Homework 7: Containers

CSCI 60

Due before class Wednesday, February 26, 2020

Krehbiel

Overview. This week's task is to write two programs using the `vector`, `set`, and `map` classes in the C++ Standard Template Library. The first program will compare the use of `vector` vs `set` in processing a large text file. For the second program, you will read the contents of a data file into a `map` that you will use to design a console-based UI to allow a user to query a password-protected database.

Submission instructions. Upload your completed implementations of the two starter files – `hw7q1.cpp` and `hw7q2.cpp` – keeping the original filenames as usual. You should not have to add any include directives to the starter files, since Q1 should only use vectors and sets and Q2 should only use a map.

Question 1 specifics. Design and analyze three algorithms for the same task: read the words in a file into a data structure, and output the alphabetized words line-by-line to a separate file without any duplicates. Implement the three algorithms described below in the stub functions provided for you in the starter code, and give your analysis in the header comment section of your program.

- Analyze (5 points): Scan the algorithm descriptions below, and before you start your implementation, hypothesize which algorithmic approach will be fastest to run and fastest to code. Test for correctness on the small test file `"quickbrownfox.txt"` (you can compare against the provided `"output.txt"` file), and then record the results of your timed experiments on the large `"don_quixote.txt"` novel. How did the development process and empirical results compare with your expectations? Include a discussion in comments at the top of your Q1 file about runtime and ease-of-development considerations.

- Algorithm 1 (10 points): Implement `readWithVector1` to populate a vector word-by-word as you read the text file. Before you insert a word into the vector, check to make sure that it is not already in the vector. Then sort the vector using the `sort` function we used in class, and finally iterate through the contents of the vector, outputting them to the output file line-by-line.

- Algorithm 2 (10 points): Implement `readWithVector2` to populate a vector word-by-word as you read the text file. Put each word (even duplicates) into the vector as you read it. Then sort the vector using the `sort` function we used in class, and finally iterate through the contents of the vector, outputting them to the output file line-by-line. Now since the vector contains duplicates, you will have to ensure that you only output one copy of each word in the vector.

- Algorithm 3 (10 points): Implement `readWithSet` to populate a set word-by-word as you read the text file. Then iterate through the contents of the set, outputting them to the file line-by-line.

The `main` and `timedExperiment` functions are fully implemented and should not be modified. The `main` function is designed so that your program can be run by command line with or without arguments. To avoid command line arguments, modify the `INFILE` constant to use a different input file and modify the `EXPERIMENT` constant to test a different algorithm. To use command line arguments, specify a filename with the first argument following the executable and the experiment number with the second as follows:

```
MATH-L-01142:hw7 skrehbiel$ g++ hw7q1.cpp -std=c++11
MATH-L-01142:hw7 skrehbiel$ ./a.out quickbrownfox.txt 2
0.000661 seconds for alg 2 to process quickbrownfox.txt
MATH-L-01142:hw7 skrehbiel$
```

Question 2 specifics. (15 points) Create a password-protected database implemented with a map. Your input file will contain several student records, one per line. Each record will contain a name, student ID, password, and GPA for a single student, separated by spaces as in the provided `"records.txt"`. Recall that a map can have any key type and any value type. You may decide that it is helpful to define your own type to bundle multiple pieces of information within a single database entry, and you are welcome to do this with either a struct or a class.

Once you have read your input file into a map, design a program that asks a user through the console to enter the ID of a student and then prompts them to enter the password. If the correct password is given, the program will display the student's name and GPA. This process continues until the user enters 0 for the ID. You do not have to handle cases where the user enters empty strings or IDs not in the database. Given the provided input file, your program should work like the image below, where user console input has been boxed for visual clarity.

```
Enter an ID number (enter 0 to quit): 6547
Enter password: track
Name: Casey
GPA: 3.2
Enter an ID number (enter 0 to quit): 8123
Enter password: 1234
That was the wrong password.
Enter an ID number (enter 0 to quit): 8123
Enter password: 12345
Name: Izzie
GPA: 3.3
Enter an ID number (enter 0 to quit): 0
Goodbye!
```