January 2025 CSE 102

Offline 4: Pointers

Problem 1: Remove Negative Numbers Using Pointers

Problem Description: Given some numbers, remove all the negative numbers.

Constraints:

- **Do not use array of some predefined size. Use pointer arithmetic and dynamic memory allocation.** You can only use two dynamically allocated arrays; one for input array and another for output array. **No additional arrays can be used.**
- You must write a function named **`void removeNegatives(int* input, int n, int** output, int* newSize)`** where:
    - **input:** Pointer to the dynamically allocated input array of size n.
    - **n:** The number of elements in the input array.
    - **output:** A pointer to the output pointer, where you will allocate and store the address of the new array containing only non-negative numbers.
    - **newSize:** A pointer to an integer where you will store the size of the output array.
- **Array indexing notation is not allowed.** (i.e., you must use *(p + i) instead of p[i]).
- After printing the filtered array, free all allocated memory.

Sample Input(s)	**Corresponding Output(s)**
6 3 -1 0 -7 8 2	4 3 0 8 2
4 -1 -1 -1 -1	0

Problem 2: Lexicographical Sorting Using Pointer Arithmetic

Problem Description: You are given **n** words (each word is a string of **lowercase** English letters, with **maximum length 25**). Your task is to sort these words **lexicographically** (dictionary order) using the **Bubble Sort** algorithm.

Constraints:

You **must not use arrays or array indexing** in your implementation. Instead:

- Declare a **double pointer** (i.e., `char**`) to store the list of words.
- Dynamically allocate memory for:
    - The list of words (`char**`)
    - Each individual word (`char*`), using `malloc()`
- You must use **pointer arithmetic only** to access and manipulate the data:
    - Allowed: **`*ptr, *(ptr + i), *(*(ptr + i) + j)`** `etc.`
    - Not allowed: `ptr[i][j]` or `array[]` syntax anywhere, such as-
        - `words[i][j]`
        - `words[i]`
        - `word[j]`
- For input handling-
    - Use **`scanf("%s", *(ptr + i))`**
    - Do **not** use array-style input like `scanf("%s", array[i])` `or` `scanf("%s", ptr[i])`
- You must implement the **sorting logic yourself** (You have to use bubble sort). **Do not use library functions** like `qsort()` or `strcmp()`. You must write your own word comparison logic and swapping logic using pointer arithmetic. Word comparison example:
    - Comparing `"apple."` and `"application."`:
        - Compare 'a' vs 'a' → equal
        - Compare 'p' vs 'p' → equal
        - Compare 'p' vs 'p' → equal
        - Compare 'l' vs 'l' → equal

        - Compare 'e' vs 'i' → since 'e' < 'i', `"apple."` comes before `"application."`
        - Each word will have a **trailing full stop (.)**. You should stop the comparison when you encounter the full stop.
- Sorting must be done **in-place,** using the same memory (**no new list or extra copy**)
- You may write your own function(s) if needed.
- Be aware of memory leaks.

Sample Input(s)	Corresponding Output(s)
3 banana. apple. grape.	apple. banana. grape.
2 application. apple.	apple. application.

Mark Disribution

Component	Marks
Problem1	10
Problem2	10
Proper memory allocation	5
Proper memory deallocation	5
Total	**30**

Deadline: 11:55 pm, June 23, 2025

Submission Guidelines:

1. Go to a drive except C drive.
2. Create a folder according to your roll number. Ex- 2405xxx.
3. Open up the folder and create two files there. Ex- 2405xxx-1.c, 2405xxx-2.c.
4. Place all the code inside the two .c files.
5. Zip the folder and submit it in the moodle.