**Подпрограмма** – это часть основной программы, которая оформлена таким образом, что становится возможным многократно использовать ее при выполнении основной программы.

……………………………………

Вызов подпрограммы

…………………………………..

Вызов подпрограммы

…………………………………..

Подпрограмма

Подпрограмма может входить в состав готовой библиотеки. В этом случае эту библиотеку требуется «заказать».

Если подпрограммы вызывается из библиотеки, то требуется знать ее имя, входные параметры, а также понимать, как подпрограмма передаст результаты своей работы.

Если же программист сам разрабатывает подпрограмму, то кроме имени, входных параметров и результатов, подпрограмму требуется оформить по правилам языка программирования. Эти правила различаются в разных языках.

В каком случае часть кода целесообразно оформлять как подпрограмму?

Имеет смысл часть кода основной программы оформить в виде подпрограммы тогда, когда он повторяется многократно в разных местах программы. Такой прием сокращает длину кода, хотя увеличивает затраты времени на вызов подпрограммы.

Вторая причина заключается в том, что большую программу целесообразно разделить на несколько блоков и в этом случае программа становится структурно более логичной.

Любая подпрограмма имеет имя (по правилам программирования), использует входные данные и в результате выполнения - выходные данные.

Входные данные называются **параметрами** подпрограммы. Иногда среди параметров могут быть и результат работы подпрограммы.

Подпрограмма может возвращать результат своей работы по-разному: через параметры, либо в виде значения (например, в виде числа).

В некоторых языках программирования (например, pascal, 1C и пр) подпрограммы делятся на два вида: **процедуры** и **функции.**

Процедуры передают результат через параметры, а функции через значение.

Любую подпрограмму можно оформить как процедуру или как функцию. Это дело программиста.

Например, всем известная подпрограмма вычисления **sin**. Если она оформлена как процедура, то ее вызов будет оформлен примерно так:

**sin(30, rez);**

где, 30 – это входные данные, а **rez** – результат работы (0.5).

При оформлении подпрограммы, как функции:

**rez=sin(30);**

где, rez – результат вычисления (0,5)

В некоторых языках различие в оформлении подпрограмм еще заключается в названиях. Процедуры называются **procedure**, а функции – **function.**

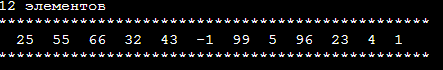
В с++ обычно все подпрограммы называются функциями.

Также в с++ нет формального разделения подпрограмм на процедуры и функции. Любая подпрограмма имеет **тип**, **имя** и **саму подпрограмму**. Отличие лишь в следующем: если в подпрограмме имеется оператор **return**, то формально это функция. Если же тип подпрограммы **void**, то **return** можно не указывать и тогда, подпрограмму можно считать процедурой. Но это достаточно условно.

Передача параметров подпрограмме имеет некоторые особенности в разных языках программирования.

Рассмотрим правила оформления подпрограммы и передачи параметров в с++ на примере.

Допустим, в программе возникла необходимость многократно выдавать на экран содержание массива, причем «красиво» оформленного.



Причем количество звездочек зависит от размерности массива. **n** – количество элементов в массиве **mas** и подсчитать это количество автоматически. Далее предлагается одно из решений**:**

int mas1[12] = { 25, 55, 66, 32, 43,-1, 99,5, 96, 23, 4, 1 };

cout << "12 элементов" << endl;

for (int i = 0; i<(12\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<12; i++) cout << mas1[i] << " ";

cout << endl;

for (int i = 0; i< (12\*4); i++) cout << "\*";

cout << endl << endl;

Если в программе выдаются несколько массивов с разным количеством элементов, то этот код потребуется повторить тоже несколько раз. Например,

#include <iostream>

using namespace std;

int main()

