Name: Praveer Raj

Course: Maths and Computing

Roll No. 1

**Week -5**

**IMPLEMETATION OF STACKS 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

**PALINDROME USING STACKS**

#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: hello

The string is not a palindrome.

**PARENTHEZIED EXPRESSION**

#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 postfix expression: 23\*54\*+

Postfix evaluation result: 26

Enter prefix expression: +\*23\*54

Prefix evaluation result: 26

**LAB 6**

**Evaluation of postfix and prefix expressions**

#include <iostream>

#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: 23\*54\*+

Postfix evaluation result: 26

Enter prefix expression: +\*23\*54

Prefix evaluation result: 26

**INFIX EXPRESSION CONVERSION**

#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

Postfix Expression: ABC\*+

Prefix Expression: +A\*BC

Enter infix expression: (A+B)\*C

Postfix Expression: AB+C\*

Prefix Expression: \*+ABC

Enter infix expression: A\*(B+C)

Postfix Expression: ABC+\*

Prefix Expression: \*A+BC

**LAB - 7**

**IMPLEMENTATION OF CIRCULAR QUEUE**

#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

**IMPLEMENTATION OF 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 is full

Queue elements: 2 3 4 5

**IMPLEMENTATION OF QUEUES USING ARRAY**

#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

1

**LAB - 8**

**Menu Driven Linked List**

#include <iostream>

using namespace std;

struct Node { int data; Node\* next; };

class LinkedList {

Node\* head = nullptr;

public:

void insertAtBeginning(int value) {

Node\* newNode = new Node{value, head}; head = newNode;

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

}

void insertAtEnd(int value) {

Node\* newNode = new Node{value, nullptr};

if (!head) head = newNode;

else { Node\* temp = head; while (temp->next) temp = temp->next; temp->next = newNode; }

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

}

void insertBefore(int beforeValue, int value) {

if (!head) return void(cout << "List is empty.\n");

if (head->data == beforeValue) return insertAtBeginning(value);

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

if (temp->next) temp->next = new Node{value, temp->next}, cout << "Inserted " << value << " before " << beforeValue << ".\n";

else cout << beforeValue << " not found.\n";

}

void insertAfter(int afterValue, int value) {

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

if (temp) temp->next = new Node{value, temp->next}, cout << "Inserted " << value << " after " << afterValue << ".\n";

else cout << afterValue << " not found.\n";

}

void deleteElement(int value) {

if (!head) return void(cout << "List is empty.\n");

if (head->data == value) { Node\* toDelete = head; head = head->next; delete toDelete; cout << "Deleted " << value << ".\n"; return; }

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

if (temp->next) { Node\* toDelete = temp->next; temp->next = temp->next->next; delete toDelete; cout << "Deleted " << value << ".\n"; }

else cout << value << " not found.\n";

}

void printList() {

if (!head) return void(cout << "List is empty.\n");

cout << "List: "; for (Node\* temp = head; temp; temp = temp->next) cout << temp->data << " "; cout << endl;

}

};

int main() {

LinkedList list; int choice, value, target;

do {

cout << "\nMenu:\n1. Insert at the beginning\n2. Insert at the end\n3. Insert before\n4. Insert after\n5. Delete\n6. Print list\n0. Exit\nEnter choice: ";

cin >> choice;

if (choice == 1) { cout << "Enter value: "; cin >> value; list.insertAtBeginning(value); }

else if (choice == 2) { cout << "Enter value: "; cin >> value; list.insertAtEnd(value); }

else if (choice == 3) { cout << "Enter value and target: "; cin >> value >> target; list.insertBefore(target, value); }

else if (choice == 4) { cout << "Enter value and target: "; cin >> value >> target; list.insertAfter(target, value); }

else if (choice == 5) { cout << "Enter value to delete: "; cin >> value; list.deleteElement(value); }

else if (choice == 6) list.printList();

} while (choice != 0);

return 0;

}

**Output:**

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 1

Enter value: 10

Inserted 10 at the beginning.

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 2

Enter value: 20

Inserted 20 at the end.

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 6

List: 10 20

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 3

Enter value and target: 15 20

Inserted 15 before 20.

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 6

List: 10 15 20

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 4

Enter value and target: 25 15

Inserted 25 after 15.

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 6

List: 10 15 25 20

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 5

Enter value to delete: 15

Deleted 15.

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 6

List: 10 25 20

Menu:

1. Insert at the beginning

2. Insert at the end

3. Insert before

4. Insert after

5. Delete

6. Print list

0. Exit

Enter choice: 0

**Stack And Queue Using Linked List:**

#include <iostream>

#include <stack>

#include <list>

using namespace std;

// Stack Implementation using STL stack

class Stack {

stack<int> stk; // Using STL stack for Stack implementation

public:

void push(int value) {

stk.push(value); // Push onto the stack

cout << "Pushed " << value << " onto the stack.\n";

}

void pop() {

if (stk.empty()) {

cout << "Stack Underflow\n";

} else {

cout << "Popped " << stk.top() << " from the stack.\n";

stk.pop(); // Remove the top element

}

}

void display() {

if (stk.empty()) {

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

} else {

cout << "Stack (Top to Bottom): ";

stack<int> temp = stk; // Copy stack to display without modifying it

while (!temp.empty()) {

cout << temp.top() << " ";

temp.pop();

}

cout << endl;

}

}

};

