PIYUSH KUMAR MISHRA

230957212

ROLL NO: -70

Week 5:

1. Implementation of Stack using arrays

```
#include <iostream>

using namespace std;


class Stack {

private:

  int arr[100];

  int top;


public:

  Stack() {

    top = -1;

  }


  void push(int x) {

    if (top == 99) {

      cout << "Error: Stack Overflow" << endl;
```

```
        return;
    }
    arr[++top] = x;
    cout << x << " pushed into the stack" << endl;
}

int pop() {
    if (top == -1) {
        cout << "Error: Stack Underflow" << endl;
        return -1;
    }
    int popped = arr[top];
    top--;
    return popped;
}

int peek() {
    if (top == -1) {
        cout << "Error: Stack Underflow" << endl;
        return -1;
    }
    return arr[top];
}
```

```
    bool isEmpty() {
        return top == -1;
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << s.pop() << " popped from the stack" << endl;
    cout << "Top element is " << s.peek() << endl;
    cout << "Stack empty? " << s.isEmpty() << endl;
    return 0;
}
```

OUTPUT:

```
10 pushed into the stack
20 pushed into the stack
30 pushed into the stack
30 popped from the stack
Top element is 20
Stack empty? 0
```

2. To check if the given parenthesized expression has properly matching open and closing parenthesis

```
#include <iostream>

#define MAX 100

using namespace std;

class Stack {
private:
   char arr[MAX];
   int top;

public:
   Stack() { top = -1; }

   void push(char c) {
      if (top >= MAX - 1) {
         cout << "Stack overflow" << endl;
         return;
      }
      arr[++top] = c;
   }

   char pop() {
      if (top < 0) {
         cout << "Stack underflow" << endl;
```

```
        return '\0';
    }
    return arr[top--];
  }

  bool isEmpty() {
    return top == -1;
  }
};

bool isMatchingPair(char opening, char closing) {
  return (opening == '(' && closing == ')') ||
      (opening == '{' && closing == '}') ||
      (opening == '[' && closing == ']');
}

bool checkParentheses(const char* expression) {
  Stack stack;

  for (int i = 0; expression[i] != '\0'; i++) {
    char ch = expression[i];
    if (ch == '(' || ch == '{' || ch == '[') {
      stack.push(ch);
    } else if (ch == ')' || ch == '}' || ch == ']') {
      if (stack.isEmpty()) {
```

```
            return false;
        }
        char lastOpening = stack.pop();
        if (!isMatchingPair(lastOpening, ch)) {
            return false;
        }
      }
    }

    return stack.isEmpty();
}

int main() {
    char expression[MAX];

    cout << "Enter a parenthesized expression: ";
    cin.getline(expression, MAX);

    if (checkParentheses(expression)) {
        cout << "The parentheses are matched." << endl;
    } else {
        cout << "The parentheses are not matched." << endl;
    }

    return 0;
```

```
}
```

OUTPUT:

```
Enter a parenthesized expression: [{]}

The parentheses are not matched.
```

3. To check a given string is palindrome or not using stack

```
#include <iostream>


#define MAX 100


using namespace std;


class Stack {

private:

   char arr[MAX];

   int top;


public:

   Stack() { top = -1; }


   void push(char c) {

      if (top >= MAX - 1) {

         cout << "Stack overflow" << endl;

         return;

      }
```

```
        arr[++top] = c;
    }

    char pop() {
        if (top < 0) {
            cout << "Stack underflow" << endl;
            return '\0';
        }
        return arr[top--];
    }

    bool isEmpty() {
        return top == -1;
    }
};

bool isPalindrome(const char* str) {
    Stack stack;
    int length = 0;

    for (int i = 0; str[i] != '\0'; i++) {
        length++;
    }

    for (int i = 0; i < length / 2; i++) {
```

```
        stack.push(str[i]);
    }

    int start = (length % 2 == 0) ? length / 2 : (length / 2) + 1;

    for (int i = start; i < length; i++) {
        if (stack.pop() != str[i]) {
            return false;
        }
    }

    return true;
}

int main() {
    char str[MAX];

    cout << "Enter a string: ";
    cin.getline(str, MAX);

    if (isPalindrome(str)) {
        cout << "The string is a palindrome." << endl;
    } else {
        cout << "The string is not a palindrome." << endl;
    }
```

```
    return 0;
}
```

OUTPUT:

Enter a string: ABCBA

The string is a palindrome.

