

# Advanced Programming (C++)

BY

Dr. EMAD SAMI

<http://www.bu.edu.eg/staff/emadattwa3>

# Course Chapters

1. Introduction
2. Variables and Constants
3. Expressions and Statements
4. Loops and Decisions
5. **Functions**
6. Arrays and Strings
7. Pointers
8. Miscellaneous

# 5. Functions

## Chapter Objectives:

5-1 What is a Function?

*5-1-1 Function Declaration (or Function Prototype)*

*5-1-2 Function Calling*

*5-1-3 Function Definition*

5-2 Function Declaration Types

*5-2-1 Receives & Returns*

*5-2-2 Receives & NOT Returns*

*5-2-3 NOT Receives but Returns*

*5-2-4 NOT Receives & NOT Returns*

5-3 Examples

5-4 Keyword (**typedef**)

5-5 Local Variables

5-6 Scope Variables

5-7 Global Variables

5-8 Special Topics About Functions

*5-8-1 Overloaded (polymorphism) Function*

*5-8-2 Default Variables*

*5-8-3 Inline Functions*

*5-8-4 Static Variables*

5-9 Assignment (5)

## 5-1 What is a function?

- A *function* is a group of statements into a unit and gives it a name. This unit can then be called from other parts of the program.
- Every C++ program has at least one function, **main ( )**.
- If you want to construct a program to do some thing (e.g., calculate the factorial of a number) using this chapter, then your program must have two *function* types: **main ( )** and **factorial ( )**.
- The *functions* are come in two types: User-defined type and Built-in type.
- User-defined function: is the function type that is made by the user to do some task.
- Built-in function: is a part of your compiler package. They are supplied by the manufacturer for your use.
- The function parts to use it in C++ code:
  - 5-1-1 function declaration (prototype)*
  - 5-1-2 function call*
  - 5-1-3 function definition*

## 5-1 What is a function?

- Here is an example to show how to use *function* in C++ code:
- [FacFunc.cpp](#)

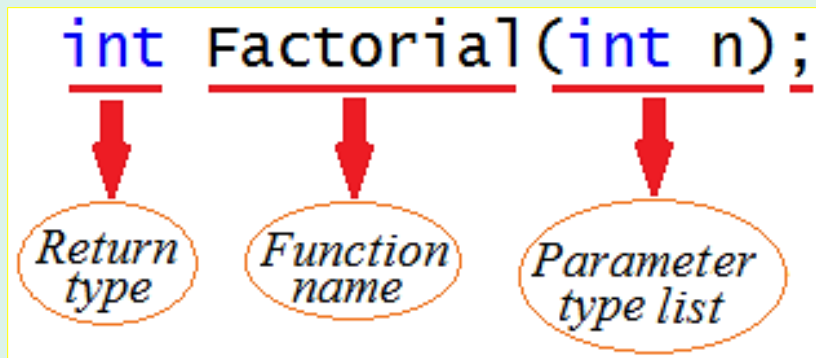
```
1 //FacFunc.cpp
2 // calculate factorial using functions
3 #include<iostream.h>
4 int Factorial(int n);           // Function Declaration
5 void main ()
6 {
7     int n,F;
8     cout<<"Factorial of: ";
9     cin>>n;
10    F=Factorial(n);             //Function calling
11    cout<<F;
12 }
13
14 int Factorial(int n)           // Function Definition
15 {
16     int Fac=1,i;
17     for(i=1; i<=n; i++)
18     Fac=Fac*i;
19     return Fac;
20 }
```

## 5-1-1 *Function Declaration (or Function Prototype)*

- From our example:

```
int Factorial(int n); // Function Declaration
```

- Function declaration is a statement and commonly used at the beginning of the C++ code.
- It consists of function's return type, name and parameter type list.



