

**«Разработка, отладка и испытание алгоритмов и программ с использованием функций.  
Использование указателей и массивов в качестве параметров»**

Функция – изолированный именованный блок кода, имеющий определенное назначение.

Информация в функцию передается с помощью аргументов (фактических параметров), задаваемых при ее вызове в функции main(). Эти аргументы должны соответствовать формальным параметрам, указанным в описании функции.

Синтаксис описания функций

типВозвращаемогоРезультата имяФункции (списокВходныхПараметров) // заголовок функции

```
{
    операторы;           // тело функции (код тела функции)
    return возвращаемыйРезультат;
}
```

Любая программа на C++ состоит из функций, одна из которых должна иметь имя main (с ее первой строки начинается выполнение программы и ее последней строкой (где написано выражение с return) заканчивается выполнение программы).

Функция начинает выполняться в момент вызова.

Любая функция должна быть объявлена и определена.

### **Объявление и определение функций**

Как и для других величин, объявлений может быть несколько, а определение только одно. Объявление функции (это прототип функции) должно находиться в тексте раньше ее вызова, для того, чтобы компилятор мог осуществить проверку правильности вызова.

Объявление функции (ее прототип, состоящий из заголовка (имени) функции и ее сигнатуры (списка входных параметров функции и типа возвращаемого ею результата) задает ее имя, тип возвращаемого значения и список передаваемых в функцию параметров. Определение функции содержит, кроме объявления, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

Рассмотрим составные части определения.

С помощью необязательного модификатора можно задать область видимости функции, используя ключевые слова extern и static.

Тип возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не должна возвращать значение, указывается тип void.

Список входных параметров определяет величины, которые требуется передать в функцию при ее вызове. Элементы списка параметров разделяются запятыми. Для каждого параметра, передаваемого в функцию, указывается его тип и имя (в объявлении (прототипе) имена можно опускать).

Пример 1 функции, возвращающей сумму двух целых величин:

```
#include <iostream>
```

```
using namespace std;
```

```
int sum(int, int); // прототип функции sum()
```

```
int main()
```

```
{
    int a=2, b=3, c, d;
    c = sum(a, b);           //вызов функции sum() с фактическими параметрами
    cin >> d;
    cout << sum(c, d) << endl; //вызов функции sum() с фактическими параметрами
    system("pause");
    return 0;
}
```

```
int sum(int a, int b) // определение функции sum()
```

```
{
    return (a + b);
}
```

Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции (той, которая вызвала нашу пользовательскую функцию).

При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции НЕ сохраняются. Если этого требуется избежать, при объявлении локальных переменных используется модификатор static.

```
#include <iostream>
```

```

using namespace std;
void f(int); // прототип функции

int main()
{
    f(3); // вызов функции с фактическими параметрами
    f(2); // вызов функции с фактическими параметрами
    system("pause");
    return 0;
}

void f(int a) // определение функции с ее телом
{
    int m = 0;
    cout << "n m p\n";
    while(a--)
    {
        static int n = 0;
        int p = 0;
        cout << n++ << ' ' << m++ << ' ' << p++ << '\n';
    }
}

```

Статическая переменная `n` размещается в сегменте данных и инициализируется один раз при первом выполнении оператора, содержащего ее определение. Автоматическая переменная `m` инициализируется при каждом входе в функцию. Автоматическая переменная `p` инициализируется при каждом входе в блок цикла.

Программа выведет на экран:

```

n m p
0 0 0
1 1 0
2 2 0,
n m p
3 0 0
4 1 0

```

При совместной работе функции должны обмениваться информацией. Это можно осуществить с помощью глобальных переменных, через параметры и через возвращаемое функцией значение.

### Возвращаемое значение

Механизм возврата из функции в вызвавшую ее функцию реализуется оператором

`return` выражение;

Примеры:

```

int f1( )
{
    return 1; // правильно, 1 – целочисленное значение и выше заявлено, что функция обязана вернуть
              // целочисленное значение
}

void f2( )
{
    return 1; // неправильно, 1 – целочисленное значение, а выше заявлено, что функция обязана вернуть «ничего»
}

double f3( )
{
    return 1; // правильно, поскольку 1 неявно преобразуется к типу double как более «емкому» типу данных
}

int* f( )
{
    int a = 5;
    return &a; // нельзя, поскольку a – локальная переменная в стеке, она будет удалена при окончании работы
              // данной функции, стало быть по этому адресу ее уже не найти будет (хоть синтаксис верный)
}

```

### Входные параметры функции

При передаче в функцию входных параметров по значению в стек заносятся **копии** значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и

возможности их изменить.

