Static Data Member, Functions and Friend Functions

Static Class Members

- Static Data Members
  - Precede a member variable's declaration with static
  - Only one copy of that variable exists
  - All objects of the class will share that variable
  - All static variables are initialized to **zero** before the first object is created
  - It is visible only within class, but its lifetime is the entire program.
- Memory storage
  - Must be defined by using :: to indicate the class ownership
  - Initialized Data Segment or Uninitialized Data Segment called Block Started by Symbols (BSS)

Example-Static Variable

```
#include <iostream>
using namespace std;
class shared {
static int a; // declare a
int b;
public:
void set(int i, int j) {a=i; b=j;}
void show();
} ;
int shared::a; // define a
void shared::show() {
cout << "This is static a: " << a;
cout << "\nThis is non-static b: " << b;
cout << "\n"; }
```

```
int main() {
shared x, y;
x.set(1, 1); // set a to 1
x.show();
y.set(2, 2); // change a to 2
y.show();
x.show(); /* Here, a has been changed for
both x and y because a is shared by both
objects. */
return 0;
}
```

Output:

```
This is static a: 1
This is non-static b: 1
This is static a: 2
This is non-static b: 2
This is static a: 2
This is non-static b: 1
```

Example-Static Variable

- A static member variable exists before any object of its class is created.

```
#include <iostream>
using namespace std;
class shared {
public:
static int a;
} ;
int shared::a; // define a
```

```
int main( ) {
// initialize ‘a’ before creating any objects
shared::a = 99;
cout << "This is initial value of a: " <<
shared::a;
cout << "\n";
shared x;
cout << "This is x.a: " << x.a;
return 0;
}
```

Output:

```
This is initial value of a: 99
This is x.a: 99
```

Static Member Functions

- Member functions may also be declared as static
- Declare it with “static” keyword
- Characteristics:
  - Refer to other static members of the class.
  - May access global functions and data
  - Cannot be both static and a non-static

Example-Static Variable Application

```
#include <iostream>
using namespace std;
class Counter {
public:
static int count;
Counter() { count++; }
~Counter() { count--; }
};
int Counter::count;
void f();
int main(void)
{
Counter o1;
cout << "Objects in existence: ";
cout << Counter::count << "\n";
Counter o2;
cout << "Objects in existence: ";
cout << Counter::count << "\n";
f();
cout << "Objects in existence: ";
cout << Counter::count << "\n";
return 0;
}
void f()
{
Counter temp;
cout << "Objects in existence: ";
cout << Counter::count << "\n";
// temp is destroyed when f() returns
}
```

Output:

```
Objects in existence: 1
Objects in existence: 2
Objects in existence: 3
Objects in existence: 2
```

The usage of global variable can be avoided.
If we have global variables, then the encapsulation will be violated.

Example

```
#include <iostream>
using namespace std;
class static_type {
static int i;
public:
static void init(int x) {i = x;}
void show() {cout << i;}
};
int static_type::i; // define i
int main()
{
// init static data before object creation
static_type::init(100);
static_type x;
x.show(); // displays 100
return 0;
}
```

Output:100

Another Example

```
class X {
int i;
static int j;
public:
X(int ii = 0) : i(ii) {
// Non-static member function can access
// static member function or data:
j = i;
}
int val() const { return i; }
static int incr() {
//! i++; // Error: static member function cannot
access non-static member data
return ++j;
}
};
static int f() {
//! val(); // Error: static member function
cannot access non-static member function
return incr(); // OK -- calls static
}
};
int X::j = 0;
int main() {
X x;
X* xp = &x;
cout<< x.f();
cout <<xp->f() <<endl;
cout <<X::f(); // Only works with static
members
}
```

Output: 1 2 3

Static Member Functions

- No access to “this” pointer
  - Static member function is shared by objects and is per class function
- Cannot be declared as constant
  - Const member function does not allow to modify the object, therefore can not make sense to declare static member function to be made constant.

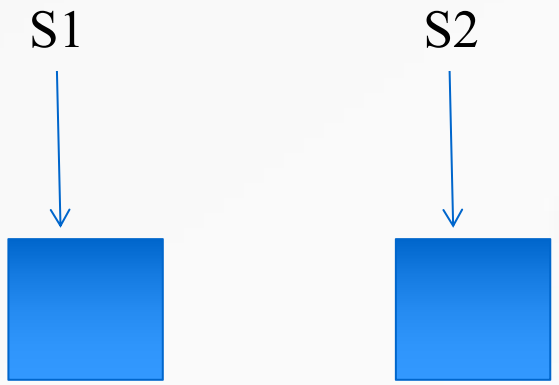
The “this” pointer

- When a member function is called, it is automatically passed an implicit argument that is a pointer to the invoking object (that is, the object on which the function is called).

- This pointer is called “this”.