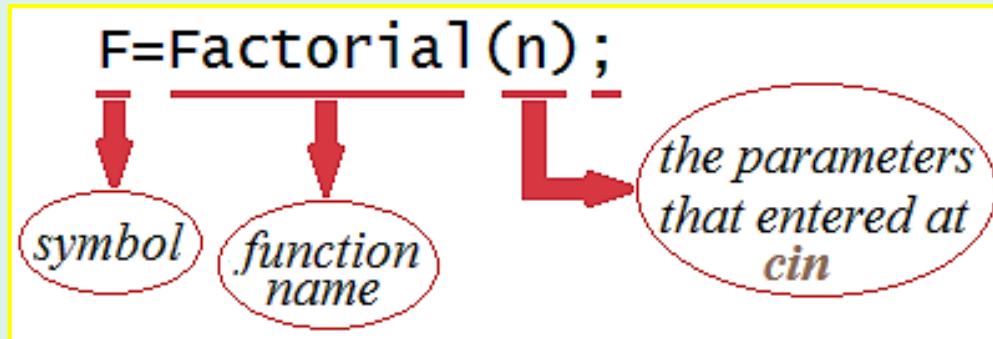
- Return type: the result number is a type of **int** variable.
- Function name: a suggested name express the aim of the mathematical calculations.
- Parameter type list: is the list of parameters (here are **int** type) that the user will enter to the doc screen when ask about it.

## 5-1-2 *Function Calling*

- From our example:

```
F=Factorial(n); //Function Calling
```

- Function calling is a statement ends with a semicolon and put inside the **main ( )** block function.




- Commonly, the function calling statement before it we use *cin* statement but after it we use *cout* statement.

## 5-1-3 Function Definition

- From our example:

```
int Factorial(int n)  Declarator
```

```
{  
  int Fac=1, i;  
  for(i=1; i<=n; i++)  
  Fac=Fac*i;  
  return Fac;  
}
```

 *Function Body*

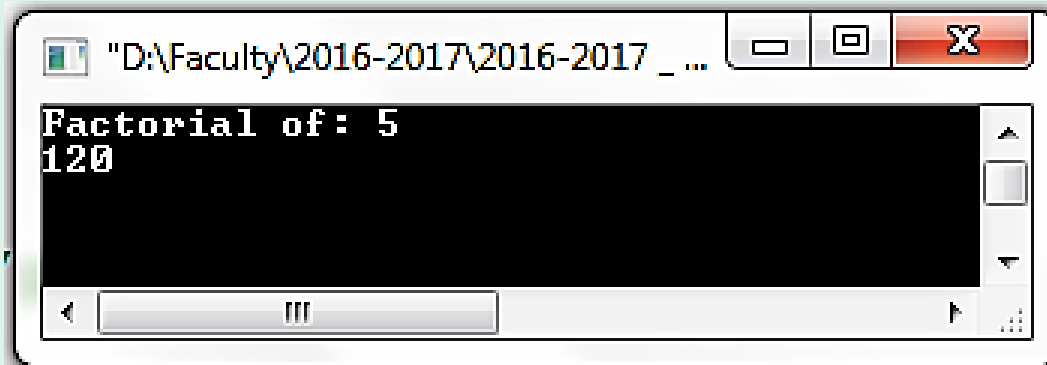
 // Function Definition

- The function definition consists of declarator followed by the function body (which is the actual code of the function).



- FacFunc.cpp

```
1 //FacFunc.cpp
2 // calculate factorial using functions
3 #include<iostream.h>
4 int Factorial(int n); // Function Declaration
5 void main ()
6 {
7     int n,F;
8     cout<<"Factorial of: ";
9     cin>>n;
10    F=Factorial(n); //Function calling
11    cout<<F;
12 }
13
14 int Factorial(int n) // Function Definition
15 {
16     int Fac=1,i;
17     for(i=1; i<=n; i++)
18     Fac=Fac*i;
19     return Fac;
20 }
```



The screenshot shows a Windows command prompt window with the title bar "D:\Faculty\2016-2017\2016-2017 \_ ...". The window contains the following text:

```
Factorial of: 5
120
```

## 5-2 Function Declaration Types

- There are *four* function declaration types:

