Data Structures and its Applications

Dinesh Singh
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Infix to Postfix and Prefix Expressions – Implementation

Dinesh Singh
Department of Computer Science & Engineering

Infix , Postfix and Prefix Expressions

- Consider the sum of A and B expressed as A + B
- This representation is called infix.
- There are two alternate notations for expressing sum of A and B using the symbols A , B and + , these are
  - Prefix : + A B
  - Postfix : A B +
- The prefixes “Pre” , “post” and “in” refers to the relative position of the operator with respect to the two operands.
- In prefix, operators precedes the two operands
- In postfix, the operator follows the two operands
- In infix, the operator is in between the two operands

Infix , Postfix and Prefix Expressions

Conversion of Infix to Postfix

- For Example : consider the expression A + B * C
- The evaluation of the above expression requires the knowledge of precedence of operators
- A + B * C can be expressed as A + (B * C) as multiplication takes precedence over addition
- Applying the rules of precedence the above infix expression can be converted to postfix as follows

A + B * C = A + (B * C)

= A + (BC *) convert the multiplication

= A (B C *) + convert the addition

= A B C * +

Infix , Postfix and Prefix Expressions

Conversion of Infix to Prefix

- For Example : consider the expression A + B * C
- Applying the rules of precedence the above infix expression can be converted to prefix as follows

A + B * C = A + (B * C)

= A + (* B C) convert the multiplication

= + A (* B C) + convert the addition

= + A * B C

Data Structures and its Applications

Infix , Postfix and Prefix Expressions

PES UNIVERSITY ONLINE

Applying the rules of precedence the table shows the conversion of Infix to Postfix and Prefix Expression

Infix	Postfix	Prefix
A + B * C + D	A B C * + D +	+ + A * B C D
(A + B) * (C + D)	A B + C D + *	* + A B + C D
A * B + C * D	A B * C D * +	+ * A B * C D
A + B + C + D	A B + C + D +	+ + + A B C D
A $ B * C – D + E / F / (G+H)	A B $ C * D – E F / G H + / +	+ - * $ A B C D / /E F + G H
((A + B)*C – (D – E)) $ (F + G)	A B + C * D E – F G + $	$ - * + A B C – D E + F G
A – B / (C * D $ E)	A B C D E $ * / -	- A / B * C $ D E

$ is the exponentiation operator and its precedence is from right to left

For example A $ B $ C = A $ (B $ C)

Data Structures and its Applications

Infix to postfix conversion - algorithm

```
opstk is the empty stack
while(not end of input)
{
  symb = next input character
// if the input symbol is an operand , add it to the postfix string
 if(symb is an operand)
    add symb to postfix string
else
   {
     // pop the contents of the stack while the precedence of
     // top of the stack is greater than the precedence of the symbol scanned
      while(!empty(opstk ) and ( prcd(stacktop(opstk),symb))
       {
          topsym=pop(opstk)
         add topsymb to postfix string
       }
```

Infix to postfix conversion - algorithm

```
   /*push the input symbol on to the stack  if the precedence is less than  the
      precedence of top of the stack */
       if( empty(opstk) || symb!=')')
          push(opstk,symb)
     else
         topsymb=pop(opstk); // pop ' ( ' from the stack
}
while(!empty(opstk)
 {
   topsymb=pop(opstk)
   add topsymb to postfix string
 }
}
```

Infix to postfix conversion - algorithm

- prcd(OP1,OP2) is a function that compares the precedence of the top of the stack(OP1) and the input symbol (OP2) and returns TRUE if the precedence is greater else returns FALSE.
- Some of the return values of prcd function
  - prcd(‘ * ‘ , ‘ + ‘) returns TRUE
  - prcd(‘ + ‘, ‘ * ‘) returns FALSE
  - prcd(‘ + ‘ , ‘ + ‘) returns TRUE
  - prcd(‘ + ‘ , ‘ – ‘) returns TRUE
  - prcd(‘ – ‘ , ‘ – ‘) returns TRUE
  - prcd(‘ $ ‘ , ‘$’) returns FALSE : exponentiation operator, associatively from right to left
  - prcd(‘ (‘, op) , prcd(op ‘ (‘) returns FALSE for any operator
  - prcd (op , ‘)‘) returns TRUE for any operator
  - Prcd(‘ (‘ , ‘) ‘) returns FALSE

