void infixToPostfix(char \*infix,char\* postfix){

    int i=0,j=0;

    char ch;

    while((ch=infix[i++])!='\0'){

        if(isdigit(ch)){

            postfix[j++]=ch;

        }

        else if(ch== '('){

            pushChar(ch);

        }

        else if(ch==')'){

            while(!isEmptyChar() && peekChar()!='('){

                postfix[j++]=popChar();

            }

            popChar();

        }

        else if(isOperator(ch)){

            while(!isEmptyChar() && precedence(peekChar())>=precedence(ch)){

                postfix[j++]=popChar();

            }

            pushChar(ch);

        }

    }

    while(!isEmptyChar()){

        postfix[j++]=popChar();

    }

    postfix[j]='\0';

}

অবশ্যই Arafat ভাই! নিচের infixToPostfix ফাংশনটা ইনফিক্স এক্সপ্রেশনকে পোস্টফিক্স (Reverse Polish Notation) এ কনভার্ট করে। নিচে **line-by-line বাংলায় ব্যাখ্যা** দেওয়া হলো:

**void infixToPostfix(char \*infix, char\* postfix){**

🔹 এই ফাংশনটি ইনফিক্স এক্সপ্রেশন (যেমন: 3+(4\*5)) কে পোস্টফিক্স এক্সপ্রেশন (যেমন: 345\*+) এ রূপান্তর করে।

**int i = 0, j = 0;**

🔹 i ইনফিক্স স্ট্রিং পড়ার জন্য,  
🔹 j পোস্টফিক্স স্ট্রিং লেখার জন্য।

**char ch;**

🔹 প্রতিবার ইনফিক্স এক্সপ্রেশন থেকে একটা ক্যারেক্টার এখানে রাখা হবে।

**while((ch = infix[i++]) != '\0'){**

🔹 যতক্ষণ না ইনফিক্স এক্সপ্রেশনের শেষ ('\0') হয়, ততক্ষণ লুপ চলবে।  
🔹 প্রতি বার infix থেকে একটি ক্যারেক্টার ch তে নেওয়া হচ্ছে।

**✅ এখন প্রতিটি ক্যারেক্টার কেমন তার উপর ভিত্তি করে কাজ হবে:**

**if(isdigit(ch)){**

**postfix[j++] = ch;**

**}**

🔹 যদি ক্যারেক্টারটি একটি **digit (সংখ্যা)** হয় (যেমন: 3, 7), তাহলে সেটা সরাসরি পোস্টফিক্স স্ট্রিংয়ে বসানো হয়।

**else if(ch == '('){**

**pushChar(ch);**

**}**

🔹 যদি ক্যারেক্টারটি **left bracket (** হয়, তাহলে তাকে **stack-এ রেখে দাও** (মানে এখনকার operator এর আগে যেগুলা আছে সেগুলা অপেক্ষা করবে)।

**else if(ch == ')'){**

**while(!isEmptyChar() && peekChar() != '('){**

**postfix[j++] = popChar();**

**}**

**popChar(); // '(' পপ করা হচ্ছে**

**}**

🔹 যদি **) অর্থাৎ right bracket** পাওয়া যায়,  
🔹 তাহলে stack থেকে operator গুলো বের করে পোস্টফিক্সে বসাও যতক্ষণ না ( পাওয়া যায়।  
🔹 শেষে ( ব্র্যাকেটটা pop করে ফেলা হয় (কিন্তু পোস্টফিক্সে বসানো হয় না)।

c

CopyEdit

else if(isOperator(ch)){

🔹 যদি ক্যারেক্টারটি একটি **operator** হয় (যেমন: +, -, \*, /, ^), তাহলে নিচের শর্ত অনুযায়ী কাজ হবে।

**while(!isEmptyChar() && precedence(peekChar()) >= precedence(ch)){**

**postfix[j++] = popChar();**

**}**

**pushChar(ch);**

**}**

🔹 যতক্ষণ stack এ operator আছে এবং উপরের operator-এর precedence **বেশি বা সমান** বর্তমান operator এর থেকে,  
🔹 ততক্ষণ stack থেকে operator বের করে পোস্টফিক্সে বসাও।

🔹 পরে বর্তমান operator-টিকে stack-এ রাখো।

**🔚 ইনফিক্স স্ট্রিং শেষ হয়ে গেলে:**

c

CopyEdit

**while(!isEmptyChar()){**

**postfix[j++] = popChar();**

**}**

🔹 যত operator এখনো stack-এ আছে (bracket বাদে),  
🔹 সব pop করে পোস্টফিক্সে বসিয়ে দাও।

**postfix[j] = '\0';**

🔹 পোস্টফিক্স স্ট্রিংয়ের শেষে null character বসানো হয়, যেন এটা একটা valid C string হয়।

