Data Structures and Algorithms
(資料結構與演算法)

Lecture 6: Stack

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

motivation

Visual Intuition of Stack

figure by Mx. Granger, licensed under CC0 1.0 via Wikimedia Commons

last-in-first-out (LIFO)

- stack of chairs
- stack of plates
- elevator

有很多限制的資料結構
放進去只能從上面放

stack: a restricted data structure,
but important for computer science

The Three Stack Operations

figure by Mx. Granger, licensed under CC0 1.0 via Wikimedia Commons

```
PEEP(S)
    // GET usually named PEEP
    // return top element of S
```

```
PUSH(S, data)
    // INSERT usually named PUSH
    // put data onto top of S
```

```
POP(S)
    // REMOVE usually named POP
    // remove and return top element of S
```

sometimes other utility functions like SIZE() or ISEMPTY()

Parentheses Balancing

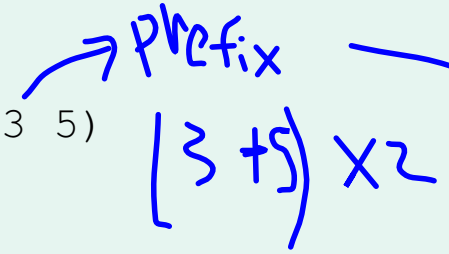
C

```
int main(){
  printf("Hello World");
  return 0;
}
```

—(), {}, "", . . . need pairing

LISP

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



—() needs pairing

how can we check parentheses balancing?

像是函數呼叫
但對電腦來說其實不算直覺
postfix是對店腦來說最直覺的
postfix轉prefix可能考

Stack Solution to Parentheses Balancing

any ')' should match last unmatched '(' (LIFO)

'(': PUSH ')': POP

右配對的對象：找最接近的左括號，即最新看到的左括號

Parentheses Balancing Algorithm

```
1   S = empty stack
2   for each c in input
3       if c is '('
4           PUSH(S, c)
5       else // c is ')'
6           if not ISEMPTY(S)
7               POP(S)
8           else
9               return FALSE
10  return ISEMPTY(S)
```

裡面還有左括號沒配到

看到右括號但沒左括號

many more sophisticated use in compiler design

System Stack

- function call: compute with a new scratch paper
- old (original) scratch paper: temporarily not used; will be first to return to
- system stack: stack of scratch papers (stack frames), each containing
  - local variables (including parameters): to be used for calculating within this function
  - previous frame (return) pointer: to be used when escaping from this function

local variable清掉後才能看到return address

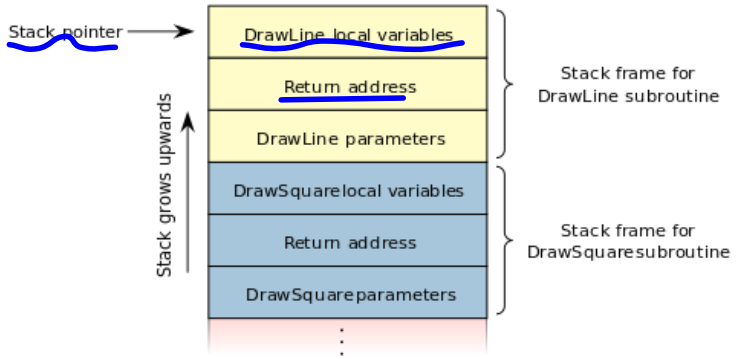


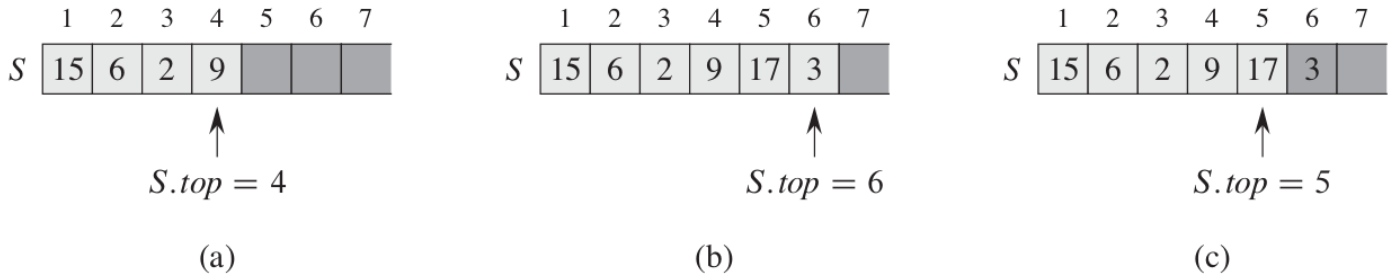
figure by Cameron McCormack, licensed under CC-BY-SA 2.5 via Wikimedia Commons

