Stack

Stack 的特性

- **先進後出（LIFO, Last In First Out）**
- 只允許在「頂端（top）」插入與刪除元素。

✅ Stack ADT（抽象資料型態）基本功能

函式	說明
push()	將元素加入堆疊頂端
pop()	移除堆疊頂端元素
top()	取得堆疊頂端元素（不移除）
isEmpty()	判斷堆疊是否為空
size()	回傳目前堆疊中的元素個數

🧱 Stack ADT 類別設計（以陣列為基礎）

```
#include <iostream>
#include <stdexcept> // for std::runtime_error

template<typename T>
class Stack {
private:
    T* data;
    int capacity;
    int topIndex;

public:
    Stack(int size = 100);    // 建構子
    ~Stack();                 // 解構子

    void push(const T& item); // 加入元素
```

```
    void pop();            // 移除元素
    T top() const;         // 取得頂端元素
    bool isEmpty() const;  // 是否為空
    int size() const;      // 目前元素數量
};
```

🔧 Stack ADT 成員函式實作

```
// 建構子
template<typename T>
Stack<T>::Stack(int size) : capacity(size), topIndex(-1) {
    data = new T[capacity];
}

// 解構子
template<typename T>
Stack<T>::~Stack() {
    delete[] data;
}

// push
template<typename T>
void Stack<T>::push(const T& item) {
    if (topIndex + 1 >= capacity)
        throw std::runtime_error("Stack overflow");
    data[++topIndex] = item;
}

// pop
template<typename T>
void Stack<T>::pop() {
    if (isEmpty())
        throw std::runtime_error("Stack underflow");
    --topIndex;
}
```

```
// top
template<typename T>
T Stack<T>::top() const {
    if (isEmpty())
        throw std::runtime_error("Stack is empty");
    return data[topIndex];
}

// isEmpty
template<typename T>
bool Stack<T>::isEmpty() const {
    return topIndex == -1;
}

// size
template<typename T>
int Stack<T>::size() const {
    return topIndex + 1;
}
```

🧪 使用範例

```
int main() {
    Stack<int> s(10);

    s.push(5);
    s.push(10);
    s.push(15);

    std::cout << "Top: " << s.top() << "\n"; // 15
    s.pop();
    std::cout << "Top after pop: " << s.top() << "\n"; // 10

    std::cout << "Size: " << s.size() << "\n"; // 2
    std::cout << "Is empty? " << (s.isEmpty() ? "Yes" : "No") << "\n";
```

```
    return 0;
}
```

📌 備註

- 上面是以陣列（array）實作，缺點是容量固定。
- 若要彈性大小，可以改用 `std::vector<T>` 或 `std::list<T>`。
- 也可使用 linked list（串列）實作，彈性更佳。

Infix to Postfix

WHILE :NOT END OF INFIX STRING
TOKEN = GET NEXT ELEMENT OF INFIX STRING
IF TOKEN IS AN OPERAND:
APPEND TOKEN TO POSTFIX STRING
ELSE IF TOKEN IS AN OPERATOR:
WHILE(STACK NOT EMPTY & PRCD(TOP,TOKEN)):
TOP OPERATOR = POP(STACK);
APPEND TOP OPERATOR TO POSTFIX STRING
END WHILE
IF TOKEN == ")":
POP(STACK); //pop the "("
ELSE:
PUSH(STACK,TOKEN);
END IF
END IF

實作

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype>  // for isdigit, isalpha

// 判斷運算子優先權函數，優先權越大數字越大
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
```

```
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}

// 判斷是否為運算子
bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '^' || c == '(' || c == ')';
}

// 判斷優先權比較: 當 stack top 運算子優先權 >= 當前運算子優先權時回傳 true
bool prcd(char stackTop, char current) {
    if (stackTop == '(') return false; // '('優先權最低，不彈出
    return precedence(stackTop) >= precedence(current);
}

std::string infixToPostfix(const std::string& infix) {
    std::stack<char> st;
    std::string postfix;

    for (size_t i = 0; i < infix.length(); ++i) {
        char token = infix[i];

        // 如果是字母或數字，直接加到 postfix 字串
        if (std::isalnum(token)) {
            postfix += token;
        }
        // 如果是運算子
        else if (isOperator(token)) {
            if (token == '(') {
                st.push(token);
            }
            else if (token == ')') {
                // 遇到右括號，彈出直到遇到左括號
                while (!st.empty() && st.top() != '(') {
                    postfix += st.top();
                    st.pop();
                }
```

```
            if (!st.empty()) st.pop(); // 彈出左括號 '('，但不加入 postfix
        }
        else {
            // 其他運算子，彈出堆疊中優先權較高或相同的運算子
            while (!st.empty() && prcd(st.top(), token)) {
                postfix += st.top();
                st.pop();
            }
            st.push(token);
        }
    }
    // 若遇到空白等忽略可自行加條件
  }

  // 將堆疊剩餘運算子加入 postfix
  while (!st.empty()) {
    postfix += st.top();
    st.pop();
  }

  return postfix;
}

int main() {
  std::string infixExpr = "A*(B+C)/D";
  std::string postfixExpr = infixToPostfix(infixExpr);
  std::cout << "Infix: " << infixExpr << std::endl;
  std::cout << "Postfix: " << postfixExpr << std::endl;

  return 0;
}
```