### 5-2-1 Receives & Returns

e.g.,

```
int Factorial(int n);           //Receives one value and Return the factorial
int max(int x, int y, int z); //Receives three values and Return the maximum
int min(int, int, int);        //Receives three values and Return the minimum
```

### 5-2-2 Receives & NOT Returns

e.g.,

```
void repchar(char, int); //Receives char and Receives number of repetition
```

### 5-2-3 NOT Receives but Returns

e.g.,

```
int random(void); //NOT Receives any value but Return random value
```

### 5-2-4 NOT Receives & NOT Returns

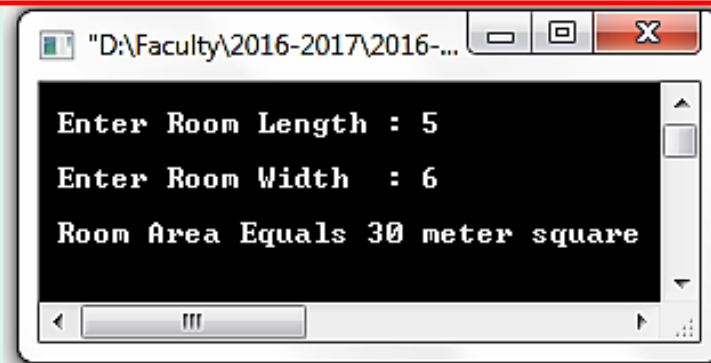
e.g.,

```
void music(void); //NOT Receives any value and NOT Return any value
```

## 5-3 Examples

- Write a program to calculate room area using functions.
- [RoomArea.cpp](#)

```
1 //RoomArea.cpp
2 //Calculate Room Area
3 #include<iostream.h>
4 int RoomArea(int L, int W);
5 void main ( )
6 {
7     int Length,Width,Area;
8     cout<<"\n Enter Room Length : "; cin>>Length;
9     cout<<"\n Enter Room Width  : "; cin>>Width;
10    Area=RoomArea(Length,Width);
11    cout<<"\n Room Area Equals "<<Area<<" meter square"<<endl;
12 }
13 int RoomArea(int L, int W)
14 {
15     int A;
16     A=L*W;
17     return A;
18 }
```

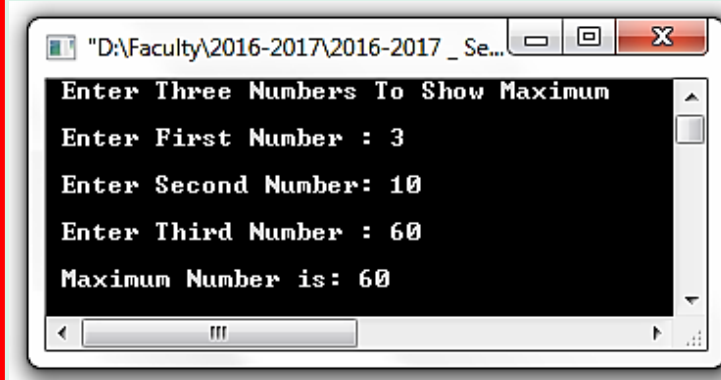


```
"D:\Faculty\2016-2017\2016-..."
Enter Room Length : 5
Enter Room Width  : 6
Room Area Equals 30 meter square
```

## 5-3 Examples ...

- Write a program to show maximum of three input numbers using functions.
- [MaxFunc.cpp](#)

