15B17CI371 – Data Structures Lab
ODD 2024
Week 1-LAB A
Practice Lab

[CO: C270.1]

Q1. You are given an empty singly linked list. Assume that this list can contain whole numbers only. Write functions to

a. Insert 'n' number of data in the singly linked list. Insert from the head.

b. Find the total number of nodes in the linked list, and give their average.

c. Print first 'm' data from the linked list. Assume that 'm' is less than 'n'. Example:
Input:
Linked list: {1, 3, -9, 45, 2, 3, 56, 100, -67}
m=4
Output: {1, 3, -9, 45, 2}

Example:
Input:
Linked list: {1, 3, -9, 45, 2, 3, 56, 100, -67}
m=10
Output: Incorrect value of m

d. Find the middle element of the linked list and check if it's odd or even. Print an appropriate output.
Example:
Input:
Linked list: {1, 3, -9, 45, 2, 3, 56, 100, -67}
Output: 3 is odd

e. Find the 'l' number from the end of the list.
Example:
Input:
Linked list: {1, 3, -9, 45, 2, 3, 56, 100, -67}
l=3
Output: {56, 100, -67}

f. Find if a given number exists in the list. If it does, write function to delete it. Example:
Input:
Linked list: {1, 3,-9, 45, 2, 3, 56, 100, -67}
Number to be found: 45
Output: 45 exists in the original list
Final list: {1, 3, -9, 2, 3, 56, 100, -67}
If the value exists multiple times, delete only the first instance.

g. Interchange a pair of values with another given pair in the linked list. Example:
Input:
Linked list: {1, 3,-9, 45, 2, 3, 56, 100, -67}

Pairs to be exchanged: {1,3} with {56,100}
Output: {56, 100, -9, 45, 2, 3, 1, 3, -67}
Hint: first check if the pair exists and then apply interchange function. If multiple or duplicate pairs are found, consider the first instance of the pair.

h. Check whether a given sub-list exists in the given linked list. If it exists, give its position (i.e., the staring position of the sub-list in the master linked list). Example:
Input:
Linked list: {1, 3,-9, 45, 2, 3, 56, 100, -67}
Sub-list to be: {3, -9, 45}
Output: Exists at position 2.
{1, 3,-9, 45, 2, 3, 56, 100, -67}
Assumption: consider only the first occurrence of the sub-list.

i. Reverse a sub-list in the given linked list.
Example:
Input:
Linked list: {1, 3,-9, 45, 2, 3, 56, 100, -67}
Sub-list to be reversed: {3, -9, 45}
Output: {1, 45,-9, 3, 2, 3, 56, 100, -67}

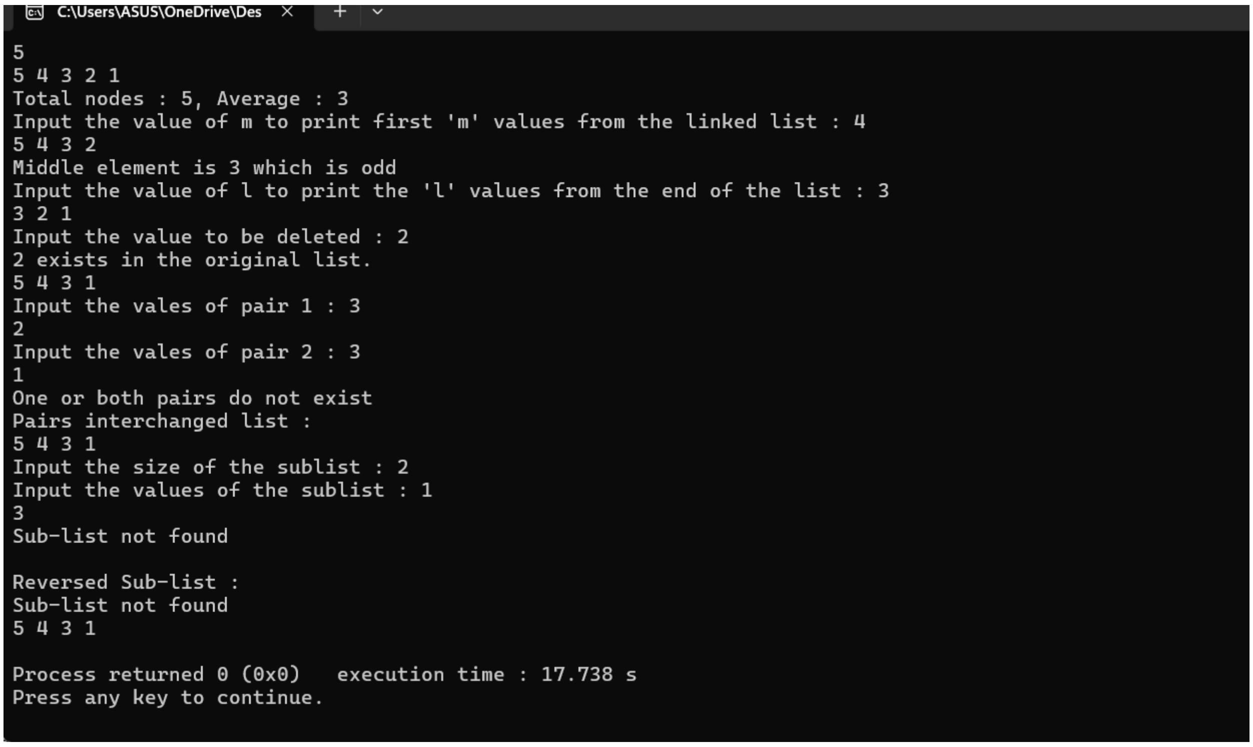
Assume that the user inputs the sub-list is found in the master linked list.