При передаче в функцию входных параметров по адресу в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и **может изменить исходные значения** аргументов:

```
#include <iostream>
```

```
using namespace std;
```

```
void f(int, int*, int&);//прототип функции, принимающей значение, указатель (то есть адрес) и ссылку (то есть адрес)
```

```
int main( )
```

```
{
```

```
    int i = 1, j = 2, k = 3;
```

```
    cout << "i j k \n";
```

```
    cout << i << ' ' << j << ' ' << k << '\n';
```

```
    f(i, &j, k);//вызов функции с фактическими параметрами
```

```
    cout << i << ' ' << j << ' ' << k;//проверяем, поменялись ли оригиналы значений в main'e
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
void f(int i, int* j, int& k)
```

```
{
```

```
    i++;
```

```
    (*j)++;
```

```
    k++;
```

```
}
```

Результат работы программы:

```
i j k
```

```
1 2 3
```

```
1 3 4
```

Если требуется запретить изменение параметра внутри функции, используется модификатор const:

```
int f(const char* s)//функция принимает s, но изменить его не сможет
```

```
{
```

```
    //
```

```
}
```

```
char* t(char* a, const int* b)//функция принимает b, но изменить его не сможет, но при том сможет изменить a
```

```
{
```

```
    //
```

```
}
```

**Алгоритм разработки функции:**

- Определяется интерфейс функции:
- какие значения подаются ей на вход
- что должно получиться в результате
- продумываются структуры данных (типы данных в функции, способы их обработки)

Решим задачу расчета  $sh\ x = x^1 / 1! + x^3 / 3! + x^5 / 5! + x^7 / 7! + \dots$  с помощью функции:

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 double cosh(double, double); //прототип функции cosh()
5 //sh x = x^1 / 1! + x^3 / 3! + x^5 / 5! + x^7 / 7! + ...
6 int main()
7 {
8     double Xn, Xk, dX, eps;
9     cout << "Enter Xn: "; cin >> Xn; //начальное значение x
10    cout << "Enter Xk: "; cin >> Xk; //конечное значение x
11    cout << "Enter dX: "; cin >> dX; //шаг приращения x
12    cout << "Enter eps: "; cin >> eps; //требуемая точность вычисления sh(x)
13    cout << "\tX\t\t\tY\t\t\n-----+-----\n"; //печать "шапки" таблицы
14    for (double x = Xn; x <= Xk; x += dX)
15    {
16        cout << "\t" << x << "\t\t\t" << cosh(x, eps) << endl; //вызов функции cosh() в потоке cout
17    }
18    cout << "-----+-----\n"; //конец печати таблицы
19    system("pause");
20    return 0;
21 }
22 double cosh(double x, double eps) //определение функции cosh()
23 {
24     const int MaxIter = 500; //зададим максимальное количество итераций для недопущения бесконечного заикливания
25     double ch = 1, y = ch; //первый член ряда и начальное значение суммы
26     for (int n = 0; fabs(ch) > eps; n++) //вычислять, пока значение очередного члена ряда больше требуемой точности
27     {
28         ch *= pow(x, 2) / ((2 * n + 1) * (2 * n + 2)); //вычисляем очередной член ряда по оптимизированной формуле
29         y += ch; //добавление очередного члена ряда к сумме
30         if (n > MaxIter) //проверка: если количество итераций превышает лимит
31         {
32             cout << "Ряд расходится!\n"; //то сообщить об этом пользователю
33             return 0; //и завершить вычисления
34         }
35     }
36     return y; //возвращаем вычисленную сумму для конкретного x
37 }

```

Тестируем:

```

C:\> D:\2019\Labs\64\Debug\Lab4_2.exe
Enter Xn: 0
Enter Xk: 2
Enter dX: 0.1
Enter eps: 0.0000000001

```

X	Y
0	1
0.1	1.005
0.2	1.02007
0.3	1.04534
0.4	1.08107
0.5	1.12763
0.6	1.18547
0.7	1.25517
0.8	1.33743
0.9	1.43309
1	1.54308
1.1	1.66852
1.2	1.81066
1.3	1.97091
1.4	2.1509
1.5	2.35241
1.6	2.57746
1.7	2.82832
1.8	3.10747
1.9	3.41773

Для продолжения нажмите любую клавишу . . .

### Параметры со значениями по умолчанию

В качестве значений параметров по умолчанию могут использоваться константы, глобальные переменные и выражения:

