Classes (Part 4)
Overloading Operators - Friends

1. Adding two Height objects

As a review let's overload the operator + for adding two Height objects. For the Height class declaration we had

```
    Height operator+(const Height& ht);
```

and the class definition we had

```
/*************************************************************
 * operator+
 * ht1 + ht2
 */
Height Height::operator+(const Height& ht) {
    Height h;
    h.feet = feet + ht.feet;
    h.inches = inches + ht.inches;
    if (h.inches >= 12) {
        h.feet++;
        h.inches = h.inches - 12;
    }
    return h;
}
```

To test out this function we can do

```
    Height newHeight;
    newHeight = myHeight + yourHeight;
```

2. Adding a Height object with an integer

The above `operator+` function will only work for adding two Height objects. What if we want to add a Height object with an integer, for something like adding 3 inches to my height like the following?

```
    newHeight = myHeight + 3;
```

It will not work because the second operand for the function requires a Height object but we are passing an integer type. In order to do this we'll need to overload another `operator+` function and pass in an int type instead of a Height type for the second operand like the following for the class declaration

```
    Height operator+(const int& in);
```

and the following for the class definition

```
/*************************************************************
```



```
 * operator+
 * ht1 + int
 */
Height Height::operator+(const int& in) {
   Height h;
   h.feet = feet;
   h.inches = inches + in;
   if (h.inches >= 12) {
      h.feet++;
      h.inches = h.inches - 12;
   }
   return h;
}
```

3. Adding an integer with a Height object – Friends of Operators

Now, what if we want to add an integer with a Height, like

```
   newHeight = 3 + myHeight;
```

Again this will not work with the above two overloaded `operator+` functions. We need to overload yet another `operator+` function. However, this time it is much more different.

Most of the arithmetic and logical operators are binary operators, meaning that they need two operands to work on. For example, for the + operator, we are adding two operands

operand1 + operand2

The first operand (operand1) is referred to as the *implicit* operand which we have been calling it "myself." The second operand (operand2) is referred to as the *explicit* operand which we have to explicitly pass into the function.

When we overloaded the + operator for the Height class for doing both

myHeight + yourHeight

and

myHeight + 3

the implicit operand in both cases is the same type as myself. In other words, the implicit operand myHeight and the operator that we are overloading are both of the same Height object. However, when we do

3 + myHeight

the implicit operand is an int. It is another type and not the same type as myself. If you were to overload the + operator for the int object then it'll be alright, but we can't add things to the built-in int object. The solution C++ provides is to make our overloaded + operator for our Height object a friend of the int object by using the **friend** keyword.

Our Height class declaration will have

```
    friend Height operator+(const int& in, const Height& ht);
```

Since with just the **friend** keyword we do not know the function is a friend of what type, so we need to pass in not only the second explicit operand but also the first implicit operand. So the first argument that we pass into the function is for the first implicit operand and the second argument is for the second explicit operand.

The Height class definition for this function is

```
Height operator+(const int& in, const Height& ht) {
    Height h;
    h.feet = ht.feet;
    h.inches = in + ht.inches;
    if (h.inches >= 12) {
        h.feet++;
        h.inches = h.inches - 12;
    }
    return h;
}
```

Notice that in the definition we do not use the **friend** keyword, we use it only in the declaration of the function.

4. Overloading << and >>

For doing cout and cin, we want to overload the two operators << and >> respectively for our Height class. Notice when we do

cout << myHeight

or

cin >> myHeight

it is just like with the binary + operator with two operands where the first implicit operand is cout/cin and the second explicit operand is myHeight. And again we see that the first implicit operand is not the same type as myself (which is a Height). Both cout and cin are an iostream; cout is an `ostream` and cin is an `istream`. So again, to overload the << or the >> operator for our Height class we need to make it into a friend function of the iostream object. The declarations for both are as follows

```
    /*************************************************************
     * friend operator<<
     * out << ht;
     */
    friend ostream& operator<<(ostream& out, const Height& ht);

    /*************************************************************
     * friend operator>>
```



```
     * cin >> ht;
     */
    friend istream& operator>>(istream& in, Height& ht);
```