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

struct Node {
   int data;
   Node* next;
   Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
   Node* head;

   Node* findValue(int value) const {
      Node* temp = head;
      while (temp) {
         if (temp->data == value) return temp;
         temp = temp->next;
      }
      return nullptr;
   }

public:
   LinkedList() : head(nullptr) {}

   void insertHead(int data) {
      Node* newNode = new Node(data);
      newNode->next = head;
      head = newNode;
   }

   pair<int, double> totalAndAverage() const {
      int count = 0;
      double sum = 0;
```

```
        Node* temp = head;
        while (temp) {
            sum += temp->data;
            count++;
            temp = temp->next;
        }
        double average = sum / (count * 1.0);
        return {count, average};
    }

    void printFirstM(int m) const {
        if (m <= 0) {
            cout << "Incorrect value of m" << endl;
            return;
        }
        Node* temp = head;
        for (int i = 0; i < m; i++) {
            if (temp) {
                cout << temp->data << " ";
                temp = temp->next;
            } else {
                cout << "Incorrect value of m" << endl;
                return;
            }
        }
        cout << endl;
    }

    void middleElement() const {
        if (!head) return;
        Node* slow = head;
        Node* fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        cout << "Middle element is " << slow->data << " which is " << (slow->data % 2 == 0 ? "even" :
"odd") << endl;
    }

    void findLFromEnd(int l) const {
        if (l <= 0) {
            cout << "Incorrect value of l" << endl;
            return;
        }
        Node* fast = head;
        Node* slow = head;
        for (int i = 0; i < l; ++i) {
            if (fast)
                fast = fast->next;
            else {
                cout << "Incorrect value of l" << endl;
                return;
            }
        }
        while (fast) {
            slow = slow->next;
            fast = fast->next;
        }
        cout << slow->data << " ";
        while (slow->next) {
            slow = slow->next;
            cout << slow->data << " ";
        }
        cout << endl;
    }

    void deleteValue(int value) {
        Node* temp = head;
        Node* prev = nullptr;
        while (temp) {
            if (temp->data == value) {
                if (prev)
                    prev->next = temp->next;
                else
                    head = temp->next;
                delete temp;
                cout << value << " exists in the original list." << endl;
                return;
            }
            prev = temp;
            temp = temp->next;
        }
        cout << value << " does not exist in the list." << endl;
    }

    void interchangePairs(int val1, int val2, int val3, int val4) {
        Node* temp = head;
        Node* temp1 = nullptr;
        Node* temp2 = nullptr;
        Node* temp3 = nullptr;
        Node* temp4 = nullptr;
        while (temp) {
            if (temp->data == val1) {
                if ((temp->next)->data == val2) {
                    temp1 = temp;
                    temp2 = temp->next;
                }
            }
            if (temp->data == val3) {
                if ((temp->next)->data == val4) {
                    temp3 = temp;
                    temp4 = temp->next;
                }
            }
            if (temp1 && temp2 && temp3 && temp4) {
                swap(temp1->data, temp3->data);
                swap(temp2->data, temp4->data);
                return;
            }
            temp = temp->next;
        }
        cout << "One or both pairs do not exist" << endl;
    }
```

```
int findSubList(const vector<int>& subList) const {
   if (subList.empty()) return -1;
   Node* temp = head;
   int index = 0;
   while (temp) {
      Node* subTemp = temp;
      bool found = true;
      for (int val : subList) {
         if (!subTemp || subTemp->data != val) {
            found = false;
            break;
         }
         subTemp = subTemp->next;
      }
      if (found) return index;
      temp = temp->next;
      index++;
   }
   return -1;
}

void reverseSubList(const vector<int>& subList) {
   int pos = findSubList(subList);
   if (pos == -1) {
      cout << "Sub-list not found" << endl;
      return;
   }
   Node* startPrev = nullptr;
   Node* start = head;
   for (int i = 0; i < pos; i++) {
      startPrev = start;
      start = start->next;
   }
   Node* end = start;
   for (size_t i = 0; i < subList.size(); i++) end = end->next;
   Node* prev = end;
   Node* curr = start;
   while (curr != end) {
      Node* next = curr->next;
      curr->next = prev;
      prev = curr;
      curr = next;
   }
   if (startPrev)
      startPrev->next = prev;
   else
      head = prev;
}

void printList() const {
   Node* temp = head;
   while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
```

