University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics

Program 1: TimeSpan

Purpose

This programming assignment will provide exercises in designing classes with proper abstraction and encapsulation. Encapsulation and abstraction are key components of the C++ programming language as well as OOP in general. In addition, the programming assignment will require understanding of operator overloading and some use of the friend concept.

Problem: TimeSpan

Design and implement a TimeSpan class which represents a duration of time in hours, minutes, and seconds. The order hours, minutes, and seconds should be respected in the constructor.

As an example

    TimeSpan duration(1, 2, 3);

is a duration of time of 1 hour, 2 minutes and 3 seconds.

In the instances of the TimeSpan class you should store the values as integers in a normalized way. The number of seconds should be between -60 and 60; number of minutes should be between -60 and 60. You do not need to worry about integer overflow for very big TimeSpans.

<u>Accessor/Mutator functions required</u>

The **TimeSpan** class should implement the following member functions:

int getHours() const;	return the number of hours as an int
int getMinutes() const;	return the number of minutes as an int
int getSeconds() const:	return the number of Seconds as an int

bool setTime(int hours, int minutes, int seconds): set the number of hours, minutes, seconds and return true if successful.

<u>Constructors</u>

The class should define constructor(s) that receive various primitive types (specifically int, float, and double) and converts them to Int. Do appropriate rounding to maintain as much accuracy as possible.

TimeSpan(1.5, 4, -10) represents 1 hour, 33 minutes, 50 seconds.

If only one parameter is passed during initialization assume it is a second. If there are two assume minutes and seconds (in that order). Round the answer to the nearest second (for example, 50.9 seconds should be rounded to 51 seconds).

TimeSpan(7, 3) represents 7 minutes, 3 seconds.

Operators

The class must overload and implement the following math operators: **addition, subtraction, Unary Negation.**

The class must overload and implement the following comparisons: **==, !=**

The class must implement **+=** and **-=** assignment statements as well.

Stream I/O

The class must implement the << and >> operators to work on streams:

Input

Take as input three values: hour, minutes, seconds and create appropriate class. If the inputs are double or float, convert them to integers.

Output

For formatting the following:

```
TimeSpan duration1(1,2,3)
std::cout << duration1;
```

Should output:

```
Hours: 1, Minutes: 2, Seconds: 3
```

Please use this EXACT format.

Turn In

A .zip file which constains:

- TimeSpan.h
- TimeSpan.cpp
- TimeMachine.cpp (your driver)
- An executable TimeMachine

Please make sure to spell the getter and setter functions exactly as I have them above. Also, your overloads should follow appropriate signatures of they will not compile appropriately for the grader.

Also, your submitted code must compile successfully on a Linux operating system! If your code will not compile on a Linux machine there will be an automatic 10 points deduction in your grade!

Grading Criteria.

Compilation on a Linux operating system	**10**
Student testing all methods in main	**10**
Grader testing correctness of all methods including all edge cases	**30**
Appropriate use of const	**5**
Proper use of private/global variables/methods	**5**
Respecting abstraction principle and information hiding	**5**
Proper commands directives to the preprocessor	**5**
Proper use of friends methods	**5**
Correct output format	**5**
No redundant or repeated code	**5**
Adhering to Coding Guidelines	**15**
Total	**100**

Coding Guidelines Reminder

- // comments   /* or this way */
- { open and closed braces aligned and on separate lines}
- indentation
- PascalCasing for class names, namespaces, public functions
- camalCasing for parameters, accessors, private functions.
- Easy identifier names
- Not using underscores, hyphens, non-alphanumeric characters
- USE CAPS FOR CONST NAMES