STATIC MEMBER VARIABLES

```
class Something{
public:
    int m_value = 1;//normal(automatic)
};
int main() {
    Something first;
    Something second;
    first.m_value = 2;
    cout << first.m_value;  // 2
    cout << second.m_value; // 1
}
```

Member variables of a class can be made static by using the `static` keyword:

```
class Something{
public:
    static int s_value;
};

int Something::s_value = 1; //defines

int main() {
    Something first;
    Something second;
    first.m_value = 2;
    cout << first.s_value;  // 2
    cout << second.s_value; // 2
}
```

Unlike normal member variables, **static member variables** are **shared by all objects of the class**.

Because `s_value` is a static member variable, it is shared between all objects of the class.

Consequently, `first.s_value` is the same variable as `second.s_value`

The above program shows that the value we set using `first` can be accessed using `second`

Static members are not associated with particular class objects:

Although you can access static members through objects of the class, static members exist even if no objects of the class have been instantiated!

Much like global variables, static members are created when the program starts, and destroyed when the program ends.

Consequently, it is better to think of static members as belonging to the class itself, not to the objects of the class.

Because `s_value` exists independently of any class objects, it can be accessed directly using the class name and the scope resolution operator:

```
Something::s_value
```

```
class Something{
public:
    static int s_value;
};

int Something::s_value = 1; //defines

int main() {
    // no object was instantiated
    Something::s_value = 2;
    cout << Something::s_value;  // 2
}
```

s_value is referenced by class name rather than through an object. We are able to access it as **Something::s_value**. This is the preferred method for accessing static members.

Defining and initializing static members:

When we declare a static member variable inside a class, we're telling the compiler about the existence of a static member variable, but not actually defining it (much like a forward declaration).

Because static member variables are not part of the individual class objects (they are treated similarly to global variables, and get initialized when the program starts), you must explicitly define the static member outside of the class, in the global scope.

```
class Something {    // Objects With Unique ID's:
private:
   static int s_idGenerator;
   int m_id;
public:
   Something() { m_id = s_idGenerator++; }
   int getID() const { return m_id; }
};

int Something::s_idGenerator = 1;//no access ctrl

int main() { //now make objects with unique ID's:
   Something first;
   Something second;
   Something third;
   cout << first.getID();  // 1
   cout << second.getID(); // 2
   cout << third.getID();  // 3
}
```