```
int f(int a, int b = 0);
```

```
void fl(int, int = 100, char* = 0); // обратите внимание на пробел между * и = (без него получилась бы операция сложного
присваивания *= (полусокращенная запись))
void err(int errValue = errno); // errno - глобальная переменная, хранящая номер последней произошедшей ошибки
int main()
{
    f(100); // варианты вызова функции f()
    f(a, 1); // варианты вызова функции f()
    fl(a); // варианты вызова функции fl()
    fl(a, 10); // варианты вызова функции fl()
    fl(a, 10, "Vasia"); // варианты вызова функции fl()
    fl(a, , "Vasia"); // неверно; можно пропустить только последний или группу последних входных параметров по
умолчанию, но пропустить один параметр в середине выражения нельзя, т.к. компилятор не сможет правильно
интерпретировать такой набор фактических параметров
}
```

### Функции с переменным числом параметров

Если список формальных параметров функции заканчивается многоточием, это означает, что при ее вызове на этом месте можно указать еще несколько параметров. Проверка соответствия типов для этих параметров не выполняется. char и short передаются как int, float — как double.

### Области видимости и классы памяти (спецификаторы памяти) переменных

Область видимости определяет, из каких частей программы возможен доступ к переменной.

Класс памяти определяет время, в течение которого переменная существует в памяти компьютера.

В C++ существуют 3 типа области видимости переменных:

- 1) локальная область видимости
- 2) область видимости файла
- 3) область видимости класса

Переменные, имеющие локальную область видимости доступны только внутри того блока, в котором они определены (блоком обычно считается код, заключенный в фигурные скобки).

Переменные, имеющие область видимости файла, доступны из любого места файла, в котором они определены.

В C++ существует 3 класса памяти:

- 1) auto (автоматический)
- 2) static (статический)
- 3) dynamic (динамический) (будет рассмотрен позднее)

Автоматическая переменная «рождается» в момент ее объявления и прекращает свое существование в момент завершения выполнения блока, где она определена. Автоматическая переменная не инициализируется автоматически. Если она инициализируется при объявлении, инициализация будет выполняться каждый раз при входе в блок и «рождении» переменной.

У переменных, имеющих класс памяти static, время жизни равно времени жизни всей программы. Статическая переменная по умолчанию инициализируется нулем. Статическая переменная создается и инициализируется один раз – при первом выполнении блока.

Глобальные переменные объявляются вне всех блоков и классов (о последних речь пойдет позже) и имеют область видимости файла. Они доступны всем функциям и блокам, начиная с той точки файла программы, где они объявлены.

Глобальные переменные можно сделать доступными и из других файлов, если программа состоит из нескольких файлов.

По умолчанию глобальные переменные имеют статический класс памяти.

Глобальные переменные живут все время выполнения программы. Если они не инициализируются явно, то по умолчанию глобальные переменные инициализируются нулевым значением.

**По возможности, использования глобальных переменных следует избегать.**

### Область видимости переменной

Пример вывода на экран значений переменной a, которая объявляется в разных областях видимости программы.

```
#include <iostream>
using namespace std;
int a = 11; // a – глобальная переменная
void main()
{
    cout << " 1 a = " << a << endl; // a = 11
    int a = 22; // «местная» локальная переменная «закрывает» собой глобальную переменную с таким же именем
    cout << " 2 a = " << a << endl; // a = 22
    {
        int a = 33; // «местная» локальная переменная «закрывает» собой «более глобальную» переменную с таким
        // же именем
    }
}
```

```

    cout << " 3 a = " << a << endl;//a = 33
}
cout << " 4 a = " << a << endl;//a = 22 при возврате в «большой» блок программы, где видна «местная» локальная
//переменная, а та (33) локальная переменная здесь не существует. Глобальная переменная «закрыта» местной
//локальной одноименной переменной по-прежнему
}

```

### Оператор расширения области видимости ::

Пример тот же, что и выше, но используется оператор расширения видимости (::)

```

#include <iostream>
using namespace std;
int a = 11;// a – глобальная переменная
void main()
{
    cout << " 1 a = " << a << endl;//a = 11 тут видна глобальная переменная
    int a = 22;
    cout << " 2 a = " << a << " " << ::a << endl;//a = 22 11 локальная переменная с таким же именем закрывает
    //одноименную глобальную переменную, но указав в коде, что «взять имя из глобального пространства имен»,
    //мы получим доступ к глобальной переменной a
    {
        int a = 33;
        cout << " 3 a = " << a << " " << ::a << endl;//a = 33 11 аналогично
    }
    cout << " 4 a = " << a << " " << ::a << endl;//a = 22 11 можно получить доступ к «местной» локальной переменной,
    //а соответствующим синтаксисом можно обратиться к глобальной одноименной переменной
}

```

### Глобальные и локальные переменные

Пример вывода на экран значения переменной a через функции. Функция f1 - без формального параметра, функция f2 выводит на экран значение переданного параметра.

