Data Structures and Algorithms for Engineers

Programming Assignment 3: Unique Integers Hard Version

Task Description: Read a list of integers from an input file. Generate an output file having a list of unique integers present in the input file.

Note: Read the instructions carefully before you attempt to write the solution.

Instructions

1) Download the code and sample data for this assignment from this location.
   Unzip this file to a folder on the VM(or similar environment). Organize the code and the sample input into the following locations:
   /home/dsa/UniqueInt_hard/code/src/
   /home/dsa/ UniqueInt_hard /code/bin/
   /home/dsa/ UniqueInt_hard /sample_inputs/

2) Implement code to:
   a) 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
   ```
   b) 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_inputs/sample_input_02.txt_results.txt
   c) The integers in result file must be sorted in increasing order.
   d) There must be one line in result file for each unique integer.
   e) For example, if the input is as shown in bullet number "2a" 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.

   f) A few sample input files and result files are given in zip file.
   g) Your code must also handle following variations in the input file:
      i) 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.
      ii) If there are any lines with no inputs or white spaces, those lines must be skipped. See example input file "sample_input_02.txt"

iii) If there are any lines with two integers separated by white space, those lines must be skipped.
iv) If there are any lines that contain a non-integer input, those lines must be skipped. See example input file "sample_input_03.txt"
- 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 UniqueInt.cpp within the following two functions:
   UniqueInt::processFile(std::string inputFilePath, std::string outputFilePath)
   UniqueInt::readNextItemFromFile(FILE* inputFileStream)
2) You can create other functions/classes within the files UniqueInt.h and UniqueInt.cpp. You must NOT create any other source files. These are the only two source files you will submit. We will make a call to the processFile function to grade your work.

Note:

The integers in the input file will range from INT_MIN to INT_MAX, i.e. from -2147483648 to 2147483647. 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. If this method is used, the Boolean vector will need to be of size: 2* 2147483648, i.e., 4MB. While most computers can handle this memory, the problem can be solved using less memory. We expect you to implement a solution that uses less memory without compromising on the run time. You can re-use your work in the BitVector assignment to implement a more optimal solution.

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) See table below for max score on run-time and memory.

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**