Week 6:

Write a C++ program for the following tasks:

1. **Conversion of infix expression to postfix and prefix forms**

```
#include <iostream>

#define MAX 100

using namespace std;

class Stack {
private:
    char arr[MAX];
    int top;

public:
    Stack() { top = -1; }

    void push(char c) {
        if (top >= MAX - 1) {
            cout << "Stack overflow" << endl;
```

```
            return;
        }
        arr[++top] = c;
    }

    char pop() {
        if (top < 0) {
            cout << "Stack underflow" << endl;
            return '\0';
        }
        return arr[top--];
    }

    char peek() {
        if (top < 0) {
            return '\0';
        }
        return arr[top];
    }

    bool isEmpty() {
        return top == -1;
    }
};

bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}
```

```
int precedence(char c) {
   if (c == '^')
      return 3;
   if (c == '*' || c == '/')
      return 2;
   if (c == '+' || c == '-')
      return 1;
   return -1;
}

void reverseString(char* str, int length) {
   int start = 0, end = length - 1;
   while (start < end) {
      char temp = str[start];
      str[start] = str[end];
      str[end] = temp;
      start++;
      end--;
   }
}

void infixToPostfix(char* infix, char* postfix) {
   Stack st;
   int j = 0;

   for (int i = 0; infix[i] != '\0'; i++) {
      char c = infix[i];

      if (isalnum(c)) {
```

```
            postfix[j++] = c;
        } else if (c == '(') {
            st.push(c);
        } else if (c == ')') {
            while (!st.isEmpty() && st.peek() != '(') {
                postfix[j++] = st.pop();
            }
            st.pop();
        } else if (isOperator(c)) {
            while (!st.isEmpty() && precedence(st.peek()) >= precedence(c)) {
                postfix[j++] = st.pop();
            }
            st.push(c);
        }
    }

    while (!st.isEmpty()) {
        postfix[j++] = st.pop();
    }

    postfix[j] = '\0';
}

void infixToPrefix(char* infix, char* prefix) {
    int length = 0;

    for (int i = 0; infix[i] != '\0'; i++) {
        length++;
    }
```

```
    reverseString(infix, length);

    for (int i = 0; i < length; i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
        } else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }

    char postfix[MAX];
    infixToPostfix(infix, postfix);
    reverseString(postfix, length);

    for (int i = 0; i < length; i++) {
        prefix[i] = postfix[i];
    }

    prefix[length] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];

    cout << "Enter infix expression: ";
    cin >> infix;

    infixToPostfix(infix, postfix);
```

```
    infixToPrefix(infix, prefix);

    cout << "Postfix Expression: " << postfix << endl;
    cout << "Prefix Expression: " << prefix << endl;

    return 0;
}
```

OUTPUT:

