



# COMP 2012H Honors Object-Oriented Programming and Data Structures

## Topic 15: Static Data Members and Member Functions

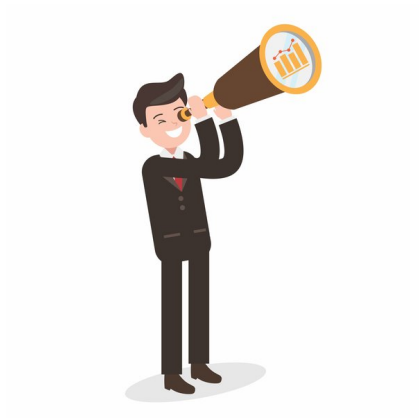
Dr. Desmond Tsoi

Department of Computer Science & Engineering  
The Hong Kong University of Science and Technology  
Hong Kong SAR, China



# Part I

## Static Variables with File/Function Scope



# Static Variables with a File/Function Scope

- **Static variables** are **global variables** which
  - ▶ are created only **once** in a program.
  - ▶ reside on the **static data** region of the loaded program.
  - ▶ have a lifetime across the **entire run** of a program.
  - ▶ still controlled by its **scope**: file, function, class.
  - ▶ if not explicitly initialized, will be **zero-initialized** for basic types (and their arrays) and **default-initialized** for objects.
- **Static variables** in a function
  - ▶ are initialized only **once** regardless how many times the function is called.
  - ▶ **retain** their values across the function calls.
  - ▶ can be accessed **only inside** the function.

## Example: Static Variable with a File Scope

```
#include <iostream> /* afile.cpp */
using namespace std;

int a;
int func();

int main() {
    a = 10;
    cout << a << " " << func() << endl;
    return 0;
}
```

---

```
int a; /* bfile.cpp */
int func() {
    a = 20;
    return a;
}
```

Question: What would happen if we compile the program using the following command?

```
g++ -o output afile.cpp bfile.cpp
```

## Example: Static Variable with a File Scope

```
#include <iostream> /* afile-static.cpp */
using namespace std;

static int a;
int func();

int main() {
    a = 10;
    cout << a << " " << func() << endl;
    return 0;
}
```

---

```
int a; /* bfile.cpp */
int func() {
    a = 20;
    return a;
}
```

Question: What is the output of the program compiled using the following command?

```
g++ -o output afile-static.cpp bfile.cpp
```

## Example: Static Variables with a Function Scope

```
#include <iostream>      /* File: static-var-function.cpp */
using namespace std;

int fibonacci(int n, int& calls)
{
    static int num_calls = 0; // Initialized only once
    calls = ++num_calls;

    if (n <= 0)
        return 0;
    else if (n == 1 || n == 2)
        return 1;
    else
        return fibonacci(n-2, calls) + fibonacci(n-1, calls);
}

int main()
{
    int n; int n_calls;
    cout << "Enter n: "; cin >> n;
    cout << "\n" << "fibonacci(" << n << ") = " << fibonacci(n, n_calls);
    cout << "\n" << "number of fibonacci calls = " << n_calls << endl;
    return 0;
}
```

Question: What is the output?

## Part II

### Static Class Data Members



"You have to study for tests, dummy — you can't just put a memory stick in your ear!"

## Example: Students Study for an Exam By Memorizing

```
#include <iostream>      /* File: student-non-static.h */
#include <string>
// vector is a template class in C++ Standard Template Lib (STL).
// vectors are smart arrays that automatically expand if necessary.
#include <vector>
using namespace std;

class Student
{
private:
    string name;          // Student's name
    vector<string> memory; // Each student has his own memory

public:
    Student(string s) : name(s) { }

    // push_back() is vector's member function that does insertion
    void memorize(string txt) { memory.push_back(txt); }
    void do_exam();
};
```



# How Do Students Take an Exam

```
#include "student-non-static.h" /* File: student-non-static.cpp */

void Student::do_exam()
{
    // empty() is vector's member function to check if it is empty
    if (memory.empty())
        cout << name << ": " << "Huh???" << endl;
    else
    {
        // Like a (generalized) const pointer to vector's elements
        vector<string>::const_iterator p;

        // begin() returns the pointer to the vector's beginning
        // end() returns the pointer to the cell beyond vector's end
        for (p = memory.begin(); p != memory.end(); ++p)
            cout << name << ": " << *p << endl;
    }

    cout << endl;
}
```

