**Polynomial Addition with Array**

#include <iostream>

using namespace std;

// max function

int max(int m, int n)

{

    return (m > n)? m: n;

}

// addition funtion

int \*add(int A[], int B[], int m, int n)

{

   int size = max(m, n);

   int \*sum = new int[size];

   for (int i = 0; i<m; i++)

     sum[i] = A[i];

   for (int i=0; i<n; i++)

       sum[i] += B[i];

   return sum;

}

// print function

void print(int poly[], int n)

{

    for (int i=0; i<n; i++)

    {

       cout << poly[i];

       if (i != 0)

        cout << "x^" << i ;

       if (i != n-1)

       cout << " + ";

    }

}

// main funtion

int main()

{

    int A[] = {10, 20, 30};

    int B[] = {40, 30, 20, 10};

    int m = sizeof(A)/sizeof(A[0]);

    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is \n";

    print(A, m);

    cout << "\nSecond polynomial is \n";

    print(B, n);

    int \*sum = add(A, B, m, n);

    int size = max(m, n);

    cout << "\nsum polynomial is \n";

    print(sum, size);

    return 0;

}

**Output :**

First polynomial is

10 + 20x^1 + 30x^2

Second polynomial is

40 + 30x^1 + 20x^2 + 10x^3

sum polynomial is

50 + 50x^1 + 50x^2 + 10x^3

**Polynomial Multiplication with Array**

#include <iostream>

using namespace std;

int \*multiply(int A[], int B[], int m, int n)

{

   int \*prod = new int[m+n-1];

   for (int i = 0; i<m+n-1; i++)

     prod[i] = 0;

   for (int i=0; i<m; i++)

   {

     for (int j=0; j<n; j++)

     {

       prod[i+j] += A[i]\*B[j];

     }

   }

   return prod;

}

void print(int poly[], int n)

{

    for (int i=0; i<n; i++)

    {

       cout << poly[i];

       if (i != 0)

       {

          cout << "x^" << i ;

       }

       if (i != n-1)

       {

          cout << " + ";

       }

    }

}

//main function

int main()

{

    int A[] = {10, 20, 30};

    int B[] = {40, 30, 20, 10};

    int m = sizeof(A)/sizeof(A[0]);

    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is n";

    print(A, m);

    cout << "nSecond polynomial is n";

    print(B, n);

    int \*prod = multiply(A, B, m, n);

    cout << "nProduct polynomial is n";

    print(prod, m+n-1);

    return 0;

}

**Output :**

**First polynomial is**

**10 + 20x^1 + 30x^2**

**Second polynomial is**

**40 + 30x^1 + 20x^2 + 10x^3**

**Product polynomial is**

**400 + 1100x^1 + 2000x^2 + 1400x^3 + 800x^4 + 300x^5**

**Polynomial Addition with LL**

#include<iostream>

using namespace std;

struct Node

{

    int coefficient;

    int pow;

    struct Node \*next;

};

void create\_node(int x, int y, struct Node \*\*temp)

{

    struct Node \*r, \*z;

    z = \*temp;

    if(z == NULL)

    {

        r =(struct Node\*)malloc(sizeof(struct Node));

        r->coefficient = x;

        r->pow = y;

        \*temp = r;

        r->next = (struct Node\*)malloc(sizeof(struct Node));

        r = r->next;

        r->next = NULL;

    }

    else

    {

        r->coefficient = x;

        r->pow = y;

        r->next = (struct Node\*)malloc(sizeof(struct Node));

        r = r->next;

        r->next = NULL;

    }

}

void polyadd(struct Node \*poly1, struct Node \*poly2, struct Node \*poly)

{

while(poly1->next && poly2->next)

    {

        if(poly1->pow > poly2->pow)

        {

            poly->pow = poly1->pow;

            poly->coefficient = poly1->coefficient;

            poly1 = poly1->next;

        }

        else if(poly1->pow < poly2->pow)

        {

            poly->pow = poly2->pow;

            poly->coefficient = poly2->coefficient;

            poly2 = poly2->next;

