Classes (Part 5)
Constructors and Destructors

1. Constructors

A constructor is a member function that is automatically called when a class object is created, i.e., when you declare a variable of that class.

A constructor is used to initialize the data members of a new instance of the variable when it is declared.

The function name for the constructor is always the same name as the class name.

Below are two constructor functions for our Height class. The first one, with no arguments, will initialize the height variable to 0 feet and 0 inches. The second one, with two arguments, will initialize the height variable to the two values passed in.

```
////////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
   int feet;
   int inches;
public:
   Height();                  // constructor method version 1
   Height(int ft, int in);    // constructor method version 2
   ...
}
```

```
////////////////////////////////////////////////////////////
// Class definition
#include "Height.h"

Height::Height() {
   feet = 0;
   inches = 0;
}

Height::Height(int ft, int in) {
   feet = ft;
   inches = in;
}
```

In the main program, you can now declare your Height variables like this

```
   Height myHeight(5, 4), yourHeight;
```

myHeight(5, 4) will call the constructor function `Height(int ft, int in)` which will initialize the variable with the two values passed in, whereas yourHeight will call the constructor function `Height()` which will initialize the variable with 0 feet and 0 inches.

The above two methods, however, can be replaced by the following method, i.e., the method below can handle the calls for both of the above methods.

```
////////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
   int feet;
   int inches;
public:
    /*************************************************************
     * conversion constructor
     * Height ht(5, 4);
     */
    Height(const int ft = 0, const int in = 0);
}
```

```
////////////////////////////////////////////////////////////
// Class definition
#include "Height.h"

/*************************************************************
 * conversion constructor
 * Height ht();
 * Height ht(5, 4);
 */
Height::Height(const int ft, const int in) {
   feet = ft;
   inches = in;
}
```

In the declaration where we have the two arguments `const int ft = 0, const int in = 0`, the default values for ft and in are 0. If the user supplies two values then those two values are used, and if the user does not supply any values then 0 is used.

2. Destructors

A destructor is a member function that is automatically called when a class object is released, i.e., when a variable is no longer used. For example when a block is exited all the local variables declared inside that block is released.

Destructors are mainly used for releasing memory storage that is dynamically allocated using the **new** command. This will be discussed further when we cover dynamically allocated memory storage.

The function name for the destructor always starts with the ~ symbol followed by the class name.

```
////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
   int feet;
   int inches;
public:
   ~Height();   // destructor method
   ...
}
```

```
////////////////////////////////////////////////////////
// Class definition
#include "Height.h"

Height::~Height() {
   ...
}
```

3. Complete file listings

Height.h file

```
#ifndef  __HEIGHT__
#define __HEIGHT__
////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
   int feet;
   int inches;
public:
    /*************************************************************
     * conversion constructor
     * Height ht();
     * Height ht(5, 4);
     */
   Height(const int ft = 0, const int in = 0);

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

/*************************************************************
 * conversion constructor
 * Height ht();
 * Height ht(5, 4);
 */
Height::Height(const int ft, const int in) {
   feet = ft;
   inches = in;
}

/*************************************************************
 * 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;
}
```


4. Exercises (Problems with an asterisk are more difficult)

1. Add the constructor for the Height class. Add in extra code for error checking to make sure that the initialization values are valid.

2. Add the constructor for the Circle class. Add in extra code for error checking to make sure that the initialization values are valid.

3. Add the constructor for the Temperature class. Add in extra code for error checking to make sure that the initialization values are valid.

4. Add the constructor for the Date class. Add in extra code for error checking to make sure that the initialization values are valid.

