Classes (Part 3)
Overloading Operators

1. Introduction

In our Height class so far we have defined a member function EqualHeight to test for equality between two height variables like this

```
    if (myHeight.EqualHeight(yourHeight)) ...
```

However, this is not the standard way in C++ to test for equality. If we want to use the same format as for other built-in types then we want to do

```
    if (myHeight == yourHeight) ...
```

In order to do this, we need to tell the system how to test for equality between two height values using the == operator. This is referred to as **overloading operator**. C++ allows you to redefine how standard operators work when used with class objects. We want to overload an operator such as == by adding a user-defined operation to the operator so that this operator will also work for our class. This is nothing more than just creating a function for the operator.

When we create a function, we need to give that function a name. Now, we want to create a function for the == operator, so we should use == as the function name, but we can't because that is not a valid name. The solution that C++ provides is to prepend the operator with the keyword **operator**. So the function name for overloading the == operator is just **operator==**. The complete overloaded == operator function declaration is

```
bool operator==(Height ht);
```

and the function definition is

```
bool Height::operator==(Height ht) {
    if (feet == ht.feet && inches == ht.inches) {
        return true;
    } else {
        return false;
    }
```

Notice that except for the name of the function `operator==`, everything else is exactly the same as in the EqualHeight function earlier.

Now, in the main code we can do the equality test the standard way

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

Recall that in the original function EqualHeight we call it like

```
    if (myHeight.EqualHeight(yourHeight)) ...
```

Since we have now changed the name of the function to `operator==` therefore, we can also call it like

```
    if (myHeight.operator==(yourHeight)) ...
```

But it is better to call it using the preferred standard way

```
    if (myHeight == yourHeight)
```

Overloading the << and >> operators is a little bit different and will be discussed in a subsequent document.

2. Optimizing parameter passing

When we pass an object as a parameter into a function as in

```
bool operator==(Height ht);
```

the object ht is passed by *value*, meaning that a complete copy of the Height object is made. Typically, objects are quite large so each time when the function is called and it has to make a copy of the entire object it slows down the performance. To improve on the performance we want to pass the object in by *reference* rather than by *value*. A reference is just a pointer to the original storage location. To do this we simply add an ampersand sign & in front of the variable name like

```
bool operator==(const Height& ht);
```

Remember however, that when we pass by reference, the function can change the value of the variable and this change is reflected outside in the caller. So to prevent the function from accidentally modifying the variable, we also add the keyword **const**.

Another thing that you often see in production code is another **const** at the end for the entire function (not just for the variable that you pass in) as in

```
bool operator==(const Height& ht) const;
```

This is done for member functions that should not modify its data members. With this second **const** keyword, the compiler will prevent the programmer from accidentally modifying the data members.

Remember to make the same changes in both the declaration and the definition of the function.

3. Returning a Height object

The == operator returns a bool value to say whether the two height objects are equal or not. Consider the + operator for adding two height objects. What type should the + operator return? If you add one height with another height the result should also be a height value. So the + operator should return a height object.

```
/**************************************************************
  * operator+
  * h1+h2;
  */
Height Height::operator+(const Height& right) const {
   Height result;
   int in;

   // first convert to inches before adding
   in = (feet * 12 + inches) + (right.feet * 12 + right.inches);
   // convert result back to feet and inches
   result.feet = in / 12;
   result.inches = in - result.feet * 12;

   return result;
}
```

4. Complete file listings

Height.h file

```
#ifndef  __HEIGHT__
#define __HEIGHT__

//////////////////////////////////////////////////////////////
// Class declaration

class Height {
private:
   int feet;
   int inches;
public:
   void InputHeight();
   void OutputHeight();
   bool operator==(const Height& ht) const;
   Height operator+(const Height& right) const;
};

#endif
```

Height.cpp file

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

//////////////////////////////////////////////////////////////
// Class definition

void Height::InputHeight() {
   cout << "Enter feet and inches? ";
   cin >> feet >> inches;
}

void Height::OutputHeight() {
   cout << feet << " feet " << inches << " inches" << endl;
}


// operator==
// h1 == h2;
bool Height::operator==(const Height& ht) const {
   if (feet == ht.feet && inches == ht.inches) {
      return true;
   } else {
      return false;
   }
}

// operator+
// h1 + h2;
Height Height::operator+(const Height& right) const {
   Height result;
   int in;

   // first convert to inches before adding
   in = (feet * 12 + inches) + (right.feet * 12 + right.inches);
   // convert result back to feet and inches
   result.feet = in / 12;
   result.inches = in - result.feet * 12;

   return result;
}
```

main.cpp file

```
//////////////////////////////////////////////////////////////////
// main

#include <iostream>
using namespace std;

#include "Height.h"

int main() {
   Height myHeight, yourHeight;

   cout << "My height " << endl;
   myHeight.InputHeight();
   myHeight.OutputHeight();

   cout << "Your height " << endl;
   yourHeight.InputHeight();
   yourHeight.OutputHeight();

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

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

1. Overload the > operator for the Height class so that you can do myHeight > yourHeight.
2. Overload the + operator for the Height class so that you can do myHeight + yourHeight. Note that this function returns a Height object.
3. Overload the == operator for the Circle class so that you can do circle1 == circle2.
4. Overload the >= operator for the Circle class.
5. * Overload the == operator for the Temperature class.
6. * Overload the + operator for the Temperature class for adding two temperature values. Note that this function returns a Temperature object.
7. * Overload the + operator for the Temperature class for adding a temperature value with an integer value like t1 + 3. This is interpreted as adding 3 degrees to the given temperature value. Note that this function returns a Temperature object.
8. Overload the == operator for the Date class.
9. * Overload the < operator for the Date class.
10. *** The + operator for the Height class returns a Height object. At this point with what you know so far, what can you do with this Height value that is returned by this + operator function? Can you store this Height value in a variable? Can you use cout to print this Height value out?