Enter infix expression: A+B-C/D

Postfix Expression: AB+CD/-

Prefix Expression: +A-B/CD

2. Evaluation of postfix and prefix expressions

```
#include <iostream>
#include <cmath>

#define MAX 100

using namespace std;

class Stack {
private:
    int arr[MAX];
    int top;

public:
```

```
    Stack() { top = -1; }

    void push(int value) {
        if (top >= MAX - 1) {
            cout << "Stack overflow" << endl;
            return;
        }
        arr[++top] = value;
    }

    int pop() {
        if (top < 0) {
            cout << "Stack underflow" << endl;
            return 0;
        }
        return arr[top--];
    }

    bool isEmpty() {
        return top == -1;
    }
};

int evaluatePostfix(char* postfix) {
    Stack st;
```

```
    for (int i = 0; postfix[i] != '\0'; i++) {
        char c = postfix[i];

        if (isdigit(c)) {
            st.push(c - '0');
        } else {
            int operand2 = st.pop();
            int operand1 = st.pop();
            switch (c) {
                case '+': st.push(operand1 + operand2); break;
                case '-': st.push(operand1 - operand2); break;
                case '*': st.push(operand1 * operand2); break;
                case '/': st.push(operand1 / operand2); break;
                case '^': st.push(pow(operand1, operand2)); break;
            }
        }
    }

    return st.pop();
}

int evaluatePrefix(char* prefix) {
    Stack st;
    int length = 0;
```

```
    for (int i = 0; prefix[i] != '\0'; i++) {
        length++;
    }

    for (int i = length - 1; i >= 0; i--) {
        char c = prefix[i];

        if (isdigit(c)) {
            st.push(c - '0');
        } else {
            int operand1 = st.pop();
            int operand2 = st.pop();
            switch (c) {
                case '+': st.push(operand1 + operand2); break;
                case '-': st.push(operand1 - operand2); break;
                case '*': st.push(operand1 * operand2); break;
                case '/': st.push(operand1 / operand2); break;
                case '^': st.push(pow(operand1, operand2)); break;
            }
        }
    }

    return st.pop();
}
```

```
int main() {
    char postfix[MAX], prefix[MAX];

    cout << "Enter postfix expression: ";
    cin >> postfix;
    cout << "Postfix evaluation result: " << evaluatePostfix(postfix) << endl;

    cout << "Enter prefix expression: ";
    cin >> prefix;
    cout << "Prefix evaluation result: " << evaluatePrefix(prefix) << endl;

    return 0;
}
```

OUTPUT:

Enter postfix expression: 456*/

Postfix evaluation result: 0

Enter prefix expression: +/567

Prefix evaluation result: 7

Week 7

1. Implementation of Queue using arrays

```
#include<iostream>
```

```
using namespace std;
class Queue{
		int* arr;
		int size;
		int qfront;
		int rear;
	public:
	Queue(){
		size=1001;
		arr=new int[size];
	qfront=0;
		rear=0;
	}
	void enque(int ele){
		if(rear==size){
			cout<<"Can't Enque because Queue is full"<<endl;
		}
		else{
			arr[rear]=ele;
			rear++;
		}

	}
	int deque(){
		if(qfront==rear){
```

```
            return -1;
        }
        else{
            int ans =arr[qfront];
            arr[qfront]=-1;
            qfront++;
            if(qfront==rear){
                qfront=0;
                rear=0;
            }
            return ans;
        }
}
bool isEmpty(){
        if(qfront==rear){
            return true;
        }
        else{
            return false;
        }
}
void  front(){
        if( qfront!=rear){
            cout<<arr[ qfront]<<endl;
        }
```

```
                else{
                        cout<<"Queue is empty"<<endl;
                }
        }
        void display() {
    if (isEmpty()) {
      cout << "Queue is empty" << endl;
    } else {
      cout << "Queue elements: ";
      for (int i = qfront; i < rear; i++) {
        cout << arr[i] << " ";
      }
      cout << endl;
    }
  }
};

int main(){

        Queue q;
        q.enque(5);
        q.enque(4);
        q.enque(3);
        q.enque(2);
        q.enque(1);
```

```
    q.display();
    q.front();
    q.deque();
    q.front();
    q.deque();
    q.front();

}
```

OUTPUT:

```
Queue elements: 5 4 3 2 1
5
4
3
```

2. Implement a circular queue of Strings with functions insert, delete and display.

```
#include <iostream>
#include <string>
#define SIZE 5

using namespace std;

class CircularQueue {
private:
  string items[SIZE];
  int front;
  int rear;
```

```
public:
    CircularQueue() {
        front = -1;
        rear = -1;
    }

    bool isFull() {
        return (rear + 1) % SIZE == front;
    }

    bool isEmpty() {
        return front == -1;
    }

    void enQueue(const string& value) {
        if (isFull()) {
            cout << "Queue Overflow" << endl;
            return;
        }

        if (isEmpty()) {
            front = 0;
        }
```

```
    rear = (rear + 1) % SIZE;
    items[rear] = value;
    cout << "Inserted: " << value << endl;
}

string deQueue() {
    if (isEmpty()) {
        cout << "Queue Underflow" << endl;
        return "";
    }

    string deletedValue = items[front];

    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % SIZE;
    }

    return deletedValue;
}

void display() {
    if (isEmpty()) {
```

```
        cout << "Queue is empty" << endl;
        return;
    }

    cout << "Circular Queue: ";
    int i = front;

    while (true) {
        cout << items[i] << " ";
        if (i == rear) break;
        i = (i + 1) % SIZE;
    }

    cout << endl;
  }
};

int main() {
  CircularQueue q;

  q.enQueue("Hello");
  q.enQueue("World");
  q.display();

  string deletedValue = q.deQueue();
```

```
    if (!deletedValue.empty()) {
        cout << "Deleted: " << deletedValue << endl;
    }

    q.display();

    q.enQueue("Circular");
    q.enQueue("Queue");
    q.enQueue("Example");

    q.display();

    return 0;
}
```