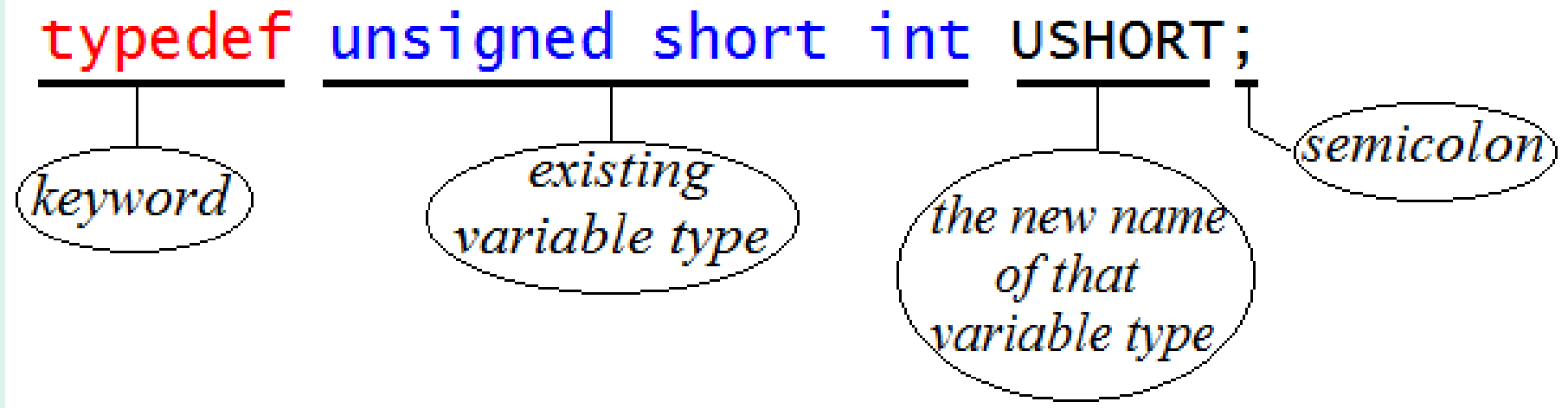
```
1 //MaxFunc.cpp
2 //Show the maximum number
3 #include<iostream.h>
4 int Maximum(int x, int y, int z);
5 void main ( )
6 {
7     int First,Second,Third,Max;
8     cout<<" Enter Three Numbers To Show Maximum "<<endl;
9     cout<<"\n Enter First Number : "; cin>>First;
10    cout<<"\n Enter Second Number: "; cin>>Second;
11    cout<<"\n Enter Third Number : "; cin>>Third;
12    Max=Maximum(First,Second,Third);
13    cout<<"\n Maximum Number is: "<<Max<<endl;
14 }
15 int Maximum(int x,int y,int z)
16 {
17     if(x>y && x>z)
18     return x;
19     else if (y>z)
20     return y;
21     else
22     return z;
23 }
```



```
"D:\Faculty\2016-2017\2016-2017 _ Se...
Enter Three Numbers To Show Maximum
Enter First Number : 3
Enter Second Number: 10
Enter Third Number : 60
Maximum Number is: 60
```

## 5-4 Keyword (**typedef**)

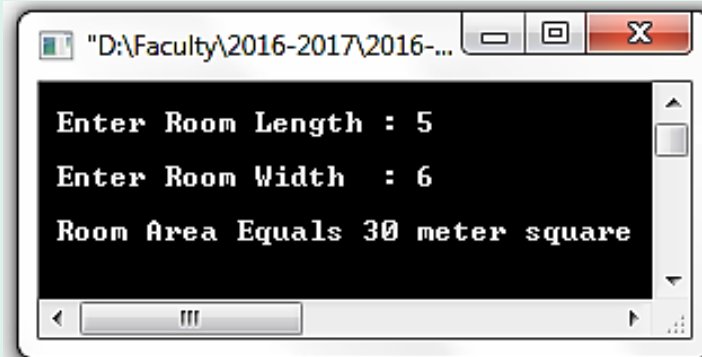
- It is become tedious and a source of errors to keep writing **unsigned short int**. C++ enables you to create an abbreviation for this phrase by using the keyword **typedef**, which stands for type definition. The keyword **typedef** is used as follows:



## 5-4 Keyword (**typedef**) ...

- Write a program to calculate room area using functions (also use the keyword **typedef**)
- RoomArea2.cpp

```
1 //RoomArea2.cpp
2 //Calculate Room Area
3 #include<iostream.h>
4 typedef unsigned short int Ushort;
5 Ushort RoomArea(int L, int W);
6 void main ( )
7 {
8     Ushort Length,width,Area;
9     cout<<"\n Enter Room Length : "; cin>>Length;
10    cout<<"\n Enter Room width : "; cin>>width;
11    Area=RoomArea(Length,width);
12    cout<<"\n Room Area Equals " <<Area<<" meter square"<<endl;
13 }
14 Ushort RoomArea(int L, int W)
15 {
16     Ushort A;
17     A=L*W;
18     return A;
19 }
```