```
    }
};

int main() {
    int l, n, m, val, p1, p2, q1, q2;
    LinkedList list;
    cout << "Enter the number of values to be entered in the linked list : ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> val;
        list.insertHead(val);
    }
    list.printList();
    auto [total, average] = list.totalAndAverage();
    cout << "Total nodes : " << total << ", Average : " << average << endl;
    cout << "Input the value of m to print first 'm' values from the linked list : ";
    cin >> m;
    list.printFirstM(m);
    list.middleElement();
    cout << "Input the value of l to print the 'l' values from the end of the list : ";
    cin >> l;
    list.findLFromEnd(l);
    cout << "Input the value to be deleted : ";
    cin >> val;
    list.deleteValue(val);
    list.printList();
    cout << "Input the vales of pair 1 : ";
    cin >> p1 >> p2;
    cout << "Input the vales of pair 2 : ";
    cin >> q1 >> q2;
    list.interchangePairs(p1, p2, q1, q2);
    cout << "Pairs interchanged list : \n";
    list.printList();
    vector<int> subList;
    cout << "Input the size of the sublist : ";
    cin >> n;
    cout << "Input the values of the sublist : ";
    for (int i = 0; i < n; i++) {
        cin >> val;
        subList.push_back(val);
    }
    int pos = list.findSubList(subList);
    if (pos != -1)
        cout << "Sub-list exists at position " << pos << endl;
    else
        cout << "Sub-list not found" << endl;
    cout << "\nReversed Sub-list : " << endl;
    list.reverseSubList(subList);
    list.printList();
    return 0;
}
```



```
5
5 4 3 2 1
Total nodes : 5, Average : 3
Input the value of m to print first 'm' values from the linked list : 4
5 4 3 2
Middle element is 3 which is odd
Input the value of l to print the 'l' values from the end of the list : 3
3 2 1
Input the value to be deleted : 2
2 exists in the original list.
5 4 3 1
Input the vales of pair 1 : 3
2
Input the vales of pair 2 : 3
1
One or both pairs do not exist
Pairs interchanged list :
5 4 3 1
Input the size of the sublist : 2
Input the values of the sublist : 1
3
Sub-list not found