```

#include <iostream>
using namespace std;
int a = 11;// a – глобальная переменная
void f1();// объявление функции f1 (прототип) без параметров
void f2(int);// объявление функции f2 (прототип), принимающей один входной параметр
void main()
{
    cout << " 1 " << endl;
    f1();// вызов функции f1      a = 11
    f2(a);// вызов функции f2     a = 11
    cout << endl << " 2 " << endl;
    a = 22;//изменим значение глобальной переменной
    f1();// вызов функции f1      a = 22
    f2(a);// вызов функции f2     a = 22
    cout << endl << " 3 " << endl;
    f1();// вызов функции f1      a = 22
    f2(33);// вызов функции f2(), которой передали фактический параметр – число 33      a = 33
}
void f1()// определение функции f1
{
    cout << " 1 a = " << a << endl;
}
void f2(int a) // определение функции f2
{
    cout << " 2 a = " << a << endl;
}

```

### Генерация случайных чисел

Для этого используется функция rand() из стандартной библиотеки C. Прототип функции описан в файле <stdlib.h>. Функция rand() генерирует целое число в диапазоне от 0 до значения rand\_max(символическая константа, определенная в заголовочном файле <stdlib.h>).

Значение rand\_max (или RAND\_MAX) должно быть равно 32 767 (максимальное положительное значение

двухбайтового целого числа; в шестнадцатеричной системе счисления это число  $0x\ 7FFF_{16} = 32\ 767_{10}$ ).

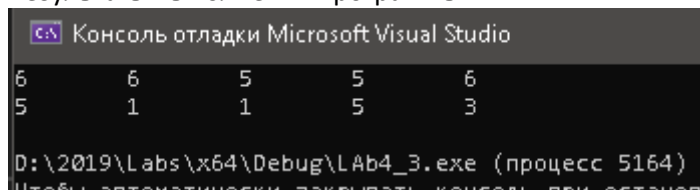
Диапазон значений, которые вырабатываются непосредственно функцией `rand`, отличается от диапазонов, которые требуются в различных программах. Например, программа, моделирующая бросание монеты, требует только двух значений: 0 (орел) и 1 (решка); программа, моделирующая бросание шестигранного кубика, требует шесть значений и т.д.

Чтобы выработать целые числа в диапазоне от 0 до 5, используем операцию вычисления целочисленного остатка `%` в сочетании с `rand`: `rand() % 6`.

Это называется **масштабированием**. Число 6 называется масштабирующим коэффициентом. Затем будем сдвигать диапазон чисел, добавляя 1 к полученному результату.

```
1  #include <iostream>
2  // #include <stdlib.h> // обычно работает и без подключения этой библиотеки
3  using namespace std;
4
5  void main()
6  {
7      for (int i = 1; i <= 10; i++)
8      {
9          cout << 1 + rand() % 6 << "\t";
10         if (i % 5 == 0)
11         {
12             cout << endl;
13         }
14     }
15 }
```

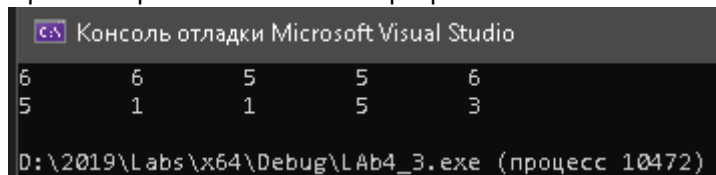
Результаты выполнения программы:



```
6      6      5      5      6
5      1      1      5      3

D:\2019\Labs\x64\Debug\LAb4_3.exe (процесс 5164)
```

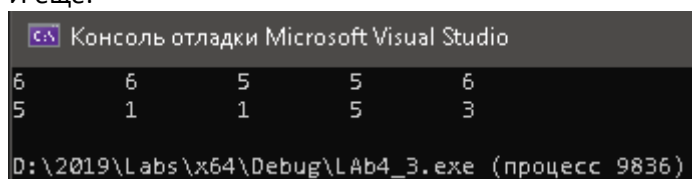
При повторном выполнении программы:



```
6      6      5      5      6
5      1      1      5      3

D:\2019\Labs\x64\Debug\LAb4_3.exe (процесс 10472)
```

И еще:



```
6      6      5      5      6
5      1      1      5      3

D:\2019\Labs\x64\Debug\LAb4_3.exe (процесс 9836)
```

Заметьте, что при повторном выполнении программы выводится точно та же последовательность чисел, которая уже была раньше (и на вашем компьютере тоже), т.е. функция `rand` генерирует псевдослучайные числа. Чтобы реализовать элемент случайности на самом деле, следует также применить стандартную библиотечную функцию `srand()`. Эта функция задает начальное число, которое функция `rand()` использует для генерации последовательности случайных чисел. Рассмотрим ту же программу бросания кубика, но с применением функции `srand()`.

```

1  #include <iostream>
2  // #include <stdlib.h> // обычно работает и без подключения этой библиотеки
3  using namespace std;
4
5  void main()
6  {
7      unsigned int a; // функция srand() принимает беззнаковое целое число
8      cout << "Input a number: ";
9      cin >> a; // вводя свое значение, вы влияете на генерацию случайного числа
10     srand(a); // функция srand() принимает число и задает его в качестве начального для работы функции rand()
11     for (int i = 1; i <= 10; i++)
12     {
13         cout << 1 + rand() % 6 << "\t";
14         if (i % 5 == 0)
15         {
16             cout << endl;
17         }
18     }
19 }

```

Результаты выполнения программы с различными исходными данными:

```

C:\> Консоль отладки Microsoft Visual Studio
Input a number: 67
6      1      4      6      2
1      6      1      6      4
D:\2019\Labs\x64\Debug\Lab4_3.exe (процесс 11044)

C:\> Консоль отладки Microsoft Visual Studio
Input a number: 432
4      6      3      1      6
3      1      5      4      2
D:\2019\Labs\x64\Debug\Lab4_3.exe (процесс 4912)

