Object-Oriented Programming and Data Structures

COMP2012: Static Data and Methods

Prof. Brian Mak
Prof. C. K. Tang

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

Part I

Static Class Data

Example: Students Study for an Exam By Memorizing

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

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

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

How Do Students Take an Exam

```
#include <iostream>          /* File: student-non-static.cpp */
#include "student-non-static.h"
using namespace std;

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;
}
```

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( );
}  // 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 <vector>                          /* File: student-static.h */
#include <string>
using std::string;
using std::vector;

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 <iostream>                          /* File: student-static.cpp */
#include"student-static.h"
using namespace std;

vector<string> Student::memory;              // Globally define static data

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( );
}  // Compile: g++ student-static.cpp exam-static.cpp
```

Result of Cheating

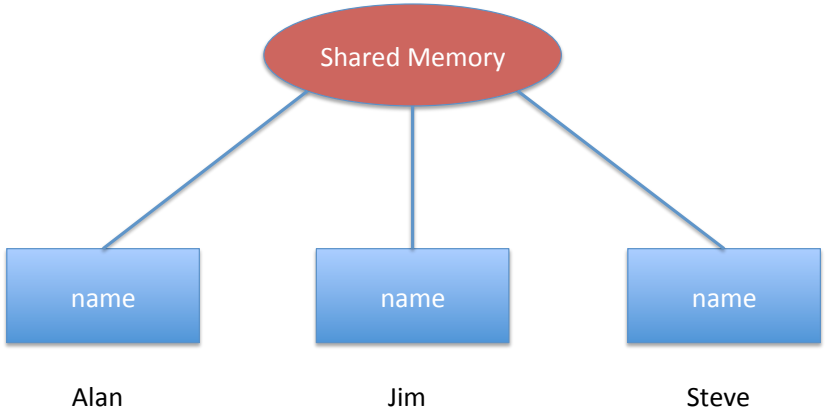
Here, all students share their memory. So even though Alan didn't memorize anything, he can access all the knowledge memorized by Jim and Steve.

```
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
```

Class Static Data: Summary



- Static class data members are really 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 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 int/enum type).
- Static variables must be defined outside the class definition, usually in the class implementation (.cpp) file.
- One still have to observe their access and const qualifier.

Part II

Static Class Methods/Functions

Named Constructors

- C++ constructors have the name of the class.
- Different constructors can only be distinguished if they have different argument types — function overloading.
- E.g., Can't have 2 Clock constructors with an int argument, interpreting either in HHMM format or # minutes after midnight.

```
class Clock {                                   /* File: incorrect-clock.h */
  public:
    Clock( ) : hour(0), minute(0) { }
    Clock(int mins) : hour(mins/60), minute(mins%60) { }
    Clock(int hhmm) : hour(hhmm/100), minute(hhmm%100) { }
    void tick( );
  private:
    int hour, minute;
};

Clock c1;                                       // 0:00
Clock c2(120);                                  // 1:20
Clock c3(180);                                  // 3:00
```

One Solution: Global Constructor-like Functions

```
class Clock                                  /* File: clock-w-global-fcn.cpp */
{
  public:
    Clock(int h = 0, int m = 0) : hour(h), minute(m) { }
    void tick( );
  private:
    int hour, minute;
};

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

Clock make_clock_minutes(int min)
    { return Clock( min/60, min%60 ); }
```

Disadvantages of Global Constructor-like Functions

1. Global functions all live in the same (global) namespace, so the names of the "constructor-like functions" have to be long.
2. It is not clear that the functions belong to the class. When the class is modified, it might be easy to forget to look at the "constructor-like functions."
3. Global constructor-like functions cannot access private data members of the class. (Though this may be solved by friend functions.)

Class Clock With Static Methods

```
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) { }
};
```

Static (Class) Method

- Classes can also have static methods.
- Static data (methods) are also called class data (methods).
- Static variables (methods) are actually global variables (functions) but with a class scope and are subject to the access control specified by the class developer.
- Static methods belong to a class, and can access all its members including private data.
- However, static methods of a class do not have the implicit this pointer like regular member functions, and may be used even when no objects of the class are created yet!
- Thus, static methods can only use static variables of the class.
- Member functions of a class, of course, may call its static methods.

Static (Class) 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 methods are services provided by the objects.
- Class data and class methods are data and services provided by the `factory`.
- Even if no object of this type has been created, we can access the class data and methods.
- E.g. A regular member function of Car like

  ```
  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 method of Car like

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

  after compilation becomes:

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

Example: Class Car

```
class Car                                          /* File: car.h */
{
  public:
    Car( ) : total_km(0) { ++num_cars; }
    ~Car( ) { --num_cars; }
    void drive(int km) { total_km += km; }

    static int cars_produced( ) { return num_cars; }

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

Example: Class Car ..

```
#include <iostream>                                    /* File: car-test.cpp */
#include "car.h"
using namespace std;
// Definition and initialization of static class member
int Car::num_cars = 0;

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

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

    { Car kia; kia.drive(400); cout << Car::cars_produced( ) << endl; }
    cout << Car::cars_produced( ) << endl;

    delete [ ] cp; cout << Car::cars_produced( ) << endl;
}
```

Example: Linked List With Static Data

A `Person` class that automatically links all persons in a linked list.

```
#include <string>                                   /* File: person.h */
using std::string;

class Person {
  public:
    static Person* first;   // This is the head of the whole link list!

    Person(string s);
    Person(const Person &p);
    ~Person( );
    string get_name( ) const { return name; }
    const Person* get_next( ) const { return next; }

  private:
    string name;
    Person* next;
};
```

Example: Linked List With Static Data ..

```
#include "person.h"                                  /* File: person.cpp */
// Definition and initialization of static class member
Person* Person::first = 0;

Person::Person(string s)
        : name(s), next(first) { first = this; }
Person::Person(const Person &p)
        : name(p.name), next(first) { first = this; }

Person::~Person( )
{
    if (first == this) { first = next; return; }

    for (Person* p = first; p; p = p->next)
        if ( p->next == this )
        {
            p->next = next; return;
        }
}
```

Example: Linked List With Static Data ...

```
#include <iostream>                                    /* File: person-test.cpp */
#include "person.h"

int main( )
{
    Person a("Alan");
    Person b("Brian");
    Person c("Cindy");
    Person d("Debbie");

    for (const Person* p = Person::first; p; p = p->get_next( ))
        std::cout << p->get_name( ) << std::endl;

    return 0;
}  // Compile: g++ person.cpp person-test.cpp
```