LECTURE 11

1. Functions:

A function is a set of statements designed to accomplish a particular task. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or (modules). Modules in C++ are called functions.

Functions are very useful to read, write, debug and modify complex programs. They can also be easily incorporated in the main program. In C++, the main() itself is a function that means the main function is invoking the other functions to perform various tasks. The main advantages of using a function are:

- Easy to write a correct small function.
- Easy to read, write, and debug a function.
- Easier to maintain or modify such a function.
- Small functions tend to be self documenting and highly readable.
- It can be called any number of times in any place with different parameters.

2. Defining a Function

A function definition has a name, parentheses pair containing zero or more parameters and a body. For each parameter, there should be a corresponding declaration that occurs before the body. Any parameter not declared is taken to be an integer by default. The general format of the function definition is :

General Form of Function:

```
return-type function-name ( parameters-list )
    {
        (body of function)
        statement1 ;
        statement2 ;
              :
        statement-n ;
        (return something)
    }
```

The type of the function may be int, float, char, etc. It may be declared as type (void), which informs the compiler not to the calling program. For example:

Void function_name (---)

Int function_name (---)

Any variable declared in the body of a function is said to be local to that function. Other variables which are not declared either as arguments or in the function body are considered "global" to the function and must be defined externally. For example

Void square (int a, int b) → a,b are the formal arguments.

Float output (void) → function without formal arguments

Example 1:

```
void printmessage ( )
{
    cout << "University of Technology";
}

void main ( )
{
    printmessage( );
}
```

Example 2:

```
int max (int a, int b)
{
    int c;
    if (a > b)  c = a;
      else   c = b;
    return (c);
}

void main ( )
{
     cout << max (5, 6);
}
```

3. Return Statement:

The keyword return is used to terminate function and return a value to its caller. The return statement may also be used to exit a function without returning a value. The return statement may or may not include an expression. Its general syntax is:

Return;

Return (expression);

The return statements terminate the execution of the function and pass the control back to the calling environment.

Example 1

Write C++ program to calculate the squared value of a number passed from main function. Use this function in a program to calculate the squares of numbers from 1 to 10:

```
#include<iostream.h>
int square ( int y )
{
        int z;
        z = y * y;
        return ( z );
}

void main( )
{
        int x;
        for ( x=1; x <= 10; x++ )
            cout << square ( x ) << endl;
}
```

Example 2

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

```
#include<iostream.h>
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;
}
```

Example 3

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.h>
int summation ( int x)
{
        int i = 1, sum = 0;
        while ( i <= x )
        {
                sum += i * i ;
                i++;
        }
        return (sum);
}
void main ( )
```

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

Example 4

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

```
#include <iostream.h>
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;
}
```

Example 5

Write program in C++, using function, presentation for logic gates (**AND, OR ,NAND, X-OR,NOT**) by in A,B enter from user.

```
#include<iostream.h>
 void ANDF(int,int);
 void ORF(int,int);
 void XORF(int,int);
 void NOTF(int);

void main()
{ char s;int a,b;
cout<<"Enter the value A,B :";
cin>>a>>b;
```

```
cout<<"Enter the select value \n";
cout<<"\ta--(AND gate)\n\to--(OR gate)\n\tx--(X-OR)\n\tn--(NOT
gate)\n\te--(<<EXIT>> :";
cin>>s;
switch(s)
        {
        case 'a':ANDF(a,b);break;
        case 'o':ORF(a,b);break;
        case 'x':XORF(a,b);break;
        case 'n':NOTF(a);cout<<" ";NOTF(b);break;
        case 'e':break;
        deafult:cout<<":bad choose";

}

}
void ANDF(int a,int b)
 {
 cout<<(a&&b);
 }
 void ORF(int a,int b)
 {
 cout<<(a||b);
 }
 void XORF(int a,int b)
 {
 cout<<(a^b);
 }
 void NOTF(int a)
 {
 cout<<(!a);
 }
```

4. 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.

Example 6:

The following program illustrates passing parameter by reference.

```
#include <iostream.h>
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";        }
```