# Лабораторная работа №10. Функции

## 1 Цель и порядок работы

Цель работы – изучить возможности языка по организации функций, получить практические навыки в составлении программ с их использованием.

Порядок выполнения работы:

* ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя, согласно своему варианту;
* написать программу и отладить ее на ЭВМ;
* оформить отчет.

## 2 Краткая теория

Функция – самостоятельная единица программы, спроектированная для реализации конкретной задачи, представляющая именованную группу операторов и выполняющая законченное действие. Вызов функции приводит к выполнению некоторых действий. К функции можно обратиться по имени, передать ей значения и получить из нее результат. Функции избавляют нас от повторного программирования, позволяют использовать их в различных программах, повышают уровень модульности программы. Функции нужны для упрощения структуры программы. Разбив задачу на подзадачи и оформив каждую из них в виде функций, мы улучшаем понятность и структурированность программы.

На практике программирования часто встречается ситуация, когда одну и ту же группу операторов, реализующих определенный алгоритм, требуется повторить без изменений в нескольких других местах программы. Такую группу операторов во всех языках программирования принято называть подпрограммой, ее можно вызвать для исполнения по имени любое количество раз из различных мест программы.

Подпрограмма имеет ту же структуру, что и монолитная программа. То есть может содержать разделы описания переменных, типов, констант и т.д. Имена объектов, описанных в подпрограмме, считаются известными только в пределах данной подпрограммы.

### 2.1 Описание функций

Функция может принимать параметры и возвращать значение. Передача в функцию различных аргументов позволяет, записав ее один раз, использовать многократно для разных данных. Чтобы использовать функцию, не требуется знать, как она работает – достаточно знать, как ее вызвать.

Для использования функции тоже требуется знать только ее интерфейс (т. е. правила обращения). Интерфейс грамотно написанной функции определяется ее заголовком, потому что в нем указывается все, что необходимо для ее вызова: имя функции, тип результата, который она возвращает, а также, сколько аргументов и какого типа ей нужно передать.

**Объявление функции** (прототип, заголовок, сигнатура) задает ее имя, тип возвращаемого значения и список передаваемых параметров. Формат простейшего заголовка (прототипа) функции:

**[ класс ] тип имя ([ список\_формальных\_параметров ]);**

**Определение функции** содержит, кроме объявления, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

**[ класс ] тип имя** (**[ список\_формальных\_параметров ]**)

{

**тело функции**

}

*Выражение, записанное в квадратных скобках, является опциональным и может быть опущено.*

Рассмотрим составные части определения.

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

- extern – глобальная видимость во всех модулях программы (по умолчанию);

- static – видимость только в пределах модуля, в котором определена функция.

2. 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.

3. Список формальных параметров определяет данные, 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 имена можно опускать), а также значение параметра по умолчанию (при этом все параметры правее данного должны также иметь значения по умолчанию).

Функция активируется с помощью оператора вызова функции, в котором содержатся имя функции и параметры (если это необходимо). Вызов функции приводит к выполнению операторов, составляющих тело функции, и выглядит следующим образом:

имя ([ список\_фактических\_параметров ]);

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничений по соответствию не накладывается, поскольку функцию можно вызывать с различными аргументами, а в прототипах имена компилятором игнорируются (они служат только для улучшения читаемости программы).

Функцию можно определить как встроенную с помощью модификатора inline, который рекомендует компилятору вместо обращения к функции помещать ее код непосредственно в каждую точку вызова. Модификатор inline ставится перед типом функции. Он применяется для коротких функций, чтобы снизить накладные расходы на вызов (сохранение и восстановление регистров, передача управления). Директива inline носит рекомендательный характер и выполняется компилятором по мере возможности. Использование inline-функций может увеличить объем исполняемой программы. Определение функции должно предшествовать ее вызовам, иначе вместо inline-расширения компилятор сгенерирует обычный вызов.

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.

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.

Тип возвращаемого значения и типы параметров совместно определяют тип функции.

