CS 211: Computer Architecture
Programming Assignment 1: Introduction to C
Spring 2018

Instructor: Prof. Abhishek Bhattacharjee
Due: Feb 15, 2018 at 11:55 pm

Introduction

This assignment is designed to give you some initial experience with programming in C, as well as compiling, linking, running, and debugging. Your task is to write 5 small C programs. Each of them will test a portion of your knowledge about C programming. They are discussed below.

First: Decision making

Write a program to check whether an integer is odd or even.

Input and output format: This program takes an integer argument from the command line. It prints "odd" if the number is odd, "even" if the number is even, and "error" if no input is given. You can assume the input will be a proper integer (> 0). Thus, it will not contain any '.' or letters. The output should to be in lowercase, finish with a newline character ("\n") and contain no space.

Example execution:

```
./first 10
even
./first 5
odd
./first
error
```

Second: Looping

Write a program that checks whether a number is a prime number (2, 3, 5, 7, 11, ...) or composite.

Input and output format: This program takes an integer argument from the command line. It prints "yes" if the number is prime, "no" if the number is not prime and "error" if no input is given. You can assume the input will be a proper integer (> 0). Thus, it will not contain any '.' or letters.

Example execution:

```
./second 10
no
./second 7
yes
./second
error
```

Third: Linked list

Implement a linked list that maintains a list integers in sorted order. Thus, if the list initially contains 2, 5 and 8, and we insert 1, 3, and 10, then 1 will be inserted at the start of the list, 3 will be inserted between 2 and 5, and 10 will be inserted at the end.

Input format: This program takes a file name as argument from the command line. The file will have a number of lines. Each line contains a character (either 'i' or 'd') followed by a tab character and an integer number. For each of the line that starts with 'i', your program should insert the number in the linked list in sorted order if it is not already there. Your program should not insert any duplicate values. If the line starts with a 'd', your program should delete the value if it is present in the linked list. Your program should silently ignore it if the requested value is not present in the linked list.

Output format: At the end of the execution, your program should print all the values of the linked list in sorted order. The values should be in a single line separated by tabs. There should be no leading or trailing white space in the output. Your program should print "error" (and nothing else) if the file does not exist or it contains lines with improper structure. Your program should print a blank line if the input file is empty or the resulting linked list has no nodes.

Example execution: Let's assume we have 3 text files with the following contents. `file1.txt` is empty and,

```
file2.txt:
i 10
i 12
d 10
i 5

file3.txt:
d 7
i 10
i 5
i 10
d 10

./third file1.txt
```

```
./third file2.txt
5 12
./third file3.txt
5
./third file4.txt
error
```

Fourth: Hash table

Implement a hash table for integers. You can assume the hash table will store at most 1000 numbers. An important part of a hash table is collision resolution. In this assignment, we want you to use linear probing – if there is a collision at a location then you move forward by a given stepsize. Please use 1 for the stepsize. More information about linear probing can be found on Wikipedia: http://en.wikipedia.org/wiki/Linear_probing.

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

Output format: For each line in the input file, your program should print the status/result of the operation. For an insert, the program should print "inserted" is the value is inserted or "duplicate" if the value is already present. For a search, the program should print "present" or "absent" based on the outcome of the search. Your program should print "error" (and nothing else) if the file does not exist. The program should print "error" for input lines with improper structure.

Example execution: Let's assume we have 2 text files with the following contents. `file1.txt` is empty and

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

./fourth file1.txt

./fourth file2.txt
inserted
inserted
present
error
```

```
duplicate
absent

./fourth file3.txt
error
```

Fifth: Matrix addition

The fifth part requires you to add 2 matrices. The matrices must have the same dimensions (number of rows and columns) for addition to be valid. The output will be of the same dimension as well.

Input and output format: This program takes a file name as an argument from the command line. The first line of the file will contain two numbers (m and n) separated by a tab, where m is the number of rows, and n is the number of columns. This will be followed by m lines for first matrix followed by a blank line and second matrix. Each row will have n tab-separated integer values. You can assume the input will be properly structured for this part of the assignment. The program should output the result matrix in m lines. Each line will contain n tab-separated values. The upper bound for m and n is 2^16-1.

Example execution:

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

```
file1.txt:
3 3
1 2 3
4 5 6
7 8 9

6 4 2
2 3 4
1 3 5

./fifth file1.txt
7 6  5
6 8  10
8 11 14
```

Submission

Please submit the assignment using Sakai. Your submission should be a tar file named `pa1.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa1`. Then, `cd` into the directory containing `pa1` (that is, `pa1`'s parent directory) and run the following command:

```
tar cvf pa1.tar pa1
```

All files should be included directly in the `pa1` directory:

```
pa1
├── Makefile
├── first.c
├── first.h
├── second.c
├── second.h
├── third.c
├── third.h
├── fourth.c
├── fourth.h
├── fifth.c
└── fifth.h
```

To check that you have correctly created the tarball, you should copy it (`pa1.tar`) into an empty directory and run the following command:

```
tar xvf pa1.tar
```

This should create a directory named `pa1` in the (previously empty) directory, with all source and header files and the makefile.

Grading Guidelines

Your grade will be based on programmatic checking of your program. We will build a binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. In particular, your code should be adept at handling exceptional cases. For example, programs should not crash if the argument is not a proper number or the file does not exist.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result up to a 100% penalty. There should be no additional information or newlines. That means you will probably not get any grade if you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask.