DS lab evaluation

Name: Ojas Apratim

Reg No:230957162

Branch :Mathematics and Computing

Week 5

A black background with white text

Description automatically generated

Code-

#include<iostream>

using namespace std;

class Stack {

private:

    char\* stk;

    int num;

    int top;

public:

    Stack(int size) {

        stk = new char[size];

        num = size;

        top = -1;

    }

    ~Stack() {

        delete[] stk;

    }

    bool isempty() {

        return top == -1;

    }

    char pop() {

        if (isempty()) {

            cout << "Underflow: Stack is empty" << endl;

            return '\0';

        }

        else {

            return stk[top--];

        }

    }

    void push(char i) {

        stk[++top] = i;

    }

    char peek() {

        if (isempty()) {

            return '\0';

        }

        else {

            return stk[top];

        }

    }

};

int strlen(char\* a) {

    int i = 0;

    while (a[i] != '\0') {

        i++;

    }

    return i;

}

bool bracketcheck(char\* a) {

    Stack stack(50);

    int i = 0;

    while (a[i] != '\0') {

        if (a[i] == '(' || a[i] == '{' || a[i] == '[') {

            stack.push(a[i]);

        }

        else if ((a[i] == ')' && stack.peek() == '(') ||(a[i] == '}' && stack.peek() == '{') ||(a[i] == ']' && stack.peek() == '[')) {

            stack.pop();

        }

        else if (a[i] == ')' || a[i] == '}' || a[i] == ']') {

            return false;

        }

        i++;

    }

    return stack.isempty();

}

bool ispalindrome(char\*a){

    Stack stack(50);

    int n=strlen(a);

    int i=0;

    for(i;i<n/2;i++){

        stack.push(a[i]);

    }

    int s = (n % 2 == 0) ? n / 2 : (n / 2) + 1;

   for (int i = s; i < n; ++i) {

        if (stack.peek() != a[i]) {

            return false;

        }

        stack.pop();

    }

    return true;

}

int main(){

    char a[20];

    cout<<"enter your expression"<<endl;

    cin>>a;

    int choice;

    while (true) {

        cout << "\nMenu:\n";

        cout << "1. check whether it is a plindrome or not\n";

        cout<<"2.check whether the expression is correctly parenthesized\n";

        cout<<"3.exit"<<endl;

        cout<<"enter your choice"<<endl;

        cin >> choice;

        switch(choice){

            case 1:

            if(ispalindrome(a)){

                cout<<"the given string is a palindrome"<<endl;

            }

            else{

                cout<<"the given string is not a palindrome"<<endl;

            }

            break;

            case 2:

            if(bracketcheck(a)){

                cout<<"the given expression is correctly parenthesized"<<endl;

            }

            else{

                cout<<"the given expression is  incorrectly parenthesized"<<endl;

            }

            break;

            case 3:

            exit(0);

            default:

            cout<<"invalid entry"<<endl;

            exit(0);

        }

    }

    return 0;

}

Output:

enter your expression

abcba

Menu:

1. check whether it is a palindrome or not

2.check whether the expression is correctly parenthesized

3.exit

enter your choice

1

the given string is a palindrome

Menu:

1. check whether it is a palindrome or not

2.check whether the expression is correctly parenthesized

3.exit

enter your choice

3

Week 6

A black background with white text

Description automatically generated

Q1-

Code-

#include<iostream>

using namespace std;

int precedence(char a) {

    if (a == '(' || a == ')') {

        return 0;

    }

    if (a == '\*' || a == '/' || a == '%') {

        return 2;

    }

    if (a == '+' || a == '-') {

        return 1;

    }

    return -1;

}

int strlen(char\* a) {

    int i = 0;

    while (a[i] != '\0') {

        i++;

    }

    return i;

}

class Stack {

private:

    char\* stk;

    int num;

    int top;

public:

    Stack(int size) {

        stk = new char[size];

        num = size;

        top = -1;

    }

    ~Stack() {

        delete[] stk;

    }

    bool isempty() {

        return top == -1;

    }

    char pop() {

        if (isempty()) {

            cout << "Underflow: Stack is empty" << endl;

            return '\0';

        }

        else {

            return stk[top--];

        }

    }

    void push(char i) {

        stk[++top] = i;

