КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ имени АЛЬ-ФАРАБИ

Б.А. Урмашев
Т.А. Шмыгалева

ПРОГРАММИРОВАНИЕ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ С++

Учебное пособие для студентов
Специальностей «Информатика»,
«Вычислительная техника и программное обеспечение»

Алматы
«Қазақ университеті»
2020

УДК
ББК
С

Рекомендовано к изданию Ученым советом факультета информационных технологий и РИСО КазНУ имени аль-Фараби (Протокол № 4 от 19.06.2020 г.)

Рецензенты:
кандидат физико-математических наук, профессор ***А.Ю. Пыркова***
кандидат технических наук, ассоциированный профессор ***Е.Г. Сатимова***

Урмашев Б.А.

Программирование на алгоритмическом языке С++: учебное пособие / Б.А.Урмашев, Т.А. Шмыгалева. – Алматы: Казақ университеті, 2020. – с.

ISBN 978-601-04-4659-5

В данном пособии изложены основы программирования на языке С++, описаны приемы программирования, а также теоретическая и практическая информация по предмету «Программирование на алгоритмическом языке С++». Пособие содержит полезные методические рекомендации по выполнению заданий на языке программирования С++. Пособие рассчитано на бакалавров университетов, специализирующихся по специальностям «Информатика», «Вычислительная техника и программное обеспечение».

УДК
ББК

ISBN 978-601-04-4659-5

ВВЕДЕНИЕ

В настоящее время происходит автоматизация различных сфер деятельности человека: банков, медицинских учреждений, различных производственных подразделений, образования, науки, социальной сферы, торговли и других отраслей. В связи с чем необходимо знание хотя бы одного языка программирования. Современным языком программирования является язык С++. На основе его далее можно изучать C#, Java, python и другие языки программирования. В данном пособии приводятся методические рекомендации по изучению основ языка С++.

В первой теме описываются основные понятия языка С++, типы данных, операторы ввода, вывода, присваивания. Вторая тема посвящена разветвляющимся операторам: условному и оператору выбора. В темах 3-5 рассматриваются операторы цикла с предусловием и с постусловием, необходимость применения конкретного оператора цикла, инструкции перехода. В теме 6 изучаются одномерные массивы, в 7-й – двумерные массивы. В теме 8 достаточно подробно описано использование функций, синтаксис, функции с параметрами и без параметров, возвращающие и не возвращающие значения, с параметрами по умолчанию, рекурсивные функции, перегрузка функций. В 9,10-й темах рассматривается передача массивов в функцию без указателей. В 11-й теме описывается использование указателей на переменные, в 12-й – указателей на одномерные массивы, в 13-й на двумерные. В 12-й и 13-й темах также рассматриваются одномерные и двумерные динамические массивы и их передача в функцию с помощью указателей. 14-я тема посвящена работе с файлами, 15-я работе со строками. В учебном пособии достаточно подробно описаны классы памяти и область видимости переменных, локальные и глобальные переменные, переменные на внутреннем и внешнем уровне, работа с несколькими файлами. Все темы сопровождаются пояснениями и большим количеством примеров с комментариями. В конце каждой работы приводятся тексты заданий по каждой теме.

Тема № 1

ОПЕРАТОР ПРИСВАИВАНИЯ. ВВОД И ВЫВОД ДАННЫХ В ЯЗЫКАХ С, С++

Происхождение языка С начинается с операционной системы UNIX, поскольку она и большинство программ для нее написаны на языке С. Благодаря популярности UNIX язык С был признан в среде программистов как язык системного программирования, который можно использовать для написания компиляторов и операционных систем. В то же время он удобен для создания многих прикладных программ. Язык С представлял собой язык программирования относительно низкого уровня, что позволяло контролировать каждую мелочь в работе алгоритма и достигать максимальной эффективности. Но, в то же время в С заложены принципы языка высокого уровня, что позволяло избежать зависимости программ от особенностей архитектуры конкретного компьютера. Это повышает эффективность процесса программирования.

Достоинства языка С:

1. В основу С положено значительно меньше синтаксических правил, чем у других языков программирования. В результате для эффективной работы компилятора достаточно всего 256 кб оперативной памяти. Список операторов и их комбинаций в языке С обширнее, чем список ключевых слов.

2. Многие функции, представленные в большинстве других языков программирования, не включены в язык С. Например, в С нет встроенных функций ввода-вывода, отсутствуют математические функции и функции работы со строками. Но если для большинства языков отсутствие таких функций было бы признаком слабости, то С взамен этого предоставляет доступ к самостоятельным библиотекам, включающим все перечисленные функции.

3. Программы написанные на С отличаются высокой эффективностью (быстро выполняются программы).

4. Упрощенный контроль за типами данных. Язык С позволяет в одном месте программы рассматривать переменную как символ, в другом месте как ASCII – код этого символа.

5. Модульное программирование.

6. Наличие указателей. Язык С предоставляет возможность обращаться к области памяти с заданным адресом, что значительно повышает скорость выполнения программы.

Язык С++ можно рассматривать как надмножество для языка С. С++ сохранил все возможности, предоставляемые языком С, дополнив их средствами объектно-ориентированного программирования.

Структура программы на языке С++:

```
#include <iostream>
using namespace std;
void main()
{
операторы
}
```

Идентификаторы

Идентификаторами называются имена, присваиваемые переменным, константам, типам данных и функциям, используемым в программе. После описания идентификатора можно ссылаться на обозначаемую им сущность в любом месте программы.

Идентификатор представляет собой последовательность символов любой длины, содержащую буквы, цифры и знаки подчеркивания, но начинающуюся обязательно с буквы или знака подчеркивания. Языки С и С++ чувствительны к регистру букв. Хотя допускается использование символа подчеркивания в начале имени идентификатора, не рекомендуется так поступать, поскольку данный способ применяется в именах встроенных системных подпрограмм и переменных. Среди программистов на С принято негласное соглашение начинать любое имя с префикса типа данных этого идентификатора.

Например, все целочисленные идентификаторы должны начинаться буквой i, идентификаторы с плавающей запятой с буквы f, указатели буквой p (pointer).

Примеры идентификаторов:

imax, imin, frange

Примеры неправильных идентификаторов:

1st_year, #a1, Not_Done!

Ключевые слова являются встроенными идентификаторами, каждому из которых соответствует определенное действие. Изменить назначение ключевого слова нельзя. Имена идентификаторов, создаваемых в программе не могут совпадать с ключевыми словами языков C/C++.

Основные операции в языке C++

\+ - * / % (остаток от деления целых чисел).
Например, a=b+c; x=y-z; c=a*b; d=r/t;
a = b%c; 12%4 (результат равен 0), 25%3 (результат равен 1).
x++ (операция с постусловием) или
++x (операция с предусловием) – аналогично x=x+1;
x-- (операция с постусловием) или
--x (операция с предусловием) – аналогично x=x-1;
Например,
int i=3, j, k=0;
k=++i; (k=4, i=4)
k=i++; (k=3, i=4)
k=i--; (k=3, i=2)
k=--i; (i=2, k=2)
i=j=k-- (k=-1, i=0, j=0)

Дополнительные операции в языке C++

+= -= *= /= %=.

Например,
a+=b; аналогично a=a+b;
a-=b; аналогично a=a-b;
a*=b; аналогично a=a*b;

a/=b; аналогично a=a/b;

a%=b; аналогично a=a%b;

v=8*(v2=5); (результат равен 40).

Допускается одновременное присваивание нескольким переменным одинакового значения, например, v1=v2=v3=0;

Основные типы данных

Целые int (4 б, диапазон значений от –32768 до 32767), short (2 б.), long (4 б.), unsigned int (unsigned), unsigned short, unsigned long;

символьные – тип char – 1 символ на 1 байт;

вещественные – тип float – числа с плавающей точкой как в фиксированном, так и в экспоненциальном формате, диапазон значений от $-3{,}4*10^{38}$ до $3{,}4*10^{38}$, размерность 4 б;

числа с плавающей точкой двойной точности – тип doudle, диапазон значений от $-1{,}7*10^{308}$ до $1{,}7*10^{308}$ байт;

void – пустые значения, применяются в функциях, не возвращающих никакого значения;

bool – логические значения, принимают 2 значения true, false;

перечисления представляются конечным набором именованных констант различных типов;

указатели в отличие от переменных других типов не содержат данных, а содержат адреса памяти, в которых хранятся данные.

Символы

В каждом языке используется определенный набор символов для построения значимых выражений. Выражения на языках С, С++ записываются с помощью 26 букв латинского алфавита, 26 прописных букв латинского алфавита, десяти цифр 0123456789 и специальных символов + - * / = , . _ ; : ? \ “ ’ - ! | # $ % & () [] {} ^ @ .

К специальным символам относится также пробел. Комбинации некоторых символов не разделенных пробелом, интерпретируются как один значимый символ. Например, ++ -- == && << >> <= >= += *= /=.

Константы

Иногда требуется, чтобы значение переменной оставалось постоянным в течение всего времени работы программы. Такие переменные называются константными. Например, если в программе вычисляется длина окружности или площадь круга, то часто придется использовать число π.

Например,

```
const float pi=3.14159;
int i=1, a=100;
double d=0.
```

В языке Си и С++ имеется встроенная операция sizeof, которая позволяет определить размер объектов в байтах.

```
cout<<"Данные типа int занимают "<<sizeof(int)<<"
байт(а)"<<endl;
```

Результат работы программы следующий:

данные типа int занимают 4 байт(а).

Аналогично вычисляется размер в байтах остальных типов данных.

В языке С++ имеет значение регистр. В выражениях следует избегать смешения типов.

Приоритет операций:

Операции	Порядок вычисления
()	Слева направо
-(унарный)	Слева направо
*/	Слева направо
+-	Слева направо
=	Слева направо

Выражения

Примеры выражений:

```
4
-5
a*(b+c/d)/20
```

x=++q%3
q>3

Выражения могут быть константами, переменными или их сочетаниями. Каждое выражение имеет значение.

Выр-е	Знач-е
C=3+8	11
5>3	1
6+(c=3+8)	17

Приоритеты операций отношения

Группа операций более высокого приоритета:

< <= > >=

Группа операций более низкого приоритета:

== !=

Подобно большинству операций, операции отношения выполняются слева направо. Например, под записью cx!=w==z подразумевается (cx!=w)==z. Логические операции имеют меньший приоритет, чем операции отношения. Например, 5>2&&4>7 ложно, поскольку истинно только одно выражение. 5>2||4>7 истинно, поскольку одно из подвыражений истинно. !(4>7) истинно, поскольку 4 не больше 7. Оно эквивалентно выражению 4<=7.

Выражение 1 && выражение 2 истинно тогда и только тогда, когда оба выражения истинны. Выражение 1|| выражение 2 истинно, если какое-нибудь одно или оба выражения истинны. ! выражение истинно, если выражение ложно и наоборот.

Приоритет логических операций

Операция ! имеет очень высокий приоритет, только круглые скобки имеют высший приоритет. Приоритет у операции &&

выше, чем у операции ||, а обе они имеют более низкий приоритет, чем операции отношения, но более высокий, чем операции присваивания.

Выражение

```
a>b&&b>c||b>d
```

интерпретируется как

```
((a>b)&&(b>c))||(b>d)
```

Директива #define

В C и C++ существует другой способ описания констант: директива препроцессора #define. Предположим в начале программы введена такая строка:

```
#define A  10
```

Встретив такую команду, препроцессор осуществит глобальную замену в программном коде имени A числом 10. Причем никакие другие значения идентификатору A не могут быть присвоены, поскольку переменная с таким именем в программе формально не описана.

Логические операторы

&& (и), || (или), ! (не)

Операторы сравнения

== (равняется) , != (не равно), > , <, >=, <=.

Оператор ввода

cin>>a>>b; (ввести с экрана 2 переменные a,b).

```
cin>>a;
cin>>b;
```

Оператор вывода

```
cout<< " Вывод данных  " <<a<<b<<endl;
float x,y;
    cin>> x;
    cin>> y;
cout.width(10) << cout.setf(ios::fixed) << cout.precision(2);
cout << x << endl;
cout.width(7);cout.precision(3);
cout << y << endl;
```

Стандартные математические функции языка С++

int abs(int x); – возвращает целое абсолютное значение.

double fabs(double x); – возвращает дробное абсолютное значение.

double sin(double x);

double cos(double x);

double tan(double x);

возвращает синус, косинус или тангенс угла, величина угла в радианах.

asin(x) арксинус x в диапазоне от [-pi/2; pi/2], x в диапазоне от [-1,1].

acos(x) арккосинус x в диапазоне от [0; pi], x в диапазоне от [-1,1].

atan(x) арктангенс x в диапазоне от [-pi/2; pi/2].

sinh(x) – гиперболический синус x.

cosh(x) – гиперболический косинус x.

tanh(x) – гиперболический тангенс x.

double exp(double x);

double log(double x); натуральный логарифм ln(x).

double log10(double x); десятичный логарифм lg(x).

double pow(double x, double y); возводит x в степень y, ошибка, если x=0 и y<=0, или x<0 и y не целое.

double sqrt(double x); извлекает корень квадратный из x.

ceil(x) – наименьшее целое в виде double, которое не меньше x.

floor(x) – наибольшее целое в виде double, которое не больше x.

Для математических функций заголовочный файл <math.h>.

Преобразование типов

Типы данных в порядке уменьшения приоритета:

double
float
long
int
short

Рассмотрим преобразование типов из одного в другой на примере.

```
int ia,ib,ic;
float fa,fb;
double da,db;
fa=(float)ia/float(ib);
da=double(fa)/5.;
ic=int(fa)+int(fb);
```

Случайные числа

Для получения случайного числа используется библиотека <stdlib.h> и 2 функции.

rand
Синтаксис
int rand(void);

Возвращает случайное целое число в диапазоне от 0 до RAND_MAX. Перед первым обращением к функции rand необходимо инициализировать генератор случайных чисел. Для этого нужно вызвать функцию srand.

srand
void srand(unsigned s);

Инициализирует генератор случайных чисел. Обычно в качестве параметра функции используют переменную, значение которой предсказать нельзя, например, это может быть текущее время.

```
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <time.h>
void main()
{int a;
 time_t t;    //текущее время для инициализации генератора случайных чисел.
srand((unsigned)time(&t)); //инициализация генератора случайных чисел.
    a=rand()%10;  //случайные числа от 0 до 9.
    cout<<a<<endl;
```

Пример 1. Вычислить среднее арифметическое 3-х чисел.

```
#include <iostream>
using namespace std;
void main()
{
    setlocale(LC_ALL, "Russian");
            float a,b,c,d;
    cin>>a>>b>>c;
    d=(a+b+c)/(float)3;
    cout<<"среднее арифметическое трех чисел="<<d<<endl;
    system("pause");
}
```

Пример 2. Написать программу, которая выводит значения целых переменных a, b в разных строках.

```
#include <iostream>
using namespace std;
void main()
```

```
{
    int a,b;
cin>>a>>b;
cout<<a<<endl<<b<<endl;
system("pause");
}
```

Пример 3. Написать программу вычисления объема конуса.
Рекомендуемый вид экрана:
Вычисление объема конуса
Введите исходные данные:
Радиус (см)-> 9
Высота (см) -> 5
Объем 424.115 куб.см.

```
#include <iostream>
#define PI 3.14159
using namespace std;
void main()
{
    setlocale(LC_ALL, "Russian");
    double r,h,v;
    cout<<"Вычисление объема конуса"<<endl;
    cout<<"Введите исходные данные:"<<endl;
    cout<<"Радиус(см)->";
    cin>>r;
    cout<<"высота(см)->";
    cin>>h;
    v=1./3.*PI*r*r*h;
    cout<<"Объем "<<v<<"куб.см"<<endl;
    system("pause").
}
```

Задания

1. Сложить 2 целых числа, результат выдать в виде:
сумма двух целых чисел =
2. Вычислить среднее арифметическое переменных x1 и x2.
3. Вычислить значение функции y = $-2,7\,x^3+0,23x^2-1,4$.

4. Вычислить площадь прямоугольника S = 0,5ah, a – основание, h – высота.

5. Вычислить площадь круга $S = \pi r^2$.

6. Написать программу, которая выводит на экран Ваше имя и фамилию.

7. Записать инструкцию, которая выводит в одной строке значения переменных a, b, c одного типа.

8. Написать программу, которая выводит значения целых переменных a, b, c в разных строках.

9. Написать программу, которая выводит на экран значения вещественных переменных в следующем виде:

a = значение b = значение c = значение.

10. Вычислить объем конуса по формуле $v=\frac{1}{3} * \pi r^2 h$.

11. Написать программу вычисления объема параллелепипеда. Рекомендуемый вид экрана:

Вычисление объема параллелепипеда

Введите исходные данные:

Длина (см)-> 9

Ширина (см) -> 7.5

Высота (см) -> 5

Объем 337.5 куб.см.

12. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей. Рекомендуемый вид экрана во время выполнения программы.

Вычисление стоимости покупки.

Введите исходные данные:

Цена тетради (тенге) -> 50

Количество тетрадей -> 5

Цена карандаша (тенге) -> 20

Количество карандашей -> 10

Стоимость покупки: 450 тенге.

13. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и такого же количества обложек к ним. Рекомендуемый вид экрана:

Вычисление стоимости покупки.

Введите исходные данные:

Цена тетради (тенге) ->50

Цена обложки (тенге) ->10

Количество комплектов (шт.) ->10

Стоимость покупки 600 тенге.

14. Написать программу вычисления площади треугольника, если известны длины двух его сторон и величины угла между этими сторонами. Рекомендуемый вид экрана во время выполнения программы:

Вычисление площади треугольника.

Введите длины двух сторон треугольника

➔ 25 17

Введите величину угла между сторонами треугольника

➔ 30

Площадь треугольника 105.25 кв.м.

15. Перевести секунды в минуты и секунды.

16. Написать программу пересчета веса из фунтов в килограммы (1 фунт – это 405, 9 грамма). Рекомендуемый вид экрана пересчет веса из фунтов в килограммы.