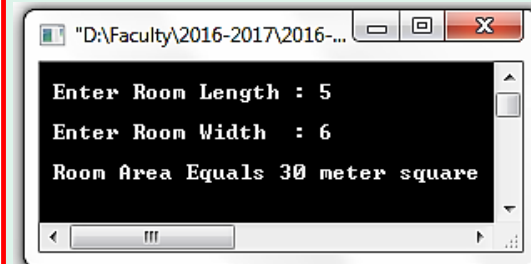


```
"D:\Faculty\2016-2017\2016-..."
Enter Room Length : 5
Enter Room Width : 6
Room Area Equals 30 meter square
```

## 5-5 Local Variables

- “Any variable declared in a certain function is local to that function”.
- For example as in the following code:

```
1 //Local variable.cpp
2 //Calculate Room Area
3 #include<iostream.h>
4 int RoomArea(int,int);
5 void main ( )
6 {
7     int Length,width,Area;
8     cout<<"\n Enter Room Length : "; cin>>Length;
9     cout<<"\n Enter Room Width : "; cin>>width;
10    cout<<"\n Room Area Equals "<<RoomArea(Length,width)<<" m2"<<endl;
11 }
12 int RoomArea(int L, int W)
13 {
14     int A;
15     A=L*W;
16     return A;
17 }
```



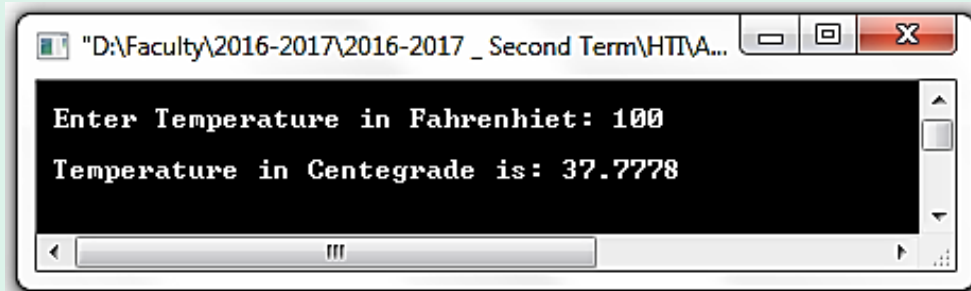
```
"D:\Faculty\2016-2017\2016-..."
Enter Room Length : 5
Enter Room Width : 6
Room Area Equals 30 meter square
```

- From the code:  
Length, Width, Area ..... are local variables to **main ( )**  
L, W, A ..... are local variables to **RoomArea( )**

## 5-5 Local Variables ...

- Write a C++ program to convert from degree Fahrenheit to degree Centegrade using function. [  $Cen = (Fer-32)*5/9$  ]
- [FarCen.cpp](#)

```
1 //FarCen.cpp
2 //Convert From Fahrenheit to Centegrade using Functions
3 #include<iostream.h>
4 float convert(float);           //Function Declaration
5 void main ( )
6 {
7     float TempFar, TempCen;
8     cout<<"\n Enter Temperature in Fahrenheit: "; cin>>TempFar;
9     TempCen= convert(TempFar); // Function Calling
10    cout<<"\n Temperature in Centegrade is: "<<TempCen;
11 }
12 float convert(float Far)       //Function Defenition
13 {
14     float Cen;
15     Cen=(Far-32)*5/9;
16     return Cen;
17 }
```



