Classes

1. Introduction

The structure user-defined type only allows the inclusion of data members. There are times when we also want to include operations to perform on the data members. A **class** in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type which holds both **data members** and **member functions**.

- Data members are the data variables just like for a structure.
- Member functions are functions (also referred to as methods) that perform operations on the data members.
- Together these data members and member functions define the properties and behavior of an object.

In the Height data structure that we declared previously with the two int variables, feet and inches, we might want to perform various operations on the Height object such as inputting and outputting a Height object. Also, we might want to be able to compare between two Height objects. So in our Height class, in addition to declaring the two integer data members, we also will declare an input, an output and a compare member functions.

2. Class Declaration

Here's the syntax for creating a class. Keywords are bolded.

class class-name {
private:
    list of data members // variable declarations
public:
    list of member functions() // functions (methods) to access the data members
};

Although not necessary, the data members are usually listed under the **private** section meaning that outside code using this object does not have direct access to the data members. In order to access the private data members, outside code must use the **public** member functions. Outside code can use the member functions because they are listed under the public section. Our Height class declaration is shown next.

```
class Height {
private:
   int feet;
   int inches;
public:
   void InputHeight();
   void OutputHeight();
   bool EqualHeight(Height ht);

};
```


3. Class Definition

After declaring the member functions in the class, we need to define these member functions by writing the actual code for them. Here is the code for the three member functions:

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

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

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

Notice that we need to qualify the function names with the name of the class and two colons (`Height::InputHeight`) so that the system knows that this function is a member function of the class.

All the member functions have direct access to all the private data members in the same class. They can be accessed simply by just using the data member variable name.

4. Creating a Variable or Instance of an Object

It is important to note that the class declaration does not define a variable for storing the actual data. It simply tells the computer what the data object is made of. In essence, it creates a new user-defined data type of that given class name. To use this new data type, you need to declare a variable of this new data type, just as you would with any other variables and data types. Declaring a variable of an object type is also referred to as creating an instance of this object. For example, the following declares two variables (or two instances) called myHeight and yourHeight of type Height.

```
   Height myHeight, yourHeight;
```

5. Accessing Data Members and Member Functions

For a member function to access its own data member, you simply just use the data member variable name. For example, in the OutputHeight member function, it can directly access the feet and inches data members by just using their names as in the statement

```
   cout << feet << " feet " << inches << " inches" << endl;
```


However, for code that is outside of the class to access the class members, whether data members or member functions, we need to use the dot operator (.). We start with the name of the variable for that class object followed by the dot operator (.) and then the variable name of the data member or the member function name. Keep in mind, however, that outside code cannot access members that are in the private section. Since feet and inches are in the private section, therefore, you cannot do the following:

```
myHeight.feet = 5;
myHeight.inches = 4;
cout << myHeight.feet << " feet " << myHeight.inches << "inches";
```

Instead, you will have to use the public member functions, which means that you better have the needed public member functions to access the private data members. In our example, we use the public member functions InputHeight, OutputHeight and EqualHeight like the following

```
myHeight.InputHeight();
myHeight.OutputHeight();
yourHeight.InputHeight();
yourHeight.OutputHeight();
```

6. Different Instances of the Same Object

In the EqualHeight member function, we want to compare between two Height objects to see whether they are equal or not, and return true if they are equal or false if they are not equal. So we are working with two different instances of the same Height object. The question is how do we access or refer to the two different instances?

First, remember that to access a member function we need to use the variable name of the object followed by the dot and then the name of the member function, like

```
myHeight.EqualHeight
```

So the object that you are referring to (myHeight) is yourself. So in the EqualHeight function, when you use the data member variable name feet and inches, you are using the feet and inches of the instance myHeight.

Now, the function needs a second instance of a Height object (yourHeight) to do the comparison with, so you need to pass in this second instance. This is why the EqualHeight function requires one argument of type Height for passing in yourHeight for this second instance.

```
bool Height::EqualHeight(Height ht) {
```

The variable name used for this second instance is ht, so to access the feet and inches of this second instance we use ht.feet and ht.inches. So if you pass yourHeight to ht, then ht.feet and ht.inches will refer to yourHeight's feet and inches, whereas using just feet and inches by itself without the dot operator will refer to myHeight's feet and inches.

So to compare the feet and inches between the two different instances myHeight and yourHeight, we have

```
    if (feet == ht.feet && inches == ht.inches) {
```

7. Complete Code

Here's the complete code listing in main.cpp.

