CHAPTER 8

Linked List

This lab task is common for everyone and you do not need to submit it to me. It is only for you to practice with different types of problems. They are important because future problems and questions will be similar and solving these problems will help you improve yourself and prepare you for future problems. I have removed backgrounds or scenarios and just laid out the bare problem. I have also included more samples as I will not be checking the codes personally, so make sure your code can handle all cases.

8.1 Traversal

Create a linked list which can insert a number and print the linked list. Each linked list node structure will have an integer member and a structure pointer to point to the next node in the list. The code will take as input **n**, which is the number of inputs and the following **n** lines will be of two types. The first type **I**, inserts values into the linked list and the second type **P**, prints the entire linked list. If the linked list does not have any node, print "The linked list is empty", without the quotes. Each number and letter of the input will be separated by spaces. The Sample Input and Output is given as follows:

8.1.1 Sample Input

2
I 4
P

Sample Output

4

8.1.2 Sample Input

5
I 4
I 2
P
I 9
P

Sample Output

4 2
4 2 9

8.1.3 Sample Input

10
P
I 3
I 7

I 4
I 22
I 13
I 1
I 17
I 100
P

Sample Output

The linked list is empty.
3 7 4 22 13 1 17 100

8.1.4 Sample Input

9
P
I 1
P
I 2
I 4
P
I 13
I 14
P

8.1.5 Sample Output

The linked list is empty.
1
1 2 4
1 2 4 13 14

8.2 A Variable List

Write a C code which can insert values into a linked list and also delete values from a linked list. The code can insert values at any position, print all the values and also delete any node through three types of commands. The first line of input will contain the number of inputs **n**, followed by **n** commands starting with **I** for inserting an element, **D** for deleting an element and **P** for printing all the elements. The insert command is followed by two values, the position after which a node needs to be inserted and the value to be inserted. The delete command is followed by the position of the node to be deleted. If there is no node at the given position (for both insert and delete), then print "There is no such node.", without the quotes. The print command is like the previous task. See the sample inputs and outputs for a better understanding.

It is to be noted that the first element of the list is denoted by *position 1.* And the end of the linked list is denoted by *position* $(number\ of\ nodes + 1)$.

8.2.1 Sample Input

```
7
I 1 3
I 1 4
P
D 1
P
D 1
P
```

8.2.2 Sample Output

```
4 3
3
The linked list is empty.
```

8.2.3 Sample Input

```
12
D 1
I 1 3
I 1 5
I 3 1
P
D 5
I 6 14
P
D 1
P
I 2 11
P
```

8.2.4 Sample Output

```
There is no such node.
5 3 1
There is no such node.
There is no such node.
5 3 1
```

3 1
3 11 1

8.3 Dynamic Linked List

Create a linked list which can perform 4 operations. Three operations are the same as the previous Task (Insert, Delete and Print). The new operation is called **Update** and the command is **U** followed by a positive integer **a**. The update operation adds the number **a** to all the key values of the linked list. Therefore, the first line of input is an integer **n** which is followed by **n** operation commands (**I**, **D**, **P** and **U**). The sample input and output is given below for your understanding.

8.3.1 Sample Input

12
I 1 3
I 2 7
P
U 1
P
I 3 4
I 3 5
U 3
P
D 5
D 2
P

8.3.2 Sample Output

3 7
4 8
7 11 8 7
There is no such node.
7 8 7

8.4 Double Linked List

Do not attempt this task until you have finished all the previous tasks. The linked lists that you have created till now, had two members in the node structures:

1. The key value which is an integer
2. The next node variable which is a structure pointer of the type "node".

In this task you have to create a linked list which can not only traverse forward but also backwards. Therefore, each structure will have another member which is also a structure pointer of the type "node" and points to the previous node. This linked list is known as a **doubly linked list**. The operations of this linked list are as follows:

1. Print the linked list from the first to last (like the previous task). The command is simply the letter **P**.
2. Print the linked list from the last element to the first element (have to traverse the list in a reverse fashion). The command is a single letter **R**.
3. Inserting nodes in any position from the beginning or the end. In order to avoid finding the size of the linked list each time, you have to implement two insert functions, one which inserts from the beginning and the other function inserts from the end. For example, for a linked list with 4 elements, inserting at *position 4* from the beginning is the same as inserting at *position 2* from the end. The commands are respectively **I** and **B** (Back) followed by the position and the key value.
4. The delete functions are also of two types: **D** which deletes in order from the beginning and **F** (Finish) which deletes from the end. The commands are followed by the position.

Hint: In order to implement these functions, a link from each node to the previous node must also be maintained. The node structure should be as follows:

```
struct node{
   int key;
   struct node *nxt;
   struct node *prv; //for link to previous node
};
```

8.4.1 Sample Input

13
I 1 2
I 2 5
P
R
B 1 3
B 2 7
P
F 1
R
I 1 9
P
D 3
R

8.4.2 Sample Output

2 5
5 2
2 5 7 3
7 5 2
9 2 5 7
7 2 9