CS 211: Computer Architecture, Spring 2017
Programming Assignment 1: Introduction to C (100 points)

Instructor: Prof. Santosh Nagarakatte

Due: February 10, 2017 at 5pm.

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 9 small C programs. Each of them will test a portion of your knowledge about C programming. They are discussed below.

First: Looping (5 points)

The first part requires you to write a program that checks whether a number is prime or not. A number is prime if it is divisible only by 1 and that number.

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 '.' or any letters. The command prompt in the examples below is indicated by '$'.

Example Execution:

```
$./first 10
no
$./first 7
yes
$./first
error
```

Second: Linked List (10 points)

In the second part, you have to implement a linked list that maintains a list of integers in sorted order. Thus, if the list contains 2, 5 and 8, 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 an argument from the command line. The file is either blank or contains successive lines of input. Each line contains a character, either 'i' or 'd', followed by a tab character and then an integer. For each of the lines that starts with 'i', your program should insert that 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 the line 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 spaces 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:

Lets assume we have 3 text files with the following contents:
"file1.txt" is empty
file2.txt:

```
i	10
i	12
d	10
i	5
```

file3.txt:

```
d	7
i	10
i	5
i	10
d	10
```

Then the result will be:

```
$./second file1.txt

$./second file2.txt
5 12
$./second file3.txt
5
$./second file4.txt
error
```

Third: Hash table (10 points)

In this part, you will implement a hash table containing integers. The hash table has 10,000 buckets. An important part of a hash table is collision resolution. In this assignment, we want you to use chaining with a linked list to handle a collision. This means that if there is a collision at a particular bucket then you will maintain a linked list of all values stored at that bucket. For more information about chaining, see `http://research.cs.vt.edu/AVresearch/hashing/openhash.php`.

For this problem, you have to use following hash function: key modulo the number of buckets.

Input format: This program takes a file name as argument from the command line. The file is either blank or contains successive lines of input. Each line contains a character, either 'i' or 's', followed by a tab and then an integer, the same format as in the Second Part. For each of the lines 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: For each line in the input file, your program should print the status/result of that operation. For an insert, the program should print "inserted" if 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 also print "error" for input lines with improper structure.

Example Execution:

Lets assume we have 2 text files with the following contents:
"file1.txt" is empty
file2.txt:

```
i	10
i	12
s	10
c	5
i	10
s	5
```

The the results will be:

```
$./third file1.txt

$./third file2.txt
inserted
inserted
present
error
duplicate
absent
$./third file3.txt
error
```

Fourth: Matrix addition (15 points)

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

Input and output format: This program takes a file name as argument from the command line. The first line of the file will contain two numbers separated by a tab, m and n, where m is the number of rows and n is the number of columns. This will be followed by the m lines of first matrix followed by a blank line and second matrix, also on m lines. Each row will have n tab-separated 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.

Example Execution:

Lets assume we have a text file, file1.txt, with the following contents:
file1.txt:

```
3	3
1	1	1
1	1	1
1	1	1

1	1	1
1	1	1
1	1	1
```

Then the output will be:

```
$./fourth file1.txt
2	2	2
2	2	2
2	2	2
```

Fifth:Matrix Multiplication (15 points)

The fifth part requires you to multiply 2 matrices. The matrices need to have consistent dimensions (number of rows and columns) for a valid multiplication: the number of columns of the first matrix must be equal to number of rows in the second.

Input and output formats: This program takes a file name as an argument from the command line. The first line of the file will contain 2 tab-separated numbers, m_1 and n_1, where m_1 is the number of rows in the first matrix and n_1 is the number of columns in the first matrix. This will be followed by the m_1 lines of first matrix, containing n_1 tab-separated values, followed by a blank line, then 2 tab-separated numbers, m_2 and n_2, where m_2 is the number of rows in the second matrix and n_2 is the number of columns in the second matrix. Again, this is followed by the m_2 lines of the second matrix. Each row will contain n_2 tab-separated values, the same as the first matrix. You can assume the input will be properly structured for this part of the assignment. The program should output the resulting matrix in m_1 lines. Each line will again contain n_2 tab-separated values.

Lets assume we have "file1.txt":

```
3 2
2 2
2 2
2 2

2 3
1 1 1
1 1 1
```

The output will be:

```
$./seventh file1.txt
4 4 4
4 4 4
4 4 4
```

Sixth: String Operations (5 points)

The sixth part requires you to read an input string representing a sentence, generate an acronym from the first letters of the words, and print it out. The letters of the acronym are the same case as their respective words in the input sentence.

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

```
$./sixth Hello World!
HW
$./sixth Welcome to CS211
WtC
$./sixth Rutgers Scarlet Knights
RSK
```

Seventh: String Operations II (5 points)

The seventh part requires you to read an input string representing a sentence, form a word whose letters are the last letters or punctuation of the words in the given sentence, and print it.

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

```
$./seventh Hello World!
o!
$./seventh Welcome to CS211
eo1
```

```
$./seventh Rutgers Scarlet Knights
sts
```

Eighth: Binary Search Tree (15 points)

In the eighth part, you have to 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 an argument from the command line. The file is either blank or contains successive lines of input. Each line starts with 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 binary search tree if it is not already there. If it is already present, you will print "duplicate" and not change the tree. 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 operation, the program should print either"inserted" with a **single space** followed by a number, the height of the inserted node in the tree, or "duplicate" if the value is already present in the tree. The height of the root node is 1. For a search, the program should either print ''present", followed by the height of the node, or "absent" based on the outcome of the search. Your program should print "error" (and nothing else) if the file does not exist or for input lines with improper structure.

