Lecture 2:-

Functions

Introduction

A function groups a number of program statements into a unit and gives it a name. This unit can then be invoked from other parts of the program. The most important reason to use functions is to aid in the conceptual organization of a program. Another reason to use functions is to reduce program size. Any sequence of instructions that appears in a program more than once is a candidate for being made into a function. The function's code is stored in only one place in memory, even though the function is executed many times in the course of the program.

User-Defined C++ Functions

Although C++ is shipped with a lot of standard functions, these functions are not enough for all users, therefore, C++ provides its users with a way to define their own functions (or user-defined function).

General formula for writing function

```
Type function_name (parameter1,parameter2, ....)
{
        statement 1;
        statement 2;
        statement 3;
        .
        .
        .
        return value
}
```

Where:

• Type is the data type specifier of the data returned by the function.

• Name is the identifier by which it will be possible to call the function.

• Parameters: Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.

• statements is the function's body. It is a block of statements surrounded by braces { }.

Here you have the first function example:

Ex:- write program to add two numbers by using function

```
#include <iostream>
#include <conio>
int addition (int a, int b)
{
int r;
r=a+b;
return (r);
}
main ()
{
int z;
z = addition (5,3);
cout << "The result is " << z;
getch();
}
```

We can see how the main function begins by declaring the variable z of type int. Right after that, we see a call to a function called addition. Paying attention we will be able to see the similarity between the structure of the call to the function and the declaration of the function itself some code lines above:

```
int addition (int a, int b)
                 ↑      ↑
z = addition (   5  ,   3   );
```

The parameters and arguments have a clear correspondence. Within the main function we called to addition passing two values: 5 and 3, that correspond to the int a and int b parameters declared for function addition.

At the point at which the function is called from within main, the control is lost by main and passed to function addition. The value of both arguments passed in the call (5 and 3) are copied to the local variables int a and int b within the function. Function addition declares another local variable (int r), and by means of the expression r=a+b, it assigns to r the result of a plus b. Because the actual parameters passed for a and b are 5 and 3 respectively, the result is 8.

The following line of code:

```
return (r);
```

finalizes function addition, and returns the control back to the function that called it in the first place (in this case, main). At this moment the program follows it regular course from the same point at which it was interrupted by the call to addition. But additionally, because the return statement in function addition specified a value: the content of variable r (return (r);), which at that moment had a value of 8. This value becomes the value of evaluating the function call.

```
int addition (int a, int b)
 8
z = addition (  5  ,   3  );
```

So being the value returned by a function the value given to the function call itself when it is evaluated, the variable z will be set to the value returned by addition (5, 3), that is 8. To explain it another way, you can imagine that the call to a function (addition (5,3)) is literally replaced by the value it returns (8).

The following line of code in main is:

```
cout << "The result is " << z;
```

That, as you may already expect, produces the printing of the result on the screen.

Ex:- program to convert temperatures from fahrenheit to celsius, using functions

sol

```
#include <iostream>
#include <conio>
float Convert (float TempFer)
{
float TempCel;
TempCel = (TempFer - 32) * 5 / 9;
return (TempCel);
}
main()
{
float TempFer;
float TempCel;
cout << "Please enter the temperature in Fahrenheit: ";
cin >> TempFer;
```

```
TempCel = Convert(TempFer);
cout << "\nHere's the temperature in Celsius: ";
cout << TempCel << endl;
getch();
}
```

EX:-Write a function to find the largest integer among three integers entered by the user in the main function

```
#include <iostream>
#include <conio>
int max(int y1, int y2, int y3)
{
int big;
big=y1;
if (y2>big) big=y2;
if (y3>big) big=y3;
return (big);
}
void main( )
{
int largest,x1,x2,x3;
cout<<"Enter 3 integer numbers:";
cin>>x1>>x2>>x3;
largest=max(x1,x2,x3);
cout<<largest;
getch();
 }
```

Ex:- Write C++ program using function to calculate the average of two numbers entered by the user in the main program.

```
#include <iostream>
#include <conio>
float aver (int x1, int x2)
{
float z;
z = ( x1 + x2) / 2.0;
return ( z);
}
void main( )
{
float x;
int num1,num2;
cout << "Enter 2 positive number \n";
cin >> num1 >> num2;
x = aver (num1, num2);
cout << x;
getch();
 }
```

Ex:- Write C++ program, using function, to find the summation of the following series:

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

```
#include <iostream>
#include <conio>
int summation ( int x)
```

```
{
int sum = 0;
for(int i=1;i<=x;++i)
sum =sum+ i * i ;

return (sum);
}
void main ( )
{
int n ,s;
cout << "enter positive number";
cin >> n;
s = summation ( n );
cout << "sum is: " << s << endl;

getch();
 }
```

Functions with no type. The use of void.

