CS 211: Computer Architecture, Spring 2024
Programming Assignment 1: Introduction to C (50 points)

Instructor: Prof. Santosh Nagarakatte

Due: February 9, 2024 at 5pm Eastern Time.

Introduction

The goal of this assignment is to get you started with programming in C, as well as compiling, linking, running, and debugging. Your task is to write 5 small C programs. Your program must follow the input-output guidelines listed in each section **exactly**, with no additional or missing output. You can assume that we will provide well-defined test cases.

No cheating or copying will be tolerated in this class. If you use any large language models (LLMs) such as ChatGPT/Bard/LLama or other models and as a result, your code is similar to another student's code in the class, it will be considered a violation of academic integrity. You should not be using large language models to copy-paste your code. Your assignments will be automatically checked with plagiarism detection tools that are pretty powerful. Hence, you should not look at your friend's code or copy-paste any code from LLMs or the Internet. See CS department's academic integrity policy at:
`http://academicintegrity.rutgers.edu/`

First: Is the Input a Product of 2 or 3 Given Numbers? (5 Points)

You have to write a program that given an array of integers determines if a particular integer input being queried is a product of 2 or 3 numbers in the array. If it is such a product, then you have to output yes. Otherwise, you will output no.

Input-Output format: Your program will take the file name as input. The first line in the input file provides the total number of integers in the input array. The next line will provide the list of these input integers. The third line in the input file provides the number of inputs queried. The subsequent lines are the queries that will the provide the specific input and the number 2 or 3. For example, if you have a query line of the form: 21 2, then you are checking if 21 is a product of 2 numbers from the input integer array. If so, output yes.

Here is a sample input file. Let us call file1.txt

```
5
3 7 8 11 17
4
21 2
12 2
1309 3
15 3
```

In the above file, the input array has 5 integers whose entries are 3, 7, 8, 11, and 17 respectively. There are 4 queries being done. The first query asks if 21 is a product of 2 integers in the array. The output is yes because $21 = 3 * 7$. In contrast, the answer is no for the query: 12 2. This is because 12 is not a product of any two integers in the input array.

Your output will contain the same number of lines as the number of query lines in the input file. Each line will either say `yes` if the corresponding input in the query is a product of the specified numbers or `no` if the corresponding input is not a product of the specified numbers.

The sample execution is as shown.

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

We will not give you improperly formatted files. You can assume that the files exist and all the input files are in proper format as above. See the submission organization format at the end of the assignment for more details.

Second: Ordered Linked List (10 points)

In this part, you have to implement a linked list that maintains a list of integers in sorted order. For example, if a list already 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 contains successive lines of input. Each line contains a string, either `INSERT` or `DELETE`, followed by a space and then an integer. For each of the lines that starts with `INSERT`, 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 `DELETE`, 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. After every `INSERT` and `DELETE`, your program should print the content of the linked list. The values should be printed in a single line separated by a single space. There should be no leading or trailing white spaces in each line of the output. You should print `EMPTY` if the linked list is empty.

Output format: At the end of the execution, your program should have printed the content of the linked list after each `INSERT` or `DELETE` operation. Each time the content is printed, the values should be on a single line separated by a single space. There should be no leading or trailing white spaces in each line of the output.You should print `EMPTY` if the linked list is empty. You can assume that there will be at least one `INSERT` or `DELETE` in each file.

Example Execution:

Lets assume we have 2 text files with the following contents:

```
file1.txt:
INSERT 1
INSERT 2
DELETE 1
INSERT 3
INSERT 4
DELETE 4
INSERT 5
DELETE 5
file2.txt:
INSERT 1
DELETE 1
INSERT 2
DELETE 2
INSERT 3
DELETE 3
INSERT 4
DELETE 4
INSERT 5
DELETE 5
```

Then the result will be:

```
$./second file1.txt
1
1 2
2
2 3
2 3 4
2 3
2 3 5
2 3
$./first file2.txt
1
EMPTY
2
EMPTY
3
EMPTY
4
EMPTY
5
EMPTY
```

Third: Matrix Exponentiation (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 computes M^n where M is a square matrix (the dimensions of the matrix will be $k \times k$ where k is the number of rows) and a number $n \geq 0$. In summary, you need to multiply the matrix with itself n times.

