Ch 14: More About Classes

CS 2308
Spring 2015
Jill Seaman

14.1 Instance and Static Members

- **instance variable**: a member variable in a class. Each object (instance) has its own copy.

- **static variable**: one variable shared among all objects of a class

- **static member function**:
  - can be used to access static member variable;
  - normal functions can access static member variables, too
  - but it cannot access instance variables
  - can be called like a standalone function

**Tree class declaration**

```cpp
// Tree class
class Tree  {
private:
    static int objectCount;
    int numOfLeaves;
public:
    Tree(int);
    int getObjectCount();
    int getNumOfLeaves();
};
```

**Program demo of static variable**

```cpp
#include <iostream>
using namespace std;
#include "Tree.h"

int main() {
    Tree oak(305);
    Tree elm(1045);
    Tree pine(999);
    cout << "We have " << pine.getObjectCount() << " Trees in our program.\n";
    cout << "The pine tree has " << pine.getNumOfLeaves() << " leaves.\n";
}
```

What will be the output?
Three instances of the Tree class, but only one objectCount variable

Instances of the Tree class

14.3 Member-wise Assignment

- Can use "=" to
  - assign (copy) one object to another, or
  - initialize an object with another object's data
- Copies member to member. e.g.,

  instance2 = instance1;  //Note: assignment

  means: copy all member values from instance1
  and assign to the corresponding member
  variables of instance2
- Also used at initialization:  Time t2 = t1;

static member function

- Declared with static before return type:

  static int getObjectCount();  

- Static member functions can access static
  member data only

  int Tree::getObjectCount() {
    return objectCount;
  }

  Don't need static keyword here.

- Can be called independently of objects (use class name):

  cout << "We have " << Tree::getObjectCount() << "Trees in our program.\n";

Member-wise assignment: demo

Time t1(10, 20);
Time t2(12, 40);

cout << "t1: " << t1.display() << endl;
cout << "t2: " << t2.display() << endl;
t2 = t1;

cout << "t1: " << t1.display() << endl;
cout << "t2: " << t2.display() << endl;

Output:  

<table>
<thead>
<tr>
<th>t2</th>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2: 12:40</td>
<td>t1: 10:20</td>
</tr>
<tr>
<td>t2: 12:40</td>
<td>t1: 10:20</td>
</tr>
</tbody>
</table>

Just like = for structs
14.4 Copy Constructors

- Special constructor used when a newly created object is initialized using another object of the same class.

- [used implicitly when passing arguments by value]

- The default copy constructor copies field-to-field (member-wise assignment).

- Default copy constructor works fine in many cases

```
Time t1;
Time t2 (t1);
Time t3 = t1;
```

IntCell declaration

- Problem: what if the object contains a pointer?

```
class IntCell
{
    private:
        int *storedValue;   //ptr to int
    public:
        IntCell (int initialValue);
        ~IntCell();
        int read () const;
        void write (int x);
};
```

IntCell Implementation

```
#include “IntCell.h”

IntCell::IntCell (int initialValue) {
    storedValue = new int;
    *storedValue = initialValue;
}

IntCell::~IntCell() {
    delete storedValue;
}

int IntCell::read () const {
    return *storedValue;
}

void IntCell::write (int x) {
    *storedValue = x;
}
```

Problem with member-wise assignment

- What we get from member-wise assignment in objects containing dynamic memory (ptrs):

```
IntCell object1(5);
IntCell object2 = object1; // calls copy constructor
//object2.storedValue=object1.storedValue
object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

What is output? 5 13 or 13 13
Problem with member-wise assignment

- Why are they both changed to 13?
- Member-wise assignment does a shallow copy. It copies the pointer’s address instead of allocating new memory and copying the value.
- As a result, both objects point to the same location in memory

Programmer-Defined Copy Constructor

- Prototype and definition of copy constructor:
  ```
  IntCell(const IntCell &obj); // Add to class declaration
  
  IntCell::IntCell(const IntCell &obj) {
    storedValue = new int;
    *storedValue = obj.read(); // or *(obj.storedValue)
  }
  ```

- Copy constructor takes a reference parameter to an object of the class
- otherwise, pass-by-value would use the copy constructor to initialize the obj parameter, which would call the copy constructor: this is an infinite loop

Programmer-Defined Copy Constructor

Each object now points to separate dynamic memory:

