University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Math

Program 1: Class Design / Operator Overloads

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 1: The Vending Bank

Design the interface for a class which models the coin-operated "bank" part of a Vending machine which sells snacks. You only need to design the interface (.h file) and not the implementation (.cpp file). Here is a start of VendingBank.h with one function already defined.

File VendingBank.h:

```
#pragma once
#include <string>
using namespace std;

class VendingBank
{

        public:
                VendingBank();
                VendingBank(int id);
                ~VendingBank();

                int getVendingBankID() const;

FILL IN PUBLIC FUNCTIONS HERE


        private:
FILL in
}
```

Problem 2: 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.

Accessor functions required

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.

Constructors

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).

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. Assume that these will be 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)
- TimeMachine.exe (if using VS) or an executable TimeMachine (if using Linux)
- VendingBank.h

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.

The following is an example **main()** you may use. However, when turning in, you must code your own **main()** to test all features of your TimeSpan class.

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

void main()
{
	int waitVar;

	TimeSpan dur1(77.4, 15, 6), dur2(127.86), dur3(8, -23, 0), dur4(0, 0, 0);
	TimeSpan dur5(-3, 73, 2), dur6(7, 35, 120), dur7, dur8;

	dur7 = dur1 + dur3;

	cout << dur1 << endl;
	cout << dur2 << endl;
	cout << dur3 << endl;
	cout << dur4 << endl;
	cout << dur5 << endl;
	cout << dur6 << endl;
	cout << dur7 << endl;

	dur7 += dur3;
	cout << dur3 << endl;
	cout << dur7 << endl;

	if (dur3 != dur6)
	{
		cout << "Durations are different." << endl;
	}
	else
	{
		cout << "Durations are the same" << endl;
	}
}
```