15.5

National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	Data Structures	Course Code:	CS2001
Degree Program:	BS (CS, SE, DS)	Semester:	Fall 2021
Exam Duration:	60 Minutes	Total Marks:	20
Paper Date:	20-Oct-2021	Weight	15
Section:	ALL	Page(s):	6
Exam Type:	Midterm-I		

Student : Name: Nabeeha Mudassir Roll No. 20L-1080 Section: BS SE 3A

Instruction/Notes: Attempt all questions. Answer in the space provided. You cannot ask for rough sheets they are attached with this exam. **Answers written on rough sheet will not be marked**. Do not use pencil or red ink to answer the questions. In case of confusion or ambiguity make a reasonable assumption.

Question:

(Marks: 20)

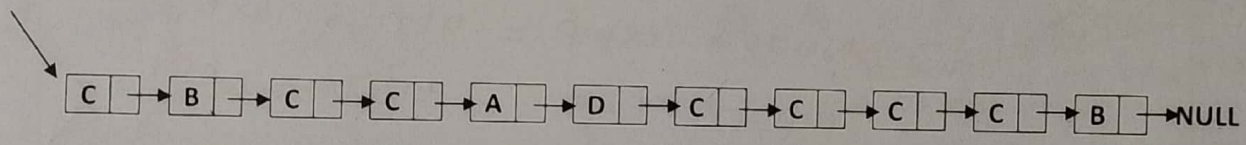
Consider a **singly linked list** class with a head pointer is already implemented for character datatype. You have to add a functionality in the class to balance out the number of consecutive occurrences of a particular character in the list.

For that you will implement a function **bool** `Equalize_Occurrences (char key, int maxcount)` of the class list, that will take a character key and maximum count for the consecutive occurrences of the key in parameters. It will then traverse the list, verify and update the consecutive occurrences of the key according to maximum count and returns true. It returns false if no occurrence of key is found.

Note: You can traverse the list only once for this task.

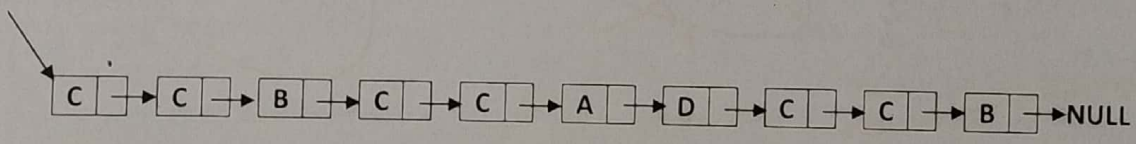
For Example, if the singly linked list **L1** contains data as follows:

Head



then after function call `L1.Equalize_Occurrences ('c', 2);` list will be updated as follows.

Head



if it occurs once, (insert another)
if it occurs twice (nothing)
if it occurs > twice. (delete)

Implement the following helper member functions.

Part (A) Insert_After, this function inserts a key value after the node to, which ptr is pointing.

```
void Insert_After (Node * ptr, char key){
    if (ptr)
    { Node * NewNode (key, 0);
      NewNode → Next = ptr → Next;
      ptr → Next = NewNode;
    }
}
```

Part (B) Delete_After, this function deletes the node after the node to which ptr is pointing. **[2]**

Assuming tail is also implemented and its next points to null

```
void Delete_After (Node * ptr){
    if ( ptr != tail && ptr != 0)
    {
        Node * temp = ptr → next;
        ptr → next = temp → next;
        delete temp;
        size--;
    }
}
```

Part (C) Equalize_Occurrences, it must use the helper functions Insert_After and Delete_After. [10]

```
bool Equalize_Occurrences (char key, int maxcount){
    Node * start = Head;
    while (start != NULL)
    {   if (start -> data == key)
        {
            if (start -> next -> data
                != key)
            {
                int i = 1;
                while (i < maxcount)
                { insert after (start)
                  i++; start = start -> nxt
                }
            }
            else
            {
                int i = 1;
                Node * temp = start -> next;
                while (temp -> data ==
                       key)
                { if (i > maxcount)
                    delete after (temp);
                  i++;
                  temp = temp -> next;
                }
                start = start -> next;
            }
        }
        else start = start -> next;
    }
}
```

// Loop works to insert maxcount - 1 times key

5.5

//

// Loop only works if >1 matches are found and deleted

Part (D) What is the time complexity Big-O of following functions, justify your answer properly?

i. Insert_After (Worst Case Big-O) [1]

Line1 O(1)
O(1)
O(1)
≈ O(1)

O(1)

only one

1

ii. Delete_After (Worst Case Big-O) [1]

O(1)
+
O(1)
+
O(1)

≈

O(1)

1

iii. Equalize_Occurrences (best Case Big-O) [1]

O(n)

in case key does not exist only external/outer condition is checked

1

iv. Equalize_Occurrences (Worst Case Big-O) [3]

.... O(n) external loop

... O(maxcount)(n) inner while loop 1

... O( )

3

O(n * maxcount) + O(n)

≈ O(n * max count)