FUNCTIONS IN C++

2.1 Introduction:-

Functions play an important role in C program development. Dividing a program into functions is one of the major principles of top-down structured programming. Another advantage of using functions is that it is possible to reduce the size of a program by calling and using them at different places in the program.

2.2 Declaring Functions: -

The general form for the C++ style of declaring functions is:

return type function_ name (typed parameter list);

Ex:

float volume (int x , float y , float Z) ;

Or

float volume (int , float , float) ;

Remember the following rules when declaring C++ functions:-

1- The return type of the C++ function appears before the function name.

2- It the parameter list is empty we used the key word "void" to explicitly state that there are no parameters.

3- You must declare each parameter explicitly even that these parameter has the same type.

4- The body of a C++ function is enclosed in braces ({}) there is no semicolon after the closing brace.

5- C++ supports passing arguments either , by value or by reference.
Example:

```
Void swap (int &a, int &b )              // uses reference
{
int    t=a ;
       a=b;
       b=t;
}
```

Now, if m and n are two integer variables, then the function-call

swap (m,n); // will exchange the values of M and n using their aliases a & b .

The passing arguments by reference, the function is working with its own arguments, it is actually working on the original data. call passes arguments by value . The called function creates a new set of variables and copies the values of arguments into them, the function dose not have access to the actual variables in the calling program, and can only work on the copies of values.

6- C++ supports local constants data type and variable although these data items can appear in nested block statements. C++ dose not support nested function.
7- The return keyword returns the functions value.
8- If the functions return type is void, you do not have to use the return keyword.

Note: C++ dictates that you either declare or define a function before you use it. Declaring a function commonly called “prototype”.

Ex:

```
        // prototype the function square

double sqr (double);
main ( )
{
  cout<<”5^2=”<<sqr(5)<<”\n”;
  return 0;
 }
  double sqr (double a)
   {  return  a*a;}
```

2.3 Inline function: -

C++ enable you to use inline functions that expand into their statements. Thus, inline functions offer faster execution time especially helpful where speed is critical at the cost of expanding the code.
The inline function can be defined as follows:

Inline return type function_ name (type parameter list)

Ex

```
Inline double cube (double a )
{  return (a*a*a);   }
```

The above inline function can be invoked by statements like.

c=cube (3.0);
d=cube (2.5+1.5);

Some of the situations where inline expansion may not work are:

1- For functions returning values, if a loop, a switch or a goto exists.
2- For functions not returning values, if a return statement exists.
3- If functions contain static variables.
4- If inline function are recursive.

Ex

```
# include <iostream.h>
# include <stdio.h>
inline float mul (float x, float y )                  // inline function .
{     return (x*y);   }

inline double div (double p , double q )       // inline function
{     return(p/q);    }

main( )
{
   float a=12.343;
   float b=9.82   ;
   cout << mul (a,b)<< "\n" ;
   cout <<div (a,b)<< "\n" ;
}
```

The output of above program is

```
        121.227898
         1.257128
```

2.4 Default Arguments:-

C++ allows us to call a function without specifying all its arguments. In Such cases, the function assigns default value to the parameter which does not have a matching argument in the function call. Default values are specified when the function is declared.

Ex:

Prototype (i.e. function declaration) with default value.

```
float amount (float principal, int period, float rate =1.5);
```

The above prototype declares a default value of 1.5 to the argument **rate**. A subsequent function calls like.

```
value = amount (5000,7);      // one argument missing
```

Passes the value of 5000 to **principal** and 7 to **period** and them lets the function use default value of 1.5 for **rate**.

Default arguments are useful in situations where some arguments always have the same value. For instance, bank interest may remain the same for all customers for a particular period of deposit. It also provides a grater flexibility to the programmers.

//////////////////////////////// Default Arguments ////////////////////////////////

```
#include <iostream.h>
#include <stdio.h>
main( )
{
float amount;
float value (float p , int n, float  r = 0.15 );              // prototype
void printline (char ch = '*', int len = 40 );             // prototype
printline ( );                                    // uses default values for arguments
amount = value (5000.00,5);
cout<<``\n Final Value = ``<<amount <<``\n\n``;
printline (`=` ) ;
}

     /* ……..… functions definitions……*/
float value (float p, int n, float r )
{
    int year = l;
    float sum = p;
    while (year <=n)
   {       sum =sum * (l+r);
           year = year + l;
   }
     return (sum);
}
void printline (char ch , int len )
{          for ((int i = l ; i<= len ; i++ )   printf (" %c" , ch );
          printf ("\n" );
}
```

********* The output of above program is **********

****

Final Value = 10056.786133

==========