

Lecture No.07

Data Structures

Infix to Postfix

Infix

$A + B$

$12 + 60 - 23$

$(A + B) * (C - D)$

$A \uparrow B * C - D + E / F$

Postfix

$A B +$

$12 60 + 23 -$

$A B + C D - *$

$A B \uparrow C * D - E F / +$

Infix to Postfix

- Note that the postfix form an expression does not require parenthesis.
- Consider '4+3*5' and '(4+3)*5'. The parenthesis are not needed in the first but they are necessary in the second.
- The postfix forms are:

4+3*5

(4+3)*5

435*+

43+5*

Evaluating Postfix

- Each operator in a postfix expression refers to the previous two operands.
- Each time we read an operand, we push it on a stack.
- When we reach an operator, we pop the two operands from the top of the stack, apply the operator and push the result back on the stack.

Evaluating Postfix

```
Stack s;  
while( not end of input ) {  
    e = get next element of input  
    if( e is an operand )  
        s.push( e );  
    else {  
        op2 = s.pop();  
        op1 = s.pop();  
        value = result of applying operator 'e' to op1 and op2;  
        s.push( value );  
    }  
}  
finalresult = s.pop();
```

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input

6

op1

op2

value

stack

6

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input

op1

op2

value

stack

6

6

2

6,2

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
↑	7	2	49	49

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
↑	7	2	49	49
3	7	2	49	49,3

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
↑	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Evaluating Postfix

Evaluate 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

Input	op1	op2	value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
↑	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Converting Infix to Postfix

- Consider the infix expressions ' $A+B*C$ ' and ' $(A+B)*C$ '.
- The postfix versions are ' $ABC*+$ ' and ' $AB+C*$ '.
- The order of operands in postfix is the same as the infix.
- In scanning from left to right, the operand ' A ' can be inserted into postfix expression.

Converting Infix to Postfix

- The '+' cannot be inserted until its second operand has been scanned and inserted.
- The '+' has to be stored away until its proper position is found.
- When 'B' is seen, it is immediately inserted into the postfix expression.
- Can the '+' be inserted now? In the case of 'A+B*C' cannot because * has precedence.

Converting Infix to Postfix

- In case of $(A+B)*C$, the closing parenthesis indicates that $+$ must be performed first.
- Assume the existence of a function $\text{prcd}(\text{op1}, \text{op2})$ where op1 and op2 are two operators.
- $\text{Prcd}(\text{op1}, \text{op2})$ returns TRUE if op1 has precedence over op2 , FALSE otherwise.

Converting Infix to Postfix

- `prcd('*', '+')` is TRUE
- `prcd('+', '+')` is TRUE
- `prcd('+', '*')` is FALSE
- Here is the algorithm that converts infix expression to its postfix form.
- The infix expression is without parenthesis.

Converting Infix to Postfix

```
1.  Stack s;
2.  While( not end of input ) {
3.      c = next input character;
4.      if( c is an operand )
5.          add c to postfix string;
6.      else {
7.          while( !s.empty() && prcd(s.top(),c) ){
8.              op = s.pop();
9.              add op to the postfix string;
10.         }
11.         s.push( c );
12.     }
13.     while( !s.empty() ) {
14.         op = s.pop();
15.         add op to postfix string;
16.     }
```

Converting Infix to Postfix

- Example: $A + B * C$

<u>symb</u>	<u>postfix</u>	<u>stack</u>
A	A	

Converting Infix to Postfix

- Example: $A + B * C$

<u>symb</u>	<u>postfix</u>	<u>stack</u>
A	A	
+	A	+

Converting Infix to Postfix

- Example: $A + B * C$

<u>symb</u>	<u>postfix</u>	<u>stack</u>
A	A	
+	A	+
B	AB	+

Converting Infix to Postfix

- Example: $A + B * C$

<u>symb</u>	<u>postfix</u>	<u>stack</u>
A	A	
+	A	+
B	AB	+
*	AB	+ *

Converting Infix to Postfix

- Example: $A + B * C$

<u>symb</u>	<u>postfix</u>	<u>stack</u>
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *

Converting Infix to Postfix

- Example: $A + B * C$

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC *	+

Converting Infix to Postfix

- Example: $A + B * C$

symb	postfix	stack
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
	ABC *	+
	ABC * +	

Converting Infix to Postfix

- Handling parenthesis
- When an open parenthesis '(' is read, it must be pushed on the stack.
- This can be done by setting `prcd(op, '(')` to be `FALSE`.
- Also, `prcd('(', op) == FALSE` which ensures that an operator after '(' is pushed on the stack.

Converting Infix to Postfix

- When a ')' is read, all operators up to the first '(' must be popped and placed in the postfix string.
- To do this, `prcd(op, ')') == TRUE`.
- Both the '(' and the ')' must be discarded: `prcd('(', ')') == FALSE`.
- Need to change line 11 of the algorithm.

Converting Infix to Postfix

```
if( s.empty() || symb != ')' )  
    s.push( c );  
else  
    s.pop(); // discard the '('
```

<code>prcd('(', op) = FALSE</code>	for any operator
<code>prcd(op, '(') = FALSE</code>	for any operator other than '('
<code>prcd(op, ')') = TRUE</code>	for any operator other than '('
<code>prcd(')', op) = error</code>	for any operator.