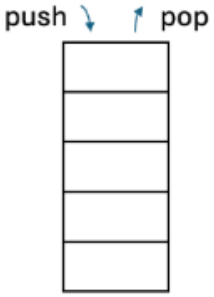
# Stack

## 一、核心概念

一種**線性資料結構**，其運作遵循 **LIFO (Last-In, First-Out，後進先出)** 原則。

**- 生活細節比喻**：就像一疊盤子，你只能從最上方放盤子，也只能從最上方取走盤子。

**- 關鍵特性**：所有的插入（Insert）與刪除（Delete）都發生在同一端，稱為**頂端（Top）**。




## 二、 基本操作 (Preliminary Operations)

時間複雜度通常皆為O(1) :

| 操作 | 描述 |
| --- | --- |
| **push(item)** | 將一個元素放入堆疊的頂端 |
| **pop()** | 移除並回傳堆疊頂端的元素 |
| **peek() / top()** | 查看頂端元素的值，但不將其移除 |
| **isEmpty()** | 檢查堆疊是否為空（若 top == -1 或 NULL） |
| **isFull()** | 檢查堆疊是否已滿（僅限陣列實現，若 top == MAX_SIZE - 1） |

## 三、 兩種實現方式對比

### 1. 陣列實現 (Array-based Stack)

- 核心變數：一個固定大小的陣列 stack[MAX_SIZE] 與一個整數索引 top（初始值為 -1）。
- 優點：存取速度快，O(1)。
- 缺點：
  - 大小固定，容易發生 Overflow（上溢）。

    - 若需擴充容量（Resizing），必須重新分配記憶體並複製資料（使用 realloc），負擔較重。
- 注意：將陣列索引 0 作為 Bottom（底部）較佳，因為當需要擴張空間時，只需修改 max_size 往後延伸即可。

**2. 鏈結串列實現 (Linked List-based Stack)**

- 核心變數：一個指標 Node* top，指向串列的頭部（Head）。
- 運作邏輯：
    - Push：在頭部新增節點（newNode -> next = top;  top = newNode;）。
    - Pop：移除頭部節點（top = top -> next; ）。
- 優點：動態分配記憶體，沒有固定大小限制，不會有陣列的上溢問題（除非系統記憶體耗盡）。
- 缺點：每個節點需額外存儲指標（Pointer overhead），且頻繁的 malloc/free 會產生額外負擔。

## 四、 重點筆記

**1. 條件判定 (Edge Cases)**

- Empty 判定：
    - 陣列：top == -1
    - 鏈結串列：top == NULL
- Full 判定：
    - 陣列：top == MAX_SIZE - 1
    - 鏈結串列：通常不需判定，除非記憶體用盡

**2. 實際應用場景 (Real-world Applications)**

「哪些問題適合用 Stack 解決？」：

- **撤銷功能 (Undo functionality)：**紀錄操作步驟
- **函式呼叫 (Function calls)：**系統使用 Stack 來存儲區域變數與回傳位址
- **運算式求值 (Expression evaluation)：**如將中序表達式轉為後序
- **遞迴問題：**例如經典的河內塔 (Tower of Hanoi)

**3. Stack vs. Queue 的本質區別**

- Stack 是 **LIFO**（後進先出），所有動作在同一端
- Queue 是 **FIFO**（先進先出），一端進（Rear）、一端出（Front)

## 五、 靈活運用練習

Q：如果你要實作一個支援無限回退（Undo）的繪圖軟體，你會選哪種實現方式？

A：建議選 Linked List-based Stack。因為繪圖步驟可能極多，使用陣列容易遇到大小限制或頻繁搬移資料的效能瓶頸，鏈結串列能隨時動態增加節點來記錄新步驟。