    }

    char peek() {

        if (isempty()) {

            return '\0';

        }

        else {

            return stk[top];

        }

    }

};

int main() {

    Stack stack(50);

    char a[100];

    cout << "Enter your expression: ";

    cin >> a;

    for (int i = 0; i < strlen(a); i++) {

        if ((a[i] >= 'a' && a[i] <= 'z') || (a[i] >= 'A' && a[i] <= 'Z') || (a[i] >= '0' && a[i] <= '9')) {

            cout << a[i];

        }

        else if (a[i] == '(') {

            stack.push(a[i]);

        }

        else if (a[i] == ')') {

            while (stack.peek() != '(') {

                cout << stack.pop();

            }

            stack.pop();

        }

        else {

            while (!stack.isempty() && precedence(stack.peek()) >= precedence(a[i])) {

                cout << stack.pop();

            }

            stack.push(a[i]);

        }

    }

    while (!stack.isempty()) {

        cout << stack.pop();

    }

    cout << endl;

    return 0;

}

Output:

Enter your expression: (a+b)\*(c/d)

ab+cd/\*

code-

#include <iostream>

using namespace std;

class Stack {

private:

    int top;

    int size;

    char\* stack;

public:

    Stack(int s) {

        top = -1;

        size = s;

        stack = new char[size];

    }

    ~Stack() {

        delete[] stack;

    }

    void push(char val) {

        if (top == size - 1) {

            cout << "Stack Overflow!" << endl;

        } else {

            stack[++top] = val;

        }

    }

    char pop() {

        if (top == -1) {

            cout << "Stack Underflow!" << endl;

            return -1;

        } else {

            return stack[top--];

        }

    }

    char peek() {

        if (top == -1) {

            return -1;

        } else {

            return stack[top];

        }

    }

    bool isEmpty() {

        return top == -1;

    }

};

bool isoperator(char c) {

    return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

int strlen(char\* a) {

    int i = 0;

    while (a[i] != '\0') {

        i++;

    }

    return i;

}

int precedence(char c) {

    if (c == '^')

        return 3;

    if (c == '\*' || c == '/')

        return 2;

    if (c == '+' || c == '-')

        return 1;

    return -1;

}

void reverse(string &str) {

    int n = str.length();

    for (int i = 0; i < n / 2; i++) {

        swap(str[i], str[n - i - 1]);

    }

}

string infixToPrefix(string infix) {

    reverse(infix);

    for (int i = 0; i < infix.length(); i++) {

        if (infix[i] == '(') {

            infix[i] = ')';

        } else if (infix[i] == ')') {

            infix[i] = '(';

        }

    }

    Stack stack(infix.length());

    string postfix;

    for (int i = 0; i < infix.length(); i++) {

        char c = infix[i];

        if (isalnum(c)) {

            postfix += c;

        }

        else if (c == '(') {

            stack.push(c);

        }

        else if (c == ')') {

            while (!stack.isEmpty() && stack.peek() != '(') {

                postfix += stack.pop();

            }

            stack.pop();

        }

        else if (isoperator(c)) {

            while (!stack.isEmpty() && precedence(stack.peek()) > precedence(c)) {

                postfix += stack.pop();

            }

            stack.push(c);

        }

    }

    while (!stack.isEmpty()) {

        postfix += stack.pop();

    }

    reverse(postfix);

    return postfix;

}

int main() {

    string infix;

    cout << "Enter an infix expression: ";

    cin >> infix;

    string prefix = infixToPrefix(infix);

    cout << "Prefix expression: " << prefix << endl;

    return 0;

}

Output:

Enter an infix expression: (a+b)\*(c/d)

Prefix expression: \*+ab/cd

Q2-

Code-

#include <iostream>

#include <cctype>

using namespace std;

int precedence(char a) {

    if (a == '(' || a == ')') {

        return 0;

    }

    if (a == '\*' || a == '/' || a == '%') {

        return 2;

    }

    if (a == '+' || a == '-') {

        return 1;

    }

    return -1;

}

class Stack {

private:

    int\* stk;

    int num;

    int top;

public:

    Stack(int size) {

        stk = new int[size];

        num = size;

        top = -1;

    }

    ~Stack() {

        delete[] stk;

