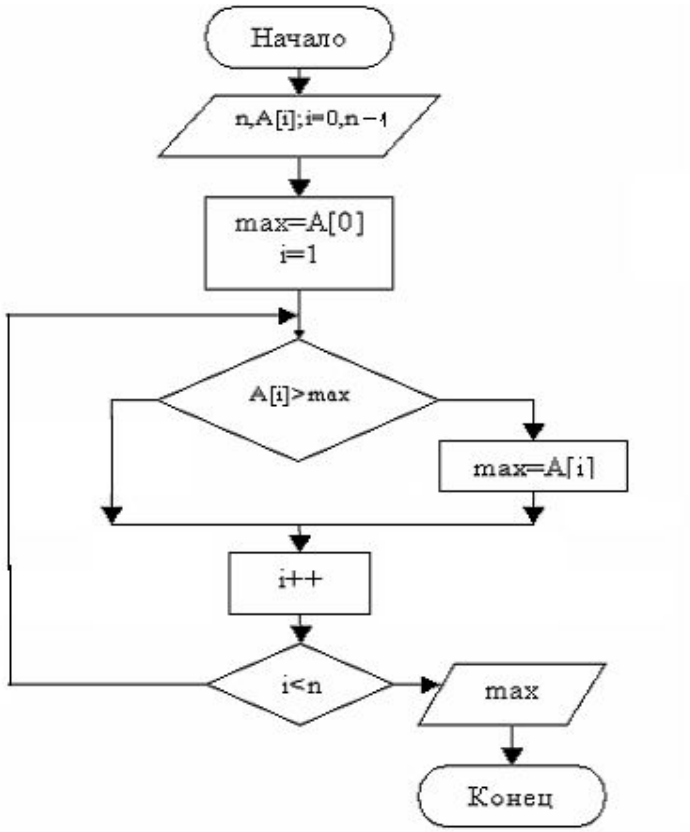
2. Поиск минимального элемента массива:

```
#include <iostream>
using namespace std;
int main() {
  int A[100], n, i, min;
  cin>>n;
  for ( i=0; i<n; i++)
    cin>>A[i];
  min=A[0];
  for (i=1; i<n; i++)
    if (A[i]<min)
      min=A[i];
  cout<<min<<endl;
  return 0;
}
```



3. Поиск максимального элемента массива:

```
#include <iostream>
using namespace std;
int main( ) {
  int A[100], n, i, max;
  cin>>n;
  for (i=0;i<n;i++)
    cin>>A[i];
  max=A[0];
  i=1;
  do {
    if (A[i]>max)
      max=A[i];
    i++;
  while (i<n);
  cout<<max<<endl;
  return 0; }
```



Двумерные массивы

Выше мы рассмотрели одномерные массивы, также существуют многомерные массивы. Например, двумерный массив int v[3][7] можно представить как три массива типа int по 7 элементов в каждом. Представим его в виде матрицы размером 3x7:

	0-ой столбец	1-ый столбец	2-ой столбец	3-ий столбец	4-ый столбец	5-ый столбец	6-ой столбец
0-ая строка	v[0][0]	v[0][1]	v[0][2]	v[0][3]	v[0][4]	v[0][5]	v[0][6]
1-ая строка	v[1][0]	v[1][1]	v[1][2]	v[1][3]	v[1][4]	v[1][5]	v[1][6]
2-ая строка	v[2][0]	v[2][1]	v[2][2]	v[2][3]	v[2][4]	v[2][5]	v[2][6]

Нумерация строк и столбцов начинается с 0 и заканчивается n-1. Первый индекс – номер строки, второй индекс - номер столбца.

Объявление двумерного массива:

<тип элементов массива> <имя > [количество строк][количество столбцов];

Например: int A[5][6];

Вместе с объявлением можно проинициализировать:

int A[2][3]={{1,2,3},{4,5,6}}; или int A[2][2]={1, 2, 3, 4};

Ввод двумерного массива с клавиатуры:

```
for ( i=0; i<5; i++)
  for ( j=0; j<5; j++)
    cin>>A[i][j];
```

Вывод двумерного массива в виде таблицы:

```
for (i=0;i<3;i++)
{   for (j=0;j<3;j++)
    cout<<A[i][j]<<" ";
      cout<<endl;
}
```

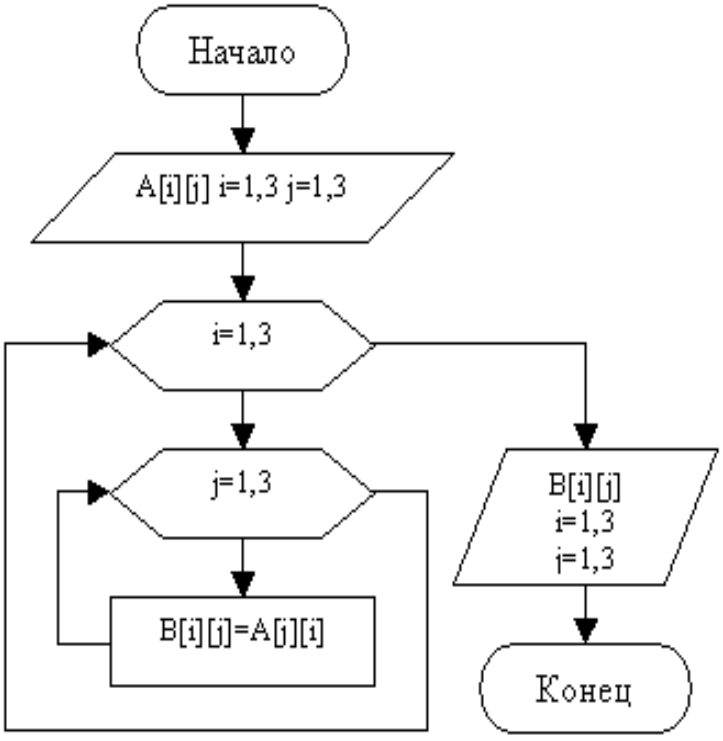
Алгоритм транспонирования двумерного массива

Дана матрица

$$A=\begin{pmatrix}10 & 11 & 12\\ 25 & 23 & 26\\ 31 & 38 & 40\end{pmatrix}$$

Получить транспонированную матрицу

$$A^T=\begin{pmatrix}10 & 25 & 31\\ 11 & 23 & 38\\ 12 & 26 & 40\end{pmatrix}$$




```
#include <iostream>
using namespace std;
int main() {
  int A[3][3],B[3][3],i,j;
  for (i=0;i<3;i++)
    for (j=0;j<3;j++)
      cin>>A[i][j];
  for (i=0;i<3;i++)
    for (j=0;j<3;j++)
      B[i][j]=A[j][i];
  for (i=0;i<3;i++) {
    for (j=0;j<3;j++)
      cout<<A[i][j]<<" ";
    cout<<endl;   }
  return 0;}
```

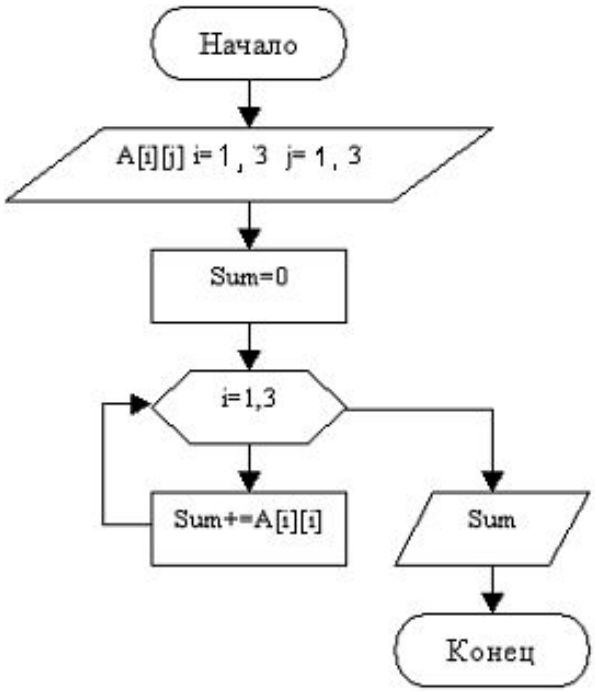
Сумма элементов главной диагонали матрицы

```
#include <iostream>
using namespace std;
int main() {
  int A[3][3],Sum=0,i,j;

 for (i=0;i<3;i++)
     for (j=0;j<3;j++)
       cin>>A[i][j];

 for (i=0;i<3;i++)
           Sum+=A[i][i];

cout<<"Sum="<<Sum<<endl;
  return 0;}
```



Генерация случайных чисел

Для генерации случайных чисел используется функция "rand()", которая находится в библиотеке <stdlib.h>. Эта функция генерирует случайные числа от 0 до RAND_MAX. Константа RAND_MAX находится в библиотеке <stdlib.h> и равна 32767. В реальных задачах, такой диапазон значений практически не используется и для того, чтобы получить другой диапазон значений используется операция масштабирования.

Формула масштабирования имеет следующий вид: n=a+rand()%b;

a – величина сдвига, которая равна первому числу в требуемом диапазоне целых чисел.

b – масштабируемый коэффициент, который равен ширине требуемого диапазона целых чисел.

Например, если требуется сгенерировать целые числа в диапазоне от 0 до 5, то форма записи будет выглядеть следующим образом: n=rand()%6;
n=1+rand()%6 - будет генерировать числа в диапазоне от 1 до 6.

Пример:

```
#include <iostream>
#include <stdlib>
using namespace std;
int main() {
  int Array[10];
  for (int i=0;i<10;i++) {
    Array[i]=rand()%6;        // Заполняет массив числами от 0 до 5
    cout<<Array[i]<<endl;}
  return 0; }
```

Если запустить эту программу несколько раз, то при каждом запуске, мы будем получать одни и те же значения элементов массива.

Функция “rand()” генерирует псевдослучайные числа. Эта характеристика функции является очень важной при отладке программы, т.е. позволяет доказать, что программа работает должным образом. (Так как программа должна возвращать одни и те же значения при одинаковых входных данных).

После того, как программа тщательно отлажена, она может быть использована для получения разных последовательностей случайных чисел при каждом её выполнении.