C:\> Консоль отладки Microsoft Visual Studio
Input a number: 67
6      1      4      6      2
1      6      1      6      4
D:\2019\Labs\x64\Debug\Lab4_3.exe (процесс 10352)

C:\> Консоль отладки Microsoft Visual Studio
Input a number: 2
4      3      1      6      1
1      5      2      2      5
D:\2019\Labs\x64\Debug\Lab4_3.exe (процесс 7780)

C:\> Консоль отладки Microsoft Visual Studio
Input a number: 2
4      3      1      6      1
1      5      2      2      5
D:\2019\Labs\x64\Debug\Lab4_3.exe (процесс 2636)

```

Чтобы не вводить каждый раз начальное число, можно использовать выражение, подобное следующему:  
srand(time(NULL));

При этом для автоматического получения начального значения компьютер считывает показания своих часов. Функция time(с аргументом NULL) возвращает текущее время на вашем компьютере в секундах. Это значение преобразуется в беззнаковое целое число и используется как начальное значение в генераторе случайных чисел. Прототип функции time() находится в библиотеке <time.h>, которую надо подключить.



```

1 #include <iostream>
2 #include <time.h> //библиотека необходима для вызова функции time.h
3 //include <stdlib.h> //обычно rand() работает и без подключения этой библиотеки
4 using namespace std;
5
6 void main()
7 {
8     srand(time(NULL)); //функция srand() принимает число (значение времени на этом ПК, измерено в секундах) и задает его в качестве начального для работы функции rand()
9     for (int i = 1; i <= 10; i++)
10     {
11         cout << 1 + rand() % 6 << "\t"; //теперь генерация случайных значений зависит от времени запуска программы, то есть тоже косвенно от вас
12         if (i % 5 == 0)
13         {
14             cout << endl;
15         }
16     }
17 }

```

Тестируем:

### Резюме:

Функция – изолированный именованный блок кода, имеющий определенное назначение.

Информация в функцию передается с помощью аргументов (фактических параметров), задаваемых при ее вызове в функции main(). Эти аргументы должны соответствовать формальным параметрам, указанным в описании функции. Синтаксис описания функции:

```

типВозвращаемогоРезультата имяФункции(списокВходныхПараметров) // заголовок функции
{
    операторы; // тело функции (код тела функции)
    return возвращаемыйРезультат;
}

```

Область видимости параметров функции, объявленных в ее заголовке и переменных, объявленных в ее теле, ограничивается блоком тела функции.

Эти переменные, если они не объявлены с атрибутом static, уничтожаются после завершения выполнения функции, а хранимые ими значения безвозвратно теряются.

Если функция не возвращает значения, в качестве типа возврата указывается void, а оператор return не содержит выражения, значение которого должно быть возвращено в вызываемую функцию, то есть пишется return;

Прототипы функций:

К моменту вызова функции компилятор должен иметь информацию о ней, чтобы скомпилировать правильно ее вызов.

Если текст функции размещен в файле с исходным текстом после ее вызова или вообще размещен в другом файле, необходимо объявить функцию с помощью оператора, называемого прототипом функции.

Все прототипы функций обычно помещаются в начале исходного файла программы. Заголовочные файлы, включаемые для использования стандартных библиотечных функций, помимо прочего включают в себя прототипы этих функций.

Описание прототипа незначительно отличается от описания заголовка функции:

