The Assignment:

You will implement and test a small class called statistician, which is similar to some of the small classes in Chapter 2 of the text.

Purposes:

Ensure that you can write a small class that meets a ***precise*** specification.

Make sure you understand how to write a class that is separated into a header file and an implementation file.

Give you experience in using a test program to track down bugs in a class's implementation.

Before Starting:

Read all of Chapter 2.

Know how to compile and run C++ programs on your system.

Files that you must write:

1. stats.h
2. Download stats.h
3. : The header file for the new statistician class. Actually, you don't have to write much of this file. Add your name and other information at the top. If some of your member functions are implemented as inline functions, then you may put those implementations in this file too.
4. stats.cxx: (or stats.cpp)The implementation file for the new statistician class. **YOU** will write all of this file, which will have the implementations of all the statistician's member functions.

Other files that you may find helpful:

1. stattest.cxx
2. Download stattest.cxx
3. : A simple interactive test program.
4. statexam.cxx:
5. Download statexam.cxx:

6. A non-interactive test program that will be used to grade the correctness of your statistician class.

Hints:

1. Any file that is a .cxx could be renames .cpp
2. In you IDE, create a New Project (A command Line C++ program)
    1. Drag in the 3 files provided into your project
    2. Rename main.cpp to (or create) stats.cxx (stats.cpp)
3. You have two main()'s! One in stattest.cxx and on in statexam.cxx. You can only compile one at a time! I change the one in the exam file to main1() and did the interactive test until I was close. Then I changes it back to main() and I changed the one in the interactive test to main2().
4. you must read EXACTLY what is required in the header!

Submission:

Include required information in the .cpp file comments and in the output (Name and Class)

Run the statexam.cxx main() using your class. Copy the output to a separate file (like HeightOut.txt).

Upload all of the project files including your file, the original files (sine those cannot be changed) with the output files to Canvas, not Dora as the test tool indicates.

Upload as .cpp, .cxx, .docx, .h, .hpp, .rtf or .txt Only!

Programs must be submitted by the due date. Programs that are not 100% correct can be fixed after the due date. Multiple Submission are allowed AND encouraged. All corrections are excepted within one week of the original due date.

And now a BIG bit of guidance from the author:

The Statistician Class
Discussion of the Assignment

As indicated above, you will implement a new class called statistician, using a header file (most of which is written for you) and an implementation file (which you will write by yourself). The statistician is a class that is designed to keep track of simple statistics about a sequence of real numbers. There are two member functions that you should understand at an informal level before you proceed any further. The prototypes for these two functions are shown here as part of the statistician class declaration:

```
   class statistician

   {

   public:

     ...

     void next(double r);

     double mean( ) const;

     ...

   };
```

The member function "next" is used to give a sequence of numbers to the statistician one at a time. The member function "mean" is a constant member function that returns the arithmetic mean (i.e., the average) of all the numbers that have been given to the statistician.

Example: Suppose that you want a statistician to compute the mean of the sequence 1.1, 2.8, -0.9. Then you could write these statements:

```
   // Declares a statistician object called s

   statistician s;



   // Give the three numbers 1.1, 2.8 and -0.9 to the
statistician

   s.next(1.1);

   s.next(2.8);

   s.next(-0.9);
```

```
    // Call the mean function, and print the result followed by a
carriage return
```

```
    cout << s.mean( ) << endl;
```

The output statement will print 1.0, since 1.0 is the mean of the three numbers 1.1, 2.8 and -0.9.