Генерация случайных чисел называется ***рандомизацией.***

Рандомизация осуществляется при помощи функции “srand()”, которая так же находится в библиотеке <stdlib.h>.

Функция “srand()” получает аргумент типа “unsigned int”, и при каждом запуске “srand()” задаёт начальное число, которое используется функцией “rand()” для генерации случайных чисел.

Пример:

```
#include <iostream>
#include <stdlib>
using namespace std;
int main() {
  unsigned int k;
  cin>>k;
  srand(k);
  for (int i=0;i<=5;i++)
  cout<<rand()%6<<endl;
  return 0; }
```

Для того, чтобы генерировать случайные числа, не вводя каждый раз новое число, используемое как начальное число, можно в качестве аргумента функции “srand()” использовать системное время компьютера srand(time(NULL));. Функция “time()” из библиотеки <time>, возвращает текущее время в секундах. Это время преобразуется в беззнаковое целое число, которое используется как начальное значение в генерации случайных чисел.

```
#include <iostream>
#include <time>
#include <stdlib>
using namespace std;
{
  srand(time(NULL));
  for (int i=0;i<=5;i++)
  cout<<rand()%6;
  return 0;
}
```

Задания для самостоятельной работы

(Одномерные и двумерные массивы)

1. Даны температуры воздуха за неделю. Определить среднюю температуру воздуха за неделю и сколько раз температура опускалась ниже 0º.

2. Даны натуральные числа N, a0, a1,…., a(N-1). Определить количество членов последовательности имеющих четные порядковые номера и являющихся нечетными числами.

3. Определить является ли данная последовательность убывающей (во избежание лишних проверок использовать оператор break) .

4. Дан массив чисел. Определить, сколько в нем пар одинаковых соседних элементов.

5. Дан массив чисел. Найти наибольший элемент, поставить его первым.

8. Задан двумерный массив, содержащий 3 строки и 4 столбца. Найти наибольший элемент, номер строки и столбца, в которых он расположен.

9. Определить количество положительных элементов каждого столбца двумерного массива, содержащего 5 строк и 5 столбцов.

10. Составить программу для вычисления средних арифметических значений положительных элементов каждого столбца двумерного массива, содержащего 6 столбцов и три строки. При условии, что в каждом столбце есть хотя бы один положительный элемент.

11. Дана действительная квадратная матрица. Заменить нулями все элементы, расположенные на главной диагонали и выше нее.

12. Даны 8 действительных чисел x1, x2, …, x8. Получить квадратную матрицу 8x8

$$x1 \quad x2 \quad \dots \; x8$$
$$x1^2 \quad x2^2 \; \dots \; x8^2$$
$$\dots\dots$$
$$x1^8 \quad x2^8 \; \dots \; x8^8$$

13. Таблица футбольного чемпионата задана квадратной матрицей порядка n, в которой все элементы, принадлежащие главной диагонали, равны нулю, а каждый элемент, не принадлежащий главной диагонали, равен 2, 1 или 0 (результат игры: 2 – выигрыш, 1- ничья, 0 – проигрыш). При заполнении таблицы желательно использовать генерацию случайных чисел

а) Найти число команд, имеющих больше побед, чем поражений.

б) Определить номера команд, прошедших чемпионат без поражений.

в) Выяснить, имеется ли хотя бы одна команда, выигравшая больше половины игр.

14. Дан двумерный массив, содержащий 3 строки и 4 столбца. Упорядочить массив по убыванию элементов 3-ей строки.

15. Дан двумерный массив, содержащий 5 строк и 2 столбца. Упорядочить массив по возрастанию элементов 2-го столбца.

16. Даны целые числа a_0, a_1, a_2. Получить целочисленную матрицу b каждый элемент, которой определяется по следующей формуле

$b_{ij} = a_i - 3a_j$, i, j=0,1,2.

17. Дана квадратная матрица A, содержащая 5 строк и 5 столбцов. Получить две квадратные матрицы B и C, элементы которых определяются по следующим формулам

$$B_{ij} = \begin{cases} A_{ij} & при \quad j \geq 0 \\ A_{ji} & при \quad j < i' \end{cases} \qquad C_{ij} = \begin{cases} A_{ij} & при \quad j < i \\ -A_{ij} & при \quad j \geq i \end{cases}$$

18. Найти наибольший элемент главной диагонали матрицы C размером 4x4 и вывести на печать всю строку, в которой он находится.

Лабораторная работа №5
Массивы символов

Форма записи:

char Array[10];

Статический массив – массив, в котором конкретное число элементов.

Для каждого массива символов существует символ конца строки – ‘\0’, поэтому память под массив символов нужно выделять на один элемент больше.

Инициализация массива символов:

char Array[]={‘f’,‘i’,‘r’,‘s’ ,‘t’,‘ ’,‘c’,‘o’,‘u’,‘r’,‘s’,‘e’,’\0’};

Обязательный элемент для символьного массива

char Array[]=”first course”; // Это упрощённая запись доступная в C++

В этом случае компьютер автоматически вычислит размер конец строки. Один символ занимает один бит памяти. Если такую строку вводить при помощи оператора “cin”, то будет введено только первое слово “first”, так как оператор “cin”, воспринимает пробел как окончание строки, после чего прекращается ввод данных.

Функция get()

Для ввода строки, обычно используется функция “get()”, в которой необходимо указать 2 аргумента.

cin.get(<аргумент1>,< аргумент2>);

аргумент1 – имя строковой переменной для записи входных данных,

аргумент2 – целое число, указывающее размер строковой переменной.

Пример блока программы с использованием этой функции:

```
char Array[80];
cout<<”Enter the string:\n”;
cin.get(Array,80);
cout<<Array<<endl;
```

Array[0] – 1-ый символ. Array[78] - 79-ый символ. Array[79] – ноль-символ (0)

Функция getline()

cin.getline(<аргумент1>,< аргумент2>);

аргумент1 – имя строковой переменной для записи входных данных.

аргумент2 – целое число указывающее размер строковой переменной, т.е. количество элементов строки, один из которых отводится под символ конца строки.

Считывание строки завершается, как только количество считанных символов строки станет равным аргумент2, даже если считывание не достигло конца строки.

В отличие от функции “get()”, функция “getline()” считывает и удаляет из строки символ разрыва строки “\n”, функция “get()” воспринимает этот символ как разделитель, и оставляет его в строке.

Массивы строк (в виде 2-мерного массива)

```
char name[5][20];    // 5 строк по 20 символов
cout<<”Введите 5 имён, по одному в строке:\n”;
for (int i=0;i<5;i++)
  cin.getline(name[i],20);
for (i=0;i<5;i++)
  cout<<name[i]<<endl;
```

Функции обработки символов из библиотеки <ctype.h>

Функция	Назначение
isdigit(символ)	Возвращает значение “истина”, если символ является цифрой и значение “ложь” в противном случае. isdigit(<имя символьной переменной>); isdigit(‘<символ>’);
isspace(символ)	Возвращает значение “истина”, если символ - пробел или символ новой строки и “ложь” в обратном случае. isspace(<имя символьной переменной>); isspace (‘символ >’);
isupper(символ)	Возвращает значение “истина”, если символ записан в верхнем регистре и значение “ложь” в противном случае. isupper(<имя символьной переменной>); isupper(‘символ >’);

islower(символ)	Возвращает значение “истина”, если символ записан в нижнем регистре и значение “ложь” в противном случае. islower(<имя символьной переменной>); islower (‘символ >’);
isalpha(символ)	Возвращает значение “истина”, если символ является буквой и значение “ложь” в противном случае. isalpha(<имя символьной переменной>); isalpha(‘символ >’);
toupper(символ)	Возвращает символ в верхнем регистре. char symbol; symbol=toupper(‘<символ в нижнем регистре>’); symbol=toupper(<имя символьной переменной в нижнем регистре>);
tolower(символ)	Возвращает символ в нижнем регистре. char symbol; symbol=tolower(‘<символ в верхнем регистре>’); symbol=tolower(<имя символьной переменной в верхнем регистре>);

Пример: Прочитать предложение до точки и вывести его на экран. Заменить все пробелы символом *.

```
#include <iostream>
#include <ctype>
using namespace std;
int main() {
  char symbol;
  do {
   cin.get(symbol); // Функция “get()” Работает как с 1 так и с 2-я параметрами
    if (isspace(symbol))   cout<<”*”;
     else cout<<symbol; }
  while (symbol!=’.’);
  return 0; }
```

Примечание: при считывании одного символа, функция “get()” работает с одним параметром.

Функции преобразования строки в число из библиотеки <stdlib.h>

Функция	Описание
atoi(<строка>)	Функция преобразования строки в число типа “int”.
atol(<строка>)	Функция преобразования строки в число типа “long”.
atof(<строка>)	Функция преобразования строки в число типа “double”.

Функции преобразования строк из библиотеки <string.h>

Функция	Описание
strlen(<строка>)	Возвращает целое число равное длине строки (символ конца строки не учитывается).
strcat(<строка1>,<строка2>)	Добавляет значение “строка2” в конец “строка1”.
strcpy(<строка1>,<строка2>)	Копирует символы из “строка2” в конец “строка1” и при этом не проверяет, достаточно ли место в “строка1” для копирования символов.
strncpy(<строка1>,<строка2>, <кол-во символов>)	Копирует не более “n” символов из <строка2> в <строка1>.
strcmp(<строка1>,<строка2>)	Сравнивает <строка1> с <строка2>, если они равны, возвращает значение “ложь”.

Функция “getch()”из библиотеки <conio.h>ожидает нажатия клавиши “Enter”.

```
#include <iostream>
#include <conio>
using namespace std;
int main() {
  getch();
 return 0;}
```

Задания для самостоятельной работы

1. Определить, содержит ли текст символы, отличные от букв и пробелов.
2. Изменить текст так, чтобы вместо строчных букв стояли соответствующие прописные буквы, а вместо прописных – строчные. Определить общее число произведенных замен.
3. Каждую группу символов one заменить на three.
4. Первое вхождение группы символов one заменить на three.
5. После каждой точки вставить пробел, если он отсутствует.
6. Удалить из текста все цифры.
7. Определить, содержит ли букву А первое слово заданного текста.
8. Определить, сколько слов оканчивается на заданный символ.
9. В тех словах, которые оканчиваются сочетанием букв ed, заменить это окончание на ing.
10. Определить число символов первого слова.
11. Вывести на экран первое слово текста, если оно начинается с заданного символа.
12. Во всех словах, начинающихся с f, заменить f на сочетание ph.
13. Изменить текст, вставив после каждого слова еще один пробел.
14. Вычислить, сколько слов встречается в тексте до первой точки.
15. Удалить из текста все символы, отличные от цифр и пробелов.

