Stacks and Queues

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 21–22, 2011
Stacks (Sec. 3.1): Abstract Data Type

**Stack**

- object: a container that holds some elements
- action: push (to the top), pop (from the top)

- last-in-first-out (LIFO): 擠電梯，洗盤子
- very restricted data structure, but important for computers — will discuss some cases later
in C, the following characters show up in pairs: ( ), [], {}, ""

good: \{xxx (xxxxxxx) xxxxx"xxxx"x\}
bad:  \{xxx (xxxxxxx) xxxxx"xxxx"x\}

the LISP programming language

\(\text{append} \ (\text{pow} \ (* \ (+ \ 3 \ 5) \ 2) \ 4) \ 3)\)

how can we check parentheses balancing?
inner-most parentheses pair $\Rightarrow$ 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
recall: function call \(\mapsto\) 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
when stack full, grow array by size $M$

- successful (direct) growth: constant time
- if unlucky, growth by copying: $O(\text{capacity})$
- $M = 1$ or any constant: very conservative — worst case, $O(n^2)$ for $n$ pushes (why?)
- $M = \text{capacity}$:
  — growth when exceeding 1, 2, 4, 8, 16, ...
  — each growth takes time around 1, 2, 4, 8, 16, ...
  — when $n$ pushes with $n = 13$?
    - push: 13
    - grow: $1 + 2 + 4 + 8 = 15$
  — $2^k < n \leq 2^{k+1}$, time $2^{k+1} - 1$ on growth and $n$ on pushes
  — $O(n)$ for $n$ pushes
Stack for Expression Evaluation (Sec. 3.6)

\[
a/b - c + d \times e - a \times c
\]

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

\[
\begin{align*}
\text{f} & \quad \Rightarrow a/b \\
\text{c} & \quad \Rightarrow a/b/c- \\
\text{d} & \quad \Rightarrow a/b/c- \text{d} \times+ \\
\text{e} & \quad \Rightarrow a/b/c- \text{d} \times+ \text{a} \times- \\
\end{align*}
\]

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: \[ 3 \div 4 - 5 + 6 \times 7 - 8 \times 9 \]

Parenthesize:

\[
(((3 \div 4) - 5) + (6 \times 7)) - (8 \times 9)
\]

For every triple in parentheses, switch orders:

\[
(((3 + 1) - 5) + (6 \times 7)) + (8 \times 9)
\]

Remove parentheses:

\[3 \div 4 - 5 + 6 \times 7 - 8 \times 9\]

difficult to parenthesize efficiently