Lecture 12

Static Keyword in C++

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following,

1. Static variable in functions
2. Static Class Objects
3. Static member Variable in class
4. Static Methods in class

Static Variables inside Functions

Static variables when used inside function are initialized only once, and then they hold there value even through function calls.

These static variables are stored on static storage area, not in stack.

```
void counter()
{
   static int count=0;
   cout << count++;
}
int main(0
{
   for(int i=0;i<5;i++)
   {
      counter();
   }
}
```

Output

0 1 2 3 4

Let's se the same program's output **without using static** variable.

```
void counter()
{
   int count=0;
   cout << count++;
}

int main(0
{
   for(int i=0;i<5;i++)
   {
      counter();
   }
}
```

Output

0 0 0 0 0

If we do not use static keyword, the variable count, is reinitialized everytime
when counter() function is called, and gets destroyed each time when counter() functions ends.
But, if we make it static, once initialized count will have a scope till the end of main() function and it will carry its value through function calls too.

If you don't initialize a static variable, they are by default initialized to zero.

Static Class Objects

Static keyword works in the same way for class objects too. Objects declared static are allocated storage in static storage area, and have scope till the end of program.

Static objects are also initialized using constructors like other normal objects. Assignment to zero, on using static keyword is only for primitive datatypes, not for user defined datatypes.

```
class Abc
{
   int i;
   public:
   Abc()
   {
      i=0;
      cout << "constructor";
```

```
    }
    ~Abc()
    {
        cout << "destructor";
    }

};

void f()
{
    static Abc obj;

}

int main()
{
    int x=0;
    if(x==0)
    {
        f();
    }
    cout << "END";
}
```

Output

constructor END destructor

You must be thinking, why was the destructor not called upon the end of the scope of if condition, where the reference of object obj should get destroyed. This is because object was static, which has scope till the program's lifetime, hence destructor for this object was called when main() function exits.

Static Data Member in Class

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.

Static member variables (data members) are not initialied using constructor, because these are not dependent on object initialization.

Also, it must be initialized explicitly, always outside the class. If not initialized, Linker will give error.

```
class X
{
   public:
     static int i;
   X()
   {
      // construtor
   }
};

int X::i=1;

int main()
{
   X obj;

   cout << obj.i;   // prints value of i
}
```

Output

1

Once the definition for static data member is made, user cannot redefine it. Though, arithmetic operations can be performed on it.

*The exception to the initialization of a static data member inside the class declaration is if the static data member is a **const** of integral or enumeration type. [For example: https://www.geeksforgeeks.org/static-const-vs-define-vs-enum/]

Static Member Functions

These functions work for the class as whole rather than for a particular object of a class.

It can be called using an object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

For example:

```
class X
```

```
{
   public:
   static void f()
   {
      // statement
   }
};

int main()
{
   X::f();   // calling member function directly with class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

static members exist as members of the class rather than as an instance in each object of the class. So, **this** keyword is not available in a static member function. Such functions may access only static data members.

Non-static member functions can access all data members of the class: static and non-static.