COS 135 Individual Assignment 9

Due: Monday 04/18/22 End of the day

Write C programs for following tasks and submit your source codes (you must submit .c files without compilation errors). Sample Program inputs are highlighted in yellow.

What to submit:

- Please **submit a .zip file with four separate .c source codes**. Your programs must produce similar outputs if the same inputs are provided.

[-10 pts each] Source codes should be able to compile and execute without errors or warnings:

- **[-10 pts if any compiler errors]** Source codes should be able to compile without any errors
- Always compile your source codes with -Wall (enable all warnings) switch to find all the compiler warnings and fix them (see example below).

```
$ gcc mycode.c  -o  myprogram -Wall
```

[-10 pts each] **Comments** are required in the following locations (in each C source code):

- **[-2 pts]** At the top of the source code, comment your name and insert a short program description
- **[-4 pts]** Comment the purpose of each variable.
- **[-4 pts]** Comment major sections of your code such as inputs, functions, and outputs.

Program Design: Your program is a professional document and must be neat and easy to read. All programs should follow these specifications.

- Comments should be aligned and entered in a consistent fashion
- Blank lines should be added to aid readability
- Code within blocks should be indented
- Comments should not contain spelling mistakes
- Variable names should be meaningful
- Define functions where necessary
- Take time to design your code (program flow and functions) before start coding

(a) (30pts): matrix dot product

To multiply a matrix by another matrix we need to calculate the ***dot product*** of rows and columns. Write a C program to multiply two matrices (two-dimensional arrays).

1. First, the program should ask the user to enter the size of the matrices (**rows x columns**) in the format of: **2 x 3** (assume user only enters single digit numbers between 0 - 9).

   Sample program input:
   Enter number of rows and columns of first matrix (format: r x c): 2 x 3
   Enter number of rows and columns of second matrix (format: r x c): 3 x 4

2. Then, your program should generate **random numbers between 1 and 9 (including the numbers 1 and 9)** to fill the two matrices.

Validation:

- To multiply a matrix by another matrix, **the number of columns of the 1st matrix must equal to the number of rows of the 2nd matrix**.
- The result should have the same number of rows as the 1st matrix, and the same number of columns as the 2nd matrix.

Background info: "How to multiply matrices" - *https://www.youtube.com/watch?v=2spTnAiQg4M*

If the orders of the matrices are such that they can't be multiplied by each other, then an error message should be displayed.

Finally, multiply them and save the results in a new matrix, and display the results in the following format:

```
 Matrix A                 Matrix B                      Matrix C

5     4    .   2     4     6     9     =     30    40    66    45
8     7        5     5     9     0           51    67    111   72
9     1                                      23    41    63    81
```

Sample output #1:

```
Enter number of rows and columns of first matrix (format: r x c): 2 x 3
Enter number of rows and columns of second matrix (format: r x c): 4 x 6
```

Program output:

```
Error: The 2 x 3 matrix can't be multiplied by 4 x 6 matrix.
Reason: the number of columns of the 1st matrix must equal to the number of rows of the 2nd matrix.
```

Sample output #2:

```
Enter number of rows and columns of first matrix (format: r x c): 2 x 0
```

Program output:

```
Error: The number of rows or columns of a matrix can’t be 0.
Please try again.
```

Sample output #3:

```
Enter number of rows and columns of first matrix (format: r x c): 2 x 2
Enter number of rows and columns of second matrix (format: r x c): 2 x 3
```

Program output:

```
2  3  .  4  9  1  =  17   42   20
5  7     3  8  6     41   101  47
```

(b) (20pts): matrix dot product (application)

In a separate .c file, modify the program you developed in (a) to apply the concept into the following real-life scenario.

A local gas station sells 3 different gas types based on octane grades: **regular** (usually 87 octane), **special** (usually 89 octane), and **premium** (usually 91 or 93 gas).

regular costs $3 per gallon
special cost $4 per gallon
premium cost $5 per gallon

Based on matrix multiplication, develop a C program to calculate the total sales (as well as sales in individual days) for three days.

You may save above cost info in a 1 x 3 matrix (array) as below:

[3 4 5]

The user must enter a particular day's sales info in the format of **reg#sp#pre**
(e.g., for day 1: 56#23#12) as in the sample outputs.

Following sales (in total gallons per day) were recorded for three days.

	Day 1	Day 2	Day 3
regular	56	45	23
special	23	34	22
premium	12	9	10

The program should use the matrix multiplication and generate the result in a new matrix as below.

$$[3\ \ 4\ \ 5] \ . \ \begin{bmatrix}56 & 45 & 23\\ 23 & 34 & 22\\ 12 & 9 & 10\end{bmatrix} = [320\ \ 316\ \ 207]$$

Output the results as:

Total sales on Day 1: $ 320
Total sales on Day 2: $ 316
Total sales on Day 3: $ 207
Total sales on weekend: $ 843

Sample output #1:

```
Enter daily sales in the format: regular#special#premium

Enter total gallons of gas sales on Day 1: 56#23#12
Enter total gallons of gas sales on Day 2: 45#34#9
Enter total gallons of gas sales on Day 3: 23#22#10
```