OUTPUT:

Inserted: Hello

Inserted: World

Circular Queue: Hello World

Deleted: Hello

Circular Queue: World

Inserted: Circular

Inserted: Queue

Inserted: Example

Circular Queue: World Circular Queue Example

3. Write a program to implement the circular queue using arrays

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

class CircularQueue {
  int front, rear, size;
  int* queue;

public:
  CircularQueue(int s) {
    front = rear = -1;
    size = s;
    queue = new int[s];
  }

  void insert(int data) {
    if ((rear + 1) % size == front) {
      cout << "Queue is full" << endl;
    } else if (front == -1) {
      front = rear = 0;
      queue[rear] = data;
    } else {
      rear = (rear + 1) % size;
      queue[rear] = data;
    }
  }
```

```
int deleteElement() {
   if (front == -1) {
      cout << "Queue is empty" << endl;
      return -1;
   } else if (front == rear) {
      int temp = queue[front];
      front = rear = -1;
      return temp;
   } else {
      int temp = queue[front];
      front = (front + 1) % size;
      return temp;
   }
}

void display() {
   if (front == -1) {
      cout << "Queue is empty" << endl;
   } else if (rear >= front) {
      cout << "Queue elements: ";
      for (int i = front; i <= rear; i++)
         cout << queue[i] << " ";
      cout << endl;
   } else {
```

```
        cout << "Queue elements: ";
        for (int i = front; i < size; i++)
            cout << queue[i] << " ";
        for (int i = 0; i <= rear; i++)
            cout << queue[i] << " ";
        cout << endl;
      }
    }

    ~CircularQueue() {
        delete[] queue;
    }
};

int main() {
    CircularQueue cq(5);
    cq.insert(1);
    cq.insert(2);
    cq.insert(3);
    cq.display();
    cout << "Deleted element: " << cq.deleteElement() << endl;
    cq.display();
    cq.insert(4);
    cq.insert(5);
    cq.insert(6);
```

```
    cq.display();
    return 0;
}
```

OUTPUT:

Queue elements: 1 2 3

Deleted element: 1

Queue elements: 2 3

Queue elements: 2 3 4 5 6

Week 8

1. 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

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

class LinkedList {
public:
    class Node {
    public:
        int data;
```

```
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

Node* head;

LinkedList() : head(nullptr) {}

  void insertAtBeginning(int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

void insertAtEnd(int value) {
    if (!head) {
        head = new Node(value);
        return;
    }

    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = new Node(value);
}
```

```
void insertBefore(int target, int value) {
    if (!head) return;

    if (head->data == target) {
        insertAtBeginning(value);
        return;
    }

    Node* temp = head;
    while (temp->next && temp->next->data != target) temp = temp->next;

    if (!temp->next) {
        cout << "Element " << target << " not found." << endl;
        return;
    }

    Node* newNode = new Node(value);
    newNode->next = temp->next;
    temp->next = newNode;
}

void insertAfter(int target, int value) {
    Node* temp = head;
    while (temp && temp->data != target) temp = temp->next;
```

```
    if (!temp) {
        cout << "Element " << target << " not found." << endl;
        return;
    }

    Node* newNode = new Node(value);
    newNode->next = temp->next;
    temp->next = newNode;
}

void deleteElement(int value) {
    if (!head) return;

    if (head->data == value) {
        Node* toDelete = head;
        head = head->next;
        delete toDelete;
        return;
    }

    Node* temp = head;
    while (temp->next && temp->next->data != value) temp = temp->next;

    if (!temp->next) {
        cout << "Element " << value << " not found." << endl;
```

```
            return;
        }

        Node* toDelete = temp->next;
        temp->next = toDelete->next;
        delete toDelete;
    }

    void printList() const {
        for (Node* temp = head; temp; temp = temp->next)
            cout << temp->data << " ";

        cout << (head ? "" : "List is empty.") << endl; // Print message if empty
    }

    ~LinkedList() {
        while (head) deleteElement(head->data);
    }
};

int main() {
    LinkedList list;

    int choice, value, target;
```

```
  do {
    cout << "\nMenu:\n1. Insert at beginning\n2. Insert at end\n3. Insert before an element\n4. Insert after an element\n5. Delete an element\n6. Print the list\n7. 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 target element and value to insert before: ";
        cin >> target >> value; list.insertBefore(target, value); break;

      case 4:
        cout << "Enter target element and value to insert after: ";
        cin >> target >> value; list.insertAfter(target, value); break;

      case 5:
        cout << "Enter value to delete: ";
```

```
            cin >> value; list.deleteElement(value); break;


        case 6:
            list.printList(); break;


        case 7:
            cout << "Exiting..." << endl; break;


        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
      }
   } while (choice != 7);


   return 0;
}
```

