**POST LAB**

#include <iostream>

using namespace std;

template <typename T>

class Node{

public:

Node()

{ data=0;}

// constructor

Node(T element)

{data=element;

next=NULL;}

//sets the KeyType data in the Node

void setData(T pVal)

{data=pVal;}

// returns the KeyType data in the Node

T getData()

{return data;}

// returns the link to the next node

Node\* GetNext()

{return next;}

// sets the link to the next node

void SetNext(Node \*x)

{next=x;}

T data;

Node \*next; };

template <typename T>

class SCList {

public:

// constructor of the Singly Circular Linked List

SCList()

{Node <T>\*pNew;

pNew= new Node<T>;}

//insert a new node in an empty list

void insertempty( Node<T>\*last, Node <T>\*new\_data)

{

// if last is not null then list is not empty, so return

if (last != NULL)

cout<<last;

// allocate memory for node

Node<T> \*temp = new Node;

// Assign the data.

temp -> data = new\_data;

last = temp;

// Create the link.

last->next = last;

cout<< last;

}

//insert new node at the beginning of the list

void insertAtBegin( Node<T> \*last, Node<T>\* new\_data)

{

//if list is empty then add the node by calling insertInEmpty

if (last == NULL)

cout<< insertInEmpty(last, new\_data);

//else create a new node

Node<T> \*temp = new Node;

//set new data to node

temp -> data= new\_data;

temp -> next = last -> next;

last -> next = temp;

cout<< last;

}

//insert new node at the end of the list

int insertAtEnd( Node<T> \*last, Node<T>\* new\_data)

{

//if list is empty then add the node by calling insertInEmpty

if (last == NULL)

return insertInEmpty(last, new\_data);

//else create a new node

Node<T> \*temp = new Node;

//assign data to new node

temp -> data = new\_data;

temp -> next= last -> next;

last -> next = temp;

last = temp;

return last;

}

void insertAfter( Node<T> \*last, Node<T>\*new\_data, Node<T> after\_item)

{

//return null if list is empty

if (last == NULL)

cout<< NULL;

Node<T> \*temp, \*p;

p = last -> next;

do

{

if (p ->data== after\_item)

{

temp = new Node;

temp -> data = new\_data;

temp -> next = p -> next;

p -> next = temp;

if (p == last)

last = temp;

cout<< last;

}

p = p -> next;

} while(p != last -> next);

cout << "The node with data "<<after\_item << " is not present in the list." << endl;

cout<< last;

}

//traverse the circular linked list

void traverseList(Node<T> \*last) {

Node<T> \*p;

// If list is empty, return.

if (last == NULL) {

cout << "Circular linked List is empty." << endl;

return;

}

p = last -> next; // Point to the first Node in the list.

// Traverse the list starting from first node until first node is visited again

do {

cout << p -> data << "==>";

p = p -> next;

} while(p != last->next);

if(p == last->next)

cout<<p->data;

cout<<"\n\n";

}

//delete the node from the list

void deleteNode(Node<T>\*\* head, int key)

{

// If linked list is empty retun

if (\*head == NULL)

return;

// If the list contains only a single node,delete that node; list is empty

if((\*head)->data==key && (\*head)->next==\*head) {

free(\*head);

\*head=NULL;

}

Node<T> \*last=\*head,\*d;

// If key is the head

if((\*head)->data==key) {

while(last->next!=\*head) // Find the last node of the list

last=last->next;

// point last node to next of head or second node of the list

last->next=(\*head)->next;

free(\*head);

\*head=last->next;

}

// end of list is reached or node to be deleted not there in the list

while(last->next!=\*head&&last->next->data!=key) {

last=last->next;

}

// node to be deleted is found, so free the memory and display the list

if(last->next->data==key) {

d=last->next;

last->next=d->next;

cout<<"The node with data "<<key<<" deleted from the list"<<endl;

free(d);

cout<<endl;

cout<<"Circular linked list after deleting "<<key<<" is as follows:"<<endl;

traverseList(last);

}

else

cout<<"The node with data "<< key << " not found in the list"<<endl;

}

private:

Node<T> \*last ;

};

int main()

{Node<int> \*a, \*b, \*c, \*d, \*e,\*f;

a = new Node<int>(1);

b = new Node<int>(2);

c = new Node<int>(3);

d = new Node<int>(4);

e = new Node<int>(5);

f = new Node<int>(6);

SCList<int>\*last;;

last-> insertempty(0, a);

last -> insertAtBegin(a, b);

last ->insertAtBegin(b, c);

last -> insertAtEnd(c, d);

last -> insertAtEnd(d, e);

last ->insertAfter(e, f,40 );

cout<<"The circular linked list created is as follows:"<<endl;

traverseList(last);

deleteNode(&last,10);

return 0;

}