Возврат вычисленного значения организуется следующим образом. В теле функции должен присутствовать оператор return после которого следует выражение, вычисляющее возвращаемое значение. Таких операторов может быть несколько; важно, чтобы хоть один из них срабатывал в процессе выполнения тела функции. Тип выражения в правой части такого присваивания должен быть совместимым с типом функции.

Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов. Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция. Если тип возвращаемого функцией значения не void, она может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.

Пример функции, возвращающей сумму двух целых величин:

#include <iostream>

using namespace std;

int sum(int а, int b); // объявление функции

void main()

{

int a = 2, b = 3, c, d;

c = sum(a, b); // вызов функции

cin >> d;

cout << sum(c, d) << endl; // вызов функции

a = 3\*sum(c, d) - b; // вызов функции

cout << a << endl;

return;

}

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

{

return (a + b);

}

Объявление и определение функции может совпадать, но перед вызовом компилятор должен знать прототип функции, поэтому **одновременное объявление и определение должно следовать перед вызовом функции**.

Пример функции нахождения максимума двух чисел:

#include <iostream>

using namespace std;

int max(int x, int y) // одновременно объявление и определение функции

{

if (x > y)

return x;

else

return y;

}

void main()

{

int a = 2, b = 3, c, d;

c = max(a, b); // вызов функции

cout << c << endl;

cin >> d;

cout << max(c, d) << endl; // вызов функции

return;

}

### 2.2 Параметры функции

Механизм параметров является основным способом обмена информацией между вызываемой и вызывающей функциями.

Параметры, перечисленные в заголовке описания функции, называются формальными параметрами, или просто параметрами, а записанные в операторе вызова функции – фактическими параметрами, или аргументами.

В описании процедуры или функции задается список формальных параметров. Каждый параметр, описанный в списке формальных параметров, является локальным по отношению к описываемой процедуре или функции, и в теле подпрограммы на него можно ссылаться по его идентификатору.

При вызове функции в первую очередь вычисляются выражения, стоящие на месте аргументов; затем в стеке выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего аргумента. При этом проверяется соответствие типов и при необходимости выполняются их преобразования.

Существует два основных способа передачи параметров в функцию: **по значению** и **по адресу**.

*При передаче 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 по адресу* в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов. Данный способ подразделяется на передачу параметров через указатель и по ссылке.

#include <iostream>

using namespace std;

void f(int a, int\* b, int& c)

{

a++;

(\*b)++;

c++;

}

void main()

{

int a = 1, b = 1, c = 1;

cout << "a b c" << endl;

cout << a << ' ' << b << ' ' << c << endl;

f(a, &b, c);

cout << a << ' ' << b << ' ' << c << endl;

return;

}

Результат работы программы:

a b c

1 1 1

1 2 2

Первый параметр (a) передается по значению. Его изменение в функции не влияет на исходное значение. Второй параметр (b) передается по адресу с помощью указателя, при этом для передачи в функцию адреса фактического параметра используется операция взятия адреса, а для получения его значения в функции требуется операция разыменования. Третий параметр (c) передается по адресу с помощью ссылки.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются. Поэтому использование ссылок вместо указателей улучшает читаемость программы, избавляя от необходимости применять операции получения адреса и разыменования. Использование ссылок вместо передачи по значению более эффективно, поскольку не требует копирования параметров, что имеет значение при передаче структур данных большого объема.

Если требуется изменить значение параметра внутри функции, то он передается, либо через ссылку, либо через указатель.

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

int f(const char \*);

Таким образом, исходные данные, которые не должны изменяться в функции, предпочтительнее передавать ей с помощью константных ссылок.

По умолчанию параметры любого типа, кроме массива и функции (например, вещественного, структурного, перечисление, объединение, указатель), передаются в функцию по значению.

Если в определении функции присутствует параметр со значением по умолчанию, то данный параметр можно опустить при вызове функции. При этом все параметры левее данного должны присутствовать. Значение данного параметра внутри функции будет совпадать со значением параметра по умолчанию.

