Classes and Objects

Week 4

Gaddis: 13.1-13.12 (classes)
15.1-15.5 (inheritance)

CS 5301
Fall 2013

Jill Seaman

The Class

- A class in C++ is similar to a structure.
- A class contains:
  - variables (members) AND
  - functions (member functions or methods)
- Members can be:
  - private: inaccessible outside the class (this is the default)
  - public: accessible outside the class.

Example class: Time

class declaration with functions defined inline

```
class Time {            //new data type
  private:
    int hour;
    int minute;
  public:
    void setHour(int hr) { hour = hr; }
    void setMinute(int min) { minute = min; }
    int getHour() const { return hour; }
    int getMinute() const { return minute; }
    void display() const { cout << hour << ":" << minute; }
};
```


Using Time class in a driver

```
int main()
{
   Time t1, t2;

   t1.setHour(6);
   t1.setMinute(30);
   cout << t1.getHour() << endl;

   t2.setHour(9);
   t2.setMinute(20);
   t2.display();
   cout << endl;
};
```

Output:

```
6
9:20
```

Example class: Time (version 2)

class declaration with functions defined outside

```
class Time {            //new data type
  private:
    int hour;
    int minute;
  public:
    void setHour(int);
    void setMinute(int);
    int getHour() const;
    int getMinute() const;
    void display() const;
};
void Time::setHour(int hr) {
  hour = hr;            // hour is a member var
}
void Time::setMinute(int min) {
  minute = min;         // minute is a member var
}
int Time::getHour() const {
  return hour;
}
int Time::getMinute() const {
  return minute;
}
void Time::display() const {
   cout << hour << ":" << minute;
}
```

The member functions can be defined outside the class

Don't forget the class name and scope resolution operator (::)

Access rules

- Used to control access to members of the class
- public: can be accessed by functions inside AND outside of the class
- private: can be called by or accessed by only functions that are members of the class (inside)

```
int main()
{
   Time t1;

   t1.setHour(6);
   t1.setMinute(30);
   cout << t1.hour << endl; //Error, hour is private

};
```


Separation of Interface from Implementation

- Class declarations are usually stored in their own header files (Time.h)
  - called the specification file
  - filename is usually same as class name.
- Member function definitions are stored in a separate file (Time.cpp)
  - called the class implementation file
  - it must #include the header file,
- Any program/file using the class must include the class's header file (#include "Time.h")

Time class, separate files

Time.h

```
#include <string>
using namespace std;

// models a 12 hour clock
class Time {

private:
    int hour;
    int minute;

public:
    void setHour(int);
    void setMinute(int);
    int getHour() const;
    int getMinute() const;

    void display() const;
};
```

Driver.cpp

```
//Example using Time class
#include<iostream>
#include "Time.h"
using namespace std;

int main() {
    Time t;
    t.setHour(12);
    t.setMinute(58);
    t.display();
    cout <<endl;
    t.setMinute(59);
    t.display();
    cout  << endl;
}
```

Time class, separate files

Time.cpp

```
#include <iomanip>
#include <sstream>
#include "Time.h"
using namespace std;

void Time::setHour(int hr) {
  hour = hr;
}

void Time::setMinute(int min) {
  minute = min;
}

int Time::getHour() const {
  return hour;
}

int Time::getMinute() const {
  return minute;
}

void Time::display() const {
   cout << hour << ":" << minute;
}
```


Constructors

- A constructor is a member function with the same name as the class.
- It is called automatically when an object is created
- It performs initialization of the new object
- It has no return type
- It can be overloaded: more than one constructor function, each with different parameter lists.
- A constructor with no parameters is the default constructor.
- If your class defines no constructors, C++ will provide a default constructor automatically.

Constructor Declaration (and use)

- Note no return type, same name as class:

```
#include <string>
using namespace std;

// models a 12 hour clock
class Time {

private:
    int hour;
    int minute;

public:
    Time();
    Time(int,int);

    void setHour(int);
    void setMinute(int);
    int getHour() const;
    int getMinute() const;

    void display() const;
};
```

```
//Example using Time class
#include<iostream>
#include "Time.h"
using namespace std;

int main() {
    Time t;
    t.display();
    cout <<endl;

    Time t1(10,30);
    t1.display();
    cout  << endl;
}
```


Constructor Definition

- Note no return type, prefixed with Class::

```
// file Time.cpp
#include <sstream>
#include <iomanip>
using namespace std;

#include "Time.h"

Time::Time() {
   hour = 12;
   minute = 0;
}
Time::Time(int hr, int min) {
   hour = hr;
   minute = min;
}
...
```

Output:

```
12:0
10:30
```

Destructors

- Member function that is automatically called when an object is destroyed
- Destructor name is ~classname, e.g., `~Time`
- Has no return type; takes no arguments
- Only one destructor per class, i.e., it cannot be overloaded, cannot take arguments
- If the class allocates dynamic memory, the destructor should release (delete) it.

```
class Time
{
   public:
      Time();      // Constructor prototype
      ~Time();     // Destructor prototype
```


Copy Constructors

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

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

Both of the last two use the copy constructor

- The **default** copy constructor copies field-to-field (member-wise assignment).
- Default copy constructor works fine in most cases
- You can re-define it for your class as needed.

