Chapter 14:

More About Classes

14.2

Friends of Classes

Friends of Classes

- Friend: a function or class that is not a member of a class
  - inside a friend class, object of the original class has access to private members of the original class

- A friend function can be a stand-alone function or a member function of another class
  - It is declared a friend of a class with `friend` keyword in the function prototype

`friend` Function Declarations

- Stand-alone function:

```
friend void setAVal(int num);
// declares setAVal function to be
// a friend of this class
```

- Member function of another class:

```
friend void SomeClass::setNum(int num)
// setNum function from SomeClass
// class is a friend of this class
```

`friend` Function Declarations

- Review the attached friends.cpp program file
  - how to create a friend function
  - how to use super objects in friend function
  - alternative to friend function (use of accessor methods)

`friend` Class Declarations

- Class as a friend of a class:

```
class FriendClass
{
  ...
};

class NewClass
{
  public:
    friend class FriendClass; // declares
  // the entire FriendClass as a friend
  // of this class
  ...
};
```

14.3

Member-wise Assignment

Memberwise Assignment

- Can use = to assign one object to another, or to initialize an object with an object's data

- Copies member to member. e.g.,

```
Rectangle window (10, 20), door (40, 100);

window = door;
// means:
// copy all values of the instance variables from door object and
// assign to the corresponding instance variables of window object
```

- Use at initialization:

```
Rectangle door (40, 100);
// at the time of declaration
Rectangle window = door;
```

Program 14-5

```
1  // This program demonstrates memberwise assignment.
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5
6  int main()
7  {
8     // Define two Rectangle objects.
9     Rectangle box1(10.0, 10.0);   // width = 10.0, length = 10.0
10    Rectangle box2 (20.0, 20.0);  // width = 20.0, length = 20.0
11
12    // Display each object's width and length.
13    cout << "box1's width and length: " << box1.getWidth()
14         << " " << box1.getLength() << endl;
15    cout << "box2's width and length: " << box2.getWidth()
16         << " " << box2.getLength() << endl << endl;
17
18    // Assign the members of box1 to box2.
19    box2 = box1;
20
21    // Display each object's width and length again.
22    cout << "box1's width and length: " << box1.getWidth()
23         << " " << box1.getLength() << endl;
24    cout << "box2's width and length: " << box2.getWidth()
25         << " " << box2.getLength() << endl;
26
27    return 0;
28 }
```

Program 14-5 *(continued)*

Program Output

```
box1's width and length: 10 10
box2's width and length: 20 20

box1's width and length: 10 10
box2's width and length: 10 10
```

14.4

Copy Constructors

Copy Constructors

- Special constructor used when a newly created object is initialized to the data of another object of the same class
- There is a default copy constructor in every class
  - Default copy constructor copies field-to-field
- Default copy constructor works fine in many cases

Default Copy Constructor: issue

Memberwise copy does not work when the class uses dynamic memory:

```
int * score = new int;
*score = 100;

int * steps = score;
*steps = 200;

cout << (*steps) << " " << (*score) << endl;

delete  score;
```

Copy Constructors

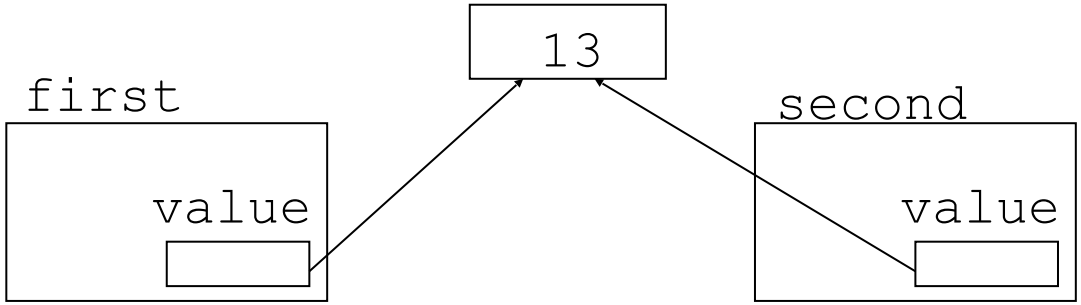
Problem: what if object contains a pointer?

```
class SomeClass
{
private:
     int *value;

public:
     SomeClass(int val = 0)
     {
          value = new int;
          *value = val;
     }
     ~SomeClass()
     {
          delete value;
     }
     int getVal();
     void setVal(int);
};
```

Copy Constructors

What we get using memberwise copy with objects containing dynamic memory:




```
SomeClass first(5);
// the value instance variable of both
// objects have the same address
SomeClass second = first;
// if we now change the content of the
// value variable by the second object
second.setVal(13);
// that will also impact the content of the
// value variable of the first object
cout << first.getVal(); // also 13
```