    }

    bool isempty() {

        return top == -1;

    }

    int pop() {

        if (isempty()) {

            cout << "Underflow: Stack is empty" << endl;

            return '\0';

        }

        else {

            return stk[top--];

        }

    }

    void push(int i) {

        stk[++top] = i;

    }

    int peek() {

        if (isempty()) {

            return '\0';

        }

        else {

            return stk[top];

        }

    }

};

int strlen(char\* a) {

    int i = 0;

    while (a[i] != '\0') {

        i++;

    }

    return i;

}

bool isoperator(char i) {

    return (i == '%' || i == '\*' || i == '/' || i == '+' || i == '-');

}

int main() {

    Stack stack(50);

    char a[100], b[100];

    int top = -1;

    cout << "Enter your expression: ";

    cin >> a;

    int n = strlen(a);

    cout << "Your infix expression: " << a << endl;

    for (int i = 0; i < n; i++) {

        if (isdigit(a[i])) {

            while (i < n && isdigit(a[i])) {

                b[++top] = a[i];

                i++;

            }

            b[++top] = ' ';

            i--;

        }

        else if (a[i] == '(') {

            stack.push(a[i]);

        }

        else if (a[i] == ')') {

            while (stack.peek() != '(') {

                b[++top] = stack.pop();

            }

            stack.pop();

        }

        else if (isoperator(a[i])) {

            while (!stack.isempty() && precedence(stack.peek()) >= precedence(a[i])) {

                b[++top] = stack.pop();

            }

            stack.push(a[i]);

        }

    }

    while (!stack.isempty()) {

        b[++top] = stack.pop();

    }

    b[++top] = '\0';

    cout << "Postfix expression: " << b << endl;

    int operand1, operand2, result;

    for (int i = 0; i < strlen(b); i++) {

        if (isdigit(b[i])) {

            int num = 0;

            while (i < strlen(b) && isdigit(b[i])) {

                num = num \* 10 + (b[i] - '0');

                i++;

            }

            stack.push(num);

        }

        else if (isoperator(b[i])) {

            operand2 = stack.pop();

            operand1 = stack.pop();

            switch (b[i]) {

                case '+': result = operand1 + operand2; break;

                case '-': result = operand1 - operand2; break;

                case '\*': result = operand1 \* operand2; break;

                case '/':

                    if (operand2 == 0) {

                        cout << "Error: Division by zero" << endl;

                        return -1;

                    }

                    result = operand1 / operand2;

                    break;

                case '%': result = operand1 % operand2; break;

            }

            stack.push(result);

        }

    }

    cout << "Evaluated result: " << stack.pop() << endl;

    return 0;

}

Output:

Enter your expression: (9+2)\*(4/2)

Your infix expression: (9+2)\*(4/2)

Postfix expression: 9 2 +4 2 /\*

Evaluated result: 22

Week7

A black text on a white background

Description automatically generated

Code-

#include<iostream>

using namespace std;

#define max 10

class queue{

    private:

    char\*a=new char[max];

    int head;

    int tail;

    int size;

    public:

    queue(){

        head=0;

        tail=-1;

        size=0;

    }

    ~queue() {

        delete[] a;

    }

    bool isfull(){

        return size==max;

    }

    bool isempty(){

        return size==0;

    }

    void enqueue(int i){

        if(isfull()){

            cout<<"the queue is full";

        }

        tail=(tail+1)%max;

        a[tail]=i;

        size++;

    }

    char dequeue(){

        if(isempty()){

            cout<<"cannot dequeue the queue is empty";

            return 0;

        }

        int temp=a[head];

        head=(head+1)%max;

        size--;

        return temp;

    }

};

int main(){

    queue q;

    q.enqueue('q');

    q.enqueue('u');

    q.enqueue('e');

    q.enqueue('u');

    q.enqueue('e');

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

    cout<<q.dequeue()<<" ";

}

Output:

q u e u e

cannot dequeue the queue is empty

cannot dequeue the queue is empty

week8

A white background with black text

Description automatically generated

Code-

#include<iostream>

using namespace std;

class node{

    public:

    int data;

    node\*next;

