# 中序式轉後序式

**December 12, 2021**

人們平常使用的運算式，是將運算元放在運算子兩旁，例如 a + b / d 這樣的式子，這稱為中序（infix）表示式；然而電腦剖析運算式時，為了有效率地判斷運算的順序，可將中序表示式轉換為後序（postfix）或前序（prefix）表示式。

## 解法思路

後序表示式又稱為逆向波蘭表示式（reverse polish notation），是由波蘭的數學家盧卡謝維奇提出，例如 (a + b) * (c + d)，表示為後序表示式時是 a b + c d + *。

想以人工轉換計算後序式的話，可以使用括號法，將運算子兩旁的運算元，依先後順序全部括號起來，然後將右括號取代為左邊最接近的運算子（從最內層括號開始），最後去掉全部的左括號就可以完成。例如：

1. a + b * d + c / d
2. ((a + (b * d)) + (c / d))
3. a b d * + c d / +

另一個方式是堆疊法，使用迴圈取出中序式的元素，遇運算元直接輸出，遇到運算子與左括號進行堆疊，堆疊中運算子優先順序若大於等於讀入的運算子優先順序，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。

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)，依演算法的輸出過程如下：

| 元素 | 堆疊 | 輸出 |
| --- | --- | --- |
| ( | ( | - |

| 元素 | 堆疊 | 輸出 |
|---|---|---|
| a | ( | a |
| + | (+ | a |
| b | (+ | ab |
| ) | - | ab+ |
| * | * | ab+ |
| ( | *( | ab+ |
| c | *( | ab+c |
| + | *(+ | ab+c |
| d | *(+ | ab+cd |
| ) | * | ab+cd+ |
| - | - | ab+cd+* |

若要用堆疊法將中序式轉為前序式，使用迴圈由後往前取出中序式的字元，遇運算元直接輸出；遇運算子與右括號進行堆疊；堆疊中運算子優先順序若大於讀入的運算子優先順序，直接輸出堆疊中的運算子，再將讀入的運算子置入堆疊；遇左括號輸出堆疊中的運算子至右括號。

# 程式實作

C | Java | Python | Scala | Ruby | JavaScript | Haskell

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 80

void inToPostfix(char*, char*); // 中序轉後序
int priority(char); // 運算子優先權

int main(void) {
    char infix[MAX] = {'\0'};
    char postfix[MAX]= {'\0'};

    printf("中序運算式：");
    scanf("%s", infix);
    inToPostfix(infix, postfix);

    int i;
    for(i = 0; postfix[i] != '\0'; i++) {
        printf("%c", postfix[i]);
    }

    return 0;
}

void inToPostfix(char* infix, char* postfix) {
    char stack[MAX] = {'\0'};
    int i, j, top;
    for(i = 0, j = 0, top = 0; infix[i] != '\0'; i++) switch(infix[i]) {
        case '(':               // 運算子堆疊
            stack[++top] = infix[i];
            break;
        case '+': case '-': case '*': case '/':
            while(priority(stack[top]) >= priority(infix[i])) {
                postfix[j++] = stack[top--];
```

```
            }
            stack[++top] = infix[i]; // 存入堆疊
            break;
        case ')':
            while(stack[top] != '(') { // 遇 ) 輸出至 (
                postfix[j++] = stack[top--];
            }
            top--;  // 不輸出 (
            break;
        default:  // 運算元直接輸出
            postfix[j++] = infix[i];
    }
    while(top > 0) {
        postfix[j++] = stack[top--];
    }
}

int priority(char op) {
    switch(op) {
        case '+': case '-': return 1;
        case '*': case '/': return 2;
        default:            return 0;
    }
}
```