CS 122
Programming Assignment 1

Objectives:

1. Create a simple, reusable framework that can serve as a starting point for future programming assignments (this part is provided for you, and you may copy it verbatim if desired).

2. Expand upon the framework provided to create a C++ program which performs simple input, output, and data manipulation functions.

Specifications:

1. You are first required to type (or download from the website) the attached skeleton.cpp file in an editor of your choice, saving the file (as "skeleton.cpp", of course) when finished. Next, verify that the program will compile and run. If using Linux, you can 'cd' to the directory where you saved the file, and then type "g++ skeleton.cpp" to compile the program, and "./a.out" to run it. If all goes well, you should see the text "Print instructions to the user here..." appear in your console. If using Windows, follow the instructions provided by your IDE in order to compile and then run the program.

2. Once you have verified that skeleton.cpp will compile and run on your system, reopen the file in an editor of your choice and select File -> Save As… and save your file as "homework1.cpp". You are now to edit this file to create a program that does the following:

    2a. Reads an integer (i.e. '-7'), a floating point number (i.e. '3.1415'), a character (i.e. 'x') , and a string (i.e. "look, a string") from the keyboard (you may use string.h and the provided string datatype if you wish).

    2b. Once this data has been gathered, the program is required to compute (and then display) the sum, difference, product, quotient, and remainder (use the modulus operator for this one) of the integer and the floating point number.

    2c. There is also to be a new function which will take as parameters the character and the string data and append a blank space followed by the character to the string, and then return the string (so if as in the above example, we have a character input of 'x' and a string input of "look, a string", your function is to return "look, a string x"). This

new string should then be displayed as well.

2d. Your program should continue reading sets of data until the user is tired of entering it. There are a number of ways in which you may address this requirement. For example, the very first thing you might want to do before reading any data sets is ask the user to input how many data sets they want to enter. Another method would be to have CLEARLY DEFINED values that the user can enter when they are ready to terminate the program.

2e. You ARE NOT required to perform error checking/handling (i.e you do not need to make sure that the user didn't just tell the program to try to divide '571' by '0'), though you may do so if you wish. For now it as acceptable to assume that the user knows enough to not enter data that will cause your program to crash. However, including in your instructions to the user and/or README file a statement about what sorts of things should not be entered into the program (i.e. a number being used as a divisor should not be 0, the number of data sets to be entered should not be negative, etc.) is always a good idea.

2f. You ARE required to follow modern coding and documentation style guidelines (see attached sheet), which includes (at minimum) printing concise instructions to the user when your program starts up, documentation of all functions in your program, and an informative README file.

3. You also must create a "README" file for your program. The file should contain, among other things, instructions on how to compile, run, and use your program, a description of what your program does as well as how it does it, as well as a discussion of any aspect of your program which you consider to be particularly interesting, unique, or clever. This is also where to note any known errors (pronounced "features") in the program, or to point out any error handling you decided to implement (or anything else you did above and beyond the base requirements), and to acknowledge any resources used to help you create the program.

Submission Guidelines:

Use whatever archiving software you prefer to archive your skeleton.cpp, homework1.cpp, and README files together. You may include a directory structure in your archive if desired, but it is not required. E-mail your archive to aroth@nmt.edu with the subject line of "[CS 122] Homework 1" by the due-date posted on the website.

Grading Criteria:

A fully working program is worth 60 points.

A useful and descriptive README file is worth 15 points.

Useful comments throughout your program are worth 15 points.

Clear, easy to read syntax (i.e. following coding style guidelines) is worth 10 points.

Enhancements:

For a maximum of an additional 10 bonus points, complete the following enhancement to the "homework1.cpp" program:

> Implement a class that has fields for all the data contained in one data set (i.e. an integer, a floating point number, a character, and a string), and member functions that perform the required computation and output. Data fields of your class should be declared as "private" members of the class, so you will need to implement get() and set() functions to access them. Member functions that perform computation and output should be "public" members of the class. Your program should then be modified so that it uses this class to store the data in a data set and to generate the required output.

CS 122
Coding Guidelines

What follows is a list detailing standards to be followed whenever you are programming for CS 122:

- Variable and function names are to be descriptive of what the variable represents, or what the function does (i.e. “int numDataSets” instead of “int n”).

- The first word in the name of a variable or function is to be written in all lowercase letters. The first letter of each subsequent word is to be capitalized. Special characters (underscores and the like) should not be used except for in special cases (i.e. “int someReallyLongVariableName”).

- Names of classes follow the same rule, with the exception that the first letter of the first word is also capitalized (i.e. “class MyClass”).

- Proper spacing and indentation should be used (see examples):

```
//GOOD
for(int loopCount = 0; loopCount < 10; loopCount++)
{
    if (loopCount >= 4) //countdown from 5 to 0
    {
            cout << ( 9 – loopCount ) << endl;
    }
}
```

```
//BAD
for(int loopCount=0;loopCount<10;loopCount++)
{
if(loopCount>=4) //countdown from 5 to 0
{
cout<<9–loopCount<<endl;
}
}
```

- Class data members should be declared as “private” members of the class unless there is a compelling reason to do otherwise (and occasionally, there will be). For every (or just about every) private data member in a class, there should be appropriate public get() and set() methods.

- Class function members should generally be declared as “public” members of the class (though there will be more frequent exceptions to this rule than to the above rule).

- Global variable declarations should be avoided except in rare instances (for example, having a global variable that specifies whether “debug” mode is on or off).

- Generally speaking, a function that does computation should produce no output (except possibly for debug output), and a function that generates output should perform little to no relevant computation. Instead, have the function that performs computation return data (or a data structure) that can then be passed to the function that generates your output. The purpose of this is to keep your modules as specific and reusable as possible.

- Every function in a program should include comments detailing, at a minimum, what the function does, what the parameters are, and what the return value is.

- In instances where a descriptive enough variable name cannot be found while still using a reasonable amount of characters, additional comments are to be used to explain what the variable is used for.

- Particularly confusing pieces of code are to be accompanied by additional comments explaining what the code is doing (if it’s confusing to you, as the person who wrote it, think how much worse it’s going to be for a complete stranger trying to just look at it and understand what’s going on).