// Queue Implementation using STL list

class Queue {

list<int> lst; // Using STL list for Queue implementation

public:

void enqueue(int value) {

lst.push\_back(value); // Insert at the end

cout << "Enqueued " << value << " to the queue.\n";

}

void dequeue() {

if (lst.empty()) {

cout << "Queue Underflow\n";

} else {

cout << "Dequeued " << lst.front() << " from the queue.\n";

lst.pop\_front(); // Remove from the front

}

}

void display() {

if (lst.empty()) {

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

} else {

cout << "Queue (Front to Rear): ";

for (int x : lst) cout << x << " ";

cout << endl;

}

}

};

// Main function to test both Stack and Queue

int main() {

Stack stack;

Queue queue;

// Testing Stack

cout << "Stack Operations:\n";

stack.push(10);

stack.push(20);

stack.display();

stack.pop();

stack.display();

// Testing Queue

cout << "\nQueue Operations:\n";

queue.enqueue(30);

queue.enqueue(40);

queue.display();

queue.dequeue();

queue.display();

return 0;

}

**OUTPUT:**

Stack Operations:

Pushed 10 onto the stack.

Pushed 20 onto the stack.

Stack (Top to Bottom): 20 10

Popped 20 from the stack.

Stack (Top to Bottom): 10

Queue Operations:

Enqueued 30 to the queue.

Enqueued 40 to the queue.

Queue (Front to Rear): 30 40

Dequeued 30 from the queue.

Queue (Front to Rear): 40

**LAB - 9**

**Adding Polynomial Using Double LinkList:**

#include <iostream>

using namespace std;

struct Node {

int coeff, exp;

Node\* prev; Node\* next;

Node(int c, int e) : coeff(c), exp(e), prev(nullptr), next(nullptr) {}

};

class Polynomial {

Node\* head;

public:

Polynomial() : head(nullptr) {}

~Polynomial() { while (head) deleteTerm(head->exp); }

void insertTerm(int c, int e) {

if (c == 0) return;

Node\* newNode = new Node(c, e);

if (!head || head->exp < e) { newNode->next = head; if (head) head->prev = newNode; head = newNode; return; }

Node\* temp = head;

while (temp->next && temp->next->exp > e) temp = temp->next;

if (temp->exp == e) { temp->coeff += c; delete newNode; if (temp->coeff == 0) deleteTerm(e); return; }

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

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

}

void deleteTerm(int e) {

Node\* temp = head; while (temp && temp->exp != e) 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->exp > p1->exp)) result.insertTerm(p2->coeff, p2->exp), p2 = p2->next;

else if (!p2 || (p1 && p1->exp > p2->exp)) result.insertTerm(p1->coeff, p1->exp), p1 = p1->next;

else result.insertTerm(p1->coeff + p2->coeff, p1->exp), p1 = p1->next, p2 = p2->next;

}

return result;

}

void printPolynomial() const {

for (Node\* temp = head; temp; temp = temp->next) cout << temp->coeff << "x^" << temp->exp << (temp->next ? " + " : "");

cout << endl;

}

};

int main() {

Polynomial poly1, poly2; int coeff, exp;

for (int coeff, exp; cin >> coeff && coeff != -1; cout << "Enter coefficient and exponent (-1 to finish): ") {

cin >> exp; poly1.insertTerm(coeff, exp);

}

for (int coeff, exp; cin >> coeff && coeff != -1; cout << "Enter coefficient and exponent (-1 to finish): ") {

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:**

Enter coefficient and exponent (-1 to finish): 3

Enter coefficient and exponent (-1 to finish): 2

Enter coefficient and exponent (-1 to finish): 5

Enter coefficient and exponent (-1 to finish): 0

Enter coefficient and exponent (-1 to finish): 1

Enter coefficient and exponent (-1 to finish): -1

Enter coefficient and exponent (-1 to finish): 4

Enter coefficient and exponent (-1 to finish): 1

Enter coefficient and exponent (-1 to finish): -1

Polynomial A: 3x^2 + 5x^0 + 1x^1

Polynomial B: 4x^1

Result of A + B: 3x^2 + 5x^0 + 5x^1

**Menu Driven Using Double LinkList:**

#include <iostream>

using namespace std;

class DoublyLinkedList {

struct Node { int data; Node \*prev, \*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 pos) {

if (pos < 1) { cout << "Position out of range" << endl; return; }

Node \*newNode = new Node(data);

if (pos == 1) { newNode->next = head; if (head) head->prev = newNode; head = newNode; return; }

Node \*temp = head;

for (int i = 1; temp && i < pos - 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 pos) {

if (!head || pos < 1) { cout << "Position out of range" << endl; return; }

Node \*temp = head;

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

for (int i = 1; temp && i < pos; 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; curr = curr->next) cout << curr->data << " ";

cout << endl;

}

};

int main() {

DoublyLinkedList list; int choice, data, position, target;

while (true) {

cout << "\nMenu:\n1. Insert at end\n2. Delete from end\n3. Insert at position\n4. Delete from position\n5. Insert after\n6. Insert before\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

6. Insert before

7. Print list

8. Exit

Enter your choice: 1

Enter data: 10

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 1

Enter data: 20

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 7

10 20

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 3

Enter position: 2

Enter data: 15

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 7

10 15 20

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 5

Enter target element: 15

Enter data: 18

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 7

10 15 18 20

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 2

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 7

10 15 18

Menu:

1. Insert at end

2. Delete from end

3. Insert at position

4. Delete from position

5. Insert after

6. Insert before

7. Print list

8. Exit

Enter your choice: 8