

```

#include <iostream>
#include <cstring>
using namespace std;

class stack{

    char op;//Operator - For conversions
    char num;//Number - For evaluations

    string output = "";//Result

    stack* link;
    stack* top;

public:

    stack(){

        top = NULL;

    }

    void push(char);//Conversion
    void push(int);//Evaluation
    void pop();
    int priority(char);
    void checkempty();
    void postfix(char []);
    void prefix(char []);
    void evalpost(char []);
    void evalpre(char []);

};

void stack::push(char c){

    stack* n = new stack();

    n->op = c;
    n->link = top;

    top = n;

}

void stack::push(int v){

    stack* n = new stack();

    n->num = v;

```

```
n->link = top;
```

```
top = n;
```

```
}
```

```
void stack::pop(){
```

```
    top = top->link;
```

```
}
```

```
int stack::priority(char c){
```

```
    if(c == '^'){
```

```
        return 3;
```

```
    }else if(c == '/' || c == '*'){
```

```
        return 2;
```

```
    }else if(c == '+' || c == '-'){
```

```
        return 1;
```

```
    }else {
```

```
        return 0;
```

```
    }
```

```
}
```

```
void stack::checkempty(){
```

```
    if(top == NULL){
```

```
        cout<<"Stack is empty."<<endl;
```

```
    }else{
```

```
        pop();
```

```
    }
```

```
}
```

```

void stack::postfix(char infix[]){

    int i = 0;

    //Checking from left to right
    while(infix[i] != '\0'){

        if(isalnum(infix[i])){

            output = output + infix[i];

        }else{

            if(top == NULL || infix[i] == '('){

                push(infix[i]);

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

                while(top->op != '('){

                    output = output + top->op;
                    pop();

                }

                pop();

            }else{

                int x = priority(infix[i]);
                int y = priority(top->op);

                if(x <= y && top != NULL){

                    while(top != NULL){

                        output = output + top->op;
                        pop();

                    }

                    push(infix[i]);

                }else{

                    push(infix[i]);

                }

            }

        }

    }

}

```

```

        }

    }

    i++;

} // End of checking

//Final output
while(top != NULL){

    output = output + top->op;
    pop();

}

cout<<output;

}

void stack::prefix(char infix[]){

    int i = 0, count = 0;

    char rev[10];

    //Reversing the string
    while(infix[count] != '\0'){

        count++;

    }

    //cout<<count;

    for(int j = 0; j < count; j++){

        rev[j] = infix[(count - 1) - j];

        if(rev[j] == '('){

            rev[j] = ')';

        }else if(rev[j] == ' '){

            rev[j] = '(';

        }

    }

```

```

    //cout<<rev[j];
}

//Checking from left to right
while(rev[i] != '\0'){

    if(isalnum(rev[i])){

        output = output + rev[i];

    }else{

        if(top == NULL || rev[i] == '('){

            push(rev[i]);

        }else if(rev[i] == '){

            while(top->op != '('){

                output = output + top->op;
                pop();

            }

            pop();

        }else{

            int x = priority(rev[i]);
            int y = priority(top->op);

            if(x <= y && top != NULL){

                while(top != NULL){

                    output = output + top->op;
                    pop();

                }

                push(rev[i]);

            }else{

                push(rev[i]);

            }

        }

    }

}

```

```

    }

}

    i++;

} // End of checking

//Final output
while(top != NULL){

    output = output + top->op;
    pop();

}

//cout<<output<<endl;

//Reversing back
char revback[15];
strcpy(revback, output.c_str());

//Checking count of elements and replacing brackets when possible
count = 0;
while(revback[count] != '\0'){

    count++;

}

//Final output
for(i = count-1 ; i >= 0; i-- ){

    cout<<revback[i];

}
cout<<endl;

}

void stack::evalpost(char postfix[]){

    int v;//Value

    int x, y, ans = 0;

    for(int i = 0; postfix[i] != '\0'; i++){

        if(isalpha(postfix[i])){

```

```

    cout<<"\nEnter the value of "<<postfix[i]<<" = ";
    cin>>v;

    push(v);

}else{

    x = top->num;
    pop();
    y = top->num;
    pop();

    if(postfix[i] == '+'){

        ans = y + x;
        push(ans);

    }else if(postfix[i] == '-'){

        ans = y - x;
        push(ans);

    }else if(postfix[i] == '*'){

        ans = y * x;
        push(ans);

    }else if(postfix[i] == '/'){

        ans = y / x;
        push(ans);

    }

}

}

cout<<"\nAnswer is = "<<ans;

}

void stack::evalpre(char prefix[]){

    int v;//Value

    int x, y, ans, count = 0;

    //Check end of array
    while(prefix[count] != '\0'){

```

```

    count++;

}

for(int i = count-1; i >= 0; i--){

    if(isalpha(prefix[i])){

        cout<<"\nEnter the value of "<<prefix[i]<<" = ";
        cin>>v;

        push(v);

    }else{

        x = top->num;
        pop();
        y = top->num;
        pop();

        if(prefix[i] == '+'){

            ans = x + y;
            push(ans);

        }else if(prefix[i] == '-'){

            ans = x - y;
            push(ans);

        }else if(prefix[i] == '*'){

            ans = x * y;
            push(ans);

        }else if(prefix[i] == '/'){

            ans = x / y;
            push(ans);

        }

    }

}

cout<<"\nAnswer is = "<<ans;

}

```



```

int main()
{

    stack s1, s2, s3, s4;

    char infix[10];
    int choice = 0;

    while(choice != 5){

        cout<<"\nChoose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-
Exit"<<endl;
        cout<<"Type your choice = ";
        cin>>choice;
        cout<<endl;

        switch(choice){

            case 1:
                cout<<"Infix to Postfix"<<endl;
                cout<<"Enter the expression = ";
                cin>>infix;

                s1.postfix(infix);

                break;

            case 2:
                cout<<"Infix to Prefix"<<endl;
                cout<<"Enter the expression = ";
                cin>>infix;

                s2.prefix(infix);

                break;

            case 3:
                cout<<"Postfix Eval"<<endl;
                cout<<"Enter the expression in alphabets = ";
                cin>>infix;

                s3.evalpost(infix);

                break;

            case 4:
                cout<<"Prefix Eval"<<endl;
                cout<<"Enter the expression in alphabets = ";
                cin>>infix;

```

```

        s4.evalpre(infix);

        break;

    case 5:
        cout<<"Program Terminated."<<endl;
        break;

    default:
        cout<<"Retry";

    }

}

return 0;
}

```

Output

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 1

Infix to Postfix

Enter the expression = a+b*c

abc*+

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 2

Infix to Prefix

Enter the expression = a+b*c

❖+a*bc

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 3

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 3

Postfix Eval

Enter the expression in alphabets = $ab+cd^*$

Enter the value of a = 1

Enter the value of b = 2

Enter the value of c = 3

Enter the value of d = 4

Answer is = 12

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 4

Prefix Eval

Enter the expression in alphabets = $*+ab-cd$

Enter the value of d = 1

Enter the value of c = 3

Enter the value of b = 9

Enter the value of a = 6

Answer is = 30

Choose 1-Infix to Postfix, 2-Infix to Prefix, 3-Postfix Eval, 4-Prefix Eval, 5-Exit

Type your choice = 5

Program Terminated.