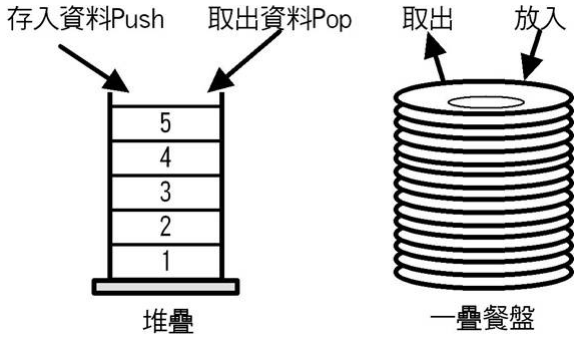
資料結構的堆疊

資訊科技系
林偉川

堆疊的基礎

- 「堆疊」屬於一種擁有特定進出規則的線性串列結構，如同在餐廳廚房的工人清洗餐盤，將洗好的餐盤疊在一起，每一個洗好的餐盤放在這疊餐盤的頂端，如下圖所示：



2

堆疊的基礎-操作

- 堆疊的基本操作，如下所示：
  - **push()**：將資料存入堆疊，在堆疊的頂端新增資料。
  - **pop()**：從堆疊取出資料，每執行一次，就從頂端取出一個資料。
  - **isStackEmpty()**：檢查堆疊是否是空的，以便判斷是否還有資料可以取出。

3

堆疊的基礎-特性

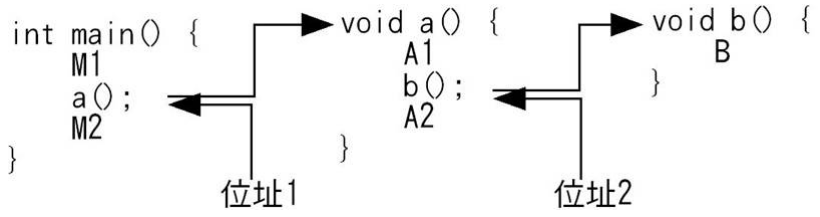
- 堆疊的資料因為是從頂端一一存入，堆疊內容是依序執行push(1)、push(2)、push(3)、push(4)和push(5)的結果，接著從堆疊取出資料，依序執行pop()取出堆疊資料，如下所示：

  pop()：5，pop()：4，pop()：3，pop()：2，pop()：1
- 取出的資料順序是5、4、3、2、1，可以看出其順序和存入時相反，稱為「先進後出」的特性。堆疊擁有的特性，如下所示：
  - 只允許從堆疊的頂端存取資料。
  - 資料存取的順序是先進後出，也就是後存入堆疊的資料，反而先行取出。

4

堆疊-C語言的函數呼叫

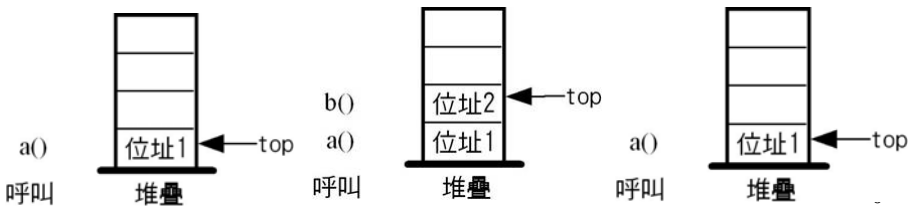
- C語言函數呼叫的執行過程就是使用作業系統的堆疊儲存目前的執行狀態，例如：C程式擁有主程式main()和a()和b()兩個函數，M1、M2、A1、A2和B分別代表程式區塊，程式的執行順序依序為：M1→A1→B→A2→M2。



5

堆疊-C語言的函數呼叫

- 當主程式main()呼叫和進入函數a()，main()的返回位址1存入堆疊，
- 接著進入函數b()，返回位址2被存入堆疊，執行完函數b()後，從堆疊取出返回位址2，繼續執行A2程式區塊，再繼續從堆疊取出返回位址1，繼續主程式的執行，如下圖所示：