{

int mas1[12] = { 25, 55, 66, 32, 43,-1, 99,5, 96, 23, 4, 1 };

int mas2[10] = {5 , 65, 88, 77, 43,-6, 99,5, 96, 9};

cout << "**12** элементов" << endl;

for (int i = 0; i<(**12**\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<**12**; i++) cout << mas1[i] << " ";

cout << endl;

for (int i = 0; i< (**12**\*4); i++) cout << "\*";

cout << endl << endl;

cout << "**10** элементов" << endl;

for (int i = 0; i<(**10**\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<**10**; i++) cout << mas2[i] << " ";

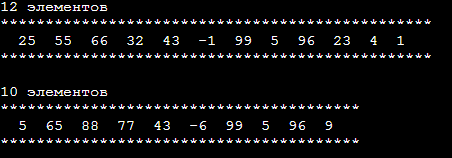
cout << endl;

for (int i = 0; i< (**10**\*4); i++) cout << "\*";

cout << endl << endl;

}

Результат

****

А если нам требуется выдать еще один массив, то еще раз это блок кода придётся повторить.

Для упрощения работы с такими повторяющимся блоками кода обычно используют подпрограммы.

Оформим наш код выдачи массива на экран в виде подпрограммы:

void WrMas(int mas[],int n)

{

cout << n << " элементов" << endl;

for (int i = 0; i<(n\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<n; i++) cout << mas[i] << " ";

cout << endl;

for (int i = 0; i< (n\*4); i++) cout << "\*";

cout << endl <<endl;

}

Наша функция называется **WR** и она не имеет типа (**void**).

Массив описывается как **int mas[],** т.е. подпрограмме не требуется указывать в квадратных скобках размерность. Но указать количество чисел **n** – обязательно.

Параметры, которые указаны при описании подпрограммы, называются **формальными** параметрами.

Формальные параметры называться произвольными именами, но тип и порядок следования – важен и должен соответствовать вызывающей программе. Т.е. mas и n можно заменить на любые другие имена, например, **pam** и **k,** но их тип порядок следования не должен изменятся, иначе будет ошибка:

void WrMas(int pam[],int k)

{

cout << k << "- elements" << endl;

for (int i = 0; i<k\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<k; i++) cout << pam[i] << " ";

cout << endl;

for (int i = 0; i< (k\*4); i++) cout << "\*";

cout << endl << endl;

}

Приведем полный текст программы:

**#include <iostream>**

using namespace std;

void WrMas(int mas[],int n)

{

cout << n << " элементов" << endl;

for (int i = 0; i<(n\*4); i++) cout << "\*";

cout << endl;

cout << " ";

for (int i = 0; i<n; i++) cout << mas[i] << " ";

cout << endl;

for (int i = 0; i< (n\*4); i++) cout << "\*";

cout << endl <<endl;

}

int main()

{

int mas1[12] = { 25, 55, 66, 32, 43,-1, 99,5, 96, 23, 4, 1 };

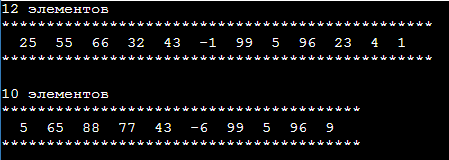
int mas2[10] = {5 , 65, 88, 77, 43,-6, 99,5, 96, 9};

WrMas(mas1,12);

WrMas(mas2,10);

}

Результат:



Нашу функцию **WrMas** вызвали два раза, для разных массивов. Код программы стал существенно короче.

Параметры mas1, mas2, 12 и 10 в главной программе называются **фактическими** параметрами. Они могут задаваться по-разному. В нашем случае это:

**WrMas(mas1, 12);**

**WrMas(mas2, 10);**

В этом случае, мы в подпрограмму передаем как бы сам массив, вернее его копию. Это не совсем экономно. Желательно, передать его адрес. Тогда лишняя память не тратится:

**WrMas(&mas1[0], 12);**

**WrMas(&mas2[0], 10);**

**&** означает «адрес» начала массива (ссылка).

**mas1[0]** – это начало массива.

Конец лекции