ASSIGNMENT-3

NAME – ANSHIKA

ROLL NO. – 102003183

SUBGROUP – 2COE8

Question 1-

(a)Circular Linked List

```
#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;


struct node{

	int info;

	struct node *next;

}*last;


class circularLinkedList{

	public:

		void createNode(int val);

		void insertBegin(int val);

		void insertEnd(int val);

		void insertAfter(int val, int pos);

		void delBegin();

		void delEnd();

		void delBetween(int val);

		void search(int val);

		void display();

		circularLinkedList(){

			last=NULL;

		}
```

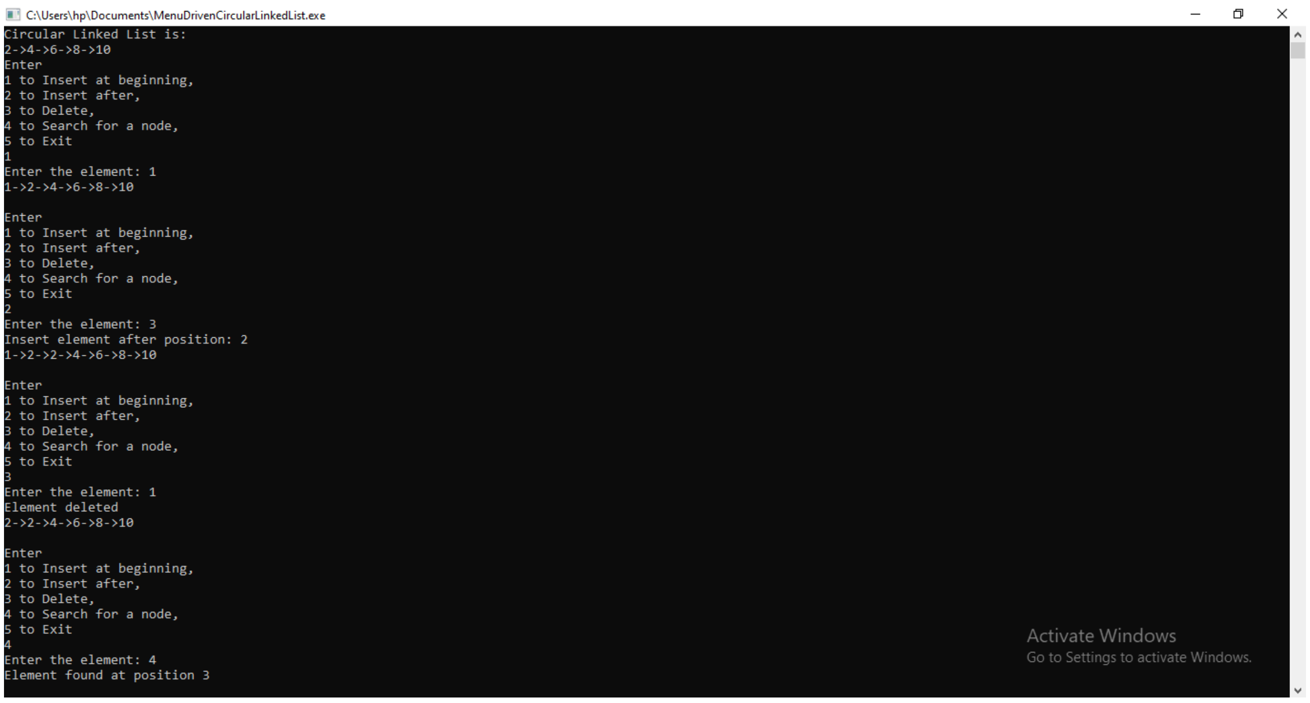
```
};

void circularLinkedList::createNode(int val){
	struct node *temp;
	temp=new(struct node);
	temp->info=val;
	if(last==NULL){
		last=temp;
		temp->next=last;
	}
	else{
		temp->next=last->next;
		last->next=temp;
		last=temp;
	}
}

void circularLinkedList::insertBegin(int val){
	if(last==NULL){
		cout<<"First create the list"<<endl;
		return;
	}
	struct node *temp;
	temp=new(struct node);
	temp->info=val;
	temp->next=last->next;
	last->next=temp;
}


void circularLinkedList::insertAfter(int val, int pos){
	if(last==NULL){
		cout<<"First create the list"<<endl;
		return;
	}
	struct node *temp, *s;
	s=last->next;
	for(int i=0; i<pos; i++){
		s=s->next;
		if(s==last->next){
			cout<<"There are less than "<<pos<<" in the list"<<endl;
			return;
		}
	}
	temp=new(struct node);
	temp->next=s->next;
	temp->info=val;
	s->next=temp;
	if(s==last){
		last=temp;
	}
}
```

```
void circularLinkedList::delBegin(){
        struct node *temp;
        while(last->next!=last){
                last=last->next;
        }
        temp=last;
        last=last->next;
        last->next=last;
        free(temp);
        cout<<"Element deleted"<<endl;
}

void circularLinkedList::delEnd(){
        struct node *prev, *current;
        current=last;
        while(current->next!=last){
                prev=current;
                current=current->next;
        }
        prev->next=last;
        free(current);
        cout<<"Element deleted"<<endl;
}

void circularLinkedList::delBetween(int val){
        struct node *temp, *s;
```