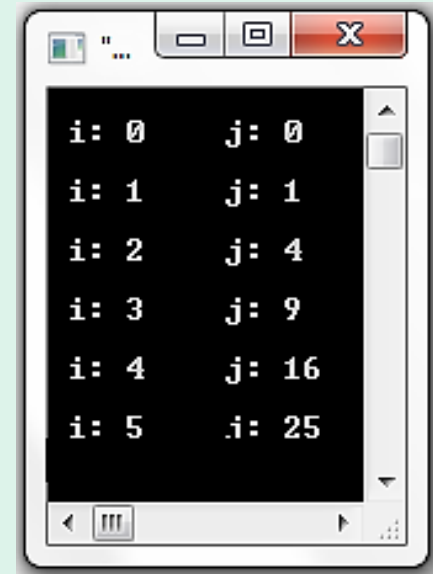
```
"D:\Faculty\2016-2017\2016-2017 _ Second Term\HTI\A...
Enter Temperature in Fahrenheit: 100
Temperature in Centegrade is: 37.7778
```



## 5-6 Scope Variables

- “the variable which is declared in a block of code. It is accessed only in that block of that code”.
- [ScopeVariable.cpp](#)

```
1 //ScopVariable.cpp
2 //demonstrates scope variables
3 #include <iostream.h>
4 void main ( )
5 {
6     int max=5;
7     for(int i=0; i<=max; i++)
8     {
9         int j;
10        j=i*i;
11        cout<<endl<<" i: " <<i<<"\t";
12        cout<<" j: " <<j<<endl;
13    }
14 }
```



```
i: 0    j: 0
i: 1    j: 1
i: 2    j: 4
i: 3    j: 9
i: 4    j: 16
i: 5    j: 25
```

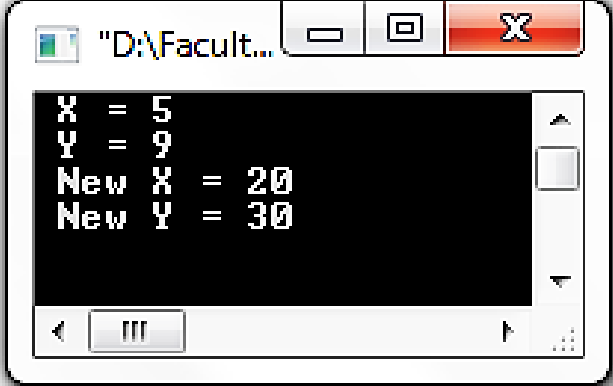
\*The variable max is defined in the scope of the **main ( )** function, but outside the main it is not known”.

\*the variables i, j are defined only in the scope of **for** block.

## 5-7 Global Variables

- “the variable can be accessed any where in all functions and the main function”.
- [GlobalVariable.cpp](#)

```
1 //GlobalVariable.cpp
2 //Demonstrates Global variable
3 #include <iostream.h>
4 void MyFunction( );
5 int X=5, Y=9;
6 void main ( )
7 {
8     cout<<" X = "<<X<<endl;
9     cout<<" Y = "<<Y<<endl;
10    MyFunction( );
11    cout<<" New X = "<<X<<endl;
12    cout<<" New Y = "<<Y<<endl;
13 }
14 void MyFunction( )
15 {
16     X=20;
17     Y=30;
18 }
```



```
X = 5
Y = 9
New X = 20
New Y = 30
```

- The variables X, Y are defined in any where in the previous program.

## **5-8 Special Topics About Functions**

**5-8-1 Overloaded (polymorphism) Functions**

**5-8-2 Default Arguments**

**5-8-3 Inline Functions**

**5-8-4 Static Variables**

## 5-8-1 Overloaded (polymorphism) Functions

- Note: **polymorphism**: the word **poly** means many, and the word **morph** means forms.
- C++ enables you to create more than one function with the same name. This is called function overloading. The function must differ in number of parameter, types of parameters or both. Here is an example;

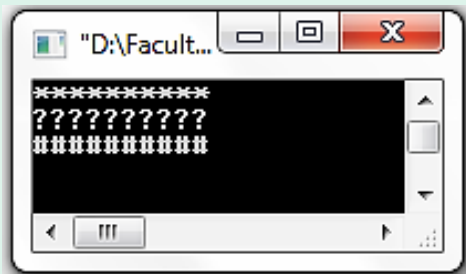
```
int MyFunction(int, int);  
int MyFunction(long);  
int MyFunction(float, float);
```

- The function MyFunction ( ) is overloaded with three different parameter lists.
- The return types can be the same or different in overloaded function.
- You should note that a two functions with the same name and parameter list, but different return types, generate *a compiler error*.

