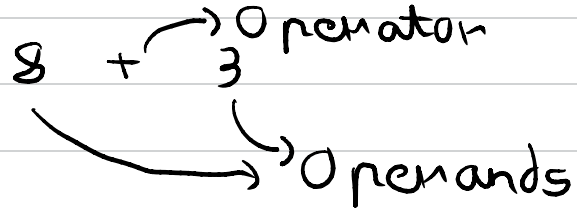
PREFIX, INFIX & POSTFIX

What is operand / operator ?

An operand is an element on which operators act. Eg :



a + b * (c - d)

E | c | A
a
+ | ab
+ | ab
+ * | ab
+ * (| ab
+ * (| abc
+ * (- | abc
+ * (- | abcd
+ * * | abcd -

Operator Priority

1. * * Power
2. * / Multiplication or Division
3. + - Addition or Subtraction

Types of Expressions

Infix : (p + q) * (m - n)
Postfix : pq + mn - *
Prefix : * - pq + mn

Operators b/w operands

Operators after operands

Operators before operands

Infix to Postfix

Eg : a + b * (c ** d - e)

i | st | ans

0	a	-	a
1	+	+	a
2	b	+	ab
3	*	+ *	ab
4	(	+ * (	ab
5	c	+ * (	abc
6	* *	+ * (* *	abc
7	d	+ * (* *	abcd
8	-	+ * (-	abcd**
9	e	+ * (-	abcd**e
10	)	+ *	abcd**e-

Procedure is simple if we follow these rules :

① Create a stack and an answer string
② Iterate through the infix operation
③ If operand is encountered, add it to answer string
④ If operator is encountered, we check priority order. If priority is higher than what it is of last operator in stack, it is added to stack. If priority is less, higher priority operators are popped out and sent to answer string.
⑤ If opening bracket is encountered it is automatically added to stack.
⑥ If closing bracket is encountered everything from the stack is

popped until opening bracket is found. Everything which is popped is also added to answer string

Pseudocode :

```
infixToPostfix(string s) {
    int i = 0;
    Stack<char> st;
    String ans = "";
    while(i < s.size) {
        if(s[i] >= 'A' && s[i] <= 'Z') || (
                s[i] >= 'a' && s[i] <= 'z') ||
            (s[i] >= '0' && s[i] <= '9') {
            ans += s[i];
        } else if (s[i] == '(') {
            st.push(s[i]);
        } else if (s[i] == ')') {
            while ( st.empty() && st.top() !=
                                        '(') {
                ans += st.top();
                st.pop();
            }
            st.pop();
        } else {
            while ( st.empty() && priority(
                        s[i] < priority(st.top()))

                ans += st.top();
                st.pop();
            }
            st.push(s[i]);
        }
        i++;
```

```
}
while(!st.empty()) {
    ans += st.top();
    st.pop();
}
```

Infix to Prefix

Simple Process :

① Reverse given infix

② Convert to postfix by standard procedure but this time operators with similar priority are directly added and nothing is popped.

③ Reverse this back to get

Postfix to Infix

AB-DE+F*/

	i	st
0	A	A
1	B	AB
2	-	(A-B)
3	D	(A-B)D
4	E	(A-B)DE
5	+	(A-B)(D+E)
6	F	(A-B)(D+E)F
7	*	(A-B)((D+E)*F)
8	/	((A-B)/((D+E)*F))

Return st.top.

Procedure is simple, maintain an iterator and a stack. If an operand is spotted, push it to the stack. If an operator is spotted, we take last two elements in the stack, put the operator between them, wrap them in a bracket and add it to the stack as a single element.

Pseudocode :

```
postfixToInfix(String s) {
    Stack <String> St;
    int i = 0;
    while(i < s.length()) {
        if(s[i] >= 'A' && s[i] <= 'Z') || (
                s[i] >= 'a' && s[i] <= 'z') ||
            (s[i] >= '0' && s[i] <= '9') {
            st.push(s[i]);
        } else {
            String t1 = st.top();
            st.pop();
            String t2 = st.top();
            st.pop();
            String c = '(' + t1 + s[i] + t2 + ')';
            st.push(c);
        }
        i++;
    }
    return st.top();
}
```

Prefix to Infix

Literally the same as postfix to infix but instead of iterating from start we iterate from end.

```
prefixToInfix(String s) {
  Stack<String> St;
  int i = s.size();
  while(i >= 0) {
    if((s[i] >= 'A' && s[i] <= 'Z') || (
          s[i] >= 'a' && s[i] <= 'z') ||
        (s[i] >= '0' && s[i] <= '9') {
      st.push(s[i]);
    } else {
      String t1 = st.top();
      st.pop();
      String t2 = st.top();
      st.pop();
      String c = '(' + t1 + s[i] + t2 + ')';
      st.push(c);
    }
    i--;
  }
  return st.top();
}
```

Postfix to Prefix

Obvious approach is to use infix as an intermediate conversion.

Can also do it directly with a similar approach.

Procedure is simple, maintain an iterator and a stack. If an operand is spotted, push it to the stack. If an operator is spotted, we take last two elements in the stack, put the operator before them, add it to the stack as a single element

AB - DE + F * /

	i	st
0	A	A
1	B	A, B
2	-	- AB
3	D	- AB, D
4	E	- AB, D, E
5	+	- AB, + DE
6	F	- AB, + DE, F
7	*	- AB, * + DEF
8	/	/ - AB * + DEF

Pseudocode :

```
postfixToPrefix (String s) {
  Stack <String> St;
  int i = 0;
  while(i < s.size()) {
    if(s[i] >= 'A' && s[i] <= 'Z') || (
          s[i] >= 'a' && s[i] <= 'z') ||
        (s[i] >= '0' && s[i] <= '9') {
      st.push(s[i]);
    } else {
```

```
            String t1 = st.top();
            st.pop();
            String t2 = st.top();
            st.pop();
            String c = s[i] + t2 + t1;
            st.push(c);
        }
        i += 1;
    }
    return st.top();
}
```

Prefix to Postfix Conversion

Exactly the same as its reverse but this time we start iterating from the end and when combined operator goes to the end.

```
prefixToPostfix(String s) {
  Stack<String> st;
  int i = s.size();
  while(i >= 0) {
    if(s[i] >= 'A' && s[i] <= 'Z') || (
            s[i] >= 'a' && s[i] <= 'z') ||
          (s[i] >= '0' && s[i] <= '9') {
      st.push(s[i]);
    } else {
      String t1 = st.top();
      st.pop();
      String t2 = st.top();
      st.pop();
      String c = t1 + t2 + s[i];
```

```
            st.push(c);
        }
        i--;
    }
    return st.top();
}
```