Once you understand the workings of the next and mean member functions, you can look at the complete specification of the statistician class, which is in the file stats.h. Notice that the statistician class in this file is part of a namespace called main_savitch_2C. You should use this namespace for your statistician. In this file you will find a precondition/postcondition contract for all the statistician's member functions, including:

- A default constructor, which merely does any initialization needed for the statistician to start its work
- The next and mean functions, described above
- A constant member function called length, which returns the count of how many numbers have been given to the statistician
- Two constant member functions called minimum and maximum, which return the smallest and largest numbers that have been given to the statistician. (By the way, these two functions and the mean function all have a precondition that requires length( ) > 0. You cannot use these three member functions unless the statistician has been given at least one number!)
- A constant member function called sum, which returns the sum of all the numbers that have been given to the statistician. This function does NOT have a precondition. It may be called even if the statistician has NO numbers (in which case it should return 0).
- An overloaded operator == which tests to see whether two statisticians are "equal". The prototype is:
- int operator ==(const statistician& s, const statistician& t);

In order for two statisticians to be equal, they must have the same length (i.e., they have been given the same number of numbers). Also, if their length is greater than zero, they must also have the same mean, the same minimum, the same maximum, and the same sum. For example: Suppose that a statistician s has been given four numbers 1, 2, 3, 4. A second statistician t has been given four numbers 1, 1.5, 3.5, 4. Then the test (s==t) must return true since both s and t have equal values for all the member functions, as shown here:

1. length( ) and t.length( ) are both 4
2. mean( ) and t.mean( ) are both 2.5
3. sum( ) and t.sum( ) are both 10.0
4. minimum( ) and t.minimum are both 1
5. maximum( ) and t.maximum are both 4

- An overloaded + operator which has two statisticians as arguments, and returns a third statistician, as shown in this prototype:
- statistician operator +(const statistician& s, const statistician& t);
- An overloaded * operator which allows you to "multiply" a double number times a statistician. Here is the prototype:
- statistician operator *(double scale, const statistician& s);

This is not a member function. The result of a multiplication such as 2*s is a new statistician that looks as if it had been given all the numbers of s, multiplied by the constant 2. Examples: Suppose that s is a statistician that has been given 1, 2, 3, and u is another statistician. Then the assignment statement u=2*s will result in u behaving as if it had been given the numbers 2, 4, 6. As another example, the assignment statement u=-3*s will result in u behaving as if it had been given the numbers -3, -6, -9. Notice that neither + nor == are member functions. (See Section 2.5 in the class notes). The result of s+t is a new statistician that looks as if it had been given all the numbers of the sequence for s, followed by all the numbers of the sequence for t. For example: Suppose that we have three statisticians s, t, and u. The statistician s has been given the numbers 1, 2, 3; the statistician t has been given the numbers 4, 5. Then the assignment statement u=s+t will result in u behaving as if it had been given the five numbers 1, 2, 3, 4, 5.

Hints and Frequently Asked Questions

The Private Member Variables

Carefully read the class definition in stats.h. Notice how the private member variables are being used to keep track of information about the statistician's sequence of numbers. The statistician does NOT keep track of all the numbers in the sequence. There is no need to do so, and trying to do so can get you into trouble. Instead, it keeps track of only the information that is relevant to its member functions: How many numbers have been seen? What is the sum of those numbers? If you have seen at least one number, then what are the smallest and largest numbers that you've seen so far? These four items should be your only private member variables.

Be careful about how you set the private member variable that keeps track of the smallest number. My suggestion is that you do NOT have the constructor initialize this member variables (because when the constructor does its work, there have not yet been any numbers, so there is no smallest number). But part of the work of the "next" function is to correctly maintain the private member variables. This means that the first time that the next function is called, it should set the private member variable that keeps track of smallest values. Later, if next is called again with a smaller number, then the next function will change the member variable that is keeping track of the smallest value. (You'll have a similar process for the member variable that's keeping track of the largest value).

Check Boundary Values

Make sure that your + and * operators work correctly when the arguments are statisticians with no numbers.

Check Preconditions

Your implementations should use the assert function to check preconditions of all functions.

Input and Output

Your implementations must NOT produce any output to cout, nor expect any input from cin. All the interaction with the member functions occurs through their parameters.