Reversed Sub-list :
Sub-list not found
5 4 3 1

Process returned 0 (0x0)   execution time : 17.738 s
Press any key to continue.
```

Q2. Assume that you have a linked list that can contain strings, i.e., each node can contain a string. Write a function to:

a. Print all the nodes in the linked list

b. Print all the strings (node values) that start with a particular alphabet.

c. Find if a given string exists in the linked list or not. Give appropriate output message.

d. Find the string with maximum length.

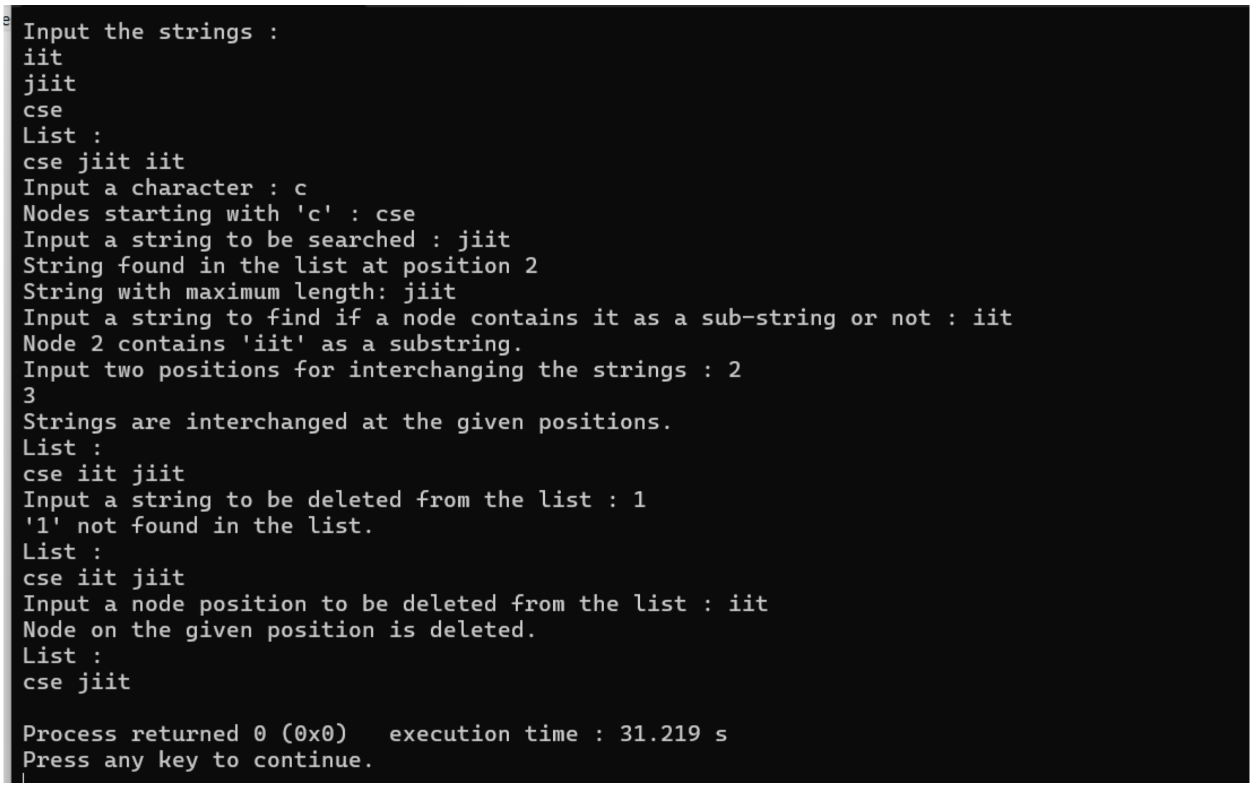
e. Find if a node contains "xyz" as a sub-string or not. Give appropriate output message.

f. Interchange the strings given in the positions p1, p2. These positions are user input. Check conditions that both p1 and p2 position exist in the given linked list, eg: suppose that your linked list consists of 4 strings only, and if user given p1=7, p2 = 10, then error message must be generated.

g. Delete a given node (either by value or by position).

```
#include <iostream>
#include <string>

using namespace std;

class Node {
public:
  string data;
  Node* next;
  Node(string val) : data(val), next(nullptr) {}
};

class LinkedList {
  Node* head;
public:
  LinkedList() : head(nullptr) {}

  void addNode(string val) {
    Node* newNode = new Node(val);
    newNode->next = head;
    head = newNode;
}

void printAllNodes() {
    cout << "List :\n";
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void printNodesStartingWith(char ch) {
    Node* temp = head;
    cout << "Nodes starting with '" << ch << "' : ";
    while (temp != nullptr) {
        if (temp->data[0] == ch) cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void findString(string s) {
    Node* temp = head;
    int c = 0;
    while (temp != nullptr) {
        if (temp->data == s) {
            cout << "String found in the list at position " << c + 1 << endl;
            return;
        }
        temp = temp->next;
        c++;
    }
    cout << "String not found in the list." << endl;
}

string findMaxLengthString() {
    Node* temp = head;
    if (!temp)
        return "";
    string max = temp->data;
    while (temp != nullptr) {
        if (temp->data.length() > max.length()) max = temp->data;
        temp = temp->next;
    }
    return max;
}

void containsSubstring(string s) {
    Node* temp = head;
    int c = 0;
    while (temp != nullptr) {
        if (temp->data.find(s) != string::npos) {
            cout << "Node " << c + 1 << " contains '" << s << "' as a substring." << endl;
            return;
        }
        temp = temp->next;
        c++;
    }
    cout << "'" << s << "' not found as a substring in any node." << endl;
}

void interchangeNodes(int p1, int p2) {
    if (p1 == p2) return;
    Node* temp = head, *temp1 = nullptr, *temp2 = nullptr;
    int c = 1;
    while (temp && !(temp1 && temp2)) {
        if (c == p1) temp1 = temp;
        if (c == p2) temp2 = temp;
        c++;
        temp = temp->next;
    }
    if (temp1 && temp2) {
        string t = temp1->data;
        temp1->data = temp2->data;
        temp2->data = t;
        cout << "Strings are interchanged at the given positions." << endl;
    } else {
        cout << "There is no node in the given position." << endl;
    }
}

void deleteNodeByValue(string val) {
    Node* temp = head;
    Node* prev = nullptr;
    if (temp->data == val) {
        head = temp->next;
        delete temp;
        return;
    }
    while (temp && temp->data != val) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "'" << val << "' not found in the list." << endl;
        return;
    }
    prev->next = temp->next;
    delete temp;
    cout << "'" << val << "' deleted from the list." << endl;
}

void deleteNodeByPosition(int p) {
    if (head == nullptr) return;
    Node* temp = head;
    if (p == 1) {
        head = temp->next;
        delete temp;
        cout << "Node on the given position is deleted." << endl;
        return;
    }
```