};

class linkedlist{

    private:

    node\*head=nullptr;

    public:

    linkedlist() : head(nullptr) {}

    bool isempty(){

        return head==nullptr;

    }

    void insert(int i){

        node\*a=new node;

        a->data=i;

        a->next=nullptr;

        if(head==nullptr){

            head=a;

        }

        else{

            node\*temp=head;

            while(temp->next!=nullptr){

                temp=temp->next;

            }

            temp->next=a;

        }

    }

    void display(){

        node\*temp=head;

        if(head==nullptr){

            cout<<"nothing to display the list is empty";

        }

        else{

            while(temp!=nullptr){

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

                temp=temp->next;

            }

        }

        cout<<endl;

    }

    void del(int i){

        node\*temp=head;

        node\*prev=nullptr;

        if (temp == nullptr) {

        cout << "The list is empty." << endl;

        return;

        }

        while (temp != nullptr && temp->data != i) {

        prev = temp;

        temp = temp->next;

        }

        if (temp == nullptr) {

        cout << "The given element does not exist in the list." << endl;

        return;

    }

        prev->next = temp->next;

        delete temp;

    }

    void linktocircle(){

    if (head == nullptr) return;

    node\*temp=head;

    while(temp->next!=nullptr){

        temp=temp->next;

    }

    temp->next=head;

    }

    void displayupto(int n){

        node\*temp=head;

        for(int i=0;i<n;i++){

            cout<<temp->data<<" ";

            temp=temp->next;

        }

    }

    void insertat(int i,int index){

        node\*a=new node;

        a->data=i;

        a->next=nullptr;

        node\*temp=head;

        int c=2;

        if(isempty()){

            head->next=a;

        }

        while(c!=index && temp!=nullptr){

            temp=temp->next;

            c++;

        }

        if(c<index){

            temp->next=a;

        }

        else{

            a->next=temp->next;

            temp->next=a;

        }

    }

    void insertbefore(int a,int b){

        node\*newnode=new node;

        newnode->data=a;

        newnode->next=nullptr;

        node\*temp=head;

        node\*prev=nullptr;

        if (temp == nullptr) {

        cout << "The list is empty." << endl;

        return;

        }

        while (temp != nullptr && temp->data != b) {

        prev = temp;

        temp = temp->next;

        }

        if (temp == nullptr) {

        cout << "The given element does not exist in the list." << endl;

        return;

    }

    newnode->next=prev->next;

    prev->next=newnode;

    }

    void insertafter(int a,int b){

        node\*newnode=new node;

        newnode->data=a;

        newnode->next=nullptr;

        node\*temp=head;

        if (temp == nullptr) {

        cout << "The list is empty." << endl;

        return;

        }

        while (temp != nullptr && temp->data != b) {

        temp = temp->next;

        }

        if (temp == nullptr) {

        cout << "The given element does not exist in the list." << endl;

        return;

    }

    newnode->next=temp->next;

    temp->next=newnode;

    }

    void insertatstart(int a){

        node\*newnode=new node;

        newnode->data=a;

        newnode->next=nullptr;

        newnode->next=head;

        head=newnode;

    }

    void insertatend(int a){

        node\* newnode = new node;

        newnode->data = a;

        newnode->next = nullptr;

        if (head == nullptr) {

            head = newnode;

        } else {

            node\* temp = head;

            while (temp->next != nullptr) {

                temp = temp->next;

            }

            temp->next = newnode;

        }

    }

     void deallocate() {

        node\* temp = head;

        while (temp != nullptr) {

            node\* nextNode = temp->next;

            delete temp;

            temp = nextNode;

        }

        head = nullptr;

    }

    ~linkedlist() {

        deallocate();

    }

};

int main() {

    linkedlist list;

    int choice, value, target;

    list.insert(10);

    list.insert(50);

    list.insert(60);

    list.insert(70);

    list.insert(80);

    list.insert(90);

    while (true) {

        cout << "\nMenu:\n";

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

        cout << "2. Insert at end\n";

        cout << "3. Insert before an element\n";

        cout << "4. Insert after an element\n";

        cout << "5. Delete an 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 the beginning: ";

                cin >> value;

                list.insertatstart(value);

                break;

            case 2:

