

C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com



Agenda

- Access Specifier in structure
- Namespace
- Scope resolution Operator
- Cin & Cout
- Inline Function
- Function Overloading



Structure

- Structure is a collection of similar or dissimilar data. It is used to bind logically related data into a single unit.
- This data can be modified by any function to which the structure is passed.
- Thus there is no security provided for the data within a structure.
- This concept is modified by C++ to bind data as well as functions.

Access Specifier

- By default all members in structure are accessible everywhere in the program by dot(.) or arrow(→) operators.
- But such access can be restricted by applying access specifiers
 - private: Accessible only within the struct
 - public: Accessible within & outside struct



Structure in C & C++

struct in c	struct in c ++
we can include only variables into the structure.	we can include the variables as well as the functions in structure.
We need to pass a structure variable by value or by address to the functions.	We don't pass the structure variable to the functions to accept it / display it. The functions inside the struct are called with the variable and DOT operator.
By default all the variables of structure are accessible outside the structure. (using structure variable name)	By default all the members are accessible outside the structure, but we can restrict their access by applying the keywords private /public/ protected.
struct Time t1;	struct Time t1;
AcceptTime(struct Time &t1);	t1.AcceptTime(); //function call



Structure in C & C++

```
struct time {  
    int hr, min, sec;  
};  
void display( struct time *p) {  
    printf("%d:%d:%d", p→hr,  
    p→min, p→sec);  
}  
struct time t;  
display(&t);
```

```
struct time {  
    int hr, min, sec;  
void display(){  
    printf("%d:%d:%d",  
this→hr, this→min, this→sec);  
}  
};  
time t;  
t.display();
```



OOP and POP

OOP (Object Oriented Programming)	POP (Procedural Oriented Programming)
Emphasis on data of the program	Emphasis on steps or algorithm
OOP follows bottom up approach.	OOP follows top down approach.
A program is divided to objects and their interactions. Programs are divide into small data units i.e. classes	A program is divided into funtions and they interacts. Programs are divided into small code units i.e. functions
Objects communicates with each other by passing messeges.	Functions communicate with each other by passing parameters.
Inheritance is supported.	Inheritance is not supported.
Access control is supported via access modifiers. (private/ public/ protected)	No access modifiers are supported.
Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
C++, Java	C , Pascal
It overloads functions, constructors, and operators.	Neither it overload functions nor operators
Classes or function can become a friend of another class with the keyword "friend". Note: " friend " keyword is used only in c++	No concept of friend function.
Concept of virtual function appear during inheritance.	No concept of virtual classes .



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 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;;`



Thank You

