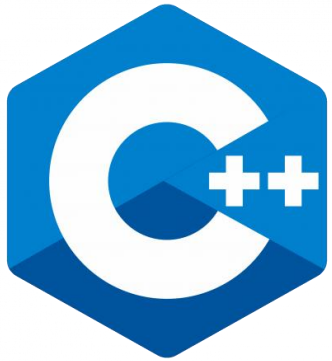


C++ Programming

Trainer : Pradnyaa S. Dindorkar

Email: pradnya@sunbeaminfo.com

Static Variable

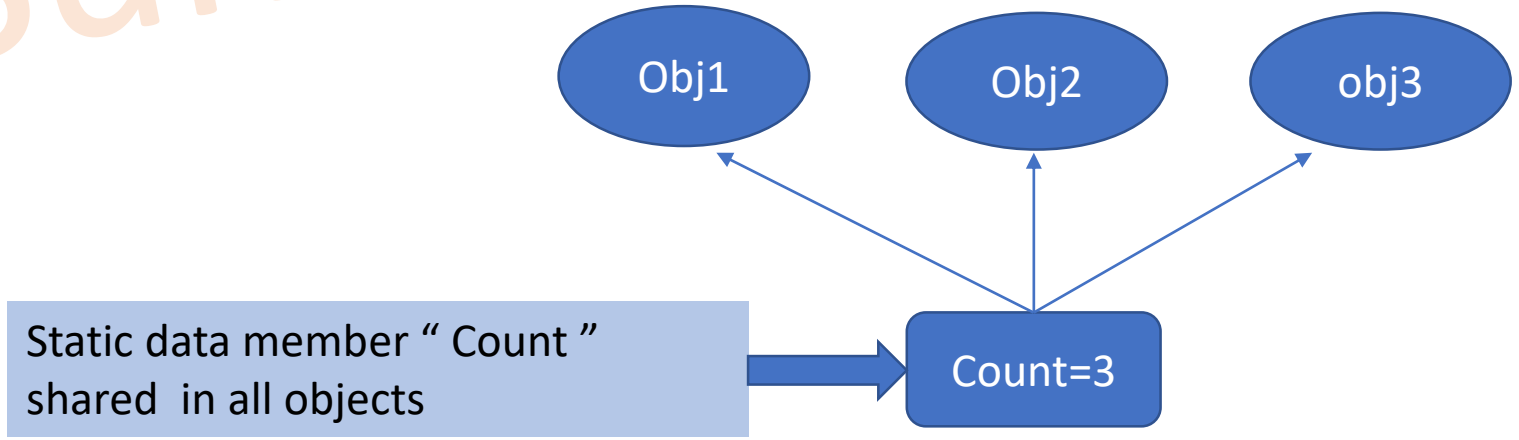
- All the static and global variables get space only once during program loading / before starting execution of main function
- Static variable is also called as shared variable.
- Initialized static and global variable get space on Data segment.
- Default value of static and global variable is zero.
- Static variables are same as global variables but it is having limited scope.

Static Data member

- If we want to share value of data member between all objects of same class then we should declare that data member as static data member.

- It is mandatory to provide global definition of static data member otherwise linker generates error.

- Static data member get space during class loading per class so it is called as **class-level variable**



Design pattern

- A design patterns are **well-proved solution** for solving the specific problem/task.
- Design patterns are programming language independent for solving the common object-oriented design problems.
- Design pattern represents an idea, not a particular implementation.
- Using design patterns you can make your code more flexible, reusable, and maintainable.
- To become a professional software developer, you must know at least some popular solutions (i.e. design patterns) to the coding problems.
- They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.

Examples

1) Singleton Design pattern
2) Factory Design pattern
3) Builder Design pattern
4) Adapter Design pattern
5) Iterator Design pattern

Singleton Design pattern

- Singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance/object. This is useful when exactly one object is needed to coordinate actions across the system.
- For example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly.