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

                cin >> value;

                list.insertatend(value);

                break;

            case 3:

                cout << "Enter target element before which to insert: ";

                cin >> target;

                cout << "Enter value to insert: ";

                cin >> value;

                list.insertbefore(target, value);

                break;

            case 4:

                cout << "Enter target element after which to insert: ";

                cin >> target;

                cout << "Enter value to insert: ";

                cin >> value;

                list.insertafter(target, value);

                break;

            case 5:

                cout << "Enter value to delete: ";

                cin >> value;

                list.del(value);

                break;

            case 6:

                list.display();

                break;

            case 7:

                exit(0);

            default:

                cout << "Invalid choice. Try again." << endl;

        }

    }

    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: 3

Enter target element before which to insert: 60

Enter value to insert: 55

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

10 50 55 60 70 80 10

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

Q2-

Code-

class linked{

    private:

    struct node{

        char data;

        node\*next;

    }\*p;

    public:

    linked(){

        p=nullptr;

    }

    ~linked();

    bool isempty();

    void insert(char i);

    void push(char i);

    void display();

    void insertat(int loc,char i);

    void del(char i);

    char pop();

    char peek();

};

void linked::insert(char i){

    node\*a=new node;

    a->data=i;

    a->next=nullptr;

    if(p==nullptr){

        p=a;

    }

    else{

        node\*temp=p;

        while(temp->next!=nullptr){

            temp=temp->next;

        }

        temp->next=a;

    }

}

void linked::display(){

    node\*temp=p;

    while(temp!=nullptr){

        cout<<temp->data<<" ";

        temp=temp->next;

    }

    cout<<endl;

}

void linked::insertat(int loc, char i){

    node\*a=new node;

    a->data=i;

    a->next=nullptr;

    if(p==nullptr){

        p=a;

    }

    else{

        node\*temp=p;

        for(int i=0 ;(i<loc-2 && temp->next!=nullptr);i++){

            temp=temp->next;

        }

        a->next=temp->next;

        temp->next=a;

    }

}

void linked::del(char i){

    node\*temp=p;

    node\*a;

    if (p->data == i) {

        a = p;

        p = p->next;

        delete a;

        return;

    }

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

        temp = temp->next;

    }

    if(temp->next==nullptr){

        cout<<"the element not found"<<endl;

        return;

    }

    a=temp->next;

    temp->next=temp->next->next;

    delete a;

}

linked::~linked(){

    node\* temp;

    while (p != nullptr) {

        temp = p;

        p = p->next;

        delete temp;

    }

}

bool linked::isempty(){

    return p==nullptr;

}

void linked::push(char i){

    node\*a=new node;

    a->data=i;

    a->next=p;

    p=a;

}

char linked::pop(){

    if (isempty()) {

        return '\0';

    }

    char a;

    node\*temp=p;

    a=p->data;

    p=p->next;

    delete temp;

    return a;

}

char linked::peek(){

    if (isempty()) {

        return '\0';

    }

    return p->data;

}

class Node {

public:

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = nullptr;

    }

};

class Queue {

private:

    Node\* front;

    Node\* rear;

public:

    Queue() {

        front = nullptr;

        rear = nullptr;

    }

    bool isEmpty() {

        return front == nullptr;

    }

    void enqueue(int value) {

        Node\* newNode = new Node(value);

        if (isEmpty()) {

            front = rear = newNode;

        } else {

            rear->next = newNode;

            rear = newNode;

        }

    }

    int dequeue() {

        if (isEmpty()) {

            cout << "Queue is empty Nothing to dequeue." << endl;

            return -1;

        }

        Node\* temp = front;

        int dequeuedValue = temp->data;

        front = front->next;

        if (front == nullptr) {

            rear = nullptr;

        }

        delete temp;

        return dequeuedValue;

    }

    void display() {

        if (isEmpty()) {

            cout << "Queue is empty." << endl;

            return;

        }

        Node\* temp = front;

        while (temp != nullptr) {

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

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    Queue q;

    q.enqueue(10);

    q.enqueue(20);

    q.enqueue(30);

    q.enqueue(40);

    q.display();

    q.dequeue();

    q.display();

    q.dequeue();

    q.display();

    return 0;

}

Output:

10 20 30 40

20 30 40

30 40

Week 9

A white background with black text

Description automatically generated

1.

