Data Structures and Algorithms Assignment for Enterprise Web Development students

Programming Assignment 1: Unique Integers

Task Description: Using any programming language of your choice, **read** a list of integers from an input file. Generate an output file having a list of unique integers present in the input file.

Instructions

1) Download sample data for this assignment from this location.
   Organize the code and the sample input into the following locations:
   /dsa/hw01/code/src/
   /dsa/hw01/sample_inputs/
   /dsa/hw01/sample_results
   However, you can choose to organize your code and data in your own way but be share to organize things properly so that It be easy to understand your work.

2) Read a file that has one integer on each line. The integer can be positive or negative.
```
5
   14
5
   -9
62
  -1
-9
-9
```
3) For each input file in the sample folder, you need to output a result file which contains a list of the unique integers in this file. For example, for input file "sample_input_02.txt", your result should be in sample_results/sample_input_02.txt_results.txt
4) The integers in result file must be sorted in increasing order.
5) There must be one line in result file for each unique integer.
6) For example, if the input is as shown in number "2" above, the result must be:
```
-9
-1
5
14
62
```
   Note that the digits 5 and -9 appeared multiple times in the input but have been printed only once.

7) Few sample input files and result files are given in the UniqueInt Sample data file for test purpose.
8) Your code must also handle following variations in the input file:
   a) Integers in each line can have a white space before or after them. A whitespace is limited to one or more tab, and space characters.
   b) If there are any lines with no inputs or white spaces, those lines must be skipped. See example input files.
   c) If there are any lines with two integers separated by white space, those lines must be skipped.
   d) If there are any lines that contain a non-integer input, those lines must be skipped. See example input file
      - Non-integer input includes alphabets, punctuation marks, non-numeric values, floating point numbers.

How to write your code:

1) Your code needs to be implemented in the file called UniqueInt with the appropriate extension based on your selected programming language, you can have the following function in the file:
   UniqueInt::processFile(std::string inputFilePath, std::string outputFilePath)
   UniqueInt::readNextItemFromFile(FILE* inputFileStream)
   2) You can create other functions/classes as needed in your project folder.

Note:
The integers in the input file will range from -1023 to 1023. A possible solution can be to implement a boolean array containing a true value if an integer is seen and a false value if the integer is not seen before.

Grading method:

1) If your code runs and generates an output file in the correct format for each input file, you get submission points.
2) If your method generates correct results for each test file, you get points for correctness.
3) We will review your source code to examine the internal documentation and award points for proper use of meaningful internal documentation.
4) We will measure the memory consumption in Bytes (submitted_memory). We will take the memory used by our implementation (our_memory). Memory score will be based on the ratio: (our_time /submitted_time) * max score for memory. If your memory usage is less than our method, you get more points. The maximum points you can get here is "max score for memory".
5) We will measure the run time in milliseconds (submitted_time). We will take the time used by our implementation (our_time). Your run-time score will be based on the ratio: (our_time /submitted_time) * max score for run-time. The maximum points you can get here is "max score for run-time".
6) Your code or implementation should show the memory usage and runtime on each sample data file you run your code against.
7) See table below for max score on run-time and memory.

Kindly note that for any programming language you decided to use to tackle this problem, you are not allowed to use the Standard built in Libraries (Array, List, Reverse, Sort etc.) of the programming language, you are meant to implement your own function/methods/classes to solve the problem. All general libraries like the ones that can allow you read file, declare data types, read input and display output from the users are allowed. Be sure to check with your facilitators for allowed libraries in the programming language you have decided to use.

Table 1: Marking Rubrics

Item	**Criteria**	**Points Awarded**
Output: Does the code run? Are the output files generated with the filenames and formats as specified in the problem?	Points obtained	
	Max Points	***4***
Correctness: Is the right output generated?	Points obtained	
	Max Points	***10***
Quality internal documentation. Does documentation aid code understandability?	Points obtained	
	Max Points	***8***
Timing	Time taken to run (ms)	
	Comparison with other students (divide by min of others)	

	Max. Timing score	***9***
Memory	Memory consumed (Bytes)	
	Comparison with other students (divide by min of others)	
	Max. Memory score	***9***
	Total points obtained	
	Total for Task	**40**