堆疊的基礎-C語言的區域變數

- C語言的全域變數是在編譯階段就配置記憶體空間，區域變數則是在執行階段進入函數後，才配置變數所需的記憶體空間，而且在結束函數執行後，就馬上釋放區域變數佔用的記憶體空間。
- 函數的參數也屬於一種區域變數，C語言函數呼叫在處理區域變數和參數時，就是將這些變數和參數的值都使用堆疊保留下來，程式在呼叫函數前，將返回位址、各區域變數和參數都一一存入堆疊，等到返回後，再一一取出堆疊內容，恢復成函數呼叫前的執行狀態。

7

使用陣列建立堆疊

```
02: #define MAXSTACK 100
03: int stack[MAXSTACK]; /* 堆疊的陣列宣告 */
04: int top = -1;            /* 堆疊的頂端 */
05: /* 抽象資料型態的操作函數宣告 */
06: extern int isStackEmpty();
07: extern int push(int d);
08: extern int pop();
```

8

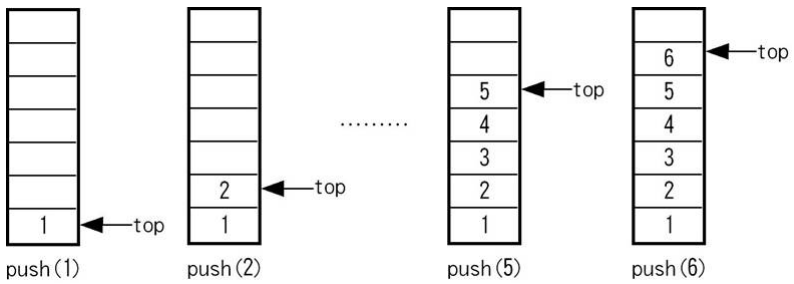
使用陣列建立堆疊-存入元素

- 因為堆疊的特性是只能從堆疊頂端存取資料，所以需要一個額外的top變數來指向堆疊頂端的陣列索引值，使用此索引值將資料存入堆疊。
- push()函數將資料存入堆疊，如下所示：
  - Step 1：將堆疊頂端的指標top加1。
  - Step 2：將參數的資料存入指標top所指的陣列元素。 stack[++top] = d;

9

使用陣列建立堆疊-存入元素

- 例如：依序將值1~6存入堆疊的圖例，如下圖所示：



10

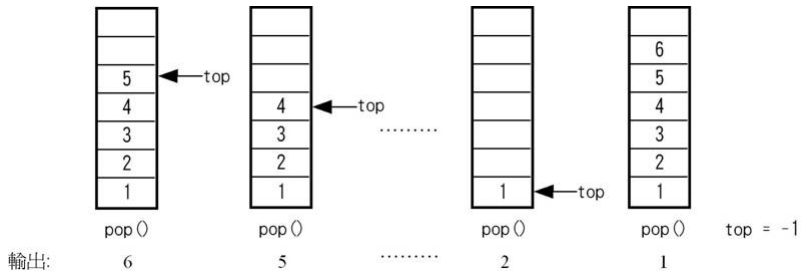
使用陣列建立堆疊-取出元素

- 從堆疊取出資料的是pop()函數，如下所示：
  - Step 1：取出目前堆疊指標top所指的陣列值。
  - Step 2：將堆疊指標top的內容減1，即指向下一個堆疊元素。 return stack[top--];