        }

        else

        {

            poly->pow = poly1->pow;

            poly->coefficient = poly1->coefficient+poly2->coefficient;

            poly1 = poly1->next;

            poly2 = poly2->next;

        }

        poly->next = (struct Node \*)malloc(sizeof(struct Node));

        poly = poly->next;

        poly->next = NULL;

    }

while(poly1->next || poly2->next)

    {

        if(poly1->next)

        {

            poly->pow = poly1->pow;

            poly->coefficient = poly1->coefficient;

            poly1 = poly1->next;

        }

        if(poly2->next)

        {

            poly->pow = poly2->pow;

            poly->coefficient = poly2->coefficient;

            poly2 = poly2->next;

        }

        poly->next = (struct Node \*)malloc(sizeof(struct Node));

        poly = poly->next;

        poly->next = NULL;

    }

}

void show(struct Node \*node)

{

while(node->next != NULL)

    {

    printf("%dx^%d", node->coefficient, node->pow);

    node = node->next;

    if(node->next != NULL)

        printf(" + ");

    }

}

int main()

{

    struct Node \*poly1 = NULL, \*poly2 = NULL, \*poly = NULL;

    create\_node(5,2,&poly1);

    create\_node(4,1,&poly1);

    create\_node(2,0,&poly1);

    create\_node(5,1,&poly2);

    create\_node(5,0,&poly2);

    printf("1st Number: ");

    show(poly1);

    printf("\n2nd Number: ");

    show(poly2);

    poly = (struct Node \*)malloc(sizeof(struct Node));

    polyadd(poly1, poly2, poly);

    printf("\nAdded polynomial: ");

    show(poly);

return 0;

}

**Output :**

**1st Number: 5x^2 + 4x^1 + 2x^0**

**2nd Number: 5x^1 + 5x^0**

**Added polynomial: 5x^2 + 9x^1 + 7x^0**

**Polynomial Multiplication with LL**

#include <iostream>

using namespace std;

struct Node {

    int coefficient, power;

    Node\* next;

};

Node\* addnode(Node\* start, int coeff, int power)

{

    Node\* newnode = new Node;

    newnode->coefficient = coeff;

    newnode->power = power;

    newnode->next = NULL;

    if (start == NULL)

        return newnode;

    Node\* ptr = start;

    while (ptr->next != NULL)

        ptr = ptr->next;

    ptr->next = newnode;

    return start;

}

void printList(struct Node\* ptr)

{

    while (ptr->next != NULL) {

        cout << ptr->coefficient << "x^" << ptr->power << " + ";

        ptr = ptr->next;

    }

    cout << ptr->coefficient << "\n";

}

void removeDuplicates(Node\* start)

{

    Node \*ptr1, \*ptr2, \*dup;

    ptr1 = start;

    while (ptr1 != NULL && ptr1->next != NULL) {

        ptr2 = ptr1;

        while (ptr2->next != NULL) {

            if (ptr1->power == ptr2->next->power) {

                ptr1->coefficient = ptr1->coefficient + ptr2->next->coefficient;

                dup = ptr2->next;

                ptr2->next = ptr2->next->next;

                delete (dup);

            }

            else

                ptr2 = ptr2->next;

        }

        ptr1 = ptr1->next;

    }

}

Node\* multiply(Node\* poly1, Node\* poly2,

               Node\* poly3)

{

    Node \*ptr1, \*ptr2;

    ptr1 = poly1;

    ptr2 = poly2;

    while (ptr1 != NULL) {

        while (ptr2 != NULL) {

            int coeff, power;

            coeff = ptr1->coefficient \* ptr2->coefficient;

            power = ptr1->power + ptr2->power;

            poly3 = addnode(poly3, coeff, power);

            ptr2 = ptr2->next;

        }

        ptr2 = poly2;

        ptr1 = ptr1->next;

    }

    removeDuplicates(poly3);

    return poly3;

}

// Driver Code

int main()

