Some Applications of Stack

Arithmetic Expressions
Polish Notation

What is Polish Notation?

- **Conventionally, we use the operator symbol between its two operands in an arithmetic expression.**

  `A+B` `C–D*E` `A*(B+C)`

  - **We can use parentheses to change the precedence of the operators.**
  - **Operator precedence is pre-defined.**
- **This notation is called *INFIX notation*.**
  - **Parentheses can change the precedence of evaluation.**
  - **Multiple passes required for evaluation.**

- **Polish notation**
  - **Named after Polish mathematician Jan Lukasiewicz.**
  - **Polish POSTFIX notation**
    - **Refers to the notation in which the operator symbol is placed after its two operands.**

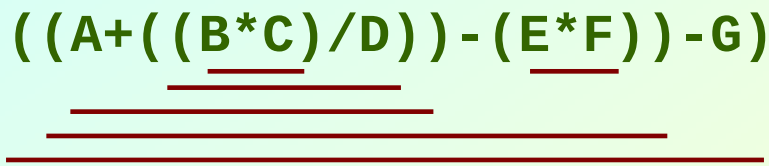
      `AB+  CD*  AB*CD+/`
  - **Polish PREFIX notation**
    - **Refers to the notation in which the operator symbol is placed before its two operands.**

      `+AB  *CD  /*AB-CD`

How to convert an infix expression to Polish form?

- Write down the expression in fully parenthesized form. Then convert stepwise.
- Example:

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

- Polish Postfix form:

  A B C * D / + E F * - G -

- Polish Prefix form:
  - Try it out ….

- **Advantages:**
  - **No concept of operator priority.**
    - **Simplifies the expression evaluation rule.**
  - **No need of any parenthesis.**
    - **Hence no ambiguity in the order of evaluation.**
  - **Evaluation can be carried out using a single scan over the expression string.**
    - **Using stack.**

Evaluation of a Polish Expression

- Can be done very conveniently using a stack.
  - We would use the Polish postfix notation as illustration.
    - Requires a single pass through the expression string from left to right.
    - Polish prefix evaluation would be similar, but the string needs to be scanned from right to left.

```
while (not end of string) do
 {
   a = get_next_token();
   if (a is an operand)
      push (a);
   if (a is an operator)
   {
      y = pop();  x = pop();
      push (x ‘a’ y);
   }
 }
 return (pop());
```

Parenthesis Matching

The Basic Problem

- **Given a parenthesized expression, to test whether the expression is properly parenthesized.**
  - **Whenever a left parenthesis is encountered, it is pushed in the stack.**
  - **Whenever a right parenthesis is encountered, pop from stack and check if the parentheses match.**
  - **Works for multiple types of parentheses**

    ( ), { }, [ ]

```
while (not end of string) do
 {
    a = get_next_token();
   if (a is ‘(‘ or ‘{‘ or ‘[‘)
       push (a);
    if (a is ‘)’ or ‘}’ or ‘]’)
    {
       if (isempty()) {
         print (“Not well formed”);
         exit();
       }
        x = pop();
        if (a and x do not match) {
           print (“Not well formed”);
           exit();
       }
    }
 }
 if (not isempty())
   print (“Not well formed”);
```

Converting an INFIX expression to POSTFIX

Basic Idea

- Let Q denote an infix expression.
  - May contain left and right parentheses.
  - Operators are:
    - Highest priority: ^ (exponentiation)
    - Then: * (multiplication), / (division)
    - Then: + (addition), – (subtraction)
  - Operators at the same level are evaluated from left to right.
- In the algorithm to be presented:
  - We begin by pushing a '(' in the stack.
  - Also add a ')' at the end of Q.

The Algorithm (Q:: given infix expression, P:: output postfix expression)

```
push (‘(’);
Add “)” to the end of Q;
while (not end of string in Q do)
{
  a = get_next_token();
  if (a is an operand) add it to P;
  if (a is ‘(’)  push(a);
  if (a is an operator)
  {
      Repeatedly pop from stack and add to P each
      operator (on top of the stack) which has the
      same or higher precedence than “a”;
      push(a);
  }
```

```
   if (a is ‘)’)
   {
        Repeatedly pop from stack and add to P each
        operator (on the top of the stack) until a
        left parenthesis is encountered;

        Remove the left parenthesis;
   }
}
```

Q: A + (B * C – (D / E ^ F) * G) * H)

Q	STACK	Output Postfix String P
A	(	A
+	(+	A
(	(+ (	A
B	(+ (	A B
*	(+ (*	A B
C	(+ (*	A B C
-	(+ (-	A B C *
(	(+ (- (	A B C *
D	(+ (- (	A B C * D
/	(+ (- (/	A B C * D
E	(+ (- (/	A B C * D E
^	(+ (- (/ ^	A B C * D E
F	(+ (- (/ ^	A B C * D E F
)	(+ (-	A B C * D E F ^ /

Q	STACK	Output Postfix String P
*	(+ (- *	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 F ^ / G * -
H	(+ *	A B C * D E F ^ / G * - H
)		A B C * D E F ^ / G * - H * +

Some Other Applications

- **Reversing a string of characters.**
- **Generating 3-address code from Polish postfix (or prefix) expressions.**
- **Handling function calls and returns, and recursion.**