If you remember the syntax of a function declaration:

type name (argument1, argument2 ...) statement

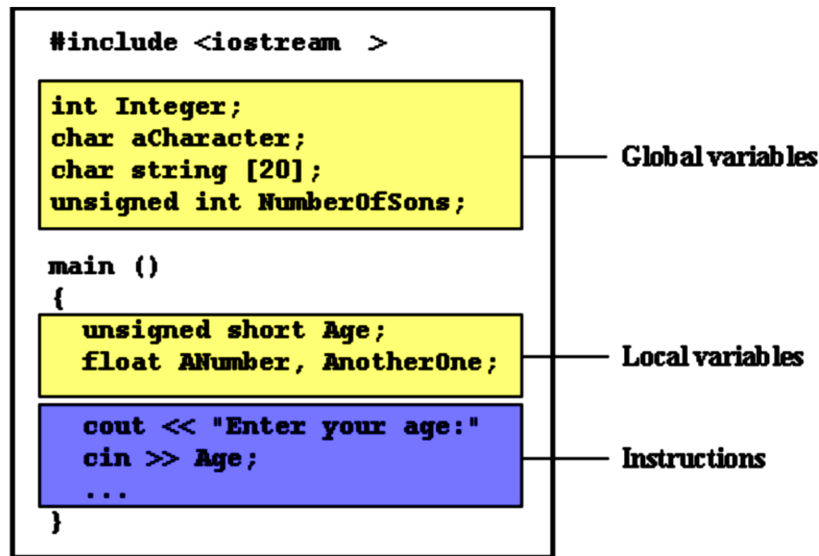
you will see that the declaration begins with a type, that is the type of the function itself (i.e., the type of the datum that will be returned by the function with the return statement). But what if we want to return no value? Imagine that we want to make a function just to show a message on the screen. We do not need it to return any value.

In this case we should use the void type specifier for the function. This is a special specifier that indicates absence of type.

```
#include <iostream>
#include <conio>
void printmessage ()
{
cout << "I'm a Iraqi!";
}
main ()
{
printmessage ();
getch();
}
```

Scope of variables

The scope of variables declared within a function or any other inner block is only their own function or their own block and cannot be used outside of them. For example, in the previous example it would have been impossible to use the variables a, b or r directly in function main since they were variables local to function addition. Also, it would have been impossible to use the variable z directly within function addition, since this was a variable local to the function main.



Therefore, the scope of local variables is limited to the same block level in which they are declared. Nevertheless, we also have the possibility to declare global variables; These are visible from any point of the code, inside and outside all functions. In order to declare global variables you simply have to declare the variable outside any function or block; that means, directly in the body of the program.

Ex:-Write a program to find the sum of the given two numbers using the global variable declaration

```
#include <iostream>
#include <conio>
int x;
int y;
int sum( )
{
int s;
s = x + y;
cout<<"x="<<x<<endl;
```

```
cout<<"y="<<y<<endl;
cout<<"s="<<s<<endl;
}
void main()
{
cin >> x >> y;
cout << endl;
sum();
getch();
 }
```

Ex:-Write a program to find the square for **0** to **n** where **n** is an integer number using a function

```
#include <iostream>
#include <conio>
void square (int n)
{
float value;
value = n*n;
cout<<" i= " << n <<'\t'<< "square = " <<value <<endl;
}
main ()
{
int max;
cout<<"Enter a value for n ? \n";
cin>> max;
for (int i=0;i<= max; ++i)
```

```
square(i);
getch();
}
```

Ex:- Write a program to find the sum of a given non-negative integer of **n** numbers using a **function**

sum =1+2+3+4 ...n

```
#include <iostream>
#include <conio>
int sum (int n)
{
int v=0;
for (int i=1;i<=n;++i)
v = v+i;
return (v);
}
main ()
{
int n;
cout << "Enter any integer number" << endl;
cin >> n;
cout << "v = " << n << "and its sum =" << sum(n);
getch();
}
```

Passing Parameters:

There are two main methods for passing parameters to a program:

1) Passing by value, and 2) passing by reference.

A- Passing by Value:

When parameters are passed by value, a copy of the parameters value is taken from the calling function and passed to the called function. The original variables inside the calling function, regardless of changes made by the function to it are parameters will not change. All the pervious examples used this method.

B- Passing by Reference:

When parameters are passed by reference their addresses are copied to the corresponding arguments in the called function, instead of copying their values. Thus pointers are usually used in function arguments list to receive passed references.

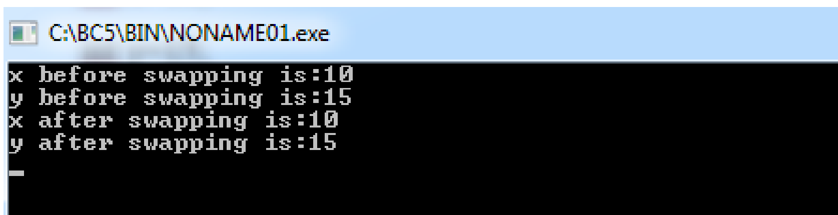
This method is more efficient and provides higher execution speed than the call by value method, but call by value is more direct and easy to use.

