DATA STRUCTURES AND ALGORITHMS

Lecture 6: Applications of Stacks

Lecturer: Mohsin Abbas

National University of Modern Languages, Islamabad

ALGEBRAIC EXPRESSIONS

- An algebraic expression is combination of operands and operators.
- Operand is the object of mathematical operation.
  - Quantity that is operated on.
    - For example, Number of boys and girls in a class.
- Operator is a symbol that signifies a mathematical or logical operation.
  - Sum of all students.
  - Comparison between strength of boys and girls in a class.

INFIX, POSTFIX AND PREFIX EXPRESSIONS

- Infix
  - Expressions in which operands surround the operators
  - Example: A+B-C
- Postfix or Reverse Polish Notation (RPN)
  - Operators comes after the operands
  - Example: AB+C-
- Prefix or Polish Notation
  - Operator comes before the operands
  - Example: -+ABC

CONVERSION FROM INFIX TO POSTFIX

- Infix: A+B*C
- Conversion: Applying the rules of precedence
  - Highest Priority: Brackets / Parenthesis ( )
  - 2nd Highest Priority: Multiplication & Division * /
  - Least Priority: Addition & Subtraction + -
- Example
  - A+(B*C) Parenthesis for emphasis
  - A+(BC*) Convert the multiplication
  - ABC*+ Postfix form

CONVERSION FROM INFIX TO POSTFIX

- Infix: ((A+B)*C-(D-E)) $ (F+G)
- Conversion: Applying the rules of precedence
  - ((AB+)*C-(DE-)) $ (FG+)
  - ((AB+C*)-(DE-)) $ (FG+)
  - (AB+C*DE--) $ (FG+)
  - AB+C*DE--FG+$
- Exercise: Convert the following to Postfix
  - (A + B) * (C – D)
  - A / B * C – D + E / F / (G + H)

INFIX, POSTFIX AND PREFIX EXPRESSIONS – EXAMPLES

Infix	Postfix	Prefix
A+B	AB+	+AB
(A+B)*(C+D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	?	?

WHY DO WE NEED PREFIX AND POSTFIX?

- Normally, algebraic expressions are written using Infix notation.
  - For example: $(3 + 4) \times 5 - 6$
- Appearance may be misleading, Infix notations are not as simple as they seem
  - Operator precedence
  - Associativity property
- Operators have precedence: Parentheses are often required
  - $(3 + 4) \times 5 - 6 = 29$
  - $3 + 4 \times 5 - 6 = 17$
  - $3 + 4 \times (5 - 6) = -1$
  - $(3 + 4) \times (5 - 6) = -7$

WHY DO WE NEED PREFIX AND POSTFIX?

- Infix Expression is ***Hard To Parse*** and difficult to evaluate.
- Postfix and prefix do not rely on operator priority and are *easier to parse*.
    - No ambiguity and no brackets are required
- Many compilers first translate algebraic expressions into some form of postfix notation.
    - Afterwards translate this postfix expression into machine code
        - `MOVE.L #$2A, D1   ; Load 42 into Register D1`
        - `MOVE.L #$100, D2  ; Load 256 into Register D2`
        - `ADD D2, D1        ; Add D2 into D1`

EXAMPLE – A + B * C

Symbol	Postfix String	opstck

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+
B	AB	+

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+
B	AB	+
*	AB	+*

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+
B	AB	+
*	AB	+*
C	ABC	+*

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+
B	AB	+
*	AB	+*
C	ABC	+*
	ABC*	+

EXAMPLE – A + B * C

Symbol	Postfix String	opstck
A	A	
+	A	+
B	AB	+
*	AB	+*
C	ABC	+*
	ABC*	+
	ABC*+	

RULES FOR INFIX TO POSTFIX CONVERSION

- Token is an operand
  - Append it to the end of postfix string
- Token is a left parenthesis
  - Push it on the opstck
- Token is a right parenthesis
  - Pop the opstck until the corresponding left parenthesis is removed
  - Append each operator to the end of the postfix string
- Token is an operator, *, /, +, or –
  - Push it on the opstck
  - First remove any operators already on the opstck that have higher or equal precedence and append them to the postfix string
- Input expression has been completely processed
  - Any operators still on the opstck can be removed and appended to the end of the postfix string

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+
B	AB	(+

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*

EXAMPLE – (A + B) * C

Symbol	Postfix String	opstck
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*
	AB+C*	

CONVERSION OF INFIX TO POSTFIX – PRACTICE

- Example: ((A-(B+C))*D) $ (E+F)

Symbol	Postfix String	opstck

CONVERSION OF INFIX TO POSTFIX – PRACTICE

- Example: ((A-(B+C))*D) $ (E+F)

Symbol	Postfix String	opstck
(		(
(		((
A	A	((
-	A	((-
(	A	((-(
B	AB	((-(
+	AB	((-(+
C	ABC	((-(+
)	ABC+	((-
)	ABC+-	(
*	ABC+-	(*
D	ABC+-D	(*
)	ABC+-D*	
$	ABC+-D*	$
(	ABC+-D*	$(
E	ABC+-D*E	$(
+	ABC+-D*E	$(+
F	ABC+-D*EF	$(+
)	ABC+-D*EF+	$
	ABC+-D*EF+$	

CONVERSION TO PREFIX EXPRESSION

- An Infix to Prefix Conversion Algorithm
  - Reverse the infix string
    - Adjust parenthesis, i.e., make every '(' as ')' and every ')' as '('
  - Perform infix to postfix algorithm on reversed string
  - Reverse the output postfix expression to get the prefix expression
- Example: `(A + B) * (B – C)`
  - `)C – B(*)B + A(` → **`(C – B) * (B + A)`** Reverse infix string
  - `C B - B A + *` Perform infix to postfix conversion
  - `* + A B - B C` Reverse postfix to get prefix expression

CONVERSION TO PREFIX EXPRESSION

- Example: (A+B^C)*D+E^5
  - 5^E+D*)C^B+A(→ **5^E+D*(C^B+A)** Reverse infix string
  - 5E^DCB^A+*+ Perform infix to postfix conversion
  - +*+A^BCD^E5 Reverse postfix to get prefix expression

EVALUATING POSTFIX STRING

For evaluation, four stacks are required, opr1, opr2, value, finalString. Following rules are applicable in evaluation process.

- Token is an operand
  - Append it to the end of final string stack
- Token is an operator, *, /, +, - or others
  - Remove the two latest operands from final string stack.
  - Place the recent operand into opr2 and following operand into opr1.
  - Perform action according to operator and store the result in value stack.
  - Append the result stored in value to the end of final string stack
- Repeat the process until all the symbols are checked and evaluated.

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2				7, 2

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2				7, 2
$	7	2	49	49

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2				7, 2
$	7	2	49	49
3				49, 3

EXAMPLE – 6 2 3 + - 3 8 2 / + * 2 $ 3 +

Symbol	opr1	opr2	value	finalString
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2				7, 2
$	7	2	49	49
3				49, 3
+	49	3	52	**52**

CONCLUSION

- In this lecture we have studied:
  - Stack Applications
  - Infix, Prefix and Postfix Notation
  - Conversion of Infix expression to Postfix Expression
  - Conversion of Infix expression to Prefix Expression
  - Evaluation of Postfix String

Question?