$A/B - C + D * E - A * C =?$

3.6

Evaluation of Expressions

3.6.1

Regular Expression

$X = A/B - C + D * E - A * C$

- Operators
  - +,-,*,/,…,etc
- Operands
  - A,B,C,D,E,F

Expression Evaluation

- For $X = A/B - C + D * E - A * C$
- If A = 4, B=C=2, D=E=3

- For $X = ((A/B) - C) + (D * E) - (A * C)$
- X = ((4/2)-2)+(3*3)-(4*2)=1

- For $X = (A/(B - C + D)) * (E - A) * C$
- X = (4/(2-2+3))*(3-4)*2 = -2.6666666

Evaluation Rules

- Operators have **priority**
- Operator with **higher priority** is evaluated first
- Operators of **equal priority** are evaluated from **left to right**
- **Unary** operators are evaluated from **right to left**

28

Priority of Operators in CPP

Priority	Operators
1	Minus, !
2	*, /, %
3	+, -
4	<, <=, >=, >
5	==, !=
6	&&
7	||

29

3.6.2

Infix and Postfix Notation

- **Infix** notation (中序式)
  - Operator comes in–between the operands
  - Ex. A+B*C
  - Hard to evaluate using code…
- **Postfix** notation (後序式)
  - Each operator appears after its operands
  - Ex. ABC*+

30

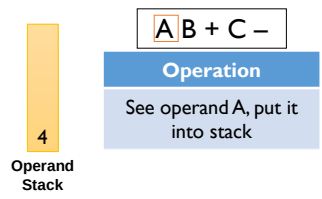
Advantages of Postfix Notation

- You don't need **parentheses**
- Priority of operators is no longer relevant!
- Expression can be efficiently evaluated by
  - Making a left to right scan
  - **Stacking operands**
  - **Evaluating operators**
  - **Push** the **result** into stack

31

Example I

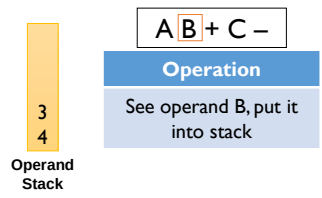
- Infix: A+B – C => Postfix: A B + C –
- Suppose A = 4, B = 3, C = 2



32

Example I

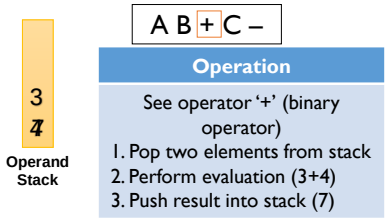
- Infix :A+B – C => Postfix :A B + C –
- Suppose A = 4, B = 3, C = 2



33

Example I

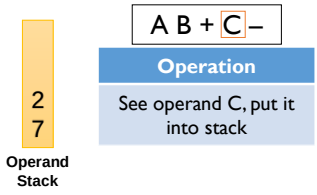
- Infix : A+B – C => Postfix : A B + C –
- Suppose A = 4, B = 3, C = 2



34

Example I

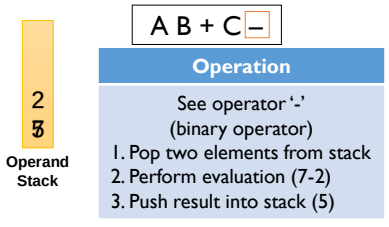
- Infix : A+B – C => Postfix : A B + C –
- Suppose A = 4, B = 3, C = 2



35

Example I

- Infix : A+B – C => Postfix : A B + C –
- Suppose A = 4, B = 3, C = 2



36

Example 2

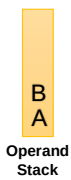
- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

Operation
See operand A, put it into stack

37

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

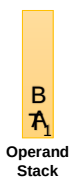


Operation
See operand B, put it into stack

38

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

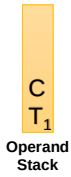


Operation
See operator '/' 1. Pop two elements from stack 2. Perform evaluation (T_1=A/B) 3. Push result into stack (T_1)

39

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

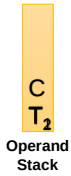


Operation
See operand C, put it into stack

40

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

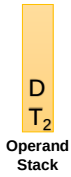


Operation
See operator '-' 1. Pop two elements from stack 2. Perform evaluation (T_2=T_1-C) 3. Push result into stack (T_2)

41

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$



Operation
See operand D, put it into stack

42

Example 2

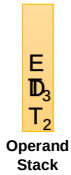
- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

Operation
See operand E, put it into stack

43

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$

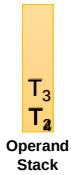


Operation
See operator '*' 1. Pop two elements from stack 2. Perform evaluation (T_3=D*E) 3. Push result into stack (T_3)

44

Example 2

- Infix: $X = A/B - C + D * E - A * C$
- Postfix: $X = AB/C - DE * +AC * -$



Operation
See operator '+' 1. Pop two elements from stack 2. Perform evaluation (T_4=T_2+T_3) 3. Push result into stack (T_4)

Try the rest of steps yourself!

45

Evaluation Pseudo Code

```
void Eval(Expression e)
{  // Assume the last token of e is '#'
   // A function NextToken is used to get next token in e
   Stack<Token> stack; // initialize stack
   for (Token x = NextToken(e); x != '#'; x = NextToken(e)){
     if(x is an operand) stack.Push(x);
     else{
         // Remove the correct number of operands from stack
         // Perform the evaluation
         // Push the result back to stack
         // ***Try to fill up the code ***
     }
   }
};
```

46

Infix to Postfix

- Fully parenthesize algorithm:
  - Fully parenthesize the expression
  - Move all operators so the they replace the corresponding right parentheses
  - Delete all parentheses

((((A / B) – C) + (D * E)) - (A * C))

A B / C – D E * + A C * -

47

Smarter Infix to Postfix Algorithm

- Utilize **stack**
- Scan the expression only once
- The order of operands does not change between infix and postfix
  - Output every visiting operand directly
- Use stack to store visited operators and pop them out at the proper sequence
  - When the ***priority*** of the operator on top of stack is ***higher or equal to*** that of the incoming operator (left-to-right associativity)

48

Example 1

- Infix: A + B * C

Next token	Stack	Output
None	Empty	None
A	Empty	A
+	+	A
B	+	AB
*	+*	AB
C	+*	ABC
	+	ABC*
	Empty	ABC*+

49

Example 2

- Infix: A * (B + C) * D

Next token	Stack	Output
None	Empty	None
A	Empty	A
*	*	A
(	*(	A
B	*(	AB
+	*(+	AB
C	*(+	ABC
)	*	ABC+
*	*	ABC+*
D	*	ABC+*D
	Empty	ABC+*D*

50

Notes: Expression with ( )

- '(' has the highest priority, always push to stack.
- Once pushed, '(' get the lowest priority.
- ')' has the lowest priority, therefore pop the operators in the stack until you see the matched '(', then eliminate both.

51

Postfix Pseudo Code

```
void Postfix(Expression e)
{  // Assume the last token of e is '#'
   // A function NextToken is used to get next token in e
   Stack<Token> stack; // initialize stack
   for (Token x = NextToken(e); x != '#'; x = NextToken(e)){
     if(x is an operand) cout << x;
     else if (x == ')'){ // pop until '('
       for(; stack.Top()!='('; stack.Pop()) cout<<stack.Top();
       stack.Top(); // pop '('
     }
     else{ // x is an operator
       for(;icp(stack.Top()) <= icp(x);stack.Pop())
           cout<<stack.Top();
       stack.Push(x);
     }
   }
   // end of expression; empty the stack
   for(;!stack.IsEmpty(); cout << stack.Top(), stack.Pop());
};
```

52