



# 資料結構

## Data Structure

### Lab 06

姓名： 曾致嘉

學號： 113AB0014

## Lab06-Q1

### Convert Infix Expression to Postfix Using a Stack Implemented with Linked List

#### Code

```
#include <iostream>
#include <cctype>
#include <cstring>
using namespace std;

// 定義 Stack 的節點結構
struct Node {
    char data; // 存儲字符 (運算子或括號)
    Node* next; // 指向下一個節點
};

// 使用 linked list 實作 Stack
class Stack {
private:
    Node* top; // 指向堆疊頂端
    int size;
public:
    Stack() { top = nullptr; size=0; } // 初始化堆疊

    // Push 操作：將元素放入堆疊
    void push(char ch) {
        Node* newNode = new Node(); // 創建新資料
        newNode->data = ch; // 放入資料
        newNode->next = nullptr; // 遇先指向下一筆資料
        if (top== nullptr){ // 當這個 linked list 為空時
            top = newNode; // 直接放入 top
        }else{
            Node *nowNode = top; // 遍歷 link list 找到最後一個資料
            while(nowNode->next!=nullptr){
                nowNode = nowNode->next;
            }
            nowNode->next=newNode; // 放入新資料
        }
    }
};
```

```

        size++; // 資料加一
    }

// Pop 操作：移除並回傳頂端元素
char pop() {
    if (isEmpty()){ // 報錯：linked list 為空
        cout<<"Linked List is empty"<<endl;
        return '\0';
    }
    Node *nowNode = top, *deledNode; // 找到目標資料的前一位
    if (size!=1){ // 資料有多筆
        for(int i=0; i<(size-2); i++){ // 遍歷 linked list
            nowNode = nowNode->next;
        }
        deledNode = nowNode->next; // 找到待查資料
        nowNode->next = nullptr; // 取消資料連結
    }else{ // 當資料只有一筆
        deledNode = top; // 找出待查資料
        top=nullptr; // 取消資料連結
    }
    char ans = deledNode->data; // 取出資料
    delete deledNode; // 刪除 Node
    size--; // 資料減一
    return ans;
}

// Peek 操作：取得頂端元素但不移除
char peek() {
    if (isEmpty()){ // 報錯：linked list 為空
        cout<<"Linked List is empty"<<endl;
        return '\0';
    }
    Node *nowNode = top; // 找到目標資料
    for (int i = 0; i < size-1; i++){
        nowNode = nowNode->next;
    }
    return nowNode->data; // 回傳資料
}

```

```

// 判斷堆疊是否為空
bool isEmpty() {
    return (size==0);
}
};

// 判斷運算子(加減乘除) 的優先順序
int precedence(char op) {
    if ((op>='0' && op<='9') || (op>='A' && op<='Z')){ // 純數字權限最小
        return 4;
    }else if (op=='('){ // 左括號 權限第 2 小
        return 3;
    }else if (op=='+' || op=='-'){ // 加、減號 權限第 3 小
        return 2;
    }else if (op=='*' || op=='/'){ // 乘、除號 權限第 4 小
        return 1;
    }else if (op==')'){ // 右括號 權限最大
        return 0;
    }else{ // 不合法符號
        return -1;
    }
}

// 將中序表達式 (infix) 轉換為後序表達式 (postfix)
void InfixToPostfix(const char* infix, char* postfix) {
    Stack stack;
    char back;
    int infixIndex =0,postIndex=0;
    while (infix[infixIndex]!='\0'){
        switch (precedence(infix[infixIndex])){
            case 4: // 一般數字
                postfix[postIndex++]=infix[infixIndex];
                break;
            case 3: // 左括號 '('
                stack.push(infix[infixIndex]);
                break;
            case 2: // 加減 '+' '-'
                while (!stack.isEmpty()){

```

```

        back= stack.peek(); //抓出最上面一筆
        if (precedence(back)<=2){ // 如果權限比較大就取出來放
            postfix[postIndex++]= stack.pop();
        }else break; // 沒有的話，結束判斷權級
    }
    stack.push(infix[infixIndex]); //推入本次結果
    break;
case 1: // 乘除 '*' '/'
    stack.push(infix[infixIndex]); //推入本結果
    break;
case 0: // 右括號 ')'
    back = stack.pop();
    while (precedence(back)!=3){ //撈資料撈到左括號(3)為止
        postfix[postIndex++]=back;
        back=stack.pop();
    }
    break;
default:
    break;
}
infixIndex++; //處理下一個資料
}
while (!stack.isEmpty()){ // 判斷完成取完剩餘資料
    char back=stack.pop();
    postfix[postIndex++]=back;
}
postfix[postIndex++]='\0'; //下入終止符號
}
int main() {
    char infix[100], postfix[100];
    cout << "Enter an Infix expression: ";
    cin.getline(infix, 100); // 輸入中序表達式

    InfixToPostfix(infix, postfix); // 轉換為後序表達式
    cout << "Postfix expression: " << postfix << endl; // 輸出後序表達式

    return 0;
}

