**Лабораторная работа №9. Функции**

**Краткая теория**

**Язык программирования С++**

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.

**Описание функций**

Функция может принимать параметры и возвращать значение. Передача в функцию различных аргументов позволяет, записав ее один раз, использовать многократно для разных данных.

Для использования функции 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 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

**}**

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 ]);

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.

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. 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**.

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.

Пример функции, возвращающей сумму двух целых величин:

#include "stdafx.h"

#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**);**

**}**

**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.

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.

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

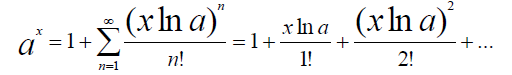
Если требуется запретить изменение параметра внутри функции, используется модификатор **const**:

**int** f**(const char \*);**

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.

Пример: Составить программу вычисления значения функции



#include "stdafx.h"

#include <iostream>

#include <iomanip>

#include <math.h>

**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 (abs(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

**Передача массивов в функцию**

Параметрами функции могут быть массивы, и функции могут возвращать указатель на массив в качестве результата. При использовании массивов в качестве параметров в функцию передается указатель на его первый элемент, т.е. массив всегда передается по адресу.

Если аргументом функции является одномерный массив, ее формальный параметр можно объявить тремя способами: как указатель, как массив фиксированного размера и как массив неопределенного размера.

Если двухмерный массив используется в качестве аргумента функции, то в нее передается только указатель на его первый элемент. При этом необходимо указать количество столбцов. Количество столбцов необходимо компилятору для того, чтобы правильно вычислить адрес элемента массива внутри функции.

**Рекурсивные функции**

Любая функция в программе на C++ может вызываться рекурсивно. При этом в стеке выделяется новый участок памяти для размещения копий параметров, а также автоматических и регистровых переменных, поэтому предыдущее состояние выполняемой функции сохраняется, и к нему впоследствии можно вернуться.

Различают прямую и косвенную рекурсию. Функция называется косвенно-рекурсивной в том случае, если она содержит обращение к другой функции, содержащей прямой или косвенный вызов определяемой первой функции. Если в теле функции используется явный вызов этой же функции, то это прямая рекурсия.

Однако у рекурсии есть и недостатки: во-первых, такую программу труднее отлаживать, поскольку требуется контролировать глубину рекурсивного обращения, во-вторых, при большой глубине стек может переполниться, а в-третьих, использование рекурсии увеличивает расход времени и памяти на повторные вызовы функции.

**Функции с переменным количеством параметров**

В C++ допустимы функции, у которых количество параметров при компиляции определения функции не определено. Кроме того, могут быть неизвестными и типы параметров. Количество и типы параметров становятся известными только в момент вызова функции, когда явно задан список фактических параметров. При определении и описании таких функций спецификация формальных параметров заканчивается многоточием:

тип имя **(**список\_явных\_параметров**, ...);**

Каждая функция с переменным списком параметров должна иметь один из двух механизмов определения их количества и типов.

Первый подход предполагает добавление в конец списка реально использованных, необязательных, фактических параметров специального параметра-индикатора с уникальным значением которое будет сигнализировать об окончании списка. В теле функции параметры перебираются, и их значение сравнивается с заранее известным концевым признаком.

Второй подход предусматривает передачу в функцию значения реального количества используемых фактических параметров, которые передаются с помощью одного из обязательных параметров.

В обоих случаях переход от одного фактического параметра к другому выполняется с помощью указателей (с использованием адресной арифметики).

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

**using namespace std;**

**long summa(int k, ...)**

**{**

int \*pik = &k;

long total = 0;

for (; k; k--)

total += \*(++pik);

return total;

**}**

**int main(int argc, char\* argv[])**

**{**

setlocale**(**LC\_ALL**,** "Russian"**);**

cout << "\n summa(2, 6, 4) = " << summa(2, 6, 4);

cout << "\n summa(6, 1, 2, 3, 4, 5, 6) =" <<

summa(6, 1, 2, 3, 4, 5, 6);

return 0;

**}**

Результат

summa(2, 6, 4)=10

summa(6, 1, 2, 3, 4, 5, 6)=21

Особенность этой программы, что указатель pik может работать только с целочисленными фактическими параметрами.