```
        int c = 1;
        while (temp && c < p - 1) {
            temp = temp->next;
            c++;
        }
        if (temp == nullptr || temp->next == nullptr) {
            cout << "Given position not found in the linked list." << endl;
            return;
        }
        Node* next = temp->next->next;
        delete temp->next;
        temp->next = next;
        cout << "Node on the given position is deleted." << endl;
    }
};

int main() {
    LinkedList list;
    string s;
    int n, p, p1, p2;
    char ch;

    cout << "Input the number of string to be entered in the list : ";
    cin >> n;
    cout << "Input the strings :\n";
    for (int i = 0; i < n; i++) {
        cin >> s;
        list.addNode(s);
    }
    list.printAllNodes();
    cout << "Input a character : ";
    cin >> ch;
    list.printNodesStartingWith(ch);
    cout << "Input a string to be searched : ";
    cin >> s;
    list.findString(s);
    cout << "String with maximum length: " << list.findMaxLengthString() << endl;
    cout << "Input a string to find if a node contains it as a sub-string or not : ";
    cin >> s;
    list.containsSubstring(s);
    cout << "Input two positions for interchanging the strings : ";
    cin >> p1 >> p2;
    list.interchangeNodes(p1, p2);
    list.printAllNodes();
    cout << "Input a string to be deleted from the list : ";
    cin >> s;
    list.deleteNodeByValue(s);
    list.printAllNodes();
    cout << "Input a node position to be deleted from the list : ";
    cin >> p;
    list.deleteNodeByPosition(p);
    list.printAllNodes();
}
```



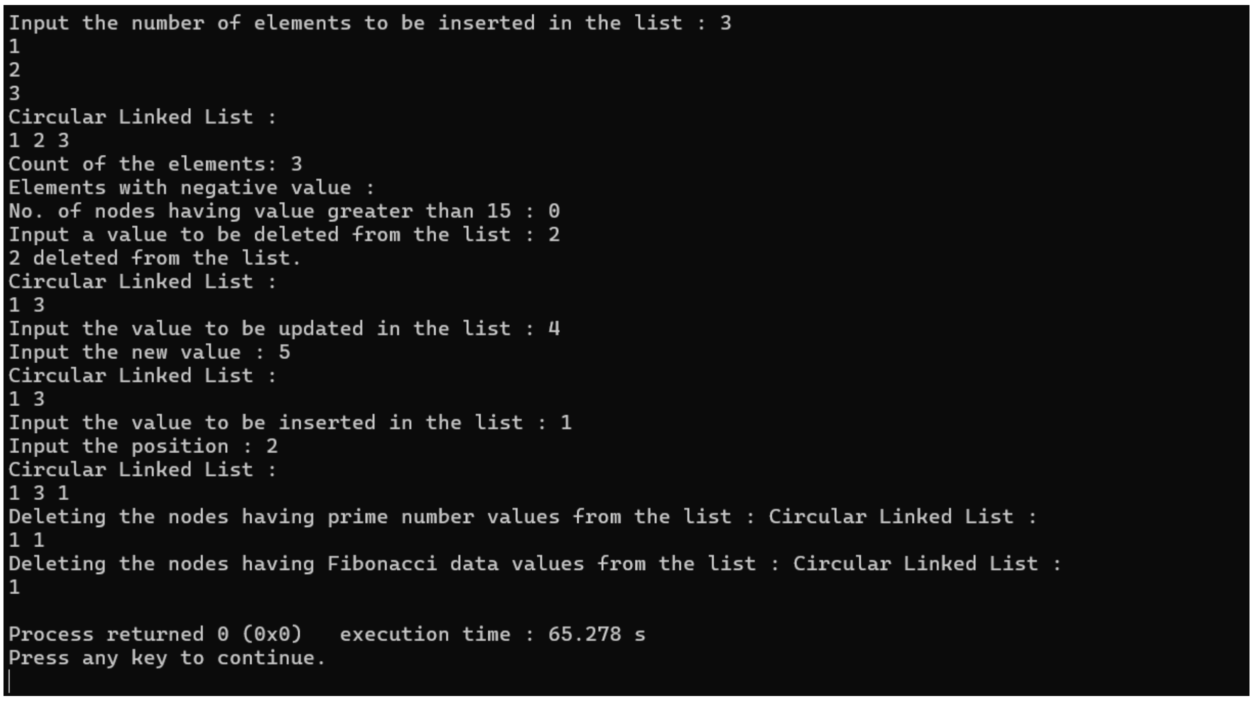
```
Input the strings :
iit
jiit
cse
List :
cse jiit iit
Input a character : c
Nodes starting with 'c' : cse
Input a string to be searched : jiit
String found in the list at position 2
String with maximum length: jiit
Input a string to find if a node contains it as a sub-string or not : iit
Node 2 contains 'iit' as a substring.
Input two positions for interchanging the strings : 2
3
Strings are interchanged at the given positions.
List :
cse iit jiit
Input a string to be deleted from the list : 1
'1' not found in the list.
List :
cse iit jiit
Input a node position to be deleted from the list : iit
Node on the given position is deleted.
List :
cse jiit

