WEEK – 3

CODE HACKATHON

(Department-CSE)

Date: 18th August 2018 Total Marks : 60

Created by: - M.pranay ¾ B-16 - 8919345427

Nishanth ¾ B-16 – 7093197160

V.Srinivas ¾ B-16 – 8374256375

Ajit panigrahi ¾ B-12- 8249852901

Q1.) You're given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order.

Delete as few nodes as possible so that the list does not contain any value more than once.

The given head pointer may be null indicating that the list is empty.

(15 MARKS)

Input Format:

You have to complete the "Node* removeDuplicates(Node* head)" method which takes one argument - the head of the sorted linked list.

The input is handled by the code.

Example:

Initial List: 1 2 3 4 4 5

After Removing Duplicates: 1 2 3 4 5

Output Format:

Delete as few nodes as possible to ensure that no two nodes have the same data.

Adjust the next pointers to ensure that the remaining nodes form a single sorted linked list.

Then return the head of the sorted updated linked list. Do NOT print anything to stdout/console.

The output is handled by the code in the editor.

```
#include<iostream>
using namespace std;

class Node
{
public:

        int data;
        Node* next;
        Node(int n)
        {
                this->data=n;
                this->next=NULL;
        }



};

class List
{
public:

        Node* head;
        Node* tail;
        List()
        {
```

```
        this->head=NULL;
        this->tail=NULL;
    }

    void insert_node(int data)
    {
        Node* n=new Node(data);
        if(this->head==NULL)
        {
            this->head=n;
        }
        else
        {
            this->tail->next=n;
        }
        this->tail=n;
    }
    void display_list()
    {
        if(head==NULL)
            cout<<"EMPTY"<<endl;
        Node* temp=head;
        while(temp)
        {
            cout<<temp->data<<" ";
            temp=temp->next;
        }
        cout<<endl;
    }
};
```

```
Node* removeDuplicates(Node* head) {
      //Complete this function.
}



main()
{

    int n;
    int value;
    cout<<"Enter the number of nodes in the first list:";
    cin>>n;
    //Declare a new List.
    List *l=new List();
    if(n<=0)
    {
      cout<<"No nodes in the list.";
    }
    else
    {
      cout<<"Enter the nodes in the list:";
      for(int i=0;i<n;i++)
      {
            cin>>value;
            l->insert_node(value);
      }
    }
    //Display the Initial list...
    cout<<"Before:"<<endl;
    l->display_list();
```

```
    //Removing Duplicates.
    List *simple_list;
    simple_list->head=removeDuplicates(l->head);
    //Display the Simplified List.
    cout<<"After:"<<endl;
    simple_list->display_list();
}
```

Q2.)

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on.

(10 MARKS)

For a string of length N, and an offset of K, print the encoded result.

Input Format:

1. First line: N, size of string
2. Second line: S, string
3. Third line: K, caeser cipher key

Constraints:

1. 1 <= N <= 100
2. S consists of only alphabets (uppercase & lowercase)
3. -10^9 <= K <= 10^9

Sample Input:

```
4
ajit
5
```

Sample Output:

fony

Sample Input:

5

AbCdE

-5

Sample Output:

VwXyZ

Q3.)

In a Facebook tagging challenge, a group of people must tag every other person in the same group exactly once or be tagged by that person, but not both. How many tags take place in a group of N people?

(10 MARKS)

Example:

In a group of 4 people, A, B, C & D:

If A tags B & C, but D tags A, A cannot tag D. --> 2 tags by A

B cannot tag A, but can tag C & D. --> 2 tags by B

C cannot tag A & B, but can tag D. --> 1 tag by C

D already tagged A but has been tagged by others instead --> 1 tag by D

Total: 2 + 2 + 1 + 1 = 6 tags

Sample Input:

4

Sample Output:

6

Sample Input:

21

Sample Output:

210

Q4.) We are will be providing you a string known as ‘VIBGYOR’(5 MARKS)

You have to print this string in the following pattern

Output should be like first you have to print the whole string as it is without any change. After printing the string the next line should begin starting letter of the given string and ‘*’ in between the letters. When it goes to nth next line it has to print n ‘*’ between the given strings.

