STACK USING ARRAY

```
#include <iostream>
#include <string>

using namespace std;

struct stack
{
    int s[5];
    int top = -1;
} st;

bool isEmpty()
{
    if (st.top == -1)
        return true;
    else
        return false;
}

bool isFull()
{
    if (st.top == 4)
        return true;
    else
        return false;
}

void push(int val)
{
    if (isFull())
        cout << "Stack Overflow" <<endl;
    else
    {
        st.top++;
        st.s[st.top] = val;
        cout << "value pushed\n";
    }
}
```

STACK USING ARRAY

```
int pop()
{
    if (isEmpty())
    {
        cout << "Stack underflow" << endl;
        return 0;
    }
    else
    {
        int popVal = st.s[st.top];
        st.top--;
        return popVal;
    }
}

int count()
{
    return (st.top + 1);
}

int peek(int pos)
{
    if (isEmpty())
    {
        cout << "Stack underflow!" << endl;
        return 0;
    }
    else
    {
        return st.s[pos];
    }
}

void change(int pos, int val)
{
    st.s[pos] = val;
    cout << "Value at " << pos << " changed to " << val;
```

STACK USING ARRAY

```
}

void display()
{
    if (isEmpty())
        cout << "Stack Underflow\n";
    else
    {
        cout<<"All the values in stack are:\n";

        for (int i = st.top; i >= 0; i--){
            cout<<st.s[i]<<endl;
        }
    }
}

int main()
{
    int option, position, value;

    do
    {
        cout << "\n\nWhat operation would you like to
perform ?\n";
        cout << "1. push\n";
        cout << "2. pop\n";
        cout << "3. isEmpty\n";
        cout << "4. isFull\n";
        cout << "5. peek\n";
        cout << "6. change\n";
        cout << "7. display\n";
        cout << "8. count\n";
        cout << "9. clear screen\n";

        cout << "Enter option: ";
        cin >> option;

        switch (option)
```

STACK USING ARRAY

```
        {
        case 1:
            cout << "Enter value to push: ";
            cin >> value;
            push(value);
            break;
        case 2:
            cout << "Popped value is: " << pop();
            break;
        case 3:
            if (isEmpty()) {
                cout<<"Stack is empty\n";
            } else {
                cout<<"Stack is not empty!\n";
            }

            break;
        case 4:
            if (isFull()){
                cout<<"Stack is full!\n";
            } else {
                cout<<"Stack is not full\n";
            }
            break;
        case 5:
            cout << "Enter the position you want to peek in
: ";
            cin >> position;
            cout << "Value at " << position << " is " <<
peek(position);
            break;
        case 6:
            cout << "Enter position of item you want to
change : ";
            cin >> position;
            cout << "\nEnter value: ";
            cin >> value;
            change(position, value);
```

STACK USING ARRAY

```
            break;
        case 7:
            cout<<"Display function called:\n";
            display();
            break;
        case 8:
            cout << "Total elements in stack are :" <<
count();
            break;
        case 9:
            system("cls");

        default:
            cout << "Enter proper option\n";
        }
    } while (option <= 9 && option > 0);

    return 0;
}
```

INFIX TO POSTFIX

```
#include<iostream>
#include<ctype.h>

using namespace std;
char stack[20];
int top = -1;

void push(char x){
    stack[++top] = x;
}

char pop(){
    if (top == -1) return -1;
    else return stack[top--];
}

// returns the priority in the form of integer
int priority(char x){
    if (x == '(') return 0;

    if (x == '+' || x == '-') return 1;

    if (x == '*' || x == '/') return 2;

    if (x == '^') return 3;
return -1;
}

// (2*3+4*(5-6))
int main(){
    char exp[20];
    char *e, x;

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

    e = exp;
```

INFIX TO POSTFIX

```
    while(*e != '\0'){
        if(isalnum(*e)){
            printf("%c", *e);
        }
        else if (*e == '(')
        {
            push(*e);
        }
        else if(*e == ')'){
            while ((x = pop()) != '(')
            {
                printf("%c", x);
            }
        }
        else{
            while(priority(stack[top]) >= priority(*e)){
                printf("%c", pop());
            }
            push(*e);
        }
        e++;
    }
    while (top != -1)
    {
        printf("%c", pop());
    }
}
```

INFIX TO PREFIX

```
#include <iostream>
#include <stack>
#include <algorithm>

using namespace std;

bool isOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
        return true;
    }
    else {
        return false;
    }
}

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

string InfixToPrefix(stack<char> s, string infix)
{
    string prefix;
    reverse(infix.begin(), infix.end());

    //exchanging '(' with ')' and vice-versa
    for (int i = 0; i < infix.length(); i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
```

INFIX TO PREFIX

```
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }

    // Expression coversion begins
    for (int i = 0; i < infix.length(); i++) {
        if ((infix[i] >= 'a' && infix[i] <= 'z') ||
(infix[i] >= 'A' && infix[i] <= 'Z')) {
            prefix += infix[i];
        }
        else if (infix[i] == '(') {
            s.push(infix[i]);
        }
        else if (infix[i] == ')') {
            while ((s.top() != '(') && (!s.empty())) {
                prefix += s.top();
                s.pop();
            }

            if (s.top() == '(') {
                s.pop();
            }
        }
        else if (isOperator(infix[i])) {
            if (s.empty()) {
                s.push(infix[i]);
            }
            else {
                if (priority(infix[i]) > priority(s.top()))
{
                    s.push(infix[i]);
                }
                else if ((priority(infix[i]) ==
priority(s.top()))
                    && (infix[i] == '^')) {
```