Input format: The program will take the file name as input. The first line in the file will provide the number of rows in the matrix. The subsequent lines will provide the contents of the matrix. The numbers are tab separated. The last line in the file after the contents of the matrix will contain the exponent n. For example, a sample input file "file.txt":

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

The first number (3) refers to the number of rows in the square matrix. The dimensions of the matrix will be 3×3. The exponent is 2. Hence, the program is required to compute M^2. You can assume that the input will be properly formatted. The output on executing the program with the above input is shown below. The output numbers should be tab separated. There should not be extra tabs or spaces at the end of the line or the end of the file.

```
30 36 42
66 81 96
102 126 150
```

Fourth: Binary Search Tree (10 Points)

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.

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
```

Fifth: Matrix Determinant(15 points)

In linear algebra, the determinant is a value that can be computed with a square matrix. The determinant describes some properties about the square matrix. Determinants are used for solving linear equations, computing inverses, etc, and is an important concept in linear algebra. In the fifth part of the assignment, you will write a program that computes the determinant of any $n \times n$ matrix. You will have to carefully manage `malloc` and `free` instructions to successfully compute the determinants.

Determinant

Given a square $n \times n$ matrix M, we will symbolize the determinant of M as $Det(M)$. You can compute $Det(M)$ as follows:

1×1 **matrix** The determinant of the 1×1 matrix is the value of the element itself. For example,

$$Det(\begin{bmatrix} 3 \end{bmatrix}) = 3$$

2×2 **matrix** The determinant of a 2×2 matrix can be computed using the following formula:

$$Det(\begin{bmatrix} a & b \\ c & d \end{bmatrix}) = ad - bc$$

For example,

$$Det(\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}) = 1 \times 4 - 2 \times 3 = 4 - 6 = -2$$

3×3 matrix The determinant of a 3×3 matrix can be computed modularly. First, let's define a 3×3 matrix:

$$M = \begin{bmatrix}a & b & c\\d & e & f\\g & h & i\end{bmatrix}$$

The formula for computing the determinant of M is as follows:

$$Det(M) = a \times Det(M_a) - b \times Det(M_b) + c \times Det(M_c)$$

The matrix M_a is a 2×2 matrix that can be obtained by eliminating the row and column that a belongs to in M. More specifically, since a is on the first row and first column, we eliminate the first row and first column from M:

$$\begin{bmatrix}a & b & c\\d & e & f\\g & h & i\end{bmatrix}$$

This gives us a 2×2 matrix for M_a:

$$M_a = \begin{bmatrix}e & f\\h & i\end{bmatrix}$$

M_b can be computed similarly. Since b is on the first row and second column, we eliminate the first row and second column from M:

$$\begin{bmatrix}a & b & c\\d & e & f\\g & h & i\end{bmatrix}$$

This gives us a 2×2 matrix for M_b:

$$M_b = \begin{bmatrix}d & f\\g & i\end{bmatrix}$$

M_c can be computed by removing the first row and the third column from M since c is on the first row and third column. Thus,

$$\begin{bmatrix}a & b & c\\d & e & f\\g & h & i\end{bmatrix}$$

$$M_c = \begin{bmatrix}d & e\\g & h\end{bmatrix}$$

Finally, the formula for computing the determinant of M is:

$$Det(M) = a \times Det(\begin{bmatrix}e & f\\h & i\end{bmatrix}) - b \times Det(\begin{bmatrix}d & f\\g & i\end{bmatrix}) + c \times Det(\begin{bmatrix}d & e\\g & h\end{bmatrix})$$

For example, we can compute the determinant of the following matrix,

$$M = \begin{bmatrix} 2 & 7 & 6 \\ 9 & 5 & 1 \\ 4 & 3 & 8 \end{bmatrix}$$

as follows:

$$Det(M) = 2 \times Det(\begin{bmatrix} 5 & 1 \\ 3 & 8 \end{bmatrix}) - 7 \times Det(\begin{bmatrix} 9 & 1 \\ 4 & 8 \end{bmatrix}) + 6 \times Det(\begin{bmatrix} 9 & 5 \\ 4 & 3 \end{bmatrix}) = 2(37) - 7(68) + 6(7) = -360$$