Program output:

```
[3   4   5]   .   [56   45   23    =   [320    316    207]
                   23   34   22
                   12   9    10]

Total sales on Day 1: $ 320
Total sales on Day 2: $ 316
Total sales on Day 3: $ 207
Total sales on weekend: $ 843
```

Sample output #2:

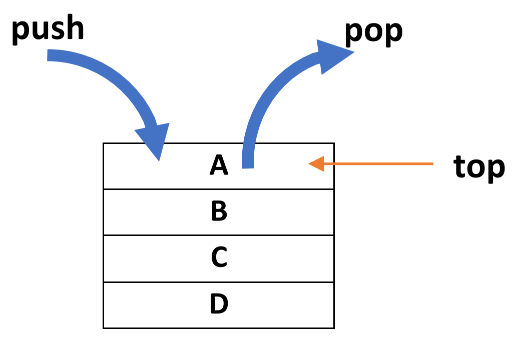
```
Enter daily sales in the format: regular#special#premium

Enter total gallons of gas sales on Day 1: 56#23#12
Enter total gallons of gas sales on Day 2: 38$72$12
```

Program output:

```
Error: please follow the correct input format
Please try again.
```

(c) (30pts): Implement a dynamic stack

A **stack** is a container of objects that are inserted and removed according to the last-in first-out (**LIFO**) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



push: Inserts a new element from the top (note: if this is the first element, it will be saving at the top, when new data is pushed this top element should go down (one level at a time) – similar to put books into a box; second and subsequent elements will be on top of the previous element). An example command: push N

pop: Removes the topmost element from the stack and output to the terminal.

Write a C program to implement this algorithm using dynamic memory, use following guidelines:

1. First, define a **dynamic char array** of size 1 (this array should grow as new data is pushed or shrink when the data is popped).
2. Implement the push function (a separate function) as explained to insert new data to the stack. A user should be able to issue 'push' command (for example, push R) to insert a new value. New value R should be added to the top and all the existing values should shift down.
3. A user should be able to issue 'pop' command to remove the topmost element from the stack and print it to the terminal. All the remaining values should shift up and delete the bottom most array element.
4. A user should be able to issue 'print' command to print all the elements in the current stack to the terminal.
5. A user should be able to issue 'quit' command to quit from the application
6. You should test your code for maximum 10 elements in the array.

Design and develop a C program using dynamic memory to fulfil above requirements.

Define functions where necessary.

Sample output #1:

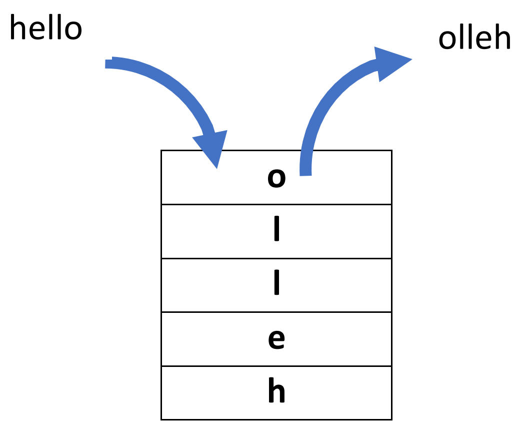
```
$./stack
Enter command (push, pop, print, or quit): push A
Enter command (push, pop, print, or quit): push B
Enter command (push, pop, print, or quit): push C
Enter command (push, pop, print, or quit): push D
Enter command (push, pop, print, or quit): push E
Enter command (push, pop, print, or quit): push F
Enter command (push, pop, print, or quit): push G
Enter command (push, pop, print, or quit): push H
Enter command (push, pop, print, or quit): push I
Enter command (push, pop, print, or quit): push J
Enter command (push, pop, print, or quit): print
J
I
H
G
F
E
D
C
B
A
Enter command (push, pop, print, or quit): push K
Error: Maximum elements in the stack
Enter command (push, pop, print, or quit): pop
Popped: J
Enter command (push, pop, print, or quit): push K
Enter command (push, pop, print, or quit): print
K
I
H
G
F
E
D
C
B
A
Enter command (push, pop, print, or quit): quit
Bye
```

Sample output #2:

```
$./stack
Enter command (push, pop, print, or quit): push N
Enter command (push, pop, print, or quit): push R
Enter command (push, pop, print, or quit): print
R
N
Enter command (push, pop, print, or quit): pop
popped: R
Enter command (push, pop, print, or quit): quit
Bye
```

(d) (20pts) Implement a dynamic stack (application of a Stack)

Write a C program to implement a simple application of a dynamic stack to reverse a word or a sentence (you may develop this program based on the program developed in part (c)). First, obtain user's input, create a dynamic char array, then push user's input to stack - letter by letter - then pop letters from the stack and print to the terminal. You must use dynamic memory allocation to create the array based on number of characters in the user's input (memory optimization).



Sample output #1:

```
$./stack_reverse
Enter a phrase:  COS135 @ UMaine
Reversed output is "eniaMU @ 531SOC"
```