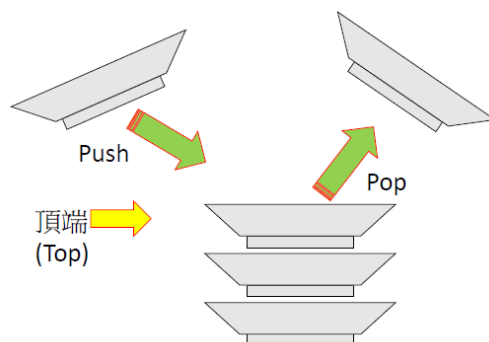


參、堆疊

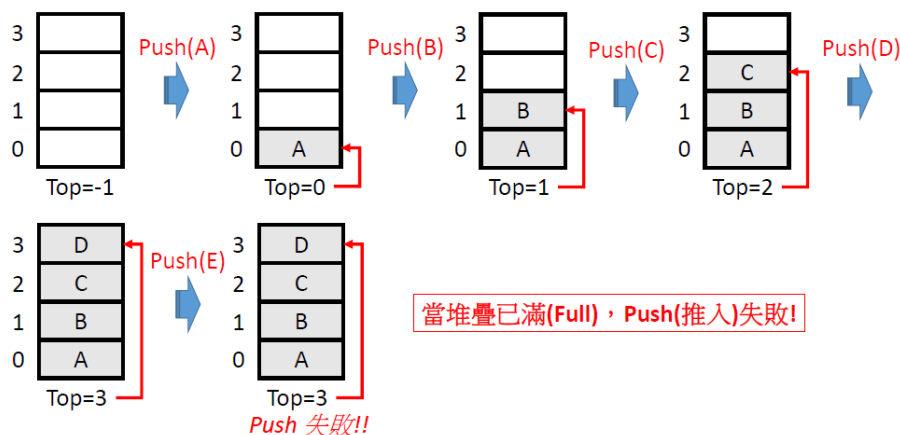
一、堆疊 (Stack) 簡介?

1. 何謂堆疊(Stack)?

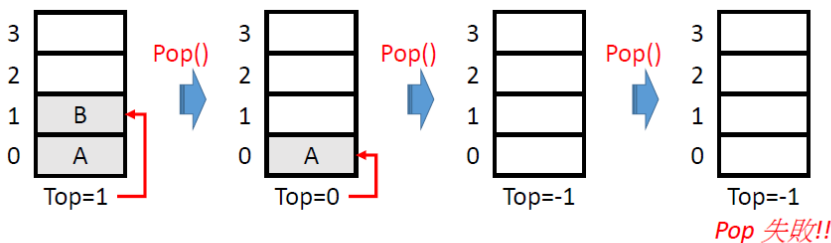
- ①是一種後進先出的有序串列
- ②資料只能從堆疊頂端進出
- ③放入的動作稱 Push(推入)
- ④取出的動作稱 Pop(彈出)



※推入元素進堆疊：



※從堆疊彈出元素：








當堆疊已空(Empty)，Pop(彈出)失敗!

2. 堆疊(Stack)的定義？

- ①一群相同性質元素的集合，即有序串列(Ordered List)
- ②具有後進先出(Last In First Out, LIFO)的特性
- ③將一個項目放入堆疊的頂端，此動作稱為推入(Push)。
- ④從堆疊頂端拿走一個項目，此動作稱為彈出(Pop)。
- ⑤Push 和 Pop 的動作，皆發生在同一端，此端稱為頂端(Top)。
- ⑥要取出資料時，只能從 Top 取出，不能從中間取出資料。
- ⑦要放入資料時，只能放置在 Top，不能從中間插入資料。