$n \times n$ **matrix** Computing the determinant of an $n \times n$ matrix can be considered as a scaled version of computing the determinant of a 3×3 matrix. First, let's say we're given an $n \times n$ matrix,

$$M = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix}$$

In essence, we have to pivot each element in the first row and create $(n-1) \times (n-1)$ matrix for each pivot element (in the case of computing the determinant of 3×3 matrix, we had M_a that corresponds to a, etc).

For example, when we pivot $x_{1,1}$, we create the corresponding $(n-1) \times (n-1)$ matrix for $x_{1,1}$ by deleting the 1^{st} row and 1^{st} column:

$$M_{1,1} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix} = \begin{bmatrix} x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & & \vdots \\ x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix}$$

Similarly, we can create $M_{1,2}$, $M_{1,3}$, … by pivoting $x_{1,2}$, $x_{1,3}$, and so on:

$$M_{1,2} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix} = \begin{bmatrix} x_{2,1} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,3} & \dots & x_{n,n} \end{bmatrix}$$

$$M_{1,3} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix} = \begin{bmatrix} x_{2,1} & x_{2,2} & x_{2,4} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,4} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,4} & \dots & x_{n,n} \end{bmatrix}$$

Finally, you can compute the determinant of M using the following formula:

$$Det(M) = x_{1,1} \times Det(M_{1,1}) - x_{1,2} \times Det(M_{1,2}) + x_{1,3} \times Det(M_{1,3}) - x_{1,4} \times Det(M_{1,4}) + x_{1,5} \times Det(M_{1,5}) \dots$$

The above formula can be shortened to the following formula:

$$Det(M) = \Sigma_{i=1}^{n}(-1)^{i-1}x_{1,i} \times Det(M_{1,i})$$

This general formula for computing the determinant of $n \times n$ matrix applies to all n. The formula for computing the determinant of 2×2 and 3×3 matrix is exactly the same as this formula.

Input-Output format:

Your program should accept a file as command line input. The format of a sample file `test3.txt` is shown below:

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

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 that n will not be greater than 20. The rest of the file contains the content of the matrix. Each line contains a row of the matrix, where each element is separated by a tab. You can assume that there will be no malformed input and the matrices will always contain valid integers.

Your program should output the determinant of the $n \times n$ matrix provided by the file.

Example Execution

A sample execution with above input file `test3.txt` is shown below:

```
$./fifth test3.txt
-360
```

Structure of your submission folder

All files must be included in the `pa1` folder. The `pa1` directory in your tar file must contain 5 subdirectories, one each for each of the parts. The name of the directories should be named first through fifth (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
```

Submission

You have to e-submit the assignment using Canvas. 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 5 subdirectories, one each for each of the parts. The name of the directories should be named first through fifth (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 a custom autograder to test your assignment. The custom autograder is provided as pa1_autograder.tar. Executing the following command will create the autograder folder.

```
$tar xvf pa1_autograder.tar
```

There are two modes available for testing your assignment with the custom 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 (i.e., pa1_autograder)

(3) Run the custom autograder with the following command

$python3 pa1_autograder.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 pa1_autograder directory

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

The command line is

$python3 pa1_autograder.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
7.5  in  fifth
2.5  in  first
Your TOTAL SCORE =  25.0 /25
Your assignment will be graded for another 25 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,undefined -g**
- 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.