

Stacks, Queues, Deques

Hsuan-Tien Lin

Dept. of CSIE, NTU

April 7, 2015

A Simple Application: Parentheses Balancing

- in C, the following characters show up in pairs: (), [], {}, ""

```
good: {xxx (xxxxxx) xxxxx "xxxx" x }
```

```
bad:  {xxx (xxxxxx} xxxxx "xxxx" x }
```

- the LISP programming language

```
(append (pow (* (+ 3 5) 2) 4) 3)
```

how can we check parentheses balancing?

Stack Solution to Parentheses Balancing

inner-most parentheses pair \implies top-most plate

'(: 堆盤子上去 ;)': 拿盤子下來

Parentheses Balancing Algorithm

```
for each  $c$  in the input do  
  if  $c$  is a left character  
    push  $c$  to the stack  
  else if  $c$  is a right character  
    pop  $d$  from the stack and check if match  
  end if  
end for
```

many more sophisticated use in compiler design

System Stack

- recall: function call \Leftrightarrow 拿新的草稿紙來算
- old (original) scrap paper: temporarily not used, 可以壓在下面

System Stack: 一疊草稿紙, each paper (stack frame) contains

- return address: where to return to the previous scrap paper
- local variables (including parameters): to be used for calculating within this function
- previous frame pointer: to be used when escaping from this function

some related issues: stack overflow? security attack?

Reading Assignment

be sure to go ask the TAs or me if you are still confused

Reading Assignment

be sure to go ask the TAs or me if you are still confused

Stack for Expression Evaluation (Supplementary)

$$a/b - c + d * e - a * c$$

- precedence: $\{*, /\}$ first; $\{+, -\}$ later
- steps
 - $f = a/b$
 - $g = f - c$
 - $h = d * e$
 - $i = g + h$
 - $j = a * c$
 - $\ell = i - j$

Postfix Notation

- same operand order, but put “operator” **after** needed operands
- can “operate” immediately when seeing operator
- no need to look beyond for precedence

Postfix from Infix (Usual) Notation

- infix:

3 / 4 - 5 + 6 * 7 - 8 * 9

- parenthesize:

3 / 4 - 5 + 6 * 7 - 8 * 9

- for every triple in parentheses, switch orders

- remove parentheses

difficult to parenthesize efficiently

Evaluate Postfix Expressions

$$34/5 - 67 * +89 * -$$

- how to evaluate? left-to-right, “operate” when see operator
- 3, 4, / \Rightarrow 0.75
- 0.75, 5, - \Rightarrow -4.25
- -4.25, 6, 7, * \Rightarrow -4.25, 42 (note: -4.25 stored for latter use)
- -4.25, 42, + \Rightarrow 37.75
- 37.75, 8, 9, * \Rightarrow 37.75, 72 (note: 37.75 stored for latter use)
- 37.75, 72, - \Rightarrow ...

stored where?

stack so closest operands will be considered first!

Stack Solution to Postfix Evaluation

Postfix Evaluation

```
for each token in the input do  
  if token is a number  
    push token to the stack  
  else if token is an operator  
    sequentially pop operands  $a_{t-1}, \dots, a_0$  from the stack  
    push token( $a_0, a_1, a_{t-1}$ ) to the stack  
  end if  
end for  
return the top of stack
```

matches closely with the definition of postfix notation

One-Pass Algorithm for Infix to Postfix

infix \Rightarrow postfix efficiently?

- at $/$, not sure of what to do (need later operands) so **store**

$$a/b - c + d * e - a * c$$

- at $-$, know that a / b can be $a b /$ because $-$ is of lower precedence

$$a/b - c + d * e - a * c$$

- at $+$, know that $? - c$ can be $? c -$ because $+$ is of same precedence but $\{-, +\}$ is left-associative

$$a/b - c + d * e - a * c$$

- at $*$, not sure of what to do (need later operands) so **store**

$$a/b - c + d * e - a * c$$

stored where? **stack** so closest operators will be considered first!

Stack Solution to Infix-Postfix Translation

```
for each token in the input do  
  if token is a number  
    output token  
  else if token is an operator  
    while top of stack is of higher (or same) precedence do  
      pop and output top of stack  
    end while  
    push token to the stack  
  end if  
end for
```

- here: infix to postfix with operator stack
—closest operators will be considered first
- recall: postfix evaluation with operand stack
—closest operands will be considered first
- mixing the two algorithms (say, use two stacks): simple calculator

Some More Hints on Infix-Postfix Translation

```
for each token in the input do  
  if token is a number  
    output token  
  else if token is an operator  
    while top of stack is of higher (or same) precedence do  
      pop and output top of stack  
    end while  
    push token to the stack  
  end if  
end for
```

- for left associativity and binary operators
 - right associativity? same precedence needs to wait
 - unary/trinary operator? same
- parentheses? highest priority
 - at '(', cannot pop anything from stack
 - like seeing '*' while having '+' on the stack
 - at ')', can pop until '(' —like parentheses matching