

C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com



Namespace

- To prevent name conflicts/ collision / ambiguity in large projects
- to group/organize functionally equivalent / related types together.
- If we want to access value of global variable then we should use scope resolution operator (::)
- We can not instantiate namespace.
- It is designed to avoid name ambiguity and grouping related types.
- If we want to define namespace then we should use **namespace** keyword.
- We can not define namespace inside function/class.
- If name of the namespaces are same then name of members must be different.
- We can not define main function inside namespace.
- Namespace can contain:
 1. Variable
 2. Function
 3. Types[structure/union/class]
 4. Enum
 5. Nested Namespace

Note :

- If we define member without namespace then it is considered as member of global namespace.
- If we want to access members of namespace frequently then we should use using directive.



Scope Resolution Operator (::)

- :: operator is used to bind a member with some class or namespace.
- It can be used to define members outside class.
- Also used to resolve ambiguity.
- It can also be used to access global members.
 - Example :- ::a =10; access global var.
- Scope resolution Operator is used to :
 - to call global functions
 - to define member functions of class outside the class
 - to access members of namespaces



cin and cout

- C++ provides an easier way for input and output.
- Console Output : Monitor
 - iostream is the standard header file of C++ for using cin and cout.
 - cout is external object of ostream class.
 - cout is member of std namespace and std namespace is declared in iostream header file.
 - cout uses insertion operator(<<)
- Console Input : Keyboard
 - cin is an external object of istream class.
 - cin is a member of std namespace and std namespace is declared in header file.
 - cin uses Extraction operator(>>)
- The output:
 - cout << "Hello C++";
- The input:
 - cin >> var;



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;;`



Name Mangling

- C++ supports function overloading, i.e., there can be more than one function with the same name but, different parameters.
- the C++ compiler distinguish between different functions by changing names by adding information about arguments.
- This technique of adding additional information to function names is called Name Mangling.
- C++ standard doesn't specify any particular technique for name mangling, so different compilers may append different information to function names.



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);`



# Class

---

- Building block that binds together data & code.
- Program is divided into different classes
- Class is collection of data member and member function.
- Class represents set/group of such objects which is having common structure and common behavior.
- Class is logical entity.
- Class has
  - Variables (data members)
  - Functions (member functions or methods)
- By default class members are private( not accessible outside class scope)
- Classes are stand-alone components & can be distributed in form of libraries
- Class is blue-print of an object



# Data Members and Member Functions

## Data Members

- Data members of the class are generally made as private to provide the data security.
- The private members cannot be accessed outside the class.
- So these members are always accessed by the member functions.

## Member Functions

- Member functions are generally declared as public members of class.
- Constructor : Initialize Object
- Destructor : De-initialize Object
- Mutators : Modifies state of the object
- Inspectors : Don't Modify state of object
- Facilitator : Provide facility like IO



# Object

- Object is an instance of class.
- Entity that has physical existence, can store data, send and receive message to communicate with other objects.
- An entity, which get space inside memory is called object.
- Object is used to access data members and member function of the class
- Process of creating object from a class is called instantiation

- **Object has**

- Data members (**state** of object)
  - Value stored inside object is called state of the object.
  - Value of data member represent state of the object.
- Member function (**behavior** of object)
  - Set of operation that we perform on object is called behaviour of an object.
  - Member function of class represent behaviour of the object.
  - is how object acts & reacts, when its state is changed & operations are done
  - Operations performed are also known as messages
- Unique address(**identity** of object)
  - Value of any data member, which is used to identify object uniquely is called its identity.
  - If state of object is same the its address can be considered as its identity.



# Few Points to note

---

- Member function do not get space inside object.
- If we create object of the class then only data members get space inside object. Hence size of object is depends on size of all the data members declared inside class.
- Data members get space once per object according to the order of data member declaration.
- Structure of the object is depends on data members declared inside class.
- Member function do not get space per object rather it gets space on code segment and all the objects of same class share single copy of it.
- Member function's of the class defines behaviour of the object.



# Access Specifier

- If we want to control visibility of members of structure/class then we should use access Specifier.
- Defines the accessibility of data member and member functions
- **Access specifiers in C++**
  1. private( - )
  2. protected( # )
  3. public( + )
- 1. Private - Can access inside the same struct/class in which it is declared Generally data members should declared as private. (data security)
- 2. Public - Can access inside the same struct/class in which it is declared as well as inside outside function(like main()). Generally member functions should declared as public.
- 3. Protected - Can access inside the same class in which it is declared as well as inside Derived class.



---

# Thank You