# Exam Takes Place Now

```
#include "student-non-static.h" /* File: exam-non-static.cpp */

int main()
{
    Student Jim("Jim");
    Jim.memorize("Data consistency is important");
    Jim.memorize("Copy constructor != operator=");

    Student Steve("Steve");
    Steve.memorize("Overloading is convenient");
    Steve.memorize("Make data members private");
    Steve.memorize("Default constructors have no arguments");

    Student Alan("Alan");

    Jim.do_exam();
    Steve.do_exam();
    Alan.do_exam();
    return 0;
} // Compile: g++ student-non-static.cpp exam-non-static.cpp
```

# Result of an Exam

Jim: Data consistency is important

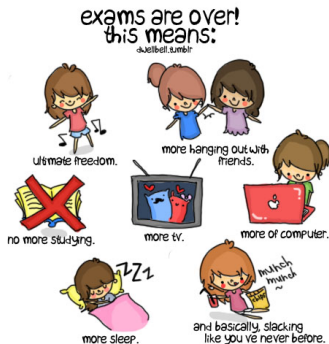
Jim: Copy constructor != operator=

Steve: Overloading is convenient

Steve: Make data members private

Steve: Default constructors have no arguments

Alan: Huh???



# Students Try to Cheat by “Collective Wisdom”

```
#include <iostream>      /* File: student-static.h */
#include <vector>
#include <string>
using namespace std;

class Student
{
private:
    string name;
    static vector<string> memory; // Students share memory!

public:
    Student(string s) : name(s) { }
    void memorize(string txt) { memory.push_back(txt); }
    void do_exam();
};
```

# Students Cheat by Collective Memory

```
#include "student-static.h"          /* File: student-static.cpp */

// Globally define class static data; here, it is
// initialized by calling vector's default constructor
vector<string> Student::memory;

void Student::do_exam()
{
    if (memory.empty())
        cout << name << ": " << "Huh???" << endl;
    else
    {
        vector< string >::const_iterator p;

        for (p = memory.begin(); p != memory.end(); ++p)
            cout << name << ": " << *p << endl;
    }

    cout << endl;
}
```

# Unfair Exam

```
#include "student-static.h" /* File: exam-static.cpp */

int main()
{
    Student Jim("Jim");
    Jim.memorize("Data consistency is important");
    Jim.memorize("Copy constructor != operator=");

    Student Steve("Steve");
    Steve.memorize("Overloading is convenient");
    Steve.memorize("Make data members private");
    Steve.memorize("Default constructors have no arguments");

    Student Alan("Alan");

    Jim.do_exam();
    Steve.do_exam();
    Alan.do_exam();
    return 0;
} // Compile: g++ student-static.cpp exam-static.cpp
```

# Result of Cheating

Here, all students **share** their memories. So even though Alan didn't memorize anything, he can access **all** the knowledge memorized by Jim and Steve.

Jim: Data consistency is important

Jim: Copy constructor != operator=

Jim: Overloading is convenient

Jim: Make data members private

Jim: Default constructors have no arguments

Steve: Data consistency is important

Steve: Copy constructor != operator=

Steve: Overloading is convenient

Steve: Make data members private

Steve: Default constructors have no arguments

Alan: Data consistency is important

Alan: Copy constructor != operator=

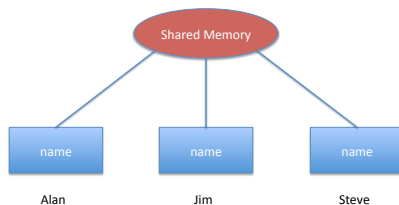
Alan: Overloading is convenient

Alan: Make data members private

Alan: Default constructors have no arguments



# Static Class Data: Summary



- **Static class data** members are actually **global variables** specified by the keyword **static** under the **scope** of a class.
- There is only one **single** copy of a **static variable** in a class, which are **shared** among **all objects** of the class.
- **Static variables** of a class exist even when there are **no** objects of the class; they do **not** take up space inside an object.
- **Static variables cannot** be initialized in the class definition (except for `const int/enum` static data).
- **Static variables** must be **defined outside** the class definition, usually in the class implementation (`.cpp`) file.
- One still has to observe their **access** and **const qualifier**.



