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

a) Insert an element at the rear end of the list

b) Delete an element from the rear end of the list

c) Insert an element at a given position of the list

d) Delete an element from a given position of the list

e) Insert an element after another element

f) Insert an element before another element

g) Print the list

**CODE:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int value) {

data = value;

next = nullptr;

prev = nullptr;

}

};

class DoublyLinkedList {

private:

Node\* head;

public:

DoublyLinkedList() : head(nullptr) {}

void insertAtRear(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

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

return;

}

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

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

}

void deleteFromRear() {

if (!head) {

cout << "List is empty. Cannot delete from rear.\n";

return;

}

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

if (temp == head) {

delete head;

head = nullptr;

} else {

temp->prev->next = nullptr;

delete temp;

}

cout << "Deleted element from the rear end.\n";

}

void insertAtPosition(int value, int position) {

if (position < 0) {

cout << "Invalid position.\n";

return;

}

Node\* newNode = new Node(value);

if (position == 0) {

newNode->next = head;

if (head) {

head->prev = newNode;

}

head = newNode;

cout << "Inserted " << value << " at position " << position << ".\n";

return;

}

Node\* temp = head;

for (int i = 0; i < position - 1 && temp; ++i) {

temp = temp->next;

}

if (!temp) {

cout << "Position is out of bounds.\n";

delete newNode;

return;

}

newNode->next = temp->next;

newNode->prev = temp;

if (temp->next) {

temp->next->prev = newNode;

}

temp->next = newNode;

cout << "Inserted " << value << " at position " << position << ".\n";

}

void deleteFromPosition(int position) {

if (position < 0 || !head) {

cout << "Invalid position or list is empty.\n";

return;

}

Node\* temp = head;

if (position == 0) {

head = temp->next;

if (head) {

head->prev = nullptr;

}

delete temp;

cout << "Deleted element from position " << position << ".\n";

return;

}

for (int i = 0; i < position && temp; ++i) {

temp = temp->next;

}

if (!temp) {

cout << "Position is out of bounds.\n";

return;

}

if (temp->next) {

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

}

if (temp->prev) {

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

}

delete temp;

cout << "Deleted element from position " << position << ".\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;

newNode->prev = temp;

if (temp->next) {

temp->next->prev = newNode;

}

temp->next = newNode;

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

}

void insertBefore(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;

newNode->prev = temp->prev;

if (temp->prev) {

temp->prev->next = newNode;

} else {

head = newNode;

}

temp->prev = newNode;

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

}

~DoublyLinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

DoublyLinkedList list;

int choice, value, position, existingValue;

do {

cout << "\nMenu:\n";

cout << "1. Insert at rear end\n";

cout << "2. Delete from rear end\n";

cout << "3. Insert at a given position\n";

cout << "4. Delete from a given position\n";

cout << "5. Insert after another element\n";

cout << "6. Insert before another element\n";

cout << "7. Print the list\n";

cout << "8. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

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

cin >> value;

list.insertAtRear(value);

break;

case 2:

list.deleteFromRear();

break;

case 3:

cout << "Enter value to insert: ";

cin >> value;

cout << "Enter position to insert at: ";

cin >> position;

list.insertAtPosition(value, position);

break;

case 4:

cout << "Enter position to delete from: ";

cin >> position;

list.deleteFromPosition(position);

break;

case 5:

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

cin >> existingValue;

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

cin >> value;

list.insertAfter(existingValue, value);

break;

case 6:

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

cin >> existingValue;

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

cin >> value;

list.insertBefore(existingValue, value);

break;

case 7:

list.printList();

break;

case 8:

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

break;

default:

cout << "Invalid choice. Please try again.\n";

}

} while (choice != 8);

return 0;

}

**OUTPUT:**

Menu:

1. Insert at rear end

2. Delete from rear end

3. Insert at a given position

4. Delete from a given position

5. Insert after another element

6. Insert before another element

7. Print the list

8. Exit

Enter your choice: 1

Enter value to insert at rear end: 5

Inserted 5 at the rear end.

