13/7/19.

Unit - III Functions in C++

* Functions in C++ :-

Function play an important role in program developement. Dividing the program into functions is one of the major principle of top-down structure programing.

Another advantage of using fun^n is that it is possible to reduce the size of program by calling and using them at different places in the program

In C++, more features are added to functions to make them reliable and flexible. C++ functions can be overloaded to make it perform different tasks depending on the aarguments passed through it.

* The main() function :-

The main() function is the starting point for the execution of the program. Every C++ program is defination of main() function.

The main() function ~~writtens~~ returns a value of type integer to the operating System. Hence it is necessary define the fun^n explicitly as .

```
int main ()        OR    void main ()
 {                        {
   ____                     ____
   ____                     ____
                            ____
   return(0);
 }                        }
```

Function prototyping :-

Function prototyping is one of the major improvements added to C++ functions. The prototype describes the function interface to the compiler by giving details, such as number and type of argument and the type of ~~writ~~ return value.

With function prototyping, a template is used when declaring and defining function. When a function is called the compiler uses the template to ensure that proper arguments are passed and return value is treated correctly.

Following is the form of function prototype during declaration.

Return type function name (argu list);

ex :-

① float volume (int x , int y , int z);

e.g

② float volume (float x , float y , float z);

OR

float Volume (int , float , float);

In general we can include or exclude the variable name in argument list of prototypes. Each variable must be declared independently.

In the function defination, names are required for the argument because argument must be reference inside the function.

```
float Volume (int a , float b , float c)
  {
float v = a * b * c ;
        return v ;
  }
```

The fun^n can be called or invo as follows.

funname (argu value);

ex. volume(10, 10.5 , 15.5);

We can also declared funⁿ with empty argument as follows.

void display (); // funⁿ with no argu.

or

void display (void);

Call By Value

A function can be called by passing value to the function defination The following program illustrates this.

// Function using Call By Value

```
# include < iostream.h>

 int area(int); // funn decloration
 int main()
 {
   int r ;
   Cout<<" Enter radius of circle" ;
   cin>>r ;
   float a = area(r) ;
   Cout<<"\n Area of circle =" << a ;
   return 0 ;
 }

   int area(int x)
   {
     int q = 3.14 * x*x ;
     return (q) ;
   }
```

* Call by Reference

It is possible to pass parameters to the function by reference. When we pass arguments by reference, the formal argument in the called function becomes aliases to the actual argument in the calling fun^n.

This means when the fun^n is working with its own argument, it is actually working on the original data.

For eg. in the following program the two nos are interchange their values

// function using call By reference.

```
# include <iostream.h>
void swap (int &a, int &b)
 int main()
  {
    int x , int y ;
   ~~cout << x~~
    cin >> x >> y;
    Cout << x << y ;
     Swap ( x, y ) ;
      Cout << "\n After swap" ;
      Cout << x << y ;
   }
}
```

```
void swap( int &a, int &b)
{
    int  t = a ;
         a = b;
         b = t;
}
```

15/7/19.

Inline function :-

One of the objective using function is to save some memory Space which becomes appritiable when a function is likely to call many time ().

However every time function is called, it takes lots of extra time in executing series of instruction for task such as jumping to the function, saving register and its address pushing arguments into stack and return into the calling function.

When a function is small, a substential percentage of execution time may be strained in such overheads.

One solution to this problem is c++ with a new feature called inline funn and inline funn is a funn that is expanded in line when it is invoke.

That is the compiler impleses the function called with the corresponding funⁿ hold.

