Assignment 3: Distance COP3330 Fall 2017

Due: Monday, October 16, 2017 at 11:59 PM

Objective

This assignment will provide experience with basic operator overloading.

Task

Your task will be to create a class called **Distance**, in the files `distance.h` and `distance.cpp`, which will involve a variety of operator overloads. A Distance object will store a quantity of distance in terms of miles, yards, feet, and inches. You will overload some basic operators for use with these objects including arithmetic, comparison, and insertion/extraction operators. These operators must work even for large distances (e.g. thousands of miles) and should not overflow the capacity of the storage variables (`int`) unless the number of miles is very close to the upper limit of int storage.

Program Details and Requirements

> The Distance class must allow for storage of a non-negative quantity of distance in terms of miles, yards, feet, and inches using integer precision. All values should be non-negative. The data should always be maintained in a simplified form. For example, if you have 14 inches, this should be expressed as 1 foot and 2 inches. You should create appropriate member data in your class, all of which must be private.

Remember that there are 12 inches in a foot, 3 feet in a yard, and 1760 yards in a mile. The only limit on the number of miles is that imposed by `int` storage (e.g. ~2 billion for 32 bit `int`).

Public Interface

1. *Constructors*
   - The class should have a default constructor (no parameters) which should initialize the object so that it represents the distance 0.
   - The class should have a constructor with a single integer parameter which represents a quantity of inches. This should be translated into the appropriate notation for a `Distance` object. Note: this will be a conversion constructor that allows automatic type conversions from `int` to `Distance`. If the parameter is negative, default the `Distance` object to represent 0.
   - The class should have a constructor that takes 4 parameters representing the miles, yards, feet, and inches to use for initializing the object. If any of the provided values are negative, default the `Distance` object to represent 0. If any of the provided values are too high (e.g. 13 inches), simplify the object to the appropriate representation.

   Examples:

```
Distance t;          // creates an object of 0 miles, 0 yards, 0 feet, 0 inches
Distance s(1234);    // creates an object of 0 miles, 34 yards, 0 feet, 10 inches
```

```
Distance r(-123);     // creates an object of 0 miles, 0 yards, 0 feet, 0 inches
Distance d(1234567); // creates an object of 19 miles, 853 yards, 1 feet, 7 inches

t = 4321; // conversion constructor allows this assignment.
          // t now stores 0 miles, 120 yards, 0 feet, 1 inch

Distance x(1, 3, 2, 7);   // 1 miles, 3 yards, 2 feet, 7 inches
Distance y(2, -4, 6, 8); // creates object representing 0, since -4 yards is not legal
Distance z(3, 5, 7, 9);   // 3 miles, 7 yards, 1 feet, 9 inches (simplified)
```

2. operator<<

This insertion operator is for the output of `Distance` objects. A distance object should be printed in the format `(Am By C' D")` where A is the number of miles, B is the number of yards, C is the number of feet, and D is the number of inches. The characters 'm' and 'y' should appear after the first two values respectively, and the markers ' and " are the standard notations for feet and inches. The whole output should be inside a set of parenthesis. Only print the first three values if they are non-zero, always print inches.

Examples:

`(2m 13y 2' 9")` means 2 miles, 13 yards, 2 feet, 9 inches
`(89m 175y 4")` means 89 miles, 175 yards, 0 feet, 4 inches
`(2' 10")` means 0 miles, 0 yards, 2 feet, 10 inches
`(0")` means a distance of 0

3. operator>>

The extraction operator is for reading `Distance` objects from an input stream. The format for the input of a Distance object is `A,B,C,D` where the user is to type the values as a comma-separated list consisting of miles, yards, feet, and inches respectively. You may assume that keyboard input will always be entered in this format (i.e. 4 integers separated by commas). This operator will need to do some error checking as well. If any of the input values are negative, this is an illegal `Distance` quantity, and the entire object should default to the value 0. If any of the values are over the allowable limit, then this function should adjust the Distance object so that it is in simplified form.

