Q1.For a given Linked List (LL), write programs to perform the following functions

1. Display the elements of a LL
2. Count and sum the nodes of a LL
3. Search for a key element in a LL
4. Delete an element from a LL
5. Check if a LL is sorted
6. Merge 2 LLs
7. Concatenate 2 LLs
8. Reverse the elements of a LL

CODE

#include<iostream>

using namespace std;

class Node{

    public:

        int key;

        Node\* next; // pointer to keep track of address of the next node in the linked list

};

void push(Node\*\* head\_ref, int new\_key)

{

    // we pass a reference to the head, so as to not change the position of the head

    //the function is designed in such a manner that the head remains at the first node.

    Node\* new\_node = new Node();

    new\_node->key=new\_key;

    new\_node->next = \*(head\_ref);

    \*(head\_ref)=new\_node;

}

void print(Node\* n)

{

    while(n!=NULL)

    {

        cout<<n->key<<" ";

        n=n->next;

    }

}

bool search(Node\* c, int s)

{

    Node\* current =c;

    while(current!=NULL)

    {

        if(current->key==s)

        {

            return true;

        }

        current=current->next;

    }

    return false;

}

void delete\_end(Node\* n)

{

    Node\* temp=n;

    while(temp->next->next!=NULL)

    //the while condition checks whether the two subsequent nodes have no NULL value

    {

        temp = temp->next;

    }

    //once we traverse till the second last element, change it's next pointer to NULLs

    temp->next=NULL;

}

void delete\_middle(Node\*n,int k)

{

    Node\* temp=n;

    while(temp->next->key!=k)

    {

        temp=temp->next;

    }

    temp=temp->next->next;

    // for(Node\* i=temp;temp->key!=k;i=i->next)

    // {

    //     if(i->next!=NULL) {

    //         i = i->next;

    //     }

    //     }

    //     temp->next = i->next->next;

}

void count\_sum(Node\* n)

{

    Node\* temp = n;

    int count=0, sum=0;

    while(temp!=NULL)

    {

        count++;

        sum += temp->key;

        temp=temp->next;

    }

    cout<<"The number of nodes are "<<count<<endl<<"And the sum of the nodes are "<<sum<<endl;

}

// bool isSorted(Node\* head)

// {

//     if(head==NULL)

//         return true;

//     //traversing the list till the last node

//     for(Node\* i = head;i->next==NULL;i=i->next)

//         if(i->key<=i->next->key)

//             return false;

//         //we use the above condition because the elements enter the linked list in descending order.

//     return true;

// }

bool isSorted(Node \*head)

{

    if (head == NULL)

        return true;

    // Traverse the list till last node and return

    // false if a node is smaller than or equal

    // its next.

    for (Node \*t=head; t->next != NULL; t=t->next)

       if (t->key >= t->next->key)

            return false;

    return true;

}

void concat\_ll(Node\* n1, Node\* n2)

{

    Node\* temp1 = n1;

    Node\* temp2 = n2;

    while(temp1->next!=NULL)

    {

        temp1 = temp1->next;

    }

    temp1->next = temp2;

    return;

}

void sort\_ll(Node\*\* head\_ref)

{

    Node\* current = \*head\_ref; // we are again passing a pointer ot a pointer.

    Node\* index=NULL;

    int temp;

    if(current==NULL)

        return;

    else{

        while(current!=NULL)

        {

            index=current->next;//this points to the node next to the current.

            while(index!=NULL)

            {

                if(current->key>index->key)

                {

                    temp = current->key;

                    current->key=index->key;

                    index->key=temp;

                }

                index=index->next;

            }

            current=current->next;

        }

    }

}

void merge\_ll(Node\*\* head\_ref1,Node\*\* head\_ref2)

{

    Node\* current1 = \*head\_ref1;

    Node\* current2 = \*head\_ref2;

    //sort\_ll(&current2);

    concat\_ll(current1,current2);

    sort\_ll(&current1);