- Sample S1, S2;
- S1.fun(&S1);
- S2.fun(&S2);

Implicitely passed



Example - this pointer

```
#include <iostream>
using namespace std;
class pwr {
double b;
int e;
double val;
public:
pwr(double base, int exp);
double get_pwr() { return val; }
};
```

```
double get_pwr() { return this->val; }
```

```
pwr::pwr(double base, int exp)
{
b = base;
e = exp;
val = 1;
if (exp==0) return;
for( ; exp>0; exp--) val = val * b;
}
```

```
pwr::pwr(double base, int exp)
{
this->b = base;
this->e = exp;
this->val = 1;
if (exp==0) return;
for( ; exp>0; exp--)
this->val = this->val * this->b;
}
```

```
int main()
{
pwr x(4.0, 2), y(2.5, 1), z(5.7, 0);
cout << x.get_pwr() << " ";
cout << y.get_pwr() << " ";
cout << z.get_pwr() << "\n";
cout<< &x << &y<<&z;
return 0;
}
```

Friend Functions

- Allows a nonmember function access to the private members of a class.
- It has access to all private and protected members of the class for which it is a friend.
- To declare a friend function, include its prototype within the class, preceding it with the keyword friend.
- It does not have an access to this pointer.

Applications of Friend Functions

- Would be useful when you are overloading certain types of operators
- They make the creation of some types of I/O functions easier.
- Two or more classes may contain members that are interrelated relative to other parts of your program

Example

```
#include <iostream>
using namespace std;
class myclass {
int a, b;
public:
friend int sum(myclass x);
void set_ab(int i, int j);
};
void myclass::set_ab(int i, int j)
{
a = i;
b = j;
}
// Note: sum() is not a member function
of any class.
int sum(myclass x) {
/* Because sum() is a friend of myclass,
it can directly access a and b. */
return x.a + x.b;
}
int main()
{
myclass n;
n.set_ab(3, 4);
cout << "The sum is " << sum(n);
return 0;
}
```

Output: The sum is 7

Friend Function Usage

```
#include <iostream>
using namespace std;
const int IDLE = 0;
const int INUSE = 1;
class C2; // forward declaration
class C1 {
// IDLE if off, INUSE if on screen
int status;
public:
void set_status(int state);
friend int idle(C1 a, C2 b);
};
```

```
class C2 {
// IDLE if off, INUSE if on screen
int status;
public:
void set_status(int state);
friend int idle(C1 a, C2 b);
};

void C1::set_status(int state)
{
status = state;
}

void C2::set_status(int state)
{
status = state;
}
```

Friend Function Usage Cont..

```
int idle(C1 a, C2 b)
{
if(a.status || b.status)
return 0;
else return 1;
}
```

```
int main()
{
C1 x;
C2 y;
x.set_status(IDLE);
y.set_status(IDLE);
if(idle(x, y)) cout << "Screen can be used.\n";
else cout << "In use.\n";
x.set_status(INUSE);
if(idle(x, y)) cout << "Screen can be used.\n";
else cout << "In use.\n";
return 0;
}
```

Output:

Screen can be used.
In use.

Using Friend of another Class

```
#include <iostream>
using namespace std;
const int IDLE = 0;
const int INUSE = 1;
class C2; // forward declaration
class C1 {
int status; // IDLE if off, INUSE if on
screen
// ...
public:
void set_status(int state);
int idle(C2 b); // now a member of C1
};
class C2 {
int status; // IDLE if off, INUSE if
on screen
// ...
public:
void set_status(int state);
friend int C1::idle(C2 b);
};
void C1::set_status(int state) {
status = state;
}
void C2::set_status(int state) {
status = state;
}
```

Using Friend of another Class

```
// idle() is member of C1, but friend of C2
int C1::idle(C2 b)
{
if(status || b.status) return 0;
else return 1;
}
```

```
int main() {
C1 x;
C2 y;
x.set_status(IDLE);
y.set_status(IDLE);
if(x.idle(y)) cout << "Screen can be used.\n";
else cout << "In use.\n";
x.set_status(INUSE);
if(x.idle(y)) cout << "Screen can be used.\n";
else cout << "In use.\n";
return 0;
}
```

Restrictions

- A derived class does not inherit friend functions.
- Friend functions may not have a storage-class specifier, i.e., they may not be declared as static or extern.

Friend Classes

- It is possible for one class to be a friend of another class.

- The friend class and all of its member functions have access to the private members defined within the other class.

Example - Friend Classes

```
// Using a friend class.
#include <iostream>
using namespace std;
class TwoValues {
int a;
int b;
public:
TwoValues(int i, int j)
{ a = i; b = j; }
friend class Min;
};
```

```
class Min {
public:
int min(TwoValues x);
};
int Min::min(TwoValues x)
{
return x.a < x.b ? x.a : x.b;
}
```

```
int main()
{
TwoValues ob(10, 20);
Min m;
cout << m.min(ob);
return 0;
}
```

References

- C++: The Complete Reference, 4th Edition by Herbert Schildt , McGraw-Hill
- Teach Yourself C++ 3rd Edition by Herbert Schildt,
- The C+ + Programming Language, Third Edition by Bjarne Stroustrup, Addison Wesley