Lab 08 Data Structures

Date: 23 Sept 2024

Q1) Write a menu driven program to perform the following operations on linked list.

a) Insert an element in the beginning of the list

b) Insert an element at the end of the list

c) Insert an element before another element in the existing list

d) Insert an element after another element in the existing list

e) Delete a given element from the list

f) Print the list

**CODE:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = nullptr;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

cout << "Inserted " << value << " at the beginning.\n";

}

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

cout << "Inserted " << value << " at the end.\n";

return;

}

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

cout << "Inserted " << value << " at the end.\n";

}

void insertBefore(int existingValue, int newValue) {

if (!head) {

cout << "List is empty.\n";

return;

}

if (head->data == existingValue) {

insertAtBeginning(newValue);

return;

}

Node\* temp = head;

while (temp->next && temp->next->data != existingValue) {

temp = temp->next;

}

if (temp->next == nullptr) {

cout << existingValue << " not found in the list.\n";

} else {

Node\* newNode = new Node(newValue);

newNode->next = temp->next;

temp->next = newNode;

cout << "Inserted " << newValue << " before " << existingValue << ".\n";

}

}

void insertAfter(int existingValue, int newValue) {

Node\* temp = head;

while (temp && temp->data != existingValue) {

temp = temp->next;

}

if (!temp) {

cout << existingValue << " not found in the list.\n";

return;

}

Node\* newNode = new Node(newValue);

newNode->next = temp->next;

temp->next = newNode;

cout << "Inserted " << newValue << " after " << existingValue << ".\n";

}

void deleteElement(int value) {

if (!head) {

cout << "List is empty.\n";

return;

}

if (head->data == value) {

Node\* temp = head;

head = head->next;

delete temp;

cout << "Deleted " << value << " from the list.\n";

return;

}

Node\* temp = head;

while (temp->next && temp->next->data != value) {

temp = temp->next;

}

if (!temp->next) {

cout << value << " not found in the list.\n";

} else {

Node\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

cout << "Deleted " << value << " from the list.\n";

}

}

void printList() {

if (!head) {

cout << "List is empty.\n";

return;

}

Node\* temp = head;

cout << "List elements: ";

while (temp) {

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

temp = temp->next;

}

cout << endl;

}

~LinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

LinkedList list;

int choice, value, existingValue;

do {

cout << "\nMenu:\n";

cout << "1. Insert at beginning\n";

cout << "2. Insert at end\n";

cout << "3. Insert before a given element\n";

cout << "4. Insert after a given element\n";

cout << "5. Delete a given element\n";

cout << "6. Print the list\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert at beginning: ";

cin >> value;

list.insertAtBeginning(value);

break;

case 2:

cout << "Enter value to insert at end: ";

cin >> value;

list.insertAtEnd(value);

break;

case 3:

cout << "Enter existing value to insert before: ";

cin >> existingValue;

cout << "Enter new value to insert: ";

cin >> value;

list.insertBefore(existingValue, value);

break;

case 4:

cout << "Enter existing value to insert after: ";

cin >> existingValue;

cout << "Enter new value to insert: ";

cin >> value;

list.insertAfter(existingValue, value);

break;

case 5:

cout << "Enter value to delete: ";

cin >> value;

list.deleteElement(value);

break;

case 6:

list.printList();

break;

case 7:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice\n";

}

} while (choice != 7);

return 0;

}

**OUTPUT:**

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 1

Enter value to insert at beginning: 20

Inserted 20 at the beginning.

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 2

Enter value to insert at end: 30

Inserted 30 at the end.

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 4

Enter existing value to insert after: 30

Enter new value to insert: 45

Inserted 45 after 30.

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 5

Enter value to delete: 30

Deleted 30 from the list.

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 6

List elements: 20 45

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before a given element

4. Insert after a given element

5. Delete a given element

6. Print the list

7. Exit

Enter your choice: 7

Exiting...

Q2) Implement Stack and Queue using linked lists

**Stacks**

**CODE:**

#include<iostream>

using namespace std;

#define MAX\_SIZE 5

class list{

struct node{

int data;

node \* link = NULL;

}\*p;

public:

int c=0;

list();

void push(int num);

void pop();

void peek();

};

list::list()

{

p = NULL;

}

void list::push(int num)

{

node \*temp;

if (p==NULL)

{

temp=new node;

temp->data=num;

p=temp;

c=c+1;

cout<<"Pushed element is =" << num <<"\n";

return;

}

else if (c>= MAX\_SIZE)

{

cout<<"Stack overflow\n";

}

else

{

temp=new node;

temp->data=num;

temp->link=p;

p=temp;

c=c+1;

cout<<"Pushed element is =" << num <<"\n";

}

}

void list::pop()

{

node \*temp;

if(p==NULL)

{

cout<<"Stack is empty\n";

return;

}

else

{

temp=p;

p=temp->link;

cout<<"popped element is="<< temp->data <<"\n";

delete temp;

return;

}

}

int main()

{

list l;

l.push(5);

l.push(6);

l.push(3);

l.push(4);

l.push(8);

l.push(9);

l.pop();

l.pop();

l.pop();

l.pop();

l.pop();

l.pop();

return 0;

}

**OUTPUT:**

Pushed element is =5

Pushed element is =6

Pushed element is =3

Pushed element is =4

Pushed element is =8

Stack overflow

popped element is=8

popped element is=4

popped element is=3

popped element is=6

popped element is=5

Stack is empty

**Queues**

**CODE:**

#include<iostream>

using namespace std;

#define MAX 5

class list{

struct node

{

int data;

node\* link;

} \*p=NULL;

node\* rear=NULL;

int c=0;

public:

void enqueue(int num);

void dequeue();

void peek();

};

void list::peek()

{

cout<< "Element of the queue are : \n";

node \* temp;

temp=p;

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

{

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

temp=temp->link;

}

}

void list::enqueue(int num)

{

node\* temp;

if(c>=5)

{

cout<<"queue is full\n";

return;

}

else if(p==NULL)

{

temp=new node;

temp->data=num;

temp->link=NULL;

p=temp;

c=c+1;

rear=temp;

cout<< "Inserted the first element at the rear of queue="<<temp->data<<"\n";

return;

}

else

{

temp=new node;

temp->data=num;

temp->link=NULL;

rear->link=temp;

rear=temp;

cout<< "Inserted element at the rear of queue="<<temp->data<<"\n";

c=c+1;

return;

}

}

void list::dequeue()

{

node\* temp;

if (c==0)

{

cout<<"queue is empty\n";

return;

}

else

{

temp=p;

cout<<"The removed element is="<<temp->data<<"\n";

p=p->link;

c=c-1;

delete temp;

return;

}

}

int main()

{

list l;

l.enqueue(5);

l.enqueue(4);

l.enqueue(3);

l.enqueue(2);

l.enqueue(1);

l.enqueue(0);

l.peek();

l.dequeue();

l.dequeue();

l.dequeue();

l.dequeue();

l.dequeue();

l.dequeue();

return 0;

}

**OUTPUT:**

Inserted the first element at the rear of queue=5

Inserted element at the rear of queue=4

Inserted element at the rear of queue=3

Inserted element at the rear of queue=2

Inserted element at the rear of queue=1

queue is full

Element of the queue are :

5

4

3

2

1

The removed element is=5

The removed element is=4

The removed element is=3

The removed element is=2

The removed element is=1

queue is empty