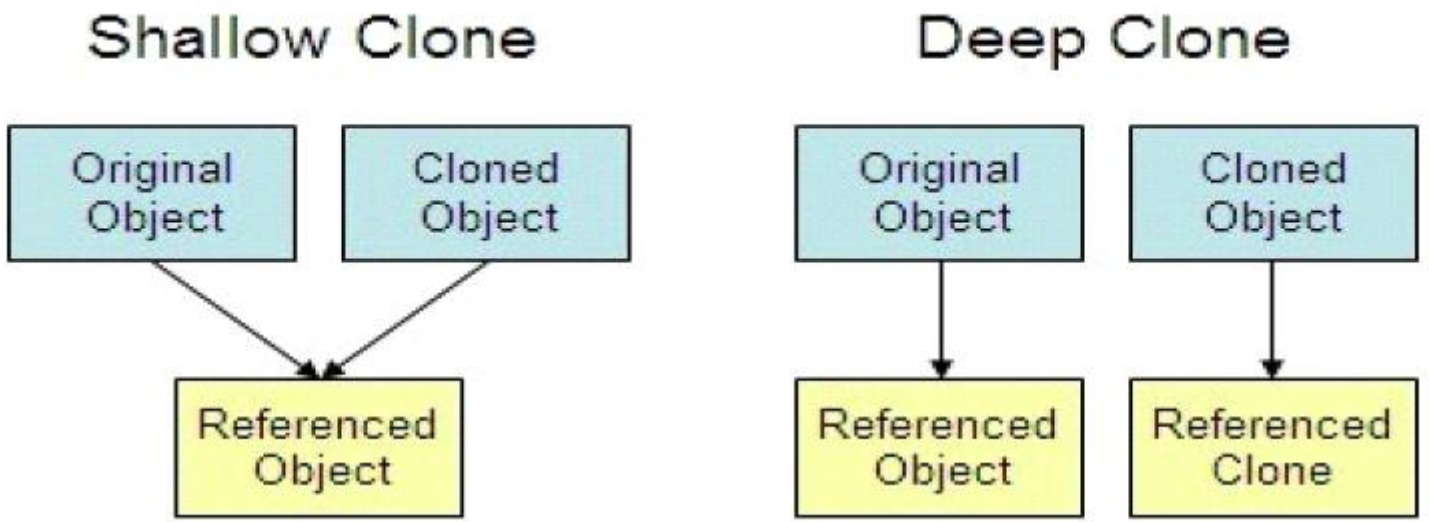
```
class singleton{
    static singleton *ptr;
    singleton(){
        cout<<"in singleton()";
    }
    public:
    static singleton* getObject(){
        cout<<"\n in getObj()";
        if(ptr==NULL)
            ptr=new singleton();
        return ptr;
    }
};
```

```
singleton* singleton::ptr=NULL;

int main()
{
    singleton *ptr=singleton::getObject();
    return 0;
}
```

Object Copying

- In object-oriented programming, “object copying” is a process of creating a copy of an existing object.
- The resulting object is called an object copy or simply copy of the original object.
- Methods of copying:
  - Shallow copy
  - Deep copy



Types of Copy

The copy constructor is used to initialize the new object with the previously created object of the same class.

- **Shallow Copy**
  - The process of copying state of object into another object.
  - It is also called as **bit-wise** copy.
  - When we assign one object to another object at that time copying all the contents from source object to destination object as it is. Such type of copy is called as shallow copy.
  - Compiler by default create a shallow copy. Default copy constructor always create shallow copy.

- **Deep Copy**
  - Deep copy is the process of copying state of the object by modifying some state.
  - It is also called as **member-wise** copy.
  - When class contains at least one data member of pointer type, and class contains user defined destructor then when we assign one object to another object then copy constructor is called.
  - At that time instead of copy base address allocate a new memory for newly created object and then copy contain from memory of source object into memory of destination object. Such type of copy is called as deep copy.

Friend :-

- A non member function of a class which designed to access private data of a class is called friend function.