{

    Node \*poly1 = NULL, \*poly2 = NULL, \*poly3 = NULL;

    poly1 = addnode(poly1, 3, 2);

    poly1 = addnode(poly1, 5, 1);

    poly1 = addnode(poly1, 6, 0);

    poly2 = addnode(poly2, 6, 1);

    poly2 = addnode(poly2, 8, 0);

    cout << "1st Polynomial:- ";

    printList(poly1);

    cout << "2nd Polynomial:- ";

    printList(poly2);

    poly3 = multiply(poly1, poly2, poly3);

    cout << "Resultant Polynomial:- ";

    printList(poly3);

    return 0;

}

**Output :**

**1st Polynomial:- 3x^2 + 5x^1 + 6**

**2nd Polynomial:- 6x^1 + 8**

**Resultant Polynomial:- 18x^3 + 54x^2 + 76x^1 + 48**

**Singly LinkList**

#include<iostream>

using namespace std;

// Structure Declaration

struct node{

    int data;

    struct node \*next;

};

// Functions Declaration

void menu(struct node \*,struct node \*);

int get\_n(char);

struct node \* insert\_beg(struct node \*,struct node \*,int);

struct node \* insert\_end(struct node \*,struct node \*, int);

struct node \* insert\_atany(struct node \*,struct node \*, int);

struct node \* delete\_data(struct node \*,struct node \*, int);

void display\_link(struct node \*);

// Void Main

int main()

{

    struct node \*struct\_new;

    struct node \*head = NULL;

    menu(struct\_new,head); // Calling menu funtion

    return 0;

}

// menu function

void menu( struct node \*struct\_new, struct node \*head )

{

    int n,getnum;

    cout << "\n 1 . Add New Data To Linklist From Begining. \n 2 . Add New Data To Linklist From Ending.\n 3 . Add New Data To Linklist At Any Place. \n 4 . Delete a Number From The Link-List. \n 5 . Display LinkList Till Now. \n 6 . Exit. \n";

    cin >> n;

    // Switch case which check the user input and run specified function

    switch(n)

    {

        case(1):

            getnum = get\_n('i');

            head = insert\_beg(struct\_new,head,getnum);   //insertion from begining linklist function call

            menu(struct\_new,head);    //void menu function call

        case(2):

            getnum = get\_n('i');

            head = insert\_end(struct\_new,head,getnum);   //insertion from ending linklist function call

            menu(struct\_new,head);    //void menu function call

        case(3):

            getnum = get\_n('i');

            head = insert\_atany(struct\_new,head,getnum);   //insertion from any point linklist function call

            menu(struct\_new,head);    //void menu function call

        case(4):

            getnum = get\_n('d');

            head = delete\_data(struct\_new,head,getnum);

            menu(struct\_new,head);

        case(5):

            display\_link(head);     //display linklist function call

            menu(struct\_new,head);     //void menu function call

        case(6):

            exit(0);    //exit function call which terminated the program

        default:

            cout << "\n Please Enter Valid Number.";

            menu(struct\_new,head);     //void menu function call

    }

}

// function for taking input from user

int get\_n(char a)

{

    int n;

    if( a == 'i' )

    {

        cout << " Enter The Number : ";

    }

    else{

        cout << " Enter The Number to Delete : ";

    }

    scanf("%d",&n);

    return n;

}

//  function insert\_beg, use for linklist begining insertion

struct node \* insert\_beg( struct node \*struct\_new, struct node \*head,int n )

{

    struct\_new = (struct node \*)malloc(sizeof(struct node));

    struct\_new->data = n;

    struct\_new->next = head;

    head = struct\_new;

    return head;

}

//  function insert\_end, use for linklist ending insertion

struct node \* insert\_end( struct node \*struct\_new, struct node \*head, int n )

{

    struct node \*temp;

    struct\_new = (struct node \*)malloc(sizeof(struct node));

    if( head == NULL )

    {

        head = struct\_new;

        temp = head;

    }

    else{

        temp = head;

        while( temp->next != NULL ) // loop until next has NULL

        {

            temp = temp->next;