11

使用陣列建立堆疊-取出元素

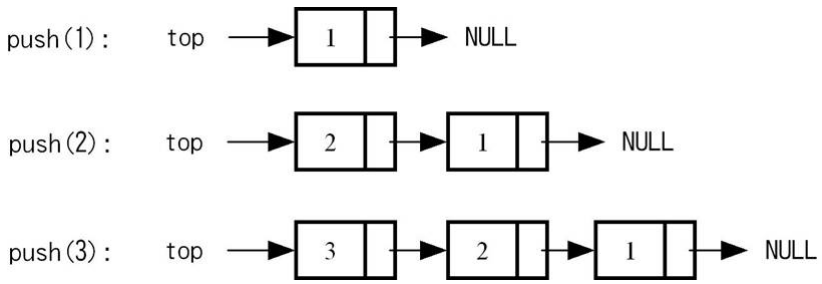
- 例如：在依序將值1~6存入堆疊後，從堆疊取出各元素的圖例，如下圖所示：



12

使用鏈結串列建立堆疊-存入元素

- 例如：依序存入值1~3到堆疊的串列，如下圖所示：



13

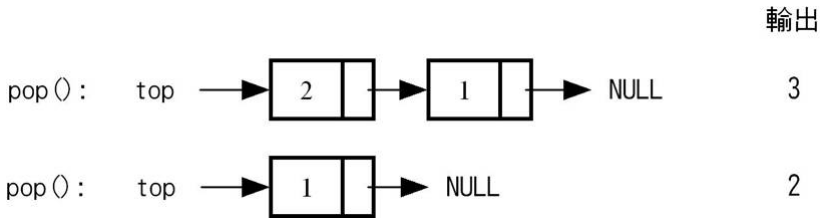
使用鏈結串列建立堆疊-取出元素

- 從堆疊取出資料的是pop()函數，也就是刪除串列的第1個節點，其取出步驟如下所示：
  - Step 1：將堆疊指標top指向下一個節點。
    top = top->next;
  - Step 2：取出原來堆疊指標所指節點的內容。
    temp = ptr->data;
  - Step 3：釋回原來堆疊指標的節點記憶體。
    free(ptr);

14

使用鏈結串列建立堆疊-取出元素

- 例如：在依序存入值1~3到堆疊後，呼叫二次pop()函數取出堆疊元素，可以看出取出2個堆疊元素，因為一共存入3個元素，所以目前堆疊還剩下一個元素1。如下圖所示：



15

堆疊的應用 - 運算式的計算與轉換

- 運算式的種類、計算與轉換
- 中序運算式轉換成後序運算式
- 後序運算式的計算

16

運算式的種類、計算與轉換

- 算術運算式是使用「運算子」和「運算元」所組成，如下所示：

  A+B

  A*B+C

- 上述運算式的A、B和C是運算元，＋和*是運算子。

17

運算式的種類、計算與轉換-種類

- 運算式種類依據運算子位在運算式中的位置，可以分為三種，如下所示：
  - **中序表示法（Infix）**：運算式中的運算子是位在兩個運算元之間，例如：A+B和A*B+C。
  - **前序表示法（Prefix）**：運算子位在兩個運算元之前，例如：+AB和+*ABC。
  - **後序表示法（Postfix）**：運算子位在兩個運算元之後，例如：AB+和AB*C+。

18

運算式計算與轉換-優先順序

- 中序表示法在執行運算式的計算和轉換前，需要注意運算子的「優先順序」，運算子的優先順序，如下表所示：

運算子	優先順序
括號( )	高 ↓ 低
負號-	
乘* 除/ 餘數%	
加+ 減-	

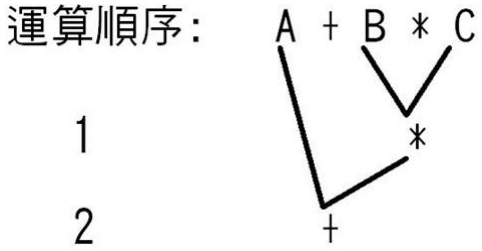
19

運算式計算與轉換-中序計算

- 運算式在先計算B*C後才和A相加，因為運算子*的優先順序大於+。如果中序運算式不考慮運算子優先順序，同一個中序運算式可能產生不同的運算結果，如下所示：

A+B*C：先算A+B，再和C相乘

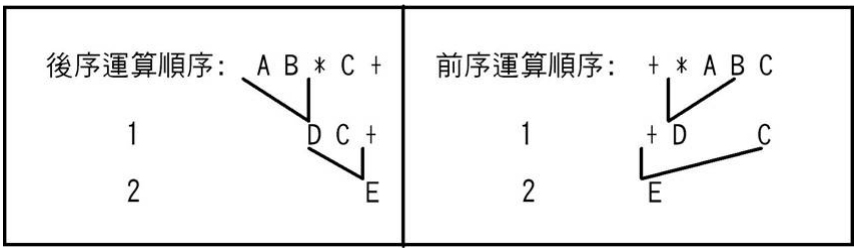
A+B*C：先算B*C再加上A



20

運算式計算與轉換-前序與後序計算

- 前序和後序表示法，就不需要考慮運算子的優先順序，如下所示：



21

運算式的種類、計算與轉換-轉換

- 運算式轉換分為中序轉前序和中序轉後序表示法，其轉換步驟十分相似，其差異只在運算子是位在運算元前或後。例如：中序運算式，如下：

  A*(B+C)

