C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

Agenda

- Constructor & its Types
- Copy Constructor
- Constructor Members Initializer List
- Destructor
- Constant
  - Constant Variable
  - Constant Data Member
  - Constant Member Function
  - Constant Object
- Reference
- Passing arguments to function by address, value and reference
- Dynamic Memory Allocation

Constructor

- It is a member function of a class which is used to initialize object.
- Constructor has same name as that of class and don’t have any return type.
- Constructor get automatically called when object is created i.e. memory is allocated to object.
- If we don’t write any constructor, compiler provides a default constructor.
- Due to following reasons, constructor is considered as special function of the class:
  1. Its name is same as class name.
  2. It doesn't have any return type.
  3. It is designed to call implicitly.
  4. In the life time of the object , it gets called only once per object and according to order of its declaration.
- We can not call constructor on object, pointer or reference explicitly. It is designed to call implicitly.
- We can not declare constructor static, constant, volatile or virtual. We can declare constructor only inline.
- Constructor overloading means inside a class more than one constructor is defined.
- We can have constructors with
  - No argument : initialize data member to default values
  - One or more arguments : initialize data member to values passed to it
  - Argument of type of object : initialize object by using the values of the data members of the passed object. It is called as copy constructor.

Types of Constructor

- Parameterless constructor
  - also called zero argument constructor or user defined default constructor
  - If we create object without passing argument then parameterless constructor gets called
  - Constructor do not take any parameter
- Parameterized constructor
  - If constructor take parameter then it is called parameterized constructor
  - If we create object, by passing argument then paramterized constructor gets called
- Default constructor
  - If we do not define constructor inside class then compiler generates default constructor for the class.
  - Compiler generated default constructor is parameterless.

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.

Copy Constructor

- Copy constructor is a single parameter constructor hence it is considered as parameterized constructor
- Example:
  - **.Complex sum(const Complex &c2)**

Destructor

- It is a member function of a class which is used to release the resources.
- It is considered as special function of the class
  - Its name is same as class name and always preceds with tild operator(~)
  - It doesnt have return type or doesn't take parameter.
  - It is designed to call implicitly.
- Destructor calling sequence is exactly opposite of constructor calling sequence.
- Destructor is designed to call implicitly.
- If we do not define destructor inside class then compiler generates default destructor for the class.
- Default destructor do not de allocate resources allocated by the programmer. If we want to de allocate it then we should define destructor inside class.

Other Member functions of class Setter & Getter

- **Mutators/setter : modify state of object**
- **inspector/getter : do not change the state of the object**
- **facilitator**

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 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 dont 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 functions
- It is C language feature which is used to create alias for existing data type.
- Using typedef, we can not define new data type rather we can give short name / meaningful name to the existing data type.

Reference

- Reference is derived data type.
- It alias or another name given to the exisiting 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
- **Using typedef we can create alias for class whereas using reference we can create alias for object.**

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;
        //int *const num2 = &num1;
        cout<<"Num2 : "<<num2<<endl;
        //cout<<"Num2 : "<<*num2<<endl;
        return 0;
}
```

Reference to array:

```
int main( void )
{
	int arr[ 3 ] = { 10, 20, 30 };
	int i;
	int (&arr2)[ 3 ] = arr;
	for(i = 0; i < 3; ++ i)
	    cout<<arr2[ i ]<<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

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