Examples:

`13,5,2,8 //` input translates to `(13m 5y 2' 8")`
`0,0,4,13 //` input translates to `(1y 2' 1")`

4. operator+
operator-

These operators allow for addition and subtraction of two quantities of distance. Results should always be returned in simplified form. For subtraction, if the first quantity of distance is less than the second (i.e. normally a negative quantity), return the `Distance` object 0 instead

Examples:

```
(3m 7y 1' 9") + (2m 6y 8") = (5m 13y 2' 5")
(3m 7y 1' 9") - (2m 6y 8") = (1m 1y 1' 1")
(1m 6y 9") + (2m 5y 2' 7") = (3m 12y 4")
(1m 6y 9") - (2m 5y 2' 7") = (0")
```

5. *operator**
   This operator allows a `Distance` object to be multiplied with an integer multiplier. The result should be expressed in simplified format. If the multiplier is negative or 0, return `Distance` object 0 instead.

   Examples:
   ```
   Disance d1(1, 500, 2, 5); // is (1m 500y 2' 5")
   cout << d1 * 3;           // yields (3m 1502y 1' 3")
   cout << d1 * 5;           // yields (6m 744y 1")
   ```

6. *operator<*
   operator>
   operator<=
   operator>=
   operator==
   operator!=
   Each of these operations should test two objects of type `Distance` and return true or false. You are testing the `Distance` objects for order and/or equality based on whether one quantity of distance is more (less than, equal to, etc) another.

7. *operator++*
   operator--
   These operations should handle increment and decrement respectively. You need to handle both the pre- and post- forms. These operators should have their usual meaning: increment will add 1 inch to the `Distance` object, decrement will subtract 1 inch. If the `Distance` object is already at 0, then decrement doesn't change it.

   Examples:
   ```
   Distance d1(2, 10, 2, 10);
   Distance d2(5, 51, 1, 1);
   cout << d1++; // prints (2m 10y 2' 10"), d1 is now (2m 10y 2' 11")
   cout << ++d1; // prints (2m 11y 0"),     d1 is now (2m 11y 0")
   cout << d2--; // prints (5m 51y 1' 1"),  d2 is now (5m 51y 1' 0")
   cout << --d2; // prints (5m 51y 11"),    d2 is now (5m 51y 11")
   ```

General Requirements

- No global variables, other than constants
- All member data of your class must be private
- The `const` qualifier should be used on any member functions where it is appropriate
- The only library that may be used in the class files is `<iostream>`
- Do not use language or library features that are C++11 only
- Since the only output involved is with operator<< your output should match mine exactly when running test programs
- When you write source code, it should be readable and well documented
- Your `distance.h` file should contain the class declaration only. The `distance.cpp` file should contain the member function definitions.

Testing Your Class

You are encouraged to create your own test programs but you can use the provided testing driver to get started. This does not give a compressive set of tests, but it should give you an idea of the types of calls that can be made.

Submitting

Archive your `distance.h`, `distance.cpp`, and `README` files into a simple tar ball (no compression). Submit to the assignment 3 link on blackboard. Make sure the submitted files are named as specified by the syllabus and this writeup.

General Advice

- Make sure to double check your blackboard submission to make sure everything works when downloaded.
- Email a copy of your finished homework files to your own FSU account. This email will have a time stamp that shows when they were sent and will also serve as a backup. Useful in case something happens to blackboard.
- Periodically (e.g. nightly) make a backup of your assignment to another machine (e.g. personal computer, linprog, email). Computers die and accidents happen, having a backup prevents you from having to start from scratch.
- Make sure to include the README file as specified in the assignment syllabus http://ww2.cs.fsu.edu/~dennis/teaching/2017_fall_cop3330/docs/syllabus.pdf

Assignment 3: Distance

COP3330 Fall 2017

Student Name: _____

Grader: _____

Grade: _____

Date Graded: _____

Description	Earned	Possible	Comments
1. Private Data		10	
2. Constructors		10	
3. operator<<		10	
4. operator>>		10	
5. operator+		10	
6. operator-		5	
7. operator*		5	
8. comparison operators		10	
9. operator++		5	
10. operator--		5	
12. README		10	
13. Submission Format		10	