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

```cpp
// 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
};

// Can modify private data of Count because
// setX is declared as a friend function of
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();
```

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

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

2. Modify object

3. Print results

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

Program Output

`private` data was changed.

Outline

```
// 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;
}
```

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

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

(Previous program without friend declared)

```
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

Program Output

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

```cpp
// 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 required
{
   cout << "        x = " << x
        << "\n  this->x = " << this->x
        << "\n(*this).x = " << ( *this ).x << endl;
}

int main()
{
   Test testObject( 12 );

   testObject.print();

   return 0;
}
```

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

1. Class definition

1.1 Function definition

1.2 Initialize object

2. Function call

Program Output

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

All three methods have the same result.

```
// 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. Class definition

Outline

1. Load header file

1.1 Function definitions

```
// 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
}

```

Returning ***this** enables cascading function calls

Outline

1.1 Function definitions

```
// 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;
```

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

```
}

// 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 ).setSe
   cout << "Military time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();

   cout << "\n\nNew standard time: ";
   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

1.1 Function definitions

----

1. Load header

1.1 Initialize `Time` object

2. Function calls

3. Print values

```
   cout << endl;

   return 0;
}
```

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

New standard time: 8:20:20 PM
```

Program Output