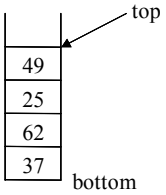
Chapter 4

堆疊與佇列(Stacks and Queues)

4-1 堆疊(Stacks)

✧ 要點：堆疊的特點

1. 定義：堆疊(stacks)是一種有序串列，其插入(insertion)與刪除(deletion)皆須在一同端進行。
2. 插入與刪除的一端稱爲頂端(top)；另一端則稱爲底部(bottom)。
3. 堆疊又稱爲後入先出(LIFO, Last-In-First-Out)的資料結構。



✧ 要點：堆疊的應用

1. 記錄返回地址：副程式呼叫時，儲存呼叫者的返回位址(Return Address)，當函數巢狀呼叫時，最後被呼叫的函數最先返回呼叫者，符合後入先出之特性，故可以用堆疊來記錄其返回位址。
2. 活動記錄：區塊結構之程式語言，其執行期環境需要使用活動記錄(activation record)，活動記錄中包含區域變數(local variables)的記憶體空間，形式參數(formal parameters)的空間，以及環境鏈結(environment links)如靜態鏈結(static link)和動態鏈結(dynamic link)。

3. 中斷之處理(Interrupt Handling)：記錄被斷程式的狀態，如返回位址、旗號等，亦是後進先出的特性。
4. 遞迴副程式改寫成爲非遞迴程式時，有時需要用到堆疊來模擬，才能改寫出來，堆疊是用來記錄呼叫函數的相關狀態。
5. 撰寫回溯式(Backtracking)演算法時，須以堆疊記錄各步驟的狀況，當發現選擇路徑行不通時，由堆疊中可取回先前的狀態。圖形的深度優先追蹤，就是一個使用堆疊來進行回溯演算法的例子。
6. 中序式、前序式與後序式，三者之間的相互轉換；以及後序式的求值計算，都需要使用堆疊來進行。
7. 堆疊機器(Stack Machine)：一種執行零位址指令的機器，計算前先將運算元(operands)置於堆疊中，指令執行時，由堆疊取出所需的運算元，計算來的結果再放回堆疊中。

✧ 要點：堆疊之抽象資料型態

```
structure Stack is
  objects: 一個有限、有序之線性串列。
  functions:
   for all  s∈ Stack, i∈ Item, M∈ positive int erger
    CreateStack(s,M):Stack ::=建立一個空的堆疊 s，其最大容量爲 M
    IsStackEmpty(s):Boolean::=if   s 中元素個數=0 then
                                    IsStackEmpty =true
                               else
                                    IsStackEmpty =false
    IsStackFull(s,M):Boolean ::= if s 中元素個數=M then
                                    IsStackFull =true
                               else
                                    IsStackFull =false
    AddStack(s,i):Stack ::= if IsStackFull (s,M) then Error
                            else 將 i 插入 s 的頂端
    DeleteStack(s):Item ::= if IsStackEmpty (q) then Error
```

else 將 s 最頂端的元素刪除並傳回其值

TopOfStack(s):Item ::= if IsStackEmpty (q) then Error

else 傳回 s 最頂端的元素

end

end Stack

✧ 要點：堆疊之陣列實作

使用一個陣列來存放資料，並且以一個整數變數sp(stack pointer)記錄堆疊頂端的位置。所需資料結構宣告如下：

```
(1)  const int N = 100;
(2)  ItemType stack[N];
(3)  int sp;
```

1. 建立堆疊：只要將 sp 指向底部即可。

```
(1)  void CreateStack()
(2)  {
(3)     sp = -1;
(4)  }
```

2. 檢查堆疊是否爲空的或滿的。

```
(1)  Boolean IsStackEmpty()
(2)  {
(3)     return (sp == -1);
(4)  }

(5)  Boolean IsStackFull()
(6)  {
(7)     return (sp == (N–1));
(8)  }
```

3. 插入新資料到堆疊頂端，一般又稱爲Push down。

```
(1)  void AddStack(ItemType item)
(2)  {
(3)     if (IsStackFull()) StackOverflow();
(4)     stack[++sp] = item;
(5)  }
```

4. 由堆疊中取出資料，一般又稱爲Pop up。

```
(1)  ItemType DeleteStack()
(2)  {
(3)     if (IsStackEmpty()) StackUnderflow();
(4)     else return stack[sp--];
(5)  }
```

5. 傳回堆疊頂端的資料，但並未從堆疊中刪除。

```
(1)  ItemType StackTop()
(2)  {
(3)     if (IsStackEmpty()) StackUnderflow();
(4)     else return stack[sp];
(5)  }
```

精選例題 1

寫出三種堆疊的應用。

解答：(1) 存放副程式呼叫時的 Activation Record 或 Interrupt Handler 記錄狀態之用。

(2) 做爲 Infix Expression 至 Postfix Expression 轉換之用。

(3) 用在 Backtracking 演算法上。

精選例題 2

(1) 使用 vector representation 來表示一個 stack, 請寫出演算法來取得第 i 項元素的值，但不刪除此項元素。

(2) 使用 vector representation 來表示一個 stack, 請寫出演算法來改變第 i 項元素的值。

解答：(1)
```
int get(int i)
{
    if (i > sp+1) error();
     else return stack[sp - i + 1];
}
```

(2)
```
void change(int i, items x);
{
    if (i > sp+1) error();
   else stack[sp - i + 1]=x;
}
```

精選例題 3

有一stack其基底爲B，指標內位址爲P，容量大小爲Z，試寫一程序(Algorithm)描述pop-up和push-down。

解答：init. P=B-1
```
void push_down(items x)
{
    if (P == B+Z-1) StackOverflow();
    else Mem[++P] = x;
}

items pop_up()
{
    if (P < B) StackUnderflow();
```

```
        else return Mem[P--];
    }
```

精選例題 4

給予如下堆疊定義(使用 C 語言)：

```
#define MAX_STACK 100;
typedef int ITEM_TYPE;
typedef enum Boolean { FALSE, TRUE } BOOLEAN;
tyoedef struct stack_type {
    ITEM_TYPE item[MAX_STACK];
    int   top;
} STACK_TYPE;
```

假設 top 是指到堆疊下一個空的位置(point to the next free spot in the stack)。請用 C 語言實作下列運算：

```
void create_stack(STACK_TYPE *stack);
void destroy_stack(STACK_TYPE *stack);
BOOLEAN empty_stack(STACK_TYPE *stack);
BOOLEAN full_stack(STACK_TYPE *stack);
void push(STACK_TYPE *stack, ITEM_TYPE new_item);
void pop(STACK_TYPE *stack, ITEM_TYPE *old_item);
```

90高考三級第二試

解答：

```
void create_stack(STACK_TYPE **stack)
{   *stack=(STACK_TYPE *)malloc(sizeof(STACK_TYPE));
    (*stack)→top=0;    }

void destroy_stack(STACK_TYPE *stack)
{    free(stack);      }

BOOLEAN empty_stack(STACK_TYPE *stack)
{    return (stack→top==0);      }
```