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
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();
```

setX a **friend** of class **Count** (can access **private** data).

setX is defined normally and is not a member function of **Count**.

Changing **private** variables allowed.

Outline

1. Class definition

1.1 Declare function a `friend`

1.2 Function definition

1.3 Initialize `Count` object


```
31    cout << "counter.x after call to setX friend function: ";
32    setX( counter, 8 );  // set x with a friend
33    counter.print();
34    return 0;
35 }
```

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

private data was changed.

Outline

2. Modify object

3. Print results

Program Output

Outline

(Previous program without `friend` declared)

```
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 }
```

`cannotSetX` is not a **`friend`** of class **Count**. It cannot access **`private`** data.

`cannotSetX` tries to modify a **`private`** variable…

Outline

Program Output

```
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
    `t.setHour(1).setMinute(2).setSecond(3);`
  - Executes **`t.setHour(1)`**, returns **`*this`** (reference to object) and the expression becomes
    `t.setMinute(2).setSecond(3);`
  - Executes **`t.setMinute(2)`**, returns reference and becomes
    `t.setSecond(3);`
  - Executes **`t.setSecond(3)`**, returns reference and becomes
    `t;`
  - Has no effect

Outline

1. Class definition

1.1 Function definition

1.2 Initialize object

2. Function call

```
1  // Fig. 7.7: fig07_07.cpp
2  // Using the this pointer to refer to object members.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  class Test {
9  public:
10    Test( int = 0 );             // default constructor
11    void print() const;
12 private:
13    int x;
14 };
15
16 Test::Test( int a ) { x = a; }  // constructor
17
18 void Test::print() const    // ( ) around *this
19 {
20    cout << "        x = " << x
21         << "\n  this->x = " << this->x
22         << "\n(*this).x = " << ( *this ).x << endl;
23 }
24
25 int main()
26 {
27    Test testObject( 12 );
28
29    testObject.print();
30
31    return 0;
32 }
```

Printing x directly.

Print **x** using the arrow **->** operator off the **this** pointer.

Printing **x** using the dot (**.**) operator. Parenthesis required because dot operator has higher precedence than *****. Without, interpreted incorrectly as **`*(this.x)`**.

Outline

Program Output

```
      x = 12
  this->x = 12
(*this).x = 12
```

All three methods have the same result.

Outline

1. Class definition

```
// Fig. 7.8: time6.h
// Cascading member function calls.

// Declaration of class Time.
// Member functions defined in time6.cpp
#ifndef TIME6_H
#define TIME6_H

class Time {
public:
   Time( int = 0, int = 0, int = 0 );  // default constructor

   // set functions
   Time &setTime( int, int, int ); // set hour, minute, second
   Time &setHour( int );    // set hour
   Time &setMinute( int );  // set min
   Time &setSecond( int );  // set sec

   // get functions (normally declared
   int getHour() const;     // return
   int getMinute() const;   // return minute
   int getSecond() const;   // return second

   // print functions (normally declared const)
   void printMilitary() const;  // print military time
   void printStandard() const;  // print standard time
private:
   int hour;             // 0 - 23
   int minute;           // 0 - 59
   int second;           // 0 - 59
};

#endif
```

Notice the **`Time &`** - function returns a reference to a **`Time`** object. Specify object in function definition.

Outline

1. Load header file

1.1 Function definitions

```
34 // Fig. 7.8: time.cpp
35 // Member function definitions for Time class.
36 #include <iostream>
37
38 using std::cout;
39
40 #include "time6.h"
41
42 // Constructor function to initialize private data.
43 // Calls member function setTime to set variables.
44 // Default values are 0 (see class definition).
45 Time::Time( int hr, int min, int sec )
46    { setTime( hr, min, sec ); }
47
48 // Set the values of hour, minute, and second.
49 Time &Time::setTime( int h, int m, in
50 {
51    setHour( h );
52    setMinute( m );
53    setSecond( s );
54    return *this;   // enables cascading
55 }
56
57 // Set the hour value
58 Time &Time::setHour( int h )
59 {
60    hour = ( h >= 0 && h < 24 ) ? h : 0;
61
62    return *this;   // enables cascading
63 }
64
```

Returning **`*this`** enables cascading function calls

Outline

1.1 Function definitions

```
65 // Set the minute value
66 Time &Time::setMinute( int m )
67 {
68    minute = ( m >= 0 && m < 60 ) ? m : 0;
69
70    return *this;   // enables cascading
71 }
72
73 // Set the second value
74 Time &Time::setSecond( int s )
75 {
76    second = ( s >= 0 && s < 60 ) ? s : 0;
77
78    return *this;   // enables cascading
79 }
80
81 // Get the hour value
82 int Time::getHour() const { return hour; }
83
84 // Get the minute value
85 int Time::getMinute() const { return minute; }
86
87 // Get the second value
88 int Time::getSecond() const { return second; }
89
90 // Display military format time: HH:MM
91 void Time::printMilitary() const
92 {
93    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
94         << ( minute < 10 ? "0" : "" ) << minute;
```

Returning **`*this`** enables cascading function calls

Outline

1.1 Function definitions

---

1. Load header

1.1 Initialize `Time` object

2. Function calls

3. Print values

```
95 }
96
97 // Display standard format time: HH:MM:SS AM (or PM)
98 void Time::printStandard() const
99 {
100    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
101         << ":" << ( minute < 10 ? "0" : "" ) << minute
102         << ":" << ( second < 10 ? "0" : "" ) << second
103         << ( hour < 12 ? " AM" : " PM" );
104}
105// Fig. 7.8: fig07_08.cpp
106// Cascading member function calls together
107// with the this pointer
108#include <iostream>
109
110using std::cout;
111using std::endl;
112
113#include "time6.h"
114
115int main()
116{
117   Time t;
118
119   t.setHour( 18 ).setMinute( 30 ).setSe
120   cout << "Military time: ";
121   t.printMilitary();
122   cout << "\nStandard time: ";
123   t.printStandard();
124
125   cout << "\n\nNew standard time: ";
126   t.setTime( 20, 20, 20 ).printStandard();
```

`printStandard` does not return a reference to an object.

Notice cascading function calls.

Cascading function calls. **`printStandard`** must be called after **`setTime`** because **`printStandard`** does not return a reference to an object.

`t.printStandard().setTime();` would cause an error.

Outline

```
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
```

Program Output