OUTPUT:

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before an element

4. Insert after an element

5. Delete an element

6. Print the list

7. Exit

Enter your choice: 1

Enter value to insert at beginning: 1

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before an element

4. Insert after an element

5. Delete an element

6. Print the list

7. Exit

Enter your choice: 1

Enter value to insert at beginning: 2

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before an element

4. Insert after an element

5. Delete an element

6. Print the list

7. Exit

Enter your choice: 6

2 1

Menu:

1. Insert at beginning

2. Insert at end

3. Insert before an element

4. Insert after an element

5. Delete an element

6. Print the list

7. Exit

Enter your choice: 7

Exiting...

2. Implement Stack and Queue using linked lists

```
#include <iostream>

using namespace std;


// Node class for Linked List

class Node {

public:

  int data;

  Node* next;


  Node(int value) {

    data = value;

    next = nullptr;

  }
```

```
};

// Stack implementation using Linked List
class Stack {
private:
  Node* top;

public:
  Stack() {
    top = nullptr;
  }

  void push(int value) {
    Node* newNode = new Node(value);
    newNode->next = top;
    top = newNode;
    cout << value << " pushed to stack\n";
  }

  int pop() {
    if (top == nullptr) {
      cout << "Stack is empty\n";
      return -1;
    }
    int poppedValue = top->data;
```

```
        Node* temp = top;
        top = top->next;
        delete temp;
        return poppedValue;
    }

    bool isEmpty() {
        return top == nullptr;
    }

    void display() {
        if (top == nullptr) {
            cout << "Stack is empty\n";
            return;
        }
        Node* temp = top;
        cout << "Stack: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

```
// Queue implementation using Linked List
class Queue {
private:
  Node* front;
  Node* rear;

public:
  Queue() {
    front = nullptr;
    rear = nullptr;
  }

  void enqueue(int value) {
    Node* newNode = new Node(value);
    if (rear == nullptr) {
      front = rear = newNode;
    } else {
      rear->next = newNode;
      rear = newNode;
    }
    cout << value << " enqueued to queue\n";
  }

  int dequeue() {
    if (front == nullptr) {
```

```
        cout << "Queue is empty\n";
        return -1;
    }
    int dequeuedValue = front->data;
    Node* temp = front;
    front = front->next;
    if (front == nullptr) {
        rear = nullptr;
    }
    delete temp;
    return dequeuedValue;
}

bool isEmpty() {
    return front == nullptr;
}

void display() {
    if (front == nullptr) {
        cout << "Queue is empty\n";
        return;
    }
    Node* temp = front;
    cout << "Queue: ";
    while (temp != nullptr) {
```

```
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Stack stack;
    Queue queue;

    // Stack operations
    cout << "Stack operations:\n";
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.display();

    cout << "Popped from stack: " << stack.pop() << endl;
    stack.display();