Введите вес в фунтах-> 5

5 фунтов – это 2.05 кг

17. Напишите программу пересчета величины временного интервала, заданного в минутах, в величину, выраженную в часах и минутах.

Рекомендуемый вид экрана во время выполнения приложения.

Введите временной интервал в минутах ->150

150 минут – это 2 ч. 30 мин.

Тема № 2

УСЛОВНЫЙ ОПЕРАТОР, ОПЕРАТОР ВЫБОРА ВАРИАНТОВ

Условный оператор

a) Инструкция if

```
if (условие)
{
инструкции
}
```

b) инструкция if – else

```
if (условие)
{
инструкции
}
else
{
инструкции
}
```

c) вложенный оператор if – else

```
if (условие 1  )
if (условие 2)
выражение 2;
else
выражение 1;
```

d) конструкция if – else – if

```
if (условие 1  )
выражение 1;
else if (условие 2)
выражение 2;
else if (условие 3)
выражение 3;
```

условный оператор можно записать в следующем виде:
условие? Выражение 1: Выражение 2;
y=(x<0)? -x:x;   (есди x<0, то y=-x, иначе y=x).

Множественный выбор

```
switch(целое выражение)
{
case константа 1:операторы; break;
case константа 2: операторы; break;
…
default: операторы;
}
```

Пример 1. Написать программу вычисления площади кольца. Программа должна проверять правильность исходных данных. Рекомендуемый вид экрана во время выполнения программы.

Вычисление площади кольца.
Введите исходные данные.
Радиус кольца (см) -> 3.5
Радиус отверстия (см) ->7
Ошибка! Радиус отверстия не может быть больше радиуса кольца.

```
#include <iostream>
#define PI 3.14159
using namespace std;
void main()
{
    setlocale(LC_ALL, "Russian");
    double r1,r2,s;
    cout<<"Вычисление площади кольца"<<endl;
    cout<<"Введите исходные данные:"<<endl;
    cout<<"Радиус кольца(см)->";
    cin>>r1;
    cout<<"радиус отверстия(см)->";
    cin>>r2;
```

```
        if(r1<r2)
        cout<<"Ошибка! Радиус отверстия не может быть
больше радиуса кольца."<<endl;
        else
        {
        s=PI*(r1*r1-r2*r2);
        cout<<s<<endl;
        }
        system("pause");
    }
```

Пример 2. Вычислить частное от деления двух чисел (с проверкой деления на 0).

```
#include <iostream>
using namespace std;
void main()
{
setlocale(LC_ALL, "Russian");
double a,b,c;
cin>>a>>b;
if(b!=0)
{
c=a/b;
cout<<c<<endl;
}
else
cout<<"деление на 0"<<endl;
system("pause");
}
```

Пример 3. Вычислить с использованием оператора switch.

$$y = \begin{cases} \lg(x) & \text{при } x = 1 \\ tg(x) & \text{при } x = 2 \\ x & \text{при } x = 3 \\ \ln(x) & \text{при } x = 4 \\ e^x & \text{при } x = 5 \end{cases}$$

```
#include <iostream>
using namespace std;
#include <math.h>
using namespace std;
void main()
{
    int x;
    double y;
    cin>>x;
    switch(x)
    {
    case 1: y=log10((double)x);break;
    case 2: y=tan((double)x);break;
    case 3: y=x; break;
    case 4: y=log((double)x); break;
    case 5: y=exp((double)x); break;
    default: y=0;
    }
    cout<<y<<endl;
    system("pause");
}
```

Задания

1. Вычислить частное от деления двух чисел (с проверкой деления на 0).

2. Вычислить

$$Y = \begin{cases} 1 & \text{при } x > 0 \\ 0 & \text{при } x = 0 \\ -1 & \text{при } x < 0 \end{cases}$$

3. Написать программу, которая проверяет, является ли введенное с клавиатуры число четным.

4. Написать программу вычисления стоимости разговора по телефону с учетом 20% скидки, предоставляемой по субботам и воскресеньям. Результат выдать в следующем виде:

Введите исходные данные:

длительность разговора -3

Тариф 30

День недели (1 – понедельник,….., 7- воскресенье) 6

Стоимость разговора: 90.

5. Написать программу, которая проверяет, является ли год високосным.

Рекомендуемый вид экрана:

Введите год 2001

2001 год – не високосный.

6. Написать программу, которая переводит время из минут и секунд в секунды. Программа должна проверять правильность введенных пользователем данных и в случае, если данные неверные, выводить соответствующее сообщение.

Рекомендуемый вид экрана во время работы программы.

Введите время (минут, секунд)->2.9

Ошибка: Количество секунд не может быть больше 60.

Для завершения нажмите <Enter>.

7. Написать программу вычисления площади кольца. Программа должна проверять правильность исходных данных. Рекомендуемый вид экрана во время выполнения программы.

Вычисление площади кольца.

Введите исходные данные.

Радиус кольца (см) -> 3.5

Радиус отверстия (см) ->7

Ошибка! Радиус отверстия не может быть больше радиуса кольца.

8. Написать программу вычисления сопротивления электрической цепи, состоящей из двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. Рекомендуемый вид экрана.

Вычисление сопротивления электрической цепи.

Введите исходные данные:

Величина первого сопротивления (см) ->15

Величина второго сопротивления (см) ->27.3

Тип соединения (1-последовательное, 2-параллельное)->2

Сопротивление цепи: 9.68 см.

9. Написать программу решения квадратного уравнения. Программа должна проверять правильность исходных данных и

в случае, если коэффициент при 2-й степени неизвестного равен 0, выводить соответствующее сообщение. Рекомендуемый вид экрана.

Введите в одной строке значения коэффициентов и нажмите <Enter>.

➔ 12 27 -10

Корни уравнения:

X1=-25.551

X2=-28.449.

5. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 10% предоставляется, если сумма покупки больше 1000 тенге. Рекомендуемый вид экрана во время выполнения программы.

Вычисление стоимости покупки с учетом скидки.

Введите сумму покупки и нажмите <Enter>

->1200

Вам предоставляется скидка 10%

Сумма покупки с учетом скидки: 1080.00 тенге.

6. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется, если сумма покупки больше 500 тенге, в 5% – если сумма больше 100 тенге. Рекомендуемый вид экрана во время работы приложения.

Вычисление стоимости покупки с учетом скидки.

Введите сумму покупки и нажмите <Enter>

->640

Вам предоставляется скидка 3%

Сумма покупки с учетом скидки: 620.80 тенге

7. Напишите программу проверки знания истории архитектуры. Программа должна вывести вопрос и 3 варианта ответа. Пользователь должен выбрать правильный ответ и ввести его номер. Рекомендуемый вид экрана во время выполнения программы.

Архитектор Исаакиевского собора:

1. Доменико Трезини

2. Огюст Монферран

3. Карл Росси

Введите номер правильного ответа и нажмите <Enter>

➔ 3

Вы ошиблись.

Архитектор Исаакиевского собора – Огюст Монферран.

8. Написать программу, которая сравнивает два введенных с клавиатуры числа. Программа должна указать какое число больше или, если числа равны, вывести соответствующее сообщение. Рекомендуемый вид экрана во время работы программы.

Введите в одной строке 2 целых числа и нажмите <Enter>

➔ 34 67

34<67

14. Вычислить с использованием оператора switch.

$$Y = \begin{cases} \sin(x) & \text{при } x = 1 \\ \cos(x) & \text{при } x = 2 \\ x^2 & \text{при } x = 3 \\ \ln(x) & \text{при } x = 4 \\ e^x & \text{при } x = 5 \end{cases}$$

15. Написать программу, которая запрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке, если введены неверные данные.

16. Написать программу, которая вычисляет стоимость междугородного телефонного разговора (цена одной минуты определяется расстоянием до города, в котором находится абонент). Исходными данными для программы являются код города и длительность разговора. Рекомендуемый вид экрана:

Город	Код	Цена минуты (минуты)
Владивосток	423	70
Москва	095	60
Мурманск	815	80
Самара	646	60

Вычисление стоимости разговора по телефону.

Введите исходные данные:

Код города ->423

Длительность (целое количество минут)->3

Город: Владивосток

Цена минуты: 70 тенге

Стоимость разговора:

Тема № 3

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА ЦИКЛА FOR

Оператор цикла for

for (инициализирующее выражение; условное выражение; модифицирующее выражение)
{Операторы;}

Первым выполняется инициализирующее выражение, в котором устанавливается счетчик цикла. Это происходит 1 раз перед запуском цикла. Затем анализируется условное выражение, которое также называется условием прекращения цикла. Пока оно выполняется, цикл не прекращается. Каждый раз после всех строк цикла выполняется модифицирующее выражение, происходит изменение счетчика цикла.

Например, нужно вычислить сумму целых чисел от 1 до 5.

```
int is,iv;
is=0;
for(iv=1;iv<=5;iv++)
is+=iv;
```

Существует много возможностей применения оператора цикла for.

Можно применять операцию уменьшения для счета в порядке убывания вместо счета в порядке возрастания.

1. for(n=10;n>0;n--)
cout<< n<<endl;

2. При желании можно счет вести двойками, десятками, и т.д.
for(n=2;n<60;n+=13)
cout<<n<<endl;

Будут напечатаны значения 2,15,28,41,54.

3. Можно вести подсчет с помощью символов, а не только цифр.

```
for(ch='a';ch<='z';ch++)
cout<<ch<<" ";
```

4. Можно проверить выполнение некоторого произвольного условия, отличного от условия, налагаемого на число итераций. В программе вычисления таблицы кубов спецификацию

```
for(num=1;num<=6;num++)
```

можно было бы заменить на

```
for(num=1;num*num*num<=216;num++)
```

Это было бы целесообразно в случае, если бы нас больше интересовало ограничение максимального значения диапазона кубов чисел, а не количества итераций.

5. Можно сделать так, чтобы значение некоторой величины возрастало в геометрической, а не в арифметической прогрессии.

```
for(d=100.0; d<150.0;d*=1.1)
cout<<d<<endl;
```

Результат будет

```
100.00
110.00
121.00
133.00
146.41
```

6. В качестве третьего выражения можно использовать любое правильно составленное выражение.

```
for(x=1;y<=75;y=5*x++ +10)
cout<<x<<"  "<<y<<endl;
```

```
1 15
2 20
3 25
4 30
5 35
6 40
7 45
8 50
9 55
10 60
```

11 65
12 70
13 75

Это не является хорошим стилем программирования, поскольку происходит смешивание процесса изменения переменной цикла с алгоритмическими вычислениями.

7. Можно опустить одно или более выражений, но при этом нельзя опустить символы;

```
ans=2;
for(n=3;ans<=25;)
ans*=n;
```

При выполнении этого цикла величина n останется равной 3. Значение переменной ans будет равно 6, 18, 54.

Тело цикла

```
for(;;)
cout<<”Hello!”<<endl;
```

будет выполняться бесконечное число раз, поскольку пустое условие всегда считается истинным.

8. Первое выражение не обязательно должно инициализировать переменную. Первое выражение выполняется 1 раз.

```
for(cout<<”Запоминайте введенные числа!”<<endl;num==6;)
cin>>num;
```

В этом фрагменте первое сообщение оказывается выведенным на печать 1 раз, а затем осуществляется прием вводимых чисел до тех пор, пока не поступит число 6.

9. Операция запятая позволяет включать в спецификацию оператора for несколько инициализирующих или корректирующих выражений.

```
for(o=1,c=10;o<=16;o++,c+=17)
cout<<o<<” “<<c<<endl;
```

Пример 1. Вычислить сумму ряда

1+1/2+1/4+1/8+1/16+…….

```
#include <iostream>
using namespace std;
void main()
{
int count;
const int limit=16;
```

```
float sum,x;
for(sum=0.0,x=1.0,count=1;count<=limit;count++,x*=2.0)
{sum+=1.0/x;
cout<<sum<<"   "<<count<<endl;
}
system("pause");
}
```

Пример 2. Вычислить сумму S=1+1/2+1/3+…..+1/n

```
#include <iostream>
using namespace std;
void main()
{
    int i,n;
    double s=0;
    cin>>n;
    for(i=1;i<=n;i++)
            s+=1./(double)i;
    cout<<s<<endl;
    system("pause");
}
```

Пример 3. Написать программу, которая вводит с клавиатуры дробные числа и вычисляет их среднее арифметическое. Количество чисел должно задаваться во время работы программы.

```
# include <iostream>
using namespace std;
void main()
{
    int i,n; double a,s=0,sr;
    cin>>n;
    for(i=1;i<=n;i++)
{cin>>a;
    s+=a;
    }
    sr= s/(double)n;
```

```
    cout<<sr<<endl;
    system("pause");
}
```

Задания

1. Написать программу, которая выводит таблицу первых десяти целых положительных чисел. Результат выдать в следующем виде.

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

2. Написать программу, которая вычисляет факториал введенного с клавиатуры числа.

3. Написать программу, которая выводит таблицу квадратов первых пяти целых положительных нечетных чисел. Рекомендуемый вид экрана во время работы программы.

1	1
3	9
5	25
7	49
9	81

4. Написать программу, которая вычисляет сумму первых n целых положительных чисел. Количество суммируемых членов должно вводиться во время работы программы. Рекомендуемый вид экрана:

Вычисление суммы положительных чисел.

Введите количество суммируемых чисел- >20

Сумма первых 20 положительных чисел равна 210.

5. Написать программу, которая вычисляет сумму первых n целых положительных четных чисел. Количество суммируемых членов должно вводиться во время работы программы. Рекомендуемый вид экрана:

Вычисление суммы четных положительных чисел.

Введите количество суммируемых чисел- >12

Сумма первых 12 положительных четных чисел равна 156.

6. Написать программу, которая вычисляет сумму первых n целых положительных нечетных чисел. Количество суммируемых членов должно вводиться во время работы программы. Рекомендуемый вид экрана:

Вычисление суммы нечетных положительных чисел.

Введите количество суммируемых чисел- >15

Сумма первых 15 положительных нечетных чисел равна 330.

7. Вычислить сумму S=1+1/2+1/3+…..+1/20.

Вычисление частичной суммы ряда 1+1/2+1/3+1/4+…..

Введите кол-во суммируемых членов ряда ->15

Сумма первых 15 членов ряда равна 3.3182

8. Написать программу, которая выводит таблицу степеней двойки от нулевой до десятой.

Рекомендуемый вид экрана.

Таблица степеней двойки

0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

---

9. Написать программу, которая вводит с клавиатуры дробные числа и вычисляет их среднее арифметическое. Количество чисел должно задаваться во время работы программы. Рекомендуемый вид экрана.

Вычисление среднего арифметического последовательности дробных чисел. После ввода каждого числа нажмите <Enter>

->5.4

->7.8

->3.0

->1.5

->2.3

Среднее арифметическое введенной последовательности: 4.00

Для завершения нажмите <Enter>.

10. Вычислить $y = \sin x + \sin^2 x + ... + \sin^{10} x$

11. Вычислить y = 1! + 2! + … + n!

12. Написать программу, которая находит минимальный и максимальный элемент последовательности. Количество чисел последовательности должно задаваться во время работы программы. Рекомендуемый вид экрана.

Введите количество чисел последовательности ->5

Вводите последовательность. После ввода каждого числа нажимайте <Enter>

->5.4

->7.8

->3.0

->1.5

->2.3

Минимальное число 1.5

Максимальное число 7.8

Для завершения нажмите <Enter>

13. Написать программу, которая генерирует последовательность из 10 случайных чисел в диапазоне от 1 до 10, выводит эти числа на экран и вычисляет их среднее арифметическое. Рекомендуемый вид экрана во время выполнения программы.

*** Случайные числа***

1 3 4 2 7 4 9 6 2 1 Среднее арифметическое 3.5

14. Написать программу, которая выводит на экран таблицу умножения, например на 7. Рекомендуемый вид экрана во время выполнения программы следующий.

7x2=14

7x3=21
7x4=28
7x5=35
7x6=42
7x7=49
7x8=56
7x9=63

15. Написать программу, которая выводит на экран квадрат Пифагора – таблицу умножения. Рекомендуемый вид экрана во время выполнения работы программы.

```
    1  2  3  4  5  6  7  8  9  10
1   1  2  3  4  5  6  7  8  9  10
2   2  4  6  8  10 12 14 16 18 20
3   3  6  9  12 15 18 21 24 27 30
……..
9   9  18 27 36 45 54 63 72 81 90
```

16. Написать программу, которая преобразует введенное пользователем десятичное число в двоичное. Рекомендуемый вид экрана во время выполнения программы.

Введите целое число от 0 до 255 и нажмите <Enter>
->5
Десятичному числу 5 соответствует 101
Для завершения нажмите <Enter>.

Тема № 4

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА ЦИКЛА WHILE

Оператор цикла while

```
while(условие)
выражение;
```

Если тело цикла состоит из нескольких строк, необходимо использовать фигурные скобки.

```
while (условие)
{
выражение 1;
выражение 2;
…….
выражение n;
}
```

Циклы for и while являются циклами с предусловием. Проверка истинности условия осуществляется перед началом каждой итерации цикла. Цикл do while – с постусловием. Какой оператор цикла лучше применять? Исходя из соображений стиля программирования применение цикла for представляется более предпочтительным в случае, когда в цикле используется инициализация и коррекция переменной, а применение цикла while, когда этого нет. Применение цикла for представляется более естественным в случаях, когда осуществляется счет прохождений с обновлением индекса.

```
for(count=1;count<=100;count++)
```

Пример 1.

```
// Определение минимального числа
// последовательности положительных чисел
```

```
#include <iostream>
using namespace std;
void main()
{
setlocale(LC_ALL, "Russian");
int a;   // очередное число
int min;  // минимальное число
cout<<"Определение минимального числа"<<endl;
cout<<"в последовательности положительных чисел"<<endl;
cout<<"Вводите числа. Для завершения введите ноль"<<endl;
cout<<"-> ";
cin>>a;
min = a;    // пусть первое число минимальное
while ( a > 0)
{
if (a < min) min = a;
cout<<"-> ";
cin>>a; }
cout<<"Минимальное число последовательности: ";
cout<<min<<endl;
system("pause");
}
```

Пример 2.

