Assignment-1

Data Structures and Algorithms
Pointers, Abstract Data Types, and Linked Lists

Due Date: 28 March, 2023 11:59 PM

Important Notes

- For questions 1 and 2, all functions mentioned must be defined with **exact** function name and function parameters and return type (we would run scripts to check this, so match the case of alphabets also)
- OPERATION names are case sensitive
- Doubt Document
- Plagiarism Policy. Please read this before you start the assignment

TOTAL: [350 PTS]

1 Pointer Warmup [40 PTS]

Pointers is a very important concept in C programming. Let us begin with a few warmup questions.

1.1 File Structure

- `functions.h` - Which has the function prototypes.
- `functions.c` - Which implements the functions defined in 1.2.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

1.2 Functions

- `void reverseString(char* str, int length)`: takes in a pointer to a string and its length and reverses the string in place.
- `char* compressString(char* str, int length)`: takes in a pointer to a string and its length, and returns a pointer to a string that is a compressed version of the original, where each character is followed by the number of times it appears consecutively in the original string. For example, "**aaabbc**" would be compressed to "**a3b2c1**".
  NOTE: No of times a character occurs continuously can be any number. (need not be single digit)
- `int* uniqueElements(int* arr, int length)`: takes in a pointer to an array of integers and its length, and returns a pointer to an array of integers that contains only the unique elements of the original array, without using any library functions except `malloc`.
  NOTE: The size of the array you are returning must be exactly equal to the number of unique elements.
- `int** transpose(int** matrix, int NumRow, int NumCol)`: takes in a pointer to a matrix of integers and its dimensions and returns a pointer to a matrix that represents the transpose of the original matrix.
  NOTE: The dimensions of the matrix you are returning must be exactly equal to the dimensions of the mathematical transpose.

1.3 Input and Output

The first line contains T, the number of OPERATION's that need to be done. $1 \leq T \leq 100$

The first line of an OPERATION consists of a string indicating the OPERATION name.

The next few lines contain the required input data according to the table i.e.

Operation name	**Corresponding function**
OPER1	*reverseString*
OPER2	*compressString*
OPER3	*uniqueElements*
OPER4	*transpose*

OPERATION	**INPUT FORMAT**	**CONSTRAINTS**	**OUTPUT FORMAT**
OPER1	First line contains an integer n indicating the length of the string. Second line contains the string	$1 \leq n \leq 10^4$ [**10 PTS**]	String (Reversed)
OPER2	First line contains one integer n indicating the length of the string. Second line contains the string	$1 \leq n \leq 10^4$ [**10 PTS**] $str[i]$ contains only lowercase alphabet	String (Compressed)
OPER3	First line contains an integer n indicating the length of the array. Second line contains n space separated integers.	$1 \leq n \leq 10^4$ [**10 PTS**] $1 \leq arr[i] \leq 10^4$	All unique elements in a line, space separated and should be in the order they appear first
OPER4	First line contains 2 integers R and C indicating the number of rows and the number of columns, respectively. Following R lines contains C space-separated elements	$1 \leq R, C \leq 10^2$ [**10 PTS**] $1 \leq Mat[i][j] \leq 10^6$	The transposed matrix. Each row should be printed on a new line

1.4 Example Test Cases

Input	**Output**
4	
OPER1 10 abcdefghij	jihgfedcba
OPER2 12 aacceeggiiaa	a2c2e2g2i2a2
OPER3 10 1 1 2 4 5 9 9 1 6 8	1 2 4 5 9 6 8
OPER4 3 4 1 2 3 4 2 3 4 5 3 4 5 6	1 2 3 2 3 4 3 4 5 4 5 6

2 Array Warmup [70 PTS]

Arrays are a very important concept in programming as they let us efficiently store data in a linear format. Let us begin with a few warmup questions.

2.1 File Structure

- `functions.h` - Which has the function prototypes.
- `functions.c` - Which implements the functions defined in 2.2.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

2.2 Functions

- `int* intersectionArray(int* arr1, int* arr2, int lenArr1, int lenArr2)`: takes in two pointers to arrays of integers and their lengths and returns a pointer to an array of integers that contains the intersection of the two arrays i.e., common elements in both arrays. Do not use any library function other than `malloc`.