16. Определить число символов, отличных от букв и пробелов, встретившихся в тексте до первой точки.

17. Проверить имеет ли место соответствие открывающихся и закрывающихся круглых скобок.

18. Определить, сколько раз в последовательности A встречается буква N и сколько раз в последовательности B встречается цифра 9.

19. Заменить в последовательности A все символы * на пробелы, а в последовательности B все точки на запятые.

20. Определить, содержит ли последовательность A строчные латинские буквы, а последовательность B – прописные.

21. Определить, сколько раз в последовательности A символ ; встречается до первой точки, и сколько раз в последовательности B символ ; встречается до первой запятой.

22. Для заданной последовательности A проверить, встречается ли среде ее первых 10 символов буква s, а для последовательности B проверить, встречается ли среди ее первых 20 символов буква q.

23. Определить номер первого вхождения заданного символа в каждую из последовательностей A, B, C.

24. Определить, сколько слов в последовательности A оканчивается на букву N и сколько слов в последовательности B оканчивается на букву Y.

25. Из последовательностей A, B, C удалить все пробелы.

Лабораторная работа № 6
Функции

Функция - это блок программного кода, не входящего в основную программу и выполняющего определенную задачу.

Функции являются основой модульного программирования (составления программ из модулей). Существует несколько причин для построения программ с использованием функций.

Во-первых, подход «разделяй и властвуй» делает разработку программ более управляемой. Вторая причина — повторное использование программных кодов, т.е. использование существующих функций как стандартных блоков для создания новых программ. Повторное использование — основной фактор развития объектно-ориентированного программирования. При продуманном присвоении имен функций и корректном их описании программа может быть создана быстрее из стандартизированных функций, соответствующих определенным задачам. Третья причина — желание избежать в программе повторения каких-то фрагментов. Код, оформленный в виде функции, может быть выполнен в разных местах программы простым вызовом этой функции.

Все программы, которые мы ранее рассматривали, содержали главную функцию, называемую main(), а также некоторые стандартные библиотечные функции, такие как математические функции из библиотеки cmath, функции ввода-вывода и др. Теперь мы научимся писать свои собственные функции.

Для использования функции ее надо **объявить**, **определить** и конечно **вызвать**.

Объявление функции называется прототипом. Прототип содержит имя функции, тип функции, параметры, принимаемые функцией и их тип. Имя функции — это любой правильно написанный идентификатор.

Синтаксис прототипа:

Тип_функции имя_функции ([тип параметр1, тип параметр2,…]);

где в квадратных скобках указаны необязательные параметры. Прототип функции обязательно заканчивается точкой запятой. В прототипе имена параметров можно не указывать.

Существуют 3 способа объявления функции:

- прототип функции записывается в другой файл и затем подключается в текущий файл с помощью директивы #include;
- прототип записывается в этом же файле до имени главной программы main( );
- прототип функции совпадает с ее определением. (Этот способ использовать не рекомендуется, так как если прототип функции записан отдельно от определения функции, у компилятора имеется дополнительная возможность на этапе компиляции проверить правильность обращения к функции, т.е. проверить тип передаваемых ей параметров).

Мы будем использовать второй способ объявления функции.

Определение функции состоит из заголовка функции и ее тела. Заголовок функции подобен прототипу и состоит из типа возвращаемого значения функции, имени функции и параметров функции:

Тип_функции имя_функции ([тип параметр1, ,…]) *//заголовок функции*
{ тело функции
return возвращаемое значение; }

Тело функции состоит из операторов языка С++. В качестве возвращаемого значения после оператора return могут быть использованы имена переменных (например, return x;), константные значения (например, return 1;), арифметические выражения (например, return 5*x*x+7;), логические выражения, функция будет возвращать либо значение «true», либо «false». Тип этого значения должен совпадать с типом функции, указанным в прототипе. Также функция может содержать несколько операторов return либо может не иметь ни одного оператора return, в этом случае функция имеет тип возвращаемого значения void. Функции могут принимать или не принимать параметры, могут возвращать значение или не возвращать значение.

При **вызове** функции в программе указывается только ее имя и в скобках через запятую параметры, передаваемые в эту функцию (без указания типов).

Параметры, используемые в прототипе или заголовке определения функции, называются формальными. Параметры, передаваемые функции при ее вызове, называются фактическими.

Функции, имеющие тип void, называются не возвращающими значения и их вызов отличается от вызова обычных функций:
Имя (параметр1, параметр2,….); //синтаксис функции, не возвращающей значение.

После вызова функции ее выполнение начинается с первого оператора стоящего после открывающейся фигурной скобки в определении функции. После того, как в функции встретится оператор ***return***, будет выполнено выражение, стоящее за этим оператором, и его результат будет возвращен в точку вызова функции.

Рассмотрим несколько примеров создания функций.

Пример 1: Написать функцию square для вычисления квадратов целых чисел от 1 до 10.

```
#include<iostream>
using namespace std;
int square ( int x );   // прототип функции
int main( )
{ for ( int x = 1; x <= 10;x ++)
cout << square ( x ) <<  “  ” ;   // вызов функции
cout << endl ;
return 0; }
int square ( int y) {   return y * y;  }// определение функции
```

Функция **square** активизируется или вызывается в **main** вызовом square(x). Функция создает копию значения x в параметре y. Затем square вычисляет y * y. Результат передается в ту точку main, из которой была вызвана square, и затем этот результат выводится на экран. Благодаря структуре повторения for этот процесс повторяется десять раз. Описание square показывает, что эта функция ожидает передачи в нее целого параметра y. Ключевое слово int, предшествующее имени функции, указывает, что square возвращает целый результат. Оператор **return** в **square** передает результат вычислений обратно в вызывающую функцию.

Пример 2. Написать функцию возведения целого числа в целую положительную степень.

```
#include<iostream>
using namespace std;
int pow_1( int a, unsigned int b);
int main( ) { int x, c ; unsigned int y,
cin >> x >> y;
c=pow_1( x, y ) ;
cout  << c;
return 0 ; }
int pow_1( int a, unsigned int b)
{ int p = 1 ;
for ( int  i = 1 ; i <= b ; i ++ )
        p = p * a ;
return p; }
```

Пример 3. Написать программу поиска суммы двух действительных чисел, определения наименьшего из этих чисел. В программе необходимо использовать функции.

```
#include <iostream>
using namespace std;
double sum (double a, double b);    // прототип функции
double min (double a, double b);    // прототип функции
void output (double c, double d);   // прототип функции
int main( ) {
double x, y, m, s;
cout << ”Enter 2 numbers \ n “ ;
cin >> x >> y ;
s = sum ( x, y );       // вызов функции
m = min ( x, y );     // вызов функции
output ( s, m );       // вызов функции
return 0; }
// определение функции sum
double sum (double a, double b) {return a + b;}
// определение функции min
double min ( double a, double b)
{if (a < b)  return a;  else      return b; }
// определение функции output
void output ( double c, double d )
 {cout << ” Sum = ” << c << endl ;
cout << ” Min = ” << d << endl ; }
```

Локальные и глобальные переменные

Переменные, объявленные внутри тела функции, называются локальными и существуют только внутри самой функции. Когда выполнение программы возвращается к выполнению основного кода, локальные переменные удаляются из памяти. Переменные, объявленные в качестве параметров функции, также называются локальными. Каждая переменная характеризуется своей областью видимости, определяющей время жизни и доступность этой переменной в программе.

Переменные, объявленные внутри некоторого блока программы, имеют область видимости, ограниченную этим блоком. Блок в программе выделяется фигурными скобками. К этим переменным можно получить доступ только внутри этого блока. После завершения блока объявленные в нем переменные удаляются.

Переменные, объявленные вне тела какой либо функции (в том числе главной функции), называются глобальными.

Глобальные переменные используются в тех случаях, если требуется сделать данные доступными для многих функций, а передавать эти данные в качестве параметров функции затруднительно. Глобальные переменные следует использовать очень осторожно, так как в результате общедоступности

они могут быть изменены любой функцией и это изменение может оказаться не замеченным пользователем.

Если имя локальной переменной совпадает с именем глобальной переменной, то локальная переменная во время выполнения того блока, где она объявлена, перекрывает глобальную переменную.

Примеры использования локальных и глобальных переменных

Пример 1. Использование локальных переменных и параметров функции

```
# include < iostream >
 using namespace std;
 float Convert ( float ) ;
 int main ( )
    { float TempFer ;
float TempCel ;
cout  <<  "Please enter the temperature in Fahrenheit: " ;
          cin >> TempFer ;
          TempCel  = Convert ( TempFer ) ;
          cout  <<  "\n Here's the temperature in Celsius: " ;
          cout  <<  TempCel  <<  endl ;
          return 0 ;   }
   float Convert ( float TempFer )
   { float TempCel ;
     TempCel = ( ( TempFer – 32 ) * 5) / 9;
     return TempCel ;       }
```

Результат программы на экране будет иметь вид:

```
Please enter the temperature in Fahrenheit: 212
 Here's the temperature in Celsius: 100
```

Пример 2. Использование глобальных и локальных переменных

```
# include  < iostream>
using namespace std;
void myFunction();          // прототип функции
int x = 5, y = 7;           // глобальные переменные
int main ( )
 {cout  <<  " x from main: " <<  x  <<  " \n ";
         cout  <<  " y from main: " <<  y  <<  " \n \n " ;
              myFunction ( ) ;
               cout  <<  " Back from myFunction! \n \n " ;
               cout  <<  " x from main: " << x << " \n " ;
               cout  <<  " y from main: " << y  << " \n " ;
               return 0;       }
 void myFunction ( )
 {      int y = 10;
    cout << "x from myFunction: " << x << " \n ";
    cout << "y from myFunction: " << y << " \n \n ";  }
```

Результат программы на экране будет иметь вид:

```
 x from main: 5
y from main: 7

x from myFunction: 5
y from myFunction: 10

Back from myFunction!

x from main: 5
y from main: 7
```

Пример 3. Использование переменных, объявленных внутри блока

```
# include  < iostream >
using namespace std;
void myFunc ( );
int main ( )
   {   int x = 5;
     cout  << "\n In main x is: " << x;
     myFunc ( ) ;
     cout << " \n Back in main, x is: " << x;
      return 0;    }
   void myFunc( )
   { int x = 8 ;
     cout  <<  "\n In myFunc, local x: " << x << endl ;
     {  cout << "\n In block in myFunc, x is: " << x;
        int x = 9;
        cout << "\n Very local x: " << x;  }
      cout << "\n Out of block, in myFunc, x: " << x << endl;  }
```

Результат программы на экране будет иметь вид:

```
In main x is: 5
In myFunc, local x: 8

In block in myFunc, x is: 8
Very local x: 9
Out of block, in myFunc, x: 8

Back in main, x is: 5
```

Проанализируйте в приведенных примерах изменения значений переменных x и y до входа в функцию, во время выполнения функции, после выхода из неё и значений переменных, ограниченных блоком.