# 5-8-1 Overloaded (polymorphism) Functions ...

- Here is a program to illustrate overloaded functions.
- [Overloaded.cpp](#)

```
1 //Overloaded.cpp
2 //Demonstrates Overloaded functions
3 #include <iostream.h>
4 void repchar( );
5 void repchar(char);
6 void repchar(char,int);
7 void main ( )
8 {
9     repchar( );
10    repchar('?');
11    repchar('#',10);
12 }
13 void repchar( )
14 {
15     for (int i=0;i<10;i++)
16         cout<<"*";
17     cout<<endl;
18 }
19 void repchar(char ch)
20 {
21     for (int i=0;i<10;i++)
22         cout<<ch;
23     cout<<endl;
24 }
25 void repchar(char ch,int n)
26 {
27     for (int i=0;i<n;i++)
28         cout<<ch;
29     cout<<endl;
30 }
```



\*In the function call: repchar( )

the compiler chooses the first function to execute because it has no argument.  
the program displays a line of 10 times of \*.

\*In the function call: repchar('?')

the compiler chooses the second function to execute because it has one argument of type **char**.

the program displays a line of 10 times of ?.

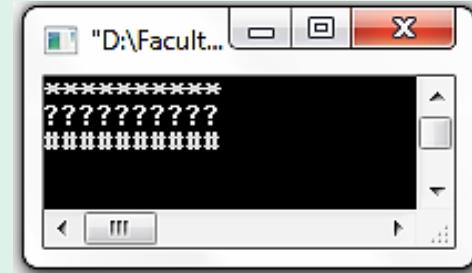
\*In the function call: repchar('#',10)

the compiler chooses the third function to execute because it has two arguments (one of type **char** and the other of type **int**).  
the program displays a line of 10 times of #.  
Argument means a piece of data (for example, **int** value)

## 5-8-2 Default Arguments

- If no data is passed to the function during *calling function*, the function variables take the default values.
- [Default.cpp](#)

```
1 //Default.cpp
2 //Demonstrates Default Arguments
3 #include <iostream.h>
4 void repchar(char='*',int=10); //Prototype with
5                               //Default Arguments
6 void main ( )
7 {
8     repchar( );           //Really Prints 10 *
9     repchar('?');       //Really Prints 10 ?
10    repchar('#',10);     //Really Prints 10 #
11 }
12 void repchar(char ch,int n)
13 {
14     for (int i=0;i<n;i++)
15         cout<<ch;
16         cout<<endl;
17 }
```



- In the 1<sup>st</sup> call: repchar( ) ..... The compiler finds *two* arguments missing and takes them equal to the default values.
- In the 2<sup>nd</sup> call: repchar('?') ..... The compiler finds *one* argument missing (it assumes that the last argument which is the missed). So it takes the default value of 10.
- In the 3<sup>rd</sup> call: repchar('#',10) ..... The compiler finds *no* arguments are missing. So the default values are not be used.

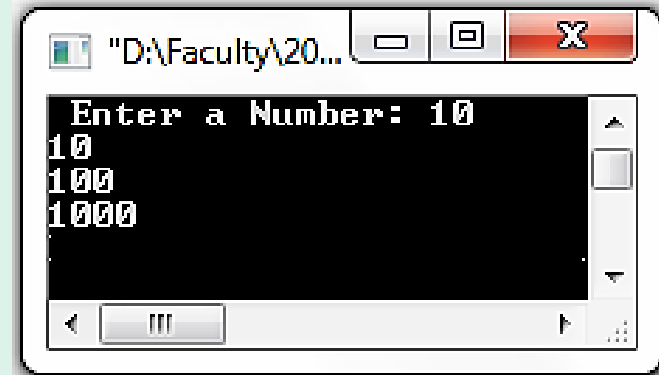
### 5-8-3 Inline Functions

- The compiler makes one copy of the function in the RAM. We call the function many times, it calls the copy of RAM and execute it, then returns the data to the **main** ( ) function, and may be called another times.
- It takes small space of memory but large time to executed and repeated.
- If the function is very large we make it normal (one copy) in RAM. But if it is very small we make it Inline function (many copies) in RAM to save time of program running neglecting the space of RAM.