}

void reverse(Node\*\* n)

    {

        // Initialize current, previous and

        // next pointers

        Node\* current = \*n;

        Node \*prev = NULL, \*next = NULL;

        while (current != NULL) {

            // Store next

            next = current->next;

            // Reverse current node's pointer

            current->next = prev;

            // Move pointers one position ahead.

            prev = current;

            current = next;

        }

        \*n = prev;

    }

int main()

{

    Node\* head = NULL;

    push(&head,6);

    push(&head,5);

    push(&head,4);

    push(&head,3);

    push(&head,2);

    push(&head,1);

    //displaying elements

    print(head);

    cout<<endl;

    //count and sum the nodes sof a linked list

    count\_sum(head);

    cout<<endl<<"Enter the element you want to search\n";

    int num;

    cin>>num;

    //deleting a code

    cout<<endl<<"from where do you want to delete the node\n 1. Beginning\n 2. End\n";

    int option;

    cin>>option;

    switch(option)

    {

        case 1:

            cout<<endl<<"Linked list after deleting the first element is \n";

            head=head->next;

            print(head);

            break;

        default:

            delete\_end(head);

            print(head);

                break;

    }

    //searching for a key element in a linked list

    if(search(head,num))

        cout<<"Element is present\n";

    else

        cout<<"Element is absent\n";

    //concatenating 2 linked lists

    Node\* head1 = NULL;

    push(&head1,12);

    push(&head1,11);

    push(&head1,10);

    push(&head1,9);

    push(&head1,8);

    push(&head1,7);

    print(head1);

    cout<<endl;

    concat\_ll(head,head1);

    cout<<endl<<"Hello world\n";

    print(head);

    //checking if linked list is sorted

    cout<<endl;

    cout<<"Hello world\n";

    Node\* head2 = NULL;

    // push(&head2,7);

    // push(&head2,6);

    // push(&head2,5);

    // push(&head2,4);

    // push(&head2,3);

    // push(&head2,2);

    push(&head2,7);

    push(&head2,6);

    push(&head2,55);

    push(&head2,4);

    push(&head2,3);

    push(&head2,22);

    print(head2);

    cout<<endl;

    if(isSorted(head2))

    cout<<"String is sorted\n";

    else

    cout<<"String is not sorted\n";

    //sorting an array

    Node\* head3 = NULL;

    push(&head3,2);

    push(&head3,1);

    push(&head3,110);

    push(&head3,19);

    push(&head3,8);

    push(&head3,77);

    print(head3);

    sort\_ll(&head3);

    cout<<endl;

    print(head3);

    //merging a linked list.

    Node\* head4 = NULL;

    push(&head4,2);

    push(&head4,1);

    push(&head4,11);

    push(&head4,19);

    push(&head4,58);

    push(&head4,79);

    cout<<endl<<"The linked list is\n";

    print(head4);

    Node\* head5 = NULL;

    push(&head5,3);

    push(&head5,13);

    push(&head5,19);

    push(&head5,5);

    push(&head5,98);

    push(&head5,69);

    cout<<endl<<"The linked list is\n";

    print(head5);

    cout<<endl<<"The merged array is\n";

    merge\_ll(&head4,&head5);

    print(head4);

    // reversing a linked list

    reverse(&head4);