        }

    }

    temp->next = struct\_new;

    struct\_new->data = n;

    struct\_new->next = NULL;

    return head;

}

//  function insert\_atany, use for linklist any-point insertion

struct node \* insert\_atany( struct node \*struct\_new, struct node \*head, int n )

{

    struct node \*first;

    struct node \*last;

    first = head;

    struct\_new = (struct node \*)malloc(sizeof(struct node));

    if( head == NULL || head->data >= n )  // check if head already NUll or input value of user need to insert at begining

    {

        struct\_new->data = n;

        struct\_new->next = head;

        head = struct\_new;

    }

    else{

        while( first != NULL && first->data < n )  // loop until user input in greater

        {

            last = first;   // store last linklist address

            first = first->next; // store next linklist address

        }

        struct\_new->data = n;

        struct\_new->next = first;

        last->next = struct\_new;

    }

    return head;

}

struct node \* delete\_data( struct node \*struct\_new, struct node \*head,int n )

{

    struct node \*temp,\*tempstore;

    temp = head;

    if( head == NULL )

    {

        cout << "\n There is Nothing To Delete. \n";

    }

    else if( temp->data == n )

    {

        head = temp->next;

        free(temp);

    }

    else{

        if( temp->data != n && temp->next == NULL )

        {

            cout << "\n No Such Data To Delete. \n";

        }

        else if( temp->data == n && temp->next == NULL )

        {

            free(temp);

            head = NULL;

        }

        else{

            while( temp->next->data != n )

            {

                if( temp->next->next != NULL )

                {

                    temp = temp->next;

                }

                else{

                    cout << "\n No Such Data To Delete. \n";

                    menu(struct\_new,head);

                }

            }

            tempstore = temp->next;

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

            free(tempstore);

        }

    }

    return head;

}

//  function display\_link will display the linklist elements

void display\_link(struct node \*head)

{

    struct node \*temp;

    if(head == NULL)    // check wheater the head is null

    {

        cout << "\nThere Is Nothing To Display.\n";

    }

    else

    {

        temp = head;

        cout << "\nThe List is : \n";

        while(temp->next != NULL)   // print all the elements from the link-list

        {

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

            temp = temp->next;

        }

       cout << temp->data;

    }

}

**Output :**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**1**

**Enter The Number : 3**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**1**

**Enter The Number : 2**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**2**

**Enter The Number : 6**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**3**

**Enter The Number : 5**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**5**

**The List is :**

**2 => 3 => 5 => 6 %d**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**4**

**Enter The Number to Delete : 5**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**5**

**The List is :**

**2 => 3 => 6**

**1 . Add New Data To Linklist From Begining.**

**2 . Add New Data To Linklist From Ending.**

**3 . Add New Data To Linklist At Any Place.**

**4 . Delete a Number From The Link-List.**

**5 . Display LinkList Till Now.**

**6 . Exit.**

**Stack With Array**

#include <iostream>

using namespace std;

template <typename T>

class Stack

{

    T \*data;

    short top, size;

    public:

        Stack(int size)

        {

            if (size < 1)

                size = 5;

                this->size = size;

                top = -1;

                data = new T[this->size];

        }

        bool isFull()

        {

            return (top > size - 1);

        }

        bool isEmpty()

        {

            return (top < 0);

        }

        void push(T item)

        {

            if (isFull())

            {

                cout << "Stack Overflow" << endl;

                return;

            }

            data[++top] = item;

        }

        T pop()

        {

            if (isEmpty())

            {

                cout << "Stack is empty!" << endl;

                return NULL;

            }

            return data[top--];

        }

        void display()

        {

            if (isEmpty())

                cout << "Stack is empty!" << endl;

            else

            {

                cout << "TOP -> " << endl;

                for (int i = top; i >= 0; i--)

                cout << "-> " << data[i] << endl;

            }

        }

        ~Stack()

        {

            if (data)

            delete data;

        }

};

void menu();

int main()

{

 menu();

 return 0;

}

void menu()

{

    short size;

    cout << "Enter the size of stack: ";

    cin >> size;