**✅ উদাহরণ সহ কাজের ব্যাখ্যা:**

ধরি ইনফিক্স এক্সপ্রেশন:

scss

CopyEdit

3+(4\*5)

**Step-by-step**:

| **ক্যারেক্টার** |  | **Stack** |  |  |  |  |  |  | **Postfix** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 |  |  |  |  |  |  |  |  | 3 |
| + |  | + |  |  |  |  |  |  | 3 |
| ( |  | + ( |  |  |  |  |  |  | 3 |
| 4 |  | + ( |  |  |  |  |  |  | 34 |
| \* |  | + ( \* |  |  |  |  |  |  | 34 |
| 5 |  | + ( \* |  |  |  |  |  |  | 345 |
| ) |  | + |  |  |  |  |  |  | 345\* |
| --- শেষে -- |  |  |  |  |  |  |  |  | 345\*+ |

**Final Postfix**: 345\*+

**Infix To Postfix**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<math.h>

struct CharNode{

    char data;

    struct CharNode\* next;

};

struct IntNode{

    int data;

    struct IntNode\* Next;

};

struct CharNode\* opTop=NULL;

struct IntNode\* valTop=NULL;

void pushChar(char val){

    struct CharNode\* newNode=(struct CharNode\*)malloc(sizeof(struct CharNode));

    newNode->data=val;

    newNode->next=opTop;

    opTop=newNode;

}

char popChar(){

    if(opTop==NULL) return '\0';

    struct CharNode\* temp=opTop;

    char val=temp->data;  //delet kora value print korvo

    opTop=opTop->next;

    free(temp);

    return val;

}

char peekChar(){

    if(opTop==NULL) return '\0';

    return opTop->data;

}

int isEmptyChar(){

    return opTop=='\0';

}

void pushInt(int val){

    struct IntNode\* newNode=(struct IntNode\*)malloc(sizeof(struct IntNode));

    newNode->data=val;

    newNode->Next=valTop;

    valTop=newNode;

}

int popInt(){

    if(valTop==NULL) return 0;

    struct IntNode\* temp=valTop;

    int val=temp->data;

    valTop=valTop->Next;

    free(temp);

    return val;

}

int isOperator(char c){

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

}

int precedence(char op){

    if(op=='^') return 3;

    if(op=='\*' || op=='/') return 2;

    if(op=='+' || op=='-') return 1;

    return 0;

}

void infixToPostfix(char \*infix,char\* postfix){

    int i=0,j=0;

    char ch;

    while((ch=infix[i++])!='\0'){

        if(isdigit(ch)){

            postfix[j++]=ch;

        }

        else if(ch== '('){

            pushChar(ch);

        }

        else if(ch==')'){

            while(!isEmptyChar() && peekChar()!='('){

                postfix[j++]=popChar();

            }

            popChar();

        }

        else if(isOperator(ch)){

            while(!isEmptyChar() && precedence(peekChar())>=precedence(ch)){

                postfix[j++]=popChar();

            }

            pushChar(ch);

        }

    }

    while(!isEmptyChar()){

        postfix[j++]=popChar();

    }

    postfix[j]='\0';

}

int evalutePostfix(char\* postfix){

    int i=0;

    char ch;

    while((ch=postfix[i++])!='\0'){

        if(isdigit(ch)){

            pushInt(ch-'0');

        }else{

            int op2=popInt();

            int op1=popInt();

            switch(ch){

                case '+': pushInt(op1+op2);break;

                case '-': pushInt(op1-op2);break;

                case '\*': pushInt(op1\*op2);break;

                case '/': pushInt(op1/op2);break;

                case '^':pushInt((int)pow(op1,op2));break;

            }

        }

    }

    return popInt();

}

int main(){

    char infix[100],postfix[100];

    scanf("%s",infix);

    infixToPostfix(infix,postfix);

    printf("Postfix: %s\n",postfix);

    int result=evalutePostfix(postfix);

    printf("Result: %d\n",result);

}

**#QuicK\_Sort**

#include <iostream>

using namespace std;

// Swap function

void swap(int &a, int &b) {

    int temp = a;

    a = b;

    b = temp;

}

// Partition function

int partition(int arr[], int low, int high) {

    int pivot = arr[high]; // পিভট ধরা হলো শেষ উপাদান

    int i = low - 1;

    for(int j = low; j < high; j++) {

        if(arr[j] < pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]); // পিভটকে সঠিক জায়গায় বসানো

    return i + 1;

}

// Quick Sort function

void quickSort(int arr[], int low, int high) {

    if(low < high) {

        int pi = partition(arr, low, high); // পিভট ঠিক করা

        quickSort(arr, low, pi - 1);  // বাম পাশে sort

        quickSort(arr, pi + 1, high); // ডান পাশে sort

    }

}

int main() {

    int n;

    cout << "Enter number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter elements: ";

    for(int i = 0; i < n; i++)

        cin >> arr[i];

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";

    for(int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}