



C++ Programming

Trainer : Pradnyaa S Dindorkar

Email: pradnya@sunbeaminfo.com



we did -

1. access specifiers
2. this pointer
3. Class and object (data member, member functions)
4. Live examples of class and object
5. Types of Member Functions within class



Today's Topics

1. namespace
2. cin and cout
3. complex class
4. Modular Approach
5. Constant
6. References
7. Difference between Pointers and reference



Constructor's member initializer list

- If we want to initialize data members according to users requirement then we should use constructor body.

```
class Test
{
private:
    int num1;
    int num2;
    int num3;

public:
    Test( void )
    {
        this->num1 = 10;
        this->num2 = 20;
        this->num3 = num2;
    }
};
```

- If we want to initialize data member according to order of data member declaration then we can use constructors member initializer list.

```
class Test
{
private:
    int num1;
    int num2;
    int num3;

public:
    Test( void ) : num1( 10 ), num2( 20 ), num3( num2 )
    {}

};
```

Except array we can initialize any member inside constructors member initializer list.



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



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.



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;



Complex class :- Ex = 5+j7

Data member = real , imaginary

member functions = complex()

complex(int r,int i)

acceptComplexNumber()

printComplexNumber()

~complex()



Example Scope Resolution

```
class complex {  
    int real, imag;  
public: complex();  
    void show();  
};
```

complex.h

```
complex::complex() {  
    real = imag = 0;  
}  
void complex::show() {  
    cout<<real<<imag;  
}
```

complex.cpp

```
main()  
{  
    complex obj;  
    obj.show();  
}
```

Program.cpp



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
```



Constant in C++

- We can declare a constant variable that cannot be modified in the app.
- If we do not want to modify value of the variable then const keyword is used.
- constant variable is also called as read only variable.
- The value of such variable should be known at compile time.
- In C++ , Initializing constant variable is mandatory
- `const int i=3; //VALID`
- `Const int val; //Not ok in c++`
- Generally const keyword is used with the argument of function to ensure that the variable cannot be modified within that function.



Constant data member

- Once initialized, if we do not want to modify state of the data member inside any member function of the class including constructor body then we should declare data member constant.
- If we declare data member constant then it is mandatory to initialize it using constructors member initializer list.

```
class Test
{
private:
    const int num1;
public:
    Test( void ) : num1( 10 ) //OK
    {
        //this->num1 = 10; //Not OK
    }
};
```



Const member function

- The member function can be declared as const. In that case object invoking the function cannot be modified within that member function.
- We can not declare global function constant but we can declare member function constant.
- If we do not want to modify state of current object inside member function then we should declare member function as constant.
- `void display() const;`
- Even though normal members cannot be modified in const function, but *mutable* data members are allowed to modify.
- In constant member function, if we want to modify state of non constant data member then we should use **mutable keyword**.
- We can not declare following function constant:
 1. Global Function
 2. Static Member Function
 3. Constructor
 4. Destructor



Const object

- If we don't want to modify state of the object then instead of declaring data member constant, we should declare object constant.
- On non constant object, we can call constant as well as non constant member function.
- On Constant object, we can call only constant member function.



Reference

- Reference is derived data type.
- It alias or another name given to the existing memory location / object.
 - Example : `int a=10; int &r = a;`
 - In above example a is referent variable and r is reference variable.
 - It is mandatory to initialize reference.
- Reference is alias to a variable and cannot be reinitialized to other variable
- When ‘&’ operator is used with reference, it gives address of variable to which it refers.
- Reference can be used as data member of any class



Reference

- We can not create reference to constant value.
 - `int &num2 = 10;` //can not create reference to constant value
- Reference is internally considered as constant pointer hence referent of reference must be variable/object.

```
int main( void )  
{  
    int num1 = 10;  
    int &num2 = num1;  
    cout<<"Num2 : "<<num2<<endl;  
    return 0;  
}
```



pass arguments to function, by value, by address or by reference.

- In C++, we can pass argument to the function using 3 ways:
 1. By Value
 2. By Address
 3. By Reference
- If variable is passed by reference, then any change made in variable within function is reflected in caller function.
- Reference can be argument or return type of any function



Diff between Pointers and reference

Pointer

- It is a variable that points to another variable.
- To access the value of a variable with the help of a pointer, we need to do dereferencing explicitly.
- We can create a pointer to pointer
- We can create a pointer without initialization. Create a NULL pointer.

Eg `int n=5; int* ptr=&n;`

Reference

- It is an alias / secondary name to an already existing memory.
- No need of dereferencing to access a value of a variable with ref.
- We can't create a reference to reference.
- We can't create a ref without initialization NULL ref can't be created.

Eg : `int n=5; int& ref=n;`



Sum function and Copy Constructor

- Copy constructor is a single parameter constructor hence it is considered as parameterized constructor
- Example: sum of two complex number

Complex sum(const Complex &c2)

Copy constructor

Complex c2(c1)

or

Complex c2=c1

C1 $\rightarrow 7 + j 6$

\downarrow \downarrow

C2 $\rightarrow 7 + j 6$

Write a function in complex class
to add 2 complex numbers

C1 $\rightarrow 7+j6$

+

C2 $\rightarrow 3+j2$

C3 $\rightarrow 10+j8$



Q: Which among the following describes a destructor?

- a) A special function that is called to free the resources, acquired by the object.
- b) A special function that is called to delete the class.
- c) A special function that is called anytime to delete an object.
- d) A special function that is called to delete all the objects of a class.



Q: Which among the following describes a destructor?

- a) A special function that is called to free the resources, acquired by the object.
- b) A special function that is called to delete the class.
- c) A special function that is called anytime to delete an object.
- d) A special function that is called to delete all the objects of a class.



Q : What is the role of a constructor in classes?

- a) To modify the data whenever required
- b) To destroy an object
- c) To initialize the data members of an object when it is created
- d) To call private functions from the outer world



Q : What is the role of a constructor in classes?

- a) To modify the data whenever required
- b) To destroy an object
- c) To initialize the data members of an object when it is created
- d) To call private functions from the outer world



Q: What is the general syntax for accessing the namespace variable?

A. namespace::operator

B. namespace,operator

C. namespace#operator

D. namespace\$operator



Q: What is the general syntax for accessing the namespace variable?

A. `namespace::operator`

B. `namespace,operator`

C. `namespace#operator`

D. `namespace$operator`



Q. What is syntax of defining a destructor of class A?

- a) A(){}
b) ~A(){}
c) A::A(){}
d) ~A(){};



Q. What is syntax of defining a destructor of class A?

a) A(){}

b) ~A(){}

c) A::A(){}

d) ~A(){};



Revise dynamic memory allocation (malloc)

Thank You

