CS 211 Homework 1

Jeff Ames

Fall 2020

Introduction

This assignment will give you some initial experience with programming in C. Your task is to write the following small C programs.

- Please be careful to follow the I/O formats exactly.
- You can assume that inputs will be correctly formatted.

sortArray.c (20 points)

Write a program that will read an array from a file and sort the given array. It should return the array sorted with all even numbers in ascending order at the front followed by all odd numbers in descending order. You may assume that input array will not have more than 20 elements.

You can use any sorting algorithm you like, but you must implement it yourself (no using library sort functions).

Input format: Your program will take the file name as a command-line input. The first line in the file provides the total number of values in the array. The second line will contain a list of numbers separated by tabs. For example a sample input file `file1.txt` is:

```
6
25 10 1 99 4 2
```

Output format: Your output will be the sorted list of numbers, even numbers (ascending) and then odd numbers (descending), each separated by tabs.

```
$ ./sortArray file1.txt
2 4 10 99 25 1
```

linkedList.c (20 points)

In this part, you'll implement a linked list that maintains a list of integers in sorted (increasing) order. The list can contain duplicate elements.

Input format: This program takes a file name as an command-line argument. The file contains a number of lines of input (possibly 0). Each line contains a character, either `i` or `d`, followed by a tab character, and then an integer.

If the line starts with `i`, your program should insert that number in the linked list in sorted order. If it is already present, your program can insert it before or after the existing entry.

If the line starts with a `d`, your program should delete the value from the linked list. If there are duplicates, your program should delete only the first occurrence of the value. If the requested value is not present in the linked list, your program should ignore this line of the input.

Output format: If the input file doesn't exist, your program should print `error`.

Otherwise, your program should print two lines of output:

- the number of nodes in the list (including duplicates)
- a tab-separated list of the values of the linked list in order, with duplicate values only printed once

Example Execution

Let's assume we have 3 text files with the following contents:

```
file1.txt:
d	7
i	10
i	12
d	10
i	12
i	10
```

Then the result will be:

```
$ ./linkedList file1.txt
3
10	12
```

hashTable.c (20 points)

In this part, you will implement a hash table containing integers that has 10,000 buckets. For collision resolution, use chaining with a linked list. This means that if there is a collision at a particular bucket then you maintain a linked list of all values stored at that bucket. Use following hash function: key modulo the number of buckets.

Note that C's modulo operator returns a negative number for negative inputs, so `x % N` will be in the range $[-N+1, N-1]$ instead of $[0, N-1]$. For this assignment, please add N to negative results to get them into the nonnegative range. For example, your hash function should map an input of -5 to bucket 9995.

Input format: This program takes a file name as a command-line argument. Each line of the file contains a character, either `i` or `s`, followed by a tab and then an integer. For each line that starts with `i`, your program should insert that number in the hash table if it is not present. If the line starts with a `s`, your program should search the hash table for that value.

Output format: Your program should print two counts: (1) the number of insertions where a collision occurred (including due to duplicate values), and (2) the number of successful searches (where the value was present).

Example Execution

Let's assume we have a text file with the following contents:

```
file2.txt:
i 10
i 12
s 10
i 10010
s 5
s 10010
```

The the results will be:

```
$ ./hashTable file2.txt
1
2
```

stringOps.c (20 points)

This part requires you to read an input string representing a sentence, form a word whose letters are all the vowels in the given sentence, and print it. You should preserve the case of the vowels in the input.

Input and output format: This program takes a string of space-separated words, and should output a single word as the output.

```
$ ./stringOps Hello World.
eoo
$ ./stringOps Welcome to CS211
eoeo
```

bst.c (20 points)

In this part, you'll implement a binary search tree. The tree must satisfy the binary search tree property: the key in each node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree. You have to dynamically allocate space for each node and free the space for the nodes at the end of the program.

Input format:

This program takes a file name as a command-line argument. The file is either blank or contains successive lines of input. Each line starts with `i` followed by a tab and then an integer. Your program should insert that number in the binary search tree if it is not already there. If it is already present, you should ignore that line of input.

Output format:

If the input file does not exist, your program should print `error`.

Otherwise, your program should print a single tab-separated line containing all the elements in the tree in ascending order. To print elements in ascending order, you can do an in-order traversal of the tree (visit the left child, then the parent, and then the right child).

Example Execution

Let's assume we have a file file1.txt with the following contents:

```
i	5
i	3
i	4
i	3
i	6
```

Executing the program in the following fashion should produce the output shown below:

```
$ ./bst file1.txt
3	4	5	6
```

Getting started

You should first download and expand the autograder and provided files (on ilab):

```
tar xvf autograder.tar
tar xvf hw1-provided.tar
```

The provided tarball gives you a Makefile and template `.c` files.

You can also download these locally and then copy them to ilab using `scp`: `scp *.tar yourNetID@ilab.cs.rutgers.edu:cs211` (assuming you have a cs211 directory there).

Note that the autograder expects `hw1` to be in the same directory as `autograder.py`, so you will have to move things to match.

Submission

If you develop on your local machine, please be sure to test your code on ilab before submitting.

Please submit the assignment on Canvas as a tar file named `hw1.tar`. To create this file, put everything that you are submitting into a directory (folder) named `hw1`. Then, `cd` into the directory containing `hw1` (that is, `hw1`'s parent directory) and run the following command:

```
tar cvf hw1.tar hw1
```

To check that you have correctly created the tar file, you can copy it (`hw1.tar`) into an empty directory and run the following command:

```
tar xvf hw1.tar
```

This will re-expand your tar file and should create a directory named `hw1` in the (previously) empty directory with your code.

Autograder

We provide an autograder in `autograder.tar` to test your assignment. Executing the following command will create the autograder folder:

```
tar xvf autograder.tar
```

There are two modes available for testing your assignment with the autograder.

First mode

Testing when you are writing code with a `hw1` folder

(1) Let's say you have a `hw1` folder with the directory structure as described in the assignment.

(2) Copy the folder to the directory of the autograder

(3) Run the autograder with the following command

```
python autograder.py
```

It will run your programs and print your scores.

If you are running this on your local machine and the autograder has syntax errors, you may be using python 3. You can probably run `python2 autograder.py` instead to force it to use version 2.

Second mode

This mode is to test your final tar file before submission.

(1) Copy `hw1.tar` to the `autograder` directory

(2) Run the `autograder` with `hw1.tar` as the argument:

```
python autograder.py hw1.tar
```