Code-

#include<iostream>

using namespace std;

class node {

public:

    int data;

    node\* next;

    node\* back;

};

class doublelink {

private:

    node\* head = nullptr;

public:

    doublelink() : head(nullptr) {}

    bool isempty() {

        return head == nullptr;

    }

    void insert(int i) {

        node\* newNode = new node;

        newNode->data = i;

        newNode->next = nullptr;

        newNode->back = nullptr;

        if (head == nullptr) {

            head = newNode;

        } else {

            node\* temp = head;

            while (temp->next != nullptr) {

                temp = temp->next;

            }

            temp->next = newNode;

            newNode->back = temp;

        }

    }

    void display() {

        node\* temp = head;

        while (temp != nullptr) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void insertafter(int a, int b) {

        node\* newNode = new node;

        newNode->data = a;

        newNode->next = nullptr;

        newNode->back = nullptr;

        node\* temp = head;

        while (temp != nullptr && temp->data != b) {

            temp = temp->next;

        }

        if (temp == nullptr) {

            cout << "The element " << b << " does not exist in the list.\n";

        } else {

            newNode->next = temp->next;

            if (temp->next != nullptr) {

                temp->next->back = newNode;

            }

            temp->next = newNode;

            newNode->back = temp;

        }

    }

    void insertbefore(int a, int b) {

        node\* newNode = new node;

        newNode->data = a;

        newNode->next = nullptr;

        newNode->back = nullptr;

        if (head != nullptr && head->data == b) {

            newNode->next = head;

            head->back = newNode;

            head = newNode;

            return;

        }

        node\* temp = head;

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

            temp = temp->next;

        }

        if (temp == nullptr || temp->next == nullptr) {

            cout << "The element " << b << " does not exist in the list.\n";

        } else {

            newNode->next = temp->next;

            newNode->back = temp;

            temp->next->back = newNode;

            temp->next = newNode;

        }

    }

    void deleteatindex(int n) {

        if (isempty()) {

            cout << "The list is empty.\n";

            return;

        }

        node\* temp = head;

        if (n == 0) {

            head = temp->next;

            if (head != nullptr) {

                head->back = nullptr;

            }

            delete temp;

            return;

        }

        for (int i = 0; temp != nullptr && i < n; i++) {

            temp = temp->next;

        }

        if (temp == nullptr) {

            cout << "The list is shorter than the given index.\n";

            return;

        }

        if (temp->next != nullptr) {

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

        }

        if (temp->back != nullptr) {

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

        }

        delete temp;

    }

    void deallocate() {

        node\* temp = head;

        while (temp != nullptr) {

            node\* nextNode = temp->next;

            delete temp;

            temp = nextNode;

        }

        head = nullptr;

        cout << "All nodes deleted, memory deallocated.\n";

    }

    ~doublelink() {

        deallocate();

    }

};

int main() {

    doublelink list;

    int choice, value, target;

    while (true) {

        cout << "\nMenu:\n";

        cout << "1. Insert at end\n";

        cout << "2. Delete a node by index\n";

        cout << "3. Insert before an element\n";

        cout << "4. Insert after an element\n";

        cout << "5. Display list\n";

        cout << "6. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

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

                cin >> value;

                list.insert(value);

                break;

            case 2:

                cout << "Enter the index of the node to be deleted: ";

                cin >> value;

                list.deleteatindex(value);

                break;

            case 3:

                cout << "Enter the target value before which the element is inserted: ";

                cin >> target;

                cout << "Enter the value to be inserted before: ";

                cin >> value;

                list.insertbefore(value, target);

                break;

            case 4:

                cout << "Enter the target value after which the element is inserted: ";

                cin >> target;

                cout << "Enter the value to be inserted after: ";

                cin >> value;

                list.insertafter(value, target);

                break;

            case 5:

                list.display();

                break;

            case 6:

                exit(0);

            default:

                cout << "Invalid choice. Try again.\n";

        }

    }

    return 0;

}

Output:

Menu:

1. Insert at end

2. Delete a node by index

3. Insert before an element

4. Insert after an element

5. Display list

6. Exit

Enter your choice: 3

Enter the target value before which the element is inserted: 60

Enter the value to be inserted before: 55

Menu:

