3.6 Evaluation of Expressions

3.6.2 Postfix Notation

- Compiler
  - Translates an expression into a sequence of machine codes
  - It first re-writes the expression into a form called postfix notation.
- Infix
  - The operators come in-between the operands
- Postfix
  - The operators appear after its operands
- Example
  - Infix: A / B – C + D * E – A * C
  - Postfix: A B / C – D E * + A C * –

3.6 Evaluation of Expressions

- 假設每次運算的結果都暫存在 T_i 裡面，則 A B / C – D E * + A C * – 運算結果如下：

operation	postfix
$T_1 = A / B$	$T_1 C - DE * + AC * -$
$T_2 = T_1 - C$	$T_2 DE * + AC * -$
$T_3 = D * E$	$T_2 T_3 + AC * -$
$T_4 = T_2 + T_3$	$T_4 AC * -$
$T_5 = A * C$	$T_4 T_5 -$
$T_6 = T_4 - T_5$	T_6

(A / B) – (C + D) * (E – A) * C 之 postfix form 為
A B / C D + E A – * C * –

3.6 Evaluation of Expressions

- 用 postfix 來計算簡單很多，不需括號、不需用到優先順序，從左到右掃過，使用 stack 即可。

```
void  eval(expression e) {
   Stack <token> stack;
   for(token x = NextToken(e); x != '#'; x = Nexttoken(e))
      if(x is an operand) stack.Add(x); //運算元放到 stack 中
      else { //拿出正確的運算元，將結果放回 stack
         remove the correct number of operands for
           operator x from stack; perform the operation x
           and store the result(if any) onto the stack;
      }
} // Program 3.18: Evaluating postfix expressions
```

3.6 Evaluation of Expressions

3.6.3 Infix to Postfix

- 將 infix 型式的表示式轉換成執行順序相同的 postfix 表示式。

  1. Fully parenthesize the expression.
  2. Move all operators so that they replace their corresponding right parentheses.
  3. Delete all parentheses.

- 例如：A / B – C + D * E – A * C 的 fully parenthesize 為
  ((((A / B) – C) + (D * E)) – (A * C))
- 將每個括號裡的部份都改成 postfix，即為

  AB/ C– DE* + AC* –

3.6 Evaluation of Expressions

- 將 infix　A + B * C 轉成 postfix　A B C * +

next token	stack	output	
none	empty	none	
A	empty	A	
+	+	A	
B	+	AB	
*	+*	AB	//因 * 優先順序比 + 高
C	+*	ABC	

- 最後再將 stack 裡的資料輸出，結果：A B C * +

3.6 Evaluation of Expressions

- 將 infix A * (B + C) * D 轉成 postfix ABC + * D *

next token	stack	output	
none	empty	none	
A	empty	A	
*	*	A	
(	*(	A	// (的 icp 比 * 的 isp 高
B	*(	AB	
+	*(+	AB	// + 的 icp 比 (的 isp 高
C	*(+	ABC	
)	*	ABC+	// (與) 之間的部份全輸出
*	*	ABC+*	// * 是左結合性
D	*	ABC+*D	

最後再將 stack 裡的資料輸出，結果： ABC+*D*

3.6 Evaluation of Expressions

- Left parenthesis
  - Behaves as an operator with hight priority when it is not in stack, in-coming priority(icp) = 0
  - Behaves as an operator with low priority when it is in stack, in-stack priority(isp) = 8
  - Only the right parenthesis could cause it to get unstacked.
- Operators are taken out of the stack as long as their in-stack priority is numerically less than or equal to the in-coming priority of the new operator.
- isp('#') = icp('#') = 8
- 將 infix 轉成 postfix 的時間複雜度為 O(n)，n 為 token 數。

```
void postfix(expression e) {
    Stack<token> stack;
    token y;
    stack.Add('#');
    for (token x = NextToken(e); x != '#'; x = NextToken(e)) {
        if(x is an operand) cout << x;
        else if(x == ')' )
            for (y = *stack.Delete(y); y != '('; y = *stack.Delete(y))
              cout << y;
        else {
            for (y = *stack.Delete(y); isp(y) <= icp(x); y = *stack.Delete(y))
                cout << y;
            stack.Add(y);
            stack.Add(x);
        }
    }
    while(!stack.IsEmpty()) cout << *stack.Delete(y);
}
```