Рассмотрим пример из предыдущей лабораторной работы, оформив вычисление значения в виде функции:

Составить программу вычисления значения функции



#include <iostream>

#include <iomanip>

#include <сmath>

using namespace std;

float ax(float a, float x, float eps = 0.001f)

{

float xn = 1, y, y0;

unsigned int n = 1, nf = 1;

y = 1;

do

{

nf \*= n;

xn \*= x\*log(a);

y0 = y;

y += xn/nf;

n++;

}

while (fabs(y - y0) > eps);

return y;

}

void main()

{

setlocale(LC\_ALL, "Russian");

const float eps = 0.1f;

float a, x, y;

cout << "Введите a, x:" << endl;

cin >> a >> x;

//передаем значение точности вычислений при вызове

y = ax(a, x, eps);

cout <<\

"Результат вычисления с точностью 0.1 (передается явно): "\

<< setw(8) << setprecision(5) << y <<endl;

//значение esp опускается

//внутри функции используется значение точности вычислений по умолчанию

y = ax(a, x);

cout <<\

"Результат вычисления с точностью 0.001 (по умолчанию): "\

<< setw(8) << setprecision(5) << y <<endl;

//вычисляем при помощи стандартных функций

y = pow(a, x);

cout <<\

"Результат вычисления с использованием стандартных функций: "\

<< setw(8) << setprecision(5) << y <<endl;

}

Результат выполнения:

Введите a, x:

2.5

3

Результат вычисления с точностью 0.1 (передается явно): 15.591

Результат вычисления с точностью 0.001 (по умолчанию): 15.625

Результат вычисления с использованием стандартных функций: 15.625

## Контрольные вопросы

1. Локальные и глобальные переменные.
2. Как выглядит определение функции?
3. Как выглядит объявление функции?
4. Что такое формальный параметр?
5. Что такое фактический параметр?
6. Что такое параметр по умолчанию? Приведите примеры.
7. Как осуществляется вызов функции?
8. Как осуществляется передача параметров в функцию?
9. Как описать функцию, не возвращающую значения?
10. В чем разница передачи параметров по значению от передачи параметров по адресу?
11. Как передать параметр по ссылке? Приведите примеры.
12. Как передать параметр через указатель? Приведите примеры.
13. Каков механизм сохранения функцией значения между вызовами функции?
14. Что такое inline-функция?
15. Что такое шаблон функции? Правила описания
16. Что такое перегрузка функции. Пример
17. Возможные ошибки при перегрузке функций.
18. Что такое рекурсия. Приведите пример

**Функции с числовыми параметрами**

**Варианты заданий**

**Задание 1**

1. Описать функцию RootCount (*A*, *B*, *C*) целого типа, определяющую количество корней квадратного уравнения *A*·*x*2 + *B*·*x* + *C* = 0 (*A*, *B*, *C* – вещественные параметры, *A* >0). С ее помощью найти количество корней для каждого из трех квадратных уравнений с данными коэффициентами. Количество корней определять по значению *дискриминанта*: *D* = *B*2 – 4·*A*·*C*.
2. Описать функцию CircleS(*R*) вещественного типа, находящую площадь круга радиуса *R* (*R* – вещественное). С помощью этой функции найти площади трех кругов с данными радиусами. Площадь круга радиуса *R* вычисляется по формуле *S* = π·*R*2. В качестве значения  использовать 3.14.
3. Описать функцию RingS(*R*1, *R*2) вещественного типа, находящую площадь кольца, заключенного между двумя окружностями с общим центром и радиусами *R*1 и *R*2 (*R*1 и *R*2 – вещественные, *R*1 > *R*2). С ее помощью найти площади трех колец, для которых даны внешние и внутренние радиусы. Воспользоваться формулой площади круга радиуса *R*: *S* = π·*R*2. В качестве значения π использовать 3.14.
4. Описать функцию TriangleP(*a*, *h*), находящую периметр равнобедренного треугольника по его основанию *a* и высоте *h*, проведенной к основанию (*a* и *h* – вещественные). С помощью этой функции найти периметры трех треугольников, для которых даны основания и высоты. Для нахождения боковой стороны *b* треугольника использовать *теорему Пифагора*:

*b*2 = (*a*/2)2 + *h*2.

