DSA ASSIGNMENT 6

QUESTION 1

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

struct Node{
   int data;
   Node* next;
   Node* prev;
};
void Insert(Node*&head,int value,int pos){
   Node*newNode=new Node();
   newNode->data=value;
   if (head==NULL){
      newNode->next=newNode;
      newNode->prev=newNode;
      head=newNode;
      return;
   }
   else if(pos==1){
      Node*last=head->prev;
      newNode->next=head;
      newNode->prev=last;
      head->prev=newNode;
      last->next=newNode;
      head=newNode;
   }
   else{
      Node*temp=head;
      int count=0;
      while (count<pos-1&&temp->next!=head) {
         temp=temp->next;
         count++;
      }
      Node*nextNode=temp->next;
      temp->next=newNode;
      newNode->prev=temp;
      newNode->next=nextNode;
      nextNode->prev=newNode;
      return;
   }
}
void Delete(Node*&head,int value){
   if(head==NULL){
      cout<<"List is empty, nothing to delete";
```

```
    }
    Node*temp=head;
    Node*toDelete=nullptr;

    do{
        if(temp->data==value) {
            toDelete=temp;
            break;
        }
        temp=temp->next;
    } while(temp!=head);

    if (!toDelete){
        cout<<"Value "<<value<<" not found.\n";
        return;
    }

    if(toDelete->next==toDelete){
        delete toDelete;
        head=nullptr;
    }
    else{
        Node*prevNode=toDelete->prev;
        Node*nextNode=toDelete->next;
        prevNode->next=nextNode;
        nextNode->prev=prevNode;

        if(toDelete == head)
            head=nextNode;

        delete toDelete;
    }

    cout<<"Node "<<value<<" deleted successfully.\n";
}
void Search(Node*&head, int value){
    Node*temp=head;
    int pos=1;
    do{
        if(temp->data == value) {
            cout<<"Value "<<value<<" found at position "<<pos<< ".\n";
            return;
        }
        temp=temp->next;
        pos++;
    } while(temp!=head);

    cout<< "Value "<<value<<" not found.\n";
}
```

```
int main(){
   Node* head = nullptr;
   int choice, value, pos;
   while(true){
      cout<<"Enter a choice\n";
      cout<<"1. Insert\n";
      cout<<"2. Delete\n";
      cout<<"3. Search\n";
      cout<<"4. Exit\n";
      cin >> choice;
      switch(choice){
         case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            cout << "Enter position to insert (1 for beginning): ";
            cin >> pos;
            Insert(head, value, pos);
            break;

         case 2:
            cout << "Enter value to delete: ";
            cin >> value;
            Delete(head, value);
            break;

         case 3:
            cout << "Enter value to search: ";
            cin >> value;
            Search(head, value);
            break;

         case 4:
            cout << "Exiting\n";
            return 0;

         default:
            cout << "Invalid choice. Try again.\n";
      }

   }
}
```

```
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
1
Enter value to insert: 1
Enter position to insert (1 for beginning): 1
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
1
Enter value to insert: 2
Enter position to insert (1 for beginning): 2
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
3
Enter value to search: 2
Value 2 found at position 2.
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
2
Enter value to delete: 2
Node 2 deleted successfully.
Enter a choice
1. Insert
2. Delete
3. Search
4. Exit
4
Exiting
```

QUESTION 2

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

struct Node {
   int data;
   Node* next;
};

void Insert(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   if(head==nullptr){
      head=newNode;
      head->next=head;
      return;
   }
   Node*temp=head;
   while (temp->next!=head)
      temp=temp->next;
   temp->next=newNode;
   newNode->next=head;
}

void displayCircular(Node* head) {
   if(head==nullptr){
      cout<<"List is empty.\n";
      return;
   }
   Node*temp=head;
   do{
      cout<<temp->data<< " ";
      temp=temp->next;
   } while(temp!=head);

   cout<<head->data<<endl;
}

int main() {
   Node*head=nullptr;

   Insert(head,20);
   Insert(head,100);
   Insert(head,40);
   Insert(head,80);
   Insert(head,60);
   cout<<"Output: ";
```

```
    displayCircular(head);

    return 0;
}
```

```
Output: 20 100 40 80 60 20
[1] + Done
```

