National University of Computer & Emerging Sciences

Applications of Stack

APPLICATIONS OF STACKS

Algebraic Expression

- An algebraic expression is a legal combination of operands and the operators.
  - Operand is the quantity on which a mathematical operation is performed.
  - Operator is a symbol which signifies a mathematical or logical operation.

Infix, Postfix and Prefix Expressions

- **INFIX:** expressions in which operands surround the operator.
- **POSTFIX**: operator comes after the operands, also Known as Reverse Polish Notation (RPN).
- **PREFIX**: operator comes before the operands, also Known as Polish notation.
- Example
  - Infix: A+B-C Postfix: AB+C- Prefix: -+ABC

Examples of infix to prefix and postfix

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

A+B*C in postfix

- Applying the rules of precedence, we obtained

A+B*C

A+(B*C) Parentheses for emphasis

A+(BC*) Convert the multiplication,

ABC*+ Postfix Form

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

Conversion to Postfix Expression

((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)

Why do we need PREFIX/POSTFIX?

- Appearance may be misleading, INFIX notations are not as simple as they seem
- To evaluate an infix expression we need to consider
  - Operators' Priority
  - Associative property
  - Delimiters

Why do we need PREFIX/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.

Why do we need PREFIX/POSTFIX?

- An expression in infix form is thus converted into prefix or postfix form and then evaluated without considering the operators priority and delimiters.

Conversion of Infix Expression to postfix

A+B*C = ABC*+

There must be a precedence function. `prcd(op1, op2), where op1 and op2 are chars representing operators.`

This function returns TRUE if op1 has precedence over op2 when op1 appears to the left of op2 in an infix expression without parenthesis. prcd(op1,op2) returns FALSE otherwise.

`prcd('*','+') and prcd('+','+') are TRUE whereas prcd('+','*') is FALSE.`

What if expression contains paranthesis?

- We modify our precedence function

```
prcd( '(' , op) = FALSE        for any operator op
prcd( op, '(' ) = FALSE        for any operator op other than ')'

prcd( op, ')' ) = TRUE         for any operator op other than '('
prcd( ')' ,op ) = undefined    for any operator op (an error)
```

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
      symb = next input character;
      if (symb is an operand)
            add symb to the postfix string
      else {
            while (!empty(opstk) &&
      prcd(stacktop(opstk),symb) ) {
                  topsymb = pop(opstk);
                  add topsymb to the postfix string;
            } /* end while */
           if ( empty(opstk) || symb != ')' )
              push(opstk, symb);
            else  //pop the paranthesis and discard it
              topsymb = pop(opstk);
      } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

Example-2: (A+B)*C

symb	Postfix string	opstk
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
C	AB+C	*
	AB+C*	

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
      symb = next input character;
      if (symb is an operand)
            add symb to the postfix string
      else {
            while (!empty(opstk) &&
      prcd(stacktop(opstk),symb) ) {
                  topsymb = pop(opstk);
                  add topsymb to the postfix string;
            } /* end while */
           if ( empty(opstk) || symb != ')' )
              push(opstk, symb);
            else  //pop the paranthesis and discard it
              topsymb = pop(opstk);
      } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

Example-3: ((A-(B+C)) *D) $ (E+F)

symb	Postfix string	opstk

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
      symb = next input character;
      if (symb is an operand)
            add symb to the postfix string
      else {
            while (!empty(opstk) &&
      prcd(stacktop(opstk),symb) ) {
                  topsymb = pop(opstk);
                  add topsymb to the postfix string;
            } /* end while */
           if ( empty(opstk) || symb != ')' )
              push(opstk, symb);
            else  //pop the paranthesis and discard it
              topsymb = pop(opstk);
      } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

Example-3: ((A-(B+C)) *D) $ (E+F)

symb	Postfix string	opstk
(		(
(		((
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

The difference in algorithm for converting an expression to PREFIX notation is minimal to the one we studied for converting an INFIX expression to POSTFIX. The only difference is that input symbols are read from right to left and added to the left of the PREFIX string.