    Stack<int> stack1(size);

    short check;

    int item;

    do

    {

        cout << "\n1. Push\n2. Pop\n3. Display\n4. Exit\n-> ";

        cin >> check;

        switch (check)

        {

            case 1:

                cout << "Enter item to push: ";

                cin >> item;

                stack1.push(item);

            break;

            case 2:

                item = stack1.pop();

                if (item)

                cout << "Deleted Item: " << item << endl;

            break;

            case 3:

                stack1.display();

            break;

            case 4:

            break;

            default:

                cout << "Select Proper Selection." << endl;

        }

    } while (check);

}

**Output :**

**Enter the size of stack: 4**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 1**

**Enter item to push: 5**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 1**

**Enter item to push: 3**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 3**

**TOP ->**

**-> 3**

**-> 5**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 2**

**Deleted Item: 3**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 3**

**TOP ->**

**-> 5**

**1. Push**

**2. Pop**

**3. Display**

**4. Exit**

**-> 4**

**Stack With LL**

#include <iostream>

using namespace std;

class Stack

{

    class Item

    {

        public:

        int data;

        Item \*nextItem;

        Item(int value)

        {

            data = value;

            nextItem = NULL;

        }

        ~Item()

        {

            if (nextItem)

            delete nextItem;

        }

    };

    Item \*top;

    short numberOfItems, size;

    public:

        Stack(int size)

        {

            if (size < 1)

                size = 5;

            this->size = size;

            numberOfItems = 0;

            top = NULL;

        }

        bool isFull()

        {

            return (numberOfItems >= size);

        }

        bool isEmpty()

        {

            return !top;

        }

        void push(int value)

        {

            if (isFull())

            {

                cout << "Stack Overflow" << endl;

                return;

            }

            Item \*item = new Item(value);

            item->nextItem = top;

            top = item;

            numberOfItems++;

        }

        int pop()

        {

            if (isEmpty())

            {

                cout << "Stack is empty!" << endl;

                return NULL;

            }

            Item \*itemToBeDeleted = top;

            top = itemToBeDeleted->nextItem;

            itemToBeDeleted->nextItem = NULL;

            int deletedData = itemToBeDeleted->data;

            numberOfItems--;

            delete itemToBeDeleted;

            return deletedData;

        }

        void display()

        {

            if (isEmpty())

                cout << "Stack is empty!" << endl;

            else

            {

                cout << "TOP -> ";

                Item \*item = top;

                while (item)

                {

                    cout << "-> " << item->data << endl;

                    item = item->nextItem;

                }

            }

        }

        ~Stack()

        {

            if (top)

                delete top;

        }

};

void menu();

int main()

{

    menu();

    return 0;

}

void menu()

{

    short size;

    cout << "Enter the size of stack: ";

    cin >> size;

    Stack stack1(size);

    short option;

    int item;

    do

    {

        cout << "\n-> 1. Push\n-> 2. Pop\n-> 3. Display\n-> 0. Exit\n-> ";

        cin >> option;

        switch (option)

        {

            case 1:

                cout << "Enter item to push: ";

                cin >> item;

                stack1.push(item);

            break;

            case 2:

                item = stack1.pop();

                if (item)

                    cout << "Deleted Item: " << item << endl;

            break;

            case 3:

                stack1.display();

            break;

            case 0:

            break;

            default:

                cout << "Wrong choice!" << endl;

        }

    } while (option);

}

**Output :**

**Enter the size of stack: 5**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 1**

**Enter item to push: 6**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 2**

**Deleted Item: 6**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 3**

**Stack is empty!**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 1**

**Enter item to push: 8**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 1**

**Enter item to push: 5**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 1**

**Enter item to push: 2**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**-> 3**

**TOP -> -> 2**

**-> 5**

**-> 8**

**-> 1. Push**

**-> 2. Pop**

**-> 3. Display**

**-> 0. Exit**

**Factorial using Recursion**

#include <iostream>

using namespace std;

// Get User Input

int input()

