MANJUNATH_CODING_CHALLENGE_2-01/06/2020

1. Program to remove duplicates in an unsorted linked list.

Solution:

```
class LinkedList
{
	static Node head;
	static class Node
	{
		 int data;
		Node next;
		Node(int d)
		{
			data = d;
			next = null;
		}
	}
	/* Function to remove duplicates from an unsorted linked list */
	void remove_duplicates()
	{
		Node ptr1 = null, ptr2 = null, dup = null;
		ptr1 = head;
		/* Pick elements one by one */
		while (ptr1 != null && ptr1.next != null)
		{
			ptr2 = ptr1;
			/* Compare the picked element with rest of the elements */
			while (ptr2.next != null)
			{
				/* If duplicate then delete it */
				if (ptr1.data == ptr2.next.data)
				{
					/* sequence of steps is important here */
					dup = ptr2.next;
					ptr2.next = ptr2.next.next;
					System.gc();
				}
```

```
                        else
                        {
                                ptr2 = ptr2.next;
                        }
                }
                ptr1 = ptr1.next;
        }
    }
    void printList(Node node)
    {
        while (node != null)
        {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }
    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();
        list.head = new Node(10);
        list.head.next = new Node(12);
        list.head.next.next = new Node(11);
        list.head.next.next.next = new Node(11);
        list.head.next.next.next.next = new Node(12);
        list.head.next.next.next.next.next = new Node(11);
        list.head.next.next.next.next.next.next = new Node(10);
        System.out.println("Linked List before removing duplicates : \n ");
        list.printList(head);
        list.remove_duplicates();
        System.out.println("");
        System.out.println("Linked List after removing duplicates : \n ");
        list.printList(head);
    }
}
```

****

2. Given a Doubly linked list containing N nodes, the task is to remove all the nodes from the list which contains elements whose digit sum is even.

Examples:

Input: DLL = 18 <=> 15 <=> 8 <=> 9 <=> 14

Output: 18 <=> 9 <=> 14

Explanation:

The linked list contains :

18 -> 1 + 8 = 9

15 -> 1 + 5 = 6

8 -> 8

9 -> 9

14 -> 1 + 4 = 5

Solution:

```
#include <bits/stdc++.h>
using namespace std;
// Node of the doubly linked list
struct Node
{
	int data;
	Node *prev, *next;
};
// Function to insert a node at the beginning
// of the Doubly Linked List
void push(Node** head_ref, int new_data)
{
	// Allocate the node
	Node* new_node = (Node*)malloc(sizeof(struct Node));
```

```
        // Insert the data
        new_node->data = new_data;
        // Since we are adding at the beginning,
        // prev is always NULL
        new_node->prev = NULL;
        // Link the old list off the new node
        new_node->next = (*head_ref);
        // Change the prev of head node to new node
        if ((*head_ref) != NULL)
                (*head_ref)->prev = new_node;
                // Move the head to point to the new node
                (*head_ref) = new_node;
}
// Function to find the digit sum
// for a number
int digitSum(int num)
{
        int sum = 0;
        while (num)
        {
                sum += (num % 10);
                num /= 10;
        }
        return sum;
}
```

```
// Function to delete a node
// in a Doubly Linked List.
// head_ref --> pointer to head node pointer.
// del --> pointer to node to be deleted
 void deleteNode(Node** head_ref, Node* del)
{
        // Base case
        if (*head_ref == NULL || del == NULL)
                return;
        // If the node to be deleted is head node
        if (*head_ref == del)
                *head_ref = del->next;
        // Change next only if node to be
        // deleted is not the last node
        if (del->next != NULL)
                del->next->prev = del->prev;
        // Change prev only if node to be
        // deleted is not the first node
        if (del->prev != NULL)
                del->prev->next = del->next;
        // Finally, free the memory
        // occupied by del
        free(del);
        return;
}
```

```
// Function to to remove all
// the Even Digit Sum Nodes from a
// doubly linked list
void deleteEvenDigitSumNodes(Node** head_ref)
{
	Node* ptr = *head_ref;
	Node* next;
	// Iterating through the linked list
	while (ptr != NULL)
	{
		next = ptr->next;
		// If node's data's digit sum
		// is even
		if (!(digitSum(ptr->data) & 1))
			deleteNode(head_ref, ptr);
		ptr = next;
	}
}
// Function to print nodes in a
// given doubly linked list
void printList(Node* head)
{
	while (head != NULL)
	{
		cout << head->data << " ";
```

```
            head = head->next;
        }
}
// Driver Code
int main()
{
        Node* head = NULL;
        // Create the doubly linked list
        // 18 <-> 15 <-> 8 <-> 9 <-> 14
        push(&head, 14);
        push(&head, 9);
        push(&head, 8);
        push(&head, 15);
        push(&head, 18);
        cout << "Original List: ";
        printList(head);
        deleteEvenDigitSumNodes(&head);
        cout << "\nModified List: ";
        printList(head);
}
```

****

3. You have to classify a string as “GOOD”, “BAD” or “MIXED”. A string is composed of lowercase alphabets and ‘?’. A ‘?’ is to be replaced by any of the lowercase alphabets. Now you have to classify the string on the basis of some rules. If there are more than 3 consonants together, the string is considered to be “BAD”.