  Note 1: The size of the array returned should be exactly the same as the number of common elements.
  Note 2: Let $o_1(i)$ and $o_2(i)$ be the number of occurences of element i in array 1 and 2 respectively. So, the number of times i occurs in the output array must be $min\{o_1(i), o_2(i)\}$
  Note 3: it is necessary that the output array is in ascending order..

- `int countCharOccurrences(const char* str, int length, char ch)`: takes in a pointer to a string, its length and a character and returns the number of occurrences of the given character in the string.
- `char findFirstNonRepeatingChar(const char* str, int length)`: takes in a pointer to a string and its length, and returns the first non-repeating character in a given string and returns the character.
- `char* findLongestCommonPrefix(char** str, int numStr, int maxLen)`: takes in an array of strings, the number of strings, and the maximum possible length. Returns the pointer to the longest common prefix among all the strings in the array.
- `int* maxMin(int* arr , int lenArr)`: takes in a pointer to an array of integers and its length and returns a pointer to an array of integers that contains the indices of the elements that are greater than all the elements to their left and smaller than all the elements to their right.

  Note: For the first element, just consider whether the elements after it are greater, and for the last element, just consider whether the elements before it are smaller.

- `char* removeSubstring(char* str, int strLength, const char* substr, int substrLength)`: takes in a string, its length, a substring, and its length. Remove all occurrences of the given substring from the given string. For example, if the input string is "abcdeabcde" and the substring is "bc", the output should be "adeade".

  Note 1: none of the prefixes of the substring match the suffixes of the substring.
  Note 2: the given substring exists in the string.

2.3 Input and Output

The first line contains T, the number of operations that need to be performed. $1 \leq T \leq 100$.

The next line consists of a string indicating the OPERATION to be executed.

The next few lines give the data needed for the operation i.e.

Operation name	**Corresponding function**
OPER1	*intersectionArray*
OPER2	*countCharOccurences*
OPER3	*findFirstNonRepeatingChar*
OPER4	*findLongestCommonPrefix*
OPER5	*maxMin*
OPER6	*removeSubstring*

OPERATION	**INPUT FORMAT**	**CONSTRAINTS**	**OUTPUT FORMAT**
OPER1	First line contains two integers n_1 and n_2, indicating lengths of the first and second array. The following two lines contains $n1$ and $n2$ integers respectively.	$1 \leq n_1, n_2 \leq 10^4$ [**10 PTS**] $1 \leq arr1[i] \leq 10^4$ $1 \leq arr2[i] \leq 10^4$	The common elements in a single line, in **ascending order**, space-separated. If there are no common elements, print **-1**.
OPER2	First line contains n, length of string Second line contains a string. The third line contains a character *ch* whose frequency is required	$1 \leq n \leq 10^4$ [**10 PTS**] $str[i]$ can be a digit or lowercase Alphabet *ch* can be a digit or lowercase Alphabet	The count of the number of occurrences of the character in the string
OPER3	First line contains n, the length of the string. Second line contains the string	$1 \leq n \leq 10^4$ [**10 PTS**] $str[i]$ is a lowercase alphabet	The first non-repeating character in the string. If there are no non-repeating characters, print **-1**.
OPER4	First line contains one integer n indicating the number of strings. The next n lines contain a integer n_i followed by a string each.	$1 \leq n \leq 10^2$ [**10 PTS**] length of each string $\leq 10^2$ all strings are of lowercase alphabet.	The longest common prefix of all the strings If there is no common prefix, print **-1**.
OPER5	First line contains one integer n indicating the length of the array. The second line contains n space separated integers.	$1 \leq n \leq 10^2$ [**10 PTS**] This question is partially graded, you get 10PTS, for solving $n \leq 10^2$ $1 \leq n \leq 10^4$ [**10 PTS**] You get an additional 10PTS if it works for $n \leq 10^4$ $1 \leq arr[i] \leq 10^6$	Indices of elements that are greater than all elements to their left and smaller than all elements to their right in a single line, space-separated The indices must be outputted in **ascending order**. If there are no such indices, print **-1**.
OPER6	First line contains two integers n, m, representing length of string and substring respectively Second line contains the string Third line contains the substring	$1 \leq m \leq n \leq 10^3$ [**10 PTS**] $str[i]$ is of lowercase alphabet *substr* is a substring of *str*	Output a string after removing the occurrences of the substring in the string If the string becomes empty, after removing all substrings, print **-1**.