Example Execution:

Lets assume we have a file file1.txt with the following contents:

```
i 5
i 3
i 4
i 1
i 6
s 1
```

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

```
$./eighth file1.txt
inserted 1
inserted 2
inserted 3
inserted 3
inserted 2
present 3
```

Ninth: Deletion with Binary Search Tree (20 points)

In the ninth part, you will extend the binary search tree in the eighth part to support the deletion of a node. The deletion of a node is slightly trickier compared to the search and insert in the eighth part.

The deletion is straightforward if the node to be deleted has only one child. You make the parent of the node to be deleted point to that child. In this scenario, special attention must be paid only when the node to be deleted is the root.

Deleting a node with two children requires some more work. In this case, you must find the minimum element in the right subtree of the node to be deleted. Then you insert that node in the place where the node to be deleted was. This process needs to be repeated to delete the minimum node that was just moved.

In either case, if the node to be deleted is the root, you must update the pointer to the root to point to the new root node.

Input format: This program takes a file name as argument from the command line. The file is either blank or contains successive lines of input. Each line contains a character, 'i', 's', or 'd', followed by a tab and an integer. For each line that starts with 'i', your program should insert that number in the binary search tree if it is not already there. If the line starts with a 's', your program should search for that value. If the line starts with a 'd', your program should delete that value from the tree.

Output format: For each line in the input file, your program should print the status/result of the operation. For insert and search, the output is the same as in the Eighth Part: For an insert operation, the program should print either "inserted" with a **single space** followed by a number, the height of the inserted node in the tree, or "duplicate" if the value is already present in the tree. The height of the root node is 1. For a search, the program should either print "present", followed by the height of the node, or "absent" based on the outcome of the search. For a delete, the program should print "success" or "fail" based on the whether the value was present or not. Again, as in the Eight Part, your program should print "error" (and nothing else) if the file does not exist or for input lines with improper structure.

Example Execution:

Lets assume we have a file file1.txt with the following contents:

```
i 5
i 3
i 4
i 1
i 6
i 2
s 1
d 3
s 2
```

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

```
./ninth file1.txt
inserted 1
inserted 2
inserted 3
inserted 3
inserted 2
inserted 4
present 3
success
present 2
```

Structure of your submission folder

All files must be included in the `pa1` folder. The `pa1` directory in your tar file must contain 9 subdirectories, one each for each of the parts. The name of the directories should be named first through ninth (in lower case). Each directory should contain a c source file, a header file (if you use it) and a make file. For example, the subdirectory first will contain, first.c, first.h (if you create one) and Makefile (the names are case sensitive).

```
pa1
|- first
   |-- first.c
   |-- first.h (if used)
   |-- Makefile
|- second
   |-- second.c
   |-- second.h (if used)
   |-- Makefile
|- third
   |-- third.c
   |-- third.h (if used)
   |-- Makefile
|- fourth
   |-- fourth.c
   |-- fourth.h (if used)
   |-- Makefile
|- fifth
   |-- fifth.c
   |-- fifth.h (if used)
   |-- Makefile
|- sixth
   |-- sixth.c
   |-- sixth.h (if used)
   |-- Makefile
|- seventh
   |-- seventh.c
   |-- seventh.h (if used)
   |-- Makefile
|- eigth
   |-- eigth.c
   |-- eigth.h (if used)
   |-- Makefile
|- ninth
   |-- ninth.c
   |-- ninth.h (if used)
   |-- Makefile
```

Submission

You have to e-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
```

To check that you have correctly created the tar file, 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.

The `pa1` directory in your tar file must contain 9 subdirectories, one each for each of the parts. The name of the directories should be named first through ninth (in lower case). Each directory should contain a c source file, a header file and a make file. For example, the subdirectory first will contain, first.c, first.h and Makefile (the names are case sensitive).

AutoGrader

We provide the AutoGrader to test your assignment. AutoGrader is provided as autograder.tar. 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 `pa1` folder

(1) Lets say you have a `pa1` 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 auto_grader.py

It will run your programs and print your scores.

Second mode

This mode is to test your final submission (i.e, pa1.tar)

(1) Copy pa1.tar to the auto_grader directory

(2) Run the auto_grader with pa1.tar as the argument.

The command line is

$python auto_grader.py pa1.tar

Scoring

The autograder will print out information about the compilation and the testing process. At the end, if your assignment is completely correct, the score will something similar to what is given below.

```
You scored
5.0  in  second
7.5  in  fourth
5.0  in  third
2.5  in  sixth
10.0  in  ninth
2.5  in  seventh
7.5  in  eighth
7.5  in  fifth
2.5  in  first
Your TOTAL SCORE =  50.0 /50
Your assignment will be graded for another 50 points with test cases not given to you
```

Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build the 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 not see or use your friend's code either partially or fully. We will run state of the art plagiarism detectors. We will report everything caught by the tool to Office of Student Conduct**.
- 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 a 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 **in up to 100% penalty**. Be

especially careful to not add extra whitespace or newlines. That means you will probably not get any credit if you forgot to comment out some debugging message.

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