**NAME:ABHINAV**

**ROLL NO.:101903587**

**2COE23**

**ASSIGNMENT 3**

**Q1**

**Using circular linked list:**

#include<iostream>

#include<stdlib.h>

//circular linked listt

using namespace std;

class node

{

public:

int data;

node \*next;

};

void display(node \*start)

{

cout<<"\nThe list is:"<<endl;

node \*ptr;

ptr=start;

while(ptr->next!=start)

{

cout<<ptr->data<<endl;

ptr=ptr->next;

}

cout<<ptr->data<<endl;

}

node\* makelist(int n)

{

node \*ptr1,\*ptr2,\*start;

start=NULL;

for(int i=0; i<n; i++)

{

ptr1=new node();

cout<<"enter data"<<endl;

cin>>ptr1->data;

ptr1->next=NULL;

if(start==NULL)

{

start=ptr1;

}

else

{

ptr2=start;

while(ptr2->next!=NULL)

{

ptr2=ptr2->next;

}

ptr2->next=ptr1;

}

if(i == n-1)

{

ptr1->next=start;

}

}

return start;

}

node\* ins\_beg(node \*start) //function of insertion at beginning

{

node \*ptr,\*tail;

tail=start;

while(tail->next!=start)

{

tail=tail->next;

}

ptr=new node();

cout<<"enter data in new node"<<endl;

cin>>ptr->data;

ptr->next=tail->next;

tail->next=ptr;

start=ptr;

return start;

}

void ins\_end(node \*start) //func to insert a new a node at end

{

node \*ptr,\*tail;

tail=start;

while(tail->next!=start)

{

tail=tail->next;

}

ptr=new node();

cout<<"enter data in new node"<<endl;

cin>>ptr->data;

ptr->next=tail->next;

tail->next=ptr;

tail=ptr;

display(start);

}

node\* ins\_mid(node \*start, int n, int pos) //func to insert at mid

{

node \*tail;

tail=start;

int i=0;

if(pos<0 || pos>n)

{

cout<<"invalid position"<<endl;

}

else if(pos==1)

{

start=ins\_beg(start);

return start;

}

else

{

while(i != pos-2)

{

tail=tail->next;

i++;

}

node \*ptr;

ptr=new node();

cout<<"enter the data in new node"<<endl;

cin>>ptr->data;

ptr->next=tail->next;

tail->next=ptr;

}

return start;

}

//deletion starts here

node\* searching(node \*start,int num) // node searching function

{

node \*ptr;

ptr=start;

while(ptr!=NULL)

{

if(ptr->data==num)

{

return ptr;

}

else

{

ptr=ptr->next;

}

}

if(ptr==NULL)

{

cout<<"sorry given number not found in list"<<endl;

}

}

node\* deletioni(node \*start,node \*ptr) //deletion in-between function

{

node \*preptr;

preptr=start;

while(preptr->next!=NULL)

{

if(preptr->next==ptr)

{

break;

}

preptr=preptr->next;

}

preptr->next=ptr->next;

ptr->next=NULL;

delete ptr;

}

node\* deletione(node \*start) //function for deletion in end

{

node \*ptr,\*temp,\*tail;

tail=start;

while(tail->next!=start)

{

tail=tail->next;

}

temp=tail->next;

while(temp->next != tail->next)

{

ptr=temp;

temp=temp->next;

}

ptr->next=temp->next;

tail=ptr;

free(temp);

return start;

}

node\* deletionb(node \*start) //functon for deletion in beginning

{

node \*temp,\*tail;

tail=start;

while(tail->next!=start)

{

tail=tail->next;

}

temp=start;

tail->next=start->next;

start=start->next;

temp->next=NULL;

free(temp);

return start;

}

//searching

int searchn(node \*start, int item)

{

node \*ptr=NULL;

ptr=start;

int count=0;

while(ptr->next!=start)

{

count=count+1;

if(ptr->data==item)

{

return count;

}

else

{

ptr=ptr->next;

}

}

}

int main()

{

node \*start,\*loc;

int n,i,num,j,k,num\_search,ser\_ching;

cout<<"enter number of nodes you want"<<endl;

cin>>n;

start=makelist(n);

display(start);

again:

cout<<"what you want to do \ninsertion: type 1\ndeletion: type 2\nsearching: type 3"<<endl;

cin>>k;

if(k==1)

{

back\_ins:

cout<<"enter where you want to insert? 1 for beginning, 2 for end, 3 for mid"<<endl;

cin>>i;

if(i==1)

{

start=ins\_beg(start);

display(start);

}

else if(i==2)

{

ins\_end(start);

}

else if(i==3)

