Data Structure and Algorithms

Session-7

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Infix, Prefix and Postfix Notations

- ➢ Let's interpret the troublesome expression A + B * C using operator precedence
- ➢ What will happen with (A + B) * C ?

(p + q) * (r + s)

<operand> <operator> <operand>

- ➢ How do you add A + B + C ?
- Two other very important expression formats
- The infix expression A + B. What would happen if we moved the operator before the two operands? The resulting expression would be + A B(infix)
- Likewise, we could move the operator to the end. We would get A B + (Postfix)

Example:

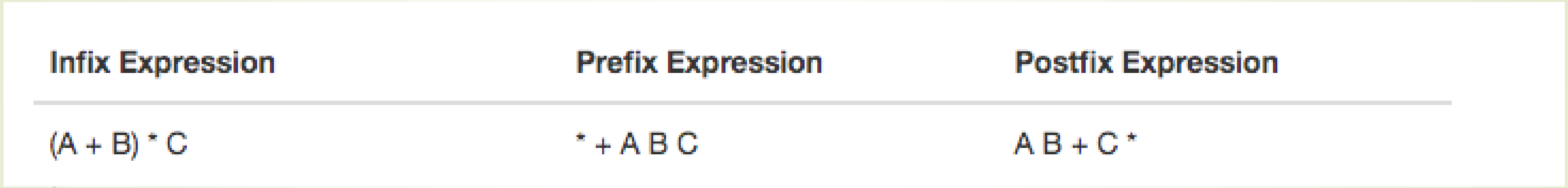
- ➢ **Prefix**
- ➢ A + B * C would be written as + A * B C in prefix

- The multiplication operator comes immediately before the operands B and C, denoting that * has precedence over +
- The addition operator then appears before the A and the result of the multiplication
- Prefix expression notation requires that all operators precede the two operands that they work on

- ➢ **Postfix**
- ➢ the expression would be A B C * +

- The order of operations is preserved since the * appears immediately after the B and the C, denoting that * has precedence
- + comes after
- Postfix, on the other hand, requires that its operators come after the corresponding operands

Now consider the infix expression (A + B) * C.



Infix Expression	Prefix Expression	Postfix Expression
(A + B) * C	* + A B C	A B + C *

- Where did the parenthesis go?
- the operators are no longer ambiguous with respect to the operands that they work on

Let’s see the expression A + B * C
A B C * + is the postfix equivalent

- The operands A, B, and C stay in their relative positions, only the operators that change position
- In the infix expression, the first operator that appears from left to right is +
- In the postfix expression, + is at the end since the next operator, *, has precedence over addition
- The order of the operators in the original expression is reversed in the resulting postfix expression
- As we process the expression, the operators have to be saved somewhere since their corresponding right operands are not seen yet
- Also, the order of these saved operators may need to be reversed due to their precedence

Infix to prefix conversion

- A stack might be used to keep the operators until they are needed
- The top of the stack will always be the most recently saved operator
- Whenever we read a new operator, we will need to consider how that operator compares in precedence with the operators, if any, already on the stack

Precedence Chart

1) (){}[]
2) ^ Associativity R-L
3) */ Associativity L-R
4) +- Associativity L-R

Notes for infix to prefix

•First, reverse the infix expression given in the problem.
•Scan the expression from left to right.
•Whenever the operands arrive, print them.
•If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
•If the incoming operator has higher precedence than the TOP of the stack, push the incoming operator into the stack.
•If the incoming operator has the same precedence with a TOP of the stack, push the incoming operator into the stack.
•If the incoming operator has lower precedence than the TOP of the stack, pop, and print the top of the stack. Test the incoming operator against the top of the stack again and pop the operator from the stack till it finds the operator of a lower precedence or same precedence.
•If the incoming operator has the same precedence with the top of the stack and the incoming operator is ^, then pop the top of the stack till the condition is true. If the condition is not true, push the ^ operator.
•When we reach the end of the expression, pop, and print all the operators from the top of the stack.
•If the operator is ')', then push it into the stack.
•If the operator is '(', then pop all the operators from the stack till it finds) opening bracket in the stack

Infix to prefix Conversion

K+L-M*N+(O^P)*W/U/V*T+Q

Reverse: Q+T*V/U/W*)P^O(+N*M-L+K

Input Expression	Stack	Prefix
Q		Q
+	+	Q
T	+	QT
*	+*	QT
V	+*	QTV
/	+*/	QTV
U	+*/	QTVU
/	+*//	QTVU
W	+*//	QTVUW
*	+*//*	QTVUW

Input Expression	Stack	Prefix
)	+*//*)	QTVUW
P	+*//*)	QTVUWP
^	+*//*)^	QTVUWP
O	+*//*)^	QTVUWPO
(	+*//*	QTVUWPO^
+	++	QTVUWPO^*//*
N	++	QTVUWPO^*//*N
*	++*	QTVUWPO^*//*N
M	++*	QTVUWPO^*//*NM
-	++-	QTVUWPO^*//*NM*
L	++-	QTVUWPO^*//*NM*L
+	++-+	QTVUWPO^*//*NM*L
K	++-+	QTVUWPO^*//*NM*LK
		QTVUWPO^*//*NM*LK+-++

Reverse: Q+T*V/U/W*)P^O(+N*M-L+K

Reverse: ++-+KL*MN*//*^OPWUVTQ

Notes for infix to postfix

1.Print the operand as they arrive.
2.If the stack is empty, push the incoming operator on to the stack.
3.If the incoming symbol is '(', push it on to the stack.
4.If the incoming symbol is ')', pop the stack and print the operators until the left parenthesis is found.
5.If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6.If the incoming symbol has lower precedence than the top of the stack, pop and print the top of the stack. Then test the incoming operator against the new top of the stack.
7.If the incoming operator has the same precedence with the top of the stack then use the associativity rules. If the associativity is from left to right then pop and print the top of the stack then push the incoming operator. If the associativity is from right to left then push the incoming operator.
8.At the end of the expression, pop and print all the operators of the

Infix to Postfix Conversion

K+L-M*N+(O^P)*W/U/V*T+Q

Input Expression	Stack	Postfix
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	-*	KL+M
N	_*	KL+MN
+	+	KL+MN*-
(	+(	KL+MN*-
O	+(	KL+MN*-O
^	+(^	KL+MN*-O
P	+(^	KL+MN*-OP
)	+	KL+MN*-OP^
*	+*	KL+MN*-OP^
W	+*	KL+MN*-OP^W

Input Expression	Stack	Postfix
/	+/	KL+MN*-OP^W*
U	+/	KL+MN*-OP^W*U
/	+/	KL+MN*-OP^W*U/
V	+/	KL+MN*-OP^W*U/V
*	+*	KL+MN*-OP^W*U/V/
T	+*	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*+Q
		KL+MN*-OP^W*U/V/T*+Q+

Thank you