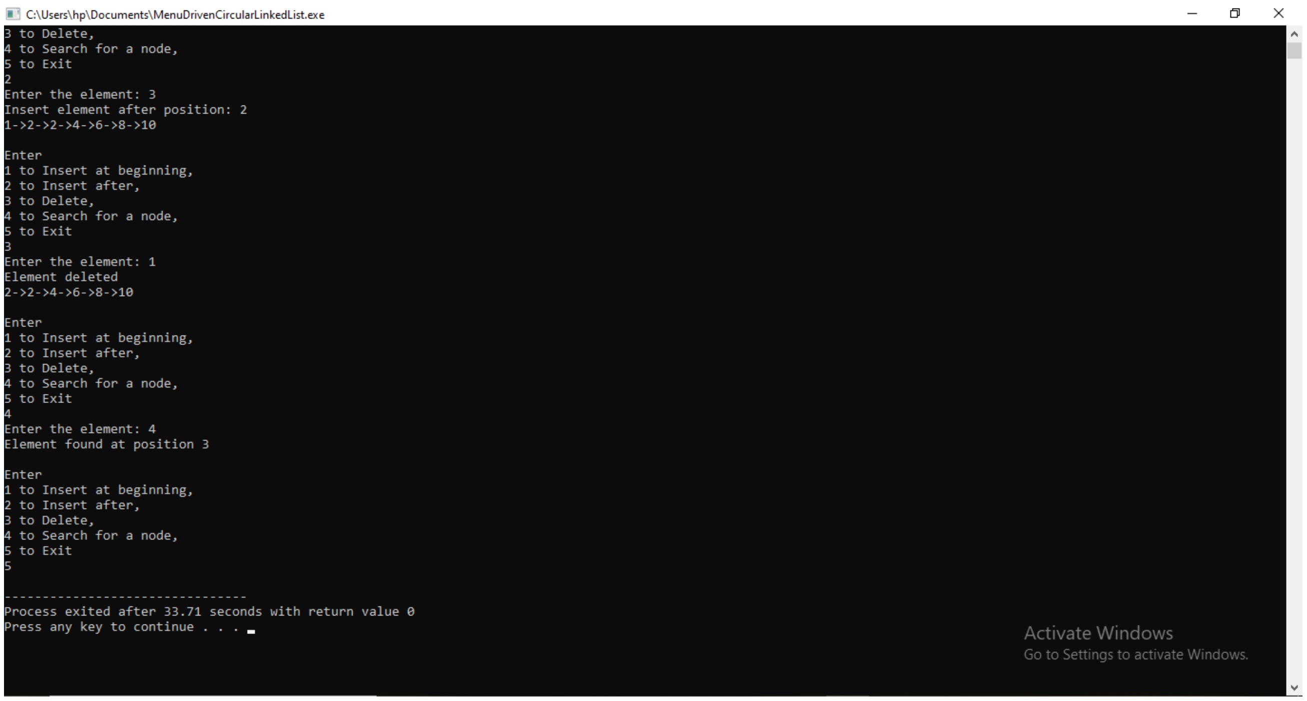
```
s=last->next;

if(last->next==last && last->info==val){

        temp=last;

        last=NULL;

        free(temp);

        return;

}

if(s->info==val){

        temp=s;

        last->next=s->next;

        free(temp);

        cout<<"Element deleted"<<endl;

        return;

}

while(s->next!=last){

        if(s->next->info==val){

                temp=s->next;

                s->next=temp->next;

                free(temp);

                cout<<"Element deleted"<<endl;

                return;

        }

        s=s->next;

}

if(s->next->info==val){

        temp=s->next;
```

```
                s->next=last->next;
                free(temp);
                cout<<"Element deleted"<<endl;
                last=s;
                return;
        }
        cout<<"Element not found in the list"<<endl;
}

void circularLinkedList::search(int val){
        struct node *s;
        int index=0;
        s=last->next;
        while(s!=last){
                if(s->info==val){
                        cout<<"Element found at position "<<index+1<<endl;
                        return;
                }
                index++;
                s=s->next;
        }
        if(s->info==val){
                index++;
                cout<<"Element found at position "<<index<<endl;
                return;
        }
        cout<<"Element not found in the list";
}

void circularLinkedList::display(){
        struct node *s;
        if(last==NULL){
                cout<<"List is empty!";
                return;
        }
        s=last->next;
        while(s!=last){
                cout<<s->info<<"->";
                s=s->next;
        }
        cout<<s->info<<endl;
}

int main(){
        int choice, element, pos;
        circularLinkedList cl;
        cl.createNode(2);
        cl.createNode(4);
        cl.createNode(6);
        cl.createNode(8);
        cl.createNode(10);
        cout<<"Circular Linked List is: "<<endl;
        cl.display();

        cout<<"Enter\n1 to Insert at beginning,\n2 to Insert after,\n3 to Delete,\n4 to Search
for a node,\n5 to Exit\n";

        cin>>choice;

        while(choice!=5){

                switch(choice){

                        case 1 : cout<<"Enter the element: ";

                                        cin>>element;

                                        cl.insertBegin(element);

                                        cl.display();

                                        break;

                        case 2 : cout<<"Enter the element: ";

                                        cin>>element;

                                        cout<<"Insert element after position: ";

                                        cin>>element;

                                        cl.insertAfter(element, pos);

                                        cl.display();

                                        break;

                        case 3 : cout<<"Enter the element: ";

                                        cin>>element;

                                        cl.delBetween(element);

                                        cl.display();

                                        break;

                        case 4 : if(last==NULL){

                                                cout<<"List is empty!"<<endl;

                                                break;

                                        }
```

```
                                                cout<<"Enter the element: ";

                                                cin>>element;

                                                cl.search(element);

                                                break;

                                case 5 : return 0;

                        }

        cout<<"\nEnter\n1 to Insert at beginning,\n2 to Insert after,\n3 to Delete,\n4 to Search for a node,\n5 to Exit\n";

        cin>>choice;

        }

        return 0;

}
```



```
C:\Users\hp\Documents\MenuDrivenCircularLinkedList.exe
Circular Linked List is:
2->4->6->8->10
Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
1
Enter the element: 1
1->2->4->6->8->10

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
2
Enter the element: 3
Insert element after position: 2
1->2->2->4->6->8->10

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
3
Enter the element: 1
Element deleted
2->2->4->6->8->10

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
4
Enter the element: 4
Element found at position 3
```



C:\Users\hp\Documents\MenuDrivenCircularLinkedList.exe

```
3 to Delete,
4 to Search for a node,
5 to Exit
2
Enter the element: 3
Insert element after position: 2
1->2->2->4->6->8->10

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
3
Enter the element: 1
Element deleted
2->2->4->6->8->10

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
4
Enter the element: 4
Element found at position 3

Enter
1 to Insert at beginning,
2 to Insert after,
3 to Delete,
4 to Search for a node,
5 to Exit
5

--------------------------------
Process exited after 33.71 seconds with return value 0
Press any key to continue . . .
```