Написать программу, которая выводит таблицу значений функции $y=-8,3x^2+5x-4$ в диапазоне от -2 до -1 с шагом 0,1.

```
#include <iostream>
using namespace std;
void main()
{
    double x=-2,y,dx=0.1;
    while(x<=-1)
    {
            y=-8.3*x*x+5*x-4;
            cout<<x<<"     "<<y<<endl;
            x+=dx;
    }
    system("pause");
}
```

Пример 3.

Написать программу, которая вычисляет число π с заданной точностью. Для вычисления значения числа π воспользоваться формулой:

π= 4(1-1/3+1/5-1/7+1/9-……..)

```
#include <iostream>
using namespace std;
void main()
{
    setlocale(LC_ALL, "Russian");
    double eps=0.00001,e=1,pi,s=0;
    int n=1;
    while(e>=eps)
    {       e=1./(2*n-1);
    if(n%2==0)
    s-=e;
    else
            s+=e;
    n++;
    }
    pi=4*s;
    cout<<"просуммировано "<<n<<"членов ряда"<<endl;
    cout<<pi<<endl;
    system("pause");
}
```

Задания

1. Написать программу, которая выводит таблицу значений функции

y = -2,4x^2+5x-3 в диапазоне от -2 до 2 с шагом 0,5.

2. Написать программу, которая вычисляет число π с заданной точностью. Для вычисления значения числа π воспользоваться формулой:

π = 4(1-1/3+1/5-1/7+1/9-……..)

Подсчитать количество суммируемых членов.

3. Вычислить с точностью Eps>0

$$y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$$

Считать, что требуемая точность достигнута, если среднее слагаемое по модулю меньше Eps, все последующие можно не учитывать.

4. Вычислить $y = \sin 1 + \sin 1.1 + \ldots + \sin 2$

5. Написать программу, которая вычисляет наибольший общий делитель двух целых чисел.

6. Написать программу, которая выводит на экран таблицу стоимости, например яблок в диапазоне от 100 г. до 1 кг. с шагом 100 г.

7. Написать программу, которая выводит таблицу значений функции y=|x|. Диапазон изменения аргумента от -4 до 4, шаг приращения аргумента 0,5.

8. Написать программу, которая выводит таблицу значений функции y=|x-2|+|x+1|. Диапазон изменения аргумента от -4 до 4, шаг приращения аргумента 0,5.

Тема № 5

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА ЦИКЛА DO WHILE

Оператор цикла do while

```
do
выражение;
while(условие);
```

Если тело цикла состоит более чем из одной строки, необходимо ставить фигурные скобки.

```
do
{
выражение 1;
выражение 2;
…….
выражение n;
}
while(условие);
```

Инструкции перехода

В языках C/C++ имеется четыре инструкции перехода: goto, break, continue и return. Инструкция break позволяет выходить из цикла еще до того, как условие цикла станет ложным. По своему действию она напоминает команду goto, только в данном случае не указывается точный адрес перехода: управление передается первой строке, следующей за телом цикла. Рассмотрим пример:

```
/*
*   break. c

В этой программе на языке С демонстрируется использо-
вание инструкции break. */
```

```
int main ()
{
 int itimes = 1,
isum = 0;
 while (itimes < 10)
 { isum += itimes;
if (isum > 20) break;
itimes++;
return (0);
}
```

Проанализируйте работу этой программы с помощью отладчика, контролируя значения переменных isum и itimes. Обратите внимание на то, что происходит, когда переменная isum достигает значения 21. Вы увидите, что в этом случае выполняется инструкция break, в результате чего цикл прекращается и выполняется первая строка, следующая за телом цикла. В нашем примере это инструкция return, которая завершает программу. Оператор continue, как и оператор break, используется только внутри операторов цикла, но в отличие от него выполнение программы продолжается не с оператора, следующего за прерванным оператором, а с начала прерванного оператора.

Формат оператора следующий:

```
continue;
```

Пример:

```
int main()
{ int a,b;
for (a=1, b=0; a<100; b+=a, a++)
{ if (b%2) continue;
... /* обработка четных сумм */
}
return 0;
}
```

Когда сумма чисел от 1 до a становится нечетной, оператор continue передает управление на очередную итерацию цикла for, не выполняя операторы обработки четных сумм. Оператор conti-

nue, как и оператор break, прерывает самый внутренний из объемлющих его циклов.

Любой оператор в составном операторе может иметь свою метку, состоящую из имени и следующего за ним двоеточия. Оператор goto передает управление на оператор, помеченный меткой имя-метки. Формат этого оператора следующий:

```
goto имя-метки;
...
имя-метки: оператор;
```

Помеченный оператор должен находиться в той же функции, что и оператор goto, а используемая метка должна быть уникальной, т.е. одно имя-метки не может быть использовано для разных операторов программы. Имя-метки – это идентификатор. Используя оператор goto, можно передавать управление и внутрь составного оператора. Но нужно быть осторожным при входе в составной оператор, содержащий объявления переменных с инициализацией, так как объявления располагаются перед выполняемыми операторами и значения объявленных переменных при таком переходе будут не определены. Использование оператора безусловного перехода goto в практике программирования на языке C++ настоятельно не рекомендуется, так как он затрудняет понимание программ и возможность их модификаций.

Оператор return завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом. Функция main передает управление операционной системе. Формат оператора:

```
return [выражение];
```

Значение выражения, если оно задано, возвращается в вызывающую функцию в качестве значения вызываемой функции. Если выражение опущено, то возвращаемое значение не определено. Выражение может быть заключено в круглые скобки, хотя их наличие не обязательно.

Пример 1. Вычислить среднее арифметическое последовательности положительных чисел.

```
#include <iostream>
using namespace std;
void main()
{
setlocale(LC_ALL, "Russian");
int a;     // число, введенное с клавиатуры
int n;     // количество чисел
int s;     // сумма чисел
float m;// среднее арифметическое
s = 0;
n = 0;
cout<<"Вычисление среднего арифметического"<<endl;
cout<<"последовательности положительных чисел"<<endl;
cout<<"Вводите числа. Для завершения введите ноль"<<endl;
do {
cout<<"-> ";
cin>>a;
if (a > 0)
{
s+=a;
n++; }

} while (a > 0);
cout<<"Введено чисел: "<<n<<endl;
cout<<"Сумма чисел: "<<s<<endl;
m =  (float) s / n;
cout<<"Среднее арифметическое:"<< m<<endl;
system("pause");
}
```

Задания

1. Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Рекомендуемый вид экрана во время выполнения программы.

Вычисление среднего арифметического последовательности положительных чисел.

Вводите после стрелки числа. Для завершения ввода нажмите 0.

->45
->23
->15
->0

Введено чисел: 3

Сумма чисел: 83

Среднее арифметическое: 27.67

2. Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Рекомендуемый вид экрана во время выполнения программы.

Определение максимального числа последовательности положительных чисел.

Введите после строки числа. Для завершения ввода нажмите 0.

->56
->75
->43
->0

Максимальное число 75.

3. Написать программу, которая определяет минимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Рекомендуемый вид экрана во время выполнения программы.

Определение минимального числа последовательности положительных чисел.

Введите после строки числа. Для завершения ввода нажмите 0.

->12
->75
->10
->0

Максимальное число 10.

4. Написать программу, которая проверяет является ли введенное пользователем число простым. Рекомендуемый вид экрана во время выполнения программы.

Введите целое число и нажмите <Enter>.

->45

45 – не простое число,

5. Написать программу, которая «Задумывает» число в диапазоне от 1 до 10 и предлагает пользователю угадать число за 5 попыток. Рекомендуемый вид экрана во время выполнения программы.

Игра «Угадай число».

Компьютер задумал число от 1 до 10.

Угадайте его за 5 попыток.

Введите число и нажмите <Enter>

->5

Нет.

->3

Вы выиграли! Поздравляю!

Тема № 6

ОДНОМЕРНЫЕ МАССИВЫ

Массив можно представить как переменную, содержащую упорядоченный набор данных одного типа.

Свойства массивов

В массиве хранятся отдельные значения, которые называются элементами. Все элементы массива должны быть одного типа. Все элементы массива сохраняются в памяти последовательно, и первый элемент массива имеет нулевой индекс. Имя массива является константой и содержит адрес первого элемента массива.

Объявление массивов

Массивы объявляются следующим образом:

```
int a[10];     //массив из 10 целых чисел
char c[12];  //массив из 12 символов
```

Объявление массива начинается с указания типа данных, после чего следует имя массива и пара квадратных скобок. Доступ к элементу массива осуществляется по индексу массива, представляющего собой порядковый номер элемента массива. Например, a[0] – 1-й элемент массива, a[1] – 2-й элемент массива и т.д. Элементы одномерного массива можно вводить в цикле. Например,

```
int  i,a[10];
for(i=0;i<10;i++)
cin>>a[i];
```

Размер массива должен быть известен заранее и может быть изменен в ходе выполнения программы.

Установка размера массива с помощью констант:

```
#define A 20
#define H 15
int ia[A];
char fa[H];
```

Инициализация массивов

Массив можно инициализировать одним из трех способов:

– при создании массива, используя инициализацию по умолчанию;

– при создании массива, явно задавая начальные константные значения;

– в процессе выполнения программы – путем записи данных в массив.

Явная инициализация осуществляется следующим образом:

```
int ia[3]= {-1,0,2};
float f[4]={1.14, 0.75, 3.5E4, -0.33E1};
int d[3]={ 4,7,8,2,9,0};
```

Показано задание большего числа элементов, чем размерность массива. Компилятор Microsoft Visual C++ выдаст ошибку «Слишком много инициализаторов». Когда при инициализации указано меньше значений, чем элементов массива, оставшиеся элементы по умолчанию примут нулевые значения. Можно вообще не задавать размерность массива. Количество значений, указанных в фигурных скобках, автоматически определит размерность массива. Например, char c[] = {‘A’,’a’,’E’,’e’,’I’,’i’}.

Инициализация безразмерных массивов

Задать массив символов можно двумя способами.

1-й способ:

```
char E[29]=”Введите значения от 0 до 9\n”;
```

Здесь необходимо подсчитать число символов в строке, добавив к полученному числу 1 для символа \0, обозначающего конец строки. Это громоздкая работа. Можно автоматически вычислить размерность массива.

2-й способ:

char E[]=”Введите значения от 0 до 9\n”;

Компилятор автоматически создаст массив такого размера, чтобы вместить все указанные элементы.

В языке С++ имеется встроенная операция sizeof, которая позволяет определить размер объектов в байтах. Например,

```
void main()
{
cout<<”Данные типа  int занимают” << sizeof(int)<<” байт”;
}
```

Часто оператор sizeof применяют, чтобы определить размер переменной, тип которой в разных системах может иметь разную размерность. При работе с массивами, размещаемыми в памяти динамически, необходимо точно знать, сколько памяти запрашивать у операционной системы.

Пример программы, вычисляющей размер массива в байтах.

```
void main()
{
int w[7]= {1,2,3,4,5,6,7};
cout<< «Массив занимает» << sizeof(w)<<” байт”;
Пример программы, вычисляющей размер массива.
void main()
{
int d[]={31,28,31,30,31,30,31,31,30,31};
int i;
for(i=0;i<sizeof d/(sizeof(int)); i++)
cout<< i+1<<”   “<<d[i];
}
```

Выход за пределы массива

При работе с массивами не стоит забывать о том, что индекс в обращении к элементу не должен превышать размерность массива. Функции контроля подобных ошибок не включены в компилятор С, С++. При компиляции ошибки не будут выявлены. Однако при выполнении программы могут быть невольно изменены значения других переменных.

Массивы символов

Языки С, С++ поддерживают тип char. Но в переменных этого типа могут храниться только одиночные символы, но не строки текста. Если в программе необходимо работать со строками, их можно создавать как массив символов. В таком массиве для каждого символа строки отводится своя ячейка, а последний элемент содержит символ конца строки \0. Рассмотрим пример заполнения символьного массива.

```
#include <iostream>
void main()
{
char s1[7],
s2[8];              //самолет
char s3[8]=”корабль”;
s1[0]=’м’;
s1[1]=’а’;
s1[2]=’ш’;
s1[3]=’и’;
s1[4]=’н’;
s1[5]=’а’;
 s1[6]=’\0’;
cin>>s2;
cout<<s1<<”\n”;
cout<<s2<<”\n”;
cout<<s3<<”\n”;
}
```

При выполнении программы на экран будет выведено
машина
самолет
корабль

Пример 1. Написать программу, которая вычисляет среднюю (за неделю) температуру воздуха. Исходные данные должны вводиться во время работы программы. Рекомендуемый вид экрана приведен ниже (данные, введенные пользователем, выделены полужирным шрифтом).

Введите температуру воздуха за неделю.
Понедельник -> 12
Вторник -> 10
Среда -> 16
Четверг -> 18
Пятница -> 17
Суббота -> 16
Воскресенье -> 14
Средняя температура за неделю: 14.71 град.

```
#include <iostream>
using namespace std;
void main()
{
setlocale(LC_ALL, "Russian");
// названия дней недели – массив строковых констант
char *day[] =
{"Понедельник","Вторник","Среда","Четверг","Пятница",
"Суббота","Воскресенье"};
float t[7];   // температура
float sum=0;    // сумма температур за неделю
float sred;   // средняя температура за неделю
 int i;
cout<<”Введите температуру воздуха”<<endl;
for (i = 0; i <= 6; i++)
{
cin>>day[i];
cin>>t[i];
```

```
sum += t[i]; }
sred = sum / 7;
cout<<sred<<endl;
system(“pause”);
}
```

Пример 2. Написать программу, которая проверяет, находится ли введенное с клавиатуры число в массиве. Массив должен вводиться во время работы программы.

```
#include <iostream>
using namespace std;
void main()
{setlocale(LC_ALL, "Russian");
    int i,n,a[10],x;
    bool f=false;
    cin>>n;
    for(i=0;i<n;i++)
            cin>>a[i];
    cin>>x;
    for(i=0;i<n;i++)
            if(a[i]==x)f=true;
    if(f==true)
            cout<<"Число содержится в массиве"<<endl;
    else
cout<<"Число не содержится в массиве"<<endl;
    system("pause");
}
```

Пример 3. Переписать элементы одномерного массива следующим образом: сначала положительные, затем отрицательные.

```
#include <iostream>
using namespace std;
void main()
{
    int a[10],n,i,b[10],k=0;
```

```
    cin>>n;
    for(i=0;i<n;i++)
            cin>>a[i];
    for(i=0;i<n;i++)
            if(a[i]>0)
                    {b[k]=a[i];
    k++;}
    for(i=0;i<n;i++)
    if(a[i]<0)
                    {b[k]=a[i];
    k++;}
    for(i=0;i<n;i++)
            cout<<b[i]<<"  ";
    cout<<endl;
    system("pause");
}
```

Задания

1. Написать программу, которая вводит с клавиатуры одномерный массив из 5 целых чисел, после чего выводит количество ненулевых элементов. Перед вводом каждого элемента должна выводиться подсказка с номером элемента.

Ввод массива целых чисел.
После ввода каждого числа нажмите <Enter>
a[1] -> 12
a[2] -> 0
a[3] -> 3
a[4] -> -1
a[5] -> 0
В массиве 3 ненулевых элемента.

2. Написать программу, которая выводит минимальный элемент введенного с клавиатуры массива целых чисел. Ниже приведен рекомендуемый вид экрана во время работы программы.

Поиск минимального элемента массива.
Введите в одной строке элементы массива (5 целых чисел)
и нажмите <Enter>
-> 23 0 45 -5 12

Минимальный элемент массива: -5

3. Написать программу, которая вычисляет среднее арифметическое элементов массива.

4. Написать программу, которая вычисляет среднее арифметическое ненулевых элементов введенного с клавиатуры массива целых чисел. Ниже приведен рекомендуемый вид экрана во время работы программы.

Введите элементы массива (10 целых чисел) в одной строке и нажмите <Enter>.

-> 23 0 45 -5 12 0 -2 30 0 64

Сумма элементов массива: 184

Количество ненулевых элементов: 7

Среднее арифметическое ненулевых элементов: 23.86

5. Написать программу, которая вычисляет среднее арифметическое элементов массива без учета минимального и максимального элементов массива. Ниже приведен рекомендуемый вид экрана во время работы программы.

Среднее арифметическое без учета min и max значений.

Введите массив (10 целых чисел в одной строке)

->12 10 5 7 15 4 10 17 23 7

Минимальный элемент: 4

Максимальный элемент: 23

Среднее арифм. без учета min и max значений: 10.36

6. Написать программу, которая вычисляет среднюю (за неделю) температуру воздуха. Исходные данные должны вводиться во время работы программы. Рекомендуемый вид экрана приведен ниже.

Введите температуру воздуха за неделю.

Понедельник -> 12

Вторник -> 10

Среда -> 16

Четверг -> 18

Пятница -> 17

Суббота -> 16

Воскресенье -> 14

Средняя температура за неделю: 14.71 град.

7. Написать программу, которая проверяет, находится ли введенное с клавиатуры число в массиве. Массив должен вводиться во время работы программы.

8. Объединить 2 одномерных массива в один.

9. Переписать элементы одномерного массива следующим образом: сначала отрицательные, затем положительные.

10. Написать программу, которая проверяет, представляют ли элементы введенного с клавиатуры массива возрастающую последовательность.

11. Написать программу, которая вычисляет, сколько раз введенное с клавиатуры число встречается в массиве.

12. Написать программу, которая проверяет, есть ли во введенном с клавиатуры массиве элементы с одинаковым значением.

13. Написать программу, которая методом обмена ("пузырька") сортирует по убыванию введенный с клавиатуры одномерный массив.

