CSCE 240 – Project / Exam Three

Due: 11:59am on Wednesday, May 3. **Late exam submissions will not be accepted.**

This is an exam. As you work on these problems, you may use your textbook, class notes, and the recorded lectures. You may ask your instructor clarifying questions. You are not to discuss the problems with other students or seek help from other individuals. All work submitted must be your own. All code submitted will be examined for plagiarism and violations will be reported to the office of Student Conduct and Academic Integrity.

Test all of your code on a Linux lab computer. All source files submitted must compile and run on a Linux lab computer chosen by the instructor. Submissions that do not compile on the Linux workstation will receive no compilation or execution/correctness points.

Problem 1

Deliverable: *problem1.h*

Purpose: Create a template function named Median that takes an unsorted array as its first argument, and an integer holding the number of elements in the array as its second argument. The function should return the median of the values in the array.

Specifications:

- *problem1.h* must not have any **include** preprocessor directives (it should have header guard preprocessor directives).
- The function should not modify the values in the array argument (the first argument should be a const argument).
- The function should work for arrays of values of any type, as long as the type can be used with relational operators, arithmetic operators, and the assignment operator.
- Write the function's implementation in *problem1.h* and place the function in the *Project3Problem1* namespace.
- Submit your completed *problem1.h* file to the assignment in Blackboard. **Do not compress/zip the file.**

Examples:

Argument / Call:
```
const int intarray[5] = {17, 22, 7, 4, 30};
int x = csce240_exam::Median(intarray, 5);
```
Result: x will be initialized to 17

Argument / Call:
```
const double doublearray[6] = {-1, -4.6, -8, 2, 8.9, 37.1};
double x = csce240_exam::Median(doublearray, 6);
```
Result: x will be initialized to 0.5

Initial Testing:

problem1examples.cc includes tests for the examples shown above. You are encouraged to develop more rigorous tests for your function prior to submitting your solution.

Points:

style: 1 point

documentation: 1 point

clean compilation: 1 point

execution / correctness: 3 points

Problem 2

Deliverables: *factcheckedstatement.h* and *factcheckedstatement.cc*

Purpose: Create a *FactCheckedStatement* class that is a child of the *Sentence* class defined in the *sentence.h* header file included in *problem2.zip*. A *FactCheckedStatement* is a *Sentence* that ends with a period (not a question mark or exclamation point). A *FactCheckedStatement* object has a double variable as a private data member to hold a value between 0 and 1, inclusive. The double's value is a measure of how true the statement is determined to be. For example, 1 means that the statement is verified as 100% correct, and 0.5 means that the statement is 50% correct.

Read the comments in the included *factcheckedstatement.h* file for details on the functionality to add to the class.

Specifications:

- Define the *FactCheckedStatement* class in *factcheckedstatement.h* as a part of the *Project3Problem2* namespace.
- You can implement the functionality of the *FactCheckedStatement* class in *factcheckedstatement.h* OR in *factcheckedstatement.cc*. If you decide to implement the functions in the header file, upload an empty *factcheckedstatement.cc* file with your problem submission.
- Submit your completed *factcheckedstatement.h* and *factcheckedstatement.cc* files to the assignment in Blackboard. **Do not compress/zip the files.**

Initial Testing:

problem2.zip includes files with initial tests for your constructor, accessor function, mutator function, assignment operator, and stream insertion operator. To run these initial tests, ensure that all of the files from *problem2.zip* and your updated *factcheckedstatement.h* and *factcheckedstatement.cc* files are in the same directory.

At the command prompt

run the provided constructor / accessor tests by typing: **make testconstructor**

run the provided mutator function tests by typing: **make testsettruth**

run the provided assignment operator tests by typing: **make testassignment**

run the provided stream insertion operator tests by typing: **make testoutput**

You are encouraged to develop more rigorous tests. Your problem submission will be graded using modified versions of the provided tests.

Points:

style: 1 point

documentation: 1 point

clean compilation: 1 point

constructor execution / correctness: 1 point

GetTruth execution / correctness: 0.5 points

SetTruth execution / correctness: 0.5 points

assignment operator execution / correctness: 1 point

stream insertion operator execution / correctness: 1 point

Problem 3

Deliverable: *set.h*

Purpose: The attached *set.h* header file (in *problem3.zip*) contains the definition of a *Set* template class that is similar to the one implemented as an example during class. You will add an *Intersection* function that should return the intersection of the *Set* the function is called on with the *Set* argument. The prototype for the function is provided in the comment on line 120 of *set.h*. You will also overload the == operator whose prototype is provided in the comment on line 133.

Specifications:

- Additional details and examples regarding the expected functionality of the *Intersection* function are provided in the comments on lines 107-119 of *set.h*
- Your Intersection function must match the prototype provided on line 120 of *set.h*
- Implement the Intersection function in *set.h*
- Your == operator must match the prototype provided on line 133 of *set.h*
- Implement the == operator in *set.h*
- You will add code to *set.h*, but do not change any of the existing code in *set.h*.
- The provided class contains two constructors, a destructor, an overloaded assignment operator, a *SetValues* function, a *Print* function, an *IsASubset* function, an *IsAnElementOf* function, and a *GetCardinality* function. Read the code and comments in *set.h* for details. You can use (call) any of these functions as needed.
- Submit your completed *set.h* file to the assignment in Blackboard. **Do not compress/zip the file.**

Initial Testing:

No test files are provided. You are encouraged to create your own tests for the member function to include checking that your code works as expected for the examples given in the comments on lines 111-119 and 127-132.

Points:

style: 1 point

documentation: 1 point

clean compilation: 1 point

execution / correctness of the Intersection function: 2 points

execution / correctness of the == operator: 2 points