(b)Doubly Linked List

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

class node{
	public:
		node* next;
		node* prev;
		int data;
};

void insertBegin(node** head){
	node* newNode=new node;
	cout<<"\nEnter value for new node: ";
	cin>>newNode->data;
	if(*head==NULL){
		newNode->next=newNode;
		newNode->prev=newNode;
		*head=newNode;
	}
	else{
		newNode->next=*head;
		newNode->prev=(*head)->prev;
		((*head)->prev)->next=newNode;
		(*head)->prev=newNode;
		*head=newNode;
	}
}

void insertEnd(node** head){
	node* newNode=new node;
	cout<<"\nEnter value for new node: ";
	cin>>newNode->data;
	if(*head==NULL){
		newNode->next=newNode;
		newNode->prev=newNode;
		*head=newNode;
	}
	else{
		node* current=*head;
		while(current->next!=*head){
			current=current->next;
		}
		newNode->next=current->next;
		newNode->prev=current;
		(current->next)->prev=newNode;
		current->next=newNode;
	}
}

void insertAfter(node** head){
```

```
node* newNode=new node;
if(*head==NULL){
        cout<<"\nThere is no element in the list";
        cout<<"\nCreating a new node";
        newNode->prev=newNode;
        newNode->next=newNode;
        *head=newNode;
}
else{
        int num;
        cout<<"Enter after element: ";
        cin>>num;
        node* current=*head;
        while(current->data!=num){
                current=current->next;
                if(current==*head){
                        cout<<"\nEntered element not found in list\n";
                        return;
                }
        }
        cout<<"Enter value for new node: ";
        cin>>newNode->data;
        newNode->next=current->next;
        newNode->prev=current;
        (current->next)->prev=newNode;
        current->next=newNode;
        }
}

void insertBefore(node** head){
        node* newNode=new node;
        if(*head==NULL){
                cout<<"List is empty! Creating new node...";
                cout<<"\nEnter value for new node: ";
                cin>>newNode->data;
                newNode->prev=newNode;
                newNode->next=newNode;
                *head=newNode;
        }
        else{
                int num;
                cout<<"\nEnter before element: ";
                cin>>num;
                if((*head)->data==num){
                        insertBegin(head);
                }
                else{
                        node* current=(*head)->next;
                        while(current->data!=num){
                                if(current==*head){
                                        cout<<"\nEntered element not found in list\n";
                                        return;
```

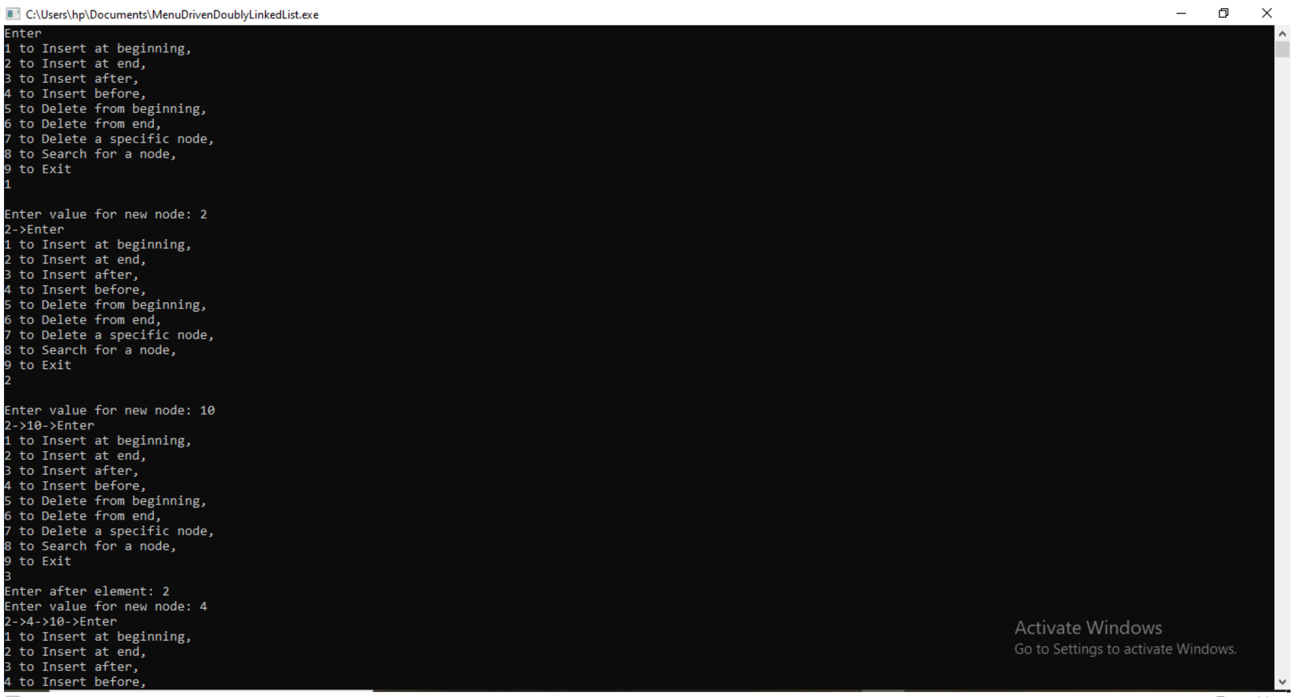
```
                    }
                    current=current->next;
                }
                cout<<"Enter data for new node: ";
                cin>>newNode->data;
                newNode->next=current;
                newNode->prev=current->prev;
                (current->prev)->next=newNode;
                current->prev=newNode;
            }
        }
}

void delBegin(node** head){
        if(*head==NULL){
            cout<<"\nList is empty!\n";
        }
        else if((*head)->next==*head){
            delete *head;
            *head=NULL;
        }
        else{
            node* current=new node;
            current=(*head)->next;
            current->prev=(*head)->prev;
            ((*head)->prev)->next=current;
```

```
                delete *head;

                *head=current;

        }

}


void delEnd(node** head){

        if(*head==NULL){

                cout<<"\nList is empty!\n";

        }

        else if((*head)->next==*head){

                delete *head;

                *head=NULL;

        }

        else{

                node* current=new node;

                current=*head;

                while(current->next!=(*head)){

                        current=current->next;

                }

                (current->prev)->next=current->next;

                (current->next)->prev=current->prev;

                delete current;

