Coding Assignment 1: CS2233

Kindly adhere to the following instructions.

- Please write a C program corresponding to each problem. Your code should be well commented and variable names should be appropriately chosen. Also prepare a `readme` text file where you can mention instructions to run the program/how to take input etc.
- Create a folder and put all the code files and `readme` text file in it, give name to the folder as "yourRollNo", zip the folder and submit it to the google classroom portal.
- Strictly follow the input and output format for each problem.
- Any code that does not follow the input-output criteria won't be evaluated and will get **ZERO**.
- Your code will also be checked against plagiarism (both from web and peer).
- Any form of plagiarism (web/chatGPT/with peers) will be severely penalised and will result in F grade.
- The submission (strict) timeline is 11th September, Thursday, 11 PM.

1. Given the ***head*** of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list. The *first* node is considered *odd*, and the *second* node is *even*, and so on.
   The relative order inside both the even and odd groups should remain as it was in the input. Solve in $O(1)$ extra space complexity and $O(n)$ time complexity.

   (10 Marks)

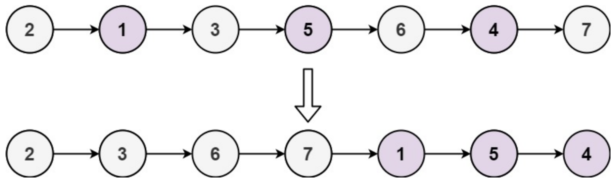
   Input Format

   - A sequence of integers separated by commas.
   - The last integer is always -1, which denotes the end of the list and should not be considered part of the input data.

   Output Format

   - A sequence of integers obtained by reordering the given input list (excluding -1) according to the required transformation.
   - The output is printed as comma-separated integers.

   Test case: 1



   Example
   Input:
   `2, 1, 3, 5, 6, 4, 7, -1`
   Output:
   `2, 3, 6, 7, 1, 5, 4`

2. Write a C program to implement doubly linked lists using only one pointer value `x->np` per item instead of the usual two `x->next` and `x->prev`. Assume that all pointer values can be interpreted as k-bit integers, and define `x->np` to be `x->np=x->next XOR x->prev`, the k-bit "exclusive-or" of `x->next` and `x->prev`. (The value `NIL` is represented by 0.) Be sure to describe what information you need to access the head of the list. Show how to implement the `SEARCH`, `INSERT`, and `DELETE` operations on such a list. Also, show how to reverse such a list in $O(1)$ time. **(20 Marks)**

   Input Format:

- First line will contain k, which indicates the number of operations to be performed.
- Next k lines will contain the details of each operation as follows:

  The input starts with an integer indicating the operation to be performed, followed by the necessary values for that operation.
  The operations are defined as follows:

  – **1:** Search for the value `x` in the list.
  – **2:** Insert the value `x` into the list.
  – **3:** Delete the value `x` from the list.
  – **4:** Reverse the list.

Output Format:
It consists of k lines corresponding to each k operations.
The output depends on the operation performed:

- **For Search:**
  – Output: `'Element is present'` if the value is found.
  – Output: `'Element is not present'` if the value is not found.
- **For Insert:**
  – Output: The list after inserting the value, with the most recently inserted element appearing first.
- **For Delete:**
  – Output: The list after deleting the specified value.
  – Output: If the deleting element is not present then print `'Element is not present'`.
- **For Reverse:**
  – Output: The list after reversing it.

Test Cases:
Input:

```
9
2 10    %Insert 10%
2 20
4       %Reverse%
1 20    %Search 20%
1 30
2 40
3 30    %Delete 30%
3 40
4
```

Output:

```
10
20 10
10 20
Element is present
Element is not present
40 10 20
Element is not present
10 20
20 10
```

3. Write a program that takes an infix arithmetic expression as input and outputs the corresponding postfix expression. You can assume that the infix statement is given in the correct format. For example, if the input is `a+b`, then the output should be `ab+`. Your algorithm's time and space complexity should be $O(n)$ and $O(n)$, respectively: **(20 Marks)**

   Input format

   - First line will contain k, which indicates the number of testcases.
   - Following k lines will contain a string of infix statement.
   - Infix string will contain

   $$[a - z] \cup [(,)] \cup [+, -, *, /, \wedge]$$

   - **Note:** Order of precedence of operator is

   $$\{\wedge\} > \{/, *\} > \{+, -\}$$

   Output format

   - Your output also should contain k lines having postfix string corresponding to infix string.