Steps are:

1. Reverse the infix expression
   i. Parenthesis are confusing
2. Convert reversed string to postfix expression
3. Reverse the postfix expression to get prefix expression

Approach for converting an infix expression to prefix notation:

Reverse the infix expression: Reverse the order of the characters in the infix expression. This step is essential because we will be processing the expression from right to left in the next steps.

Convert the reversed infix expression to postfix notation:
Initialize an empty stack to hold operators.
Initialize an empty list to store the postfix expression.
Start scanning the reversed infix expression from left to right.
For each symbol in the reversed infix expression:
If it's an operand (a variable or number), add it to the output list.
If it's an operator, push it onto the stack if the stack is empty or has an operator with lower precedence on top. If the operator on top of the stack has higher or equal precedence, pop it and add it to the output list. Repeat this until the stack is empty or an operator with lower precedence is encountered.
If it's an opening parenthesis '(', push it onto the stack.
If it's a closing parenthesis ')', pop operators from the stack and add them to the output list until an opening parenthesis '(' is encountered. Pop and discard the opening parenthesis.
After processing all symbols, pop any remaining operators from the stack and add them to the output list.
The resulting list is the postfix expression of the reversed infix expression.

Reverse the postfix expression: Reverse the order of the characters in the postfix expression obtained in the previous step to get the prefix expression.

example:

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

Reverse the infix expression:)D - C(* B + A

Convert the reversed infix expression to postfix:

Reversed infix:)D - C(* B + A

Postfix: AD+CB*- (using the algorithm for converting infix to postfix)

Reverse the postfix expression: -*B+CB+DA

So, the prefix notation of the original infix expression "A + B * (C - D)" is "*+AB-CD."

Evaluating a Postfix Expression

Each operator in a postfix string refers to the previous two operands in the string.

Algorithm to Evaluate a Postfix Expression

```
opndstk = the empty stack
/* scan the input string reading one
     element */
/* at a time into symb */
while (not end of input) {
     symb = next input character;
     if (symb is an operand)
          push(opndstk, symb)
     else {
          /* symb is an operator */
          opnd2 = pop(opndstk);
          opnd1 = pop(opndstk);
          value = result of applying
     symb to opnd1 and opnd2;
          push(opndstk, value);
     } /* end else */
} /* end while */
return (pop(opndstk));
```

Example:

Postfix Expression: 6 2 3 + - 3 8 2 / + * 2 $ 3 +

symb	opnd1	opnd2	value	opndstk
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Questions?

Algorithm to Convert Infix to Postfix

```
opstk = the empty stack;
while (not end of input) {
      symb = next input character;
      if (symb is an operand)
            add symb to the postfix string
      else {
            while (!empty(opstk) &&
      prcd(stacktop(opstk),symb) ) {
                  topsymb = pop(opstk);
                  add topsymb to the postfix string;
            } /* end while */
           if ( empty(opstk) || symb != ')' )
              push(opstk, symb);
            else  //pop the paranthesis and discard it
              topsymb = pop(opstk);
      } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

A-B/(C*D^E)

symb	Postfix string	opstk

Algorithm to Convert Infix to Prefix

```
opstk = the empty stack;
while (not end of input) {
      symb = next input character;
      if (symb is an operand)
            add symb to the postfix string
      else {
            while (!empty(opstk) &&
      prcd(stacktop(opstk),symb) ) {
                  topsymb = pop(opstk);
                  add topsymb to the postfix string;
            } /* end while */
           if ( empty(opstk) || symb != ')' )
              push(opstk, symb);
            else  //pop the paranthesis and discard it
              topsymb = pop(opstk);
      } /* end else */
} /* end while */
/* output any remaining operators */
while (!empty(opstk) ) {
      topsymb = pop(opstk);
      add topsymb to the postfix string;
} /* end while */
```

A – B + C $ D $ E + F * G / H

symb	Postfix string	Opstk