Задания к самостоятельной работе

1. Напишите функцию integerPower(base,exponent), которая возвращает значение $base^{exponent}$. Например, integerPower(3,4)=3*3*3*3. Предполо-

жим, что exponent является положительным ненулевым целым, а base целым. Для управления вычислением функция integerPower(base,exponent) должна применять цикл for. Не используйте никаких функций математической библиотеки.

2. Напишите функцию multiple для двух целых, которая определяет, кратно ли второе число первому. Функция должна получать два целых аргумента и возвращать 1, если второе число кратно первому и 0 в противном случае. Используйте эту функцию в программе, которая вводит серию пар целых чисел.

3. Напишите функцию, которая выводит у левой границы экрана сплошной квадрат из символов заданного символа (например, *), сторона которого определяется целым параметром side. Например, если side равно 4, функция выведет следующее изображение

```
* * * *
* * * *
* * * *
* * * *
```

4. Даны действительные числа s и t.

Используйте функции и вычислите

$$f(t,-2s,1.17)+f(2.2,t,s-t)\text{, где } f(a,b,c)=\frac{2a-b-\sin c}{5+|c|}$$

5. Даны действительные числа a,b,c. Использовать функции и получить

$$\frac{\max(a,a+b)+\max(a,b+c)}{1+\max(a+bc,1.15)}$$

6. Напишите программу, которая вводит последовательность целых чисел и передает их по одному функции even, использующей операцию вычисления остатка для определения четности числа. Функция должна принимать целый аргумент и возвращать 1, если аргумент — четное число, и 0 — в противном случае.

7. Говорят, что целое число является совершенным, если его сомножители, включая 1 (но не само число) в сумме дают это число. Например, 6 — это совершенное число, так как 6 = 1 + 2 + 3. Напишите функцию perfect, которая определяет, является ли параметр number совершенным числом. Используйте эту функцию в программе, которая определяет и печатает все совершенные числа в диапазоне от 1 до 1000. Напечатайте сомножители каждого совершенного числа, чтобы убедиться, что число действительно совершенное.

8. Говорят, что целое число является простым числом, если оно, делится только на 1 и на само себя. Например, 2, 3, 5 — простые числа, а 4, 6, 8 — нет.

а) Напишите функцию, определяющую, является ли число простым или нет.

b) Используйте эту функцию в программе, которая определяет и печатает все простые числа, лежащие в диапазоне от 1 до 10000.

c) Вначале вы могли бы подумать, что верхней границей, до которой вы должны проводить проверку, чтобы увидеть, является ли число n простым, является n/2, но в действительности вам нужно проверить количество чисел, равное корню квадратному из n. Почему? Перепишите программу и запустите ее для обоих способов. Оцените улучшение производительности.

9. Напишите программу, моделирующую бросание монеты. Для каждого броска монеты программа должна печатать **Орел** или **Решка.** Промоделируйте с помощью этой программы бросание 100 раз и подсчитайте, сколько раз появилась каждая сторона монеты. Напечатайте результаты. Программа должна вызывать отдельную функцию **flip,** которая не принимает никаких аргументов и возвращает 0 для **Орла** и 1 для **Решки.**
Замечание: если программа действительно моделирует бросание монеты, каждая сторона монеты должна появляться примерно в половине случаев.

Лабораторная работа № 7

Указатели

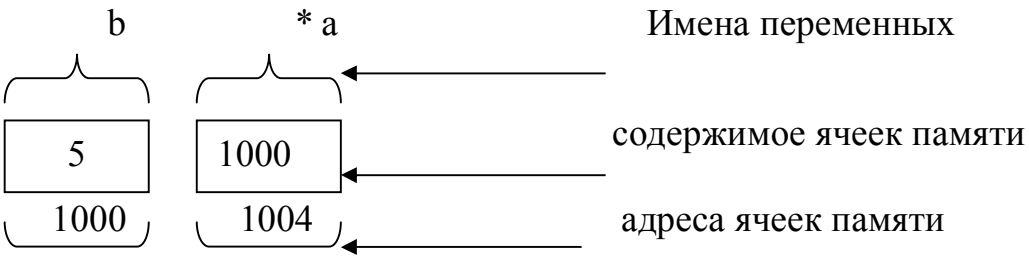
Указатели представляют собой переменные, значениями которых являются адреса памяти. Обычная переменная содержит конкретное значение, т.е. переменная ссылается непосредственно на это значение, а указатель содержит адрес, по которому находится конкретное значение переменной.

Синтаксис объявления указателя:

```
int *a ;        // объявленный указатель ссылается на объект целого типа.
```

После объявления указатель должен быть инициализирован. Указатель инициализируется адресом.

```
int * a ; int b = 5;    //  в данном случае * показывает объявление указателя, во всех
                        // остальных случаях * указывает на операцию косвенной адресации.
a = & b ;               //  инициализация указателя адресом переменной b.
```



Для работы с указателями используются две операции:

* - операция косвенной адресации или операция разыменования,

& - операция взятия адреса (возвращает адрес своего операнда).

Программа на рассмотренный выше пример:

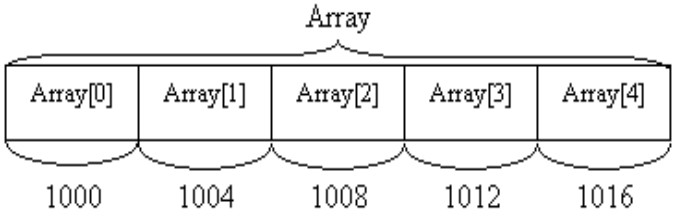
```
# include < iostream>
using namespace std;
int main ( ) {
  int * a ;
  int b = 5 ;
  a = & b ;            // Выполняется операция взятия адреса.
  cout << a << endl ;  // Выведется адрес ячейки памяти, в которой хранится
                       //  переменная “b”.
  cout << * a << endl ; // Выведется значение переменной b, на которую ссылается
                       // указатель.
  return 0 ;}
```

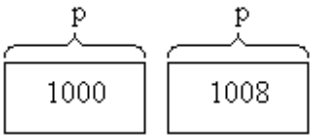
При объявлении указатель можно инициализировать адресом ячейки памяти.

```
…
int b ;
int * a = & b ;  // ~~* a = & b~~ – нельзя!
…
```

Арифметические операции над указателями

Над указателями можно выполнять следующие арифметические операции: + , + = , - , - = , ++, --. Кроме того, можно вычислять разность 2-х указателей. Все операции можно выполнять только при работе с массивами.






```
int Array [ 5 ] , * p ;
p = & Array [ 0 ] ;
p += 2 ;                    // 1000+2*4=1008
```

В С++ под тип “int” выделяется 4 байта. Поэтому при выполнении операции сложения, указатель увеличится не на число 2, а на произведение числа и размера ячейки (4 байта) и будет после выполнения данной операции ссылаться на элемент массива Array[2], т.е. результат арифметических операций над указателями зависит от размера ячейки памяти, на которую ссылается указатель.

Например, при выполнении операции инкремента указатель будет ссылаться на следующий элемент массива, а при операции декремента - на предыдущий. При выполнении разности 2-х указателей, результатом будет число элементов массива, расположенных, начиная с адреса, хранящегося в первом указателе, до адреса, хранящегося в другом указателе.

```
int A [ 5 ] , * v , * p ;
v = & A [ 0 ] ;
p = & A [ 2 ] ;
x = p – v ; // x = 2 ( 1008 – 1000 = 8 / 4 = 2 )
```

При работе с указателями, следует помнить:

- указатель может быть присвоен другому указателю, но в этом случае обязательно должны совпадать типы указателей;
- нельзя разыменовывать указатель типа “void”, так как такой указатель содержит адрес для неизвестного типа данных.

Чтобы правильно разыменовывать указатель, компилятор должен знать размер этого элемента в байтах.

Передача параметров в функцию по ссылке

Различают три способа передачи параметров в функцию:

1. По значению;
2. По указателю;
3. По ссылке.

Два последних способа очень похожи и вместе их называют передачей параметров в функцию по ссылке.

1. Обычные функции, принимающие параметры по значению, могут вернуть в вызывающую функцию одно или ни одного значения. При передаче параметров по значению в функцию передаются не сами переменные, а их копии, поэтому при выходе из функции переменные в главной функции не меняют значения. Мы рассмотрели такие функции в предыдущей теме, посмотрим еще раз на примере.

Пример передачи параметров в функцию по значению:

```
# include  < iostream >
using namespace std;
void swap ( int x , int y ) ;
  int main ( )
  { int x = 5, y = 10 ;
   cout  <<  "Main. Before swap, x: " << x << " y: " << y << " \n";
   swap ( x , y ) ;
   cout << "Main. After swap, x: " << x << " y: " << y << " \n";
     return 0 ;
  }
  void swap ( int x, int y )
  {  int temp;
     cout << "Swap. Before swap, x: " << x << " y: " << y << " \n";
     temp = x;
      x = y;
      y = temp;
      cout << "Swap. After swap, x: " << x << " y: " << y << " \n";
  }
```

Результат программы на экране будет иметь вид:

```
Main. Before swap, x: 5 y: 10
Swap. Before swap, x: 5 y: 10
Swap. After swap, x: 10 y: 5
Main. After swap, x: 5 y: 10
```

Проанализируйте изменения значений переменных x и y до обращения к функции, во время выполнения функции и после выхода из неё.

