Static data members and methods

We looked at the static keyword earlier in the respect of functions. How declaring an identifier as static gives its lifetime scope of the program, such that it retains its value even after successive function calls. But, static has a different meaning when it comes to classes.

Static data-members

Declaring a data member in a class as static gives it class-scope. i.e. The variable no longer remains specific and bound to one particular object instance. Thereafter, changing the value from one instance reflects over all the instances. As an example:

```
#include <bits/stdc++.h>
using namespace std;

class Test {
    public:
        static int x;
};

/*static members need to be
defined outside the class*/
int Test::x = 1;

int main()
{
    Test t1, t2;

    cout << "Access from instance: " << t1.x << endl
         << "Access from Class: " << Test::x << endl;

    Test::x = 5;

    cout << "t1.x: " << t1.x << endl
         << "t2.x: " << t2.x << endl
         << "Test::x: " << Test::x << endl;
```

```
    return 0;
}
```

Output:

```
Access from instance: 1
Access from Class: 1
t1.x: 5
t2.x: 5
Test::x: 5
```

As we can see from the above program, any change made to the member *x*, it gets reflected over both instances *t1* and *t2*. We can also access static members using Class-name and scope-resolution *~Test::x*.

NOTE : When we declare a static variable inside a class, we are just telling the compiler the existence of such a variable. It is treated as a variable with a global scope and is initialized only when the program starts. No memory is allocated at that point. Thus, we can't directly initialize a static data-member along with the declaration. i.e. *static int x = 1;* is reported as a compilation error. We must explicitly initialize it outside the class. Another thing to note is that this initializer statement works regardless of whether we declare the static variable as public, private or protected.

Static member-functions

Static Member Functions are similar to the static data-members implying that they too have class-scope. In addition to that, they are allowed to access only other static fields (data & member). However, they can be called using object instances. (i.e.

*obj.static_method( )*is valid). Some important points regarding static member functions are given below:

- Static member-functions can't access non-static data-members.
- Static member-functions can't access other non-static member-functions.
- Static member-functions have no *this pointer, as it is not associated with any particular instance.
- There is no concept of a static constructor.
- Static member-functions are useful for accessing static data-members which are not declared public.

As an example:

```
#include <bits/stdc++.h>
using namespace std;

class Test {
    private:
        static int x;

    public:
        //set private static integer x
        static void setX(int x) { Test::x = x; }

        //get private static integer x
        static int getX() { return x; }
};

//Initializer statement is valid
//even though variable is declared
//private
int Test::x = 1;

int main()
{
    Test::setX(5);
    cout << Test::getX();

    return 0;
}
```

Output:

```
5
```