## 5-8-3 Inline Functions ...

- Here is a program to demonstrate the Inline function.
- [InlineFunc.cpp](#)

```
1 //InlineFunc.cpp
2 //Demonstrates Inline Function
3 #include <iostream.h>
4 inline int large(int num); //Function is declared
5                               //as Inline Function
6 void main ( )
7 {
8     int n;
9     cout<<" Enter a Number: "; cin>>n;
10    n=large(n); //First calling
11    cout<<n<<endl;
12    n=large(n); //Second calling
13    cout<<n*n<<endl;
14    n=large(n); //Third calling
15    cout<<n*n*n<<endl;
16 }
17 int large(int num)
18 {
19     return num;
20 }
```



- The function **large ( )** is declared as **inline function** in the statement: **inline int large (int num)**; it takes **int** value and returns **int** value.



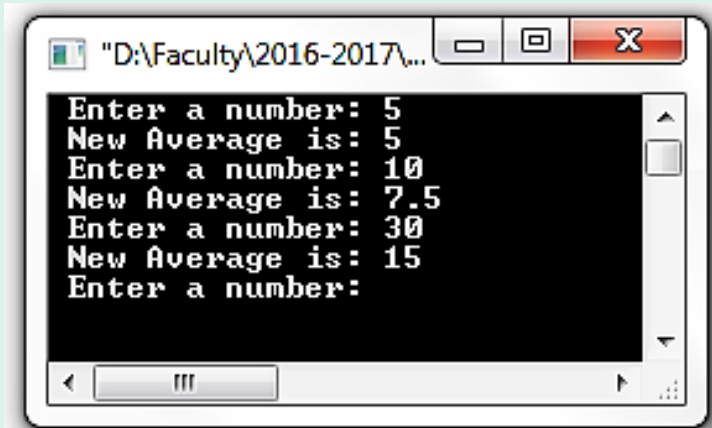
## 5-8-4 Static Variables

- Sometimes we need variables which keep in existence after the function returns. Thus, a static variable is only visible inside the function in which it is defined but it remains in existence for the life of the program.
- *i.e.: The static variables in a certain function remains after the function is called (not initialized again). If it is called another time the old values still exist.*
- Static variables are used when it is necessary for a function to remember a value from one call to another.
- Note: *static variable has the visibility of a local variable but the lifetime of a global variable.*

# 5-8-4 Static Variables ..

## • [StaticVariable.cpp](#)

```
1 //staticVariable.cpp
2 //Demonstrates Static Variable
3 #include <iostream.h>
4 float getavg(float); //Function Prototype
5 void main ( )
6 {
7     float data, avg;
8     data=1;
9     while (data!=0)
10    {
11        cout<<" Enter a number: "; cin>>data;
12        avg=getavg(data);
13        cout<<" New Average is: "<<avg<<endl;
14    }
15 }
16 float getavg(float newdata)
17 {
18     static float total=0; //static variable is initialized
19                          //only once per program
20     static int count=0; //static variable is initialized
21                        //only once per program
22     count++;
23     total+=newdata;
24     return(total/count);
25 }
```



```
"D:\Faculty\2016-2017\...
Enter a number: 5
New Average is: 5
Enter a number: 10
New Average is: 7.5
Enter a number: 30
New Average is: 15
Enter a number:
```

### Analysis of the code:

5 → count=0+1  
total=0+5  
result= 5/1= 5

10 → count=(0+1)+1  
total=(0+5)+10  
result= 15/2= 7.5

30 → count=((0+1)+1)+1  
total=((0+5)+10)+30  
result= 45/3= 15

⋮