- 上述運算式轉換成前序和後序表示法的步驟，以運算子優先順序來進行處理，如下表所示：

步驟	說明	前序表示法	後序表示法
1	轉換加法	A*(+BC)	A*(BC+)
2	轉換乘法	*A(+BC)	A(BC+)*
3	刪除括號	*A+BC	ABC+*

運算式的種類、計算與轉換-轉換

- 另一種方法是先替中序運算式加上完整括號來確認運算的優先順序，如下所示：

  中序運算式： A+B*(C+D)-E

  加上括號的中序運算式： ((A+(B*(C+D)))-E)

- 上述是加上括號的中序運算式，現在只需從最中間的括號開始，將運算子移到右括號的位置且刪除右號，直到刪除所有右括號為止，如下所示：

  將運算子搬移到右括號： ((A(B(CD+*+E-

  刪除所有的左括號： ABCD+*+E-

23

運算式的種類、計算與轉換-轉換

- 一些中序運算式轉換成前序和後序運算式的範例，如下表所示：

中序表示法	前序表示法	後序表示法
A+B	+AB	AB+
(A+B)/C	/+ABC	AB+C/
(A+B)*(C+D)	*+AB+CD	AB+CD+*

24

後序運算式的計算

- 後序運算式的計算和前序運算式類似，都屬於無括號和優先順序的運算式計算，例如：一個後序運算式如下所示：

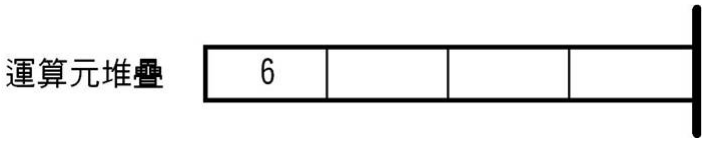
  67*45++

- 上述運算式的運算子是位在運算元之後，換句話說，當讀取運算式的運算元或運算子時，只需一個堆疊存放運算元即可，如果是運算子馬上取出堆疊的運算元，然後將計算結果存回到堆疊。

25

後序運算式的計算-過程1

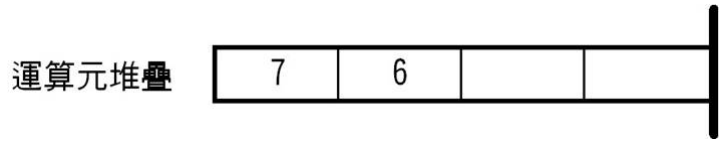
- 後序運算式：67*45++的計算過程是在主迴圈依序讀取運算式的運算元或運算子，第1個讀入的字元是'6'，因為是運算元，所以存入運算元堆疊，如下圖所示：



26

後序運算式的計算-過程2

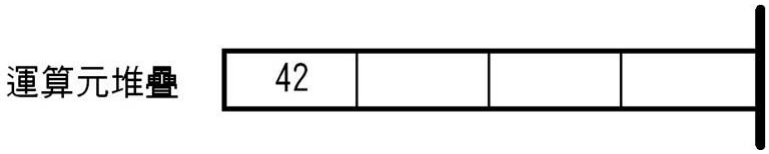
- 接著讀入的字元'7'是運算元，再存入運算元堆疊，如下圖所示：



27

後序運算式的計算-過程3

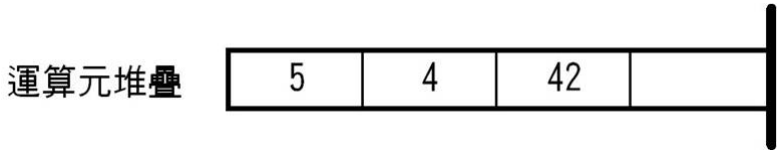
- 繼續讀入字元'*'是運算子，從運算元堆疊取出2個運算元7和6，將6*7的計算結果42存回堆疊，如下圖所示：



28

後序運算式的計算-過程4

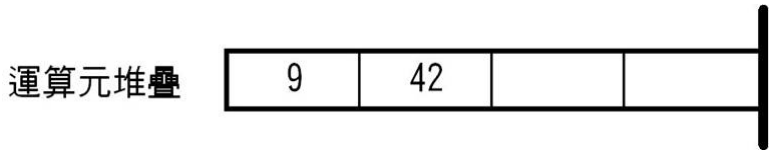
- 接著讀入的是字元'4'和字元'5'兩個運算元，依序存入運算元堆疊，如下圖所示：



