CS 211: Computer Architecture, Fall 2019
Programming Assignment 1: Introduction to C (100 points)

Instructor: Prof. Santosh Nagarakatte

Due: September 30, 2019 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. Your program must follow the input-output guidelines listed in each section **exactly**, with no additional or missing output.

No cheating or copying will be tolerated in this class. Your assignments will be automatically checked with plagiarism detection tools that are pretty powerful. Hence, you should not look at your friend's code. See CS department's academic integrity policy at:
`https://www.cs.rutgers.edu/academic-integrity/introduction`

First: Right Truncatable Prime (10 Points)

You have to write a program that will read an array from a file and print if the numbers in the file are right truncatable primes.

A right truncatable prime is a prime number, where if you truncate any numbers from the right, the resulting number is still prime. For example, 3797 is a truncatable prime number number because 3797, 379, 37, and 3 are all primes.

Input-Output format: Your program will take the file name as input. The first line in the file provides the total number of values in the array. The subsequent lines will contain an integer value.

For example a sample input file "file1.txt" is:

```
3
397
73
47
```

Your output will be a yes/no for each value in the input file.

```
$./first file1.txt
no
yes
no
```

We will not give you improperly formatted files. You can assume all your input files will be in proper format as above.

Second: Linked List (10 points)

In this 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 the number of nodes in the list in the first line of the output and all the values of the linked list in sorted order in the next line. 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. Your program should print 0 followed by 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
0

$./second file2.txt
2
5 12
$./first file3.txt
1
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 1000 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`.

A hash table can be implemented in many ways in C. You must find a simple way to implement a hash table structure where you have easy access to the buckets through the hash function. As a reminder, a hash table is a structure that has a number of buckets for elements to "hash" into. You will determine where the element falls in the table using the hash function.

You must not do a linear search of the 1000 element array. We will not award any credit for O(n) time implementation of searches or insertions in the common case.

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 contains successive lines of input. Each line 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: 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. You can assume that the program inputs will have proper structure.

Example Execution:

Lets assume we have a text file with the following contents:

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

The the results will be:

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

Fourth: Matrix Multiplication (10 Points)

This program will test your ability to manage memory using **malloc()** and provide some experience dealing with 2D arrays in C.

Your task is to create a program that multiplies two matrices and outputs the resulting matrix. The input matrices can be the same or different sizes.

Input-Output format: Your program will take the file name as input. The first line in the file will provide the number of rows and columns in the matrix separated by a tab. The subsequent lines will provide the contents of the matrix. The numbers in the same row are tab separated and the rows are separated with new lines. This will be followed by the same format for the dimensions and content of the second matrix.

For example, a sample input file "file1.txt":

```
2 3
1 2 3
4 5 6
3 2
1 2
3 4
5 6
```

The first number (2) refers to the number of rows and the second number (3) refers to the number of columns in the matrix. The dimensions of the of the first matrix will be 2x3 and the second matrix will be 3x2. The output on executing the program with the above input is shown below. The outputted numbers should be tab separated in the same row with a newline between rows. There should not be extra tabs or spaces at the end of the line or at the end of the file.

```
$./fourth file1.txt
22 28
49 64
```

We will not give you improperly formatted files. You can assume all your input files will be in proper format as above with no matrix having 0 rows or columns.

For matrices that cannot be multiplied your program should output "bad-matrices".

Fifth: Matrix Squares(10 points)

A magic square is an arrangement of the numbers from 1 to n^2 in an (n x n) matrix, with each number occurring exactly once, and such that the sum of the entries of any row, any column, or any main diagonal is the same.

An example of a Magic Square is as such:

```
2 7 6
9 5 1
4 3 8
```

In this case, the sum of all entries in a given row, column or main diagonal is equal to 15.

More examples and information about magic squares can be found here: `http://mathforum.org/alejandre/magic.square/adler/adler.whatsquare.html`

Input-Output format:

Your program should accept a file as command line input. The format of a sample file "file1.txt" is shown below:

```
3
8 1 6
3 5 7
4 9 2
```

The first number (3) corresponds to the size of the square matrix (n). The dimensions of the matrix will be n x n. You can assume there will be no malformed input and the matrices will always contain valid integers.

If the given matrix follows the rules for a magic square, your program should output **"magic"**. If the matrix thats given is valid, but does not follow rules for a Magic Square, your program should output **"not-magic"**.

Example Execution

A sample execution with above input file "file1.txt" is shown below:

```
$./fifth file1.txt
magic
```

Sixth: Pig Latin (10 Points)

For this part of the assignment, you will need to write a program that reads an input string representing a sentence, and convert it into pig latin. We'll be using two simple rules of pig latin:

1. If the word begins with a consonant then take all the letters up until the first vowel and put them at the end and then add "ay" at the end.
2. If the word begins with a vowel then simply add "yay" to the end.

For this problem vowels are defined as: a, e, i, o, and u. There will only be characters in your input (no numbers or punctuation).

Input-Output format: This program takes a string of space-separated words and should output the same space-separated words translated into pig latin.

Example 1:

```
$./sixth Hello and welcome to computer architecture
elloHay andyay elcomeway otay omputercay architectureyay
```

Example 2:

```
$./sixth a program
ayay ogrampray
```

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.

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.

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 4
```

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 Makefile. 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
5.0  in  fourth
5.0  in  third
5.0  in  sixth
10.0  in  ninth
2.5  in  seventh
7.5  in  eighth
5.0  in  fifth
5.0  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`.
- Your compilation command with gcc should include the following flags: **-Wall -Werror -fsanitize=address -std=c11**
- You should test your code as thoroughly as you can. For example, programs should *not* crash with memory errors.
- 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.
- **Your folder names in the path should have not have any spaces. Autograder will not work if any of the folder names have spaces.**

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