QUESTION 3

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

struct Node{
   int data;
   Node* next;
   Node* prev;
};

void InsertDoubly(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   newNode->next=nullptr;
   newNode->prev=nullptr;

   if(head==nullptr){
      head=newNode;
      return;
   }

   Node*temp=head;
   while(temp->next!=nullptr)
      temp=temp->next;

   temp->next=newNode;
   newNode->prev=temp;
}

int sizeDoubly(Node* head){
   int count=0;
   Node* temp=head;
   while(temp){
      count++;
      temp=temp->next;
   }
   return count;
}

struct NodeC{
   int data;
   NodeC* next;
};

void InsertCircular(NodeC*& head, int value) {
   NodeC*newNode=new NodeC();
   newNode->data=value;
```

```
    if(head==nullptr){
        head=newNode;
        head->next=head;
        return;
    }
    NodeC*temp=head;
    while (temp->next!=head)
        temp=temp->next;
    temp->next=newNode;
    newNode->next=head;
}
int sizeCircular(NodeC*head2) {
    if(!head2)
        return 0;

    int count=0;
    NodeC*temp=head2;
    do{
        count++;
        temp=temp->next;
    } while(temp!=head2);

    return count;
}

int main() {
    Node* head=nullptr;
    InsertDoubly(head,10);
    InsertDoubly(head,20);
    InsertDoubly(head,30);
    InsertDoubly(head,40);

    cout<<"Size of Doubly Linked List: "<<sizeDoubly(head)<< endl;

    NodeC* head2=nullptr;
    InsertCircular(head2, 20);
    InsertCircular(head2, 100);
    InsertCircular(head2, 40);
    InsertCircular(head2, 80);
    InsertCircular(head2, 60);

    cout<<"Size of Circular Linked List: "<<sizeCircular(head2)<<endl;
    return 0;
}
```

```
Size of Doubly Linked List: 4
Size of Circular Linked List: 5
[1] + Done
```

QUESTION 4

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

struct Node {
   char data;
   Node* next;
   Node* prev;
};

void Insert(Node*& head, char value) {
   Node*newNode=new Node();
   newNode->data=value;
   newNode->next=nullptr;
   newNode->prev=nullptr;

   if(head==nullptr){
      head=newNode;
      return;
   }

   Node*temp=head;
   while(temp->next!=nullptr)
      temp=temp->next;

   temp->next=newNode;
   newNode->prev=temp;
}

bool isPalindrome(Node*head) {
   if(head==nullptr)
      return true;

   Node*left=head;
   Node*right=head;
   while(right->next!=nullptr)
      right=right->next;

   while(left!=right && right->next!=left) {
      if (left->data!=right->data)
         return false;
      left=left->next;
      right=right->prev;
   }
   return true;
}

int main() {
```

```
    Node*head=nullptr;
    string input;

    cout<<"Enter characters (no spaces): ";
    cin>>input;

    for(char ch : input)
        Insert(head,ch);

    if(isPalindrome(head))
        cout<<"The doubly linked list is a palindrome."<<endl;
    else
        cout<<"The doubly linked list is NOT a palindrome."<<endl;

    return 0;
}
```

```
Enter characters (no spaces): Kush
The doubly linked list is NOT a palindrome.
[1] + Done                       "/usr/bin/g
```

```
Enter characters (no spaces): eye
The doubly linked list is a palindrome.
[1] + Done                       "/usr/bin/g
```

QUESTION 5

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

struct Node {
   int data;
   Node* next;
};

void Insert(Node*& head, int value) {
   Node*newNode=new Node();
   newNode->data=value;
   if(head==nullptr){
      head=newNode;
      head->next=head;
      return;
   }
   Node*temp=head;
   while (temp->next!=head)
      temp=temp->next;
   temp->next=newNode;
   newNode->next=head;
}

bool isCircular(Node* head){
   if(head==nullptr)
      return false;

   Node*temp=head->next;
   while(temp!=nullptr && temp!=head)
      temp=temp->next;

   return(temp==head);
}

int main() {
   Node* head=nullptr;

   Insert(head,10);
   Insert(head,20);
   Insert(head,30);
   Insert(head,40);
   Insert(head,50);

   if (isCircular(head))
      cout<<"The linked list is circular."<<endl;
   else
```

```
        cout<<"The linked list is NOT circular."<<endl;

    return 0;
}
```

```
The linked list is circular.
[1] + Done
```