#include "stdafx.h"

#include <iostream>

**using namespace std;**

**double prod(double arg, ...)**

**{**

double result = 1.0;

double \*prt = &arg;

if (\*prt == 0.0)

return 0.0;

for( ; \*prt; prt++)

result \*= \*prt;

return result;

**}**

**int main(int argc, char\* argv[])**

**{**

setlocale**(**LC\_ALL**,** "Russian"**);**

cout <<"\n prod(2e0,4e0,3e0,0e0) = " << prod(2e0,4e0,3e0,0e0);

cout <<"\n prod(1.5,2.0,3.0,0.0) = " << prod(1.5,2.0,3.0,0.0);

cout <<"\n prod(1.4,3.0,0.0,16.0,84.3,0.0) = " <<

prod(1.4,3.0,0.0,16.0,84.3,0.0);

cout << "\n prod(0e0) ="<< prod(0e0);

return 0;

**}**

Результат

prod(2e0,4e0,3e0,0e0) = 24

prod(1.5,2.0,3.0,0.0) = 9

prod(1.4,3.0,0.0,16.0,84.3,0.0) = 4.2

prod(0e0) = 0

Признаком окончания списка фактических параметров служит параметр с нулевым значением. Чтобы функция с переменным количеством параметров воспринимала параметры различных типов, то для однотипных параметров необходимо передавать информацию функции с помощью обязательного дополнительного параметра.

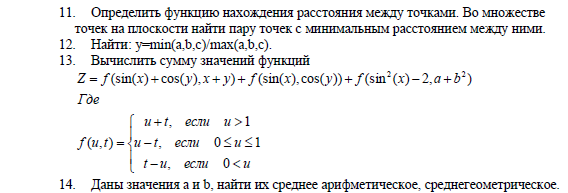
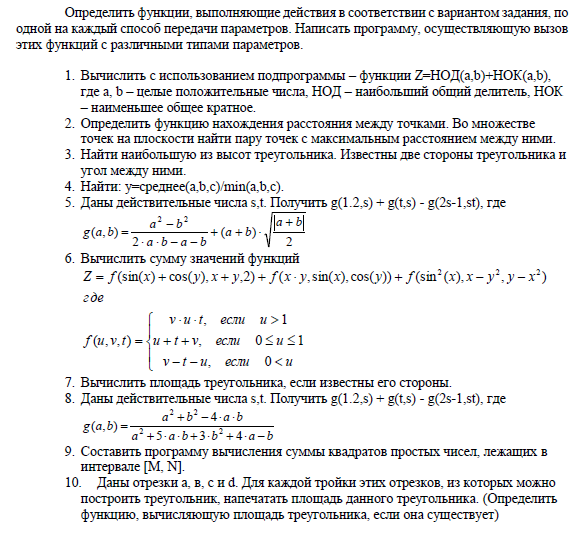
**Передача массива в функцию**

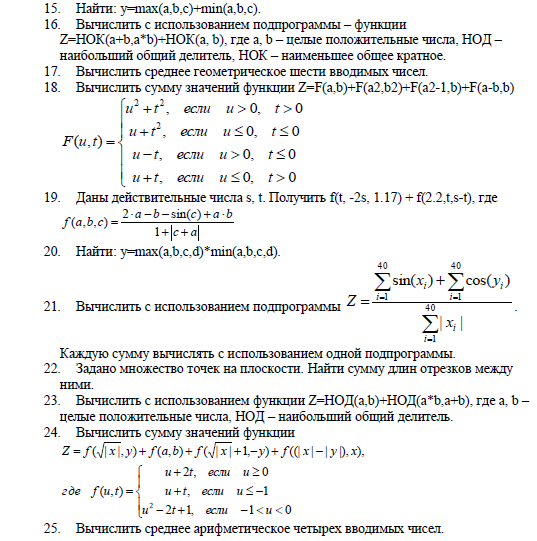
Можно почитать здесь **https://metanit.com/cpp/tutorial/4.7.php**