14. Написать программу, которая определяет количество учеников в классе, чей рост превышает средний. Рекомендуемый вид экрана во время работы программы приведен ниже.

*** Анализ роста учеников ***

Введите рост (см) и нажмите <Enter>.

Для завершения введите 0 и нажмите <Enter>

-> **175**

-> **170**

-> **180**

-> **168**

-> **170**

-> 0

Средний рост: 172.6 см

У 2 человек рост превышает средний.

15. Написать программу, которая обрабатывает результаты экзамена. Для каждой оценки программа должна вычислить процент от общего количества оценок. Рекомендуемый вид экрана во время работы программы приведен ниже.

Обработка результатов экзамена.

Введите исходные данные:

пятерок -> 12

четверок -> 10
троек -> 7
двоек -> 1
Результаты экзамена
пятерок12 %
четверок 10 %
троек 7 %
двоек 1 %
Для завершения программы нажмите <Enter>.

Тема № 7

ДВУМЕРНЫЕ МАССИВЫ

Многомерные массивы

Под размерностью массива понимают число индексов, которые необходимо указать для получения доступа к отдельному элементу массива. Чтобы определить размерность массива достаточно посмотреть на его объявление. Если в нем указана только одна пара квадратных скобок, то имеем одномерный массив, если 2, то двумерный и т.д. Массивы с более чем одной размерностью называются многомерными. В реальных программах размерность массива не превышает 3-х.

Двумерный массив

Примеры объявлений:

```
int a[5][5],b[12][10];
float s[10][10];
int i,j;
for (i=0;i<5;i++)
for (j=0;j<5;j++)
cin>>a[i][j];
```

Первый индекс – индекс строки, 2-й индекс столбца. Например, a[2][3] – доступ к элементу третьей строки и четвертого столбца.

Двумерные массивы можно инициализировать так же, как и одномерные:

```
double a[3][3]= {
1.1, 2.1, 3.1,
```

```
2.2, 3.2, 4.2,
1.5, 1.6, 1.7;
}
```

Пример объявления и инициализации двумерного массива, состоящего из трех строк и пяти столбцов.

```
int aMatrix[3][5] = {{3, 5, 5, 7, 8},
                     {4, 1, 1, 2, 9},
                     {3, 8, 8, 9, 7}};
```

Двумерный массив имеет два индекса (на то он, в принципе, так и называется). Сразу при объявлении мы его инициализируем целочисленными величинами. Для удобства мы их записываем в виде матрицы (таблицы): каждая строка с новой строки (их у нас 3), в каждой строке 5 столбцов. Можно и так записать, как показано ниже. Разницы для компилятора не будет никакой. Разве лишь разница будет в визуальном восприятии для человека.

```
int aMatrix[3][5] = {{3, 5, 5, 7, 8}, {4, 1, 1, 2, 9}, {3, 8, 8, 9, 7}};
```

Либо вообще так, без указания фигурных скобок, которые логически разделяют строки друг от друга.

```
int aMatrix[3][5] = {3, 5, 5, 7, 8, 4, 1, 1, 2, 9, 3, 8, 8, 9, 7};
```

Последняя запись демонстрирует то, как на самом деле элементы массива размещаются в памяти компьютера, они идут последовательно. Если инициализация двумерного массива происходит одновременно с объявлением, то можно даже не указывать первый индекс, т.е. количество строк массива.

```
int aMatrix[][5] = {{3, 5, 5, 7, 8},
                    {4, 1, 1, 2, 9},
                    {3, 8, 8, 9, 7}};
```

Зная количество столбцов, компилятор при компиляции сам рассчитает количество строк двумерного массива. Для доступа к

элементам двумерного массива нужно, так же, как и для одномерного указать индекс. В данном случае нам нужно будет позаботиться об указании двух индексов. Например, чтобы перезаписать последний элемент второй строки, мы должны использовать такую запись

```
aMatrix[1][4] = 0;
```

В этом случае мы перезапишем значение 9 на 0.

Число байт в памяти, требуемых для размещения двумерного массива, вычисляется следующим образом:

*Число байт = размер второго измерения **
** размер первого измерения * sizeof (базовый тип).*

Предполагая наличие в системе 2-байтных целых, целочисленный массив с размерностями 10 на 5 будет занимать 10 * 5 * 2, то есть 100 байт. Для прохода по двумерному массиву удобнее всего использовать два цикла *for*, вложенных друг в друга. Ниже на примере мы выводим содержимое двумерного массива на экран.

```
//Работа с двумерным массивом
int main()
{
   int aMatrix[3][5] = {{3, 5, 5, 7, 8},
                        {4, 1, 1, 2, 9},
                        {3, 8, 8, 9, 7}};
   aMatrix[1][4] = 0;
for(int i = 0; i < 3; i++)
   {
      for(int j = 0; j < 5; j++)
      {
         cout << aMatrix[i][j] << ", ";
      }
      cout << endl;
   }
   return 0;
}
```

Пример 1. Вычислить среднее арифметическое элементов двумерного массива.

```
#include <iostream>
using namespace std;
#include <math.h>
void main()
{
int x,sum,i,j,a[2][2];
sum=0;
for (i=0;i<=1;i++)
for (j=0;j<=1;j++)
{
    cout<<"a["<<i+1<<","<<j+1<<"]=";
    cin>>a[i][j];
    sum+=a[i][j];
}
cout<<"SREDNEE "<<(float)sum/4<<endl;
system("pause");
}
```

Пример 2. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет сумму его элементов по столбцам.

```
#include <iostream>
using namespace std;
void main()
{
int i,j,a[2][2],b[2];
for(i=0;i<2;i++)
    for(j=0;j<2;j++)
            cin>>a[i][j];

for(i=0;i<2;i++)
    {b[i]=0;
    for(j=0;j<2;j++)
            b[i]+=a[j][i];}
for(i=0;i<2;i++)
    cout<<b[i]<<"  ";
```

```
cout<<endl;
system("pause");
}
```

Задания

1. Написать программу, которая складывает два двумерных массива, результат записывает в третий.

2. Объявить двухмерный массив A[5][5]. Рассчитать элементы массива по формуле:

A[i][j]=(i + 10)*j , результат выдать в виде:

A[1][1] = A[1][2] = A[1][5] =

A[2][1] = A[2][5] =

…

…

A[5][1] = A[5][5] =

3. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет среднее арифметическое его элементов.

4. Написать программу, которая вычисляет определитель квадратной матрицы второго порядка. Рекомендуемый вид экрана во время работы программы приведен ниже.

Введите матрицу второго порядка.

После ввода элементов строки нажимайте <Enter>

-> 5 -7

-> 1 3

Определитель матрицы

5.00 -7.00

1.00 3.00 равен 22.00

5. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет сумму его элементов по столбцам.

6. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет сумму его элементов по строкам.

7. Написать программу, которая определяет номер строки квадратной матрицы, сумма элементов которой максимальна.

8. Найти минимальный и максимальный элементы в двумерном массиве и их индексы.

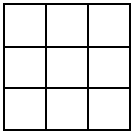
9. Переписать элементы двумерного массива в одномерный.

10. Вычислить среднее арифметическое элементов массива без учета минимального и максимального значений.

11. Выполнить перемножение двух квадратных матриц.

12. Написать программу, которая проверяет, является ли введенная с клавиатуры квадратная матрица "магическим" квадратом.

"Магическим" квадратом называется матрица, у которой сумма чисел в каждом горизонтальном ряду, в каждом вертикальном и по каждой из диагоналей одна и та же (см. приведенный ниже рисунок).



13. Вычислить сумму элементов двумерной квадратной матрицы, находящихся выше главной диагонали.

14. Вычислить сумму элементов двумерной квадратной матрицы, находящихся ниже главной диагонали.

15. Вычислить сумму элементов двумерной квадратной матрицы, находящихся по диагонали.

16. Выполнить произведение двух матриц.

Тема № 8

ФУНКЦИИ В ЯЗЫКЕ С++

Для написания больших программ, опыт показывает, что лучше пользоваться функциями. Программа будет состоять из отдельных фрагментов кода, под отдельным фрагментом кода понимается функция. Отдельным, потому, что работа отдельной функции не зависит от работы какой-нибудь другой. То есть алгоритм в каждой функции функционально достаточен и независим от других алгоритмов программы. Однажды, написав функцию, её можно будет с лёгкостью переносить в другие программы. Функция (в программировании) – это фрагмент кода или алгоритм, реализованный на каком-то языке программирования, с целью выполнения определённой последовательности операций. Итак, функции позволяют сделать программу модульной, то есть разделить программу на несколько маленьких подпрограмм (функций), которые в совокупности выполняют поставленную задачу. Еще один огромнейший плюс функций в том, что их можно многократно использовать. Данная возможность позволяет многократно использовать один раз написанный код, что в свою очередь, намного сокращает объем кода программы. С именем функции неразрывно связан адрес первой инструкции (оператора), входящей в функцию, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается обратно в адрес возврата – точку программы, где данная функция была вызвана. Функция может принимать параметры и должна возвращать некоторое значение, возможно пустое. Функция должна быть соответствующим образом *объявлена* и *определена*. *Объявление функции*, кроме имени, содержит список имён и типов, передаваемых параметров (или: аргументов), а также, тип возвращаемого функцией значения.

В С++ до всех программ должен задаваться прототип функции.

Прототип функции имеет синтаксис:

Тип результата имя функции (тип_аргумента имя аргумента, …….).

Функция может возвращать значения типа void, int, float,……. Имя функции может быть произвольным, но желательно, чтобы оно указывало на ее назначение. Если для выполнения функции нужно предоставить ей некоторую информацию в виде аргумента, то в круглых скобках должен быть указан тип аргумента и при необходимости его имя. Тип аргумента может быть любым. Если аргументов несколько, их описания разделяются запятыми.

Описание функции представляет собой часть программного кода, которая, как правило, следует за телом функции main().

Синтаксис описания функции следующий:

Тип результата имя функции (тип_аргумента имя аргумента, …….)

```
{
.
.
Тело функции
.
.
}
```

Пример 1. Написать функцию, находящую сумму 2-х целочисленных аргументов и возвращающую результат в виде целого числа.

```
#include <iostream>
int sum(int ix, int iy);
int main()
{
int ia=23;
int ib=13;
int ic;
```

```
ic=sum(ia,ib);
cout<<ic<<endl;
return 0;
system(“pause”);
}
int sum(int ix, int iy)
{
int iz;
iz=ix+iy;
return(iz);
}
```

В данном примере осуществлена передача аргументов по значению. Любая функция содержит список формальных аргументов, который может быть и пустым, если функция не принимает аргументов. Когда в программе осуществляется вызов функции, ей передается список значений аргументов, который называется списком фактических параметров. Типы идентификаторов в обоих списках должны совпадать. Когда функции предоставляется меньше аргументов, чем содержится в списке параметров, недостающим присваиваются неопределенные значения. Во избежание ошибок в С++ предусмотрено задание аргументам стандартных значений. Тогда, если при вызове функции соответствующий аргумент не указан, ему автоматически будет присвоено значение по умолчанию. Например, если прототип функции выглядит следующим образом:

```
int имя функции(int ft, float fu=4.2, int iv=10).
```

Если в вызове функций не будут указаны аргументы fu и/или iv, то им по умолчанию будут присвоены значения 4,2 и 10 соответственно. Стандарт языка С++ требует, чтобы аргументы, для которых заданы значения по умолчанию, находились в конце списка формальных аргументов. Допустимы следующие варианты вызова функции.

```
Имя функции (10)
Имя функции (10, 15.2)
Имя функции (10, 15.2, 8)
```

Параметры, которые передаются функции, могут передаваться как по значению, так и по ссылке: для переменной, переданной по значению создаётся локальная копия и любые изменения, которые происходят в теле функции с переданной переменной, на самом деле, происходят с локальной копией и никак не сказываются на самой переменной, в то время как изменения, которые происходят в теле функции с переменной, переданной по ссылке, происходят с самой переданной переменной.

Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции.

В следующем примере функция не получает аргументов и не возвращает никаких аргументов.

Пример 2. Вычислить с помощью функции корень квадратного числа.

```
#include <iostream>
#include <math.h>
using namespace std;
void voutput(void);
int main()
{
cout<<”программа вычисляет квадратный корень числа”<<endl;
voutput();
return(0);
system(“pause”);
}
void voutput(void)
{
int it=12345;
double du;
du=sqrt(it);
cout<<”Квадратный  корень числа равен”<< du<<endl;
}
```

Пример 3. Написать программу, которая считывает символ, введенный с клавиатуры и передает его в функцию, осуществляющую вывод нескольких копий символа на экран.

```
#include <iostream>
using namespace std;
void Symvol(char c);
int main()
{
char c;
cin>>c;
Symvol(c);
    return(0);
system(“pause”);
}
void Symvol(char c)
{
int i;
for(i=0;i<7;i++)
cout<<c<<" ";
cout<<endl;
}
```

Пример 4. Написать с использованием функции.

$$t=\frac{\sin 10+\sin 11+...+\sin 50}{(1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{20})^2}$$

```
#include <iostream>
#include <math.h>
double sum(void);
double sinus(void);
int main()
{double s;
s=sinus()/sum();
cout<<s<<endl;
system("pause");
}
```

```
double sinus(void)
{
int i;
double S=0;
for(i=10;i<=50;i++)
{double x=(double)i*3.14159/180.;
S+=sin(x);
return(S);
}
double sum(void)
{int i; double S=0;
for(i=1;i<=20;i++)
S+=1./(double)i;
S*=S;
return(S);
}
```

В С функции могут вызывать сами себя. Функция является рекурсивной, если оператор в теле функции вызывает функцию, содержащую данный оператор. Иногда называемая круговым определением, рекурсия является процессом определения чего-либо с использованием самой себя. Простым примером является функция fact(), вычисляющая факториал целого числа. Факториал числа N является произведением чисел от 1 до N. Например, факториал 3 равен 1*2*3 или 6. Как fact(), так и его итеративный эквивалент показаны ниже:

```
/* Вычисление факториала числа */
int factr(int n) /* рекурсивно */
{
int answer;
if(n==1) return(1);
answer = factr(n-1)*n;
return(answer);
}
/* Вычисление факториала числа */
int fact (int n)    /* нерекурсивно */
{
```

```
int t, answer;
answer = 1;
for(t=1; t<=n; t++)
answer*=t;
return(answer);
}
```

Действие нерекурсивной версии fact() должно быть совершенно очевидно. Она использует цикл, начиная с 1 и заканчивая указанным числом, последовательно перемножая каждое число на ранее полученное произведение.

Действие рекурсивной функции factr() немного более сложно. Когда factr() вызывается с аргументом 1, функция возвращает 1. В противном случае, она возвращает произведение factr(n- 1) * n. Для вычисления этого значения factr() вызывается с n-1. Это происходит, пока n не станет равно 1.

При вычислении факториала числа 2, первый вызов factr() приводит ко второму вызову с аргументом 1. Данный вызов возвращает 1, после чего результат умножается на 2 (исходное значение n). Ответ, таким образом, будет 2. Можно попробовать вставить printf() в factr() для демонстрации уровней и промежуточных ответов каждого вызова.

Если задать n=5 и выводить последовательно все значения, то результат будет следующим:

```
2
6
24
120
120
```

Когда функция вызывает сама себя, в стеке выделяется место для новых локальных переменных и параметров. Код функции работает с данными переменными. Рекурсивный вызов не создает новую копию функции. Новыми являются только аргументы. Поскольку каждая рекурсивно вызванная функция завершает работу, то старые локальные переменные и параметры удаляются из стека и выполнение продолжается с точки, в которой

было обращение внутри этой же функции. Рекурсивные функции вкладываются одна в другую как элементы подзорной трубы.

Рекурсивные версии большинства подпрограмм могут выполняться немного медленнее, чем их итеративные эквиваленты, поскольку к необходимым действиям добавляются вызовы функций. Но, в большинстве случаев, это не имеет значения. Основным преимуществом применения рекурсивных функций является использование их для более простого создания версии некоторых алгоритмов по сравнению с итеративными эквивалентами.

Под перегрузкой функции понимается, определение нескольких функций (две или больше) с одинаковым именем, но с различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом. Таким образом, перегрузка функций нужна для того, чтобы избежать дублирования имён функций, выполняющих сходные действия, но с различной программной логикой. Например, рассмотрим функцию areaRectangle(), которая вычисляет площадь прямоугольника.

```
float areaRectangle(float a, float b) //функция, вычисляющая площадь //прямоугольника с двумя параметрами a(см) и b(см)
{
return (a * b);
// умножаем длины сторон прямоугольника и возвращаем полученное //произведение
}
```

Итак, это функция с двумя параметрами типа float, причём аргументы передаваемые в функцию должны быть в сантиметрах, возвращаемое значение типа float – тоже в сантиметрах. Предположим, что наши исходные данные (стороны прямоугольника) заданы в метрах и сантиметрах, например, такие: a = 2 м 35 см; b = 1 м 86 см. В таком случае, удобно было бы использовать функцию с четырьмя параметрами. То есть, каждая длина сторон прямоугольника передаётся в функцию по двум параметрам: метры и сантиметры.

```
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm)
// функция, вычисляющая площадь прямоугольника с 4-мя параметрами //a(м) a(см); b(м) b(см)
{
return (a_m * 100 + a_sm) * (b_m * 100 + b_sm);
}
```

В теле функции значения, которые передавались в метрах (a_m и b_m) переводятся в сантиметры и суммируются со значениями a_sm, b_sm, после чего перемножаем суммы и получаем площадь прямоугольника в см.