1. Описать функцию SumRange(*A*, *B*) целого типа, находящую сумму всех целых чисел от *A* до *B* включительно (*A* и *B* – целые). Если *A* > *B*, то функция возвращает 0. С помощью этой функции найти суммы чисел от *A* до *B* и от *B* до *C*, если даны числа *A*, *B*, *C*.
2. Описать функцию Calc(*A*, *B*, *Op*) вещественного типа, выполняющую над ненулевыми вещественными числами *A* и *B* одну из арифметических операций и возвращающую ее результат. Вид операции определяется целым параметром *Op*: 1 – вычитание, 2 – умножение, 3 – деление, остальные значения – сложение. С помощью Calc выполнить для данных *A* и *B* операции, определяемые данными целыми *N*1, *N*2, *N*3.
3. Описать функцию Quarter(*x*, *y*) целого типа, определяющую номер координатной четверти, в которой находится точка с ненулевыми вещественными координатами (*x*, *y*). С помощью этой функции найти номера координатных четвертей для трех точек с данными ненулевыми координатами.
4. Описать функцию Even(*K*) логического типа, возвращающую True, если целый параметр *K* является четным, и False в противном случае. С ее помощью найти количество четных чисел в наборе из 10 целых чисел.
5. Описать функцию IsSquare(*K*) логического типа, возвращающую True, если целый параметр *K* (> 0) является квадратом некоторого целого числа, и False в противном случае. С ее помощью найти количество квадратов в наборе из 10 целых положительных чисел.
6. Описать функцию IsPower5(*K*) логического типа, возвращающую True, если целый параметр *K* (> 0) является степенью числа 5, и False в противном случае. С ее помощью найти количество степеней числа 5 в наборе из 10 целых положительных чисел.
7. Описать функцию IsPowerN(*K*, *N*) логического типа, возвращающую True, если целый параметр *K* (> 0) является степенью числа *N* (> 1), и False в противном случае. Дано число *N* (> 1) и набор из 10 целых положительных чисел. С помощью функции IsPowerN найти количество степеней числа *N* в данном наборе.
8. Описать функцию IsPrime(*N*) логического типа, возвращающую True, если целый параметр *N* (> 1) является простым числом, и False в противном случае (число, большее 1, называется *простым*, если оно не имеет положительных делителей, кроме 1 и самого себя). Дан набор из 10 целых чисел, больших 1. С помощью функции IsPrime найти количество простых чисел в данном наборе.
9. Описать функцию DigitCount(*K*) целого типа, находящую количество цифр целого положительного числа *K*. Используя эту функцию, найти количество цифр для каждого из пяти данных целых положительных чисел.
10. Описать функцию DigitN(*K*, *N*) целого типа, возвращающую *N*-ю цифру целого положительного числа *K* (цифры в числе нумеруются справа налево). Если количество цифр в числе *K* меньше *N*, то функция возвращает –1. Для каждого из пяти данных целых положительных чисел *K*1, *K*2, …, *K*5 вызвать функцию DigitN с параметром *N*, изменяющимся от 1 до 5.
11. Описать функцию IsPalindrome(*K*), возвращающую True, если целый параметр *K* (> 0) является *палиндромом* (то есть его запись читается одинаково слева направо и справа налево), и False в противном случае. С ее помощью найти количество палиндромов в наборе из 10 целых положительных чисел. При описании функции можно использовать функции DigitCount и DigitN из заданий выше.
12. Описать функцию DegToRad(*D*) вещественного типа, находящую величину угла в радианах, если дана его величина *D* в градусах (*D* – вещественное число, 0 < *D* < 360). Воспользоваться следующим соотношением: 180° = π радианов. В качестве значения π использовать 3.14. С помощью функции DegToRad перевести из градусов в радианы пять данных углов.

[**Задание**](https://study.chuvsu.ru/mod/assign/view.php?id=18955) **2. Перегрузка функций**

Задание: Выполните задание 1 с возможностью перегрузки функции.

**Задание 3. Шаблон функции**

Задание: Создайте шаблон для функции из задания 1.