INFIX TO PREFIX

```
                    while ((priority(infix[i]) ==
priority(s.top()))
                        && (infix[i] == '^')) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
                else if (priority(infix[i]) ==
priority(s.top())) {
                    s.push(infix[i]);
                }
                else {
                    while ((!s.empty()) &&
(priority(infix[i]) < priority(s.top()))) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
            }
        }
    }

    while (!s.empty()) {
        prefix += s.top();
        s.pop();
    }

    reverse(prefix.begin(), prefix.end());
    return prefix;
}

int main()
{

    string infix, prefix;
    cout << "Enter a Infix Expression :" << endl;
```

INFIX TO PREFIX

```
    cin >> infix;
    stack<char> stack;
    cout << "INFIX EXPRESSION: " << infix << endl;
    prefix = InfixToPrefix(stack, infix);
    cout << endl
         << "PREFIX EXPRESSION: " << prefix;

    return 0;
}
```

POSTFIX

```
#include<iostream>
#include<string>
#include<stack>
#include<math.h>
using namespace std;

int postfixEvaluation(string s){
    stack<int> st;
    int op2, op1;

    for (int i = 0; i < s.length(); i++){
        if (s[i] >= '0' && s[i] <= '9'){
            st.push(s[i] - '0');
        }
        else{
            op2 = st.top();
            st.pop();
            op1 = st.top();
            st.pop();

            switch (s[i])
            {
            case '+':
                st.push(op1 + op2);
                break;
            case '-':
                st.push(op1 - op2);
                break;
            case '*':
                st.push(op1 * op2);
                break;
            case '/':
                st.push(op1 / op2);
                break;
            case '^':
                st.push(pow(op1, op2));
                break;
```

POSTFIX

```
            default:
                break;
            }
        }
    }

    return st.top();
}

int main(){
    string exp;
    cout<<"Enter the postfix expression: ";
    cin>>exp;
    cout<<"Evaluated value: "<<postfixEvaluation(exp);

    return 0;
}
```

PREFIX

```
#include <iostream>
#include <string>
#include <stack>
#include <math.h>
#include <algorithm>
using namespace std;

int prefixEvaluation(string s)
{
    int op1, op2;
    stack<int> st;

    reverse(s.begin(), s.end());

    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] >= '0' && s[i] <= '9')
        {
            st.push(s[i] - '0');
        }
        else
        {
            op2 = st.top();
            st.pop();
            op1 = st.top();
            st.pop();

            switch (s[i])
            {
            case '+':
                st.push(op2 + op1);
                break;
            case '-':
                st.push(op2 - op1);
                break;
            case '*':
                st.push(op2 * op1);
                break;
```

PREFIX

```
            case '/':
                st.push(op2 / op1);
                break;
            case '^':
                st.push(pow(op2, op1));
                break;

            default:
                break;
            }
        }
    }
    return st.top();
}

int main()
{
    string exp;
    cout<<"Enter the prefix expression: ";
    cin>>exp;

    cout<<"Evaluated value: "<<prefixEvaluation(exp);
    return 0;
}
```

POSTFIX TO PREFIX

```
#include<iostream>
#include<stack>
#include<ctype.h>
using namespace std;


string postfixToPrefix(string postfix){
    stack<string> st;
    string op1, op2;

    for (int i = 0, len = postfix.length(); i < len; i++){
        if (isalnum(postfix[i])){
            string op (1, postfix[i]);
            st.push(op);
        }
        else {
            op2 = st.top();
            st.pop();

            op1 = st.top();
            st.pop();

            st.push(postfix[i] + op1 + op2);
        }
    }

    return st.top();
}


int main(){
    string postfix, prefix;
    cout<<"Enter POSTFIX expression: ";
    cin>>postfix;
    cout<<"POSTFIX expression: "<<postfix<<endl;

    cout<<"PREFIX expression: ";
    cout<<postfixToPrefix(postfix);
}
```

PREFIX TO POSTFIX

```
#include <iostream>
#include <math.h>
#include <stack>
#include <ctype.h>
using namespace std;

string prefixToPostfix(string prefix)
{
    stack< string > st;

    string op1, op2;

    for (int i = prefix.length() - 1; i >= 0; i--)
    {
        if (isalnum(prefix[i]))
        {
            string op(1, prefix[i]);
            st.push(op);
        }
        else
        {
            op1 = st.top();
            st.pop();
            op2 = st.top();
            st.pop();

            st.push(op1 + op2 + prefix[i]);
        }
    }
    return st.top();
}

int main()
{
    string prefix, postfix;
    cout << "Enter a PREFIX expression: ";
    cin >> prefix;
    cout << "PREFIX expression: \n"<< prefix;
```

PREFIX TO POSTFIX

```
    postfix = prefixToPostfix(prefix);
    cout << "POSTFIX expresion: " << postfix;
    return 0;
}
```