1. Insert at end

2. Delete a node by index

3. Insert before an element

4. Insert after an element

5. Display list

6. Exit

Enter your choice: 5

10 50 55 60 70 80 10

Menu:

1. Insert at end

2. Delete a node by index

3. Insert before an element

4. Insert after an element

5. Display list

6. Exit

Enter your choice: 6

2-

Code-#include <iostream>

using namespace std;

class Node {

public:

    int coeff;

    int exp;

    Node\* next;

    Node\* back;

    Node(int c, int e) {

        coeff = c;

        exp = e;

        next = nullptr;

        back = nullptr;

    }

};

class Polynomial {

private:

    Node\* head;

public:

    Polynomial() {

        head = nullptr;

    }

    void insertTerm(int c, int e) {

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

        if (!head) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != nullptr && temp->exp > e) {

            temp = temp->next;

        }

        if (temp->exp == e) {

            temp->coeff += c;

        } else if (temp->exp < e) {

            newNode->next = temp;

            newNode->back = temp->back;

            if (temp->back != nullptr) {

                temp->back->next = newNode;

            } else {

                head = newNode;

            }

            temp->back = newNode;

        } else {

            temp->next = newNode;

            newNode->back = temp;

        }

    }

    void display() {

        if (!head) {

            cout << "0" << endl;

            return;

        }

        Node\* temp = head;

        while (temp != nullptr) {

            if (temp->coeff != 0) {

                cout << temp->coeff << "x^" << temp->exp;

                if (temp->next != nullptr && temp->next->coeff > 0) {

                    cout << " + ";

                }

            }

            temp = temp->next;

        }

        cout << endl;

    }

    static Polynomial add(Polynomial& p1, Polynomial& p2) {

        Polynomial result;

        Node\* temp1 = p1.head;

        Node\* temp2 = p2.head;

        while (temp1 != nullptr && temp2 != nullptr) {

            if (temp1->exp == temp2->exp) {

                result.insertTerm(temp1->coeff + temp2->coeff, temp1->exp);

                temp1 = temp1->next;

                temp2 = temp2->next;

            } else if (temp1->exp > temp2->exp) {

                result.insertTerm(temp1->coeff, temp1->exp);

                temp1 = temp1->next;

            } else {

                result.insertTerm(temp2->coeff, temp2->exp);

                temp2 = temp2->next;

            }

        }

        while (temp1 != nullptr) {

            result.insertTerm(temp1->coeff, temp1->exp);

            temp1 = temp1->next;

        }

        while (temp2 != nullptr) {

            result.insertTerm(temp2->coeff, temp2->exp);

            temp2 = temp2->next;

        }

        return result;

    }

};

int main() {

    Polynomial poly1, poly2, result;

    int n, c, e;

    cout << "Enter the number of terms in the first polynomial: ";

    cin >> n;

    for (int i = 0; i < n; ++i) {

        cout << "Enter coefficient and exponent for term " << i + 1 << ": ";

        cin >> c >> e;

        poly1.insertTerm(c, e);

    }

    cout << "Enter the number of terms in the second polynomial: ";

    cin >> n;

    for (int i = 0; i < n; ++i) {

        cout << "Enter coefficient and exponent for term " << i + 1 << ": ";

        cin >> c >> e;

        poly2.insertTerm(c, e);

    }

    cout << "\nFirst Polynomial: ";

    poly1.display();

    cout << "Second Polynomial: ";

    poly2.display();

    result = Polynomial::add(poly1, poly2);

    cout << "Resultant Polynomial after addition: ";

    result.display();

    return 0;

}

Output:

Enter the number of terms in the first polynomial: 3

Enter coefficient and exponent for term 1: 3

1

Enter coefficient and exponent for term 2: 0

4

Enter coefficient and exponent for term 3: 5

3

Enter the number of terms in the second polynomial: 4

Enter coefficient and exponent for term 1: 5

1

Enter coefficient and exponent for term 2: 7

3

Enter coefficient and exponent for term 3: 6

3

Enter coefficient and exponent for term 4: 0

0

First Polynomial: 5x^3 + 3x^1

Second Polynomial: 13x^3 + 5x^1

Resultant Polynomial after addition: 18x^3 + 8x^1