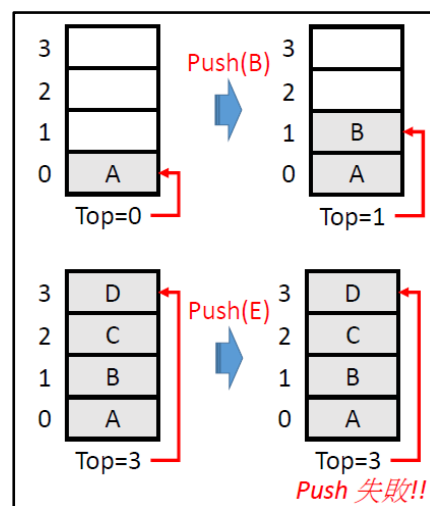
3. 堆疊常用的運算或操作：

-  **Push**→加入新項目在堆疊的頂端
-  **Pop**→取出堆疊頂端一個項目
-  **TopItem**→查看堆疊頂端的項目內容
-  **IsEmpty**→判斷堆疊是否為空，若是傳回真(True)、否傳回假(False)
-  **IsFull**→判斷堆疊是否為滿，若是傳回真(True)、否傳回假(False)

4. 以演算法呈現堆疊：

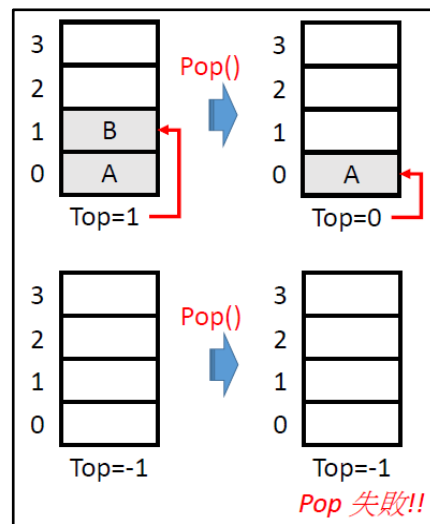
(1) Push

```
Procedure Push(item, Stack)
Begin
If (Top=N-1)
Stack is full;
Else
{
Top = Top + 1;
Stack[Top]=item;
}
End
```



(2) Pop

```
Procedure Pop(item, Stack)
Begin
If (Top=-1)
Stack is Empty;
Else
{
Item = Stack[Top];
Top = Top-1;
}
End
```



二、以陣列實現堆疊

(1) Push

```
intpush( intn, double stack[], double item, int*top ){
    if(isFull(n,*top))return 1;//Stack Full
    else{
        *top = *top + 1;
        stack[*top] = item;
        return 0;//Job finish
    }
}
```

(2) Pop

```
intpop( intn, double stack[], double *item, int*top){
    if(isEmpty(*top))return 1;//Stack Empty
    else{
        *item = stack[*top];
        *top = *top -1;
        return 0;//Job finish
    }
}
```

(3) isEmpty

```
intisEmpty(inttop){
    if(top<0)return 1;//True
    else return 0;//False
}
```

(4) isFull

```
intisFull(intn, inttop){
    if(n<=top+1)return 1;//True
    else return 0;//False
}
```

(5) topItem

```
inttopItem(intn, double stack[], double *item,
inttop){
    if(isEmpty(top))return 1;//Stack Empty
    else{
        *item = stack[top];
    }
}
```

```
    return 0; //Job finish
}
```