# Part III

## Static Class Member Functions/Methods



## Example: Class Clock With Static Member Functions

```
class Clock                                /* File: clock-w-static-fcn.h */
{
    friend ostream& operator<<(ostream& os, const Clock& c)
        { return os << c.hour << " hr. " << c.minute << " min. "; }

public:
    Clock() : hour(0), minute(0) { }

    static Clock HHMM(int hhmm)
        { return Clock(hhmm/100, hhmm%100); }

    static Clock minutes(int m)
        { return Clock(m/60, m%60); }

private:
    int hour, minute;
    Clock(int h, int m) : hour(h), minute(m) { }
};
```

# Class Clock With Static Member Functions — clock-test.cpp

```
#include <iostream>          /* File: test-clock.cpp */
using namespace std;
#include "clock-w-static-fcn.h"

int main()
{
    Clock c1;                // 0:00
    Clock c2 = Clock::HHMM(123); // 1:23
    Clock c3 = Clock::minutes(123); // 2:03

    cout << c1 << endl;
    cout << c2 << endl;
    cout << c3 << endl;

    return 0;
}
```



# Static Member Function / Class Member Function

- Classes may also have **static member functions** or **methods**.
- **Static data member (member functions)** are also called **class data (member functions)**.
- **Static member variables (member functions)** are actually **global** variables (functions) but with a **class scope** and are subject to the **access control** specified by the class developer.
- **Static member functions** can be called in 2 ways:
  1. like a global function by using the class scope operator::.
  2. like a member function of the class using the . operator.
- Still have to observe their access control: **static data member/member functions** may still be **public|protected|private**.

## Static Member Function / Class Member Function ..

**Static member functions** belong to a class, not to a particular object of the class. Therefore, **static member functions** of a class

1. do not have the **implicit this** pointer like regular non-static member functions.
2. may be used even when there are **no** objects of the class!
3. can only make use of **static data members** of the class.
4. **cannot** be **const** nor **virtual** functions.
5. **cannot** be **overloaded** with a non-static member function of the same prototype.

## Example: Class Car — car.h

```
#include <iostream>      /* File: car.h */
using namespace std;

class Car
{
public:
    Car() { ++num_cars; }
    ~Car() { --num_cars; }

    void drive(int km) { total_km += km; }
    static int cars_still_running() { return num_cars; }

private:
    static int num_cars;
    int total_km = 0;
};
```

## Example: Class Car — car.cpp

```
#include "car.h" /* File: test-car.cpp */
int Car::num_cars = 0; // Define + initialize static class member

int main()
{
    cout << Car::cars_still_running() << endl;
    Car vw; vw.drive(1000);
    Car bmw; bmw.drive(10);
    cout << Car::cars_still_running() << endl;

    Car *cp = new Car[100];
    cout << Car::cars_still_running() << endl;

    {
        Car kia; kia.drive(400);
        cout << Car::cars_still_running() << endl;
    }
    cout << Car::cars_still_running() << endl;
    delete [] cp;
    cout << Car::cars_still_running() << endl; return 0;
}
```

# Static Data Members and Member Function / Method

Compare a class **Car** with a factory:

- The **Car** objects are the products made by the factory.
- Data members are **data** on the products, and member functions are **services** provided by the objects.
- **Static class data/member functions** are data/services provided by the factory.
- Even if **no** object of this type has been created, we can access the **static class data/member functions**.

- A regular member function of **Car**, such as

```
void drive(int km) { total_km += km; }
```

after **compilation** becomes:

```
void Car::drive(Car* this, int km) { this->total_km+=km; }
```

- On the other hand, a **static member function** of **Car** such as

```
static int cars_still_running() { return num_cars; }
```

after **compilation** becomes:

```
int Car::cars_still_running() { return Car::num_cars; }
```



That's all!

Any questions?

