CS 271
October 31, 2017

Class Scope

```
class Time {

    public:

      Time();
      void setHour( int );
      …

    private:

       int hour;
       int minute;
       int second;
};
```

All data members (hour, minute, and second) as well as all member functions (constructors, accessors, mutators, etc) have class scope.

Ternary conditional

```
boolean expression ? true value : false value
```

```
c[x] == oldCh ? newCh : c[x]
```

Default Memberwise Assignment

```
Time timeOne;
Time timeTwo(3, 30, 15);

timeOne = timeTwo;
```

By default, the data members of timeTwo are assigned to the data members of timeOne one at a time.

If you want the assignment operator = to work in a different way, you can overload =.

Friend Classes and Functions

There is no such thing as mandatory friendship.

```
class B {

   friend A;  // class B gives friendship to
              // to class A

              // class A is a friend of class B

}
```

```
class C {

   friend B;  // class B is a friend of class C

}
```

Friendship is not transferred from one class to another. In the example above, class A is NOT a friend of class C.

A function can be given friend status. This is common with the stream insertion and stream extraction operator functions.

```
class Time {

    friend ostream& operator<< ( ostream &, const Time & );
```

Binary operators

A binary operator can be a member function IF the left operand is an object of the class.

```
Time timeOne, timeTwo, timeThree;

timeOne.setHour( 6 );
timeTwo.setHour( 3 );

timeThree = timeOne + timeTwo;
```

In order to perform this statement with Time objects, I have to overload +

operator+ can be a member function of class Time because the left operand is a Time object.

A binary operator that is a member function takes the left operand from the calling object. The right operand is taken from the parameter.

Member function version:

```
class Time {

  public:

   Time operator+ ( const Time & ) const;
.
.
.
};
```

Non-member function version:

```
// this is NOT inside class Time

   Time operator+ (const Time & t1, const Time & t2);
```

Stream extraction(input) and stream insertion(output) operators cannot be member functions of a class.

```
cin >> timeOne;
left    right           The left operand is not an object of the Time class.

cout << timeOne;
left    right           The left operand is not an object of the Time class.
```

Mutators that return a reference

```
Time & setHour ( int );

// here's how we can use this idea

Time timeOne, timeTwo;

timeOne.setHour(3).setMinute(8).setSecond(12);
```

this this this

"this" in C++

this is a pointer to the calling object

```
return *this;
```

Returns the calling object (the pointer is dereferenced)

Examples of how the accessors can be implemented, both without and with the pointer **this**.

```
class Time {

   public:
      int getHour() const;

};

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

// or  😐

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

// or ☹
int Time::getHour() const {
   return (*this).hour;
}
```

An example of how the overloaded + operator could use **this**.

*** It's not a good idea to write it this way.

```
Time operator+ ( const Time & x ) const {

   Time answer;
   answer.hour = this->hour + x.hour;
```

this-> is not needed.   Leave it off.

How does the compiler know which function to call?

The compiler decides which version of a function is being called by examining the function signature.

signature = function name and parameters

1) timeThree = timeOne + timeTwo;

   function name is operator+
   parameter(s) is one Time object

2) timeThree = timeOne + 3;

   function name is operator+
   parameter is one int

3) timeThree = timeOne + 3.0;
   function name is operator+
   parameter is one double

If the compiler doesn't find a function definition that matches the call, you get a syntax error.

On Exam 2

No mutators that return references

No operators overloaded except << >>   +   -   *   /

Yes Binary operators will be on the exam. However, if it is possible for a binary operator to be a member function, I will make it a member function on the exam.

```
class Time {

   public:

      Time operator+( const Time & ) const;
```

Yes Know the four operators that cannot be overloaded.