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend**

```
class MyData
{
private:
        int pin;
        int pass;

public:
        MyData(int pin,int pass);
        void PrintMyAccDetails();
        friend void anyFunction();
};
```

```
void anyFunction()
{
        MyData d1;
        d1.pass=9898;
        d1.pin=9999;
        d1.PrintMyAccDetails();
}
```

C++ is not a pure Object Oriented Language Because

- You can write code without creating a class in C++, and main() is a global function.
- Support primitive data types, e.g., int, char, etc. Instances(variable) of these primitive types are NOT objects.
- C++ provides "Friend" which is absolute corruption to the OO-Principle of encapsulation.

Operator Overloading

- operator is token in C/C++. And it is used to generate expression.
- Types of operator:
  - Unary operator (++,--,&,!,~,sizeof())
  - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
  - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should **overload operator**.
- To overload operator, we should define **operator function.**
- **We can define operator function using 2 ways:**
  - **Using member function**
  - **Using non member function**

Operator Overloading

using member function

- operator function must be member function
- If we want to overload, binary operator using member function then operator function should take only one parameter.
  - Example :

```
c3 = c1 + c2;   //will be called as
                //c3 = c1.operator+( c2 )
```

using non member function

- Operator function must be global function
- If we want to overload binary operator using non member function then operator function should take two parameters.
  - Example :

```
c3 = c1 - c2;  // will be called as
               // c3 = operator-(c1,c2);
```

We can not overloading following operator using member as well as non member function:

1. dot/member selection operator(.)
2. Pointer to member selection operator(.*)
3. Scope resolution operator(::)
4. Ternary/conditional operator(? :)
5. sizeof() operator
6. typeid() operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator

We can not overload following operators using non member function:

- Assignment operator(=)
- Subscript / Index operator([])
- Function Call operator[()]
- Arrow / Dereferencing operator(→)

Overloading Subscript or array index operator []

- The Subscript or Array Index Operator is denoted by '[ ]'.
- This operator is generally used with arrays to retrieve and manipulate the array elements.
- Overloading of [] may be useful when we want to check for index out of bound.
- We must return by reference in function because an expression like "arr[i]" can be used an lvalue.

```
int& operator[](int index)
    {
        if (index >= this->size)
        {
            cout << "Array index out of bound, exiting";
            exit(0);
        }
        return this->ptr[index];
    }
```

Exception Handling

- If we give wrong input to the application then it generates runtime error/exception.
- To handle exception then we should use 3 keywords:
- 1**. try**
  - try is keyword in C++.
  - If we want to inspect exception then we should put statements inside try block/handler.
  - Try block may have multiple catch block but it must have at least one catch block.
- **2. catch**
  - If we want to handle exception then we should use catch block/handler.
  - Single try block may have multiple catch block.
  - Catch block can handle exception thrown from try block only.
  - A catch block, which can handle any type of exception is called generic catch block / catch-all handler.
  - For each type of exception, we can write specific catch block or we can write single catch block which can handle all types of exception. A catch block which can handle all type of exception is called generic catch block.
- **3. throw**
  - throw is keyword in C++.
  - If we want to generate exception explicitly then we should use throw keyword.
  - "throw statement" is a jump statement.
  - To generate new exception, we should use throw keyword. Throw statement is jump statement.

Note : For thrown exception, if we do not provide matching catch block then C++ runtime gives call to the std::terminate() function which implicitly gives call to the std::abort() function.

Consider the following code

In this code, int type exception is thrown

but matching catch block is not available.

Even generic catch block is also not

available. Hence program will terminate.

Because , if we throw exception from try block then catch block can handle it. But with the help of function we can throw exception from outside of the try block.

```
int main( void )
{
	int num1;
	accept_record(num1);
	int num2;
	accept_record(num2);
	try
	{
		if( num2 == 0 )
			throw 0;
		int result = num1 / num2;
		print_record(result);
	}
	catch(char ex )
	{
		cout<<ex<<endl;
	}
	return 0;
}
```

Thank You