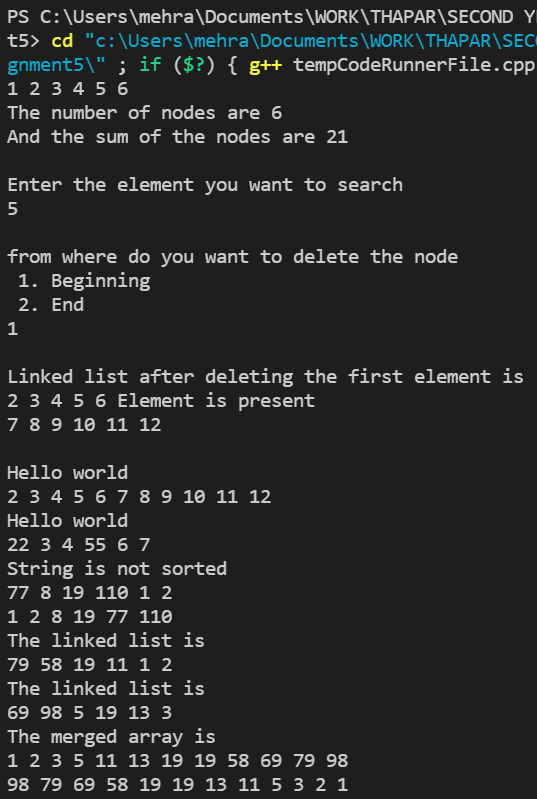
    cout<<endl;

    print(head4);

    return 0;

}

OUTPUT



Q2

Implement Double Ended Queue using linked list and implement the following functions:

* Insert or enqueue at front as well as on rear
* Remove or dequeue from front as well as on rear

Write main function to exemplify the results. Also write a main function to make the implementation a “Menu-Driven”.

CODE

#include<iostream>

using namespace std;

class Node

{

public:

Node  \*next; int data; Node(int x)

{

data = x; next  =  NULL;

}

};

Node  \*pushFront(Node  \*head,  int  x)

{

Node  \*newD  =  new  Node(x);  //  creating  a  new  node

newD->next  =  head;    //  head  goes  after  the  new  node  created return  newD;    //  return  the  new  node

}

Node  \*pushEnd(Node  \*head,  int  x)

{

Node  \*newD  =  new  Node(x);  //  new  node  created Node  \*curr  =  head;

if (head == NULL) return  newD;

while  (head->next  !=  NULL)  //  reaching  the  end  of  the  list head  =  head->next;

head->next  =  newD;  //  adding  new  node  to  end  of  exiting  list return  curr;   //  returning  head  of  the  previous  list

}

Node  \*popFront(Node  \*head)

{

if (head == NULL) return  head;

Node  \*temp  =  head;  //  fetching  the  head  of  the  list head  =  head->next;  //  head  moves  to  the  next  node cout  <<  "Popped  element  is:  "  <<  temp->data  <<  endl; delete  temp;  //  deleting  the  head  of  the  list

return  head;

}

Node \*popEnd(Node \*head)

{

if (head == NULL) // when no nodes exist return  head;

 return head;

else  if  (head->next  ==  NULL)  //  when  single  node  is  present

{

    delete  head;

    return  NULL;

}

Node  \*curr  =  head;

while  (curr->next  !=  NULL  &&  curr->next->next  !=  NULL)  //  going  to  the second  last  element  of  the  list

curr  =  curr->next;

cout  <<  "Popped  element  is:  "  <<  curr->next->data  <<  endl; delete  curr->next;  //  deleting  last  element

curr->next  =  NULL; return  head;

}

int main()

{

int ch, x; Node  \*q  =  NULL;

ch  =  -1;

while (ch != 0)

{

cout  <<  "Enter  0  to  exit"  <<  endl;

cout  <<  "Enter  1  for  push  front"  <<  endl; cout  <<  "Enter  2  for  push  back"  <<  endl; cout  <<  "Enter  3  for  pop  front"  <<  endl; cout  <<  "Enter  4  for  pop  back"  <<  endl; cin >> ch;

switch  (ch)

{

case  1:

cout  <<  "Enter  element  to  be  pushed:  "; cin  >>  x;

q  =  pushFront(q,  x); break;

case  2:

cout  <<  "Enter  element  to  be  pushed:  "; cin  >>  x;

q  =  pushEnd(q,  x); break;

case  3:

q  =  popFront(q); break;

case  4:

q  =  popEnd(q); break;

default:

break;

}

}

return  0;

}

OUTPUT

