# Prefix to Infix conversion

String prefix = +-*AB+CDE → prefix → input
output → A*B-(C+D)+E → infix → output

A, B, C, D, E → operand

**Step1 →** Read the prefix expression from right to left, and create the stack.

```
Stack<String> stack = new Stack();
for(int i=prefix.length()-1 ; i>=0; i--){
    char ch = prefix.charAt(i);
}
```

**Step2 →** If the incoming symbol is operand then push into the stack.

```
Stack<String> stack = new Stack();
for(int i=prefix.length()-1 ; i>=0 ; i--){
    char ch = prefix.charAt(i); //E
    if(!isOperand(ch)){
        stack.push(ch+"");
    }
}
```

A → Z
a → z
'a' → 97

(A+B)

```
public static boolean isOperand(char ch){
    return (ch>='a' && ch<='z') || (ch>='A' && ch<='Z');
}
```

OR (ch >= 97 && ch <= 122) || (ch >= 65 && ch <= 90)

**Step3 →** If the incoming symbol is operator then pop the two element from stack, create a String by concatenating the two operands and the operator between them.

```
String temp = "(" + op1 + ch + op2 + ")";
```

and then resultant string push into the stack;

```
Stack<String> stack = new Stack<>();
for(int i=prefix.length()-1 ; i>=0; i--){
    char ch= prefix.charAt(i);
    if(!isOperand(ch)){ // A, B, C, D, E
        stack.push(ch);
    }else{
        String v1 = stack.pop();
        String v2 = stack.pop();
        String temp = "(" + v1 + ch + v2 + ")";
        stack.push(temp);
    }}
```

+, -, *, / → operators

```
public static boolean isOperand(char ch){
    return (ch>='a' && ch<='z') ||
           ch>='A' && ch<='Z');
}
```

**Step 4 →** Repeat the above steps until the end of prefix expression.

**Step5 →** At the end stack will have only one String i.e resultant String

return stak.pop();

R → L

i-1
i=8 7 6 5 4 3 2 1 0

(input) prefix → +-*AB+CDE

E D C * B A + - +

```
char ch = prefix.charAt(i);
```

String v1 = stack.pop() → E + (A*B)
String v2 = stack.pop() → D + (C+D)
String temp = "(" + v1 + ch + v2 + ")"; → (C+D)
stack.push(temp);

v1 = ((A*B)-(C+D))
v2 = E

((A*B)-(C+D))+E → ((A*B)-(C+D))+E

Pop(stack.pop());

(((A*B)-(C+D))+E) → final Resultant infix expression

A+B (A+B)
A+B+C+DE

| | |
|---|---|
| AX | |
| CX | B * (A*B) |
| DX | (E+D) * (((A*B)-(C+D))+E) |
| EX | (((A*B)-(C+D))+E) → only one element |

top = 4 3 2 1 0

A*B-(C+D)+E → infix

(((A*B)-(C+D))+E) →
(((A*B)-(C+D))+E) →
(-*AB+CD+E)
+-*AB+CDE → Prefix
+-*AB+CDE

**29-08-2023 covered topic**

1. Prefix to infix conversion

**25-08-2023 topic**

1. Prefix to Postfix conversion
2. Postfix to prefix conversion
3. Postfix to infix conversion

## Java Program for Prefix to Infix conversion →

```
Public class PrefixToInfixConversion{
    public static isOperand(char ch){
        return (ch>='a' && ch<='z') || (ch>='A' && ch<='Z');
    }
    public static String prefixToInfix(String prefix){
        Stack<String> stack = new Stack<>();
        int length= prefix.length();
        for(int i=length-1; i>=0; i--){
            char ch = prefix.charAt(i);
            if(!isOperand(ch)){ → checking incoming character is operand or not
                stack.push(ch+"");
            }else{ // it operator then popping two element from stack.
                String operand1 = stack.pop();
                String operand2 = stack.pop();
                String temp = "(" + operand1 + ch + operand2 + ")";
                stack.push(temp);
            }
        }
        return stack.pop();
    }
    public static void main(String arg[]){
        String prefix = "*-A/BC-/AKL";
        Sop("Prefix :" + prefix);
        String infix = prefixToInfix(prefix);
        Sop("Infix : " + infix);
    }
}
```