```
IntCell object1(5);
IntCell object2 = object1; // now calls MY copy constructor

object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

Output: 5 13

Copy Constructor: limitations

- Copy constructor is called ONLY during initialization of an object, NOT during assignment.
- If you use assignment with IntCell, you will still end up with member-wise assignment and a shared value:

```
IntCell object1(5);
IntCell object2(0);
object2 = object1; // object2.storedValue = object1.storedValue

object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

Output: 13 13
14.5 Operator Overloading

- Operators such as =, +, <, ... can be defined to work for objects of a programmer-defined class
- The name of the function defining the over-loaded operator is `operator` followed by the operator symbol:
  - `operator+` to define the + operator, and
  - `operator=` to define the = operator
- Just like a regular member function:
  - Prototype goes in the class declaration
  - Function definition goes in implementation file

Operator Overloading

- Prototype in Time class declaration:
  ```
  int operator- (Time right);
  ```
- `operator-` is the function name
- The operator function is defined from the perspective of the object on the left side of the minus
  - Inside the `operator-` function definition, hour and minute will be from the left hand side (t1)
  - `Time right` is the parameter for the right hand side of operator (t2)
  - Inside the `operator-` function definition, `right.hour` and `right.minute` will be from the right hand side (t2)

Calling an Overloaded Operator

- The operator function is called on the object on the left side of the operator
- It can be called like a normal member function:
  ```
  int minutes = object1.operator-(object2);
  ```
- It can also be called using the more conventional operator syntax:
  ```
  int minutes = object1 - object2;
  ```
- Both call the same `operator-` function, from the perspective of object1

Example: minus for Time objects

```cpp
class Time {
private:
  int hour, minute;
public:
  int operator- (Time right) {
    //Note: 12%12 = 0
    return (hour%12)*60 + minute - ((right.hour%12)*60 + right.minute);
  }
}
```

```cpp
int main() {
  Time time1(12,20), time2(4,40);
  int minutesDiff = time2 - time1;
  cout << minutesDiff << endl;
}
```

Output: 260
Overloading == and < for Time

```cpp
class Time {
private:
    int hour;
    int minute;
    void addHour();
public:
    Time();
    Time(int);
    Time(int,int);
    void addMinute(); //adds one minute
    void addMinute(int); //adds n minutes
    int getHour();
    int getMinute();
    int operator- (Time right);
    bool operator== (Time right) {
        if (hour == right.hour && minute == right.minute)
            return true;
        else
            return false;
    }
    bool operator< (Time right) {
        if (hour == right.hour)
            return (minute < right.minute);
        return (hour%12) < (right.hour%12);
    }
    int main() {
        Time time1(12,20), time2(12,21);
        if (time1<time2) cout << "correct" << endl;
        time1.addMinute();
        if (time1==time2) cout << "correct again" << endl;
    }
};
```

Overloading + for Time

```cpp
class Time {
private:
    int hour, minute;
public:
    Time operator+ (Time right);
};
Time Time::operator+ (Time right) { //Note: 12%12 = 0
    int totalMin = (hour%12)*60 + minute +
        (right.hour%12)*60 + right.minute;
    int h = totalMin / 60; //integer division, total hours
    h = h%12; //keep it between 0 and 11
    if (h==0) h = 12; //convert 0:xx to 12:xx
    int resultMin = totalMin % 60; //create new Time obj
    return Time(h, resultMin);
};
int main() {
    Time t1(12,5);
    Time t2(2,50);
    Time t3 = t1+t2;
    cout << t3.display() << endl;
}
```

Overload = for IntCell

```cpp
class IntCell {
private:
    int *storedValue;
public:
    IntCell(const IntCell &obj);
    IntCell(int);
    ~IntCell();
    int read() const;
    void write(int);
    void operator= (IntCell);
};
void IntCell::operator= (IntCell rhs) {
    write(rhs.read());
};
//in a driver:
IntCell object1(5), object2(0);
object2 = object1;
object2.write(13);
cout << object1.read() << endl; //object1 is unchanged
```