{

int pos;

cout<<"enter the position you want to add a new node"<<endl;

cin>>pos;

start=ins\_mid(start,n,pos);

display(start);

}

else

{

cout<<"please enter from 1, 2, and 3"<<endl;

goto back\_ins;

}

}

//deletion

else if(k==2)

{

back\_del:

cout<<"enter what you want to do: 1 for deletion in between, 2 for deletion in end, 3 for deletion in beginning"<<endl;

cin>>j;

if(j==1)

{

cout<<"enter the data you want to delete: "<<endl;

cin>>num;

loc=searching(start,num); //for searching the node

deletioni(start,loc); //deletion in between

display(start);

}

else if(j==2)

{start=deletione(start); //deletion in end

display(start);}

else if(j==3)

{start=deletionb(start); //deletion in beginning

display(start);}

else

{

cout<<"please enter from 1, 2, and 3 only: "<<endl;

goto back\_del;

}

}

else if(k==3)

{

cout<<endl<<"enter which number's location you want to find"<<endl;

cin>>num\_search;

ser\_ching=searchn(start, num\_search);

cout<<"required data is at "<<ser\_ching<<"th loction in linked list";

}

else

{

cout<<"please enter only 1 or 2: "<<endl;

goto again;

}

}

**Using doubly linked list:**

#include<iostream> using namespace std;

struct node

{

int data;

struct node \*prev;

struct node \*next;

};

class doubly\_linked\_list

{

public:struct node \*head;

struct node \*tail;

doubly\_linked\_list ()

{

head = NULL;

tail = NULL;

} void append (int n)

{

struct node \*temp = new node;

temp->data = n;

temp->next = NULL;

temp->prev = NULL;

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

temp->prev = tail;

tail->next = temp;

tail = temp;

}

}

void insert\_after (int n, int e)

{

struct node \*temp = head;

while (temp != NULL)

{

if (temp->data == e)

break;

temp = temp->next;

}

cout << "Success\n";

if (temp == NULL)

{

cout << "Error : Entered node not found\n";

return;

}

else if (temp->next == NULL)

{

struct node \*new\_node = new node;

new\_node->next = NULL;

new\_node->prev = temp;

temp->next = new\_node;

new\_node->data = n;

tail = new\_node;

}

else

{

struct node \*new\_node = new node;

struct node \*temp2 = temp->next;

new\_node->next = temp2;

new\_node->prev = temp;

temp2->prev = new\_node;

temp->next = new\_node;

new\_node->data = n;

}

} void find\_node\_loc (int n)

{

struct node \*temp;

temp = head;

int i = 1;

while (temp != NULL)

{

if (temp->data == n)

{

cout << "node found at position : " << i;

return;

}

temp = temp->next;

i++;

}

cout << "Node not found\n";

}

void delete\_node (int n)

{

struct node \*temp = head;

if (head->data == n)

{

head = head->next;

delete (temp);

return;

}

while (temp != NULL)

{

if (temp->data == n)

{

break;

}

temp = temp->next;

}

if (temp == NULL)

{

cout << "Node not found\n";

return;

}

else if (temp != tail)

{

struct node \*temp2 = temp->prev;

struct node \*temp3 = temp->next;

temp2->next = temp3;

temp3->prev = temp2;

delete (temp);

}

else

{

tail = temp->prev;

tail->next = NULL;

delete (temp);

}

}

void print ()

{

struct node \*temp;

temp = head;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

void print\_rev ()

{

struct node \*temp;

temp = tail;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->prev;

}

cout << endl;

}

};

int

main ()

{

doubly\_linked\_list x;

x.append (1);

x.append (2);

x.append (3);

x.append (4);

x.append (5);

x.print ();

x.insert\_after (100, 5);

x.insert\_after (200, 1);

x.insert\_after (300, 2);

x.print ();

x.delete\_node (1);

x.delete\_node (100);

x.delete\_node (3);

x.print ();

x.find\_node\_loc (4);

}

**Q2**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

class linked\_list

{

public:struct node \*head = new node;

struct node \*tail = new node;

linked\_list ()

{

head = NULL;

tail = NULL;

} void append (int n)

{

struct node \*temp = new node;

temp->next = NULL;

temp->data = n;

if (head == NULL)

{

head = temp;

tail = temp;

tail->next = head;

}

else

{

tail->next = temp;

tail = temp;

tail->next = head;

}

}

void print ()

{

if (head == NULL)

{

cout << "Error : no element to print\n";

}

struct node \*temp = head;

cout << head->data << " ";

temp = temp->next;

while (temp != head)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << head->data << endl;

}

};

int

main ()