Process returned 0 (0x0)   execution time : 31.219 s
Press any key to continue.
```

Q3. Implement a circular linked list that can contain integer elements. Add functions to:
a. Insert elements.
b. Print elements
c. Count the number of elements
d. Find if any element has a negative value.
e. Find the number of nodes having a value greater than 15.
f. Delete a particular element from the list.
g. Update the value of a particular element.
h. Insert a value at a given position.
i. Delete all nodes that have a prime number as their value.
j. Remove all the nodes from the list which contains Fibonacci data values.

```
#include <iostream>
#include <cmath>
using namespace std;
struct Node
{
   int data;
```

```
    Node* next;
};
class CircularLinkedList
{
    Node* head;
    bool isPrime(int num)
    {
        if(num<=1)
            return false;
        for(int i=2;i<=sqrt(num);i++)
            if(num%i==0)
                return false;
        return true;
    }
    bool isFibonacci(int num)
    {
        int  a=0,b=1;
        if(num==a||num==b) return true;
        int c=a+b;
        while(c<=num)
        {
            if(c==num) return true;
            a=b;
            b=c;
            c=a+b;
        }
        return false;
    }
public:
    CircularLinkedList():head(nullptr){}
    void insert(int value)
    {
        Node* newNode=new Node{value,nullptr};
        if(!head)
        {
            head=newNode;
            head->next=head;
        }
        else
        {
            Node* temp=head;
            while(temp->next!=head) temp=temp->next;
            temp->next=newNode;
            newNode->next=head;
        }
    }
    void print()
    {
        cout<<"Circular Linked List :"<<endl;
        if(!head)
            return;
        Node* temp=head;
        do
        {
            cout<<temp->data<<" ";
            temp=temp->next;
        }
        while(temp!=head);
        cout<<endl;
    }
    int count()
    {
        if(!head)
            return 0;
        int cnt=0;
        Node* temp=head;
        do
        {
            cnt++;
            temp=temp->next;
        }
        while(temp!=head);
        return cnt;
    }
    void Negative()
    {
        if(!head)
            return;
        Node* temp=head;
        do
        {
            if(temp->data<0)
                cout<<temp->data<<" ";
            temp=temp->next;
        }
        while(temp!=head);
    }
    int countGreaterThan15()
    {
        if(!head)
            return 0;
        int cnt=0;
        Node* temp=head;
        do
        {
            if(temp->data>15) cnt++;
            temp=temp->next;
        }
        while(temp!=head);
        return cnt;
    }
    void deleteValue(int value)
    {
        if(!head)
```

```
        return;
    Node* temp=head;
    Node* prev=nullptr;
    do
    {
        if(temp->data==value)
        {
            if(prev)
                prev->next=temp->next;
            else
            {
                Node* tail=head;
                while(tail->next!=head)
                    tail=tail->next;
                head=temp->next;
                tail->next=head;
            }
            Node* toDelete=temp;
            temp=temp->next;
            delete toDelete;
            if(temp==head) break;
            cout<<value<<" deleted from the list."<<endl;
            return;
        }
        else
        {
            prev=temp;
            temp=temp->next;
        }
    }
    while(temp!=head);
    cout<<value<<" not found in the list."<<endl;
}
void updateValue(int oldValue,int newValue)
{
    if(!head)
        return;
    Node* temp=head;
    do
    {
        if(temp->data==oldValue)
        {
            temp->data=newValue;
            cout<<"Value updated."<<endl;
        }
        temp=temp->next;
    }
    while(temp!=head);
}

void insertAtPosition(int value,int position)
{
    Node* newNode=new Node{value,nullptr};
    if(position==0)
    {
        if(!head)
        {
            head=newNode;
            head->next=head;
        }
        else
        {
            Node* tail=head;
            while(tail->next!=head) tail=tail->next;
            newNode->next=head;
            head=newNode;
            tail->next=head;
        }
    }
    else
    {
        Node* temp=head;
        for(int i=0;i<position-1&&temp->next!=head;i++)
            temp=temp->next;
        newNode->next=temp->next;
        temp->next=newNode;
    }
}