29

後序運算式的計算-過程5

- 繼續讀入的字元'+'是運算子，所以從運算元堆疊取出2個運算元5和4，然後將4+5計算結果9存回運算元堆疊，如下圖所示：



30

後序運算式的計算-過程6

- 最後一個讀入的字元是運算子'+'，從運算元堆疊取出2個運算元9和42，就可以得到後序運算式的計算結果是51。

31

後序運算式的計算-演算法

- 後序運算式計算的演算法可以依照前述計算過程來推導，其完整步驟如下所示：
  - Step 1：使用迴圈從左至右依序讀入後序運算式。
    - (1) 如果讀入的是運算子, 則：
      - 1) 從運算元堆疊pop 2個運算元，先pop為第2個運算元。
      - 2) 計算此運算的值後，存回運算元堆疊。
    - (2) 如果讀入的是運算元，直接存入運算元堆疊。
  - Step 2：最後pop運算元堆疊的內容，就是後序運算式的計算結果。

32

中序運算式轉換成後序運算式

- 中序運算式轉換成後序運算式可以使用堆疊配合運算子優先順序來進行轉換。例如：相同運算結果的中序和後序運算式，如下所示：

  中序運算式： (9+6)*4 ，後序運算式： 96+4*

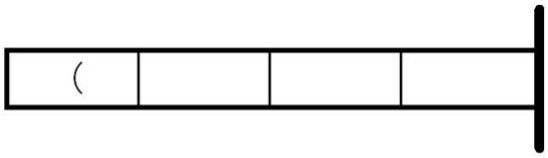
- 後序運算式是從中序運算式轉換而成，可以看出兩個運算式中的運算元排列順序是相同的，只有運算子執行順序有優先順序的差異，所以中序運算式的轉換只需一個運算子堆疊，如果讀入運算元馬上輸出即可，運算子需要進行優先順序的比較，以決定輸出或存入堆疊。

33

中序運算式轉換成後序運算式

- 中序運算式：(9+6)*4的計算過程是在主迴圈從左至右依序讀取運算式的運算元或運算子，第1個讀入的字元是'('左括號，存入運算子堆疊，如下圖所示：

運算子堆疊

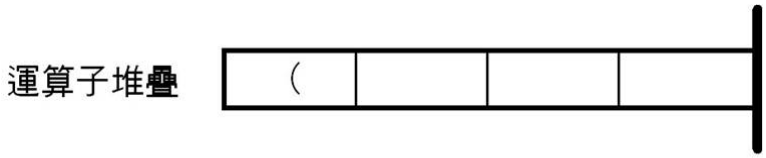


輸出：

34

中序運算式轉換成後序運算式

- 接著讀入的字元是運算元'9'，直接輸出運算元，如下圖所示：

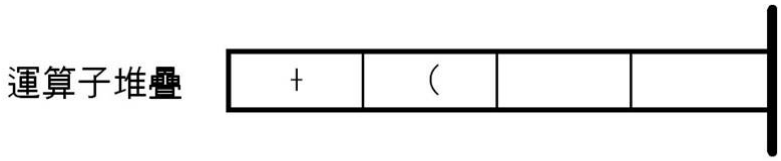


輸出：　　9

35

中序運算式轉換成後序運算式

- 繼續讀入字元'+'是運算子，因為不是右括號且'+'號的優先順序大於堆疊中的'('號（注意！左括號的優先順序最小），所以將運算子存入運算子堆疊，如下圖所示：

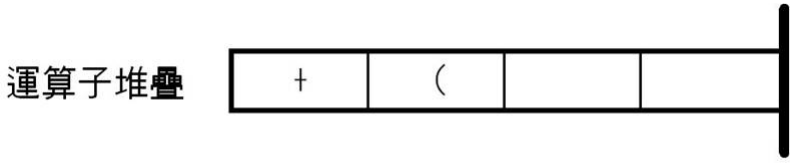


輸出：　　9

36

中序運算式轉換成後序運算式

- 然後讀入的字元是運算元'6'，直接輸出運算元，如下圖所示：

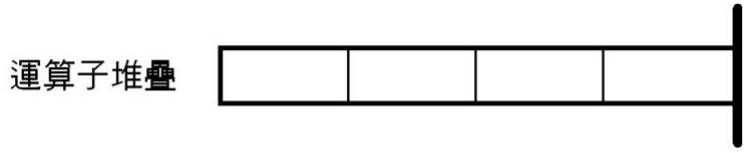


輸出：　　96

37

中序運算式轉換成後序運算式

- 接著讀入的是右括號')'，我們需要從運算子堆疊取出運算子'+'輸出，直到左括號為止，左括號在此的功能只是作為一個標籤，所以並不用輸出，此時的堆疊已經全空，如下圖所示：