※測試程式

```
#include <stdlib.h> //For using random number
#include <time.h> //For seed of srand()
#include <stdio.h> //For printf()

int main(int argc, char * argv[]){
    int N=4; //stack size is 4
    //make a stack sized N
    double stack[N];
    int top=-1; //initialize stack
    double item;
    int stackIsFull;
    int stackIsEmpty;
    int repeat_times= 10; //maximum times about actions.
    int action; //Decide do push or pop. if action>50 do push,
    otherwise, do pop.
    int i;

    //Check Stack Status
    stackIsEmpty= isEmpty(top);
    if(stackIsEmpty==1){
        printf("Stack is Empty!!\n");
    }else{
        printf("Stack is Not Empty!!\n");
    }
    stackIsFull= isFull(N,top);
    if(stackIsFull){
        printf("Stack is Full!!\n");
    }else{
        printf("Stack is Not Full!!\n");
    }

    //push and pop data
    srand(time(NULL));
    for(i=0; i<repeat_times; i++){
```

```
action = rand()%100+1; //Randomly take a number between 100~1.
if(action > 50){
    //do push
    printf("---DO PUSH---\n");
    item = (double)(rand()%1000+1)/100.0;
    if(push(N,stack,item,&top)==1){
        printf("Stack is full!! PUSH FAILURE!!\n");
        stackIsFull=1;
    }else{
        printf("PUSH SUCCESSFULLY!!\n");
        topItem(N,stack,&item,top);
        printf("Current item on top of the stack is %f\n",item);
        stackIsEmpty=0;
    }
    printf("---END PUSH---\n");
}else{
    //do pop
    printf("---Do POP---\n");
    if(pop(N,stack,&item,&top)==1){
        printf("Stack is empty!! POP FAILURE!!\n");
        stackIsEmpty=1;
    }else{
        printf("POP SUCCESSFULLY!!\n");
        printf("Get item:%f from stack.\n",item);
        if(topItem(N,stack,&item,top)==1){
            printf("Stack is Empty!!\n");
            stackIsEmpty=1;
        }else{
            printf("Current item on top of the stack is %f\n",item);
        }
        stackIsFull=0;
    }
    printf("---END POP---\n");
}
return 0;
}
```

三、堆疊應用-中序運算式

1. 中序運算式→[運算元 運算子 運算元] [x+y]

2. 表示法及運算：

①使用電腦程式處理，需使用到兩個堆疊

②演算法的處理邏輯

(1)建立運算元堆疊和運算子堆疊，初始為空

(2)從左往右開始逐一讀取運算式

(3)若讀取的是運算元，推入運算元堆疊內

(4)反之，讀取到的是運算子

a. 若讀取到 $($ ，推入運算子堆疊；(中序表示法，左括號優先權最低)

b. 若讀取到 $)$ ，依序彈出運算子堆疊內的運算子，直到彈出 $($ 為止。每次彈出一個運算子，就從運算元堆疊內彈出兩個運算元，並計算之，然後結果推入運算元堆疊內；

c. 若讀取到非以上之任何運算子，與運算子堆疊頂端元素比較，若堆疊為空，或優先權高於該頂端元素，就推入堆疊內；反之，依序從堆疊內彈出運算子，直到頂端元素的優先權低於該運算子，然後將其推入堆疊。每次彈出一個運算子，就依據該運算子所需的運算元數量，從運算元堆疊內，彈出相同數量的運算元，依據運算子計算後，結果推入運算元堆疊內。

③當運算式讀完後，若運算子堆疊非空，依序彈出運算子，並計算後，結果推入運算元堆疊，直到運算子堆疊為空，計算結束；

④從運算元堆疊彈出最頂端元素，就是計算結果。

◎範例講解： $A+B * (C-D)$

3. 演算法：

①假設陣列的索引值從 0 開始

②建立兩個空堆疊，一個稱為運算元堆疊 `stack_operand`，一個稱為運算子堆疊 `stack_operator`

③運算式儲存於一個字串陣列 `exp` 內。陣列索引範圍 0~N-1，N 等於運算式內元素的數量。陣列第一個位置開始，依序，每個位置，放置一個運算式的元素。運算式由左往右，依序從陣列位置 0 處開始放置

④設定一個索引值 `p`，以此索引值來指向陣列內的中序運算式。令 `p=0`，初始指向第一個元素

4. 演算法(以程式表示)：

否則若 element == ")" , //若是")" 運算子

```
Do_Loop
Action = pop( stack_operator)
Number = Operand_number(Action);
K = 0;
Do_Loop
Operands[k] =pop( stack_operand);
K = K+1;
Until( K >= Number );
push( stack_operand, Compute( Action, Operands,
Number ) );
Until( Action == "(" )
```

