Programming Assignment 1 (100 points)

CS211 Sections 10-12

Spring 2025

Abstract

Welcome to CS211. This assignment aims to get you comfortable with the C programming language and understand important concepts like basic syntax, reading from a file, and the use of stack and heap memory.

Contents

1 Introduction

1.1 Academic Integrity

- You can discuss the assignment with your classmates, but you **cannot copy code from your classmates**.
- You can use the internet to research the basics of the C programming language, the documentation for useful C functions, and to search for error messages you encounter. However, you cannot use websites that offer ready-made answers specific to this assignment and this class.
- We will use automated tools to check the assignments for identical and/or plagiarized code.
- It is not acceptable to share or post any course materials online without explicit permission from the instructor.
  - If you wish to use an online Version Control System, like Github, **ensure your repository is private**.

We take academic Integrity very seriously; please consult Rutgers Academic Integrity for more information.

1.2 iLab Machines

- If you do not already have access to the iLab machines, make an account here.
- more details here.
- We test all code on the iLab machines (iLab1-4). Before submission, check and make sure your code works as intended on the iLabs before submitting

1.3 Accessing the Assignment

The autograder, test cases and sample code will be on GitHub; you may clone it on the ilab machine with the following command:

```
git clone https://github.com/AFE123x/sp25-cs211-assignments.git
```

2 Part 1 - Finding Mersenne Prime (20 points)

2.1 Introduction

In this part of the assignment, you'll be finding the Mersenne prime of a perfect number.

2.1.1 What is a Perfect Number + Mersenne prime number

A perfect number is a number where the sum of the divisors equals the number. For example, 6 is a perfect number because the sum of its divisors besides itself is 6 ($6 = 3 + 2 + 1$). Another example is 28 ($28 = 14 + 7 + 4 + 2 + 1$)

There's a fascinating correlation to observe, where given a perfect number, x, $x = 2^{p-1} \times (2^p - 1)$, where $2^p - 1$ is a Mersenne prime number. In this part, given a number, you will find if it's a perfect sum and figure out the Mersenne prime.

2.2 Sample Input/Output

```
$ ./first 6
3
$ ./first 5
-1
$
```

If the number is perfect, you print out the Mersenne prime. If it's not perfect or the Mersenne prime doesn't exist, print out -1.

2.2.1 Explanation

`$ ./first 6`

First, we need to figure out if 6 is a perfect number.

- First, we find all the divisors of 6 besides 6: 1,2,3
- We add them all up, and determine if the sum equals 6
  - $1 + 2 + 3 = 6$
  - $6 == 6$, so we know that this is a perfect number.
- Next, we need to see if there exists a p, such that $2^{p-1} \times (2^p - 1) = 6$
- we can test all values of p, to find that p = 2: $2^{2-1} \times (2^2 - 1) = 2^1 \times (4 - 1) = 6$
- Finally, we can calculate, using p = 2, $2^p - 1 = 2^2 - 1 = 3$, which is our mersenne prime number.

```
$ ./first 5
```

First, we need to figure out if 6 is a perfect number.

- First, we find all the divisors of 5 besides 5: 1
- We add them all up, and determine if the sum equals 5
- 5 is not equal to 1, so it's not a perfect number, so we can print -1.

2.3 Assumptions

- You may assume that the input number is positive and greater than zero.

3 Part 2 - Rotate 2D Array (20 points)

In this part, you'll " rotate" a 2D array 90 degrees to the right.

3.1 sample input/output

Below are the contents of test1.txt

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

this text file tells us this is a 3 x 3 2D array with the numbers 1 - 9

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

The expected output of the program:

```
$ ./second test1.txt
7   4   1
8   5   2
9   6   3
```

3.2 assumptions

- You can assume that the input file is correctly formatted.
- For simplicity, you can assume that the input matrix is a square matrix (N rows, N columns).