        }

}


void delBetween(node** head){
        if(*head==NULL){
                cout<<"\nList is empty!\n";
        }
        else{
                int val;
                cout<<"\nEnter element to be deleted: ";
                cin>>val;
                if((*head)->data==val){
                        delBegin(head);
                }
                else{
                        node* current=(*head)->next;
                        while((current->data)!=val){
                                if(current==(*head)){
                                        cout<<"\nEntered element not found in list\n";
                                        return;
                                }
                                current=current->next;
                        }
                        (current->prev)->next=current->next;
                        (current->next)->prev=current->prev;
                        delete current;
                }
        }
}
```

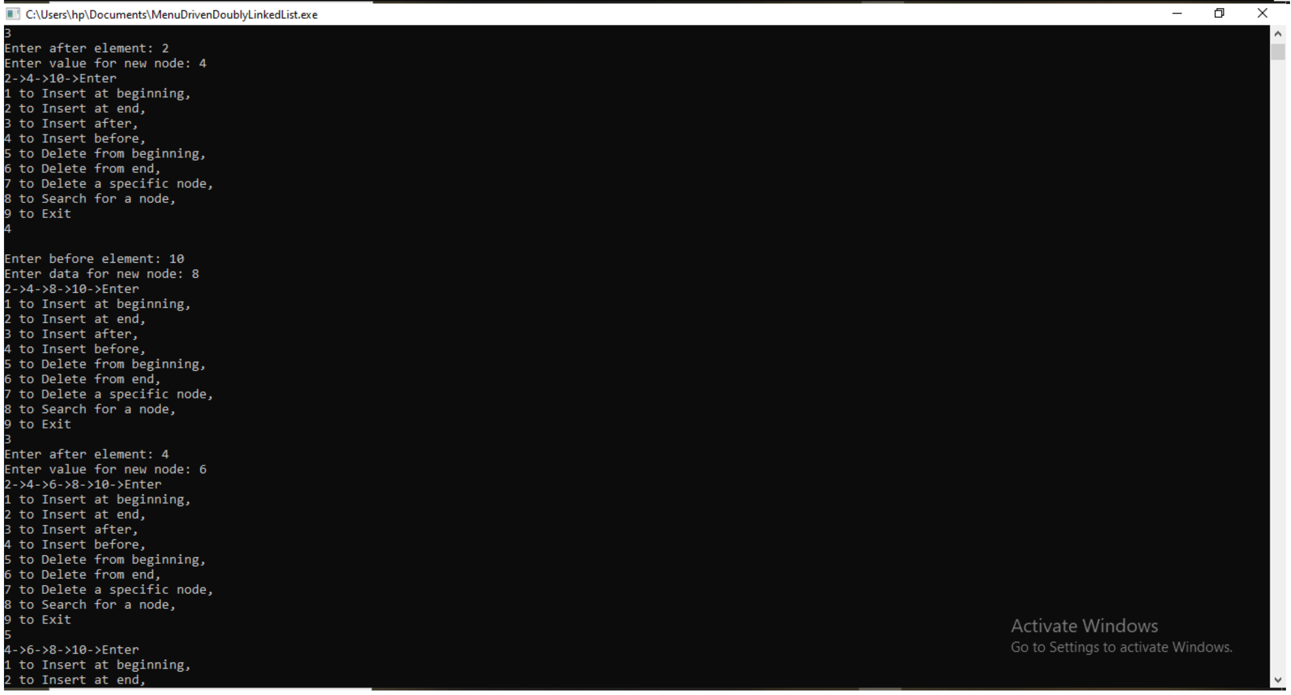
```
void search(node* head){
        if(head==NULL){
                cout<<"List is empty!";
                return;
        }
        int val;
        cout<<"\nEnter value to be searched: ";
        cin>>val;
        node* current=head;
        int index=0, count=0;
        do{
                if(current->data==val){
                        cout<<"Value found at position: "<<index+1;
                        count++;
                }
                index++;
                current=current->next;
        }while(current!=head);
        if(count==0){
                cout<<"Value searched not found in list";
        }
}

void display(node* head){
        node* current=head;
        if(current==NULL){
```

```
		cout<<"\nList is empty!";
	}
	else{
		do{
			cout<<current->data<<"->";
			current=current->next;
		}while(current!=head);
	}
}


int main(){
	int choice;
	node* head=NULL;
	cout<<"Enter\n1 to Insert at beginning,\n2 to Insert at end,\n3 to Insert after,\n4 to
Insert before,\n5 to Delete from beginning,\n6 to Delete from end,\n7 to Delete a specific
node,\n8 to Search for a node,\n9 to Exit\n";
	cin>>choice;
	while(choice!=9){
		switch(choice){
			case 1 : insertBegin(&head);
					display(head);
					break;
			case 2 : insertEnd(&head);
					display(head);
					break;
			case 3 : insertAfter(&head);
					display(head);
```

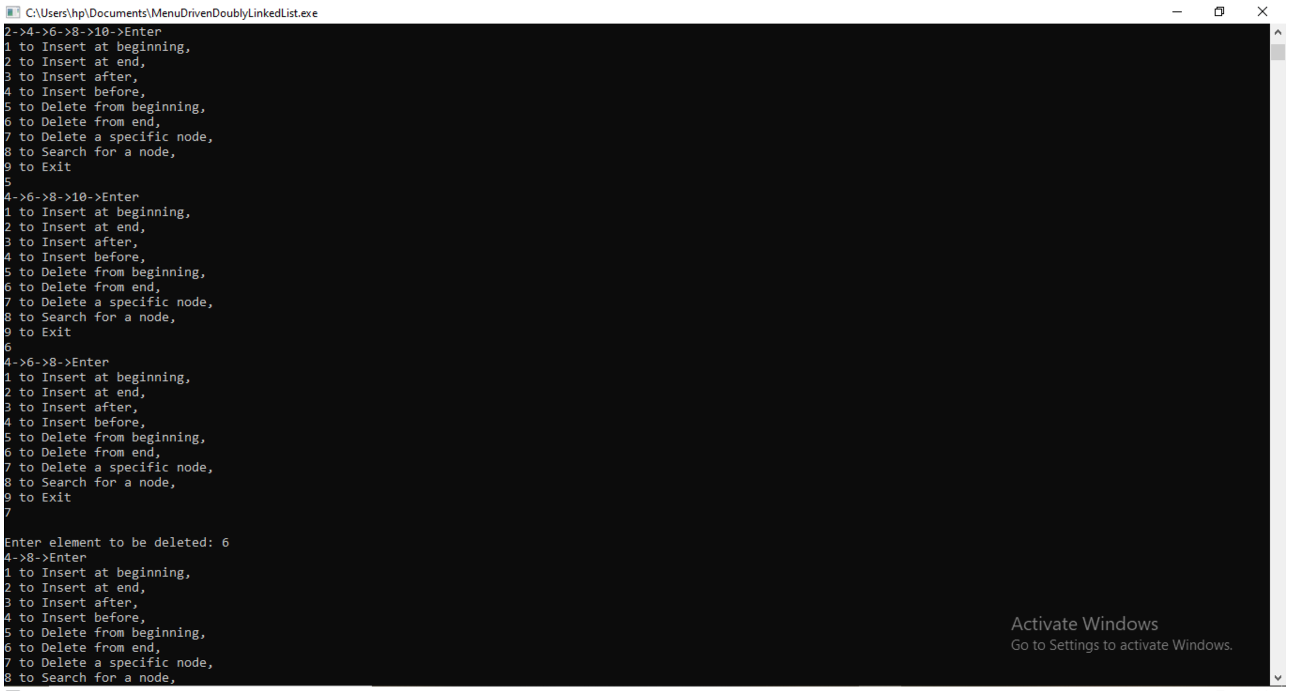
```
                                        break;

