W8.1 Continuing Classes

• friend Functions and friend Classes
• Using the this Pointer
• Cascading Function Calls

7.4 friend Functions and friend Classes

• friend function and friend classes
  – Can access private and protected members of another class
  – friend functions are not member functions of class
    • Defined outside of class scope
• Properties of friendship
  – Friendship is granted, not taken
  – Not symmetric (if B a friend of A, A not necessarily a friend of B)
  – Not transitive (if A a friend of B, B a friend of C, A not necessarily a friend of C)
When to Use a friend

- Using friend functions can enhance performance.
- friend functions may be used to overload operators for classes and to create iterator classes.
  - Objects of iterator class used to successively select items or perform an operation on items in a container class (Ch.7.9)
  - Objects of container classes are capable of storing items.
- Using friend functions is appropriate when member function cannot be used (operator overloading, see Ch.8.4, later)

7.4 friend Functions and friend Classes

- friend declarations
  - To declare a friend function
    - Type friend before the function prototype in the class that is giving friendship
      
      friend int myFunction( int x );

      should appear in the class giving friendship
  - To declare a friend class
  - Type friend class Classname in the class that is giving friendship
  - if ClassOne is granting friendship to ClassTwo,
    
    friend class ClassTwo;
  - should appear in ClassOne’s definition
1 // Fig. 7.5: fig07_05.cpp
2 // Friends can access private members of a class.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // Modified Count class
9 class Count {
10    friend void setX( Count &, int ); // friend declaration
11 public:
12    Count() { x = 0; }                // constructor
13    void print() const { cout << x << endl; }  // output
14 private:
15    int x;  // data member
16    }
17
18 // Can modify private data of Count because
19 // setX is declared as a friend function of Count;
20 void setX( Count &c, int val )
21 {
22    c.x = val;  // legal: setX is a friend of Count
23 }
24
25 int main()
26 {
27    Count counter;
28
29    cout << "counter.x after instantiation: ";
30    counter.print();
31
32    cout << "counter.x after call to setX friend function: ";
33    setX( counter, 8 ); // set x with a friend
34    counter.print();
35    return 0;
36 }

Program Output

counter.x after instantiation: 0
counter.x after call to setX friend function: 8

Changing private variables allowed.
// Fig. 7.6: fig07_06.cpp
// Non-friend/non-member functions cannot access
// private data of a class.
#include <iostream>

using std::cout;
using std::endl;

// Modified Count class
class Count {
public:
    Count() { x = 0; } // constructor
    void print() const { cout << x << endl; } // output
    private:
    int x; // data member
};

// Function tries to modify private data of Count,
// but cannot because it is not a friend of Count.
void cannotSetX( Count &c, int val )
{
    c.x = val; // ERROR: 'Count::x' is not accessible
}

int main()
{
    Count counter;

cannotSetX( counter, 3 ); // cannotSetX is not a friend
    return 0;
}
7.5 Using the this Pointer

- **this** pointer
  - Allows objects to access their own address
  - Not part of the object itself
  - Implicit first argument on non-static member function call to the object
  - Implicitly reference member data and functions
  - The type of the **this** pointer depends upon the type of the object and whether the member function using **this** is **const**
  - In a non-**const** member function of *Employee*, **this** has type
    
    ```
    Employee * const
    ```
    • Constant pointer to an *Employee* object
  - In a **const** member function of *Employee*, this has type
    
    ```
    const Employee * const
    ```
    • Constant pointer to a constant *Employee* object

- **Examples using this**
  - For a member function print data member *x*, either
    
    ```
    this->x
    ```
    or
    
    ```
    (*this).x
    ```

- **Cascaded member function calls**
  - Function returns a reference pointer to the same object
    
    ```
    { return *this; }
    ```
  - Other functions can operate on that pointer
  - Functions that do not return references must be called last
7.5 Using the this Pointer

- Example of cascaded member function calls
  - Member functions `setHour`, `setMinute`, and `setSecond` all return `*this` (reference to an object)
  - For object `t`, consider
    ```cpp
t.setHour(1).setMinute(2).setSecond(3);
    ```
  - Executes `t.setHour(1)`, returns `*this` (reference to object) and the expression becomes
    ```cpp
t.setMinute(2).setSecond(3);
    ```
  - Executes `t.setMinute(2)`, returns reference and becomes
    ```cpp
t.setSecond(3);
    ```
  - Executes `t.setSecond(3)`, returns reference and becomes
    ```cpp
t;
    ```
  - Has no effect
Program Output

All three methods have the same result.

1 // Fig. 7.8: time6.h
2 // Cascading member function calls.
3
4 // Declaration of class Time.
5 // Member functions defined in time6.cpp
6 #ifndef TIME6_H
7 #define TIME6_H
8
9 class Time {
10 public:
11    Time( int = 0, int = 0, int = 0 );  // default constructor
12
13    // set functions
14    Time &setTime( int, int, int ); // set hour, minute, second
15    Time &setHour( int );    // set hour
16    Time &setMinute( int );  // set minute
17    Time &setSecond( int );  // set second
18
19    // get functions (normally declared const)
20    int getHour() const;     // return hour
21    int getMinute() const;   // return minute
22    int getSecond() const;   // return second
23
24    // print functions (normally declared const)
25    void printMilitary() const; // print military time
26    void printStandard() const; // print standard time
27 private:
28    int hour;              // 0 - 23
29    int minute;            // 0 - 59
30    int second;            // 0 - 59
31};
32
33 #endif

Notice the Time & - function returns a reference to a Time object. Specify object in function definition.
// Fig. 7.8: time.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time6.h"

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
    { setTime( hr, min, sec ); } 

// Set the values of hour, minute, and second.
Time &Time::setTime( int h, int m, int s )
{
    setHour( h );
    setMinute( m );
    setSecond( s );
    return *this; // enables cascading
}

// Set the hour value
Time &Time::setHour( int h )
{
    hour = ( h >= 0 && h < 24 ) ? h : 0;
    return *this;   // enables cascading
}

// Set the minute value
Time &Time::setMinute( int m )
{
    minute = ( m >= 0 && m < 60 ) ? m : 0;
    return *this;   // enables cascading
}

// Set the second value
Time &Time::setSecond( int s )
{
    second = ( s >= 0 && s < 60 ) ? s : 0;
    return *this;   // enables cascading
}

// Get the hour value
int Time::getHour() const { return hour; }

// Get the minute value
int Time::getMinute() const { return minute; }

// Get the second value
int Time::getSecond() const { return second; }

// Display military format time: HH:MM
void Time::printMilitary() const
{
    cout << ( hour < 10 ? "0" : "" ) << hour << ":
    << ( minute < 10 ? "0" : "" ) << minute;
1.1 Function definitions

1. Load header
2. Function calls
3. Print values

// Fig. 7.8: fig07_08.cpp
// Cascading member function calls together
// with the this pointer
#include <iostream>
#include "time6.h"

int main()
{
   Time t;
   t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
   cout << "Military time: ";
   t.printMilitary();
   cout << "Standard time: ";
   t.printStandard();
   cout << endl;
   t.setTime( 20, 20, 20 ).printStandard();
   cout << endl;
   return 0;
}

Military time: 18:30
Standard time: 6:30:22 PM
New standard time: 8:20:20 PM