UNIT -1.1 PRINCIPLES OF OOP,TOKENS,LOOPING

Index

- The main method
- Call by reference
- Inline Function
- Default argument
- Const argument
- Function overloading

The main method

- main() function. It is the entry point of a program execution.

- **Syntax:**
- Return type main()
- {
  - Body of the main function
- }**Or as a command line argument :**

- Return type main([int argc, char *argv[ ],[char ** envp]])
- {
  - Body of the main function
- }

The main method

- According to the above format the **return type of the**
- **main() function must be either void or int.**
- Here, return type specifies the status of the program termination.
- The main() function can also takes arguments from command
- prompt to.
- It is known as command line arguments.
- argc specifies the total number at argument. (Argument Count)
- It is the argument counter.
- Its value is always positive.
- Argv represents the argument vector(array).
- It holds pointer to the argument passed from the command line.
- **argv[] is one kind of an array so it holds the data in following manner:**
- argv[0] = pointer to the name of the executable program.
- Argv[1], argv[2]..... argv[n] = pointers to argument strings

The main method

- #include<iostream.h>
#include<conio.h>
- int main(int argc, char *argv[])
- {
- clrscr();
- int i;
- cout<<endl<<"Total arguments="<<argc;
cout<<endl<<"Program name is="<<argv[0];
cout<<endl<<"Other Arguments are\n\n"; for(i=1;i<argc;i++)
- {
  - cout<<endl<<argv[i];
- }
- cout<<endl<<“Total Number of Argument are : ”<<argc; getch();
- return(0);
- }

- run the above program from **dos shell** and enter following arguments:
- C:\TC\BIN\SOURCE> prog_name.exe hello

Call by reference

- Call by reference means we can call the function by its reference means address of variable.
- A reference as its name, is like alias.
- It refer to the same entity.
- A variable and its reference are tightly attached with each other.So, change in one it will also change in the other.
- When call any function by its reference any modifications made through the formal pointer parameter is also reflected in the actual parameter.
- It has functionality of pass-by-pointer and the syntax of call-by-value.
- In the function declaration parameter are to be received by reference must be preceded by the **& operator and arguments or parameters pass same as call by value.**
- However any modification in the variable in function body directly reflected to the actual parameter.
- With the call by reference we can directly change the value of variable in the user define function because we use the reference of variable.
- In general case we can not change the value of variable permanently.

Call by reference

```
#include<iostream.h>
#include<conio.h>
void swap(int *a,int *b)
{
        int temp=*b;
        *b=*a;
        *a=temp;
}
int main()
{
        int x=o,y=o;
        clrscr();
        cout<<"enter value of a"<<x<<endl;
        cin>>x;
        cout<<"enter value of b"<<y<<endl;
        cin>>y;
        swap(&x,&y);
        cout<<"value of a"<<x<<endl;
        cout<<"value of b"<<y<<endl;
        getch();
        return o;
}
```

Inline Function

- This function is expanded inline at a time of compilation that is a function body is inserted in place of function call and so run time overhead for function linkage is reduced, but executable file size is increase.
- •inline function definition must be known to the compiler before function call occurs.
- •If function body contains loops, goto, switch, or a static variables then such function can not expanded inline.
- •A recursive function also can not be expanded inline.
- •If inline expansion is not possible then compiler ignores the word "inline" and considers it has normal UDF.
- •In short, inline function is very much similar to macro definition with #define.
- **•Syntax :**
- inline return_typefun_name(arguments)
- {
- // body of function
- }

Inline Function

```
#include<iostream.h>
#include<conio.h>
inline int product(int a,int b)
{
   return a*b;
}
int main()
{
        int a=o,b=o;
        clrscr();
        cout<<"enter value a"<<a<<endl;
        cin>>a;
        cout <<"enter value b"<<b<<endl;
        cin>>b;
        cout <<"the product of a and b="<<product(a,b)<<endl;
        cout <<"the product of a and b="<<product(a,b)<<endl;
        cout <<"the product of a and b="<<product(a,b)<<endl;
        cout<<"the product of a and b="<<product(a,b)<<endl;
        getch();
        return o;
}
```

Default argument

- We can provide default values for function arguments in function definition or in function declaration and so if one or more arguments are mission in a function call then it takes its default value.
- We can assign default values for argument in the order from right

```
#include <iostream>
int sum(int x, int y, int z = 0, int w = 0)
{
return (x + y + z + w);
}
void main()
{
cout << sum(10, 15, 25, 30) << endl;
}
```

Const arguement

- The keyword const specifies that the value of variable will not change throughout the program.
- If anyone attempt to after the value of variable defined with this qualifier an
  - error can be created.
- A function can also take an argument as a const. which is specifies no any
  - modification on the value.
- Const Arguments means value of arguments can not change.
- **Example :**

```
void fun(const int n) //n value is constant now
{
    //n=5; not possible
    cout<<n;
}
main()
{
    fun(43);
}
```

Function Overloading

- Overloading means use of same name / symbol to perform different work.
- Function overloading means function overloading, multiple functions can have the same name with different parameters.
- Function overloading means the use of same function to perform different action.
- Function overloading is also called function polymorphism.
- Poly means many, and morph means form: a polymorphic function is many-formed.
- In other word, the function is known as overloaded function if any other function with the same name is defined.
- This overloaded functions must differ either in number of arguments or in there data types.
- At the time of function call depending upon the number of actual arguments & their data types, the appropriate function definition will be executed.

Function Overloading

- **Rules of Function Overloading in C++ :**

1. The functions must have the same name
2. The functions must have different types of parameters.
3. The functions must have a different set of parameters.
4. The functions must have a different sequence of parameters.

- **Examples :**
  - void test(int x, int y);
  - void test(int x, int y, int z);
  - void test(int x, float y, char z);
  - void test(int a, float b);

Function Overloading

- #include<iostream.h> void fun(int a)
- {
  - cout<<a<<endl;
- }
- void fun(char f[ ])
- {
  - cout<<f<<endl;
- }
- void fun(char c)
- {
  - cout<<c<<endl;
- }
- void main()
- {
  - fun(12);
  - fun("hello");