```
типВозвращаемогоРезультата имяФункции(списокВходныхПараметров);
```

Описание прототипа, в отличие от заголовка, заканчивается точкой с запятой. В прототипе указываются типы входных параметров, их следование друг относительно друга, но не указываются их формальные имена, поскольку они не нужны в прототипе.

Способы передачи данных в вызываемую функцию:

- 1) Путем передачи аргументов функции (они пишутся в круглых скобках)
- 2) С использованием глобальных переменных (возможный, но нежелательный вариант)
- 3) Через файлы на внешних запоминающих устройствах (HDD, SSD, Flash Memory)

Способы передачи данных из вызываемой функции:

- Через возвращаемое значение (return значение);
- Через формальные параметры, вызываемые по ссылке (функция приняла ссылки и может посредством их изменить)

- Путем изменения значений глобальных переменных (возможный, но нежелательный вариант)
- Через файлы на внешних запоминающих устройствах (HDD, SSD, Flash Memory)

### Разработка функции:

- 1) Определяется интерфейс функции:
- 2) какие значения подаются ей на вход
- 3) что должно получиться в результате
- 4) продумываются структуры данных в функции и их обработка для получения требуемого результата

Для функций (пользовательских функций) используется следующая терминология (рамки выделяют **участки**, имеющие свое **название**, которое написано рядом и тем же **цветом**, что и относящаяся к ним **рамка**):

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 double cosh(double x, double eps); //прототип функции cosh() | Прототип функции - это описание функции
5 //sh x = x^1 / 1! + x^3 / 3! + x^5 / 5! + x^7 / 7! + ...
6 int main()
7 {
8     double Xn, Xk, dX, eps;
9     cout << "Enter Xn: ";    cin >> Xn; //начальное значение x
10    cout << "Enter Xk: ";    cin >> Xk; //конечное значение x
11    cout << "Enter dX: ";    cin >> dX; //шаг приращения x
12    cout << "Enter eps: ";   cin >> eps; //требуемая точность вычисления sh(x)
13    cout << "\tX\t\t\tY\t\t\n-----+\t\t\t\t\t\n"; //печатать "шапки" таблицы
14    for (double x = Xn; x <= Xk; x += dX)
15    {
16        cout << "\t" << x << "\t|\t" << cosh(x, eps) << endl; //вызов функции cosh() в потоке cout
17    }
18    cout << "-----+\t\t\t\t\t\n"; //конец печати таблицы
19    system("pause");
20    return 0;
21 }
22 double cosh(double x, double eps) //определение функции cosh()
23 {
24     const int MaxIter = 500; //зададим максимальное количество итераций для недопущения бесконечного заикливания
25     double ch = 1, y = ch; //первый член ряда и начальное значение суммы
26     for (int n = 0; fabs(ch) > eps; n++) //вычислять, пока значение очередного члена ряда больше требуемой точности
27     {
28         ch *= pow(x, 2) / ((2 * n + 1) * (2 * n + 2)); //вычисляем очередной член ряда по оптимизированной формуле
29         y += ch; //добавление очередного члена ряда к сумме
30         if (n > MaxIter) //проверка: если количество итераций превышает лимит
31         {
32             cout << "Ряд расходится!\n"; //то сообщить об этом пользователю
33             return 0; //завершить вычисления
34         }
35     }
36     return y; //возвращаем вычисленную сумму для конкретного x
37 }

```

**Тип возвращаемого функцией значения**

**Список входных параметров, аргументы функции, формальные параметры функции**

**Имя функции**

**Заголовок функции**

**Вызов пользовательской функции в main'e**

**Фактические параметры, передаваемые в функцию**

**Тело функции**

### Задание 1

Для решения поставленных задач написать функции. За взаимодействие с пользователем должна отвечать только функция `main()`, в которой создаются необходимые переменные, массивы и т.д., вызываются функции с фактическими параметрами. Функции должны находиться за `main()`ом, а перед ним должны записываться прототипы функций. Задания можно решить в одном проекте с меню или разных проектах.

1	1	Функция принимает одномерный знаковый целочисленный массив, целое число и проверяет, что сумма элементов массива не больше данного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный вещественный массив и должна вернуть в <code>main()</code> сумму, произведение элементов массива и сумму модулей значений элементов массива.
2	1	Функция принимает одномерный целочисленный массив, целое число и проверяет, что сумма элементов массива равна квадратному корню из данного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный вещественный массив и должна вернуть в <code>main()</code> сумму и произведение модулей значений элементов массива и количество четных по индексу элементов массива.
3	1	Функция принимает одномерный целочисленный массив и проверяет, что все его элементы равны между собой. Функция возвращает результат логического типа.
	2	Функция принимает одномерный вещественный массив и должна вернуть в <code>main()</code> сумму и произведение модулей значений элементов массива и количество нечетных по индексу элементов массива.
4	1	Функция принимает одномерный <code>short int</code> массив и проверяет, что все его элементы не равны между собой. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение четных значений элементов массива и количество нечетных по индексу элементов массива.
5	1	Функция принимает одномерный целочисленный массив, целое число и проверяет, что сумма элементов массива равна данному числу. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение нечетных значений элементов массива и количество четных по индексу элементов массива.
6	1	Функция принимает одномерный <code>long int</code> массив, целое число и проверяет, что сумма элементов массива меньше данного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение квадратных корней значений элементов массива и количество положительных элементов массива.
7	1	Функция принимает одномерный знаковый целочисленный массив, целое число и проверяет, что сумма элементов массива больше данного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение квадратных корней значений элементов массива и количество отрицательных элементов массива.
8	1	Функция принимает двумерный вещественный массив и проверяет, равны ли суммы элементов его первой строки и последнего столбца. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение квадратных корней из положительных значений элементов массива и количество отрицательных элементов массива.
9	1	Функция принимает двумерный вещественный массив и проверяет, меньше ли сумма элементов его первой строки чем сумма элементов последнего столбца. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение положительных значений элементов массива и количество отрицательных элементов массива.
10	1	Функция принимает двумерный вещественный массив и проверяет, больше ли сумма элементов его первой строки чем сумма элементов последнего столбца. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение отрицательных значений элементов массива и количество положительных элементов массива.
11	1	Функция принимает двумерный вещественный массив и проверяет, что сумма элементов его главной диагонали больше нуля. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение кратных трем значений элементов массива и количество положительных элементов массива.
12	1	Функция принимает двумерный вещественный массив и проверяет, что сумма элементов его главной диагонали является отрицательным числом. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение кратных четырем значений элементов массива и количество положительных элементов массива.
13	1	Функция принимает двумерный <code>float</code> массив, число и проверяет, что сумма элементов его главной диагонали больше полученного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение кратных пяти значений элементов массива и количество положительных элементов массива.
14	1	Функция принимает двумерный <code>float</code> массив, число и проверяет, что сумма элементов его главной диагонали