или здесь **http://www.c-cpp.ru/books/peredacha-massivov-v-funkcii**

***Задания для самостоятельной работы***

**Задание 1. *(Номер варианта = Номер по списку % 25 + 1)***





**Задание 2.**

**Решить задачи из лабораторной работы №8** (согласно **своему** варианту в лабораторной **№8**)**:**

1. Разбейте задачу на подзадачи (например, **подзадача 1** – заполнение массива, **подзадача 2** – вывод элементов массива на экран, **подзадача 3** – нахождение максимума среди элементов массива, **подзадача** **4** – нахождение суммы элементов массива)

2. Определите функции для каждой подзадачи

3. Вызовите функции в нужном порядке для решения задачи.

**Задание 3.**

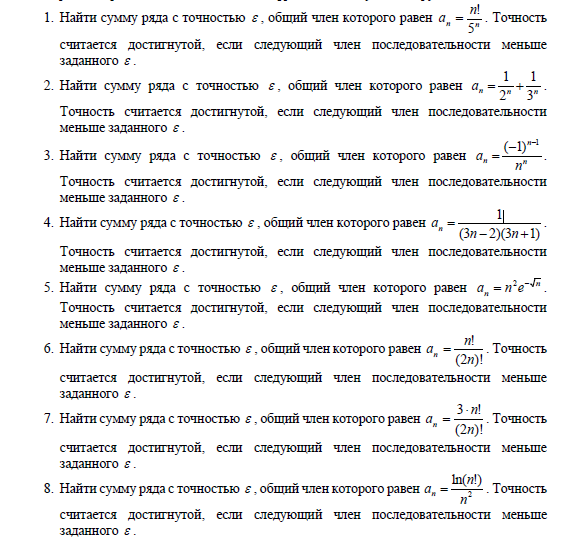
**Решить задачи из лабораторной работы №8+** (согласно **своему** варианту в лабораторной **№8+**)**:**1. Разбейте задачу на подзадачи (например, **подзадача 1** – заполнение массива, **подзадача 2** – вывод элементов массива на экран, **подзадача 3** – нахождение максимума среди элементов массива, **подзадача** **4** – нахождение суммы элементов массива)

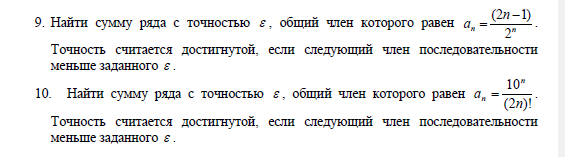
2. Определите функции для каждой подзадачи

3. Вызовите функции в нужном порядке для решения задачи.

**Задание 4. (*Номер варианта = Номер по списку % 10 + 1)***

***Решить задачи с использованием рекурсии***





***Контрольные вопросы***

**Язык программирования С++**

1. Как выглядит определение функции?

2. Как выглядит объявление функции?

3. Что такое формальный параметр?

4. Что такое фактический параметр?

5. Как осуществляется вызов функции?

6. Как осуществляется передача параметров в функцию?

7. Что такое **inline**-функция?

8. Как описать функцию, не возвращающую значения?

9. В чем разница передачи параметров по значению от передачи параметров по адресу?

10. Как передать параметр по ссылке?

11. Как задать значения параметра по умолчанию?

12. Как передать массив в функцию?

13. Как передать многомерный массив в функцию?

14. Что такое рекурсия?

15. Какие виды рекурсии вы знаете?

16. Как описываются функции с переменным количеством параметром?

17. Как передаётся массив в функцию?