void deletePrimes() {
    if(!head) return;
    Node* temp=head;
    Node* prev=nullptr;
    do
    {
        if(isPrime(temp->data))
        {
            if(prev)
                prev->next=temp->next;
            else
            {
                Node* tail=head;
                while(tail->next!=head) tail=tail->next;
                head=temp->next;
                tail->next=head;
            }
            Node* toDelete=temp;
            temp=temp->next;
            delete toDelete;
            if(temp==head)
                break;
        }
        else
        {
                prev=temp;
                temp=temp->next;
            }
        }
        while(temp!=head);
    }

    void deleteFibonacci()
    {
        if(!head)
            return;
        Node* temp=head;
        Node* prev=nullptr;
        do
        {
            if(isFibonacci(temp->data))
            {
                if(prev) prev->next=temp->next;
                else
                {
                    Node* tail=head;
                    while(tail->next!=head) tail=tail->next;
                    head=temp->next;
                    tail->next=head;
                }
                Node* toDelete=temp;
                temp=temp->next;
                delete toDelete;
                if(temp==head) break;
            }
            else
            {
                prev=temp;
                temp=temp->next;
            }
        }
        while(temp!=head);
    }
};
int main()
{
    int val,n,oldval,newval,p;
    CircularLinkedList cll;
    cout<<"Input the number of elements to be inserted in the list : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>val;
        cll.insert(val);
    }
    cll.print();
    cout<<"Count of the elements: "<<cll.count()<<endl;
    cout<<"Elements with negative value : ";
    cll.Negative();
    cout<<"\nNo. of nodes having value greater than 15 : "<<cll.countGreaterThan15()<<endl;
    cout<<"Input a value to be deleted from the list : ";
    cin>>val;
    cll.deleteValue(val);
    cll.print();
    cout<<"Input the value to be updated in the list : ";
    cin>>oldval;
    cout<<"Input the new value : ";
    cin>>newval;
    cll.updateValue(oldval,newval);
    cll.print();
    cout<<"Input the value to be inserted in the list : ";
    cin>>val;
    cout<<"Input the position : ";
    cin>>p;
    cll.insertAtPosition(val,p);
    cll.print();
    cout<<"Deleting the nodes having prime number values from the list : ";
    cll.deletePrimes();
    cll.print();
    cout<<"Deleting the nodes having Fibonacci data values from the list : ";
    cll.deleteFibonacci();
    cll.print();
    return 0;
}
```



```
Input the number of elements to be inserted in the list : 3
1
2
3
Circular Linked List :
1 2 3
Count of the elements: 3
Elements with negative value :
No. of nodes having value greater than 15 : 0
Input a value to be deleted from the list : 2
2 deleted from the list.
Circular Linked List :
1 3
Input the value to be updated in the list : 4
Input the new value : 5
Circular Linked List :
1 3
Input the value to be inserted in the list : 1
Input the position : 2
Circular Linked List :
1 3 1
Deleting the nodes having prime number values from the list : Circular Linked List :
1 1
Deleting the nodes having Fibonacci data values from the list : Circular Linked List :
1

Process returned 0 (0x0)   execution time : 65.278 s
Press any key to continue.
```

Q4. Create an empty doubly linked list to store integers. Perform the following by writing appropriate functions to:

a. Insert and print elements of the list.

b. Traverse all nodes and check if the value is divisible by a number 'm'.

c. Delete all the nodes from the list that are greater than the given value 'x'.

d. Find the number of elements between two duplicate values.

Example:

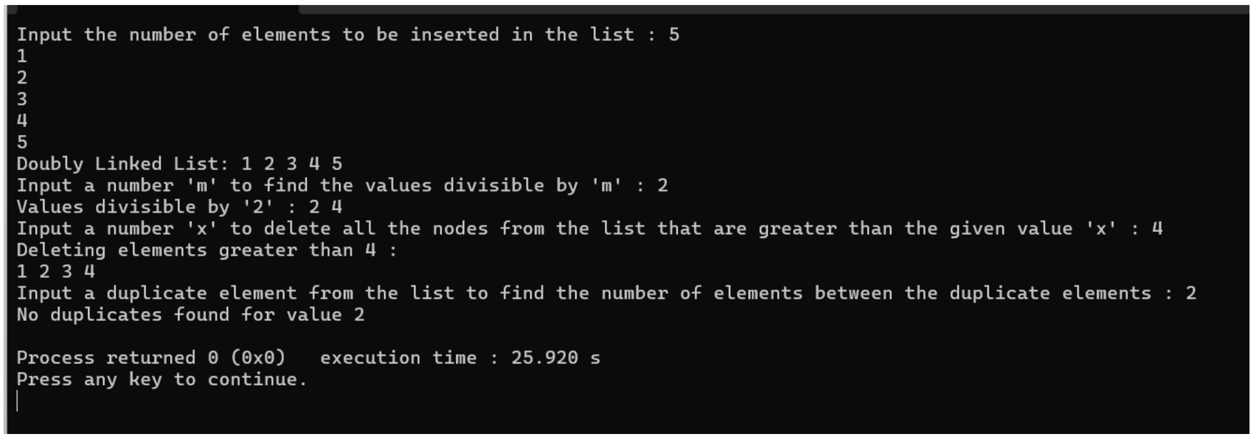
Input:

Doubly Linked list: {1, 3, -9, 45, 2, -56, 3, 56, 100, -67, 3, 3}
Duplicate element: 3
Output: No. of elements between a pair of ‘3’ = 4.
Assumption: You are considering only the first instance of duplicity, i.e., between {3, -9, 45, 2, -56, 3} and not for instances like {3, 56, 100, -67, 3} nor for {3, 3} or any others.