Теперь, самое главное – у нас есть две функции, с разной сигнатурой, но одинаковыми именами (перегруженные функции). Сигнатура – это комбинация имени функции с её параметрами. Как же вызывать эти функции? А вызов перегруженных функций ничем не отличается от вызова обычных функций, например:

```
areaRectangle( 32, 43);
// будет вызвана функция, вычисляющая площадь прямоугольника с //двумя параметрами a(см) и b(см)
areaRectangle( 4, 43, 2, 12);
// будет вызвана функция, вычисляющая площадь прямоугольника с 4-мя //параметрами a(м) a(см); b(м) b(см)
```

Видно, что компилятор самостоятельно выберет нужную функцию, анализируя только лишь сигнатуры перегруженных функций. Минуя перегрузку функций, можно было бы просто объявить функцию с другим именем, и она бы хорошо справлялась со своей задачей. Но представьте, что будет, если таких функций надо больше, чем две, например 10. И для каждой нужно придумать осмысленное имя, а сложнее всего их запомнить. Вот именно поэтому проще и лучше перегружать функции, если, конечно, в этом есть необходимость.

Здесь приведен весь код программы:

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

```
using namespace std;
// прототипы перегруженных функций
float areaRectangle(float a, float b);
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm);
int main()
{
cout<< "S1 = " <<areaRectangle(32,43) <<endl;
// вызов перегруженной  функции 1
cout<< "S2 = " <<areaRectangle(4, 43, 2, 12) <<endl;
// вызов перегруженной функции 2
return 0;
}
// перегруженнаяфункция 1
float areaRectangle(float a, float b)
//функция, вычисляющая площадь прямоугольника с двумя //параметрами a(см) и b(см)
{
return (a * b);
// умножаем длины сторон прямоугольника и возвращаем полученное //произведение
}
// перегруженная функция 2
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm)
// функция, вычисляющая площадь прямоугольника с 4-мя параметрами //a(м) a(см); b(м) b(см)
{
return (a_m * 100 + a_sm) * (b_m * 100 + b_sm);
}
```

Могут существовать функции, имеющие одинаковые имена, но отличающиеся типами передаваемых параметров. Например, можно сделать две функции для возведения в степень чисел типа int и типа double:

```
int power(int x, int n);
double power(double x, double n);
```

При этом первая функция будет работать быстрее, поскольку операции с целочисленными данными выполняются быстрее, чем операции над числами с плавающей точкой, поэтому для целых чисел пользоваться первой функцией предпочтительнее.

Задания

1. Написать функцию, которая вычисляет объем цилиндра. Параметрами функции должны быть радиус и высота цилиндра.

2. Написать функцию, которая возвращает максимальное из двух целых чисел, полученных в качестве аргумента.

3. Написать функцию, которая сравнивает два целых числа и возвращает результат сравнения в виде одного из знаков: >, <или =.

4. Написать функцию, которая вычисляет сопротивление цепи, состоящей из двух резисторов. Параметрами функции являются величины сопротивлений и тип соединения (последовательное или параллельное). Функция должна проверять корректность параметров: если неверно указан тип соединения, то функция должна возвращать -1.

5. Написать функцию, которая вычисляет значение a^b. Числа a и b могут быть любыми дробными положительными числами.

6. Написать функцию Procent, которая возвращает процент от полученного в качестве аргумента числа.

7. Написать функцию "Факториал" и программу, использующую эту функцию для вывода таблицы факториалов.

8. Написать функцию Dohod, которая вычисляет доход по вкладу. Исходными данными для функции являются: величина вклада, процентная ставка (годовых) и срок вклада (количество дней).

9. Написать функцию, которая выводит на экран строку, состоящую из звездочек. Длина строки (количество звездочек) является параметром функции.

10. Написать функцию, которая выводит строку, состоящую из одинаковых символов. Длина строки и символ являются параметрами функции.

11. Написать функцию, которая вычисляет объем и площадь поверхности параллелепипеда.

12. Вычислить с помощью функции:

Y = sh(x) * tg(x+1) – tg^2(2+sh(x-1));

$$sh(x) = \frac{e^x - \frac{1}{e^x}}{2} = \frac{e^x - e^{-x}}{2}$$

x=1 y=-7.34226

13. Написать с использованием функции.

$$t = \frac{\sin 10 + \sin 11 + ... + \sin 50}{(1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{20})^2}$$

14. С использованием функции по заданным вещественным значениям x, y вычислить:

U = max(x+y, x*y);
v = max(0.5,U).

Тема № 9

ПЕРЕДАЧА ОДНОМЕРНЫХ МАССИВОВ В ФУНКЦИИ

Пример 1. Передать элементы массива в функцию и напечатать их.

```
#include <iostream>
using namespace std;
void fp(double farray[]);
int main()
{
double      fm[7]={1.1,2.2,3.3,4.4,5.5,6.6,7.7};
fp(fm);
            return(0);
system("pause");
}
void  fp(double farray[])
{
int i;
for(i=0;i<7;i++)
cout<<farray[i]<<" "<<endl;
}
```

Пример 2. Передать в функцию одномерный массив, вычислить произведение элементов одномерного массива.

```
#include <iostream>
using  namespace std;
double  fp(double farray[], int n);
int main()
{
double       fm[7]={1.1,2.2,3.3,4.4,5.5,6.6,7.7};
```

```
double f;
int n; double r;
r=sizeof fm/(sizeof (double));
n=(int)r;
f=fp(fm,n);
cout<<f<<endl;
return(0);
system(“pause”);
}
double fp(double farray[], int n)
{
    double p;
int i;
p=farray[0];
for(i=1;i<n;i++)
    p*=farray[i];
return(p);
}
```

В следующей программе создаются два прототипа функции с одним именем и общей областью видимости, но аргументами разных типов. При вызове функции adder() будут обрабатываться данные либо типа int, либо типа float.

```
// Эта программа на языке C++ содержит пример перегрузки функции.
#include <iostream.h>
int adder (int iarray[ ]);
float adder (float farray [ ]);
int main()
{
int iarray[7]  = {5,1, 6, 20,15,0, 12);
float farray[7] = {3.3, 5.2, 0.05, 1.49, 3.12345, 31.0, 2.007};
int isum;
float fsum;
isum    = adder (iarray) ;
fsum= adder (farray) ;
cout<< "Сумма массива целых чисел равна " << isum << endl;
```

```
cout<< "Сумма массива дробных чисел равна " << fsum<< endl;
return ( 0 );
}
int adder(int iarray[ ])
{
int i;
int ipartial;
ipartial = iarray[0];
for(i= 1; i < 7; i++)
 ipartial += iarray[i];
return(ipartial);
}
float adder (float farray[ ])
 {
int i;
float fpartial;
fpartial = farray [0];
for(i= 1; i < 7; i++)
fpartial += farray[i];
return (fpartial) ;
}
```

При использовании перегруженных функций часто допускается ряд ошибок. Например, если функции отличаются только типом возвращаемого значения, но не типами аргументов, такие функции не могут иметь одинаковое имя. Также недопустим следующий вариант перегрузки:

```
int имя_функции(int имя_аргумента);
int имя_функции(int имя_аргумента);
// недопустимая перегрузка имени
```

Это неправильно, поскольку аргументы имеют одинаковый тип.

В программировании, область видимости обозначает область программы, в пределах которой идентификатор (имя) некоторой переменной продолжает быть связанным с этой переменной и возвращать её значение. За пределами области види-

мости тот же самый идентификатор может быть связан с другой переменной, либо быть свободным (не связанным ни с какой из них).

В большинстве языков программирования область видимости переменной определяется местом её объявления. Кроме того, область видимости может задаваться явно с помощью классов памяти или пространств имен.

Классы памяти

Имеются четыре спецификации класса памяти: auto, register, static, extern. Переменные, объявленные со спецификациями «auto» и «register» являются локальными, «static» и «extern» – глобальные. Спецификации класса памяти *auto* и *register* могут быть использованы только на внутреннем уровне. Память для локальной переменной выделяется заново всякий раз, когда выполнение программы достигает блока, в котором объявлена переменная и удаляется по завершении блока.

Спецификация класса памяти *register* требует, чтобы переменной была выделена память в регистре, если это возможно. Так как работа с регистрами происходит быстрее, спецификация класса памяти *register* обычно используется для переменных, к которым предполагается обращаться очень часто.

Переменная, объявленная со спецификатором «static» на внутреннем уровне будет глобальной, но доступ к ней можно получить, только внутри ее блока. В отличие от автоматической переменной, статическая сохранит свое значение после завершения блока. Память для глобальной переменной выделяется 1 раз при запуске программы и удаляется по завершении программы.

Указанные 4 спецификатора определяют также область видимости переменных и функций, т.е. часть программы, в пределах которой к идентификатору можно обратиться по имени. На область видимости переменной и функции влияет место ее объявления в программе. Если объявление расположено вне любой функции, то говорят о внешнем уровне объявления. Если же оно находится в теле функции, то говорят о внутреннем уровне объявления.

Объявление переменных на внешнем уровне

Переменная объявленная на внешнем уровне является глобальной и по умолчанию имеет класс памяти extern. Внешнее объявление может включать инициализацию или просто быть ссылкой на переменную, инициализируемую в другом месте.

```
static  int iv1;   // по умолчанию присваивается 0.
static  int iv1=10;
```

Область видимости глобальной переменной распространяется до конца файла. Обращение к переменной не может находиться выше той строки, где она была объявлена. Переменная объявляется на внешнем уровне только 1 раз. Если в одном из файлов создана переменная с классом памяти static, то она может быть объявлена под тем же именем с тем же спецификатором static в любом другом исходном файле. Так как статические переменные доступны только в пределах своего файла, конфликтов имен не возникает.

С помощью спецификатора extern можно объявить переменную, которая будет доступна из любого места программы. Это может быть ссылка на переменную, описанную в другом файле, или ниже в том же файле.

Пример использования локальных переменных.

```
#include <iostream>
using namespace std;
int main()
{
for(int i=0;i < 10;i++)
{
int k = i*i;
cout<<k<<endl;
}
system(“pause”);
return 0;
}
```

В этом примере объявляются две переменные i и k. Причем переменная k объявлена внутри цикла for. Спрашивается можно

ли ее использовать и за пределами этого цикла? В данном случае ответ будет отрицательный, т.к. переменная k «пропадает» за пределами тела цикла и существует только внутри него. Условия данного примера можно обобщить и сказать, что обычная переменная, объявленная внутри фигурных скобок {}, видна только в них и не может быть использована за их пределами. По этой причине переменные объявленные, например, внутри функции main() недоступны в других функциях и наоборот.

Однако если объявить переменную в начале программы, а не внутри функции, то они становятся доступными всем функциям и будут иметь глобальную область видимости. Такие переменные называются глобальными. Следующий пример показывает объявление и использование глобальных переменных.

Определение глобальных переменных.

```
#include <iostream>
using namespace std;
int global_var = 0;
void my_func(void)
{global_var = 10;}
int main()
{
cout<<global_var<<endl;
my_func();
cout<<global_var<<endl;
system(“pause”);
return 0;
}
```

Результатом выполнения программы будет следующий текст:

```
0
10
```

В этом примере объявлена глобальная переменная global_var, которая может использоваться и в функции main() и в функции my_func() и соответственно меняет свое значение. В языке C++ можно задавать переменные с одинаковыми именами и типами если они принадлежат разной области видимости. Например, можно задать глобальную переменную var типа int и такую же

переменную внутри функции main(). В этом случае простое обращение к переменной по ее имени будет означать работу с локальной переменной, а если необходимо работать с глобальной, то перед ее именем необходимо поставить два двоеточия ‘::’ .

Следующий пример показывает такой способ работы с переменными.

Работа с глобальными и локальными переменными.

```
#include <iostream>
using namespace std;
int var = 5;
int main()
{
int var = 10;
cout<<var<<“  “<<::var;
system(“pause”);
return 0;
}
```

В результате получим

var = 10 ::var = 5

В языке С++ можно задавать класс переменных, которые будучи объявленными внутри фигурных скобок {} не исчезают и за их пределами, но в то же время имеют область видимости только внутри них. Класс таких переменных называется статическим и задается с помощью ключевого слова static. Поясним сказанное на следующем примере.

Использование статических переменных.

```
#include <iostream>
using namespace std;
void iter(void);
int main()
{
for(int i=0;i < 4;i++)
iter();
system(“pause”);
return 0;
```

```
}
void iter(void)
{
int var = 1;
static int var_st = 1;
cout<<var++<<“  “<<var_st++<<endl;
}
```

В результате на экране появятся следующие строчки:

```
var = 1, var_st = 1
var = 1, var_st = 2
var = 1, var_st = 3
var = 1, var_st = 4
```

Анализ полученных результатов показывает, что статическая переменная var_st объявленная внутри функции iter() не исчезает после ее завершения и при повторном вызове функции не инициализируется заново, т.к. она уже существует.

Рассмотрим пример:

```
#include <iostream>
using namespace std;
void t(void);
int main(void)
{
int count;
for(count=1;count<=3;count++)
{
cout<<“начинается итерация “<<count<<endl;
t();
}
system(“pause”);
return 0;
}
void t(void)
{
int fade=1;
static int stay=1;
cout<<fade++<<“  “<<stay++<<endl;
}
```

Результат будет следующий:
Начинается итерация 1:
fade=1 stay=1
Начинается итерация 2:
fade=1 stay=2
Начинается итерация 3:
fade=1 stay=3

Значение статической переменной stay увеличивается на 1, в то время как переменная fade создается заново каждый раз. Отсюда следует, что инициализация этих переменных производится по-разному: переменная fade инициализируется каждый раз, когда вызывается функция t(), а переменная stay только 1 раз во время компиляции функции t(). Статические переменные инициализируются нулем, если они явно не инициализированы другим значением. Статические и внешние переменные уже находятся в нужном месте сразу же после того, как программа будет загружена в память. Тот факт, что оператор помещен в функцию t(), говорит о том, что только функции t() предоставлена возможность видеть эту переменную. В следующих 3-х примерах показаны возможные комбинации внешних и автоматических переменных.

```
int h;
int magic();
int main()
{extern int h;
}
```
h объявлена как внешняя переменная.
```
int magic()
{extern int h;
}
```
Та же переменная, что и выше.
```
int h;
int magic();
int main()
{extern int h;
}
```

```
Переменная h объявлена  как внешняя.
int magic()
{
}
Переменная h не объявлена, но известна.
#include <iostream.h>
using namespace std;
int units=0;
void critic(void);
int main()
{
extern int units;
cout<<“Сколько стоит маленький бочонок меда”<<endl;
cin>>units;
while(units!=56)
critic();
cout<<“Вы должны это проверить ”<<endl;
system(“pause”);
return 0;
}
void critic(void)
{
cout<<“Вам не повезло. Попытайтесь еще раз.”<<endl;
cin>>units;
}
```

Результат будет следующий:

```
Сколько весит маленький бочонок меда?
14
Вам не повезло. Попытайтесь еще раз.
56
Вы должны это проверить.
```

Функции main() и critic() используют идентификатор units для доступа к одной и той же переменной. Переменная units имеет область видимости в пределах файла. Аналогично можно не объявлять переменную units в функции main(), чтобы обеспечить доступ функции main() к переменной units.

Рассмотрим пример с двумя файлами.

Файл A

```
#include <iostream>
using namespace std;
int ivalue=10;
void a(void);
void b(void);
void main()
{ extern int ivalue;
    ivalue++;
    cout<<ivalue<<"\n"; //Выводит значение  11
    a();
system(“pause”);
}
void a(void)
{
    ivalue++;
    cout<<ivalue<<"\n"; //Выводит значение  12
    b();
cout << ivalue << "\n";
}
```

Файл B

```
using namespace std;
extern int ivalue ;
void b (void)
{
ivalue++;
// ссылка на переменную ivalue,
// описанную в файле A
// выводит значение 13
}
```

Рассмотрим еще 1 пример с двумя файлами.

Первый файл (**main.c**)

```
#include <iostream>
#include "D:\second.h“
using namespace std;
float x = 99, y = 77;
```

```
char ch;
void func1(void);
int main()
{ch = 'Z';
func1();
cout<<“Press any key:“<<endl;
system(“pause”);
return 0;}
void func1(void)
{func22();
func23();
cout<<“x=“<<x<<“y=“<<y<<“ch=“<<ch<<endl;
}
```

Второй файл (**second.h**)

```
extern float x, y;
extern char ch;
void func22(void)
{y = 100;}
void func23(void)
{
x = y/10;
ch = 'R';
}
```

В программе первый файл – это основная часть программного проекта. Второй файл создан как текстовый файл (с помощью блокнота) с расширением *.h. Список глобальных переменных (**x, y, ch**) копируется из первого файла во второй, а затем добавляется спецификатор extern. Он сообщает компилятору, что имена и типы переменных, следующих далее, объявлены в другом месте. Все ссылки на внешние переменные распознаются в процессе редактирования связей. Подключение второго файла выполнено с указанием имени диска (D:), на котором расположен файл second.h. Для подключения имени файла, созданного пользователем, его заключают в двойные кавычки.

Результат выполнения программы будет следующий:

```
x=10 y=100  ch=R
press any key
```

Объявление переменных на внутреннем уровне

При объявлении переменных на внутреннем уровне можно использовать любой из четырех спецификаторов класса памяти (по умолчанию устанавливается класс auto). Переменные, имеющие класс памяти auto являются локальными. Их область видимости ограничена блоком, в котором они объявлены.

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

Переменная, объявленная на внутреннем уровне со спецификатором static, будет глобальной, но доступ к ней можно получить только внутри ее блока. В отличие от автоматической переменной, статическая переменная сохранит свое значение после завершения блока. По умолчанию статической переменной присваивается значение «0», но можно инициализировать ее некоторым константным выражением.

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

```
#include <iostream>
using namespace std;
int ivalue1=1;
void a (void);
void main ()
{
extern   int ivalue1;
static     int ivalue2;
(auto-по умолчанию)   int ivalue3=0;
register int revalue=0;
```

```
cout<<ivalue1<<endl<<ivalue2<<endl<<ivalue3<<endl<<reval
ue<<endl;
//1 0  0  0
a() ; // обращение к функции
system (“pause”);
return 0;
}
void a (void)
    {
//сохранение адреса глобальной переменной ivalue1
static int *pivalue1=&ivalue1;
    // создание новой, локальной переменной ivalue1;
// тем самым глобальная переменная ivalue1 становится
 //недоступной
int ivalue1=32;
// создание новой статической переменной ivalue2,
// видимой только внутри функции  a()
static int ivalue2=2;
    ivalue2+=2;
    cout<<ivalue1<<”  “<<ivalue2<<” “<<*pivalue1<<endl;
 //32    4    1
    }
```

Поскольку переменная ivalue1 переопределяется внутри функции a (), доступ к глобальной переменной ivalue1 блокируется. Тем не менее с помощью указателя pivalue1 можно получить доступ к глобальной переменной, сославшись на нее по адресу.

