static specifier

Inside a class, declares members not bound to specific instances.

Syntax

`static` *data_member*	(1)
`static` *member_function*	(2)

1) Declares a static data member
2) Declares a static member function.

Explanation

Static members of a class are not associated with the objects of the class: they are independent objects with static storage duration or regular functions defined in namespace scope, only once in the program.

The `static` keyword is only used with the declaration of a static member, inside the class definition, but not with the definition:

```
class X {
    static int n; // declaration (uses 'static')
};
int X::n = 1; // definition (does not use 'static')
```

The declaration inside the class body is not a definition and may declare the member to be of incomplete type (other than `void`), including the type in which the member is declared.

```
struct Foo;
struct S {
   static int a[]; // incomplete type
   static Foo x; // incomplete type
   static S s; // incomplete type (inside its own definition)
};

int S::a[10]; // definition, complete type
struct Foo {};
Foo S::x; // definition, complete type
S S::s; // definition, complete type
```

To refer to a static member n of class T, two forms may be used: qualified name `T::m` or member access expression `e.m` or `e->m`, where e is an expression that evaluates to T or T* respectively. When in the same class scope, the qualification is unnecessary:

```
struct X {
    static void f(); // function declaration
    static int n; // member declaration
};

X g() { return X(); } // some function returning X

void f() {
    X::f();  // qualified name access to static function
    g().f(); // expression member access to static function
}

int X::n = 7; // definition

void X::f() { // definiton
    n = 1; // X::n is accessible as just n in this scope.
}
```

Static members obey the class member access rules (private, protected, public).

Static member functions

Static member functions are not associated with any object. When called, they have no `this` pointer. They cannot be `virtual`, `const`, or `volatile`. They cannot access non-static data members of the class. Address of a static member function may be stored in a regular pointer to function, but not in a pointer to member function.

Static data members

The static member objects are not part of the object. If the static member is declared thread_local(since C++11), there is one such object per thread. Otherwise, there is only one instance of the static member object in the entire program, with static storage duration. The static members exist even if no objects of the class have been defined.

Static data members cannot be `mutable`.

Static data members of a class in namespace scope have external linkage if the class itself has external linkage (i.e. it's not a member of unnamed namespace)

Local classes (classes defined inside functions) and unnamed classes, including member classes of unnamed classes, cannot have static data members.

Constant static members

If a static data member of integral or enumeration type is declared `const` (and not `volatile`), it can be initialized with a brace-or-equal initializer that is a constant expression, right inside the class definition:

```
struct X {
    const static int n = 1;
    const static int m{2}; // since C++11
};
```

If a static data member of `LiteralType` is declared `constexpr`, it can be initialized with a brace-or-equal initializer that is a constant expression inside the class definition:

```
struct X {
    constexpr static int n = 1;
};
```

(since C++11)

If such a member is odr-used, a definition at namespace scope is still required, but it should not have an initializer.

```
struct X {
    const static int n = 1;
};
const int* p = &X::n; // X::n is odr-used
const int X::n;       // ... so a definition is necessary
```

References

- C++11 standard (ISO/IEC 14882:2011):
  - 9.4 Static members [class.static]
- C++98 standard (ISO/IEC 14882:1998):
  - 9.4 Static members [class.static]

See also

- `static` storage specifier