C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

Functions / User Defined Functions

- It is a set of instructions written to gather as a block to complete specific functionality.
- Function can be reused.
- It is a subprogram written to reduce complexity of source code
- Function may or may not return value.
- Function may or may not take argument
- Function can return only one value at time
- Function is building block of good top-down, structured code function as a "black box“
- **Writing function helps to**
  - improve readability of source code
  - helps to reuse code
  - reduces complexity
- **Types of Functions**
  - Library Functions
  - User Defined Functions

User Defined Functions

- **Function declaration / Prototype / Function Signature**

  <return type> <functionName> ([<arg type>...]);

- **Function Definition**

```
  <return type> < functionName > ([<arg type> <identifier>...])
    {
              //function body
    }
```

- **Function Call**

  <location> = < functionName >(<arg value/address>);

Inline Function

- C++ provides a keyword *inline* that makes the function as inline function.
- Inline functions get replaced by compiler at its call statement. It ensures faster execution of function just like macros.
- Advantage of inline functions over macros: inline functions are type-safe.
- Inline is a request made to compiler.
- If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

When to use Inline function?

- We can use Inline function as per our needs.

- We can use the inline function when performance is needed.

- We can use the inline function over macros.

- We prefer to use the inline keyword outside the class with the function definition to hide implementation details of the function.

Function Overloading

- Functions with same name and different signature are called as overloaded functions.
- Return type is not considered for function overloading.
- Function call is resolved according to types of arguments passed.
- Function overloading is possible due to name mangling done by the C++ compiler (Name mangling process , mangled name)
- Differ in number of input arguments
- Differ in data type of input arguments
- Differ at least in the sequence of the input arguments

- Example :
    - int sum(int a, int b) { return a+b; }
    - float sum(float a, float b) { return a+b; }
    - int sum(int a, int b, int c) { return a+b+c;;

Default Arguments

- In C++, functions may have arguments with the default values. Passing these arguments while calling a function is optional.
- A default argument is a default value provided for a function parameter/argument.
- If the user does not supply an explicit argument for a parameter with a default argument, the default value will be used.
- If such argument is not passed, then its default value is considered. Otherwise arguments are treated as normal arguments.
- Default arguments should be given in right to left order.
- int sum (int a, int b, int c=0, int d=0) {

```
		return a + b + c + d;
}
```

- The above function may be called as
  - Res=sum(10,20);
  - Res=sum(10,20,40);
  - Res=sum(10,30,40,50);

Modular Approach

- "/usr/include" directory is called standard directory for header files.
- It contains all the standard header files of C/C++
- If we include header file in angular bracket (e.g #include<filename.h>) then preprocessor try to locate and load header file from standard directory only(/usr/include).
- If we include header file in double quotes (e.g #include“filename.h") then preprocessor try to locate and load header file first from current project directory if not found then it try to locate and load from standard directory.

Header Guard

```
#ifndef HEADER_FILE_NAME_H_
#define HEADER_FILE_NAME_H_
//TODO : Type declaration here
#endif
```

Dynamic Memory Allocation

- If we want to allocate memory dynamically then we should use new operator and to deallocate that memory we should use delete operator.
- If pointer contains, address of deallocated memory then such pointer is called dangling pointer.
- When we allocate space in memory, and if we loose pointer to reach to that memory then such wastage of memory is called memory leakage.
- Example :

```
int main()
{
   int *ptr = new int;          //int *ptr = ( int* )::operator new( sizeof( int ) * 1 );
   *ptr = 125;        //Dereferencing
   cout<<"Value     :          "<<*ptr<<endl; //Dereferencing
   delete ptr;        //::operator delete( ptr );
   ptr = NULL;
   return 0;
}
```

Thank You