4 Part 3 - Ordered Linked List(20 points)

In this part of the assignment, you'll implement a linked list of integers in sorted order (ascending). For example, if a linked list contains 2, 4, and 5, then 1 would be inserted at the start of the list, and 3 would be stored between 2 and 4.

4.1 Input/Output Format

contents of test1.txt

```
INSERT 1
INSERT 3
INSERT 5
INSERT 2
DELETE 1
INSERT 0
INSERT 10
DELETE 3
DELETE 2
DELETE 5
DELETE 10
DELETE 0
INSERT 1
```

Each line consists of a command and an integer:

- INSERT inserts a number into the linked list.
- DELETE deletes a number from the linked list.

4.1.1 Sample Execution

```
$ ./third test1.txt
1
1   3
1   3   5
1   2   3   5
2   3   5
0   2   3   5
0   2   3   5   10
0   2   5   10
0   5   10
0   10
0
EMPTY
1
$
```

If the linked list is empty after executing a command, you print "EMPTY"

4.2 Assumptions

- You may assume that the input file is formatted correctly.
- You can assume there are no duplicate numbers.
- If there's a DELETE command for a number not in the linked list, you can ignore it, and print the list as is.

5 Part 4 - Gaussian Elimination (20 points)

In this part of the assignment, you'll be implementing Gaussian Elimination. Gaussian elimination is a simple and fast method of solving a system of equations.

5.1 Introduction

You may be familiar with systems of equations:

$$x + y = 3$$
$$3x - 2y = 4$$

We can express this as a matrix:

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix}$$

We can perform a series of matrix operations to solve for x and y. In this assignment, you'll be solving this.

5.2 Matrix Operations

5.2.1 Operation one: row swapping

We can swap rows in our matrix:

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix} R_1 \leftrightarrow R_2 \begin{bmatrix} 3 & -2 & 4 \\ 1 & 1 & 3 \end{bmatrix}$$

5.2.2 Operation two: scaling row

We can scale the row by a constant factor.

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix} \frac{1}{3}R_2 \to R_2 \begin{bmatrix} 1 & 1 & 3 \\ 1 & \frac{-2}{3} & \frac{4}{3} \end{bmatrix}$$

5.2.3 Operation three: adding rows

We can add the values in one row to another row ($R_1 + CR_2 \to R_2$), where C is some constant we scale.

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix} R_1 + 1R_2 \to R_2 \begin{bmatrix} 1 & 1 & 3 \\ 4 & -1 & 7 \end{bmatrix}$$

5.3 Gaussian Elimination

We can perform a sequence of operations on a matrix to turn the original matrix into its reduced echelon form.

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

the transformed matrix tells us that $x = 2$ and $y = 1$
$x + y = 3 \rightarrow 2 + 1 = 3$
$3x - 2y = 4 \rightarrow 3(2) - 2(1) = 6 - 2 = 4$
Which are indeed correct

5.4 Sample Input/Output

contents of test1.txt

```
2
3
1   1   3
3   -2  4
```

Below is the expected output:

```
$ ./fourth test1.txt
1.000000    0.000000    2.000000
-0.000000   1.000000    1.000000
$
```

Let's look at how we got the answer:

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & -2 & 4 \end{bmatrix}$$

$$R_2 - 3R_1 \rightarrow R_2$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & -5 & -5 \end{bmatrix}$$

$$\frac{1}{-5} R_2 \rightarrow R_2$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \end{bmatrix}$$

$$R_1 - R_2 \rightarrow R_1$$

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

Which is the final matrix.

5.5 Assumptions and notes

- Use the double type for the matrix. when printing, use the "%lf" fornat specifier.
- You can assume the input file is correctly formatted.