2. В случае, если требуется, чтобы функция изменяла значения переменных или возвращала несколько значений, необходимо использовать передачу параметров по указателю или по ссылке. Рассмотрим на примерах.

Пример. Требуется написать функцию, которая принимает два параметра (два целых числа) и вычисляет сумму и произведение этих чисел.

```
# include < iostream>
using namespace std;
void SP ( int a , int b , int *S , int *P );
int main( ) {
int x , y , z , R ;
cin >> x >> y ;
SP ( x , y , & z , & R ) ;
cout << ” s u m = ” << z ;
cout << ” p r o d u c t = ” << R ;
return 0 ;
}
void SP ( int a , int b , int *S , int *P )
{  *S = a + b ;
 *P = a * b ; }
```

В примере функция имеет смешанный список параметров. Первые два параметра передаются по значению, вторые два - по указателю.

При выполнении функции main ( ) в памяти создаются 4 локальные переменные x, y, z, R. При вызове функции SP в памяти будут созданы две переменные a и b и две переменные указатели S и P. В переменные a и b будут скопированы значения, хранящиеся в переменных x и y, а в переменные указатели S и P с помощью операции взятия адреса, будут скопированы адреса ячеек z и R.

Зная адреса ячеек z и R и используя операцию косвенной адресации, функция SP запишет результаты произведения и суммы двух чисел непосредственно в ячейки z и R.

3. В случае вызова по ссылке оператор вызова дает вызываемой функции возможность прямого доступа к передаваемым данным, а также возможность изменения этих данных. Вызов по ссылке хорош в смысле про-

изводительности, потому что он исключает накладные расходы на копирование больших объемов данных.

Ссылочный параметр — это псевдоним соответствующего аргумента. Чтобы показать, что параметр функции передан по ссылке, после типа параметра в прототипе функции ставится символ амперсанда (&); такое же обозначение используется в списке типов параметров в заголовке функции. Например, объявление ***int &count*** в заголовке функции может читаться как «**count** является ссылкой на int». В вызове функции достаточно указать имя переменной и она будет передана по ссылке. Тогда упоминание в теле вызываемой функции переменной по имени ее параметра в действительности является обращением к исходной переменной в вызывающей функции и эта исходная переменная может быть изменена непосредственно вызываемой функцией.

Для передачи параметров по ссылке эта программа примет вид:

```
# include < iostream>
using namespace std;
void SP ( int a , int b , int & S , int & P ) ;
int main ( ) {
int x , y , z , R ;
cin >> x >> y ;
SP ( x , y , z , R ) ;
cout << ” s u m = ” << z ;
cout << ” p r o d u c t = ” << R ;
return 0 ;
}
void SP ( int a , int b , int & S , int & P )
{  S = a + b ;
 P = a * b ;}
```

В отличие от предыдущего способа, функция SP не создает дополнительных переменных S и P , а получая через список параметров адреса ячеек z и R , непосредственно записывает в них результат.

Задания к самостоятельной работе

1. Даны два катета прямоугольного треугольника. Написать функцию, определяющую гипотенузу и площадь треугольника. Функция не должна возвращать значение. Результат передается в главную программу main через список параметров. Функция должна иметь 4 параметра: 1-й катет, 2-й катет, гипотенузу, площадь. Реализовать передачу двух последних параметров по ссылке и по указателю.
2. Даны радиус основания и высота цилиндра. Написать функцию, вычисляющую площадь основания и объем цилиндра. Результат в главную программу передается через список параметров по указателю или по ссылке.
3. Даны курсы доллара и рубля к сому. Написать функцию, определяющую количество денег в долларах и рублях. Функция имеет 5 парамет-

ров: курс доллара, курс рубля, количество денег в сомах, в долларах и рублях.

4. Напишите функцию, которая воспринимает время как три целых аргумента (часы, минуты я секунды) и возвращает количество секунд, прошедших со времени, когда часы в последний раз показали 12. Используйте эту функцию для вычисления интервала времени в секундах между двумя моментами, находящимися внутри двенадцатичасового цикла.

5. Разработайте следующие целые функции:

a) функцию Celsius, которая возвращает температуру по Цельсию, эквивалентную температуре по Фаренгейту;

b) функцию Fahrenheit, которая возвращает температуру по Фаренгейту, эквивалентную температуре по Цельсию;

c) используйте эти функции для написания программы, которая печатает таблицу, показывающую эквивалент по Фаренгейту всех температур по Цельсию от 0 до 100 градусов и эквивалент по Цельсию всех температур по Фаренгейту от 32 до 212 градусов. Сделайте листинг в аккуратном табулированном формате, с минимальным количеством строк при сохранении хорошей читаемости.

6. Напишите функцию quality_Points, которая вводит среднюю оценку студентов и если она в диапазоне 90-100, то возвращает 4, если в диапазоне 80-89, возвращает 3, если в диапазоне 70-79, возвращает 2, если в диапазоне 60-69, возвращает 1, и если средняя оценка меньше 60, то возвращает 0.

Лабораторная работа № 8
Массивы как параметры функции

Рассмотрим случаи, когда параметрами функций являются массивы фиксированного и произвольного размеров.

При **передаче в функцию одномерного массива фиксированного размера** в прототипе и заголовке функции указывается тип массива, его имя и количество элементов. При вызове функции указывается только имя массива.

```
 #include < iostream >
using namespace std;
void input ( int  B [ 10 ] ); // функция ввода одномерного массива
int main ( ) {
int A [ 10 ] ;
input ( A ) ;
return 0 ; }
void input ( int B [ 10 ] )
{ for ( int i = 0; i < 10 ; i ++ )
        cin >> B [ i ] ; }
```

Передача в функцию двумерного массива фиксированного размера выполняется аналогично, как и для случая одномерного массива, только добавляется второй индекс, например void input(int B [10] [10]) ; .

В случае **передачи одномерного массива произвольного размера** функция имеет два параметра:

1. Массив без указания количества элементов;
2. Количество элементов массива.

```
# include  < iostream>
using namespace std;
void input ( int B [ 10 ] , int n ) ;
int main ( ) {
int A [ 100 ] , n ;
cin >> n ;
input ( A , n ) ;
return 0;
}
void input ( int B [ ] , int n )
{ for ( int i = 0; i < n ; i ++ )
cin >> B [ i ] ;}
```

В случае **передачи двумерного массива произвольного размера** массив может иметь только произвольное количество строк, количество столбцов должно быть фиксированным. Функция будет иметь два параметра:

1. массив с конкретным количеством столбцов;
2. количество строк.

```
 # include < iostream>
using namespace std;
void input ( int B [ ] [ 5 ] , int n ) ;
int main( ) {
int A [ 100 ] [ 100 ] , n ;
cin >> n ;
input ( A , n ) ;
return 0 ;
}
void input ( int  B [ ] [ 5 ] , int n )
{ for ( int i = 0 ; i < n ; i ++ )
        for ( int j = 0 ; j < n ; j ++ )
                cin >> B [ i ] [ j ] ; }
```

Локальное и динамическое распределение памяти

Локальное распределение памяти – переменной или объекту выделяется память в программном стеке.

Стек – область в оперативной памяти, выделяемая программой при её запуске.

Вся память, необходимая программе для создания переменных, вызовов функций и прочих действий, берётся из стека. Эта память высвобождается при завершении работы программы, при выходе из функции или при выходе из локального блока.

Преимущество стека - память в стеке выделяется очень быстро.

Недостаток стека - стек имеет фиксированный размер, который не может быть изменён во время выполнения программы.

Для переменных встроенных типов и небольших массивов достаточно использовать локальное распределение памяти, т.е. стек. Но при работе с большими массивами, структурами и классами, рекомендуется использовать **динамическое** распределение памяти. Динамическая память складывается из свободной оперативной памяти и свободного пространства на жёстком диске. Обращение к динамической памяти происходит немного медленнее, чем к памяти из стека. Динамическая память – это виртуальная память компьютера.

В C++ динамическое распределение памяти осуществляется через оператор “new”, который выделяет память под динамический объект. Оператор “delete” удаляет объект из динамической памяти. После оператора “new” должен быть указан тип объекта, который будет размещён в памяти.

```
int *a;
a = new int; // Указатель ссылается на ячейку в области динамической памяти
```

Когда память, выделяемая под объект, не используется, её необходимо очистить: delete a; // очищение ячейки памяти, на которую ссылается указатель.

Динамическое распределение памяти чаще всего используется для работы с динамическими массивами.

Динамический массив – массив, размер которого заранее не известен и задается в процессе выполнения программы.

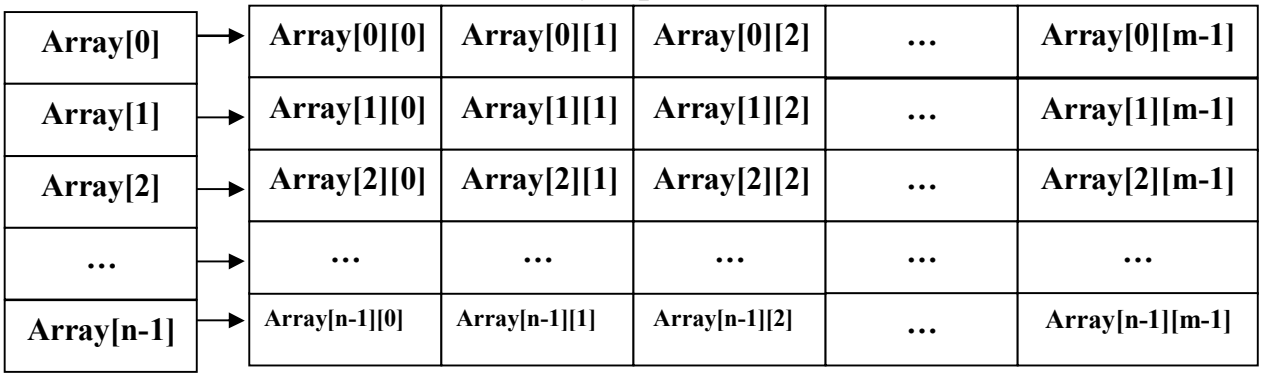
```
int  * Array, n;
cin >> n ;
Array = new int [ n ] ;        // Выделение памяти под одномерный динамический массив,
…                              // при этом “Array” содержит адрес первого элемента.
delete [ ] Array ;             // Освобождение динамической памяти.
```

[ ] - указывают на то, что будет удалён массив, без них будет удалён только первый элемент массива.

