ПРОГРАММНЫЕ МОДУЛИ В C++. ФУНКЦИИ.

10 лекция

Дисциплина «Программирование на языке C++»

Модульное программирование

При разработке программы иногда появляются повторяемые группы действий или возникает необходимость разбить программу на функциональные модули, сделать ее структуру иерархической. Для этого во всех языках программирования существуют средства организации подпрограмм (модулей).

Подпрограмма ***(модуль)*** - это специальным образом оформленный алгоритм, который может многократно использоваться при решении более общей задачи.

Программирование, основанное на разбиение программы на независимые модули, называется ***модульным***.

Модульное программирование

Модули в C++ называются функциями и классами.

Функции являются основными компонентами при построении программ на языке C++. Правильное проектирование функций в значительной степени определяет эффективность, читаемость и переносимость программного кода.

В C++ можно создавать пользовательские функции. Собственно говоря, мы с самого начала создавали функции: main() — не что иное, как главная функция пользователя.

В библиотеке C++ имеется немало встроенных функций. Они размещены в отдельных заголовочных файлах, которые подключаются с помощью #include.

Пользовательские функции

Функция – самостоятельная единица программы, представляющая собой группу операторов, выполняющих законченное действие.

Функции позволяют обеспечить внутреннюю организованность программы и сократить размер ее кода, присваивая повторяющимся фрагментам программы имя и заменяя этим именем все повторения, встречающиеся в программе.

К функции можно обратиться по имени, передать ей значение и получить из нее результат. Функции обычно располагаются до основной программы

Для использования собственных функций в программу необходимо добавить три обязательных компонента:

- объявление функции,
- определение функции,
- вызов функции.

Объявление функции

Объявления функций называют также прототипами функций. Прототип функции является одной из важных особенностей C++. Прототип функции указывает компилятору тип возвращаемого значения, количество параметров, которые ожидает функция, тип параметров и порядок их следования. Компилятор использует прототип функции для проверки правильности вызовов функции.

Прототип функции имеет вид:

<возвращаемый тип> ИмяФункции (параметры);

В конце списка параметров стоит знак "точка с запятой", это указывает на то, что мы имеем дело не с описанием функции, а с ее прототипом.

Определение функции

Определение функции состоит из заголовка и тела функции. Тело функции состоит из последовательности операторов, заключенной в фигурные скобки.

Аргументом называют единицу данных, передаваемую программой в функцию. Переменные, используемые внутри функции для хранения значений аргументов, называются параметрами. При определении функции в ее заголовке указываются типы и имена параметров.

Количество аргументов у функции может быть сколько угодно большим, но возвращаемое значение всегда только одно (либо не возвращает значение).

Заголовок функции должен соответствовать ее прототипу: имя функции и тип возвращаемого ей значения должны совпадать с указанными в прототипе. Аргументы функции (если они есть) должны иметь те же типы и следовать в том же порядке, в каком они указывались в прототипе.

Определение функции

Определение функции имеет следующий формат:

```
<возвращаемый тип> ИмяФункции (параметры)
{
//тело функции, состоящее из операторов
}
```

Возвращаемое значение определяется служебным словом **return (*возвращаемое_значение*),** которое размещается в конце функции (перед закрывающей фигурной скобкой).

Тип возвращаемого значения:

1) тип *void* означает, что функция не возвращает никакого значения.

2) функция может возвращать типы, определенные пользователем.

Если тип возвращаемого значения не указан, то по умолчанию он предполагается целочисленным (*int*).

В списке параметров указываются имя и тип для каждой переменной, элементы списка разделяются запятыми.

Знак "точка с запятой" не ставится после списка параметров.

Вызов функции

Функция активизируется (т.е. начинает выполнять запроектированную для неё задачу) путём вызова функции.

ИмяФункции (<набор параметров>);

Вызов функции осуществляется именем функции и передачей набора параметров. Если параметры не передаются, нужно указать пустые скобки

При вызове функции существуют три способа передачи аргументов в функцию:

- вызов по значению,
- вызов по ссылке с аргументами ссылками
- вызов по ссылке с аргументами указателями

Формальные и фактические параметры функции

Для удобства передачи данных в функцию и получения результата используются формальные и фактические аргументы.

Формальные параметры – условные обозначения в описании функции.

Фактические параметры – аргументы, с которыми требуется выполнить функцию. Фактические параметры перечисляются при вызове функции.

Пример: составить функц ет наибольшее из двух значений

тип результата

формальные параметры

```
int Max ( int a, int b )
{
  if ( a > b ) return a ;
  else         return b ;
}
```

return - вернуть результат функции

Локальные и глобальные переменные

Локальные переменные – переменные, которые описываются только внутри подпрограммы (функции) и являющиеся промежуточными при ее выполнении.

Глобальные переменные – переменные, описанные в главной программе.

Особенности:

- можно объявлять и использовать **локальные переменные**

```
float qq ( int a, int b)
{

 float x, y;

}
```

локальные переменные

Локальные переменные недоступны в основной программе и других процедурах и функциях.

Вызов функции по значению

При вызове функции фактические имена параметров заменяют формальные имена, которые использовались при описании функции. Следовательно, совсем не обязательно использовать одинаковые имена при вызове и при описании.

Аргументы в функцию обычно передаются по значению, т.е. в функцию передается только копия текущего значения. Сама переменная в этом случае внутри функции не может быть изменена. Главным ограничением вызова по значению является то, что функция может возвращать только одно значение.

Способ передачи аргументов, при котором функция создает копии передаваемых значений, называется передачей аргументов по значению.