                        case 4 : insertBefore(&head);

                                        display(head);

                                        break;

                        case 5 : delBegin(&head);

                                        display(head);

                                        break;

                        case 6 : delEnd(&head);

                                        display(head);

                                        break;

                        case 7 : delBetween(&head);

                                        display(head);

                                        break;

                        case 8 : search(head);

                                        break;

                        case 9 : return 0;

                }

        cout<<"Enter\n1 to Insert at beginning,\n2 to Insert at end,\n3 to Insert after,\n4 to
Insert before,\n5 to Delete from beginning,\n6 to Delete from end,\n7 to Delete a specific
node,\n8 to Search for a node,\n9 to Exit\n";

        cin>>choice;

        }

        return 0;

}
```



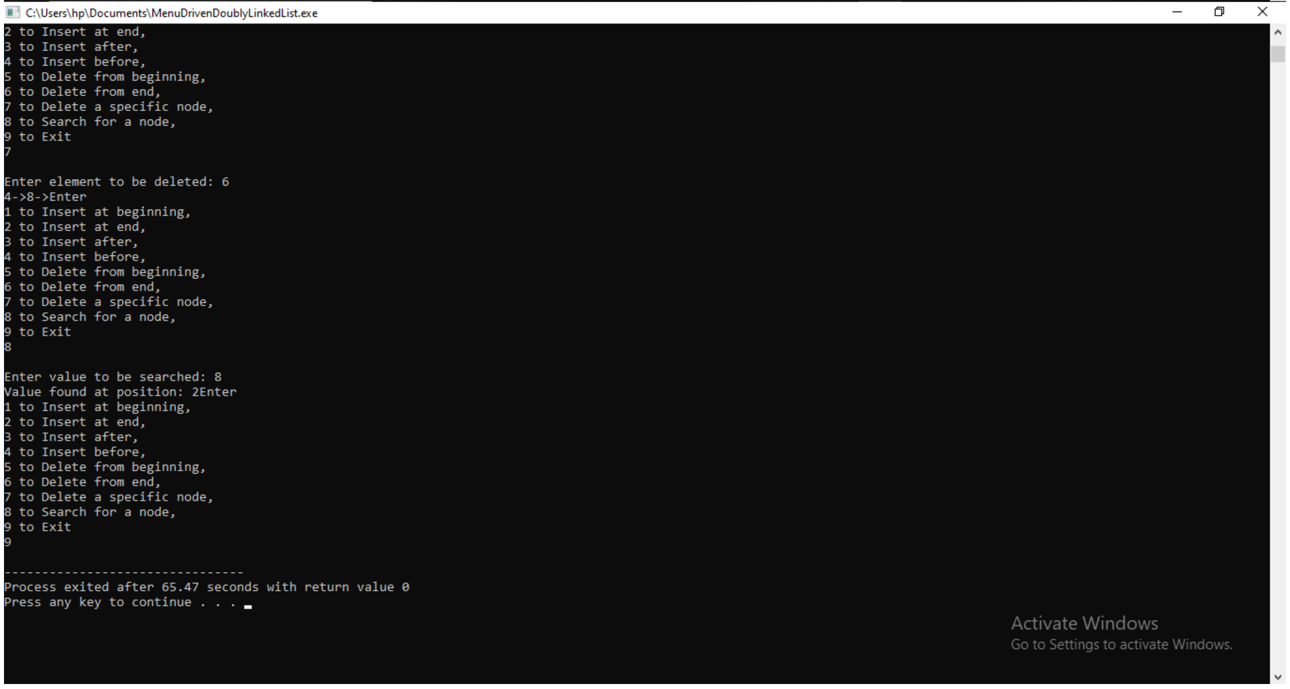
```
C:\Users\hp\Documents\MenuDrivenDoublyLinkedList.exe
Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
1

Enter value for new node: 2
2->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
2

Enter value for new node: 10
2->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
3
Enter after element: 2
Enter value for new node: 4
2->4->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
```



```
C:\Users\hp\Documents\MenuDrivenDoublyLinkedList.exe
3
Enter after element: 2
Enter value for new node: 4
2->4->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
4