Implement and Test Small Pieces

Don't tackle to whole project at once. Start by implementing what you can, using "stubs" for the harder functions. A "stub" is the implementation of a function with the lines of the body omitted. For example:

```
    void statistician::next(double r)

    {

    // This is just a stub, to be implemented later.

    }
```

A first implementation might have only:

- The constructor
- A simple version of next that increments the private member variable to keep track of how many numbers have been seen
- The length function

Even with just stubs, your stats.cxx file will correctly compile and link with the interactive test program,stattest.cxx.

Frequently Asked Questions

1. I run the stattest (or statexam) and the program crashes with a failed assertion. Is it a good idea to remove the assertions (which I put in stats.cxx to check preconditions).

ANSWER: No, leave those preconditions in there! The TA will clobber you (and I will too) if you delete the checks of the preconditions. Instead, you must find out where one of your functions is violating a precondition. Here is a typical example: Some students started by implementing the operator == along these lines:

```
bool operator == (const statistician& s1, const statistician&
s2)

{

     return

     (s1.length( ) == s2.length( ))

     &&

     (s1.sum( ) == s2.sum( ))

     &&

     (s1.minimum( ) == s2.minimum( ))

     &&

     (s1.maximum( ) == s2.maximum( ));

}
```

The problem with this implementation is that the operator == is allowed to be called even if s1 or s2 or both are empty. In such a case, the function will eventually get down to the test (s1.minimum( ) == s2.minimum( )) and...assertion failed! because you cannot call minimum for an empty statistician.

How do you fix this problem? In your operator == you should start with a test to see whether s1 or s2 is empty (and handle those cases in a way that does not call minimum() or maximum()).

MORAL: The functions you write can call other functions, but they must be careful to not violate preconditions.

2. How should my constructor initialize the private member variables tiniest and largest?

ANSWER: There are several solutions. One idea is to not initialize them at all. In this case, you must be careful to make sure of two things: (A) When the first number is given to the next function, it puts that first number into both tiniest and largest. (B) None of the other functions ever try to use tiniest or largest for an empty statistician.

3. What other functions might need special cases for an empty statistician?

ANSWER: Well, any function that accesses tiniest, largest, minimum() or maximum() probably needs a special case. Sometimes the special case can be simple. For example, the start of my operator + has two special cases:

```
if (s1.length( ) == 0)

     return s2;

if (s2.length( ) == 0)

     return s1;
```

...now the rest of my code doesn't need to worry about s1 or s2 being empty.

4. What strange things happen in the operator * when the scalar is negative?

ANSWER: Here's an example: Suppose that a statistician x has been given three numbers 10, 20 and 40. Then we execute the statement y = -1*x; The statistician y must act as if it had been given -10, -20 and -40 so y.minimum() will be -40 and y.maximum() will be -10.

5. I'm having trouble compiling or linking my stats.cxx with statexam.cxx or stattest.cxx

ANSWER: First of all, make sure that you've build and run Hello World! (Sorry to be so stern, but it's really needed.) Second: Make sure that all the code in your stats.cxx is in the namespace main_savitch_2C (look at the similar example of the point class on page 62).

6. When I write a friend function, the compiler still won't let the function access the private member variables of the statistician. Help!

ANSWER: See the second part of the answer to the previous question...The namespace!

7. Should I worry about warnings that occur when I compile.

ANSWER: Yes. At this point of the game, about 60% of warnings are errors. Spotting the cause of the warnings is an important part of learning about C++.

8. Should I worry about putting lots of comments in my program.

ANSWER: Not much of that is needed until a function gets longer than 10-15 lines.

- What causes these compilation errors:

```
   // Problem 1:

   s1.length( ) = s2.length( ) + s3.length( );

   // You can't assign to a function such as length. Try
assigning to

   // s1.sum (the variable) instead.


   // Problem 2:

   if (s1.length == s2.length)

   // You have to call the function. Try (s1.length() ==
s2.length()).
```