Infix to postfix conversion - algorithm

- The motivation behind the conversion algorithm is the desire to output the operators in the order in which they are to be executed.
- If an incoming operator is of greater precedence than the one on top of the stack, this new operator is pushed on to the stack.
- If on the other hand , the precedence of the new operator is less than the one on the top of the stack, the operator on the top of the stack should be executed first.
- Therefore the top of the stack is popped out and added to the postfix and the incoming symbol is compared with the new top and so on.
- Parenthesis in the input string override the order of operations
- When a left parenthesis is scanned, it is pushed on to the stack
- When its associated right parenthesis is found, all the operators between the two parenthesis are placed on the output string. Beacuse they are to be executed before any operators appearing after the parenthesis

Infix to postfix conversion – Trace of the algorithm

PES UNIVERSITY ONLINE

Trace of the algorithm for A + B * C

symb	postfix string	opstk	Remarks
A	A	Empty	symb is operand , add ‘A’on to the postfix string
+	A	+	symb is operator, and stack empty, push + on to the stack
B	AB	+	symb is operand , Add ‘B’ to the postfix string
*	AB	+*	symb is operator, precedence of + (stack top) is < than precedence of *, therefore push * on to the stack

Infix to postfix conversion – Trace of the algorithm

PES UNIVERSITY ONLINE

Trace of the algorithm for A + B * C

symb	postfix string	opstk	Remarks
C	ABC	+ *	symb is operand , add C on to the postfix string
End of input	ABC*	+	Pop * from the stack and add to the postfix string
-	ABC*+	Empty	Pop + from the stack and add to the postfix string

Data Structures and its Applications

Infix to postfix conversion – Trace of the algorithm

PES UNIVERSITY ONLINE

Trace of the algorithm for (A + B)* C

symb	postfix string	opstk	Remarks
(	-	Empty	Stack empty, push '(' on to the stack
A	A	(	Symb is an operand, add 'A' to the post fix string
+	A	(+	Precedence of top of the stack '(' is less than '+' , push '+' on to the stack
B	AB	(+	Symb is an operand, add B to the postfix string
)	AB+	(	Precedenc of stack top '+' is > than ')' Pop '+' from the stack and add to the postfix string
	AB+	empty	Precedence of stack top stack top '(' is not greater than ')' and symb is ')', therefor pop '(' from the stack

Infix to postfix conversion – Trace of the algorithm

Trace of the algorithm for (A + B)* C

symb	postfix string	opstk	Remarks
*	AB+	*	Stack empty, push '*' on to the stack
C	AB+C	*	Symb is an operand, add 'C' to the postfix string
End of string	AB+C*	Empty	End os input, pop * from the stack and add to the postfix.

Infix to posfix conversion – another algorithm

```
void convert_postfix(char *infix,char*postfix)
 {
    i=0;
   char ch;
   j=0;
   push(s,&top,'#');
   while(infix[i]!='\0')
   {
     ch=infix[i];
   //while the prcedence of top of stack is greater than the
//precedence of the input symbol, pop and add to the postfix
    while(stack_prec(peep(s,top))>input_prec(ch))
        postfix[j++]=pop(s,&top);
```

Infix to posfix conversion – another algorithm

```
if(input_prec(ch)!=stack_prec(peep(s,top)))
        push(s,&top,ch);
      else
        pop(s,&top);
    i++;
   }
   while(peep(s,top)!='#')
     postfix[j++]=pop(s,&top);
   postfix[j]='\0';
 }
```

Infix to posfix conversion – another algorithm

Precedence table

Operator	Input Precedence	Stack Precedence
+ , -	1	2
* , /	3	4
$	6	5
Operands	7	8
)	0	- : Never pushed on to stack
(	9	0
#	-	-1

Data Structures and its Applications

Infix to prefix conversion – algorithm

Example 1:
Input : A * B + C / D
Output : + * A B/ C D

Example 2 :
Input : (A - B/C) * (A/K-L)
Output : *-A/BC-/AKL

1: Reverse the infix expression i.e A+B*C will become C*B+A.
Note while reversing each '(' will become ')' and each ')' becomes '('.

2: Obtain the postfix expression of the modified expression i.e CB*A+.

3: Reverse the postfix expression. Hence in our example prefix is +A*BC.

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

dineshs@pes.edu

+91 8088654402