Enter before element: 10
Enter data for new node: 8
2->4->8->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
3
Enter after element: 4
Enter value for new node: 6
2->4->6->8->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
5
4->6->8->10->Enter
1 to Insert at beginning,
2 to Insert at end,
```


C:\Users\hp\Documents\MenuDrivenDoublyLinkedList.exe

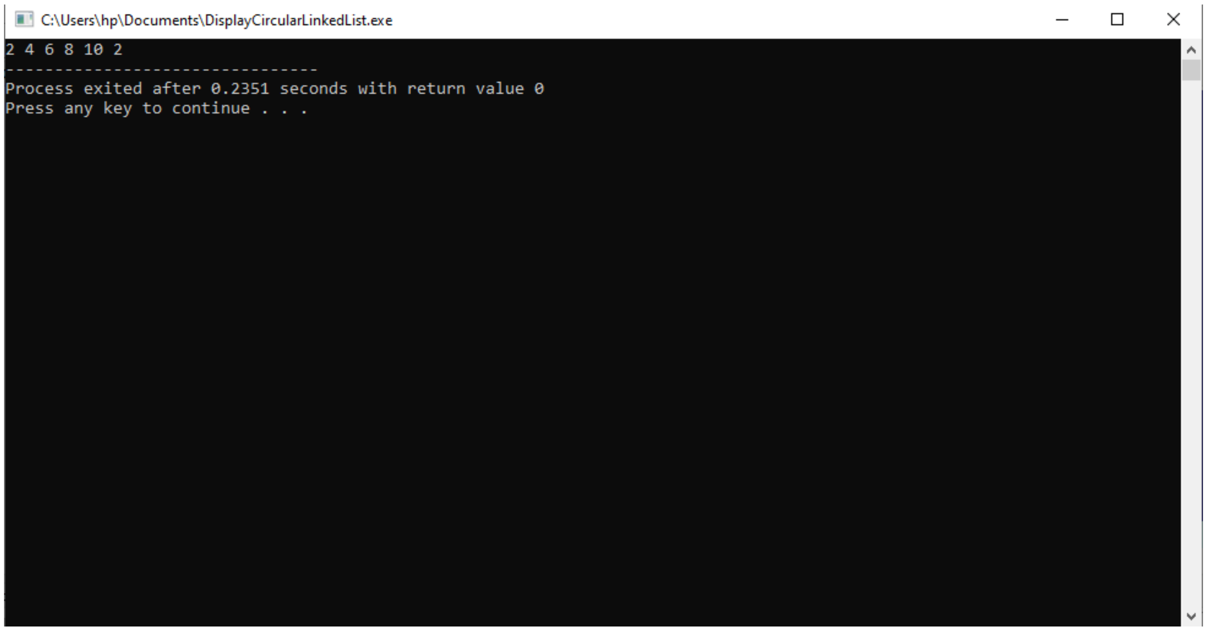
```
2->4->6->8->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
5
4->6->8->10->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
6
4->6->8->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
7

Enter element to be deleted: 6
4->8->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
```


C:\Users\hp\Documents\MenuDrivenDoublyLinkedList.exe

```
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
7

Enter element to be deleted: 6
4->8->Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
8

Enter value to be searched: 8
Value found at position: 2Enter
1 to Insert at beginning,
2 to Insert at end,
3 to Insert after,
4 to Insert before,
5 to Delete from beginning,
6 to Delete from end,
7 to Delete a specific node,
8 to Search for a node,
9 to Exit
9

--------------------------------
Process exited after 65.47 seconds with return value 0
Press any key to continue . . .
```

Question 2-

```
#include <iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;


struct node {

   int data;

   struct node *next;

};


struct node* head = NULL;


void insert(int data) {

   struct node *newNode = (struct node *)malloc(sizeof(struct node));

   struct node *ptr = head;

   newNode->data = data;

   newNode->next = head;

   if (head!= NULL) {

      while (ptr->next != head)

      ptr = ptr->next;

      ptr->next = newNode;

   }

   else{

             newNode->next = newNode;

   }
   head = newNode;
}

void display() {
   struct node* ptr;
   ptr = head;
   do {
      cout<<ptr->data <<" ";
      ptr = ptr->next;
   } while(ptr != head);
   cout<<ptr->data;
}

int main() {
   insert(10);
   insert(8);
   insert(6);
   insert(4);
   insert(2);
   display();
   return 0;
}
```



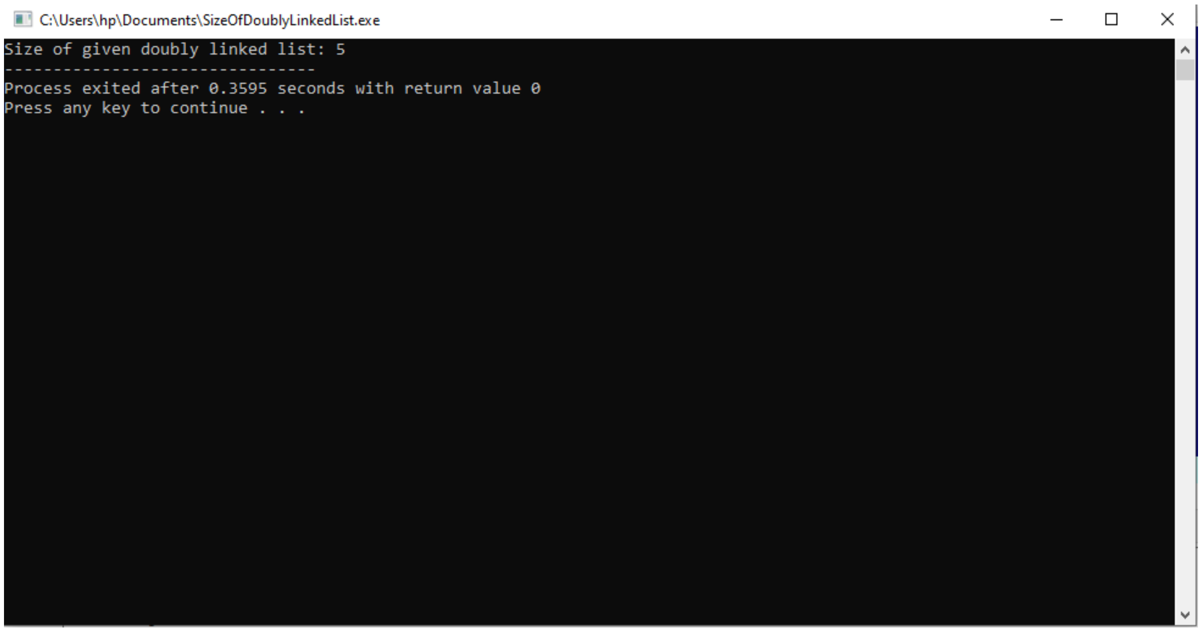
```
2 4 6 8 10 2
--------------------------------
Process exited after 0.2351 seconds with return value 0
Press any key to continue . . .
```

Question 3-

(a) #include<iostream>