Inline function define as follows.

```
Inline function header.
  {
      ___        / funⁿ body
      ___
      ___
  }
```

e.g.
```
Inline double cuber (double a)
  {
    return (a * a * a);
  }
```

The following program illustrate the use of the Inline funⁿ.

```
// use of inline funⁿ
# include <iostream.h>
inline float mul (float x, float y) // funⁿ declaration & defination
 {
   return (x * y);
 }
  inline float div (float p, float q);
  {
       (p/q);
  }
  Void main ()
      { ~~float a = 12.45;~~
        ~~float b = 3.5;~~
```

Date: / /

```
cout<<" multiplication =" << mul (a,b);
cout << "\nDivision =" << div (a,b) ;
}
```

Default Argument :-

C++ allows us to all a funⁿ without specifing its all arguments in such cases the funⁿ assigns a default value to the parameter which does not have a matching argument in the funⁿ call. defau values are specified when the funⁿ is declared .

e.g. float amount (int principal, int period, float rate = 0.15);

Here, the value of rate is assigned as 0.15 & this is default value for rate variable .

In a subsequent funⁿ called like

amount (500, 7); // one argument missin

passes the value of 500 to principle & 7 to period & when let's the funⁿ use default value for rate as 0.15 .

In another example, if we call the funⁿ as

amount (1000, 5, 0.12) // no missing argu

Here, it passes an explicit value 0.12 to rate variable of funⁿ .

17/7/19

Const argument :- In C++ an argument to funⁿ can be declared as const as shown below .

$\ulcorner\rightarrow$ fun^n name $\qquad\qquad \ulcorner\rightarrow$ pointer / variable

```
int strlen (const char * P);
int length (const string & S);
```

The colifier const test the compiler that the fun^n should not modify the argument. The compiler will generate an error when this condition is voileted. This type of declaration is significant only when we pass argument by reference or pointer.

fun^n overloading :- Overloading refers to the use of same thing for different purposes. C++ also permites overloading of fun^n.

This is we can use the same fun^n name to create fun^n that perform a variety of different task. This is known as fun^n polymorphism in OOP.

Using the concept of fun^n overloading we can design a family of fun^n with one fun^n name but with different argument list. The fun^n perform different operations depending on the argument list in the fun^n call. The correct fun^n to be envoked is determine by checking the number & type of argument.

For.eg :- Overload the fun^n add() which handles different type of data as shown below.

```
int add (int a, int b); // Proto type 1.
int add (int a, int b, int c); // Proto type2
```

double add (double x , double y); // prototype 3
double add (int p , double q); // prototype 4.
double add (double p , int q); // prototype 5
(for function call)
// function call
cout << add (5,10); // uses proto type 1.
cout << add (5,10,15); // uses proto type 2
cout << add (5,15.012); // uses proto type 4
cout << add (12.7, 5.5); // uses prototype 3.
cout << add (27.75 , 10); // uses prototype 5.

A funⁿ call first matches the prototype having the same no. and type of argument & then call the appropriate funⁿ. A Best match must be unique with following steps the funⁿ select the prototype.
1st - The compiler first tries to find the the an exact match.
2nd - If the exact match not found, the compiler uses the integral promosion to the actual argument.
~~argu~~
for eg ① char to int
② float to double.

* Write a program to calculate volume of cylinder, cube, rectangle using the conce of funⁿ overloading.

```
// Function overloading
# include < iostream.h >
int volume (int); // prototype declaration
double volume (double, int);
long volume ( long, int, int);
void main ()
  {
   cout << " Volume of cube is = ";
   cout << " volume (5);
   cout << " volume of cylinder = " << volume
                                        (2.5, 8);
  cout<<" volume of rectangle = " << volume
  }                                   (100L, 5, 5);
   int volume (int 5)
     {
      return (5 * 5 * 5);
     }
      double volume ( doubleh, int r )
      {
       return (3.14 * h * r * r);
      }
       long volume ( long l, int b,
       {                         int h)
        return ( l * b * h);
       }
```

Scanned by CamScanner

* Liabrary Function:-

Liabrary function which are avilable with the compilar inside the header file.

These Functions not required to define we can directly use them in the program by Calling them For that the respected header File must be included in the program.

For eg:-
To use the Function clrscr() or getch() it is nessessary to add <iomap.h> to use the string function in the program it is nessessary to use that File <String.h> etc.

— o —