Composition

- When one class contains another as a member:

This class declaration uses inlined function definitions

```
class Calls          // must #include Time.h
{
  private:
    Time calls[10];  // times of last 10 phone calls
    // array is initialized using default constructor

  public:
    Calls() { }

    void set(int i; Time t) { calls[i] = t; }

    void displayAll ()
    { for (int i=0; i<10; i++) {
         calls[i].display();     //calls member function
         cout << "  ";
      }
    }
};
```


Inheritance

- Provides a way to create a new class from an existing class
- The new class is a specialized version of the existing class
- Base class (or parent)
- Derived class (or child) – inherits from the base class
- The derived class has access to all the public (and protected) data and function members of the base class (but NOT to the private members)

Class Access Specification

- Determines how private, protected, and public members of base class are inherited by the derived class

```
class Grade {
  private:
    char letter;
    float score;
    void calcGrade();
  public:
    void setScore(float);
    float getScore();
    char getLetter();
}
```

```
class Test: public Grade {
  private:
    int numQuestions;
    float pointsEach;
    int numMissed;
  public:
    Test(int, int);
}
```

- class Test extends class Grade.

Class Access Specification

class Grade

private members:
`char letter;`
`float score;`
`void calcGrade();`
public members:
`void setScore(float);`
`float getScore();`
`char getLetter();`

class Test : public Grade

private members:
`int numQuestions;`
`float pointsEach;`
`int numMissed;`
public members:
`Test(int, int);`

When `Test` class inherits from `Grade` class using `public` class access, it looks like this:

private members:
`int numQuestions:`
`float pointsEach;`
`int numMissed;`
public members:
`Test(int, int);`
`void setScore(float);`
`float getScore();`
`float getLetter();`

An instance of Test contains letter and score, but they are not accessible from inside the Test member functions.

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created, the base class's (default) constructor is executed first, followed by the derived class's constructor
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class

Constructors and Destructors: example

```
class BaseClass {
public:
   BaseClass()  // Constructor
      { cout << "This is the BaseClass constructor.\n"; }

   ~BaseClass() // Destructor
      { cout << "This is the BaseClass destructor.\n"; }
};

class DerivedClass : public BaseClass {
public:
   DerivedClass()  // Constructor
      { cout << "This is the DerivedClass constructor.\n"; }

   ~DerivedClass()  // Destructor
      { cout << "This is the DerivedClass destructor.\n"; }
};
```

Constructors and Destructors:
example

```
int main() {
   cout << "We will now define a DerivedClass object.\n";

   DerivedClass object;

   cout << "The program is now going to end.\n";
}
```

Output:

```
We will now define a DerivedClass object.
This is the BaseClass constructor.
This is the DerivedClass constructor.
The program is now going to end.
This is the DerivedClass destructor.
This is the BaseClass destructor.
```


Passing Arguments to a non-default Base Class Constructor

- Allows programmer to choose among multiple base class constructors
- Specify arguments to base constructor in the derived constructor function header:

```
Square::Square(int side) : Rectangle(side, side) {
  // code for Square goes here, if any
}
//assuming Square is derived from Rectangle
```

- Must specify a call to a base class constructor if base class has no default constructor

Redefining Base Class Functions

- Redefining function: a function in a derived class that has the same name and parameter list as a function in the base class
- Not the same as overloading – with overloading, parameter lists must be different
- Objects of base class use base class version of function; objects of derived class use derived class version of function.
- To call the base class version from the derived class version, you must prefix the name of the function with the base class name and the scope resolution operator

Redefining Base Class Functions:
example

```
class Animal {
 private:
  string species;
 public:
  Animal()           { species = "Animal";}
  Animal(string spe) { species = spe ;}
  void display()
         {cout << "species " << species; }
};
```

```
class Primate: public Animal {
 private:
  int heartCham;
 public:
  Primate() : Animal("Primate")  { }
  Primate(int in) : Animal ("Primate") { heartCham = in; }
  void display()
    { Animal::display();
      cout << ", # of heart chambers " << heartCham; }
};
```

Redefining Base Class Functions:
example

```
int main()
{
    Animal  jasper;       //  Animal()
    Primate fred(4);      //  Primate(int)
    jasper.display(); cout << endl;
    fred.display();   cout << endl;
}
```

Output:

```
species Animal
species Primate, # of heart chambers 4
```