Пример: Объявление одномерного динамического массива, выделение памяти, ввод данных и очищение памяти.

```
# include  < iostream >
using namespace std;
int main ( )
{ double * dinArray ;
  int n, i ;
  cout << ”Enter size of array \n” ;
  cin >> n ;
  dinArray = new double [ n ] ;
  for ( i = 0 ; i < n ; i ++)
    cin >> dinArray [ i ] ;
  delete [ ] dinArray ;
  return 0; }
```

Выделение памяти под двумерный динамический массив




```
# include < iostream >
using namespace std;
int main ( ) {
int **Array;                    // Указатель на массив указателей
int n, m, j, i;
cout << ”Enter number of rows \n ” ;
cin >> n ;
cout << ”Enter number of columns \n ” ;
cin >> n ;
Array = new int * [ n ];  //Выделяется память под одномерный массив,
                  //каждый элемент которого, является указателем
for ( int i = 0 ; i < n ; i ++)
Array[ i ] = new int [ m ];  // Каждому указателю присваивается  адрес
                             //первой ячейки памяти, выделяемой под “m”
                               //  элементов массива
  for ( i = 0 ; i < n ; i ++)          // Ввод 2-мерного массива
    for ( j = 0 ; j < m ; j ++)        // Ввод 2-мерного массива
      cin >> Array [ i ] [ j ];        // Ввод 2-мерного массива
  for  (i = 0 ; i < n ; i ++)
    delete Array [ i ] ;          // Удаление строк массива
  delete [ ] Array ;              // Удаление массива указателей
  return 0 ; }
```

Передача динамических массивов в функцию производится следующим образом:

```
// Передача одномерного динамического массива в функцию
    void funct ( double *A , int n ) ;
    int main ( ) {
    double * B ;
    int n ;
    cin >> n ;
    B = new double [ n ] ;
    funct ( B , n );
    …
```

```
// Передача двумерного динамического массива в функцию
    void funct ( double **A , int m , int n ) ;
    int main ( ) {
    double **B ;
    int m , n ;
    cin >> n >> m ;
    B = new double * [ n ] ;
          for ( int j = 0 ; j < n ; j ++ )
          B [ j ] = new double [ m ] ;
    funct ( B , n , m ) ;
     …
```

Задания к самостоятельной работе

1. Дан одномерный массив. Написать функцию, определяющую минимальный, максимальный элементы массива и среднее арифметическое минимального и максимального элементов. Кроме того, программа должна иметь функцию ввода одномерного массива и функцию вывода.

2. Написать функцию перемножения матриц A размером nxm и B размером mxl. Элементы результирующей матрицы получить с помощью следующей формулы. $C_{ik} = \sum_{j=1}^{m} A_{ij} B_{jk}$ Массивы должны быть динамическими.

3. Написать функции вычисления суммы элементов каждой строки матрицы A размером 6x6, определения наибольшего значения этих сумм.

4. Дана действительная матрица размера 6x9. Найти среднее арифметическое наибольшего и наименьшего значений ее элементов. Программа должна быть составлена с использованием функций.

5. В квадратной матрице размера mxn найти значение наибольшего по модулю элемента матрицы, а также определить индексы этого элемента. Предполагается, что такой элемент - единственный. Программа должна быть составлена с использованием функций.

6. В данной действительной квадратной матрице порядка N найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный. Программа должна быть составлена с использованием функций.

7. В одномерном массиве, состоящем из n вещественных чисел, вычислить:

а) количество элементов массива, меньших C;

б) сумму положительных элементов, расположенных после первого положительного элемента.

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

Программа должна быть составлена с использованием функций.

Лабораторная работа № 9
Перегрузка функций

Перегрузка функции - это определение двух или более функций с одним и тем же именем, но разным набором параметров. Работа перегруженных функций основана на различии списков параметров (на различии их типов и/или количестве), принимаемых функцией. При обращении к функции компилятор использует то определение функции, тип и количество параметров которой совпадают с типом и количеством параметров, передаваемых в функцию при обращении к ней.

Перегруженные функции не могут отличаться только типом возвращаемого значения.

Пример перегрузки функций, вычисляющих среднее арифметическое значение для двух целых чисел, для трех целых чисел, для двух действительных чисел и трех действительных чисел.

```
# include < iostream >
using namespace std;
double avg ( int a , int b ) ;
double avg ( int a , int b , int c ) ;
double avg ( double a , double b ) ;
double avg ( double a , double b , double c ) ;
int main( )
{
int x , y , z ;
cout << ” Enter 3 integers \n ” ;
cin >> x >> y >> z ;
cout << ” average value of 2 numbers = ” << avg ( x , y ) ;
cout << ” average value of 3 numbers = ” << avg ( x , y , z ) ;
double n , m , k ;
cout << ” Enter 3 real  numbers \n ” ;
cin >> n >> m >> k ;
cout << ” average value of 2 real numbers = ” << avg ( n , m ) ;
cout << ” average value of 3 real numbers = ” << avg ( n , m , k ) ;
return 0 ;
}
double avg ( int a , int b ) { return ( a + b ) / 2.0 ;}
double avg ( int a , int b , int c) { return ( a + b + c ) / 3.0 ; }
double avg ( double a , double b ) { return ( a + b ) / 2 ; }
double avg ( double a, double b, double c) {return ( a + b + c ) /3;}
```

Шаблоны функции

Шаблоны функции позволяют создавать функции, которые могут использовать аргументы различных типов. Большинство функций используют алгоритмы, которые могут работать с различными типами данных. Например, функция сортировки может сортировать символьные, вещественные и целочисленные массивы.

Использовать несколько определений идентичных функций, работающих с данными разного типа, не эффективно. В этом случае следует использовать шаблоны функции. При этом определение и прототип функции начинаются со строки **template < class T >.** Эта строка является префиксом шаблона и указывает компилятору, что следующее за ним определение функции является шаблоном, а **T** - параметром типа. Слово **class** в этом случае обозначает тип. В качестве **T** может быть указан любой тип.

Определение шаблонов функции на самом деле является большим набором количества определений функции. Каждый из этих определений получается замещением параметра типа **T** именем соответствующего типа. Компилятор при обращении к функции сам создает ее определение на основе шаблона. Прежде, чем создавать шаблон функции, необходимо создать обыкновенную функцию, и только после ее отладки преобразовать в шаблон.

Пример: универсальная функция сортировки.

```
# include < iostream >
using namespace std;
template < class T  >
void sort ( T a[ ] , int size ) ;
int main ( ) {
char c [ 8 ] = {‘ p ’, ’ r ’, ’ o ’, ’ g ’, ’ r ’, ’ a ’, ’ m ’, ’ \0 ’ } ;
sort ( c , 8 ) ;
int A [ 5 ] = { 5 , 10 , 1 , 4 , 9 } ;
sort ( A , 5 ) ;
double B [ 4 ] = { 3.2 , 1.5 , 9.8 , 2.8 } ;
sort ( B , 4 ) ;
return 0 ;
}
template < class T > void sort ( T a [ ], int size )
{int i , P ;
T D ;
for ( p = 1 ; p <= size - 1 ; p ++ )
for ( i = 0; i < size - 1 ; i ++ )
if (a [ i ] > a [ i + 1 ] ) {
D = a [ i ] ;
a [ i ] = a [ i + 1 ] ;
a [ i +1] = D ; }
```

Рекурсивные функции

Программы, которые мы обсуждали до сих пор, в общем случае состояли из функций, которые вызывали какие-либо другие функции в строгой иерархической манере. В некоторых случаях полезно иметь функции, которые вызывают сами себя. **Рекурсивная функция** — это функция, которая вызывает сама себя либо непосредственно, либо косвенно с помощью другой функции.

Рекурсивная задача в общем случае разбивается на два этапа. Для решения задачи вызывается рекурсивная функция. Эта функция знает, как решать только простейшую часть задачи — так называемую **базовую задачу** (или несколько таких задач). Если эта функция вызывается для решения базовой задачи, она просто возвращает результат. Если функция вызывается для решения более сложной задачи, она делит эту задачу на две части: одну часть, которую функция умеет решать, и другую, которую функция решать не умеет. Чтобы сделать рекурсию выполнимой, последняя часть должна быть похожа на исходную задачу, но быть по сравнению с ней несколько проще или несколько меньше. Поскольку эта новая задача подобна исходной, функция вызывает новую копию самой себя, чтобы начать работать над меньшей проблемой — это называется **рекурсивным вызовом,** или **шагом рекурсии.** Шаг рекурсии включает ключевое слово **return,** так как в дальнейшем его результат будет объединен с той частью задачи, которую функция умеет решать, и сформируется конечный результат, который будет передан обратно в исходное место вызова, возможно, в **main.**

Шаг рекурсии выполняется до тех пор, пока исходное обращение к функции не закрыто, т.е. пока еще не закончено выполнение функции. Шаг рекурсии может приводить к большому числу таких рекурсивных вызовов, поскольку функция продолжает деление каждой новой подзадачи на две части. Чтобы завершить процесс рекурсии, каждый раз, как функции вызывает саму себя с несколько упрощенной версией исходной задачи, должна формироваться последовательность все меньших и меньших задач, в конце концов, сходящаяся к базовой задаче. В этот момент функция распознает базовую задачу, возвращает результат предыдущей копии функции, и последовательность возвратов повторяет весь путь назад, пока не дойдет до первоначального вызова и не возвратит конечный результат в функцию **main.**

Недостаток рекурсии: повторный запуск рекурсивного механизма вызовов функции приводит к росту накладных расходов: к нарастающим затратам процессорного времени и требуемого объема памяти.