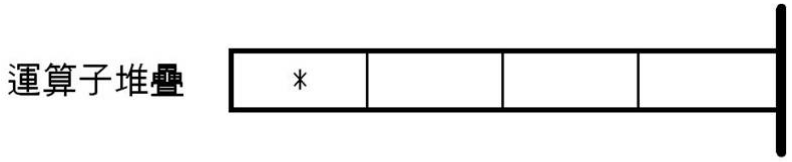


輸出：　　96+

38

中序運算式轉換成後序運算式

- 繼續讀入字元'*'是運算子，存入運算子堆疊，如下圖所示：

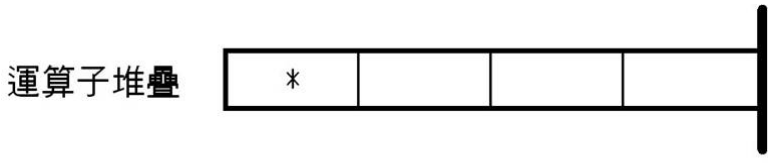


輸出：　　96+

39

中序運算式轉換成後序運算式

- 接著讀入運算元'4'，直接輸出運算元，如下圖所示：



輸出：　　96+4

40

中序運算式轉換成後序運算式

- 現在已經讀完整個中序運算式，接著從運算子堆疊取出剩下的運算子'*'且輸出，可以得到轉換的後序運算式，如下所示：

  96+4*

41

中序運算式轉換成後序運算式

- 中序轉成後序運算式的演算法可以依照前述計算過程來推導，其完整步驟如下所示：
  - Step 1：使用迴圈讀取中序運算式的運算元和運算子，若：
    - (1) 讀取的是運算子：
      - 1) 如果運算子堆疊是空的或是左括號，存入運算子堆疊。
      - 2) 如果是右括號，從堆疊取出運算子輸出，直到左括號為止。
      - 3) 如果堆疊不是空的，持續和堆疊的運算子比較優先順序，若：
        - » a. 優先順序比較低或相等，pop()輸出運算子再push。
        - » b. 優先順序比較高或堆疊空了，將運算子存入運算子堆疊。
    - (2) 讀取的是運算元，直接輸出運算元。
  - Step 2：如果運算子堆疊不是空的，依序取出運算子堆疊的運算子。

42

中序運算式轉換成後序運算式

- 中序轉後序的演算法其完整步驟如下：
  - (1) 讀取的是運算子：
    - 如果輸入運算子是(則push (
    - 如果輸入運算子堆疊是空的或是(， push輸入運算子。
    - 如果輸入運算子是)，從堆疊pop運算子輸出，直到左括號。
    - 如果輸入運算子遇到堆疊不是空的，持續和堆疊的運算子比較優先順序，若：
      - 輸入運算子優先順序比較低或相等，從堆疊pop運算子輸出再push。(內高外低)
      - 輸入運算子優先順序比較高或堆疊空了，push輸入運算子。(內低外高)
  - (2) 讀取的是運算元，直接輸出運算元。
  - 如果最後運算子堆疊不是空的，依序取出運算子堆疊的運算子。

43

堆疊與遞迴的應用 - 走迷宮問題

- 使用堆疊的回溯控制走迷宮
- 堆疊與遞迴有異曲同工之妙，因為遞迴函數是直接使用作業系統的堆疊，當然我們也可以自行在程式建立堆疊執行回溯控制，走迷宮問題可以使用堆疊或遞迴方式來找出走出迷宮的路。