The following program illustrates passing parameter by Value.

```
#include <iostream>
#include <conio>
void swap(int a,int b)
{
int t;
t=a;
a=b;
b=t;
}
void main( )
```

```
{
int x=10;
int y=15;
cout<<"x before swapping is:"<<x<<"\n";
cout<<"y before swapping is:"<<y<<"\n";
swap(x,y);
cout<<"x after swapping is:"<<x<<"\n";
cout<<"y after swapping is:"<<y<<"\n";
 getch();
 }
```

Ans

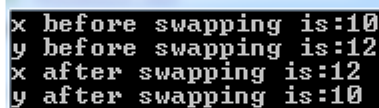

C:\BC5\BIN\NONAME01.exe
```
x before swapping is:10
y before swapping is:15
x after swapping is:10
y after swapping is:15
```

The following program illustrates passing parameter by Reference

```
#include <iostream>
#include <conio>
void swap(int& x,int& y)
{
  int t;
  t=x;
  x=y;
  y=t;
}
void main()
{
```

```
 int x=10,y=12;
cout<<"x before swapping is:"<<x<<"\n";
cout<<"y before swapping is:"<<y<<"\n";
  swap(x,y);
 cout<<"x after swapping is:"<<x<<"\n";
cout<<"y after swapping is:"<<y<<"\n";
 getch();
}
```

Ans



Recursive Functions:

A function which calls itself directly or indirectly again and again is known as the recursive function. Recursive functions are very useful while constructing the data structures like linked lists, double linked lists, and trees. There is a distinct difference between normal and recursive functions. A normal function will be invoked by the main function whenever the function name is used, where as the recursive function will be invoked by itself directly or indirectly as long as the given condition is satisfied.

Ex :- Write C++ program to find the factorial of n.

```
#include <iostream>
#include <conio>
int factorial (int a)
```

```
{
if (a > 1)
return (a * factorial (a-1));
else
return (1);
}
main ()
{
int number;
cout << "Please type a number: ";
cin >> number;
cout << number << "! = " << factorial (number);
getch();
}
```

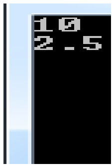
Overloaded functions.

In C++ two different functions can have the same name if their parameter types or number are different. That means that you can give the same name to more than one function if they have either a different number of parameters or different types in their parameters. For example:

Ex :- what is output of the program

```
#include <iostream>
#include <conio>
int operate (int a, int b)
{
return (a*b);
}
float operate (float a, float b)
```

```
{
return (a/b);
}
 main ()
{
int x=5,y=2;
float n=5.0,m=2.0;
cout << operate (x,y);
cout << "\n";
cout << operate (n,m);
cout << "\n";
getch();
}
```



inline functions.

The inline specifier indicates the compiler that inline substitution is preferred to the usual function call mechanism for a specific function. This does not change the behavior of a function itself, but is used to suggest to the compiler that the code generated by the function body is inserted at each point the function is called, instead of being inserted only once and perform a regular call to it, which generally involves some additional overhead in running time.

The format for its declaration is:

inline type name (arguments ...) { instructions ... }

and the call is just like the call to any other function. You do not have to include the inline keyword when calling the function, only in its declaration.

Most compilers already optimize code to generate inline functions when it is more convenient. This specifier only indicates the compiler that inline is preferred for this function.

Ex :- Write a program to find the sum of the two numbers using inline function

```
#include <iostream>
#include <conio>
inline int sum( int x, int y)
{
 return x+y;
}
void main()
{
 cout<<"the result is: "<<sum(1,2);
 getch();
}
```

STANDARD FUNCTIONS:

- **Standard** libraries are used to perform some predefined operations on characters, strings; etc. The standard libraries are invoked using different names such as library functions, built in functions or predefined functions. As the term library function indicates, there are a great many of them, actually they are not part of C++ the language.

- Most of the C++ compilers support the following **standard library** facilities.

1) Operations on characters.

2) Operations on strings.

3) Mathematical operations.

4) Input /output operations.

All the **built in functions** need the header file **<math.h>** in the program.

- These functions are:

1. cosine **double cos(double arg);**

2. sine **double sin(double arg);**

3. tangent **double tan(double arg);**

4. arc cosine **double acos(double arg);**

5. arc sine **double asin(double arg);**

6. arc tangent **double atan(double arg);**

7. arc tangent (y/x) **double atan2(double y, double x);**

8. hyperbolic cosine **double cosh(double arg);**

9. hyperbolic sine **double sinh(double arg);**

10. hyperbolic tangent **double tanh(double arg);**

11. exponential **double exp(double arg);**

12. absolute **double fabs(double num);**

13. natural logarithm **double log(double num);**

14. base 10 logarithm **double log10(double num);**

15. square root **double sqrt(double num);**

16. exponential base **double pow(double base,double exp);**

Example 2: - Write a program to solve the following equation (3x+5y+2x2)

```
#include <iostream>
#include <conio>
#include <math.h>
void main()
{
int x,y;
```

```
cout<<"the equation is: 3x+5y+2x2\n";
cout<<"enter the frist number:\t";
 cin>>x;
cout<<"\nenter the second number:";
cin>>y;
cout<<"\n the result is:\t"<<((3*x)+(5*y)+(2*pow(x,2)) );
getch();
 }
```

H.W

Q1:- Write a C++ program, using function, to find x^y.

Q2:- Write a program to print the message Programming Using C++ for three times using multifunction program.