Lab Homework

CS 2461

Instructions

Complete Sentence Count and ArrayList problems. For the questions, include a .pdf document in your submission with your answers. Submit a .zip file. Note: the only files you should modify are the .c files! Do not modify the .h files or the Makefiles. Write your own test cases and put them in test.c. Your submitted programs will be tested on the SEAS shell - make sure your code compiles and runs on it!

Sentence Count

Text processing and file I/O are important tools to be familiar with. For this portion of the assignment, you will write a function that counts the total number of sentences within a given input file name. You will also keep track of the number of questions (sentences ending in **?**), exclamations (sentences ending in **!**), and statements (sentences ending in **.**). You will then create a file with the given output file name and write the results into that file.

The function you will implement is as follows (defined in **sentence_count.h**):

```
void sentence_count(char* input_file_name, char* output_file_name);
```

sentence_count will read all words from the input file, and return the number of sentences. For our purposes, all sentences end in either **.**, **?**, or **!**. You should output the number of questions, exclamations, and statements into the output file. The output should look like the following:

```
Questions: 5
Exclamations: 14
Statements: 2
Total: 21
```

To compile and test your code, run the following commands in the directory containing your code files. To ensure that your code compiles correctly, run make. To run your tests in **test.c**, run ./test after you have run make. To remove a previous build, run make clean. **Do not modify** the Makefile. Your code must compile with the Makefile that was provided. Create your own input file for testing.

QUESTION 1: Analyze the time complexity of your function. If there are areas that could be improved, briefly describe how you might go about doing that.

ArrayList

Many of you are probably familiar with the ArrayList class in Java. For the final portion of this assignment, you will implement our own equivalent in C that is an ArrayList that stores elements of type int. An ArrayList is simply a wrapper around an Array that automatically resizes itself when it runs out of space. Our ArrayList will have an initial size of 10 (defined in **array_list.h**). For our purposes, when our array fills up, we will resize it to a new size of 1.5 * the old size (this is the same algorithm many Java implementations use).

`Struct` definitions and function declarations have been provided for you in **array_list.h**. All you are required to do is fill in the function in **array_list.c** as well as write tests that utilize your ArrayList in **test.c**. A few example tests have been provided for you.

The functions you will implement are as follows (defined in **array_list.h**):

```
      struct array_list* al_create(void);
      void al_add(struct array_list* list, int value);
      int al_get(struct array_list* list, int index);
      void al_set(struct array_list* list, int index, int value);
      void al_remove(struct array_list* list, int index);
      int al_size(struct array_list* list);
      void al_destroy(struct array_list* list);
```

`al_create` will allocate (hint hint) a new ArrayList with a capacity of `INITIAL_SIZE` elements and return a pointer to it.

`al_add` will add the value to the end of the ArrayList. If this causes the ArrayList to become full, resize the underlying Array to be 1.5 times its original size and copy all elements from the old Array into the new one.

`al_get` returns the value stored at the specified index in the list.

`al_set` sets the value at the specified index in the list, overwriting the existing value at that index.

`al_remove` removes the value at the specified index and shifts all elements to the right of that index to the left by one spot to fill the empty spot.

`al_size` returns the number of elements in the ArrayList that is passed in.

`al_destroy` will deallocate (free) the ArrayList and its Array.

To compile and test your code, run the following commands in the directory containing your code files. To ensure that your code compiles correctly, run make. To run your tests in **test.c**, run ./test after you have run make. To remove a previous build, run make clean. **Do not modify** the Makefile. Your code must compile with the Makefile that was provided.

QUESTION 2: Compare and contrast a linked list and an array list in terms of time complexity (big O) and storage. Which one has a faster index-based access time (i.e. which one has faster time to access the k-th element in the list)? Which one has faster add time (when adding to the end of the list)? Which one has faster remove time? Be sure to explain why for each question.