44

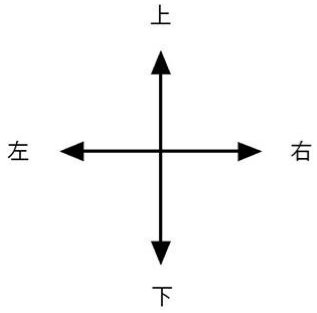
使用堆疊的回溯控制走迷宮

- 「回溯」屬於人工智慧的重要觀念，使用嘗試錯誤方式來找尋問題的解答。在迷宮中找出一條走出迷宮的路，這就是回溯最常見的應用。

45

使用堆疊的回溯控制走迷宮

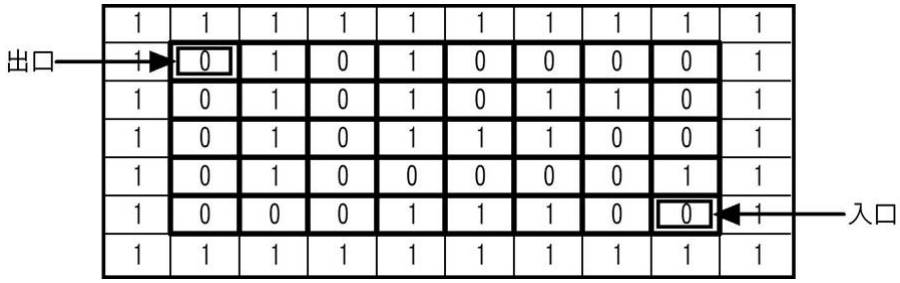
- 假設：迷宮內的路只有向上下左右四個方向，而且行走的優先順序是上、下、左和右，對角線方向並不允許行走，迷宮的行走方式是依照上述四個方向，每次走一步，如果遇到牆壁，就需要嘗試剩下幾個方向是否有路可行，繼續相同的走法直到找出一條走出迷宮的路，如右圖所示：



46

使用堆疊的回溯控制走迷宮

- 迷宮的圖形是使用一個二維陣列來表示，陣列元素值0表示是可走的路，1代表是牆壁，如下圖所示：



1	1	1	1	1	1	1	1	1	1
1	0	1	0	1	0	0	0	0	1
1	0	1	0	1	0	1	1	0	1
1	0	1	0	1	1	1	0	0	1
1	0	1	0	0	0	0	0	1	1
1	0	0	0	1	1	1	0	0	1
1	1	1	1	1	1	1	1	1	1

47

使用堆疊的回溯控制走迷宮-走迷宮的路徑1

0	1	0	0
0	1	0	1
0	0	0	0
0	1	1	●

0	1	0	0
0	1	0	1
0	0	0	●
0	1	1	2

48

使用堆疊的回溯控制走迷宮-走迷宮的路徑2

0	1	0	0
0	1	0	1
0	0	●	2
0	1	1	2

0	1	2	●
0	1	2	1
0	0	2	2
0	1	1	2

49

使用堆疊的回溯控制走迷宮-走迷宮的路徑3

0	1	3	3
0	1	3	1
0	0	●	2
0	1	1	2

0	1	3	3
0	1	3	1
●	2	2	2
0	1	1	2

50

請寫出答案

1. Push a; push b; push c; pop; pop; push d; 請列出堆疊中處理資料的過程為何? d,a
2. push(5); push(6); push(7); printf("%d\n",pop());
   push(8); printf("%d\n",pop()-pop());
   push(pop()+5); printf("%d\n",pop()); 列出堆疊中處理資料過程並印出及堆疊中的資料為何?
3. push(3); push(4); printf("%d\n",pop()+5);
   push(5); push(6); push(7); push(8);
   push(pop()+pop()); printf("%d\n",pop()*2);列出堆疊中處理資料過程並印出及堆疊中的資料為何?

51

請寫出答案

4. 請將下列中續運算轉換為後序運算的轉換過程及堆疊內容?
   A. a+(b-c/d)*e B. (a+b+c)*(f-e*d/g)
   C. a*(b+c/(d-e)) D. a+b*c+d-e/f
   E. a*c-f/9*3/2
5. 若a=6,b=3,c=8,d=4,e=2，求下列後序運算之值?
   a. abcde+-*/ b. abc+-de*/ c. ab-cd*/e+

52

作業

- 試寫一程式，輸入為一迷宮陣列，輸出為此迷宮從出口至入口的反推位置。
- 試寫一程式，輸出為西洋棋中之八個皇后所擺設的位置。

53