計算紙
算完後撕掉計算紙，就會看到原本的計算紙

some related issues: security attack?

implementation

Stacks Implemented on Array



(Textbook Figure 10.1)

(a) stack with 4 elements
(b) after PUSH(S, 17) and PUSH(S, 3)
(c) after POP(S) which returns 3

用array模擬stack

PUSH(S, *data*)
1 $S.top = S.top + 1$
2 $S.arr[S.top] = data$

POP(S)
1 $S.top = S.top - 1$
2 **return** $S.arr[S.top + 1]$

usually: consecutive array with $S.top$
at 'tail' of array for $O(1)$ operations

Stacks Implemented on Linked List

if singly-linked list, top at head or tail?
which would you choose?

application: postfix evaluation

Stack for Expression Evaluation

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

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

電腦漢人步一樣
電腦看到的是有A有B，做除法(postfix)
不向人看是A除B（infix）

僅移動符號
不移動數字

$$ab/c - de * + ac * -$$

Postfix Notation

same operand order, but put "operator" after needed operands
—can "operate" immediately when seeing operator
—no need to look beyond for precedence

Evaluate Postfix Expressions

$$3 * 4 - (5 + 6) * 7 + 8 * 9 \Longrightarrow 3\ 4\ *\ 5\ 6\ +\ 7\ *\ -\ 8\ 9\ *\ +$$

- how to evaluate? left-to-right, “operate” when see operator
- 3, 4, * $\Rightarrow$ 12
- 12, 5, 6, + $\Rightarrow$ 12, 11
- 12, 11, 7, * $\Rightarrow$ 12, 77
- 12, 77, - $\Rightarrow$ -65
- -65, 8, 9, * $\Rightarrow$ -65, 72
- -65, 72, + $\Rightarrow$ 7

stored where?

stack so closest operands will be considered first!

最接近的抓出來算

Stack Solution to Postfix Evaluation

Postfix Evaluation

```
1  S = empty stack
2  for each token in input
3      if token is a number
4          PUSH(S, token)
5      elseif token is an operator
6          b = POP(S)
7          a = POP(S)
8          PUSH(S, token(a, b))
9  return POP(S)
```

3 4 ∗ 5 6 + 7 ∗ − 8 9 ∗ +

- 3, 4, * ⇒ 12
- 12, 5, 6, + ⇒ 12, 11
- 12, 11, 7, * ⇒ 12, 77
- 12, 77, - ⇒ -65
- -65, 8, 9, * ⇒ -65, 72
- -65, 72, + ⇒ 7

先塞回去

matches closely with the definition of postfix notation

application: expression parsing

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

34/ 5- 67*

- remove parentheses

need multi-passes if using computers

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

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

-稍微比+優先議點因為在左邊

- 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

```
1  S2 = empty stack
2  for each token in input
3        if token is a number
4              output token
5        elseif token is an operator
6              while not IS-EMPTY(S2) and PEEP(S2) is higher/same precedence
7                    output POP(S2)
8              PUSH(S2, token)
```

不同stacks分別作寫起來可能會輕鬆易點

- here: infix to postfix with operator stack S_2
  —closest operators will be considered first
- recall: postfix evaluation with operand stack S
  —closest operands will be considered first
- mixing the two algorithms (say, use two stacks): simple calculator

Some More Hints on Infix-Postfix Translation

```
1  S2 = empty stack
2  for each token in input
3      if token is a number
4          output token
5      elseif token is an operator
6          while not IS-EMPTY(S2) and PEEP(S2) is higher/same precedence
7              output POP(S2)
8          PUSH(S2, token)
```

- 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

Summary

Lecture 6: Stack

- motivation

  temporary storage with LIFO

- implementation

  Last-In-First-Out

  $O(1)$ push/pop from tail of array

- application: postfix evaluation

  stack as temporary storage of partial results

- application: expression parsing

  stack as temporary storage of waiting operands