```

## Discussion Section

測資分析：

case 1:

	stack					
input	stack[0]	stack[1]	stack[2]	stack[3]	stack[4]	anser
(	(					
A	(					A
+	(	+				A
B	(	+				AB
)						AB+
*	*					AB+
D	*					AB+D
+	+					AB+D*
E	+					AB+D*E
/	+	/				AB+D*E
(	+	/	(			AB+D*E
F	+	/				AB+D*EF
+	+	/	(	+		AB+D*EF
A	+	/	(	+		AB+D*EFA
*	+	/	(	+	*	AB+D*EFA
D	+	/	(	+	*	AB+D*EFAD
)	+	/				AB+D*EFAD*+
+	+					AB+D*EFAD*+ / +
C	+					AB+D*EFAD*+ / +C
						AB+D*EFAD*+ / +C+

case 2:

	Stack						
input	stack[0]	stack[1]	stack[2]	stack[3]	stack[4]	stack[5]	anser
A							A
+	+						A
B	+						AB
*	+	*					AB
(	+	*	(				AB
C	+	*	(				ABC
*	+	*	(	*	(		ABC
(	+	*	(	*	(		ABC
D	+	*	(	*	(		ABCD
+	+	*	(	*	(	+	ABCD
)	+	*	(				ABCD+
)							ABCD+*
							ABCD+**+

case 3:

	Stack				
input	stack[0]	stack[1]	stack[2]	stack[3]	anser
3					3
+	+				3
1	+				31
2	+				312
*	+	*			312
+	+				312*+
(	+	(			312*+
4	+	(			312*+4
+	+	(	+		312*+4
5	+	(	+		312*+45
*	+	(	+	*	312*+45
6	+	(	+	*	312*+456
)	+				312*+456*+
/	+	/			312*+456*+
(	+	/	(		312*+456*+
9	+	/	(		312*+456*+9
-	+	/	(	-	312*+456*+9

7	+	/	(	-	312*+456*+97
)	+	/			312*+456*+97-
					312*+456*+97-/+

結果：

The screenshot displays a C++ IDE with a file named `LAB06_Q1.cpp` open. The code implements a function `InfixToPostfix` that converts an infix expression to a postfix expression using a stack. The code includes comments in Chinese explaining the logic, such as handling operator precedence and parentheses. The terminal window on the right shows the execution of the program, demonstrating the conversion of three different infix expressions to their postfix equivalents.

```

LAB06_Q1.cpp
97 void InfixToPostfix(const char* infix, char* postfix) {
98     int infixIndex = 0, postfixIndex = 0;
99     while (infix[infixIndex] != '\0') {
100         switch (precedence(infix[infixIndex])) {
101             case 4: // 一般數字
102                 postfix[postfixIndex++] = infix[infixIndex];
103                 break;
104             case 3: // 左括號 '('
105                 stack.push(infix[infixIndex]);
106                 break;
107             case 2: // 加減 '+' '-'
108                 while (!stack.isEmpty()) {
109                     back = stack.peek(); // 取出最上面一筆
110                     if (precedence(back) <= 2) { // 如果權限比較大就取出來放
111                         postfix[postfixIndex++] = stack.pop();
112                     } else break; // 沒有的話，結束判斷權限
113                 }
114                 stack.push(infix[infixIndex]); // 推入本次結果
115                 break;
116             case 1: // 乘除 '*' '/'
117                 stack.push(infix[infixIndex]); // 推入本次結果
118                 break;
119             case 0: // 右括號 ')'
120                 back = stack.pop();
121                 while (precedence(back) != 3) { // 開資料擷到左括號(3)為止
122                     postfix[postfixIndex++] = back;
123                     back = stack.pop();
124                 }
125                 break;
126             default:
127                 break;
128         }
129         infixIndex++; // 處理下一個資料
130     }
131     while (!stack.isEmpty()) { // 判斷完成取完剩餘資料
132         char back = stack.pop();
133         postfix[postfixIndex++] = back;
134     }
135 }
136
C:\WINDOWS\system32\cmd.exe
C:\Users\user>cd C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB6
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB6>main.exe
Enter an Infix expression: (A+B)*D+E/(F+A*D)+C
Postfix expression: AB+D*EFAD*+/+C+
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB6>main.exe
Enter an Infix expression: A+B*(C*(D+E))
Postfix expression: ABCDE+*+*+
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB6>main.exe
Enter an Infix expression: 3+1*2+(4+5*6)/(9-7)
Postfix expression: 312*+456*+97-/+/+
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB6>
  
```