Как пример рассмотрим рекурсивную программу для расчета факториала.

Факториал неотрицательного целого числа n, записываемый как n!, равен n*(n-1)*(n-2)*…*1, причем считается, что 1! = 1 и 0! = 1. Например, 5! вычисляется как **5 x 4 x 3 x 2 x 1** и равен 120.

Факториал целого числа, **number,** большего или равного 0, может быть вычислен итеративно (нерекурсивно) с помощью оператора for следующим образом:

```
factorial = 1;
for   (int counter = number;   counter >=1;   counter--)
factorial  *= counter;
```

Теперь используем рекурсию для вычисления и печати факториалов целых чисел от 0 до 10. Рекурсивная функция **factorial** сначала проверяет, истинно ли условие завершения рекурсии, т.е. меньше или равно 1 значение **number.**

Если действительно **number** меньше или равно 1, **factorial** возвращает 1, никаких дальнейших рекурсий не нужно и программа завершает свою работу. Если **number** больше 1, оператор

return number * factorial (number - 1);

представляет задачу как произведение number и рекурсивного вызова factorial, вычисляющего факториал величины number-1. Отметим, что factorial (number - 1) является упрощенной задачей по сравнению с исходным вычислением factorial (number).

В объявлении функции factorial указано, что она получает параметр типа unsigned long и возвращает результат типа unsigned long. Это является краткой записью типа unsigned long int. Описание языка C++ требует, чтобы переменная типа unsigned long int хранилась по крайней мере в 4 байтах (32 битах), и таким образом могла бы содержать значения в диапазоне по крайней мере от 0 до 4294967295. (Тип данных long int также хранится по крайней мере в 4 байтах и может содержать значения по крайней мере в диапазоне от 2147483647). Функция factorial начинает вырабатывать большие значения так быстро, что даже unsigned long не позволяет нам напечатать много значений факториала до того, как будет превышен допустимый предел переменной unsigned long.

```
// Рекурсивная функция факториала
# include < iostream>
  using namespace std;
 unsigned long factorial ( unsigned long ) ;
            int main ( )
            { for ( int i = 0; i <= 10; i++)
            cout << i << "! = " << factorial (i) << endl;
            return 0;
            }
//  рекурсивное описание функции вычисления факториала
unsigned long factorial ( unsigned long number )
            { if ( number <= 1) return 1;
                  else
                        return ( number * factorial ( number – 1 );
            }
```

Задания к самостоятельной работе

1. Напишите программу, которая использует шаблон функции maximum для поиска максимального из трех целых чисел, трех чисел с плавающей запятой и трех символов.

2. Напишите программу, которая использует шаблон функции по имени min для определения наименьшего из трех аргументов. Проверьте программу, используя пары целых чисел, символов и чисел с плавающей запятой.

3. Определите, содержат ли следующие фрагменты программы ошибки. Для каждой ошибки укажите, как она может быть исправлена.

Замечание: в некоторых фрагментах ошибки могут отсутствовать.

```
a) template < class A >
int sum ( int numl , int num2, int num3 )
 {  return numl + num2 + num3; )
b) void printResults ( int x, int y )
{ cout « "Сумма равна " « x + y « ' \n' ;
      return x + y; }
c) template < class A>
A product ( A numl, A num2, A num3 )
{ return  numl * num2 * num3; }
double cube ( int ) ;
int cube ( int );
```

4. Ряд Фибонначи состоит из чисел, каждое из которых является суммой двух предыдущих(1, 1, 2, 3, 5, 8, 13, …). Найти n-ный элемент ряда, используя рекурсию.

5. Наибольший общий делитель (НОД) двух целых чисел x и y — это наибольшее целое, на которое без остатка делится каждое из двух чисел. Напишите рекурсивную функцию nod, которая возвращает наибольший общий делитель чисел x и y. НОД для x и y определяется рекурсивно следующим образом: если y равно 0, то nod(x, y) возвращает x; в противном случае nod(x, y) равняется nod(y, x % y), где % — это операция вычисления остатка.

Лабораторная работа № 10
Файлы

В C++ ввод и вывод осуществляются через потоки. Поток (stream) - абстрактный канал связи, который создается в программе для обмена данными с файлами и стандартными устройствами ввода-вывода. По направлению передачи данных различают следующие потоки:

1. Входные потоки, из которых извлекаются данные (istream);
2. Выходные потоки, в которые записываются значения из программы (ostream);
3. Двунаправленный поток, который допускает и чтение, и запись.

При подключении библиотечного файла iostream создаются потоки cin и cout. Работа с файлами также осуществляется через потоки, при этом требуется подключить файл fstream. Этот файл позволяет создавать следующие потоки:

ifstream для чтения данных,

ofstream для записи данных,

fstream для чтения и записи данных.

Работу с файлами можно подразделить на 4 этапа:

1. Создание потока (объявление потоковой переменной). Переменные потока ввода из файла имеют тип ifstream, переменные потока вывода в файл имеют тип ofstream.

2. Связывание потока с файлом и открытие файла для работы в определенном режиме. Для связи файла с потоком используется функция ***open().***

3. Обмен данными с файлом через поток: запись в поток, чтение из потока. Информация из потоков считывается с помощью операций “ >> ” - извлечь из потока. Запись информации в файл осуществляется с помощью операции “ << ” - отправить в поток.

4. Разрыв связи потока с файлом: закрытие файла и разрыв его связи с потоком. Файл закрывается с помощью функции close ( ).

Пример. Написать программу, которая считывает две переменные из файла и записывает результат в другой файл.

Для решения этой задачи необходимо создать в программе Блокнот файл T1.txt и записать два числа, а затем запустить программу.

```
# include < fstream>
using namespace std;
int main ( ) {
ifstream in;                 //   объявление входного потока
ofstream out;                //   объявление выходного потока
in.open (“ T1.txt ”);    //  связь потока с файлом
out.open (“T2.txt”);
int a, b, c;
in >> a >> b ;           //   чтение данных из файла
c = a + b ;
out << c ;               //   запись результата в файл
in.close ( ) ;           //  разрыв потока с файлом
out.close ( ) ;          //  разрыв потока с файлом
return 0; }
```

Режимы открытия файла

Файлы могут открываться в различных режимах. Режим открытия файла определяется с помощью определенных констант.

Константа	Режим	Позиция в файле
app	Используется для добавления данных в файл	Конец файла
nocreate	Не создает новый файл. Если файл не существует, операция открытия не выполняется	
noreplace	Не замещать. Если файл существует, то операция его открытия не выполняется	
in	Открытие файла для чтения	Начало файла
out	Открытие файла для записи	Начало файла

Функция проверки успешности открытия файла fail ( )

Если при работе с функцией open() идет обращение к несуществующему файлу, то компилятор создаст новый пустой файл, поэтому необходимо проверять, был ли открыт уже существующий файл. Функция fail( ) из библиотеки fstream.h и результатом этой функции является логическое выражение, если обращение к функции open() было выполнено неудачно, то fail( ) возвращает значение “истина”.

```
# include < iostream>
# include < fstream>
# include < stdlib>
using namespace std;
int main ( ) {
ifstream in;
in.open ( “data.txt”, ios::nocreate ) ;
if (in.fail( ) ) {
cout << ” File doesn’t exist \n ” ;
exit (1);} }
```

Для проверки конца файла применяется функция eof( ) (end of file).

Пример. Требуется переписать последовательность действительных чисел из одного файла в другой.

```
# include < fstream>
using namespace std;
int main ( ) {
ifstream A; ofstream B;
A.open ( “C:\\Documents and Settings\\POVT\\Data.txt” );
B.open ( “C:\\result.txt” );
double a;
while (!A.eof ( ) )       //пока не конец файла
{ A >> a;
B << a << endl ; }
A.close ( );
B.close ( );
return 0; }
```

Условие в скобках оператора while можно написать другим способом, тогда программа будет иметь вид:

```
# include < fstream>
using namespace std;
int main ( ) {
ifstream A;
ofstream B;
A.open ( “C:\\Documents and Settings\\POVT\\Data.txt” );
B.open ( “C:\\result.txt” );
double a;
while (A >> a)       // пока из потока A идут переменные a
{ B << a << endl ; }
A.close ( ); B.close ( );
return 0; }
```

Форматированный вывод данных

Форматирование – преобразование данных в соответствии с установленными параметрами.

При выводе форматирование позволяет получить данные в файле или на экране в определенном формате. При вводе данных форматирование позволяет считывать данные как значения определенного типа.

Для форматирования в языке C++ используются манипуляторы и функции.

Манипулятор – это функция, которая используется во входном или выходном потоке после операций >> и << (взять или отправить в поток). Манипуляторы бывают двух типов: без параметров и с параметрами.

Манипуляторы без параметров находятся в библиотеке iostream.h

Манипулятор	Описание
dec	Используется для перевода числа в десятичную систему счисления
hex	Используется для перевода числа в шестнадцатеричную систему счисления
oct	Используется для перевода числа в восьмеричную систему счисления
endl	Используется для вставки конца строки, выгружения из буфера
ends	Вставляет нулевой признак конца строки

Пример использования:

```
int a = 10;
cout << oct << a << ends ;
cout << hex << a << endl ;
cout << dec << a ;
```

Манипуляторы с параметрами находятся в библиотеке iomanip.h

Манипулятор	Описание
setw(n)	Устанавливает ширину поля вывода размером в **n** позиций
setfill(c)	Устанавливает символ - «заполнитель» **c**
setprecision(n)	Устанавливает точность при выводе действительных чисел
setiosflags(flag)	Используется для установки флагов форматирования
resetiosflags(flag)	Используется для сброса флагов форматирования

Флаги форматирования устанавливают в потоке правила форматирования.

Флаг	Описание
fixed	Используется для вывода вещественных чисел в формате с фиксированной запятой
scientific	Используется для представления вещественных чисел с плавающей запятой
showpoint	Используется для отображения в числе десятичной точки, даже в тех случаях, когда, дробная часть равна нулю
showpos	Используется для вывода знака «+» перед положительными значениями

Пример использования:

```
double a = 2.5 ;
cout << setw(10) << setfill ( '*') << setprecision (2) ;
cout << setiosflags (ios::showpoint) << a << endl ;
a = 5E-10;
cout << setw (10) << setfill ( '*' ) << setprecision (2) ;
cout << setiosflags ( ios::showpoint | ios::scientific | ios::showpos ) << a ;
cout << resetiosflags ( ios::showpos ) << a ;
```

Функции форматирования

Основное отличие использования функций форматирования от манипуляторов заключается в формате обращения к функции. Манипулятор используется после операции " << " , функция используется после операции "." (cout.функция; cout<<манипулятор).