Menu:

1. Insert at rear end

2. Delete from rear end

3. Insert at a given position

4. Delete from a given position

5. Insert after another element

6. Insert before another element

7. Print the list

8. Exit

Enter your choice: 3

Enter value to insert: 4

Enter position to insert at: 1

Inserted 4 at position 1.

Menu:

1. Insert at rear end

2. Delete from rear end

3. Insert at a given position

4. Delete from a given position

5. Insert after another element

6. Insert before another element

7. Print the list

8. Exit

Enter your choice: 6

Enter existing value to insert before: 4

Enter new value to insert: 3

Inserted 3 before 4.

Menu:

1. Insert at rear end

2. Delete from rear end

3. Insert at a given position

4. Delete from a given position

5. Insert after another element

6. Insert before another element

7. Print the list

8. Exit

Enter your choice: 7

List elements: 5 3 4

Menu:

1. Insert at rear end

2. Delete from rear end

3. Insert at a given position

4. Delete from a given position

5. Insert after another element

6. Insert before another element

7. Print the list

8. Exit

Enter your choice: 8

Exiting...

Q2) Write a program to add two polynomials using doubly linked list.

**CODE:**

#include <iostream>

using namespace std;

class Node {

public:

int coefficient;

int exponent;

Node\* next;

Node\* prev;

Node(int coeff, int exp) : coefficient(coeff), exponent(exp), next(nullptr), prev(nullptr) {}

};

class Polynomial {

private:

Node\* head;

public:

Polynomial() : head(nullptr) {}

void insert(int coefficient, int exponent) {

Node\* newNode = new Node(coefficient, exponent);

if (!head || head->exponent < exponent) {

newNode->next = head;

if (head) {

head->prev = newNode;

}

head = newNode;

return;

}

Node\* temp = head;

while (temp->next && temp->next->exponent >= exponent) {

temp = temp->next;

}

if (temp->exponent == exponent) {

temp->coefficient += coefficient;

delete newNode;

} else {

newNode->next = temp->next;

newNode->prev = temp;

temp->next = newNode;

if (newNode->next) {

newNode->next->prev = newNode;

}

}

}

Polynomial add(const Polynomial& other) {

Polynomial result;

Node\* temp1 = head;

Node\* temp2 = other.head;

while (temp1 || temp2) {

if (temp1 && (!temp2 || temp1->exponent > temp2->exponent)) {

result.insert(temp1->coefficient, temp1->exponent);

temp1 = temp1->next;

} else if (temp2 && (!temp1 || temp2->exponent > temp1->exponent)) {

result.insert(temp2->coefficient, temp2->exponent);

temp2 = temp2->next;

} else { // temp1->exponent == temp2->exponent

result.insert(temp1->coefficient + temp2->coefficient, temp1->exponent);

temp1 = temp1->next;

temp2 = temp2->next;

}

}

return result;

}

void display() const {

if (!head) {

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

return;

}

Node\* temp = head;

while (temp) {

cout << temp->coefficient << "x^" << temp->exponent;

if (temp->next) {

cout << " + ";

}

temp = temp->next;

}

cout << endl;

}

~Polynomial() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

Polynomial poly1, poly2;

poly1.insert(4, 3);

poly1.insert(2, 1);

poly1.insert(3, 0);

poly2.insert(5, 2);

poly2.insert(1, 1);

poly2.insert(2, 0);

cout << "Polynomial 1: ";

poly1.display();

cout << "Polynomial 2: ";

poly2.display();

Polynomial result = poly1.add(poly2);

cout << "Result of addition: ";

result.display();

return 0;

}

**OUTPUT:**

Polynomial 1: 4x^3 + 2x^1 + 3x^0

Polynomial 2: 5x^2 + 1x^1 + 2x^0

Result of addition: 4x^3 + 5x^2 + 3x^1 + 5x^0