[illegible]

		меньше данного числа. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение некратных одиннадцати значений элементов массива и количество элементов с индексами, меньшими либо равными числу <code>x</code> , принимаемому данной функцией.
29	1	Функция принимает 3 целых числа и проверяет, равны ли они между собой. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение кратных семи значений элементов массива и количество элементов с индексами, большими либо равными числу <code>a</code> , принимаемому данной функцией.
30	1	Функция принимает 3 целых числа и проверяет, что они все не равны между собой. Функция возвращает результат логического типа.
	2	Функция принимает одномерный целочисленный массив и должна вернуть в <code>main()</code> сумму и произведение кратных одиннадцати значений элементов массива и количество элементов с индексами, меньшими либо равными числу <code>a</code> , принимаемому данной функцией.

Смотри далее.

## Задание 2

Для  $X$ , наращиваемого от  $X_n$  до  $X_k$  с шагом  $h$  вычислить значение функции  $f(X)$ . Написать для этого пользовательские функции. Main() запрашивает у пользователя значения, вызывает функции, печатает результат на консоль в виде таблицы значений. Проверить на предложенных тестовых данных (их вводит пользователь).

№ вар.	$f(x)$	$[x_n; x_k]$	$h$
1	$f(x) = 1.5 \cdot  \sin(u) , \text{ где}$ $u = \begin{cases} \sum_{k=1}^5 kx^2 - x + k, & x \leq 2 \\ \operatorname{arctg} x, & x > 2 \end{cases}$	$x \in [0; 5]$	$h=0.25$
2	$f(x) = \sum_{k=1}^5 \frac{(-2)^{k+1} (k+1)!}{(x+2.5)^{k+1}}$	$x \in [-2; 3]$	$h=0.5$
3	$f(x) = \begin{cases} \sqrt[5]{x^3}, & \text{если } x > 0 \\ x^2 + \sum_{k=1}^3 \frac{x^k}{k}, & \text{если } x \leq 0 \end{cases}$	$x \in [-2; 2]$	$h=0.4$
4	$f(x) = \sum_{k=1}^{\infty} \frac{(-1)^{3k} x^{3k}}{(3k)!}$	$x \in [1; 2]$	$h=0.1$
5	$f(x) = \begin{cases}  \cos(x) ^x, & \text{если } x > \pi \\ x^{\cos x}, & \text{если } x \leq \pi \end{cases}$	$x \in [\frac{\pi}{2}; 2\pi]$	$h = \frac{\pi}{4}$
6	$f(x) = \begin{cases} \sum_{k=1}^8 \frac{(-1)^k x^k}{k!}, & \text{если } x > 0 \\ \operatorname{tg}(\pi^x), & \text{если } x \leq 0 \end{cases}$	$x \in [-3; 2.5]$	$h=1.1$
7	$f(x) = 1.5 + \ln \sin(u) , \text{ где}$ $u = \begin{cases} x^2 - x + 1, & x \leq 1.5 \\ \operatorname{arctg} x, & 1.5 < x \leq 2.5 \\ \sin^3(x-1) + \cos(x+1), & x > 2.5 \end{cases}$	$x \in [0; 4]$	$h=0.4$
8	$f(x) = \begin{cases} \ln \sqrt{x^3} , & \text{если } x > 0 \\ \sum_{k=1}^9 kx^2, & \text{если } x \leq 0 \end{cases}$	$x \in [-1; 2]$	$h=0.15$
9	$f(x) = \sum_{k=1}^{\infty} \frac{(-3)^k x^k}{k!}$	$x \in [0.5; 2]$	$h=0.15$
10	$f(x) = \begin{cases} \sum_{k=3}^9 x^{-k}, & \text{если } x > 0 \\ (2-x)^{2-x}, & \text{если } x \leq 0 \end{cases}$	$x \in [-2; 1]$	$h=0.5$
11	$f(x) = \begin{cases} u^{ \cos(x) }, & \text{если } x > \pi \\ u^{\sin x}, & \text{если } x \leq \pi \end{cases}, \text{ где}$ $u = x^2 - \arcsin x^2$	$x \in [-1; 1]$	$h=0.25$
12	$f(x) = e^u, \text{ где}$ $u = \begin{cases} x^2 - \frac{x+1}{3-x}, & x \leq 2 \\ \sin^3(x-1)^2, & x > 2 \end{cases}$	$x \in [1; 3]$	$h=0.2$