Правила определения области видимости переменных

Чтобы определить, в какой части программы будет доступна та или иная переменная, нужно воспользоваться следующими правилами. Областью видимости переменной может быть программный блок, функция, файл и вся программа. Переменные, объявленные внутри блока или функции, доступны только в их пределах.

Переменные, объявленные на уровне файла, т.е. вне любой функции, и имеющие спецификатор static, доступны во всем файле, начиная со строки объявления. Переменные, объявленные в одном из файлов программы на внешнем уровне без спецификатора static или со спецификатором extern, видимы в пределах всей программы.

Объявление функций

При объявлении функций на внешнем или внутреннем уровне можно указывать спецификаторы static или extern. В отличие от переменных, функции всегда глобальны. Правила видимости функций слегка отличаются от правил видимости переменных.

Функции, объявленные как статические, можно вызывать внутри файла, в котором они реализованы, но они не доступны из других файлов программы. Кроме того, в разных файлах можно создавать статические функции с одинаковыми именами, что не приведет к конфликтам имен.

Функцию, объявленную как внешняя, можно использовать во всех файлах программы (если только в одном из них она не переопределена как статическая). Если при объявлении функции не указан класс памяти, то по умолчанию она считается внешней.

Основные концепции:

• Локальные переменные представляют собой переменные, объявленные внутри функции.

• Локальные переменные известны только той функции, внутри которой они объявлены.

• Несколько функций могут использовать одно и то же имя для локальной переменной без каких-либо конфликтов.

• Глобальная переменная представляет собой переменную, чье имя и значение известны на протяжении всей программы.

• Чтобы объявить глобальную переменную, объявите переменную в начале вашего исходного файла вне какой-либо функции.

• Поскольку глобальные переменные могут быть легко изменены любой функцией, они привносят возможность появления трудно обнаруживаемых ошибок, поэтому избегайте использования глобальных переменных в своих программах.

Массивы

Массивы могут иметь те же типы данных и классы памяти, что и простые переменные.

```
int t[365]; //внешний массив
void main()
{float r[365];
static char c[12];
extern t[];
}
```

Инициализация массивов

Массив инициализируется 3-мя способами:

при создании массива, используем инициализацию по умолчанию;

при создании массива, явно указывая начальные значения;

в процессе выполнения программы.

Явная инициализация:

```
static float f[4]={1.14,0.75,5.5,-33};
```

Можно задавать элементы массива до всех программ:

```
static float f[]={1.14,0.75,5.5,-33};
int main()
{
int n;
n=sizeof f/sizeof(float);
int i;
for(i=0;i<n;i++)
cout<<f[i]<<"  ";
cout<<endl;
system("pause");
}
```

Инициализация внешнего массива:

```
int d[12]={31,28,31,30,31,30,34,34,65,78,12, 43};
void main()
{
int i;
extern int d[ ];
```

```
for(i=0;i<12;i++)
cout<<i+1<<“   “<<d[i]<<endl;
}
```

Вычисление размера массива в байтах.

```
void main()
{
static int w[7]={1,2,3,4,5,6,7};
cout<<”массив занимает”(int) sizeof (w)<<”байт”;
}
```

Вычисление размера внешнего массива.

```
int days[ ]={21,34,45,67};
void main()
{int i;
for (i=0;i<sizeof days/sizeof(int);i++)
cout<<i<<“    “<<days[i]<<endl; system(“pause”);
}
```

Задания

1. Вычислить среднее арифметическое значение элементов массива с помощью функции. Элементы массива задать в программе.

2. Вычислить произведение элементов массива с помощью функции. Элементы массива задать в программе.

3. Передать в функцию 2 массива и сложить их. Элементы массива задать в программе.

4. Передать в функцию массив. Переписать элементы массива в обратном порядке.

5. Передать в функцию одномерный массив. Вычислить выражение:

$$Y = a_1 * a_{11} + a_2 * a_{22} + \ldots + a_{10} * a_{20}$$

6. Передать в функцию 2 одномерных массива. Объединить эти массивы в один.

Тема № 10

ПЕРЕДАЧА ДВУМЕРНЫХ МАССИВОВ В ФУНКЦИИ

Пример 1. Передать в функцию двумерный массив. Вычислить сумму элементов двумерного массива.

```
#include <iostream>
int mn(int p[][2],int m,int n);
int main()
{
int r;
int a[2][2]={
{1,2},{3,4}
    };
r=mn(a,2,2);
cout<<r<<endl;
return 0;
system("pause");
}
int mn(int p[][2],int m,int n)
    {
int i,j,s=0;
for(i=0;i<m;i++)
for(j=0;j<m;j++)
s+=p[i][j];
return(s);
}
```

Пример 2. Передать в функцию двумерный массив. Вычислить сумму элементов массива, находящихся на дополнительной диагонали.

```
#include <iostream>
using namespace std;
```

```
int diag(int p[][2],int m);
int main()
{
    int r;
int a[2][2]={
{3,7},{9,1}
    };
r=diag(a,2);
cout<<r<<endl;
system("pause");
return 0;
}
int diag(int p[][2],int m)
    {
int i,j,s=0;
for(i=0;i<m;i++)
for(j=0;j<m;j++)
    if((i+j)==(m-1))
s+=p[i][j];
return(s);}
```

Задания

1. Передать в функцию двумерный массив. Вычислить произведение элементов двумерного массива.

2. Передать в функцию двумерный массив. Вычесть из всех столбцов последний, кроме последнего столбца.

3. Передать в функцию двумерный массив. Вычислить сумму элементов массива, находящихся на главной диагонали.

4. Передать в функцию два двумерных массива. Выполнить сложение этих массивов.

5. Передать в функцию 2 двумерных массива. Выполнить перемножение этих массивов.

6. Передать в функцию двумерный массив. Вычислить сумму элементов матрицы выше главной диагонали.

7. Передать в функцию двумерный массив. Найти минимальное и максимальное значения массива.

Тема № 11

УКАЗАТЕЛИ НА ПЕРЕМЕННЫЕ

Начинающие программисты, как правило, работают только с обычными переменными. Для таких переменных память выделяется автоматически при запуске программы или функции, в которой они объявлены, и удаляется также автоматически при завершении программы или функции. Доступ к ним организуется достаточно просто – по имени:

s + = 10.

Другой способ получения доступа к значению переменной заключается в использовании другой переменной, которая содержит ее адрес. Предположим, имеется переменная типа int с именем i и переменная p_address, являющаяся указателем на нее. Как вы уже знаете, в C/C++ есть оператор взятия адреса &, который возвращает адрес своего операнда. Поэтому вам будет нетрудно разобраться в синтаксисе присвоения одной переменной адреса другой переменной:

p_address = &i;

Переменные, которые хранят адреса других переменных, называются указателями. В языке C++ имеются переменные типа указатель. Значением переменной типа указатель служит адрес некоторой величины. Если имя указателя ptr, то ptr=&p присваивает адрес p переменной ptr. & – операция взятия адреса. Чтобы определить значение, на которое указывает ptr, нужно воспользоваться операцией косвенной адресации.

V = *ptr;

При работе с указателями нам нужен лишь один адрес – адрес первой ячейки, именно он и послужит значением, которым мы проинициализируем указатель. Например,

```
void main()
{     int  i_val = 7;
  int  *i_ptr = &i_val;
}
```

Используя унарную операцию взятия адреса **&**, мы извлекаем адрес переменной *i_val* и присваиваем ее указателю. Здесь стоит обратить внимание на следующие вещи:

Тип, используемый при объявлении указателя в точности должен соответствовать типу переменной, адрес которой мы присваиваем указателю.

Примеры описания указателей:

int *pi; //указатель на переменную типа целого

char *pc; //указатель на символьную переменную

float *pf,*pg; //указатели на переменные с плавающей точкой.

Бывают случаи, когда нельзя использовать функцию без указателей.

Рассмотрим пример функции без указателей, в которой выполняется обмен местами 2-х переменных.

```
#include <iostream>
using namespace std;
void interchange(int u,int v);
void main()
{
int x=5,y=10;
cout<<x<<" "<<y<<endl;
interchange(x,y);
cout<<x<<" "<<y<<endl;
system("pause");
}
interchange(int u,int v)
{
```

```
int temp;
temp=u;
u=v;
v=temp;
}
```

После выполнения программы результат будет следующим

```
5 10
5 10
```

Внутри функции interchange переменные u, v меняются местами. Однако это никак не влияет на переменные x,y. Оператор return возвращает в вызывающую программу только одну величину, а нужно передать 2, для этого нужно воспользоваться указателями.

```
#include <iostream>
void interchange(int *u,int *v);
void main()
{
int x=5,y=10;
cout<<x<<" "<<y<<endl;
interchange(&x,&y);
cout<<x<<" "<<y<<endl;
}
interchange(int *u,int *v)
{
int temp;
temp=*u;
*u=*v;
*v=temp;
}
```

После выполнения программы результат будет следующим

```
5 10
10 5
```

При вызове функции информация о переменной может передаваться функции в 2-х видах. Если используется форма обращения function1(x), происходит передача значения переменной x.

Если же используется форма обращения function2(&x), происходит передача адреса переменной x.

Инициализация указателей

Указатели можно инициализировать при объявлении.

Например,

int ir;

int *pr=&ir;

При объявлении указателя *pr, ему присваивается адрес переменной ir, а переменная ir не инициализирована.

Оператор взятия адреса можно применять не с каждым выражением.

Приведем примеры неправильного использования операции &.

P=&48;

int ir=5; p=&(ir+15);

Теперь мы можем оперировать не только непосредственно с помощью самой переменной, но и с помощью указателя на нее.

Рассмотрим простой пример:

```
#include <iostream>
using namespace std;
void main()
{
  int  i_val = 7;
  int * i_ptr = &i_val;
 // выведем на экран значение переменной i_val
cout <<  i_val << endl;
 cout << *i_ptr << endl;
}
```

1. Здесь все ясно – используем саму переменную.

2. Во втором случае – мы обращаемся к значению переменной *i_val* через указатель. Но, как видно, мы не просто используем имя указателя – здесь используется операция разыменования: она позволяет перейти от адреса к значению.

Можем ли мы непосредственно через указатель оперировать со значением переменной, на которую он указывает? Да, конеч-

но, для этого они и реализованы. Все, что нужно – сделать разыменование указателя:

```
(*i_ptr)++;
// результат эквивалентен операции инкремента самой переменной:
i_val++;
// т.е. в данном случае в i_val сейчас хранится значение не 7, а 8.
```

Обычно указатели используются для работы с динамически созданными объектами, для построения связанных структур данных, таких, как связанные списки и иерархические деревья, и для передачи в функции больших объектов – массивов и объектов классов – в качестве параметров. Каждый указатель ассоциируется с некоторым типом данных, причем их внутреннее представление не зависит от внутреннего типа: и размер памяти, занимаемый объектом типа указатель, и диапазон значений у них одинаков. Разница состоит в том, как компилятор воспринимает адресуемый объект. Указатели на разные типы могут иметь одно и то же значение, но область памяти, где размещаются соответствующие типы, может быть различной:

указатель на int, содержащий значение адреса 1000, направлен на область памяти 1000-1003 (в 32-битной системе);

указатель на double, содержащий значение адреса 1000, направлен на область памяти 1000-1007 (в 32-битной системе).

Например:

```
int         *ip1, *ip2;
double      *dp;
```

Указатель обозначается звездочкой перед именем. В определении переменных списком звездочка должна стоять перед каждым указателем. В примере ниже lp – указатель на объект типа long, а lp2 – объект типа long:

```
long *lp, lp2;
```

В следующем случае fp интерпретируется как объект типа float, а fp2 – указатель на него:

```
float fp, *fp2;
```

Пусть задана переменная типа int:

int ival = 1024; Ниже приводятся примеры определения и использования указателей на int pi и pi2:

```
    // pi2 инициализирован адресом ival
int *pi2 = &ival;
    int *pi;
    // правильно: pi и pi2 содержат адрес ival
pi = pi2;
//*pi и *pi2 имеют значения 1024.
```

Переменной, используемой для хранения адреса, не может быть присвоена величина, не являющаяся адресом.

```
    pi=ival;
```

Нельзя присвоить указателю одного типа значение, являющееся адресом объекта другого типа. Если определены следующие переменные:

```
    double dval;
    double *ps = &dval;
```

то выражение присваивания, приведенное ниже, вызовет ошибку компиляции:

```
    // ошибки компиляции недопустимое присваивание типов данных:
    int* <== double*
    pi = &dval;
```

Для того чтобы обратиться к объекту, имея его адрес, нужно применить операцию разыменования, или косвенную адресацию, обозначаемую звездочкой (*). Имея следующие определения переменных:

```
    int ival = 1024;
    Int ival2 = 2048;
    int *pi = &ival;
```

мы можем читать и сохранять значение ival, применяя операцию разыменования к указателю pi:

```
    // косвенное присваивание переменной ival  значения ival2
    *pi = ival2;
```

Значение ival станет равно 2048.

Операции с указателями

• **Присваивание**. Указателю можно присвоить адрес. Обычно мы выполняем это действие, используя имя массива или операцию получения адреса &.

• **Определение значения**. Операция * выдает значение, хранящееся в указанной ячейке.

• **Получение адреса указателя**. Подобно любым переменным, переменная типа указатель имеет адрес и значение. Операция & сообщает нам, где находится сам указатель.

• **Увеличение указателя**. Мы можем выполнять это действие с помощью обычной операции сложения либо с помощью операции увеличения. Увеличивая указатель, мы перемещаем его на следующий элемент массива.

• **Разность**. Можно находить разность двух указателей. Обычно это делается для указателей, ссылающихся на элементы одного и того же массива, чтобы определить, на каком расстоянии друг от друга находятся элементы. Помните, что результат имеет тот же тип, что и переменная, содержащая размер массива.

```
int var = 123;
int *p = &var;
```

(*p)++ и ++*p это одно и то же, увеличивается значение переменной var на 1, т.е. будет 124.

Тогда почему *p++ и *p=*p+1 это не одно и то же? Почему в первом случае мы сдвигаем адрес памяти, а во втором значение переменной? *p++ – сдвигается значение адреса, *p=*p+1 – сдвигается значение переменной, будет 124.

Разработаем программу, которая будет использовать указатели.

```
# include "stdafx.h"
# include <iostream>
using namespace std;
int main()
{int var = 123; // инициализация переменной var числом 123
int *ptrvar = &var; // указатель на переменную var (присвоили адрес //переменной указателю)
 cout << "&var    = " << &var << endl;// адрес переменной var, содержащийся в //памяти, извлечённый операцией взятия адреса
 cout << "ptrvar  = " << ptrvar << endl;
// адрес переменной var, является значением указателя ptrvar
cout << "var     = " << var << endl;
// значение в переменной var
```

```
cout << "*ptrvar = " << *ptrvar << endl;
// вывод значения, содержащегося в переменной var через указатель, //операцией разыменования указателя
 system("pause");
  return 0;
}
```

Разработаем программу, которая будет сравнивать адреса указателей.

```
# include "stdafx.h"
# include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
int var1 = 123; // инициализация переменной var1 числом 123
  int var2 = 99; // инициализация переменной var2 числом 99
 int *ptrvar1 = &var1; // указатель на переменную var1
 int *ptrvar2 = &var2; // указатель на переменную var2
cout << "var1    = " << var1 << endl;
 cout << "var2    = " << var2 << endl;
 cout << "ptrvar1 = " << ptrvar1 << endl;
cout << "ptrvar2 = " << ptrvar2 << endl;
  if (ptrvar1 > ptrvar2)
// сравниваем значения указателей, то есть адреса переменных
cout << "ptrvar1 > ptrvar2" << endl;
  if (*ptrvar1 > *ptrvar2)
 //сравниваем значения переменных, на которые ссылаются указатели
   cout << "*ptrvar1 > *ptrvar2" << endl;
    system("pause");
    return 0;
}
```

Ссылки в языке C++

Язык C++ предоставляет возможность прямого обращения к переменным по адресам, что даже проще, чем использовать ука-

затели. В C++ допускается создание как обычных переменных, так и указателей. В первом случае резервируется область памяти для непосредственного сохранения данных, тогда как во втором случае в памяти компьютера выделяется место для хранения адреса некоторого объекта, который может быть создан в любое время. В дополнение к этому язык C++ предлагает третий вид объявления переменных – ссылки. Как и указатели, ссылки содержат данные о размещении других переменных, но для получения этих данных не нужно использовать специальный оператор косвенного обращения. Синтаксис использования ссылок в программе довольно прост:

```
int iresult_a = 5;
int &riresult_a = iresult_a;
```

В данном примере создается ссылка (на это указывает символ & после имени типа данных) riresult_a, которой присваивается адрес существующей переменной iresult_a. С этого момента на одну и ту же ячейку памяти ссылаются две переменные: iresult_a и riresult_a. По сути, это два имени одной и той же переменной. Любое присвоение значения переменной riresult_a немедленно отразится на содержимом переменной iresult_a. Справедливо и обратное утверждение: любое изменение переменной iresult_a вызовет изменение значения переменной riresult_a. Таким образом, можно сказать, что использование ссылок в языке C++ позволяет реализовать систему псевдонимов переменных.