2.4 Example Test Cases

Input	Output
6	
OPER5 10 1 4 8 7 5 9 17 11 40 10	 0 1 5
OPER1 6 7 1 2 3 4 5 6 7 9 1 4 8 2 3	 1 2 3 4
OPER2 10 helloworld l	 3
OPER3 9 lleetcode	 t
OPER4 3 6 flower 4 flow 6 flight	 fl
OPER6 10 2 abcdeabcde bc	 adeade

3 Circular Linked List [100 PTS]

In this question, the aim is to create a self-adjusting circular linked list. A self-adjusting list is like a regular list, except that all insertions are performed at the front. When an element is accessed by a *Find* function, it is moved to the front of the list without changing the relative order of the rest of the elements.

3.1 File Structure

- `functions.h` - Which has the function prototypes.
- `functions.c` - Which implements the functions defined in 3.3.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

3.2 Structs

Each node in the circular linked list needs to have the following attributes:

- `int` Element
- `PtrNode` NextNode

Note: You can add other attributes if it helps with the code.

3.3 Functions

- `void Insert(PtrNode Head, int num)`: creates a new node containing the integer `num` and inserts (adds it to the front) it into the circular linked list (identified by `Head`).
- `PtrNode Find( PtrNode Head, int num )`: searches the circular linked list (identified by `Head`) for the node with the integer `num` and then moves the node containing `num` to the front of the circular linked list. The function returns the pointer to the node that contains `num`.(If `num` doesn't exist in the circular linked list, then return a `NULL` pointer).
- `void Print( PtrNode Head)`: prints the elements of the circular linked list in order.

Note: After completion of each operation the linked list must necessarily be circular

3.4 Input and Output

The first line contains T, the number of operations that need to be performed. $1 \leq T \leq 1000$.

The next line consists of a string indicating the OPERATION to be executed.

The next few lines give the data needed for the operation i.e.

Operation name	**Corresponding function**
OPER1	*Insert*
OPER2	*Find*
OPER3	*Print*

OPERATION	**INPUT FORMAT**	**CONSTRAINTS**	**OUTPUT FORMAT**
OPER1	First line contains an integer n that must be inserted into the circular linked list.	$1 \leq n \leq 10^6$	
OPER2	First line contains an integer n that must be searched in the circular linked list.	$1 \leq n \leq 10^6$	
OPER3			Print all the elements in the circular linked list in order.

3.5 Example Test Cases

Input	Output
10	
OPER1 1	
OPER1 2	
OPER1 3	
OPER2 2	
OPER2 3	
OPER1 5	
OPER2 1	
OPER1 9	
OPER1 7	
OPER3	7 9 1 5 3 2

3.6 Explanation

We have:

1. `Insert 1` : Inserts 1. The current state of the circular linked list will be:

$$1$$

2. `Insert 2` : Inserts 2. The current state of the circular linked list will be:

$$2 - 1$$

3. `Insert 3` : Inserts 3. The current state of the circular linked list will be:

$$3 - 2 - 1$$

4. `Find 2` : Finds 2 and re-adjusts it. The current state of the circular linked list will be:

$$2 - 3 - 1$$

5. `Find 3` : Finds 3 and re-adjusts it. The current state of the circular linked list will be:

$$3 - 2 - 1$$

6. `Insert 5` : Inserts 5. The current state of the circular linked list will be:

$$5 - 3 - 2 - 1$$

7. `Find 1` : Finds 1 and re-adjusts it. The current state of the circular linked list will be:

$$1 - 5 - 3 - 2$$

8. `Insert 9` : Inserts 9. The current state of the circular linked list will be:

$$9 - 1 - 5 - 3 - 2$$

9. `Insert 7` : Inserts 7. The current state of the circular linked list will be:

$$7 - 9 - 1 - 5 - 3 - 2$$

10. `Print` : Prints the circular linked list.