```
#include <iostream>
using namespace std;
struct Node
{
   int data;
   Node* prev;
   Node* next;
   Node(int val):data(val),prev(nullptr),next(nullptr){}
```

```
};
class DoublyLinkedList
{
   Node* head;
   Node* tail;
public:
   DoublyLinkedList():head(nullptr),tail(nullptr){}
   void insert(int val)
   {
      Node* newNode=new Node(val);
      if(tail==nullptr)
         head=tail=newNode;
      else
      {
         tail->next=newNode;
         newNode->prev=tail;
         tail=newNode;
      }
   }
   void print()
   {
      Node* temp=head;
      while(temp!=nullptr)
      {
         cout<<temp->data<<" ";
         temp=temp->next;
      }
      cout<<endl;
   }
   void checkDivisibility(int m)
   {
      Node* temp=head;
      while(temp!=nullptr)
      {
         if(temp->data%m==0)
            cout<<temp->data<<" ";
         temp=temp->next;
      }
      cout<<endl;
   }
   void deleteGreater(int x)
   {
      Node* temp=head;
      while(temp!=nullptr)
      {
         if(temp->data>x)
         {
            Node* toDelete=temp;
            if(temp->prev)
               temp->prev->next=temp->next;
            else
               head=temp->next;
            if(temp->next)
               temp->next->prev=temp->prev;
```

```
                else
                    tail=temp->prev;
                temp=temp->next;
                delete toDelete;
            }
            else
                temp=temp->next;
        }
    }
    int elementsBetweenDuplicates(int val)
    {
        Node* first=nullptr;
        Node* second=nullptr;
        Node* temp=head;
        while(temp!=nullptr)
        {
            if(temp->data==val)
            {
                if(first==nullptr)
                    first=temp;
                else
                {
                    second=temp;
                    break;
                }
            }
            temp=temp->next;
        }
        if(first==nullptr||second==nullptr)
            return -1;
        int count=0;
        temp=first->next;
        while(temp!=second)
        {
            count++;
            temp=temp->next;
        }
        return count;
    }
    ~DoublyLinkedList()
    {
        Node* temp=head;
        while(temp!=nullptr)
        {
            Node* next=temp->next;
            delete temp;
            temp=next;
        }
    }
};
int main()
{
    DoublyLinkedList dll;
    int n,val,m,x,duplicate;
    cout<<"Input the number of elements to be inserted in the list : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>val;
        dll.insert(val);
    }
    cout<<"Doubly Linked List: ";
    dll.print();
    cout<<"Input a number 'm' to find the values divisible by 'm' : ";
    cin>>m;
    cout<<"Values divisible by '"<<m<<"' : ";
    dll.checkDivisibility(m);
    cout<<"Input a number 'x' to delete all the nodes from the list that are greater than the given
value 'x' : ";
    cin>>x;
    cout<<"Deleting elements greater than "<<x<<" :"<<endl;
    dll.deleteGreater(x);
    dll.print();
    cout<<"Input a duplicate element from the list to find the number of elements between the
duplicate elements : ";
    cin>>duplicate;
    int elementsBetween=dll.elementsBetweenDuplicates(duplicate);
    if(elementsBetween!=-1)
        cout<<"Number of elements between the first pair of '"<<duplicate<<"' :
"<<elementsBetween<<endl;
    else
        cout<<"No duplicates found for value "<<duplicate<<endl;
    return 0;
}
```



```
Input the number of elements to be inserted in the list : 5
1
2
3
4
5
Doubly Linked List: 1 2 3 4 5
Input a number 'm' to find the values divisible by 'm' : 2
Values divisible by '2' : 2 4
Input a number 'x' to delete all the nodes from the list that are greater than the given value 'x' : 4
Deleting elements greater than 4 :
1 2 3 4
Input a duplicate element from the list to find the number of elements between the duplicate elements : 2
No duplicates found for value 2

Process returned 0 (0x0)   execution time : 25.920 s
Press any key to continue.
```