13	$f(x) = \begin{cases} \sum_{k=1}^7 \frac{x^k}{k!}, & \text{если } x > 0 \\ \operatorname{arctg}(\pi^x), & \text{если } x \leq 0 \end{cases}$	$x \in [-1; 2]$	$h=0.2$
14	$f(x) = \sum_{k=1}^{\infty} \frac{x^{2k}}{(2k)!}$	$x \in [1; 2]$	$h=0.1$
15	$f(x) = 2.51x^2 \lg 8 - \sin(u) , \text{ где}$ $u = \begin{cases} x^2 - x + 1, & x > -3 \\ \operatorname{arctg} x, & -3 \leq x \leq 3 \\ \sin(x-1) + \cos(x+1), & x > 3 \end{cases}$	$x \in [-5; 5]$	$h=0.5$
16	$f(x) = \begin{cases} \sum_{k=1}^5 x^k, & \text{если } x > 0 \\ \pi^{3.5x}, & \text{если } x \leq 0 \end{cases}$	$x \in [-3; 2]$	$h=0.25$
17	$f(x) = \begin{cases} \lg\left(\sum_{k=1}^4 \frac{x^k}{(2k)!}\right), & \text{если } x > 0 \\ 5^x \sin x^2, & \text{если } x \leq 0 \end{cases}$	$x \in [-3; 8]$	$h=1.1$
18	$f(x) = \sum_{k=1}^5 \frac{(-1.5)^k x^k}{k!}$	$x \in [0.5; 2]$	$h=0.15$
19	$f(x) = \begin{cases} \log_5 u^{ \cos(x) }, & \text{если } u > 1 \\ \arccos u^4, & \text{если } u \leq 1 \end{cases}, \text{ где}$ $u = x^2 - \sin x^2$	$x \in [-2; 2]$	$h=0.4$
20	$f(x) = \begin{cases} \sum_{k=1}^5 \frac{x^{-k}}{k+2}, & \text{если } x > -2 \\ (3+x)^{2-x}, & \text{если } x \leq -2 \end{cases}$	$x \in [-3; 0.5]$	$h=0.25$
21	$f(x) = \sum_{k=1}^6 \frac{(-3)^{k+1} (k+1)!}{(x+5)^{k+1}}$	$x \in [3; 7]$	$h=0.5$
22	$f(x) = \sum_{k=1}^{\infty} \frac{e^k x^k}{(2k-1)!}$	$x \in [1; 2]$	$h=0.1$
23	$f(x) = \arccos(0.5 \cdot  \sin(u-3) ) + 3x, \text{ где}$ $u = \begin{cases} \frac{x^2 - 7x + 3}{x - e^x}, & x \leq 2 \\ \operatorname{arctg}^2 x, & x > 2 \end{cases}$	$x \in [1; 2]$	$h=0.1$
24	$f(x) = \begin{cases} \cos\left(\sum_{k=1}^4 \frac{x^k}{k!} - 5\right), & \text{если } x > 0 \\ e^x \operatorname{arctg} x, & \text{если } x \leq 0 \end{cases}$	$x \in [-2; 2]$	$h=0.4$
25	$f(x) = \sum_{k=1}^{\infty} \frac{(-1.5)^k x^k}{(2k)!}$	$x \in [0.5; 2]$	$h=0.15$



26	$f(x) = \begin{cases} \lg \cos(x) , & \text{если } x > \pi \\ a^{\sin x}, & \text{если } x \leq \pi \end{cases}, \text{ где } a = \arcsin(1 -  \cos x^3 )$	$x \in \left[ \frac{\pi}{2}; 2\pi \right]$	$h = \frac{\pi}{4}$
27	$f(x) = \begin{cases} (t \cdot x - 3 \ln x )^{\frac{2}{3}}, & t < 1.5 \\ (1+t) \cdot x^2 - \sin x, & 1.5 \leq t \leq 3 \\ ((1-t) \cdot x^3 +  \cos x )^{\frac{1}{3}}, & t > 3 \end{cases}, \text{ где } t = \cos^2 x - 3.5$	$x \in [-2; 4]$	$h = 0.6$
28	$f(x) = \begin{cases} 10 - \sum_{k=1}^5 x^k, & \text{если } x > 1 \\ e^{\cos(3,5x+3)}, & \text{если } x \leq 1 \end{cases}$	$x \in [0; 2]$	$h = 0.25$
29	$f(x) = \frac{\arccos(0.5 -  \sin(u-3) )}{e^x - \pi^2}, \text{ где } u = \begin{cases} \frac{x^2 - 7x + 3}{x - e^x}, & x \leq 4 \\ \lg^2 x^2, & x > 4 \end{cases}$	$x \in [1; 7]$	$h = 0.6$
30	$f(x) = \begin{cases} \sum_{k=1}^5 \frac{x^k}{15 - k^2}, & \text{если } x > 0 \\ e^{3,5x}, & \text{если } x \leq 0 \end{cases}$	$x \in [-1; 2]$	$h = 0.3$

### Задание 3

Знать материал, помещенный в начале данной лабораторной работы. Уметь создавать функции.