   Example:
   Input:

```
3
a*(b+c)-d
a+b*c
a^b*c+d
```

   Output:

```
abc+*d-
abc*+
ab^c*d+
```

Instructions: The input is a string of characters, and you can assume that the input given is the correct infix format. You need to check which data structure is suitable for the purpose, and for the problem you need to use the functionality of that data structure only.

4. Consider the following dataset click here. The dataset consists of a key parameter - `NHS No.`, and three value parameters, `first name, email and gender`, respectively. The number of elements in the datasets is $n = 1000$.

   Recall the static `dictionary problem` in which, after creating the dictionary, no further insertion/deletion is allowed. Create a static dictionary data structure using `perfect hashing` algorithm. The space required to store the elements should be at most cn, for some constant c.

   `Hint:` First, you need to create a family of universal hash functions from $\mathcal{H} : \mathcal{U} \mapsto \{0, 1, \ldots, m-1\}$, where $|\mathcal{U}| = 10^{10}$, and m denotes the number of slots in the hash table. You can use the dot-product hash function for the same (covered in the class). You need to create two levels of hashing using a universal hash function. For the first level of hashing pick $h_1 \in_R \mathcal{H}$ by setting $m = 5n$. If $\sum_{i=0}^{m-1} l_i^2 > 5n$, then try with another hash function h_1, where l_i denotes the number of keys mapped in the i-th slot. For each of the keys mapped in i-th slot, pick another hash function $h_{2,i} \in_R \mathcal{H}$ by setting $m = 10 l_i^2$. If there are collisions in the second level repick $h_{2,i}$ for $i \in \{0, \ldots, m-1\}$. **(30 Marks)**

   Test criteria: Once you have created the static dictionary, to evaluate the code, write a search function in which you pass the key "`search(NHS No)`", and it should return the corresponding parameters `first name, email and gender` if the key is present, otherwise it should report `key not found`.

   Input Format
   - A single NHS number (integer)

   Output Format
   - If the NHS number exists in the dictionary, output the corresponding *name*, *email*, and *gender*, separated by commas.
   - If the NHS number does not exist in the dictionary, output the message: `Key not found`.

   Example 1
   Input:
   ```
   5808001331
   ```
   Output:
   ```
   Lauryn, lborleace0@behance.net, Non-binary
   ```

   Example 2
   Input:

```
9999999999
```

Output:
```
Key not found
```

5. You are given a stack, and the task is to **sort the stack using recursion**.

   - The stack must be implemented using a **singly linked list**.
   - After building the stack, apply a **recursive method** to sort its elements in **descending order** (i.e., the largest element should be on the top of the stack).

   Input Format:

   - The first line contains an integer n, the number of elements in the stack.
   - The second line contains n space-separated integers, representing the elements of the stack from **bottom to top**.

   Output Format:

   - Print the elements of the sorted stack from **top to bottom**, separated by spaces.

   Example 1:

```
Input:
5
11 2 32 3 41

Output:
41 32 11 3 2
```

(10 Marks)

6. Design a stack data structure that supports the following four operations, each in constant time complexity, i.e., $O(1)$:

   - `push(x):` Insert an element x onto the top of the stack.
   - `pop():` Remove and return the element at the top of the stack.
   - `findMiddle():` Retrieve the middle element of the stack without removing it.
   - `deleteMiddle():` Remove the middle element from the stack.

(10 Marks)

Input Format
- The first line contains an integer Q, the number of operations.
- Each of the next Q lines contains one operation in one of the following forms:

(a) `1 x` : Push integer x onto the stack.

(b) `2` : Pop the top element from the stack.

(c) `3` : Find the middle element of the stack.

- If the stack has an odd number of elements, return the exact middle.
- If the stack has an even number of elements, return the **lower middle** (i.e., the $\frac{n}{2}$-th element from the bottom, using 1-based indexing).

(d) `4` : Delete the middle element from the stack.

Output Format
- For each operation:

- Operation `1 x`: No output.
- Operation `2`: Print the popped element, or `Stack Underflow` if empty.
- Operation `3`: Print the middle element, or `Stack Empty` if empty.
- Operation `4`: Print the deleted middle element, or `Stack Empty` if empty.

Example 1
Input:

```
8
1 1
1 2
1 3
1 4
3
4
3
2
```

Output:

```
2
2
3
4
```

Explanation: - After pushes: stack = [1, 2, 3, 4] (top = 4). - `findMiddle()` $\rightarrow$ 2 (lower middle of 4 elements). - `deleteMiddle()` deletes 2 $\rightarrow$ stack = [1, 3, 4]. - `findMiddle()` $\rightarrow$ 3. - `pop()` removes 4.

Example 2
Input:

```
2
2
3
```

Output:

```
Stack Underflow
Stack Empty
```