Пример использования функции maximum для вычисления и возвращения наибольшего из трёх целых чисел в ту точку main, из которой вызвана функция maximum.

```
#include <iostream>
using namespace std;
int maximum(int, int, int);                    // объявление функции(прототип)
int main()
{
        int a,b,c,max_main;
        cout <<"Input 3 numbers: ";
        cin >>a >>b >>c;
        max_main=maximum(a,b,c); // вызов функции
        cout <<"Maximum is " <<max_main <<endl;
        return 0;
}
// Определение функции maximum
int maximum(int x, int y, int z)               // заголовок функции
{
        int max=x;
        if(y>max)
                max=y;                         // тело функции
        if(z>max)
                max=z;
        return max;
}
```

1)Прототип функции maximum имеет вид:

int maximum(int, int, int);

Этот прототип указывает, что функция maximum имеет три аргумента типа int и возвращает результат типа int. Необходимо отметить, что прототип функции такой же, как и заголовок описания функции maximum, за исключением того, что в него не включены имена параметров (x, y , и z).

2) В данной программе сначала вводятся три целых числа. Затем эти целые числа передаются в функцию maximum, которая определяет наибольшее из этих чисел. Это значение возвращается в функцию main с помощью оператора return в maximum и выводится на экран.

3) Рассмотрим:

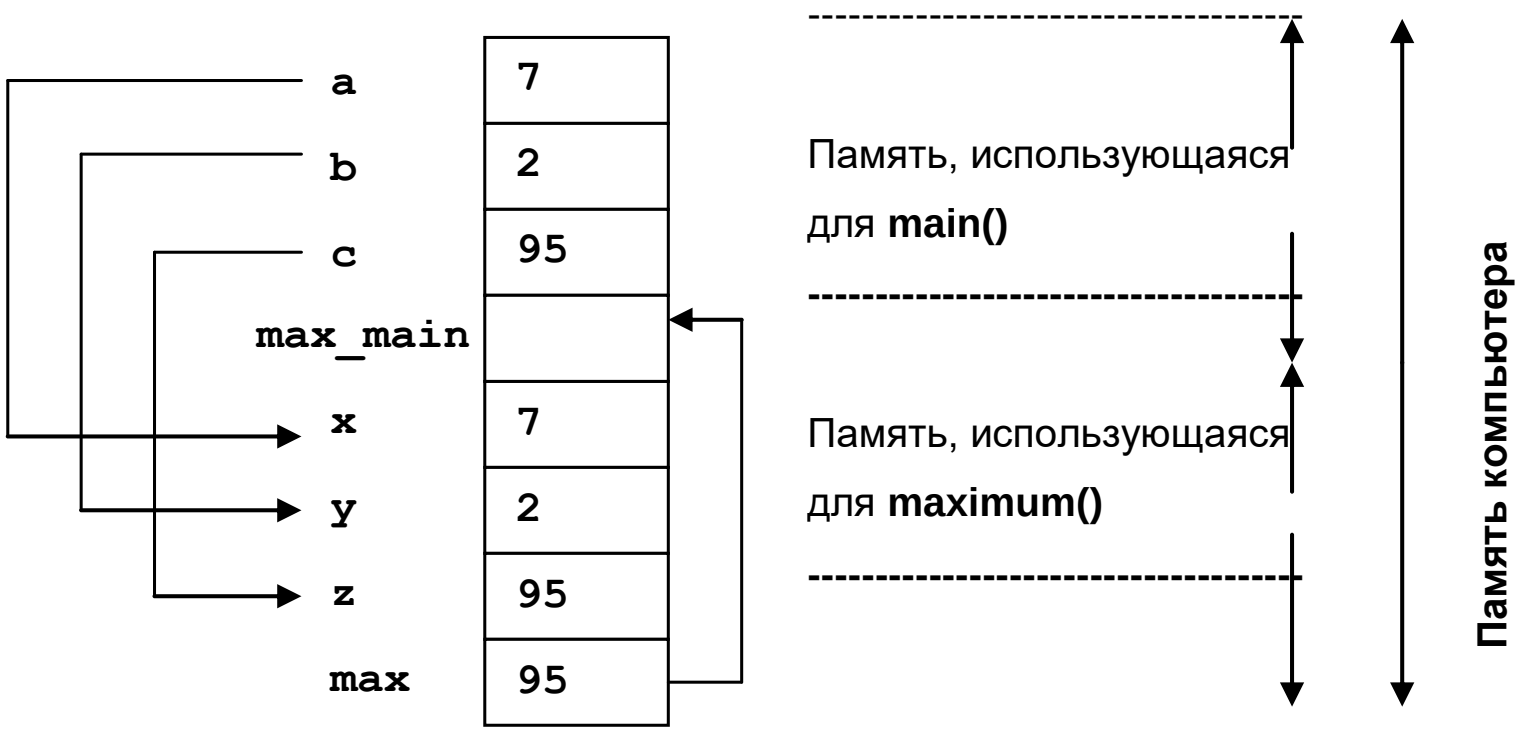
1 – как main() выполняет функцию maximum():

```
max_main=maximum(a,b,c);
```

2 – как пишется первая строка функции maximum():

```
int maximum(int x, int y, int z)
{
}
```

Ячейки памяти



В этих выражениях каждая переменная хранится в отдельной ячейке памяти компьютера. Так, a, b, и c имеют свои собственные ячейки памяти. Из за способа, который мы использовали при записи функции maximum(), x, y, и z имеют собственные ячейки памяти. При вызове функции значение переменной a было скопировано в ячейку памяти x, b в ячейку памяти y, а c в ячейку z. Так же переменная max_main в main() занимает абсолютно другую ячейку памяти, чем переменная max в maximum().