```
#include<bits/stdc++.h>
using namespace std;

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

void push(struct node** head, int data){
    struct node* newNode=new node;
    newNode->data=data;
    newNode->next=*head;
    newNode->prev=NULL;
    if((*head)!=NULL){
        (*head)->prev=newNode;
    }
    *head=newNode;
}

int size(struct node *Node){
    int ans=0;
    while(Node!=NULL){
        ans++;
```

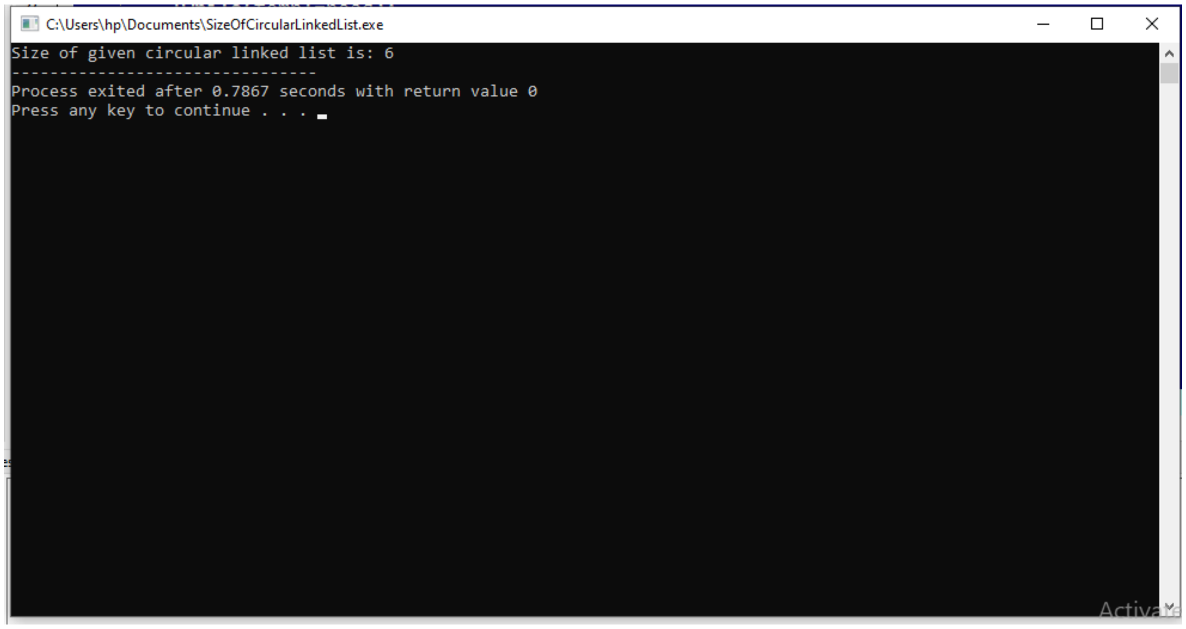
```
                Node=Node->next;
        }
        return ans;
}

int main(){
        struct node* head=NULL;
        push(&head, 2);
        push(&head, 4);
        push(&head, 6);
        push(&head, 8);
        push(&head, 10);
        cout<<"Size of given doubly linked list: "<<size(head);
        return 0;
}
```



```
Size of given doubly linked list: 5
--------------------------------
Process exited after 0.3595 seconds with return value 0
Press any key to continue . . .
```

(b) #include<iostream>

```
#include<bits/stdc++.h>
using namespace std;

struct node{
        int data;
        node* next;
        node(int x){
                data=x;
                next=NULL;
        }
};

struct node* push(struct node* last, int data){
        if(last==NULL){
                struct node* temp=(struct node*)malloc(sizeof(struct node));
                temp->data=data;
                last=temp;
                temp->next=last;
                return last;
        }
        struct node* temp=(struct node*)malloc(sizeof(struct node));
        temp->data=data;
        temp->next=last->next;
        last->next=temp;
        return last;
}

int size(node* head){
        node* temp=head;
        int ans=0;
        if(head!=NULL){
                do{
                        temp=temp->next;
                        ans++;
                }while(temp!=head);
        }
        return ans;
}

int main(){
        node* head=NULL;
        head=push(head, 2);
        head=push(head, 4);
        head=push(head, 6);
        head=push(head, 8);
        head=push(head, 10);
        head=push(head, 12);
        cout<<"Size of given circular linked list is: "<<size(head);
        return 0;
}
```

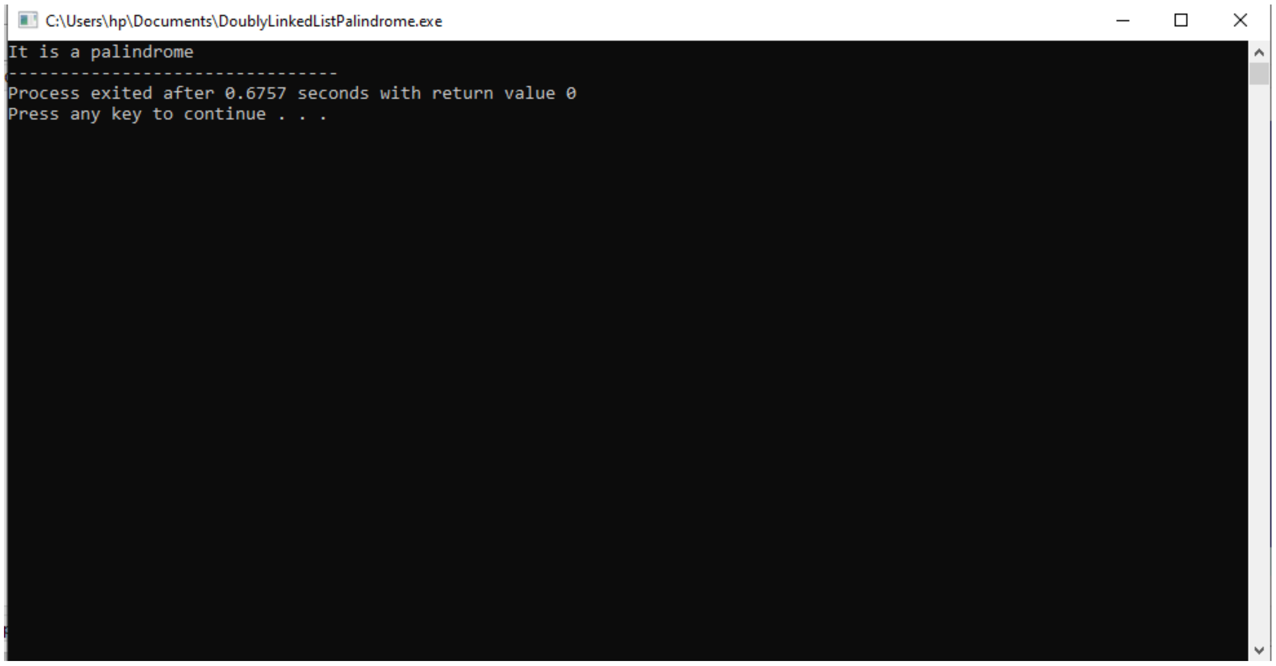


C:\Users\hp\Documents\SizeOfCircularLinkedList.exe