If there are more than 5 vowels together, the also string is considered to be “BAD”. A string is “GOOD” if its not “BAD”. Now when question marks are involved, they can be replaced with consonants or vowels to make new strings. If all the choices lead to “GOOD” strings then the input is considered as “GOOD”, and if all the choices lead to “BAD” strings then the input is “BAD”, else the string is “MIXED?

LOGIC

1. Convert the string in **0’s and 1’s**. All consonants will be considered as 1 and vowels as 0.
2. Make 2 D array (this is dynamic programming approach) and start matching.
3. For string matching, one side will be the string (converted with 0’s and 1’s) and another side with vowels and consonants as 0 and 1.

cons & Vow/ String		W	A	Y	T	O	C	R	A	C	K
	0	1	0	1	1	0	1	1	0	1	1
0	0	0	1	0	0	1	0	0	1	0	0
1	0	1	0	1	2	0	1	2	0	1	2

1. Whenever the 0 will be matched with 0, increment previous array element by 1. Similarly when 1 will be matching with 1, increment previous array element by 1.
2. In case of 3 consecutive consonants, the current array value reaches to 3 or 5 consecutive vowels the string is said to **“BAD**”.Now here comes ? marks case

cons & Vow/ String		W	A	Y	T	**?**	C	R	A	C	K
	0	1	0	1	1	**?**	1	1	0	1	1
0	0	0	1	0	0	**1**	0	0	1	0	0
1	0	1	0	1	2	**3**	**4**	**5**	0	1	2

I**n case of question mark "?" add one to both consonant and vowel array**. Also, make a Boolean flag which will be true in case of "?" occurs. If "?" and 3 consecutive consonants or 5 consecutive vowels occurred then the string is said to be **"MIXED"**, otherwise the string is **"GOOD**".

Solution:

```
#include "string.h"

#include"iostream"

using namespace std;

//alphabet check

bool alphabetcheck(char alphabet)

{

	if (alphabet >= 'a' && alphabet <= 'z')

		return true;

	return false;

}

//vowel check

int vowelcheck(char vowel)

{

	if (vowel == 'a' || vowel == 'e' || vowel == 'i' || vowel == 'o' || vowel == 'u')

		return 0;

	return 1;

}

int main(void)

{

	//the two sequences

	//string X = "waytocrack";// ----  GOOD
```

```
//string X = "waaaaaaytocrack";//--BAD
string X = "wayt?arack";  // mixed
 //length of the sequences
int XLen = X.size();
int Arr[2][20];
memset(Arr, 0, sizeof(Arr[0][0]) * 2 * 20);
int max0 = 0;
int max1 = 0;
int index;
bool mixed_value = false;
for (size_t i = 0; i < XLen; i++)
{
	//alphabet check
	if (alphabetcheck(X[i]))
	{
		//vowel check fucntion:
		if ((vowelcheck(X[i])) == 0)
	{
		Arr[0][i+1] = Arr[0][i] + 1;
		if (Arr[0][i + 1] > max0)
		max0 = Arr[0][i + 1]; //check for max 0
		if ((mixed_value == true) && (max0 >= 5) && (Arr[0][i + 1] >= 5))
		// Arr[0][i + 1] >= 5 when current value is greater than 5
		{
			if ((i - index >= 5 || (Arr[1][index] + Arr[0][index + 5] == 7)))
```

```
                        mixed_value = false;
                }
        }
        else
        {
                Arr[1][i + 1] = Arr[1][i] + 1;
                if (Arr[1][i + 1] > max1)
                max1 = Arr[1][i + 1]; //check for max 1
                if (mixed_value == true && max1 >= 3 && Arr[1][i + 1] >= 3 )
                // Arr[0][i + 1] >= 5 when current value is greater than 3
                {
                        if ((i - index >= 3 || (Arr[0][index] + Arr[1][index + 3] == 7)) )
                                mixed_value = false;
                }
        }
        if (mixed_value == false && (max0 >= 5 || max1 >= 3))
        {
                //checking the count value GREATER than  or 3
                cout << " BAD " << endl;
                exit(0);
        }
}
else if (X[i] == '?')
{
        //increament both o count and 1 count.
```

```
                Arr[0][i + 1] = Arr[0][i] + 1;
                if (Arr[0][i + 1] > max0)
                max0 = Arr[0][i + 1]; //check for max 1
                Arr[1][i + 1] = Arr[1][i] + 1;
                if (Arr[1][i + 1] > max1)
                max1 = Arr[1][i + 1]; //check for max 1
                index = i;
                mixed_value = true;
        }
    }
    if (mixed_value & (max0 >= 5 || max1 >= 3))
        cout << " MIXED " << endl;
    else cout << " GOOD " << endl;
        return 0;
}
```

****