{

linked\_list x;

x.append (1);

x.append (2);

x.append (3);

x.append (4);

x.append (5);

x.print ();

}

**Q3**

**For doubly linked list:** //doubly linked list

#include<iostream> using namespace std; struct node{ int data; struct node \*prev; struct node \*next;}; class doubly\_linked\_list{public:struct node \*head; struct node \*tail; doubly\_linked\_list () { head = NULL; tail = NULL; } void append (int n) { struct node \*temp = new node; temp->data = n; temp->next = NULL; temp->prev = NULL; if (head == NULL) { head = temp; tail = temp; } else { temp->prev = tail; tail->next = temp; tail = temp; } } void find\_size () { struct node \*temp = head; int i = 0; while (temp != NULL) { i++; temp = temp->next; } cout << "Size of list = " << i << endl; } void print () { struct node \*temp; temp = head; while (temp != NULL) { cout << temp->data << " "; temp = temp->next; } cout << endl; }};intmain (){ doubly\_linked\_list x; x.append (1); x.append (2); x.append (3); x.append (4); x.append (5); cout << "List is "<<endl; x.print (); x.find\_size ();}

**For circular linked list:**

#include<iostream> using namespace std;

struct node

{

int data;

struct node \*next;

};

class linked\_list

{

public:struct node \*head = new node;

struct node \*tail = new node;

linked\_list ()

{

head = NULL;

tail = NULL;

} void append (int n)

{

struct node \*temp = new node;

temp->next = NULL;

temp->data = n;

if (head == NULL)

{

head = temp;

tail = temp;

tail->next = head;

}

else

{

tail->next = temp;

tail = temp;

tail->next = head;

}

}

void find\_size ()

{

struct node \*temp = head;

int i = 1;

while (temp != tail)

{

i++;

temp = temp->next;

}

cout << "Size of list = " << i << endl;

}

void print ()

{

if (head == NULL)

{

cout << "Error : no element to print\n";

}

struct node \*temp = head;

cout << head->data << " ";

temp = temp->next;

while (temp != head)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

int

main ()

{

linked\_list x;

x.append (1);

x.append (2);

x.append (3);

x.append (4);

x.append (5);

x.print ();

x.find\_size ();

}

**Q4**

#include<iostream> using namespace std;

struct node

{

int data;

struct node \*prev;

struct node \*next;

};

class doubly\_linked\_list

{

public:struct node \*head;

struct node \*tail;

doubly\_linked\_list ()

{

head = NULL;

tail = NULL;

} void append (int n)

{

struct node \*temp = new node;

temp->data = n;

temp->next = NULL;

temp->prev = NULL;

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

temp->prev = tail;

tail->next = temp;

tail = temp;

}

}

void is\_palindrome ()

{

struct node \*forw = head;

struct node \*rev = tail;

while (forw != NULL)

{

if (forw->data != rev->data)

{

cout << "The list is not a palindrome\n";

return;

}

forw = forw->next;

rev = rev->prev;

}

cout << "The list is a palindrome\n";

}

void print ()

{

struct node \*temp;

temp = head;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

int

main ()

{

doubly\_linked\_list pal;

doubly\_linked\_list not\_pal;

for (int i = 0; i < 5; i++)

not\_pal.append (i);

not\_pal.print ();

not\_pal.is\_palindrome ();

for (int i = 0; i < 3; i++)

pal.append (i);

for (int i = 2; i >= 0; i--)

pal.append (i);

pal.print ();

pal.is\_palindrome ();

}

**Q5**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

class linked\_list

{

public:

struct node \*head = new node;

struct node \*tail = new node;

linked\_list ()

{

head = NULL;

tail = NULL;

}

void append\_circle (int n)

{

struct node \*temp = new node;

temp->next = NULL;

temp->data = n;

if (head == NULL)

{

head = temp;

tail = temp;

tail->next = head;

}

else

{

tail->next = temp;

tail = temp;

tail->next = head;

}

}

void append\_single (int n)

{

struct node \*temp = new node;

temp->next = NULL;

temp->data = n;

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = temp;

}

}

};

void

check\_circular (linked\_list x)

{

struct node \*temp = x.head->next;

while (temp != NULL)

{

if (temp == x.head)

{

cout << "List is circular\n";

return;

}

temp = temp->next;

}

cout << "The list is not circular\n";

}

int

main ()

{

linked\_list circle;

circle.append\_circle (1);

circle.append\_circle (2);

circle.append\_circle (3);

circle.append\_circle (4);

linked\_list single;

single.append\_single (10);

single.append\_single (20);

single.append\_single (30);

single.append\_single (40);

check\_circular (circle);

check\_circular (single);

}