{

    int n;

    cout << "Enter The Number";

    cin >> n;

    return n;

}

// factorial function return all the factorial value until the limit

int factorial(int n)

{

   if (n < 2)

   {

       return 1;

   }

   else

   {

       return n\*factorial(n-1);

   }

}

int main() {

   int n = input();

   cout << "Factorial of " << n << " is " << factorial(n);

   return 0;

}

**Output :**

**Factorial of 7 is 5040**

**Fibonacci using Recursion**

#include <iostream>

using namespace std;

int fibonacci(int num)

{

   if((num==1)||(num==0))

   {

      return(num);

   }

   else

   {

      return(fibonacci(num-1)+fibonacci(num-2));

   }

}

int main()

{

   int limit, i = 0;

   cout << "Enter the limit for Fibonacci : ";

   cin >> limit;

   cout << "\nFibonnaci Series : ";

   while( i < limit ) {

      cout << " " << fibonacci(i);

      i++;

   }

   return 0;

}

**Output :**

**Enter the limit for Fibonacci : 6**

**Fibonnaci Series : 0 1 1 2 3 5**

**Infix To Postfix**

#include<iostream>

#include <stack>

using namespace std;

int prec(char c)

{

    if(c == '^')

    return 3;

    else if(c == '\*' || c == '/')

    return 2;

    else if(c == '+' || c == '-')

    return 1;

    else

    return -1;

}

void infixToPostfix(string s)

{

    std::stack<char> st;

    st.push('N');

    int l = s.length();

    string ns;

    for(int i = 0; i < l; i++)

    {

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

            ns+=s[i];

        else if(s[i] == '(')

            st.push('(');

        else if(s[i] == ')')

        {

            while(st.top() != 'N' && st.top() != '(')

            {

                char c = st.top();

                st.pop();

               ns += c;

            }

            if(st.top() == '(')

            {

                char c = st.top();

                st.pop();

            }

        }

        else

        {

            while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))

            {

                char c = st.top();

                st.pop();

                ns += c;

            }

            st.push(s[i]);

        }

    }

    while(st.top() != 'N')

    {

        char c = st.top();

        st.pop();

        ns += c;

    }

    cout << ns << endl;

}

int main()

{

    string exp = "x-y\*(a+b+c+d)";

    infixToPostfix(exp);

    return 0;

}

**Output :**

**xyab+c+d+\*-**

**Postfix Evaluation**

#include <iostream>

#include <string.h>

using namespace std;

struct Stack{

    int top;

    unsigned capacity;

    int\* array;

};

struct Stack\* createStack( unsigned capacity ){

    struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));

    if(!stack){

        return NULL;

    }

    stack->top = -1;

    stack->capacity = capacity;

    stack->array = (int\*) malloc(stack->capacity \* sizeof(int));

    if(!stack->array){

        return NULL;

    }

    return stack;

}

int isEmpty(struct Stack\* stack){

    return stack->top == -1 ;

}

char peek(struct Stack\* stack){

    return stack->array[stack->top];

}

char pop(struct Stack\* stack){

    if(!isEmpty(stack)){

        return stack->array[stack->top--];

    }

    return '$';

}

void push(struct Stack\* stack, char op){

    stack->array[++stack->top] = op;

}

int evaluatePostfix(char\* exp){

    struct Stack\* stack = createStack(strlen(exp));

    int i;

    if(!stack){

        return -1;

    }

    for(i = 0; exp[i]; ++i){

        if(isdigit(exp[i])){

            push(stack, exp[i] - '0');

        }

        else{

            int val1 = pop(stack);

            int val2 = pop(stack);

            switch(exp[i]){

                case '+': push(stack, val2 + val1); break;

                case '-': push(stack, val2 - val1); break;

                case '\*': push(stack, val2 \* val1); break;

                case '/': push(stack, val2/val1); break;

            }

        }

    }

    return pop(stack);

}

int main(){

    char s[] = "1234+5+6+7+\*-";

    cout<<evaluatePostfix(s);

    return 0;

}

**Output :**

**-49**