The definitions are as follows

```
/*************************************************************
 * friend operator<<
 * out << ht;
 */
ostream& operator<<(ostream& out, const Height& ht) {
	out << ht.feet << " feet " << ht.inches << " inches";
	return out;
}

/*************************************************************
 * friend operator>>
 * cin >> ht;
 */
istream& operator>>(istream& in, Height& ht) {
	in >> ht.feet >> ht.inches;
	return in;
}
```

5. Complete file listings

Height.h file

```
#ifndef  __HEIGHT__
#define __HEIGHT__
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
    int feet;
    int inches;
public:
    /*************************************************************
     * operator==
     * ht1 == ht2
     */
    bool operator==(const Height& ht);
    /*************************************************************
     * operator+
     * ht1 + ht2
     */
    Height operator+(const Height& ht);
    /*************************************************************
     * operator+
     * ht + int
     */
    Height operator+(const int& in);

    /*************************************************************
     * friend operator+
     * int + ht
     */
    friend Height operator+(const int& in, const Height& ht);

    /*************************************************************
     * friend operator>>
     * cin >> ht;
     */
    friend istream& operator>>(istream& in, Height& ht);

    /*************************************************************
     * friend operator<<
     * out << ht;
     */
    friend ostream& operator<<(ostream& out, const Height& ht);
};
#endif
```

Height.cpp file

```
////////////////////////////////////////////////////////////
// Class definition
```



```cpp
#include "Height.h"

/*************************************************************
 * operator==
 * ht1 == ht2
 */
bool Height::operator==(const Height& ht) {
    if (feet == ht.feet && inches == ht.inches) {
        return true;
    }
    else {
        return false;
    }
}

/*************************************************************
 * operator+
 * ht1 + ht2
 */
Height Height::operator+(const Height& ht) {
    Height h;
    h.feet = feet + ht.feet;
    h.inches = inches + ht.inches;
    if (h.inches >= 12) {
        h.feet++;
        h.inches = h.inches - 12;
    }
    return h;
}

/*************************************************************
 * operator+
 * ht + int
 */
Height Height::operator+(const int& in) {
    Height h;
    h.feet = feet;
    h.inches = inches + in;
    if (h.inches >= 12) {
        h.feet++;
        h.inches = h.inches - 12;
    }
    return h;
}

/*************************************************************
 * friend operator+
 * int + ht
 */
Height operator+(const int& in, const Height& ht) {
    Height h;
    h.feet = ht.feet;
    h.inches = in + ht.inches;
    if (h.inches >= 12) {
        h.feet++;
        h.inches = h.inches - 12;
    }
    return h;
```



```
}

/*************************************************************
 * friend operator>>
 * cin >> ht;
 */
istream& operator>>(istream& in, Height& ht) {
   in >> ht.feet >> ht.inches;
   return in;
}

/*************************************************************
 * friend operator<<
 * out << ht;
 */
ostream& operator<<(ostream& out, const Height& ht) {
   out << ht.feet << " feet " << ht.inches << " inches" << endl;
   return out;
}
```

main.cpp file

```
////////////////////////////////////////////////////////////
// main
#include <iostream>
using namespace std;
#include "Height.h"

int main() {
   Height myHeight, yourHeight;

   cout << "My height " << endl;
   cin >> myHeight;
   cout << myHeight;

   cout << "Your height " << endl;
   cin >> yourHeight;
   cout << yourHeight;

   if (myHeight == yourHeight) {
      cout << "We have the same height!" << endl;
   } else {
      cout << "We have different heights" << endl;
   }

   Height newHeight;
   newHeight = myHeight + yourHeight;
   cout << "myHeight + yourHeight = " << newHeight;

   newHeight = myHeight + 3;
   cout << "myHeight + 3 = " << newHeight;

   newHeight = 4 + myHeight;
   cout << "4 + myHeight = " << newHeight;
}
```


6. *Exercises* (Problems with an asterisk are more difficult)

1. Overload the - operator for the Height class where the implicit (first) operand is of a different class such as int.

2. Overload the + operators for the Temperature class where the implicit (first) operand is of a different class such as int.

3. Overload the << and >> operators for the Temperature class.

4. Overload the + operator for the Circle class where the implicit (first) operand is of a different class such as int.

5. Overload the << and >> operators for the Circle class.

6. Overload the + operators for the Date class where the explicit (second) operand is an int. How do you want the user to interpret this operation? In other words, what does it mean to have a date object + an int?

7. Overload the << and >> operators for the Date class.