4 Sparse Matrices as Linked Lists [140 PTS]

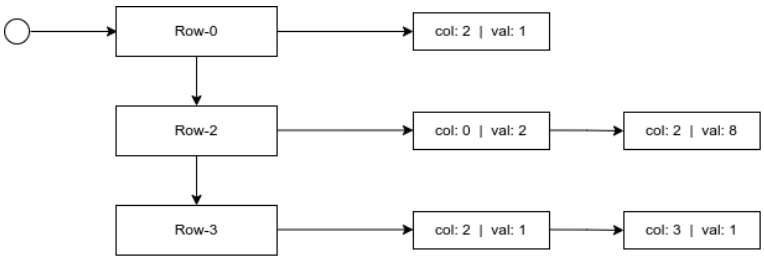
Linked lists are commonly used in the representation of sparse matrices because they can efficiently store and retrieve data without using too much memory. In a sparse matrix, most elements are zero, so storing all of these zeros in memory is inefficient. Instead, only the non-zero elements and their row and column indices are stored.

One way to use linked lists to represent a sparse matrix is to use a linked list for each non-zero row of the matrix. Each node in the lists corresponds to a non-zero element in the row and holds the column index and the value of the element.
Another linked list that stores pointers to the heads of the row lists connect these row lists together.

Example:

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 8 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Can be represented as



4.1 File Structure

- `functions.h` - Which has the required function prototypes.
- `functions.c` - Which implements the required functions.
- `main.c` - Which must use the functions and perform various operations on input and produce output as specified later.

4.2 Problems:

1. Represent a sparse matrix using linked lists.
2. Find Transpose of a matrix
3. Add two matrices.
4. Multiply 2 Matrices

4.3 Input and Output

The first line contains a string OPERATION name (ADD, MUL, or TRA) denoting the type of operation to be performed on the matrix/matrices.

Only one OPERATION would be asked per run

OPR	INPUT FORMAT	CONSTRAINTS
TRA	The second line contains the dimensions of the matrix N and M, followed by K denoting the number of non-zero elements the matrix. (i.e. $N\ M\ K$ for matrix of size $N \times M$) The next K lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix.	$1 \leq N, M \leq 10^9$ $1 \leq K \leq min(N * M, 10^3)$ [**10 PTS**] $1 \leq K \leq min(N * M, 10^6)$ [**5 PTS**] $0 \leq i < n$, $0 \leq j < m$ $-10^9 \leq val \leq 10^9$,$val \neq 0$
ADD	The second line contains the dimensions of the matrices N and M, followed by K_1, K_2 denoting the number of non-zero elements in each of the matrices. (i.e. $N\ M\ K_1\ K_2$ for matrices of size $N \times M$) (i.e. $N\ M\ K_1\ K_2$ for matrices of size $N \times M$) The next K_1 lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix. Followed by K_2 lines representing the non-zero elements of the second matrix in the same format	$1 \leq N, M \leq 10^9$ $1 \leq K_1, K_2 \leq min(N * M, 10^3)$ [**15 PTS**] $1 \leq K_1, K_2 \leq min(N * M, 10^6)$ [**35 PTS**] $0 \leq i < n$, $0 \leq j < m$ $-10^9 \leq val \leq 10^9$, $val \neq 0$
MUL	The second line contains the dimensions of the matrices N, M, L, followed by K_1, K_2, denoting the number of non-zero elements in each of the matrices. First matrix dimensions $N \times M$, Second matrix dimensions $M \times L$ (i.e. $N\ M\ L\ \ K_1\ K_2$ for matrices of size $N \times M$) The next K_1 lines contain three integers i, j, val representing the row index, column index, and value of the non-zero elements in the first matrix. Followed by K_2 lines having the non-zero elements in the second matrix in the same format	$1 \leq N, M, L \leq 10^9$ $1 \leq K_1 \leq min(N * M, 10^6)$ $1 \leq K_2 \leq min(M * L, 10^6)$ $0 \leq i < n$, $0 \leq j < m$ $-100 \leq val \leq 100$,$val \neq 0$ max(no of non zero rows of mat1, no of non zero columns of mat2) $\times \max(K_1, K_2)$, $\leq 10^3$ [**35 PTS**] max(no of non zero rows of mat1, no of non zero columns of mat2) $\times \max(K_1, K_2)$, $\leq 10^6$ [**40 PTS**]

Example:

ADD
4 3 5 4
0 2 1
2 0 1
2 2 8
3 1 1
3 2 1
1 1 1
2 0 -1
2 2 -8
3 2 4

Corresponds to:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 8 \\ 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & -8 \\ 0 & 0 & 4 \end{bmatrix}$$

Output:
The first line of the output should contain the number of non-zero elements (W) in the resultant matrix after the operation.
The next W lines should contain three spaced integers denoting i, j, val, denoting the row number(0 indexed), column number (0 indexed), and value of non-zero elements in the resultant matrix.