    // Queue operations
    cout << "\nQueue operations:\n";
    queue.enqueue(10);
    queue.enqueue(20);
```

```
    queue.enqueue(30);
    queue.display();

    cout << "Dequeued from queue: " << queue.dequeue() << endl;
    queue.display();

    return 0;
}
```

OUTPUT:

```
Stack operations:
10 pushed to stack
20 pushed to stack
30 pushed to stack
Stack: 30 20 10
Popped from stack: 30
Stack: 20 10

Queue operations:
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
Queue: 10 20 30
Dequeued from queue: 10
Queue: 20 30
```

Week-9

1. 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

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

class DoublyLinkedList {
private:
  struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
  };

  Node* head;

public:
```

```
DoublyLinkedList() : head(nullptr) {}

~DoublyLinkedList() {
   while (head) deleteFromEnd();
}

void insertAtEnd(int data) {
   Node* newNode = new Node(data);
   if (!head) {
      head = newNode;
      return;
   }
   Node* temp = head;
   while (temp->next) temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
}

void deleteFromEnd() {
   if (!head) { cout << "List is empty" << endl; return; }
   if (!head->next) { delete head; head = nullptr; return; }
   Node* temp = head;
   while (temp->next) temp = temp->next;
   temp->prev->next = nullptr;
   delete temp;
```

```
}

void insertAtPosition(int data, int position) {
   if (position < 1) { cout << "Position out of range" << endl; return; }

   Node* newNode = new Node(data);
   if (position == 1) {
      newNode->next = head;
      if (head) head->prev = newNode;
      head = newNode;
      return;
   }

   Node* temp = head;
   for (int i = 1; temp && i < position - 1; i++) temp = temp->next;

   if (!temp) { cout << "Position out of range" << endl; delete newNode; return; }

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

   if (temp->next) temp->next->prev = newNode;

   temp->next = newNode;
}
```

```
void deleteFromPosition(int position) {
    if (!head || position < 1) { cout << "Position out of range" << endl; return; }

    Node* temp = head;

    if (position == 1) {
        head = temp->next;
        if (head) head->prev = nullptr;
        delete temp;
        return;
    }

    for (int i = 1; temp && i < position; i++) temp = temp->next;

    if (!temp) { cout << "Position out of range" << endl; return; }

    if (temp->prev) temp->prev->next = temp->next;
    if (temp->next) temp->next->prev = temp->prev;

    delete temp;
}

void insertAfter(int target, int data) {
    Node* temp = head;
```

```
    while (temp && temp->data != target) temp = temp->next;

    if (!temp) { cout << "Target element not found" << endl; return; }

    Node* newNode = new Node(data);

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

                if (temp->next) {
                        temp->next->prev = newNode;
                }

                temp->next = newNode;
}

void insertBefore(int target, int data) {
                if (!head) { cout << "List is empty" << endl; return; }

                if (head->data == target) { insertAtPosition(data, 1); return; }

                Node* temp = head;

                while (temp && temp->data != target) {
```

```
            temp = temp->next;
        }

        if (!temp) { cout << "Target element not found" << endl; return; }

        Node* newNode = new Node(data);

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

        if (temp->prev) {
            temp->prev->next = newNode;
        }

        temp->prev = newNode;

        if (temp == head) {
            head = newNode;
        }
    }

    void printList() const {
        if (!head){ cout << "List is empty." << endl; return; }

        for (Node* curr = head; curr != nullptr; curr = curr -> next)
```

```
        cout << curr -> data << " ";

        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;

    int choice, data, position, target;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Insert at end\n2. Delete from end\n3. Insert at position\n4. Delete from position\n5. Insert after element\n6. Insert before element\n7. Print list\n8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1: cout << "Enter data: "; cin >> data; list.insertAtEnd(data); break;
        case 2: list.deleteFromEnd(); break;
        case 3: cout << "Enter position: "; cin >> position; cout << "Enter data: "; cin >> data; list.insertAtPosition(data, position); break;
        case 4: cout << "Enter position: "; cin >> position; list.deleteFromPosition(position); break;
        case 5: cout << "Enter target element: "; cin >> target; cout << "Enter data: "; cin >> data; list.insertAfter(target, data); break;
```

```
        case 6: cout << "Enter target element: "; cin >> target; cout << "Enter data: "; cin >> data;
list.insertBefore(target, data); break;

        case 7: list.printList(); break;

        case 8: return 0;

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

        }

  }

}
```