**Задание 4. Рекурсивные функции**

1. Описать рекурсивные функции Fact(N) и Fact2(N) вещественного типа, вычисляющие значения факториала N! и двойного факториала N!! соответственно (N > 0 – параметр целого типа). С помощью этих функций вычислить факториалы и двойные факториалы пяти данных чисел.
2. Описать рекурсивную функцию PowerN(x, n) вещественного типа, находящую значение *n*-й степени числа *x* по формуле: , при *n* > 0, при *n* < 0 (*x* ≥ 0 – вещественное число, *n* – целое). С помощью этой функции найти значения xn при 5 различных значениях *n* для данного *x*.
3. Описать рекурсивную функцию SqrtK(x, k, n) вещественного типа, находящую приближенное значение корня k-й степени из числа x по формуле:   где yn обозначает SqrtK(x, k, n) (*x* – вещественный параметр, *k* и *n* – целые; *x* > 0, *k* > 1, *n* > 0). С помощью этой функции найти приближенные значения корня *k*-й степени из *x* при 6 различных значениях *n* для данных *x* и *k*.
4. Описать рекурсивную функцию FibRec(N) целого типа, вычисляющую *N-*е число Фибоначчи F(N) по формуле:   
   F(1) = F(2) = 1, F(k) = F(k-2) + F(k-1), k = 3, 4, ... С помощью этой функции найти пять чисел Фибоначчи с указанными номерами и вывести эти числа вместе с количеством рекурсивных вызовов функции FibRec, потребовавшихся для их нахождения.
5. Описать рекурсивную функцию C(m, n) целого типа, находящую число сочетаний из n элементов по m, используя формулу: C(0, n) = C(n, n) = 1, C(m, n) = C(m, n-1) + C(m-1,n-1) при 0 < m < n (m и n – целые параметры; n > 0, 0 <= m <= n). Дано число *n* и пять различных значений *m*. Вывести числа C(m, n) вместе с количеством рекурсивных вызовов функции C, потребовавшихся для их нахождения.
6. Описать рекурсивную функцию NOD(A, B) целого типа, находящую наибольший общий делитель двух натуральных чисел A и B, используя алгоритм Евклида: NOD(A, B) = NOD(B mod A, A), если A <> 0; NOD(0,B) = B. С помощью этой функции найти наибольшие общие делители пар A и B, A и C, A и D, если даны числа A, B, C, D.
7. Описать рекурсивную функцию MinRec(A,N) вещественного типа, которая находит минимальный элемент вещественного массива A размера N, не используя оператор цикла. С помощью функции MinRec1 найти минимальные элементы массивов A, B, C размера NA, NB, NC соответственно.
8. Описать рекурсивную функцию Digits(S) целого типа, находящую количество цифр в строке S без использования оператора цикла. С помощью этой функции найти количество цифр в данных пяти строках.
9. Описать рекурсивную функцию Simm(S) логического типа, проверяющую, является ли симметричной строка S, без использования оператора цикла. С помощью этой функции проверить данные пяти строк.
10. Алгоритм сортировки фон Неймана. Упорядочить массив а1, а2, … аn по неубыванию с помощью алгоритма сортировки слияниями: а) каждая пара соседних элементов сливается в одну группу из двух элементов (последняя группа может состоять из одного элемента); б) каждая пара соседних двухэлементных групп сливается в одну четырехэлементную группу и т.д. При каждом слиянии новая укрупненная группа сортируется. Использовать функцию сортировки.
11. Задано положительное и отрицательное число в двоичной системе. Составить программу вычисления суммы этих чисел, используя функцию сложения чисел в двоичной системе счисления.
12. Описать рекурсивную функцию Root (а, b, ε), которая методом деления отрезка пополам находит с точностью ε корень уравнения f(x) = 0 на отрезке [а, b] (считать, что ε > 0, а < b, f(a) - f(b) < 0 и f(x) – непрерывная и монотонная на отрезке [а, b] функция).
13. Описать функцию min(X) для определения минимального элемента линейного массива X, введя вспомогательную рекурсивную функцию minl(k), находящую минимум среди последних элементов массива X, начиная с *k*-го.
14. Напишите рекурсивную функцию SumTo(n), которая для данного n вычисляет сумму чисел от 1 до *n*.
15. Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке. Например, для числа 1234 получаем результат 4321.