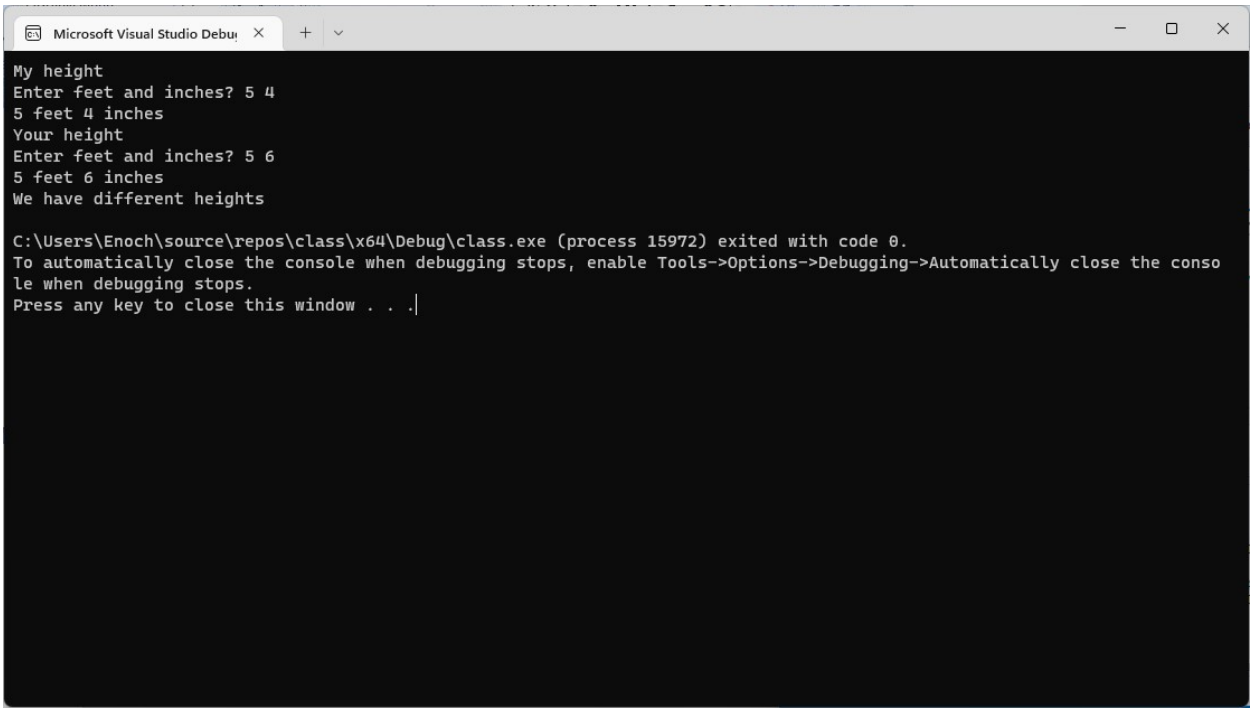
```
#include <iostream>
using namespace std;

////////////////////////////////////////////////////////////
// Class declaration
class Height {
private:
    int feet;
    int inches;
public:
    void InputHeight();
    void OutputHeight();
    Height AddHeight(Height ht);
    bool EqualHeight(Height ht);
};

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

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

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

////////////////////////////////////////////////////////////
// main
int main() {
    Height myHeight, yourHeight;
    cout << "My height " << endl;
    myHeight.InputHeight();
    myHeight.OutputHeight();
    cout << "Your height " << endl;
    yourHeight.InputHeight();
    yourHeight.OutputHeight();
    if (myHeight.EqualHeight(yourHeight)) {
```



```
        cout << "We have the same height!" << endl;
    } else {
        cout << "We have different heights" << endl;
    }
}
```

8. Output

Here's a sample run.



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

1. Create a Circle class with the following members:
   - A data member for storing the radius of a circle.
   - A member function for inputting a radius value.
   - A member function for outputting a radius value.
   - A member function that calculates and returns the area of the circle. The equation for the area is πr^2 where π is the constant 3.1415 and r is the radius.
   - A member function for testing whether two circles have the same radius or not. The function returns a Boolean value.
   - A member function for testing whether two circles have the same area or not. The function returns a Boolean value. This one is different from the previous question in that this one needs to call the area member function.

2. Write the code in main to test out the Circle class from question 1.

3. Create a Temperature class with the following members:
   - A data member for storing the degree of the temperature.
   - A data member for storing the scale (either a C, F, or K for Celsius, Fahrenheit, or Kelvin respectively).
   - A member function for inputting a temperature value. Remember that a temperature value consists of both a degree and a scale.
   - A member function for outputting a temperature value.
   - A member function for testing whether two temperature values are equal or not. The function returns a Boolean value.

4. Write the code in main to test out the Temperature class from question 3.

5. Declare a data structure for storing 10 temperature values.

6. Write the code in main to input 10 temperature values into the data structure declared in question 5 and then print out these 10 temperature values.

7. Continuing with question 6, input another temperature value and then search through the 10 temperature values to see if there is one that is equal. Print out an appropriate message.

8. Describe five other member functions that a user might like to have for the Temperature class.