OUTPUT:

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after element

6. Insert before element

7. Print list

8. Exit

Enter your choice: 1

Enter data: 1

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after element

6. Insert before element

7. Print list

8. Exit

Enter your choice: 1

Enter data: 2

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after element

6. Insert before element

7. Print list

8. Exit

Enter your choice: 7

1 2

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after element

6. Insert before element

7. Print list

8. Exit

Enter your choice: 8

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

```
#include <iostream>

using namespace std;


struct Node {

   int coefficient, exponent;

   Node* prev;

   Node* next;

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

};


class Polynomial {

   Node* head;


public:

   Polynomial() : head(nullptr) {}


   ~Polynomial() {

      while (head) deleteTerm(head->exponent);

   }
```

```
void insertTerm(int coefficient, int exponent) {
    if (coefficient == 0) return;

    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;
        if (temp->coefficient == 0) deleteTerm(temp->exponent);
        return;
    }

    newNode->prev = temp;
    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
```

```
    }

    void deleteTerm(int exponent) {
        Node* temp = head;

        while (temp && temp->exponent != exponent) temp = temp->next;

        if (!temp) return;

        if (temp->prev) temp->prev->next = temp->next;
        else head = temp->next;

        if (temp->next) temp->next->prev = temp->prev;
        delete temp;
    }

    Polynomial add(const Polynomial& other) const {
        Polynomial result;
        for (Node *p1 = head, *p2 = other.head; p1 || p2;) {
            if (!p1 || (p2 && p2->exponent > p1->exponent))
                result.insertTerm(p2->coefficient, p2->exponent), p2 = p2->next;
            else if (!p2 || (p1 && p1->exponent > p2->exponent))
                result.insertTerm(p1->coefficient, p1->exponent), p1 = p1->next;
            else
                result.insertTerm(p1->coefficient + p2->coefficient, p1->exponent), p1 = p1->next, p2 = p2-
>next;
```

```
        }
        return result;
    }

    void printPolynomial() const {
        for (Node* temp = head; temp; temp = temp->next) {
            cout << temp->coefficient << "x^" << temp->exponent << (temp->next ? " + " : "");
        }
        cout << endl;
    }
};

int main() {
    Polynomial poly1, poly2;

    for (int coeff, exp; true;) {
        cout << "Polynomial A: Enter coefficient and exponent (-1 to finish): ";
        cin >> coeff; if (coeff == -1) break; cin >> exp; poly1.insertTerm(coeff, exp);
    }

    for (int coeff, exp; true;) {
        cout << "Polynomial B: Enter coefficient and exponent (-1 to finish): ";
        cin >> coeff; if (coeff == -1) break; cin >> exp; poly2.insertTerm(coeff, exp);
    }
```

```
    Polynomial result = poly1.add(poly2);


    cout << "Polynomial A: "; poly1.printPolynomial();
    cout << "Polynomial B: "; poly2.printPolynomial();
    cout << "Result of A + B: "; result.printPolynomial();


    return 0;
}
```

OUTPUT:

```
Polynomial A: Enter coefficient and exponent (-1 to finish): 3 2
Polynomial A: Enter coefficient and exponent (-1 to finish): 5 1
Polynomial A: Enter coefficient and exponent (-1 to finish): 7 0
Polynomial A: Enter coefficient and exponent (-1 to finish): -1
Polynomial B: Enter coefficient and exponent (-1 to finish): 5 2
Polynomial B: Enter coefficient and exponent (-1 to finish): 7 1
Polynomial B: Enter coefficient and exponent (-1 to finish): 6 0
Polynomial B: Enter coefficient and exponent (-1 to finish): -1
Polynomial A: 3x^2 + 5x^1 + 7x^0
Polynomial B: 5x^2 + 7x^1 + 6x^0
Result of A + B: 8x^2 + 12x^1 + 13x^0
```