Infix-Postfix Conversion

Infix, Postfix and Prefix notations

Position of OPERATOR with respect to OPERANDS.

Infix	operand1 operator operand2
	A + B
Prefix	operator operand1 operand2
	+ A B
Postfix	operand1 operand2 operator
	A B +

Conversion Examples 1

A+B/C

Infix	**Postfix**	**Prefix**
A+B/C	A B C / +	+ A / B C

Conversion Examples 2

(A+B)/C

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

Conversion Examples 3

A*B+C*D

Infix	Postfix	Prefix
A*B+C*D	A B * C D * +	+* A B * C D

Conversion Examples 4

A+B+C/C

Infix	**Postfix**	**Prefix**
A+B+C/C	A B + C C / +	+ A + B / C C

Conversion Examples 5

(A+B+C)/D

Infix	Postfix	Prefix
(A+B+C)/D	A B + C + D /	/ + + A B C D

Point to Note

- No parentheses required in pre and postfix forms, i.e., operator precedence problems are absent
- Order of operands does not change, only operators are moved around in terms of sequence.

More examples from Tanenbaum book

Convert : POSTFIX & PREFIX

(assume $ for exponentiation operator, A $ B is A^B)

A + B

A + B – C

(A+B)*(C-D)

A$B*C-D+E/F/(G+H)

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

A-B/(C*D$E)

Answers Postfix

AB+	A + B
AB+C-	A + B - C
AB+CD-*	(A+B)*(C-D)
AB$C*D-EF/GH+/+	A$B*C-D+E/F/(G+H)
AB+C*DE--FG+$	((A+B)*C-(D-E))$(F+G)
ABCDE$*/-	A-B/(C*D$E)

Answer Prefix

+AB	A + B
-+ABC	A + B - C
*+AB-CD	(A+B)*(C-D)
+-*$ABCD//EF+GH	A$B*C-D+E/F/(G+H)
$-*+ABC-DE+FG	((A+B)*C-(D-E))$(F+G)
-A/B*C$DE	A-B/(C*D$E)

Evaluation of postfix expression using stack

```
opndstk = the empty stack;
/* scan the input string reading one*/
/* element at a time into symb */
while(not end of input){
   symb = next character;
   if(symb is an operand)
       push(opndstk,symb);
```

Evaluation of postfix expression

```
    else { /* symb is an operator*/
      opnd2 = pop(opndstk)
      opnd1 = pop(opndstk)
      value =  apply operator to opnd1
               and opnd2
      push(opndstk,value)
     }
  }
return pop(opndstk);
```

Evaluation of postfix expression using stack

1. Scan the expression from left to right.
2. If the symbol is an operand, push it into the stack.
3. If the symbol is an operator, pop two operands, apply the operators on the two operands, and push the result back into the stack.
4. Continue until we come to the end of the expression.
5. Pop the stack to find the final result of the entire expression.

Evaluation of postfix expressions

7 8 + 6 – 3 *

7	push 7
8	push 8
+	pop 8, pop 7, ADD 8 and 7, push 15
6	push 6
-	pop 6, pop 15, SUBTRACT 6 from 15, push 9
3	push 3
*	pop 3, pop 9, MULTIPLY 3 and 9, push 27
END	pop 27 which is the final result

Infix to Postfix

- Initialize a Stack for operators, output list
- Split the input into a list of tokens.
- for each token (left to right):

```
    if it is operand:  append to output
    if it is '(': push onto Stack
    if it is ')': pop & append till '('
    if it in '+-*/':
        while peek has precedence ≥ it:
            pop & append
        push onto Stack
pop and append the rest of the Stack.
```

Infix to Postfix

- Input (infix string A)
- Initialize S (an operator stack) and O(the output array)
- Parencount=0;
- x=next character from A
- While(x!='\0'){

```
        If(x is an operand){
                put x in output array (O);
        else if( x  is '(' ){
                push x in S;
                parencount ++; }
```

Infix to postfix

```
else if( x  is  ‘)’ ){
              while(S->top!=‘(‘){
              y=pop(S);
              put y in output array (O); }
    pop(S);
    parencount --;
}elseif(priority(S->a[s->top]>=priority(x)){
   while(priority(S->a[S->top]>=priority(x)){
                    y=pop(S);
                    write y in O;
                              }
  push x in S
}else
```

Infix to postfix

```
            push x in S
x=next character from A;
}/* end of first while*/
While(S->top >-1){
        y=pop(S);
        put y in O;
}
Put ‘/O’ in the output array O;
If (parencount == 0)
         Expression OK;
Else
 bracket mismatch;
}// end procedure
```

Dry run (A+B)*(C+D/E)

X	Output array	Stack	Parencount
(	-	(	1
A	A	(	1
+	A	(+	1
B	AB	(+	1
)	AB+	-	0
*	AB+	*	0

DRY RUN

X	Output array	Stack	Parencount
(	AB+	*(	1
C	AB+C	*(	1
+	AB+C	*(+	1
D	AB+CD	*(+	1
/	AB+CD	*(+/	1
E	AB+CDE	*(+/	1
)	AB+CDE/+	*	0

Converting from infix to postfix using stack

The Precedence evaluation function

prcd(op1, op2) is TRUE

if op1 has precedence over op2

```
prcd('*','+')   TRUE
prcd('+','+')   TRUE
prcd('+','*')   FALSE
```

Parentheses Balancing

The Parentheses Balancing Problem

`()()`	Balanced
`())`	Illegal
`)(`	Illegal
`()()(()`	Illegal
`((()))`	
`()))(((()`	
`((())`	

Algorithm???

Parentheses Balancing using a Stack

```
read next character from string
if '(' push into stack
else if ')'
    if stack is empty – ERROR
    else pop matching ')' from stack
until string is exhausted
if stack is not empty - ERROR
```

General Balancing Problem

`{}()`	Balanced
`[{]}`	Illegal
`{[()]}{}(())`	Balanced
`[[{]}()`	Illegal

Algorithm for matching brackets, braces, and parentheses???

THANK YOU!