The expected output for the input shown above is:

```
4
0 2 1
1 1 1
3 1 1
3 2 5
```

Sample Test Cases:

Input	**Output**
TRA 4 3 4 0 2 1 2 0 1 2 2 8 3 2 1	4 0 2 1 2 0 1 2 2 8 2 3 1
ADD 4 3 5 4 0 2 1 2 0 1 2 2 8 3 1 1 3 2 1 1 1 1 2 0 -1 2 2 -8 3 2 4	4 0 2 1 1 1 1 3 1 1 3 2 5
MUL 3 2 2 3 2 1 0 1 1 1 9 2 1 4 0 1 7 1 0 2	3 1 0 18 1 1 7 2 0 8

NOTE:

- For matrix transpose, though it is easier to produce the output by simply swapping i and j from the input, we recommend you build a linked list representation of the matrix and perform transpose on it. (This may be useful in matrix multiplication)
- As shown in the examples above, you can assume that the input is provided in sorted order of (i, j). The output need not be sorted.

5 Submission Details

- Write a makefile that can compile all 4 of your solutions
- For questions 1,2, and 3 **OJ** is only used to check the correctness of your code. You are eligible for a full score(for a given function) if and only if the required functions are implemented as instructed. You will be graded on your moodle submission.
- Question 4 will be graded directly as your OJ score, but you still have to submit it in moodle
- **All solutions submitted in OJ will be considered for plagiarism**
- **Your submission to moodle will also be checked for plagiarism**
- Moodle submission should be of the given format. A *roll_number.zip*, which contains a folder named your roll number.It should also contain a makefile

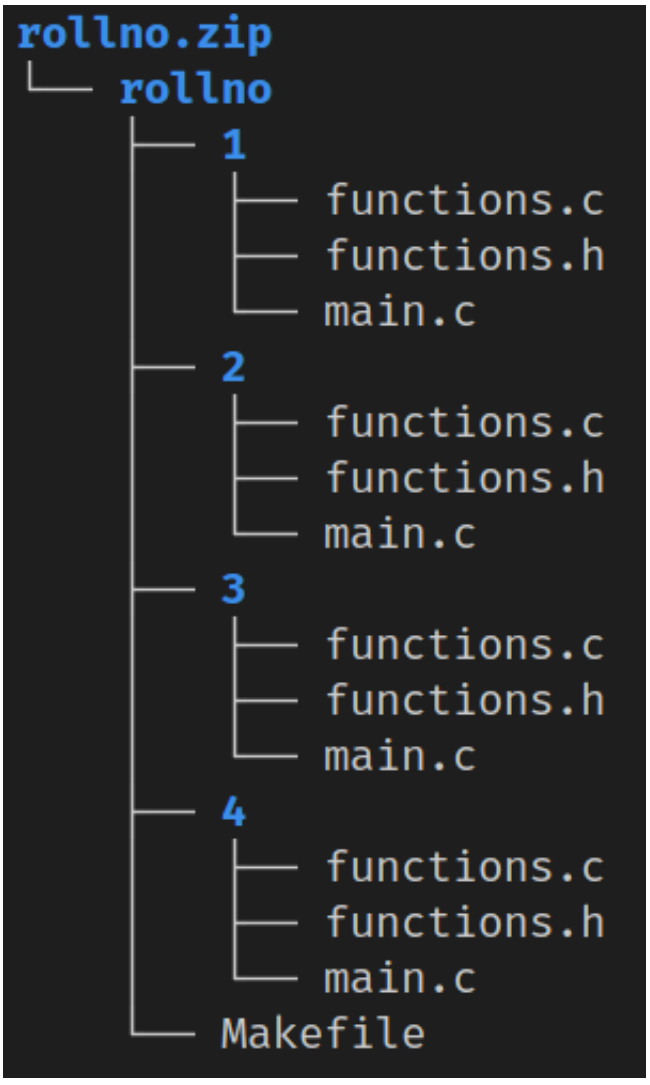


Figure 1: File format