```
Size of given circular linked list is: 6
--------------------------------
Process exited after 0.7867 seconds with return value 0
Press any key to continue . . .
```

Question 4-

```
#include<iostream>

#include<bits/stdc++.h>

using namespace std;


struct Node{

        char data;

        struct Node *next;

        struct Node *prev;

};


void push(struct Node** head, char data){

        struct Node* node=new Node;

        node->data=data;

        node->next=*head;

        node->prev=NULL;

        if((*head)!=NULL){

                (*head)->prev=node;

        }

        *head=node;

}


bool isPalindrome(struct Node *left){

        if(left==NULL){

                return true;

        }
        struct Node *right=left;
        while(right->next!=NULL){
                right=right->next;
        }
        while(left!=right){
                if(left->data!=right->data){
                        return false;
                }
                left=left->next;
                right=right->prev;
        }
        return true;
}

int main(){
        struct Node* head=NULL;
        push(&head, 'l');
        push(&head, 'e');
        push(&head, 'v');
        push(&head, 'e');
        push(&head, 'l');
        if(isPalindrome(head)){
                cout<<"It is a palindrome";
        }
        else{
                cout<<"It is not a palindrome";
        }

        return 0;

}
```

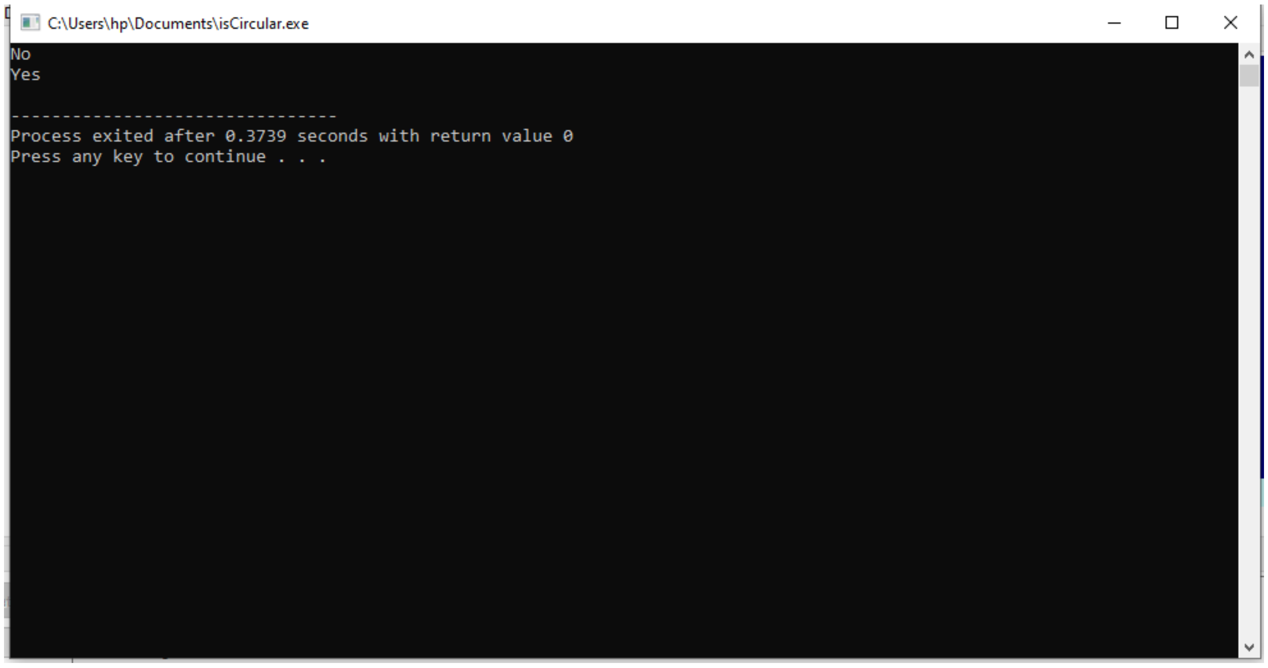


C:\Users\hp\Documents\DoublyLinkedListPalindrome.exe

```
It is a palindrome
--------------------------------
Process exited after 0.6757 seconds with return value 0
Press any key to continue . . .
```

Question 5-

```
#include<iostream>

#include<bits/stdc++.h>

using namespace std;


struct Node{

	int data;

	struct Node* next;

};


bool isCircular(struct Node *head){

	if(head==NULL){

		return true;

	}

	struct Node *node=head->next;

	while(node!=NULL && node!=head){

		node=node->next;

	}

	return (node==head);

}


Node *newNode(int data){

	struct Node *temp=new Node;

	temp->data=data;

	temp->next=NULL;

	return temp;
}

int main(){
	struct Node* head=newNode(1);
	head->next=newNode(2);
	head->next->next=newNode(3);
	head->next->next->next=newNode(4);
	if(isCircular(head)){
		cout<<"Yes"<<endl;
	}
	else{
		cout<<"No"<<endl;
	}
	head->next->next->next->next=head;
	if(isCircular(head)){
		cout<<"Yes"<<endl;
	}
	else{
		cout<<"No"<<endl;
	}
	return 0;
}
```



```
No
Yes

--------------------------------
Process exited after 0.3739 seconds with return value 0
Press any key to continue . . .
```