四 則運算 VS Stack

- ## 認 識 PreFix、InFix、PostFix

PreFix（前序式）：* + 1 2 + 3 4

InFix（中序式）：(1+2)*(3+4)

PostFix（後序式）：1 2 + 3 4 + *

- ## 後 序式的運算

說明：將中序式轉換為後序式 的好處是，不用處理運算子先後順序問題，只要依序由運算式由前往後讀取即可。

例如：
運算時由 後序式的前方開始讀取，遇到運算元先存入堆疊，如果遇到運算子，則由堆疊中取出兩個運算元進行對應的運算，然後將結果存回堆疊，如果運算式讀取完 畢，那麼堆疊頂的值就是答案了，例如我們計算12+34+*這個運算式（也就是(1+2)*(3+4)）：

讀取	堆疊
1	1
2	1 2
+	3 // 1+2 後存回
3	3 3
4	3 3 4
+	3 7 // 3+4 後存回
*	21 // 3 * 7 後存回

Algorithm Gossip: 中序式轉後序式（前序式）

說 明

平常所使用的運算式，主要是將運算元放在運算子的兩旁，例如a+b/d這樣的式子，這稱之為中序（Infix）表示 式，對於人類來說，這樣的式子很容易理 解，但由於電腦執行指令時是有順序的，遇到中序表示式時，無法直接進行運算，而必須進一步判斷運算的先後順序，所以必須將中序表示式轉換為另一種表示方 法。

可以將中序表示式轉換為後序（Postfix）表示式，後序表示式又稱之為逆向波蘭表示式（Reverse polish notation），它是由波蘭的數學家盧卡謝維奇提出，例如(a+b)*(c+d)這個式子，表示為後序表示式時是ab+cd+*。

解 法

用手算的方式來計算後序式相當的簡單，將運算子兩旁的運算元依先後順序全括號起來，然後將所有 的右括號取代為左邊最接近的運算子（從最內層括號開始），最後去掉所有的左括號就可以完成後序表示式，例如：

```
a+b*d+c/d    =>     ((a+(b*d))+(c/d)) -> abd*+cd/+
```

如果要用程式來進行中序轉後序，則必須使用堆疊，演算法很簡單，直接敘述的話就是使用迴圈，取出中序式的字元，遇運算元直接輸出；堆疊運算子與左括號； 堆疊中運算子優先順序大於讀入的運算子優先順序的話，直接輸出堆疊中的運算子，再將讀入的運算子置入堆疊；遇右括號輸出堆疊中的運算子至左括號。

演 算法

以下是虛擬碼的運算法，\0表示中序式讀取完畢：

```
Procedure Postfix(infix) [
 Loop [
 op = infix(i)
 case [
 :x = '\0':
 while (stack not empty)
 // output all elements in stack
 end
 return
 :x = '(':
 // put it into stack
 :x is operator:
 while (priority(stack[top]) >=
 priority(op)) [
 // out a element from stack
 ]
 // save op into stack
 :x = ')':
 while ( stack(top) != '(' ) [
 // out a element from stack
 ]
 top = top - 1 // not out '(
 :else:
 // output current op
 ]
 i++;
 ]
]
```

例如(a+b)*(c+d)這個式子，依演算法的輸出過程如下：

OP	STACK	OUTPUT
(	(	-
a	(	a
+	(+	a
b	(+	ab
)	-	ab+
*	*	ab+
(	*(	ab+
c	*(	ab+c
+	*(+	ab+c
d	*(+	ab+cd
)	*	ab+cd+
-	-	ab+cd+*

如果要將中序式轉為前序式，則在讀取中序式時是由後往前讀取，而左右括號的處理方式相反，其餘 不變，但輸出之前必須先置入堆疊，待轉換完成後再將堆疊中的 值由上往下讀出，如此就是前序表示式。

實 作：C++

- C++ stack.cpp

```
#include
#include
#define N 50
#define OP 5

char op[OP] = {'(','+','-','*','/',};
int op_priority[OP] = {0,1,1,2,2};//與op[OP]對應,用以存放運算子的優先順序

int priority(char c);
void to_postfix(char infix[], char postfix[]);


char stack[N]; // This is a global variable.
int top=-1;
// -------------------------
//  將資料 item 放入堆疊
// -------------------------

void push(char item){
        if (top>=N-1){
```

```
                printf("Stack full!\n");
                exit(-1);
        }
        stack[++top]=item;
}
// ------------------------------------
// 傳回堆疊頂端的資料，但並非取出
// ------------------------------------
int pop(){
        if (top==-1){
                printf("Stack empty!\n");
                exit(-1);
        }
        return stack[top--];
}

void stackPrint(){
        int i;
        printf("stack =");
        for (i=0; i<=top; i++)
                printf(" %c", stack[i]);
        printf("\n");
}

// ----------------------
// 判斷是否為空堆疊
// ----------------------
bool IsEmpty(void)
{
        return (top < 0) ? true : false;
}

// ----------------------
//  判斷堆疊是否滿溢
// ----------------------
bool IsFull()
{
        return (top >= N - 1) ? true : false;
}

// --------------------------
// 傳回堆疊頂端的資料
// --------------------------
char top_data()
{
        return stack[top];
}

// --------------------------
// 傳回運算子 c 的優先序
// --------------------------
int priority(char c)
{
        int i;

        for( i=0; i < OP; i++)
        if(op[i] == c)
        return op_priority[i];
        return -1;
}

// ------------------------------------
// 將中置式infix轉成後置式postfix
// ------------------------------------
void to_postfix(char infix[], char postfix[])
{
        int i=0, j=-1;
        char x, y;

        while((x=infix[i++]) != '\0'){
                switch(x){
                        case '(' : push(x);
                                        break;
                        case ')' : while(! IsEmpty() && (x=pop()) != '(')
                                             postfix[++j]=x;
                                             break;
                        case '+' :
                        case '-' :
                        case '*' :
                        case '/' : y=top_data();
                                        while(priority(y) >= priority(x)){
```

```
                                                        postfix[++j]=pop();
                                                        y=top_data();
                                        }
                                        push(x);
                                        break;
                        default : // x 觰緽咀啋
                                        postfix[++j]=x;
                }
        }
        while(! IsEmpty())
                postfix[++j]=pop();
        postfix[++j]='\0';
}
bool IsDight(char c)
{
        return c>='0' && c<='9';
}
int calculate(char postfix[])
{
        int point=0;
        while(postfix[point]!='\0')
        {
                while(IsDight(postfix[point]))
                        push(postfix[point++]);
        int a=pop()-'0', b=pop()-'0',c=0;
        switch(postfix[point])
        {
                case'+':c=b+a;
                                break;
                case'-':c=b-a;
                                break;
                case'*':c=b*a;
                                break;
                case'/':c=b/a;
                                break;
        }
        push(c+'0');
        point++;
        }
        return pop()-'0';
}
int main(void)
{
        char infix[50], postfix[50];

        printf("請輸入運算式: ") ;
        scanf("%s",infix);
        to_postfix(infix,postfix);
        printf("\n\n中序式 : %s \t後序式 : %s\n",infix , postfix);
        printf("答案： %d\n", calculate(postfix));
        return 0;
}
```