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
// Fig. 7.5: fig07_05.cpp
// Friends can access private members of a class.
#include <iostream>

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

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

void setX( Count &c, int val ) {
    c.x = val;  // legal: setX is a friend of Count
}

int main() {
    Count counter;

    cout << "counter.x after instantiation: ";
    counter.print();
}
cout << "counter.x after call to setX friend function: ";
setX( counter, 8 );  // set x with a friend
counter.print();
return 0;
}

Program Output

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

private data was changed.
1 // Fig. 7.6: fig07_06.cpp
2 // Non-friend/non-member functions cannot access
3 // private data of a class.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // Modified Count class
10 class Count {
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 // Function tries to modify private data of Count,
19 // but cannot because it is not a friend of Count.
20 void cannotSetX( Count &c, int val )
21 {
22    c.x = val;  // ERROR: 'Count::x' is not accessible
23 }
24
25 int main()
26 {
27    Count counter;
28
29    cannotSetX( counter, 3 ); // cannotSetX is not a friend
30    return 0;
31 }
Compiling...
Fig07_06.cpp
D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\Fig07_06.cpp(22) :
  error C2248: 'x' : cannot access private member declared in
class 'Count'
   D:\books\2000\cpphtp3\examples\Ch07\Fig07_06\Fig07_06.cpp(15) : see declaration of 'x'
Error executing cl.exe.
test.exe - 1 error(s), 0 warning(s)

Expected compiler error - cannot access private data
### 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
7.5 Using the this Pointer

- **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
// Fig. 7.7: fig07_07.cpp
// Using the this pointer to refer to object members.
#include <iostream>

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

class Test {
 public:
  Test( int = 0 ); // default constructor
  void print() const;
 private:
  int x;
};

Test::Test( int a ) { x = a; } // constructor

void Test::print() const // () around *this
{
    cout << "        x = " << x
        << "\n this->x = " << this->x
        << "\n(*this).x = " << ( *this ).x << endl;
}

int main()
{
    Test testObject( 12 );
    testObject.print();
    return 0;
}
x = 12
this->x = 12
(*this).x = 12

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 #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
}
```cpp
// 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;
}
```
// Display standard format time: HH:MM:SS AM (or PM)
void Time::printStandard() const
{
    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
         << ":" << ( minute < 10 ? "0" : "" ) << minute
         << ":" << ( second < 10 ? "0" : "" ) << second
         << ( hour < 12 ? " AM" : " PM" );
}

// Fig. 7.8: fig07_08.cpp
// Cascading member function calls together
// with the this pointer
#include <iostream>
using std::cout;
using std::endl;
#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 << "\n\nNew standard time: ";
t.setTime( 20, 20, 20 ).printStandard();

```cpp
127   cout << endl;
128
129   return 0;
130 }
```

Military time: 18:30
Standard time: 6:30:22 PM

New standard time: 8:20:20 PM