В отличие от указателей, на ссылки накладывается следующее ограничение: их инициализация должна производиться непосредственно в момент объявления, и однажды присвоенное значение не может быть изменено в ходе выполнения программы. То есть если при объявлении ссылки вы присвоили ей адрес какой-либо переменной, то эта ссылка на протяжении всего времени работы программы будет связана только с этой ячейкой памяти. Вы можете свободно изменять значение переменной, но адрес ячейки, указанный в ссылке, изменить невозможно. Таким образом, ссылку можно представить как указатель с константным значением адреса.

Рассмотрим некоторые примеры. В следующей строке значение, присвоенное выше переменной iresult_a (в нашем примере – 5), будет удвоено:

riresult_a *= 2;

Следующее выражение также является допустимым:

int *piresult_a = &riresult_a;

Здесь адрес ссылки riresult_a (т.е., по сути, адрес переменной iresult_a) присваивается указателю piresult_a типа int. Результатом iresult_a, riresult_a, *piresult_a будет 10.

Пример 1. Заменить 2 числа на их сумму и разность.

```
#include <iostream>
using namespace std;
void sumrazn(int *u,int *v);
void main()
{  int x,y;
cin>>x>>y;
sumrazn(&x,&y);
cout<<x<<endl<<y<<endl;
system("pause");
}
void sumrazn(int *u,int *v)
{
    int t;
    t=*u;
    *u=*u+*v;
    *v=t-*v;
}
```

Задания

1. Заменить 2 числа на их сумму и разность.
2. Заменить 2 числа на их произведение и частное.
3. Решить систему линейных уравнений

a11*x1+a12*x2=b1

a21*x1+a22*x2=b2

4. Написать функцию, обеспечивающую решение квадратного уравнения. Параметрами функции должны быть коэффициенты и корни уравнения. Значение, возвращаемое функцией, должно передавать в вызывающую программу информацию о наличии у уравнения корней: 2 – два разных корня, 1 – корни одинаковые, 0 – уравнение не имеет решения.

5. Найти площадь прямоугольника и периметр с помощью функции и указателей.

6. Вычислить произведение и частное 2-х чисел с помощью функции и указателей.

7. Найти сумму 2-х чисел и среднее арифметическое с помощью функции и указателей.

8. Найти длину круга и площадь окружности.

Тема № 12

УКАЗАТЕЛИ НА ОДНОМЕРНЫЕ МАССИВЫ. ОДНОМЕРНЫЕ ДИНАМИЧЕСКИЕ МАССИВЫ. УКАЗАТЕЛИ МАССИВОВ

Обозначение массива представляет собой скрытую форму использования указателей. Например, имя массива определяет также его первый элемент, т.е., если f[ ] – массив, то f=&f[0]. Обе части определяют адрес 1-го элемента массива. Оба обозначения являются константами типа указатель, поскольку они не изменяются на протяжении всей программы. Однако их можно присваивать переменной типа указатель и изменять значения переменной.

Пример.

```
#include <iostream.h>
void main()
{
    int d[4]={1,2,3,4};int *p,i,g; float b[4]={1,2,3,4}; float *p1;
p=d;g=*(p+2);
p1=b;i=*(p1+1);
cout<<g<<" "<<i;
}
```

Результат 3 2.

Имеем следующие равенства:

d+2==&d[2] – один и тот же адрес.

*(d+2)==d[2] – одно и то же значение.

Следует различать *(d+2) – значение 3-го элемента массива, и (*d+2) – к значению 1-го элемента массива добавляется 2.

Операторы

```
int y[ ];
int *y; - синонимы.
```

Пример 1. Вычислить среднее арифметическое элементов массива с помощью указателей.

```
#include <iostream>
using namespace std;
void main()
{
    double d[ ]={1,2,3,4},r,*p,i,n;
    float s=0,sr;
    n=sizeof d/(sizeof(int));
    p=d;
    for(i=0;i<n;i++)
            s+=*(p+i);
    sr=s/n;
    cout<<sr<<endl;
system("pause");
}
```

Пример 2. Передать в функцию одномерный массив с помощью указателей. Вычислить среднее арифметическое элементов массива.

```
#include <iostream>
using namespace std;
double sum(int *p, int n);
void main()
{
    int d[ ]={1,2,3,4}; double r;
r=sum(d,4);
cout<<r<<endl;
system("pause");
}
double sum(int *p, int n)
{
    int i; double s,sr;
    if(n>0)
    {
            for(i=0,s=0;i<n;i++)
                    s+=*(p+i);
sr=s/n;
            return(sr);}
}
```

Пример 3. Найти минимальный и максимальный элементы в массиве. Выполнить с помощью функции и указателей.

```
#include <iostream>
using namespace std;
void minmax(int *p, int *min, int *max, int n);
void main()
{
    int d[]={1,2,3,4},*p,n;
    int min,max;
    n=sizeof d/(sizeof(int));
    p=d;
    minmax(d,&min,&max,n);
    cout<<min<<endl<<max<<endl;
system("pause");
}
void minmax(int *p,int *min,int *max,int n)
{
    int i;
    *min=*p;*max=*p;
    for(i=0;i<n;i++)
            if(*(p+i)<*min)
                    *min=*(p+i);
            else
                    if(*(p+i)>*max)
                            *max=*(p+i);

}
```

Таким образом, если в функцию передается указатель на одномерный массив, то в самой функции его можно объявить одним из трех вариантов: как указатель, как массив определенного размера и как массив без определенного размера. Например, чтобы функция func1() получила доступ к значениям, хранящимся в массиве i, она может быть объявлена как:

```
void func1(int *x) /* указатель */
```

или как

```
void func1(int x[10]) /* массив определенного размера */
```

и наконец, как

```
void func1(int x[ ]) /* массив без определенного размера */.
```

Эти три объявления тождественны, потому что каждое из них сообщает компилятору одно и то же: в функцию будет передан указатель на переменную целого типа. В первом объявлении используется указатель, во втором – стандартное объявление массива. В последнем примере измененная форма объявления массива сообщает компилятору, что в функцию будет передан массив неопределенной длины. Как видно, длина массива не имеет для функции никакого значения, потому что в C++ проверка границ массива не выполняется.

Пример 4. Сгенерировать элементы одномерного динамического массива случайным образом и напечатать их.

```
#include <iostream>
 #include<stdlib.h>
 #include<time.h>
using namespace std;
  int main()
{ setlocale(LC_ALL,"rus");
int *arr;
int size;
srand(time(NULL));
cout<<"Введите размер массива: ";
cin>>size;
arr = new int[size]; // память динамически  выделяется.
cout<<"Сгенерированный массив: ";
 for(int i=0; i<size; i++)
{
 arr[i]=rand() % 10;
  cout<<arr[i]<<" ";
```

```
 }
   cout<<"\n";
delete[ ]arr; // память освобождается
system(“pause”);
return 0;
 }
```

new – оператор языка программирования C++, обеспечивающий выделение динамической памяти. Оператор new пытается выделить достаточно памяти для размещения новых данных и, в случае успеха, возвращает адрес свежевыделенной памяти.

p_var = new type [size].

В данном случае, size указывает размерность (длину) создаваемого одномерного массива. Адрес первого элемента возвращается и помещается в p_var, поэтому p_var[n] означает значение n-го элемента (считая от нулевой позиции).

Любая динамическая память выделенная при помощи new должна освобождаться при помощи оператора delete. Существует два варианта: один для массивов, другой – для единичных объектов.

```
int *p_var = new int;
int *p_array = new int [50];
delete p_var;
delete[] p_array;
```

Пример 5. Передать в функцию динамический одномерный массив с помощью указателей и вычислить среднее арифметическое элементов массива.

```
#include <iostream>
using namespace std;
double  s_arr(double *a, int n)
{
     double summ=0,sr ;
  for (int i=0; i<n; i++)
          summ+=a[i];
sr=summ/n;
    return (sr);
```

```
}
int main()
{      int N;
   cin>>N;
double sr;
   double *arr = new (double [N]);
 for (int i=0; i<N; i++)
    cin>>arr[i];
  sr= s_arr(arr,N);
          cout<<sr<<endl;
system(“pause”);
}
```

Задания

1. Вычислить среднее арифметическое элементов одномерного массива.

2. Переписать элементы одномерного массива в обратном порядке. Выполнить с помощью указателей.

3. Перемножить элементы одномерного массива. Вычислить с помощью указателей.

4. Объединить 2 одномерных массива в один с помощью указателей и функции.

5. Переписать сначала отрицательные элементы массива, затем положительные, с помощью указателей и функции.

6. Вычислить количество ненулевых элементов одномерного массива.

7. Сложить 2 одномерных массива.

8. Найти минимальный и максимальный элемент в одномерном массиве.

Тема № 13

УКАЗАТЕЛИ НА ДВУМЕРНЫЕ МАССИВЫ. ДВУМЕРНЫЕ ДИНАМИЧЕСКИЕ МАССИВЫ

Указатели и многомерные массивы.
Предположим, что есть описания
int z[4][2]; int *p;
Тогда,
p==&z[0][0] – указывает на 1-й элемент 1-й строки.
p+1==&z[0][1] – 1-я строка, 2-й столбец.
p+2 ==&z[1][0]
p+3 ==&z[1][1]

```
#include <iostream>
using namespace std;
void main()
{
    int d[2][2]={ {1,2},{3,4} };
    int *p,*p1,g,g1;
p=&d[0][0]; p1=p+1;
g=*p;
g1=*p1;
cout<<g<<g1;
}
```

Пример 1. Передать в функцию 2 двумерных динамических массива с помощью указателей. Выполнить сложение двух массивов, результат записать в 3-й.

```
#include <iostream>
using namespace std;
int func(int **a, int **b, int **c, int m, int n)
{
int i,j;
```

```
for(i=0;i<m;i++)
for(j=0;j<n;j++)
c[i][j]=a[i][j]+b[i][j];
return 0;
}
int main()
{int i,j,m,n;
cin>>m>>n;
int **a=new int *[m];
for(i=0;i<m;i++)
a[i]=new int[n];
for(i=0;i<m;i++)
for(j=0;j<n;j++)
cin>>a[i][j];
int **b=new int *[m];
for(i=0;i<m;i++)
b[i]=new int[n];
for(i=0;i<m;i++)
for(j=0;j<n;j++)
cin>>b[i][j];
int **c=new int*[n];
for(i=0;i<m;i++)
c[i]=new int[n];
func(a,b,c,m,n);
for(i=0;i<m;i++)
for(j=0;j<n;j++)
cout<<c[i][j]<<" ";
cout<<endl;
}
for(i=0;i<m;i++)
{delete [] a[i];
delete [] b[i];
delete [] c[i];}
delete []a;
delete[]b;
delete[]c;
system("pause");
}
```

```
Освобождение памяти
for (int i = 0; i < x; i++)
 {   delete []mas[i]; }
 delete []mas;
```

Задания

1. Сложить элементы двухмерного массива. Выполнить с помощью указателей.

2. Сложить элементы 2-х мерных массивов с помощью указателей.

3. Найти минимальный и максимальный элемент в 2-х мерном массиве с помощью указателей.

4. Переписать элементы 2-х мерного массива в одномерный с помощью указателей.

5. Вычислить сумму элементов двумерного массива по столбцам. Результат записать в одномерный массив (с помощью функции и указателей).

6. Вычислить сумму элементов двумерного массива по строкам. Результат записать в одномерный массив (с помощью функции и указателей).

7. Перемножить элементы двумерного массива.

8. Перемножить 2 двумерных массива.

9. Подсчитать количество неотрицательных элементов двумерного массива.

Тема № 14

ФАЙЛЫ

Заголовочный файл: <stdio.h>

fopen

Синтаксис:

FILE* fopen(const char * *Имя,* const char* *Режим)*

открывает файл с указанным именем для действия, которое задается параметром *Режим.*

r – только чтение, файл открывается только для чтения.

w – запись, файл открывается для записи. Если файл с указанным в качестве первого параметра функции fopen уже существует, то новые данные записываются поверх старых, т. е. старый файл фактически уничтожается.

A – добавление, файл открывается для записи данных в конец существующего файла. Если файл с указанным в качестве первого параметра функции fopen не существует, то он будет создан.

fprintf

Синтаксис:

int fprintf(FILE * *Поток, Формат, Список Переменых)*;

выполняет форматированный вывод в файл, связанный с потоком, указанным в качестве первого параметра. Файл, связанный с потоком, должен быть открыт как текстовый, в режиме, допускающем запись.

Заголовочный файл: <stdio.h>

Возвращаемое значение

В случае успеха, возвращается общее число записанных символов. В случае неудачи, возвращается отрицательное число.

Пример 1.

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

```
 using namespace std;
int main ()
{
  FILE * pFile;
  int n;
  char name [100];
pFile = fopen ("C:\\myfile.txt","w");
for (n=0 ; n<3 ; n++)
  {
   puts ("please, enter a name: ");
   gets (name);
   fprintf (pFile, "Name %d [%-10.10s]\n", n,name);
  }
 fclose (pFile);
 system("pause");
return 0;
}
```

В этом примере программа 3 раза запрашивает имя пользователя, а затем записывает их в файл myfile.txt каждый из них в соответствии с фиксированной длиной (всего 19 символов + строка).

Используется два тега форматирования:

%D: десятичное число

%-10.10s: выровненное влево (-), не менее десяти символов (10), максимум десять символов (0,10), String (s).

Предположим, что мы ввели, Vasja, Kolja и Petja. Итак myfile.txt должен содержать 3 имя:

Имя 0 [Vasja]

Имя 1 [Kolja]

Имя 2 [Petja]

fscanf

Синтаксис:

int fscanf(FILE **Поток,***__**const char***__ *Формат, Список Адр)*;

выполняет форматированное чтение значений переменных из файла, связанного с потоком, указанным в качестве первого параметра.

Файл, связанный с потоком, должен быть открыт как текстовый, в режиме, допускающем чтение.

Заголовочный файл: <stdio.h>

fgets

Синтаксис:

char* fgets(char **Строка,* int *Кол Символов,* FILE **Поток)*

читает из указанного потока символы и записывает их в строку, указанную при вызове функции. Чтение заканчивается, если прочитан символ с номером *КолСимволов-1* или если очередной символ является символом новой строки.

Прочитанный из файла символ новой строки заменяется нулевым символом. Файл, связанный с потоком, должен быть открыт как текстовый, в режиме, допускающем чтение (см. f open).

Заголовочный файл: <stdio.h>

fputs

Синтаксис:

char* fputs(char **Строка,* FILE **Поток)*

записывает в указанный поток строку символов. Символ конца строки, нуль-символ, в поток не записывается. Файл, связанный с потоком, должен быть открыт как текстовый, в режиме, допускающем запись (см. fopen).

Заголовочный файл: <stdio.h>

ferror

Синтаксис:

int ferror(FILE* *Поток)*

возвращает ненулевое значение, если последняя операция с указанным потоком завершилась ошибкой.

Заголовочный файл: <stdio.h>

feof

Синтаксис:

int feof(FILE* *Поток)*

возвращает ненулевое значение, если в результате выполнения последней операции чтения из потока достигнут конец файла.

Заголовочный файл: <stdio.h>

fclose

Синтаксис:

int fclose(FILE* *Поток)*

закрывает указанный поток.
Заголовочный файл: <stdio.h>

Пример 2.

Напишите программу, которая на жестком диске компьютера (диск C:) создает файл numbers.txt и записывает в него 5 введенных пользователем целых чисел. Просмотрите при помощи редактора текста созданный файл. Убедитесь, что каждое число находится в отдельной строке.

```
// Создает на диске файл
#include "stdio.h"
#include "conio.h“
#include <iostream>
#define FNAME “C:\\numbers.txt\0"
// имя файла
#define N 5
// количество чисел
// Создает на диске C: файл и записывает в него целые числа, введенные //пользователем
using namespace std;
void main() {
setlocale(LC_CTYPE,”Rus”);
char fname[20] = FNAME;
puts("После ввода каждого числа нажимайте <Enter>\n");
FILE *out;  // файл чисел
int n;     // число
puts("Создание файла\n");
printf("Введенные числа будут записаны в файл");
printf("%s\n", fname);
// Открыть файл в режиме записи (w) текста (t)
// Если файл с таким именем уже есть, то новые
// данные будут записаны поверх старых
// Для дозаписи в конец файла используйте
// режим добавления (a)
if ((out = fopen(fname, "wt")) == NULL)
{
printf("Ошибка открытия файла для записи");
getch();
```

```
return; }
for (int i = 0; i < N; i++) {
printf("->");
scanf("%i", &n);
fprintf(out,"%i\n",n); }
fclose(out);    // закрыть файл
printf("Введенные числа записаны в файл %s\n", fname);
puts("\n, Для завершения нажмите <Enter>");
 getch();
}
```

Пример 3.

```
#include "stdio.h"
 #include "conio.h"
#include <iostream>
#define FNAME "C:\\numbers.txt\0" // имя файла
using namespace std;
void main()
{
setlocale(LC_CTYPE,"Rus");
char fname[20] = FNAME;
FILE *in;    // текстовый файл
int a; // число
int n = 0; // количество чисел
int sum =0; // сумма чисел
float sr; // среднее арифметическое
puts("вычисление среднего арифметического");
printf("чисел, находящихся в файле  %s\n", fname);
// Открыть файл в режиме чтения (r) текста (t)
if ((in =fopen(fname, "rt")) == NULL)
{
printf("Ошибка открытия файла для чтения\n");
getch();
return; }
while (!feof(in))
{
fscanf(in,"%i", &a);
sum += a;
n++;
}
```

```
fclose(in);    // закрыть файл
sr = (float) sum / n;
printf("введено чисел: %i\n", n);
 printf("сумма чисел: %i\n", sum);
 printf("сред. арифм.: %3.2f\n", sr);
puts("Для завершения нажмите <Enter>\n");
getch();
}
```

Задания

1. Напишите программу, которая на жестком диске компьютера (диск D:) создает файл numbers.txt и записывает в него 5 введенных пользователем целых чисел. Просмотрите при помощи редактора текста созданный файл. Убедитесь, что каждое число находится в отдельной строке.

