/
CS 52 Test 2
This document gives you specifications of class Distance and main function to be written. It has 12 sections (A to L), guiding you to write the entire program. You can write all code in one cpp file.

Only questions, that will be answered would be to clarify any part of the text in this paper. No compiling or debugging help is available as this is not lab. Rather, it is a test.This test paper presents a working plan and description of the class Distance.

Write, the following missing member and, friend headers and later member and friend implementation for the class Distance that has two data members; feet and inches: (1 feet = 12 inch).

- **Only one Explicit constructor is allowed. It takes default value for all arguments. Thus this acts as a default and explicit constructor together. Example of such a constructor is below.**

```
class Student
{
private:
string name;
int age;
double GPA;
public:
Student(string in_Name = "Name not set", int In_Age = 0, double In_GPA = 0.0)

};

//Implementation of constructor
Student:Student(string in_Name, int In_Age, double In_GPA)// Notice that there are no default args in the implementation file
{
    name = in_Name;
   age = In_Age;
   GPA = In_GPA;
}

int main( )
{
// All below constructor calls are valid due to default arguments passed to constructor
Student S1;
Student S2("Jim");
Student S3("Jim", 20);
Student S4("Jim", 30, 2.9);
}
```

- **overloaded friend functions operator +**
- **overloaded friend function operator -**
- **The toString function**
- **The extra member function normalize is optional. Writing normalize,however, may simplify your code.**

- **main function testing all class member and friend functions. Tests must have full code coverage, covering all possibilities.**
- **NO EXTRA MEMBERS ARE ALLOWED! Fifteen points will be taken off for each extra member added.**

***/**

//Add any necessary includes below. All function descriptions are ONLY done inside the document covering the class body below. You shoud print all of this test and use it when you write implementation of member functions. Reading from electronic screen is bewitching, and often people make reading errors. Such errors are reduced when reading from paper copy. So please do not put yourself at the risk of reading errors. Print this test and use it after printing.

```
class Distance
{
private:
        int feet;
        int inches;
public:
```

/* Task A: Write the header of the explixit constructor that takes default arguments for both constructor parameters. The constructor should do conversion from inch to feet, in case the client make a constructor call such as:Distance (3, 23); second argument being inches. In call such above feet shall be set to 4 and inches to 11. The constructor should take default arguments for its parameters, otherwise some automatic type conversions will not work.*/

***/* Task B*:** Write the header of the friend function overloaded + (plus operator) below which will allow the client to perform a sum like the following:
Distance D1, D2, D3;
D3 = D1 + D2;
or
D3 = 55 + D2;
or
D4 = D3 + 16;
You must make sure that D3 and D4 have inches less than 12 and all inches that are multiple of 12 are added to member feet. This friend function must return a const object. Both arguments passed to this friend function must be const objects passed by reference.*/

***/* Task C*:** Write the header of the friend function overloaded - (minus operator) which will allow the client to subtract one distance from another. For the sake of simplicity, if right operand of - operator is larger than the left operand , then obviouly the result will be a negative Distance. But function can just return the absolute value of the result in all cases so that both feet and inch members will always stay positive. In other words the operator – is simply returning the difference between two distances,
For example
Distance D1, D2, D3;
if (D2 >D1) then
you are free to have function return D2 – D1, even if client wrote the code as below
D3 = D1 - D2;
The below should also work.
D3 = 55 - D2;

or
D4 = D3 - 16;
You must make sure that D3 and D4 have inches less than 12 and all inches multiple of 12 are added to member feet. . This friend function must return a const object. Both arguments passed to this friend function must be const objects passed by reference.
***/**

/* *Task D*: Write the header of toString member function. It must return a const string and must have an end const modifier. **See more details in the implementation section.**
***/**

/**Task E**:* Write the header of (optional) member function normalize, which ascertains that data member inches remains less than 12, and feet field is adjusted up accordingly.*/
};

//Implementation of class Distance members and friends

/**Task F*:** Write the full body of the Explicit constructor described in Task A, on previous page */

/*
***Task G*:** Write the implementation of the friend function plus (+) operator described in the Task B on previous page:*/
/*
Task H: Write the implementation of the friend function minus (-) operator described in the Task C on previous page:*/
***/**
Task I: Write the implementation of toString member function described in Task D, previous page. Additional details are below. The table below gives the code fragment example and what would be outputted using toString.*/

Code Fragment	**Output to Console**
Distance D(12, 25); //(12 is feet, 25 is inches) **cout<<D2.toString();**	**Feet: 14** **Inches: 1** **----------**

/*
***Task J*:** Write the implementation of (optional) normalize function as described in Task E on page 1.*/

/*
Task K: Write the main function that tests the class. Testing class means that all member, friend functions and constructors are tested.Use toString member function to print object after every object creation and change in it's state. Confirm that each constructor and member function is working to it's specification.
***/**