否則, //不是")" 運算子

```
Do_Loop( priority(element) <=
priority(topItem( stack_operator) 或
IsEmpty(stack_operator) ==False )
Action = pop( stack_operator)
Number = Operand_number(Action);
K = 0;
Do_Loop
Operands[k] =pop( stack_operand);
K = K+1;
Until( K >= Number );
push( stack_operand, Compute( Action, Operands,
Number ) );
Continue_Do_Loop;
push( stack_operator);
```

```
Do_Loop( IsEmpty( stack_operator) <> true )
Action = pop( stack_operator)
Number = Operand_number(Action);
K = 0;
Do_Loop
Operands[k] =pop( stack_operand);
K = K+1;
Until( K >= Number );
push( stack_operand, Compute( Action, Operands,
Number ) );
```

```
Continue_Do_Loop;  
Answer = pop( stack_operand);
```

四、堆疊應用-前序運算式

1. 前序運算式→[運算子 運算元 1 運算元 2] [+xy]

2. 中序轉前序

①加括號去除法

⊙例題 1：轉換 $A+B*(C-D)$ 成為前序表示法

②堆疊處理法

(1)由右至左依序取得資料項(以 d_i 表示取得的資料項)

(2)如果 d_i 是運算元，則直接輸出

(3)否則 d_i 必是運算子(包含左右括號)，則：

a. 如果 $d_i == ")"$ ，推入堆疊內

b. 如果 $d_i == "("$ ，依序彈出堆疊內的運算子，並將其輸出，直到取出 $)$ 為止。 $(")"$ 和 $"("$ 不輸出

c. 如果 $d_i != ")"$ and $d_i != "("$ ，則與堆疊頂點的運算子(以 ds 表示)進行優先權比較：

※1. 當 d_i 比 ds 優先或推疊為空， d_i 推入堆疊，跳到步驟 4；反之，彈出 ds ，並將其輸出，再重複此步驟 3-3-1

(4)如果運算式尚未讀取完畢，回到步驟 1，反之，已經讀取完畢，而堆疊中尚有運算子，依序由頂端彈出運算子，並輸出

(5)反轉輸出的字串

⊙例題 2：轉換 $A+B*(C-D)$ 成為前序表示法

五、堆疊應用-後序運算式

1. 後序運算式 \rightarrow [運算元 1 運算元 2 運算子] [xy+]

⊙範例講解： $A+B * (C-D)$

2. 中序運算式轉後序運算式

①加括號去除法

⊙例題 3：轉換 $A+B * (C-D)$ 成為後序表示法

②堆疊處理法

(1)由左至右依序取得資料項(以 d_i 表示取得的資料項)

(2)如果 d_i 是運算元，直接輸出

a. 否則若 $d_i == "("$ ，直接推入堆疊

b. 否則若 $d_i == ")"$ ，依序彈出並輸出堆疊中的運算子，直到彈出 "(" 為止，此 "(" 和 ")" 不輸出

c. 否則若 d_i 的優先序高於堆疊頂端的運算子(以 d_s 表示)或堆疊為空，推入堆疊

d. 否則

e. 重複：

①彈出 d_s 並輸出，

②直到 d_i 的優先權高於 d_s 或堆疊為空

③將 d_i 推入堆疊

(3)若還有未取得資料項，回到步驟 1，否則繼續下步驟

(4)若堆疊非空，重複：彈出 d_s 並輸出

(5)結束。

⊙例題 4：轉換 $A+B*(C-D)$ 成為後序表示法

3. 後序運算式的計算

- ①建立一個空堆疊 S
- ②令 $k=1$ ； L =運算式元素的數量
- ③取得後序運算式最左邊的第 k 個運算元素，放入 item 變數內
- ④若 item 為運算元，推入堆疊 S 內：
 - a. 否則，從堆疊內彈出兩個運算元
 - b. 依據 item 運算子，進行計算，然後運算結果再推入堆疊 S 內
- ⑤ $k = k + 1$ ；
- ⑥若 $k \leq L$ ，回到步驟 3；否則往下繼續
- ⑦從堆疊 S 內彈出，輸出此值(此值就是此運算式的運算結果)
- ⑧結束

⊙範例講解： $3+2 * (5 - 1) \rightarrow 3251 - * +$