2. Напишите программу, которая дописывает в файл D:\numbers.txt пять введенных пользователем целых чисел. Убедитесь при помощи редактора текста, что в файле находятся 10 чисел.

3. Напишите программу, которая выводит на экран содержимое файла D:\numbers.txt.

4. Напишите программу, которая вычисляет среднее арифметическое чисел, находящихся в файле D:\numbers.txt.

5. Напишите программу, которая дописывает в находящийся на диске D: файл phone.txt имя, фамилию и номер телефона, например, вашего товарища. Если файла на диске нет, то программа должна создать его. В файле каждый элемент данных (имя, фамилия, телефон) должен находиться в отдельной строке. Рекомендуемый вид экрана во время работы программы приведен ниже.

Добавление в телефонный справочник
Фамилия -> **Сидоров**
Имя **-> Вася**
Телефон -> **234-84-37**
Информация добавлена.
Для завершения работы нажмите <Enter>

6. Напишите программу, которая позволяет за один сеанс работы добавить информацию о нескольких людях в файл D:\phone.txt.

Рекомендуемый вид экрана во время работы программы приведен ниже.

Добавление в телефонный справочник.

Для завершения вместо ввода фамилии нажмите <Enter>

Фамилия -> **Сидоров**

Имя **-> Вася**

Телефон -> **234-84-37**

Информация добавлена.

Фамилия -> **Орлов**

Имя **-> Андрей**

Телефон -> **552-18-40**

Информация добавлена.

Фамилия ->

Ввод завершен

Для завершения работы нажмите <Enter>

7. Напишите программу, которая позволяет вывести данные из файла D:\phone.txt. Программа должна подсчитать и вывести количество записей в файле.

8. Напишите программу, которая позволяет найти в телефонном справочнике (D:\phone.txt) нужные сведения. Программа должна запрашивать фамилию человека и выводить его телефон. Если в справочнике есть люди с одинаковыми фамилиями, то программа должна вывести список всех этих людей. Рекомендуемый вид экрана во время работы программы приведен ниже.

Поиск в телефонном справочнике.

Введите фамилию и нажмите <Enter>. Для завершения работы с

программой сразу после приглашения нажмите <Enter>

-> **Петров**

В справочнике данных о Петров нет.

-> **Иванов**

Иванов Вася 578-12-45

Иванов Сергей 244-34-02

->

Тема № 15

СТРОКИ

Символ – элементарная единица, некоторый набор которых несет определенный смысл. В языке программирования С++ предусмотрено использование символьных констант.

Объявление символьной переменной

char symbol = 'a'; где symbol – имя переменной типа char, char – тип данных для хранения символов.

Символьные строки состоят из набора символьных констант заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-символа, и отводить дополнительный байт под него.

Пример объявления строки char c[10]; где c – имя строковой переменной, 10 – размер массива, то есть в данной строке может поместиться 9 символов, последнее место отводится под нуль-символ. Строка при объявлении может быть инициализирована начальным значением, например, так:

char c[10] = “privetik1”.

Если подсчитать количество символов в двойных кавычках после символа равно их окажется 9, а размер строки 10 символов, последнее место отводится под нуль-символ, причём компилятор сам добавит его в конец строки.

Посимвольная инициализация строки:

char c[10] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'f', '\0'}; десятый символ это нуль-символ. При объявлении строки не обязательно указывать её размер, но при этом обязательно нужно её инициализировать начальным значением. Тогда размер строки определится автоматически и в конец строки добавится нуль-символ.

Инициализация строки без указания размера:

char c[ ] = “privetik1";

всё то же самое только размер не указываем. Строка может содержать символы, цифры и специальные знаки. В С++ строки заключаются в двойные кавычки. Имя строки является константным указателем на первый символ. Разработаем программу, с использованием строк.

Пример 1.

```
#include <iostream>
using namespace std;
int main()
{
char st[ ] = "this is string - ";
// объявление и инициализация строки
 cout << "Enter the st: ";
 char in_string[500]; // строковый массив для ввода
gets(in_string); // функция gets() считывает все введённые символы с //пробелами до тех пор, пока не будет нажата клавиша Enter
cout << st << in_string << endl; // вывод строкового значения
 system("pause");
 return 0;
}
```

С помощью функции gets() считаются все введённые символы с пробелами до тех пор, пока во вводимом потоке не встретится код клавиши enter. Если использовать операцию cin, то из всего введённого считается последовательность символов до первого пробела.

Пример 2. Написать программу, которая проверяет, является ли введенная с клавиатуры строка целым числом. Рекомендуемый вид экрана во время выполнения программы приведен ниже (данные, введенные пользователем, выделены полужирным шрифтом).

Введите число и нажмите <Enter> -> **23.5**

Введенная строка не является целым числом.

```
#include <stdio.h>
#include <conio.h>
```

```
void main()
{
char st [40]; int i; printf("->");
 scanf("%s",&st);
i = 0;
while (st[i] >= '0' && st[i] <= '9') i++;
if(st[i]=='\0')
printf("javl");
else
printf(" ne javl");}
```

каждому символу соответствует число – код символа.

В С++ строка – это массив символов, последним символом строки обязательно должен быть нуль-символ, код которого равен 0, и который в тексте программы изображается так: '\0'; сообщения или подсказки, используемые в программе, удобно представить как массив указателей на строки и инициализировать массив, задать сообщения в инструкции объявления массива:

char *mes[] ={"Сообщение 1","Сообщение 2", ... , "«Сообщение"};

Если вводимая во время работы программы строка содержит пробелы, то функция scanf вводит только часть строки до первого пробела, а функция gets – всю строку, в том числе и соответствующий клавише <Enter> символ ' \n'.

Функции для работы со строками и символами

Синтаксис:

char *strcat**(char*** *Строка1,* **const char*** *Строка2)*

объединяет строки *Строка1* и *Строка2* и записывает результат в строку *Строка 1.*

Заголовочный файл: <string.h>

strncat(s1,s2,n)

объединяет n символов строки s2 со строкой s1. Результат сохраняется в s1.

strcpy

Синтаксис:

char *strcpy**(char*** *Строка 1,* **const char*** *Строка2)* – копирует строку *Строка2* в строку *Строка1.*

Заголовочный файл: <string.h>

strncpy(s1,s2, n**)** выполняет побайтное копирование n символов из строки s2 в строку s1, возвращает значение s1.

```
int main()
{ char *str = "образец строки";
char buf[32];
// очистка буфера для вывода.
 *buf = '\0';
printf("строка: \"%s\"\n", str);
printf("буфер перед копированием: \"%s\"\n", buf);
strcpy(buf, str);
printf("буфер после копирования: \"%s\"\n", buf); return 0; }
```

Вывод:

строка: "образец строки"

буфер перед копированием: “ “

буфер после копирования: "образец строки"

strlen

Синтаксис:

int strlen**(const char*** *Строка)*

Возвращает длину строки. Нулевой символ не учитывается. Заголовочный файл: <string.h>

strcmp

Синтаксис:

int strcmp **(const char*** Строка1, **const char*** Строка2)

сравнивает строки *Строка 1* и *Строка2*. Возвращает 0, если строки равны, число меньше нуля, если *Строка 1* < *Строка2* и число больше нуля, если *Строка 1* > *Строка2*.

Заголовочный файл: <string.h>

strncmp(s1,s2,n)

сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 – если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 с учётом регистра.

stricmp(s1,s2**)** сравнивает строку s1 со строкой s2 и возвращает результат типа **int**: 0 – если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 без учёта регистра.

strnicmp(s1,s2,n**)** сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 – если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 без учёта регистра.

Пример 3.

```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdio.h>
void main()
{
    char st[15],st1[15];
gets(st);
gets(st1);
            int s;
s=strcmp(st,st1);
    cout<<s<<endl;
system("pause");
}
```

strlwr

Синтаксис:

char* strlwr**(char*** *Строка)*

преобразует строчные символы строки в прописные (обрабатывает только буквы латинского алфавита).

Заголовочный файл: <string.h>.

strupr

Синтаксис:

char* strupr(char* *Строка)*

преобразует прописные символы строки в строчные (обрабатывает только буквы латинского алфавита).

Заголовочный файл: <string.h>

Пример 4.

```
#include <iostream>
#include <string.h>
#include <stdio.h>
void main()
{
    char st[15];
gets(st);
```

```
            int d,v;
    d=strlen(st);
    cout<<d<<endl;
strupr(st);
    cout<<st<<endl;
}
```

strset

Синтаксис:

char* strset**(char*** *Строка,* **char** *Символ)*

заполняет строку указанным при вызове функции символом. Заголовочный файл: <string.h>.

strchr

Пример 5. Дана символьная строка, не содержащая пробелов. Заменить все символы '+', расположенные за первой буквой 'A', на символы '*'.

Возможный вариант реализации:

```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdio.h>
int main()
{
char s[30];
cout << "Input s:" << endl;
 cin >> s;
char *u = strchr(s,'A');
 if(u)
  {
    u++;
     for(int i=0; i < strlen(u); i++)
   if(u[i] == '+')
          u[i] = '*';
 }
 cout <<"s=" << s << endl;
  return 0;
}
```

После того как найден символ A в строке s, указатель перейдет на следующий символ, длина строки считается, начиная с него.

strcspn(s1,s2) определяет длину начального сегмента строки s1, содержащего те символы, которые не входят в строку s2.

strspn(s1,s2) возвращает длину начального сегмента строки s1, содержащего только те символы, которые входят в строку s2.

Функции стандартной библиотеки ввода/вывода <stdio>

getchar(c) считывает символ со стандартного потока ввода, возвращает символ в формате int.

gets(s) считывает поток символов со стандартного устройства ввода в строку s до тех пор, пока не будет нажата клавиша ENTER.

Пример 6.

```
#include <iostream>
using namespace std;
int main( )
{
   char s2[27] = "Counter-Strike 1.6 forever";                    // инициализация строки s2
   char s1[27];                                                   // резервируем строку для функции strcpy();
 cout << "strcpy(s1,s2) = " << strcpy(s1,s2) << endl;
// содержимое строки s2 скопировалось в строку s1, возвращается указатель //на s1
cout << "s1=                " << s1                << endl; // вывод содержимого строки s1
 char s3[7];
// резервируем строку для следующей функции
 cout << strncpy(s3, s2, 7) << endl;
 // копируем 7 символов из строки s2 в строку s3
 system("pause");
   return 0;
}
```

Словосочетание "Counter-Strike 1.6 forever”

содержит 26 символов, последнее место в массиве займет нуль-символ. Функция strcpy(s1,s2) копирует значение строки s2 в строку s1 и возвращает указатель на строку s1. Если строка s1 будет меньше строки s2, то скопируется то количество символов, которое вместится в строку s1. В строке s1 содержится скопированное значение. Функция strncpy (s3, s2, sizeof(s3)) выполняет копирование 7 символов строки s2 в строку s3.

Результат будет выглядеть следующим образом:

strcpy(s1,s2) = Counter-Strike 1.6 forever s1= Counter-Strike 1.6 forever CounterCounter-Strike 1.6 forever

Для продолжения нажмите любую клавишу . . .

Конкатенация строк

Использование функций strcat() и strncat(), для объединения строк, то есть для их конкатенации.

Пример 7.

```
#include <iostream>
using namespace std;
int main()
{    char s1[30] = "I am ";
 char s2[] = "programmer on the C++!!!!";
cout << strcat(s1,s2) << endl;
 // объединяем строки s1 и s2, результат сохраняется в строке s1
char s3[23] = "I am a good ";
 cout << strncat(s3,s2,10) <<  "!!!" << endl;
 // объединяем 10 символов строки s2 со строкой s3
system("pause");
return 0;}
```

```
I am programmer on the C++!!!!
 I am a good programmer!!!
Для продолжения нажмите любую клавишу . . .
```

Функция strcat(s1,s2) объединяет строки s1 и s2, результат сохраняется в строке s1. Поэтому при объявлении строки s1 её размер установлен на 30 символов. Функция **strncat(s3,s2,10)** объединяет 10 символов из строки s2 (как раз помещается слово programmer) со строкой s3, результат сохраняется в строке s3.

```
char s[6];
int i;
for(i = 0; i < 4; i++)
s[i] = i + 'A';
s[i]='\0';
```

Заполняем s символьной строкой "ABCD". После выхода из цикла переменная i равна 4. В эту позицию и записываем нуль-символ.

Пример 8. Дана символьная строка. Подсчитать количество слов, из которых состоит эта строка. Под понятием «**слово**» будем подразумевать последовательность любых вводимых с клавиатуры (или из текстового файла) символов, разделённых одним или несколькими пробелами.

Возможный вариант реализации:

```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdio.h>
int main()
{
char s[80];
cout << "Input s:" << endl;
gets(s);
 int i, d = strlen(s), k = 0;
   if(d > 0)
 {
   for(i = 0; i < d - 1; i++)
      if(s[i] != ' ' && s[i+1] == ' ')
        k++;
      if(s[d-1] != ' ')
         k++;
   }
cout <<"k=" << k << endl;
return 0;
}
```

Пример 9. Подсчитать длину строки без использования функции strlen.

```
#include <iostream>
using namespace std;
 int main()
{setlocale(LC_ALL, "rus");
char St[128] = "";
cout << "Введите текст латиницей (не больше 128 символов):\n";
cin.getline(St, 128); int a = 0;
while (St[a] != '\0'){a++;}
cout << "Строка" << St<< состоит из "<< a << " символов! <<endl;
return 0;
}
```

Задания

1. Написать программу, которая запрашивает имя пользователяи здоровается с ним. Рекомендуемый вид экрана во время выполнения программы приведен ниже.

Как Вас зовут?

Введите свои имя и фамилию, затем нажмите <Enter>

-> **Вася Иванов**

Здравствуйте, Вася Иванов!

2. Напишите программу, которая вычисляет длину введенной с клавиатуры строки.

3. Скопировать в 1-ю неинициализированную строку, 2-ю строку, заданную в самой программе). Вывести ее. Затем заданную строку скопировать в первую строку. Вывести ее.

4. Определить какая строка из 2-х длиннее.

5. Определить совпадают ли строки по длине. Строки задать в программе.

6. Напишите программу, которая выводит код введенного пользователем символа. Программа должна завершать работу в результате ввода, например, точки. Рекомендуемый вид экрана во время выполнения программы приведен ниже.

Введите символ и нажмите <Enter>.

Для завершения введите точку.

-> 1

Символ: 1 Код: 4 9
-> 2
Символ: 2 Код: 50
-> ы
Символ: ы Код: 235
->.

7. Написать программу, которая в введенной с клавиатуры строке преобразует строчные буквы русского алфавита в прописные Рекомендуемый вид экрана во время выполнения программы приведен ниже. Введите строку текста и нажмите <Enter>

-> изучив основы C++,
можно начать программировать под Windows

Строка, преобразованная к верхнему регистру:
ИЗУЧИВ ОСНОВЫ C++, МОЖНО НАЧАТЬ
ПРОГРАММИРОВАТЬ ПОД WINDOWS

8. Написать программу, которая удаляет из введенной с клавиатуры строки начальные пробелы.

9. Написать программу, которая проверяет, является ли введенная с клавиатуры строка целым числом. Рекомендуемый вид экрана во время выполнения программы приведен ниже.

Введите число и нажмите <Enter> -> **23.5**
Введенная строка не является целым числом.

10. Написать программу, которая проверяет, является ли введенная с клавиатуры строка дробным числом.

11. Получите от пользователя: год рождения, место рождения и возраст. Затем напечатать предложение, использующее эту информацию.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шилдт Г. Теория и практика С++. – СПб.: BHV, 1996. – 416 с.
2. Уэйт М., Прата С., Мартин Д. Язык Си. Руководство для начинающих. –М.: Мир,1988. – 512 с.
3. Культин Н.Б. С,С++ в задачах и примерах. – СПб.: БХВ-Петербург, 2001. – 288 с.
4. Кетков А., Кетков Ю. Практика программирования: Бейсик, Си, Паскаль. Самоучитель. – СПб.: БХВ-Петербург, 2001. – 480 с.
5. Абрамов С.А., Трифонов К.К. Задачи по программированию. – М.: Наука, 1988.– 320 с.
6. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1977. –401 с.
7. Кнут Д. Искусство программирования для ЭВМ. – М.: Мир, 1976-79. – Т 1. 734 с.
8. Кнут Д. Искусство программирования для ЭВМ. – М.: Мир, 1976-79. – Т 2. 834 с.
9. Кнут Д. Искусство программирования для ЭВМ. – М.: Мир, 1976-79. – Т 3. 834 с.
10. Дал У., Дейкстра Э., Хоор К. Структурное программирование. – М.: Мир, 1975. – 248 с.
11. Абрамов С.А., Гнездилова Г.Г., Капустина, Селюн М.И. Задачи по программированию. – М.: Наука, 1988.– 224 с.
12. Климова Л.М. Основы практического программирования на языке С++. – М.: «Изд. Приор», 1999. – 464 с.
13. Франка Л. С++. Учебный курс. – СПБ.: Питер,2003. – 521 с.
14. Крис Паппас, Уильям Мюррей VisualC++6. Руководство разработчика. BHV, 2000. – 624 с.

ОГЛАВЛЕНИЕ

Учебное издание

Б.А. Урмашев
Т.А. Шмыгалева

**ПРОГРАММИРОВАНИЕ
НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ С++**

*Учебное пособие для студентов
Специальностей «Информатика»,
«Вычислительная техника и программное обеспечение»*

Редактор *З. Усенова*
Компьютерная верстка *Г. Калиевой*
Дизайн обложки *Р. Скакова*

В оформлении обложки использованы фотографии с сайтов
https://wallpapercave.com/wp/wp2921838.jpg

ИБ №
Подписано в печать 04.07.2020. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Печать цифровая. Объем п.л. Тираж экз. Заказ №.
Издательский дом «Қазақ университеті»
Казахского национального университета им. аль-Фараби.
050040, г. Алматы, пр. аль-Фараби, 71.

Отпечатано в типографии издательского дома «Қазақ университеті».