6 Part 5 - Matrix Determinant (20 points)

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

6.1 Calculating the 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 a 1×1 matrix is the value of the element itself. For example,

$$det\left(\begin{bmatrix}3\end{bmatrix}\right) = 3$$

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

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

3×3 matrix: The determinant of a 3×3 matrix can be computed recursively. Let's use the following matrix:

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

We can express the determinant of M as follows:

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

Here, how do we get M_a? We eliminate the row and column on the matrix that a belongs to:

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

Now, we have a 2×2 matrix, which we know how to handle. The process of calculating M_b and M_c is similar to finding M_a

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

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

Finally, we can compute the determinant for M:

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

To calculate an $n \times n$ matrix, let's use the example matrix:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & ... & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & ... & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & ... & x_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & ... & x_{n,n} \end{bmatrix}$$

We'll be going through each element in the first row, create an $(n-1)\times(n-1)$ matrix for each element (for instance, we made a matrix M_a for element a), and calculate the determinant.

We first make the matrix $M_{1,1}$:

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

Next, we make the matrices for $M_{1,2}, M_{1,3}$ and so on.

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

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

We can express the Determinant of M as:

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

We can shorten this into a smaller formula:

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

6.2 Input Format

Below are the contents of test1.txt

```
3
1   2   1
0   3   4
3   1   4
```

This input file tells us that we need to calculate the determinant of a 4×4 matrix with the following numbers:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 3 & 4 \\ 3 & 1 & 4 \end{bmatrix}$$

6.3 Output format

For the output, you print out the determinant of the input matrix:

```
$ ./fifth test1.txt
23
$
```

6.4 Assuptions/tips

- You may assume that the inputs will be properly formatted.
- Make seperate functions for creating and freeing the matrix. It'll help you keep your code clean, organized, and easy to troubleshoot.

7 Criteria, Grading, Submission

7.1 Compiling your code

When you compile your code (in your Makefile), you must include the following flags:

```
-fsanitize=address, undefined -Wall -Werror -std=c11 -lm -g -Og
```

- fsanitize is a memory bug checker that checks for memory bugs (segmentation faults, stack/heap overflow, etc.)
- -Wall will print out all warnings
- -Werror will treat all warnings as errors, not letting you compile.
- -Og optimizes for debugging
- -std=c11 specifies that we'll be using the c11 standard.
- If you use math.h (which you'll probably need), you must include -lm, which **links** the math library to your program.
- -g -Og are useful for debugging

We will check the Makefile to ensure that these flags are included. If not, you'll receive a 0 for the part.

7.2 Using the Autograder

We provide an autograder for students to test their code automatically against test cases. You're given 50% of the test cases, so passing them does not guarantee a 100%. For this reason, you are encouraged to test your code and ensure it works as intended **without crashing**.

7.2.1 Using the autograder

You can test individual parts of the assignment with the following command:

```
python3 autograder.py --part <part-number>
```

For the part number, fill in the part you want to be checked (put 1 in if you want to test part 1 of the assignment).

You can also test all parts using:

```
python3 autograder.py --part all
```

This will test all parts.

7.3 Submitting your code

You'll be compressing your pa1 folder into a .tar file and submitting it. To do this, you do the following:

```
tar -cvzf pa1.tar pa1
```

7.3.1 Expected file structure

Below is how your submission should look:

```
pa1
|--- first
|    |--- Makefile
|    |--- first.c
|    +--- first.h (optional)
|--- second
|    |--- Makefile
|    |--- second.c
|    +--- second.h (optional)
|--- third
|    |--- Makefile
|    |--- third.c
|    +--- third.h (optional)
|--- fourth
|    |--- Makefile
|    |--- fourth.c
|    +--- fourth.h (optional)
+--- fifth
     |--- Makefile
     |--- fifth.c
     +--- fifth.h (optional)
```

7.4 checking tar file

You should ensure that you **only tar the pa1 folder**, and it should follow the file structure depicted above. To ensure that you compressed the file correctly, we've included an additional feature on the autograder to test the tar file.

```
python3 autograder.py pa1.tar
```

This will extract the tar file and grade it. If it grades your assignment as intended, you compressed your file correctly and can submit it. Compressing the tar file on the lab machine is ideal and prevents many issues.