Функции выполняют те же действия что и манипуляторы.

Функция	Описание
width(n)	Устанавливает ширину поля в n позиций
fill(c)	Устанавливает символ «заполнитель» **c**
precision(n)	Устанавливает точность при выводе действительных чисел
setf(flags)	Используется для установки флагов форматирования
unsetf(flags)	Используется для сброса флагов форматирования

Пример использования:

```
double a=2.5;
cout.widht (10) ;
cout.fill ( '*' );
cout.precision (2) ;
cout.setf (ios::showpoint | ios::scientific | ios::showpos ) ;
cout << a ;
```

Задания к самостоятельной работе

1. Записать в двумерный массив размера 8X8 случайные вещественные числа, значения которых от 0 до 100. Вывести полученный массив на экран и записать в файл в виде выровненной матрицы с двумя знаками после запятой.

2. В файле есть сведения об автомобилях: марка автомобиля, номер и фамилия владельца.

а) Вывести сведения о владельцах и номерах автомобилей каждой марки автомобиля.

б) Подсчитать количество автомобилей каждой марки.

3. Текст записан одной длинной строкой. Признаком красной строки служит символ $. Переформатировать текст в 60-символьные строки, формируя абзацы. Исходный текст должен быть взят из файла, название которого будет введено с клавиатуры, а результирующий текст должен быть выведен в файл «Result_file.txt».

4. Текст, не содержащий собственных имён и сокращений, набран полностью прописными буквами. Заменить все прописные буквы, кроме букв, стоящих после точки, строчными буквами. Исходный текст должен быть взят из файла, название которого будет введено с клавиатуры, а результирующий текст должен быть выведен в файл «Result_file.txt».

5. За стоянку до трех часов парковочный гараж запрашивает плату минимум $2.00. В случае стоянки более трех часов гараж дополнительно запрашивает $0.50 за каждый полный или неполный час сверх трех часов. Максимальная плата за сутки составляет $10.00. Допустим, что никто не паркуется более, чем на сутки за раз. Напишите программу расчета и печати оплаты за парковку для каждого из трех клиентов, которые парковали свои автомобили вчера в этом гараже. Вы должны вводить длительность парковки для каждого клиента. Ваша программа должна печатать результаты в аккуратном табулированном формате и должна рассчитывать и печатать общий вчерашний доход. Программа должна использовать функцию calculateCharges, чтобы определять плату для каждого клиента. Результаты работы должны представляться в следующем формате:

Автомобиль	Часы	Плата
1	1.5	2.00
2	4.0	2.50
3	24.0	10.00
Итого	29.5	14.50

Лабораторная работа № 11

Структуры

Структура – одна или несколько переменных, которые для удобства работы с ними сгруппированы под одним именем. В отличие от массивов переменные структуры могут иметь различный тип.

Структура – абстрактный тип данных, т.е. тип данных, определяемый пользователем.

Синтаксис определения структуры:

struct имя
{ список переменных;
};

Ключевое слово struct указывает на то, что далее приведено имя структуры как нового типа данных. Имя структуры иначе называют телом структуры.

Идентификаторы, объявленные внутри фигурных скобок, называются элементами структуры. Элементы одной и той же структуры должны иметь уникальные имена. Структуры, как и глобальные переменные, должны быть объявлены вне каких-либо функций.

Структуры используются в организации сложных данных, так как позволяют представлять группу связанных между собой переменных не как отдельные элементы, а как единое целое.

Примеры структур: точка, описываемая двумя координатами x и y; структура «платежная ведомость», содержащая сведения о служащем: имя, адрес, зарплата, номер карточки и так далее.

Программа, объявляющая точку с координатами x и y, используя структуру:

```
# include < iostream>
# include < cmath>
using namespace std;
struct Point {
int x ;
int y ;};
int main ( ) {
Point A , B;  // объявление двух точек A и B с координатами x и y.
cout << ” Enter coordinates of point A \n ” ;
cin >> A.x >> A.y ;
cout << ” Enter coordinates of point B \n ” ;
cin >> B.x >> B.y ;
double d  ;
d = sqrt ( pow ( ( A.x - B.x ) , 2 ) + pow ( ( A.y - B.y ) , 2 ) ) ;
cout << d ;
return 0 ; }
```

Доступ к отдельным элементам структуры производится с помощью операции “точка” (”.”), которая называется оператором прямого доступа к элементам структуры.

```
Point A;
A.x=1;
A.y=2;
```

Существует вторая операция доступа к элементам структуры, которая называется “операция указателя структуры” и обозначается через знак ->. Эта операция используется, если переменная была объявлена как указатель структурного типа.

```
Point * A;
A->x=-5;
A->y=0;
```

или

```
(*A).x=-5;
(*A).y=0;
```

Скобки необходимы в этой записи, потому что операция “.” имеет более высокий приоритет, чем операция косвенной адресации “*”.

Так же, как и любую другую переменную, переменную структурного типа можно одновременно объявлять и инициализировать:

```
Point A={-1,0};
Point B={-2,-6};
```

Значения переменных структуры присваиваются в порядке их объявления при определении структуры.

Структуры внутри структур

Допускается объявление вложенных структур. Это означает, что структура может содержать 1 или несколько элементов структурного типа. Пример:

```
struct MD {
        int headers;
        int cylinders;
        int speed;
        char name[10];
        double size;
    };
struct Computer {
    char model[20];
    int memory;
    int diagonal;
    MD disk; };
```

Для доступа к элементу структуры MD используется следующая запись:

```
Computer A;
A.disk.size=20;
```

Структуры в качестве параметров функции

Структуру можно передать в функцию как параметр. При этом используется передача параметра по ссылке, то есть в структуру передается адрес переменной структурного типа.

```
# include < iostream>
using namespace std;
struct Student  {
int ID ;
char name [ 10 ] ;
int age ;
} ;
void input ( Student * ) ;
int main ( ) {
Student List ;
input ( &List ) ;
cout << List.ID << " "<<List.name<<"  " << List.age<<endl ;
return 0 ;
}
void input ( Student *L ) {
cin >> L->ID ;
cin >> L->name ;
cin >> L->age ;
}
```

Задания к самостоятельной работе

1. Описать структуру с именем Student, содержащую следующие поля: name, group, ses(успеваемость - массив из 5 элементов). Ввести с клавиатуры данные в массив stud1, состоящий из 10 структур типа student. Записи упорядочить по алфавиту. Вывести на экран фамилии и номера групп для всех студентов, имеющих оценки 4 и 5, если таких нет, то вывести соответствующее сообщение.

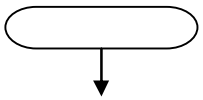
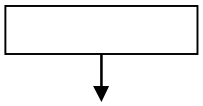
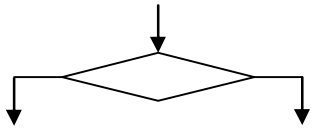
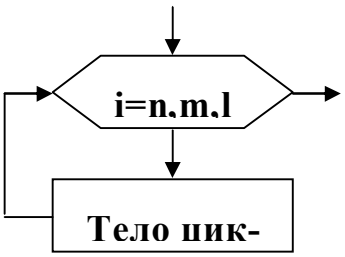
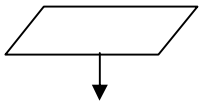
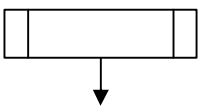
2. Описать структуру с именем Train, содержащую следующие поля: nazn(название пункта назначения), num(номер поезда), time(время отправления). Ввести с клавиатуры данные в массив rasp, состоящий из 6 структур типа Train. Записи упорядочить по номерам поездов. Вывести на экран информацию о поезде, номер которого введен с клавиатуры, если такого поезда нет, то вывести соответствующее сообщение.

3. Описать структуру с именем worker, содержащую следующие поля: name, pos(должность), year(год поступления на работу). Ввести с клавиатуры данные в массив tabl, состоящий из 10 структур типа worker. Записи упорядочить по дате поступления на работу. Вывести на экран фамилии работников, чей стаж работы превышает значение, введенное с клавиатуры, если таких нет, то вывести соответствующее сообщение.

Приложение 1.

Существует несколько способов записи алгоритмов. Наиболее популярен графический способ записи – “блок-схема”.

Блок-схема – последовательность блоков предписывающих выполнение определённых действий и связи между ними.

Основные блоки “блок-схемы”		
Наименование	Обозначение	Функции
Пуск, остановка		Начало, конец, остановка программы, вход и выход из функции (подпрограммы)
Процесс		Выполнение одной или группы операций
Решение		Проверка условия, выбор направления выполнения алгоритма
Модификация		Цикл
Ввод/вывод		Ввод данных и вывод результата
Предопределённый процесс		Вызов функции (подпрограммы)
Соединитель		Разрыв линий схемы алгоритма
Комментарий	- - - [	Пояснение в схеме алгоритма, формулы

Литература

1. Харви Дейтел, Пол Дейтел. Как программировать на C++. Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с.
2. Марченко А.Л. C++. Бархатный путь
3. М. Эллис, Б. Страуструп. Справочное руководство по языку C++ с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
4. Э.А.Ишкова. C++. Начала программирования – М.: ЗАО «Издательство БИНОМ»,2000. 304 с.
5. Стенли Б. Липпман. C++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.
6. Бруно Бабэ. Просто и ясно о Borland C++: Пер. с англ. - Москва: БИНОМ, 1994. 400с.
7. В.В. Подбельский. Язык C++: Учебное пособие. - Москва: Финансы и статистика, 1995. 560с.
8. Т. Сван. Освоение Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996. 544с.
9. Г. Шилдт. Самоучитель C++: Пер. с англ. - Санкт-Петербург: BHV-Санкт-Петербург, 1998. 620с.
10. У. Сэвитч. C++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. 736с.