INPUT format: first line contains the range upto which the star has to be printed.

Sample Input 1: 4

Sample Output 1: VIBGYOR

V*I*B*G*Y*O*R

V**I**B**G**Y**O**R

V***I***B***G***Y***O***R

V****I****B****G****Y****O****R

Sample input 2: 2

Sample output 2: VIBGYOR

V*I*B*G*Y*O*R

V**I**B**G**Y**O**R

Q5).

For a given decimal number N, check if its binary representation has exactly one '1' or not.

(5 MARKS)

Sample Input:

2097152

Sample Output:

True

Sample Input:

32767

Sample Output:

False

Q6.)

You are given a function - Node* mergelists(Node* head1,Node* head2).This function takes two head pointers of two different SORTED Linked lists.

(15-MARKS)

Given these two head pointers, MANIPULATE THE LINKS between the two SORTED lists such that it forms a Merged List that is still SORTED.

Return the head pointer that points to the third SORTED list after manipulating the list via the function "mergelists".

YOU ONLY HAVE TO RETURN THE POINTER , THE OUTPUT IS ALREADY MANAGED.

COMMENT OUT THE FUNCTION "mergelists" AND COMPLETE IT ACCORDING TO THE TASK.

SAMPLE CASE:

List 1:

1->2->3->NULL

List 2:

3->4->NULL

OUTPUT LIST:

1->2->3->3->4-NULL

OUTPUT LIST STRUCTURE:

```
1->2->3-|
        |->3->4->NULL.
```

IMPORTANT NOTE:Either of the head pointer passed to the function head1 or head2 may be NULL.

*/

```
#include<iostream>
using namespace std;
```

```
class Node
{
public:

                int data;
                Node* next;
        Node(int n)
        {
                this->data=n;
                this->next=NULL;
        }


};

class List
{
public:

        Node* head;
        Node* tail;
        List()
        {
                this->head=NULL;
                this->tail=NULL;
        }

        void insert_node(int data)
        {
                Node* n=new Node(data);
                if(this->head==NULL)
```

```
                {
                        this->head=n;
                }
                else
                {
                        this->tail->next=n;
                }
                this->tail=n;
        }
        void display_list()
        {
                if(head==NULL)
                    cout<<"EMPTY"<<endl;
                Node* temp=head;
                while(temp)
                {
                        cout<<temp->data<<" ";
                        temp=temp->next;
                }
                cout<<endl;
        }
};



Node* mergelists(Node* head1,Node *head2)//Complete this
function.........
               // such that it return a pointer t the merge list.
{
}
//.....................................................................
.
```

```
main()
{

        int n1,n2;
        int value;
        cout<<"Enter the number of nodes in the first list:";
        cin>>n1;
        cout<<"Enter the number of nodes in the second list:";
        cin>>n2;
        List* l1=new List();
        List* l2=new List();
        List* l3=new List();
        //Input List 1.....................
        if(n1>0)
        {
            cout<<"Enter the first list:";
            for(int i=0;i<n1;i++)
            {
                    cin>>value;
                    l1->insert_node(value);
            }
        }


        //Input List 2.....................
        if(n2>0)
        {
            cout<<"Enter the second list:";
            for(int i=0;i<n2;i++)
            {
                    cin>>value;
```

```
                l2->insert_node(value);
            }
        }
        //.....................................

        //Displaying the Initial Lists........
        cout<<"List 1:"<<endl;
        l1->display_list();
        cout<<endl;
        cout<<"List 2:"<<endl;
        l2->display_list();
        cout<<endl;

        //Output..............................
        cout<<"Output:"<<endl;
        l3->head=mergelists(l1->head,l2->head);
        l3->display_list();
        //.....................................
}
```

----------THE END----------

ALL THE BEST ☺