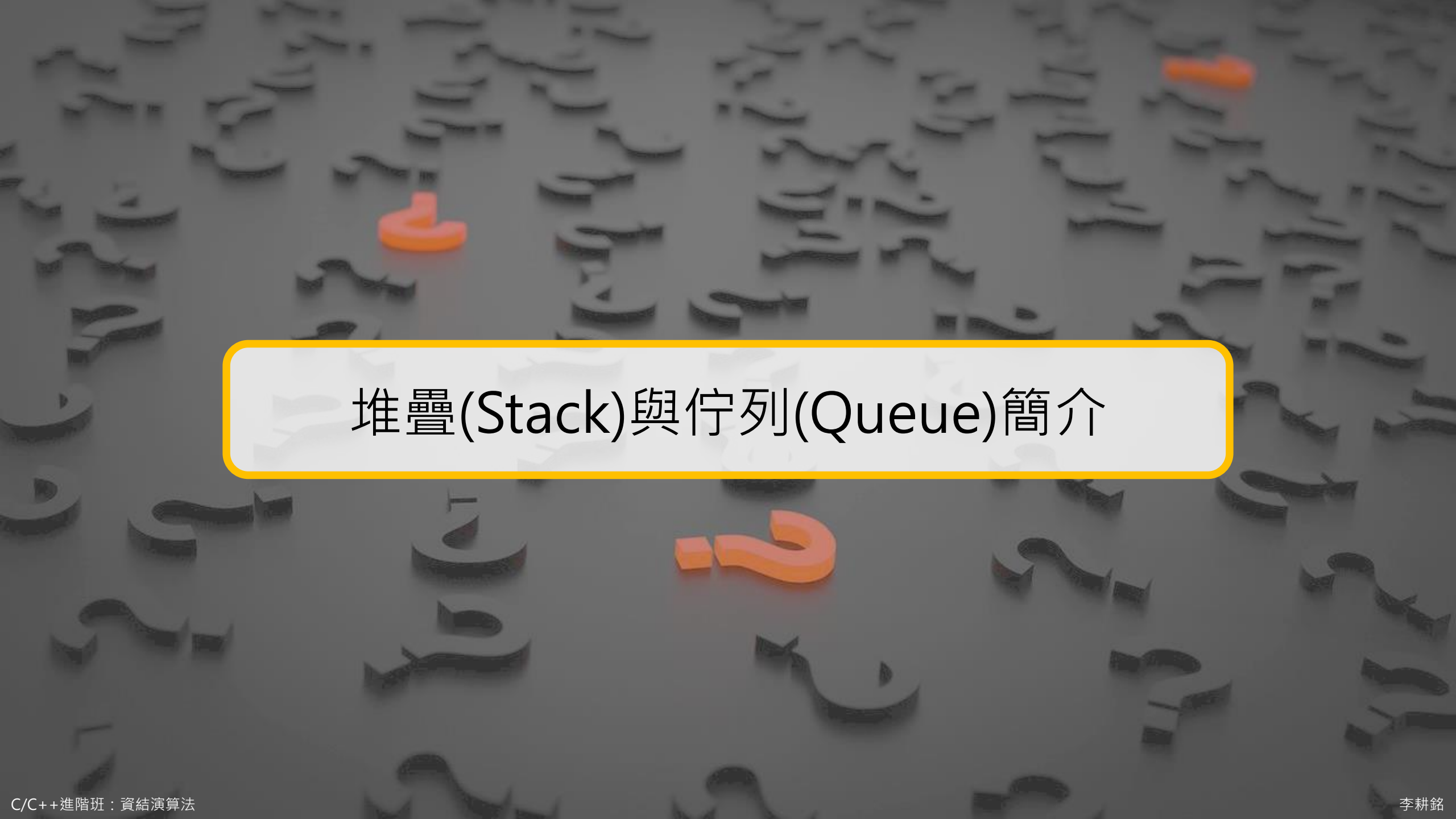
C/C++ 進階班
資料結構

堆疊
(Stack)

李耕銘

課程大綱

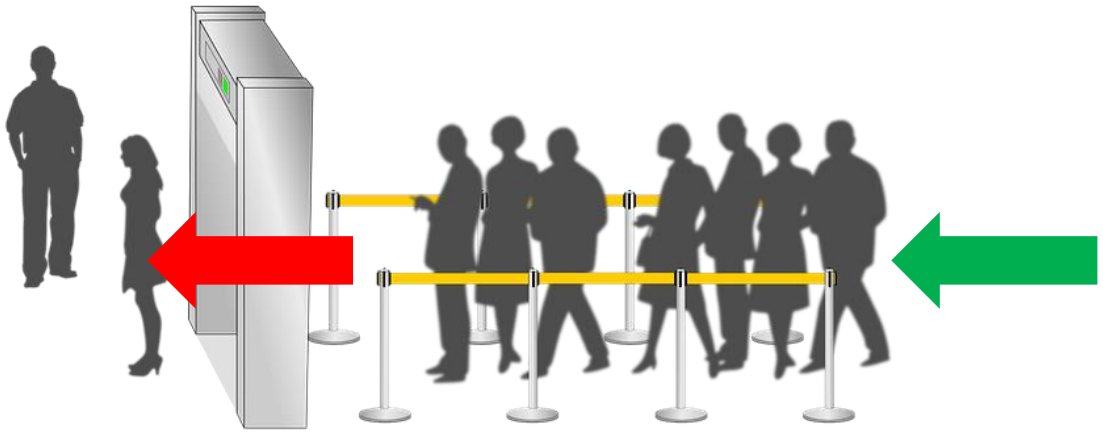
- 堆疊(Stack)與佇列(Queue)簡介
- 堆疊(Stack)簡介
- 堆疊(Stack)實作
- C++ STL 中的堆疊
- 堆疊(Stack)應用



堆疊(Stack)與佇列(Queue)簡介

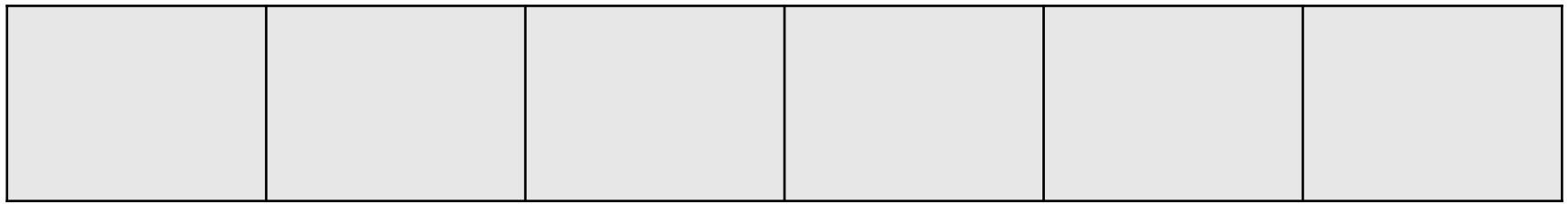
堆疊(Stack)與佇列(Queue)

- 堆疊(Stack)與佇列(Queue)
  - 都只能操作兩端的值
  - 不支援搜索
- 堆疊(Stack)
  - 插入、刪除在同側
  - last-in-first-out(LIFO)
- 佇列(Queue)
  - 插入、刪除在不同側
  - first-in-first-out(FIFO)

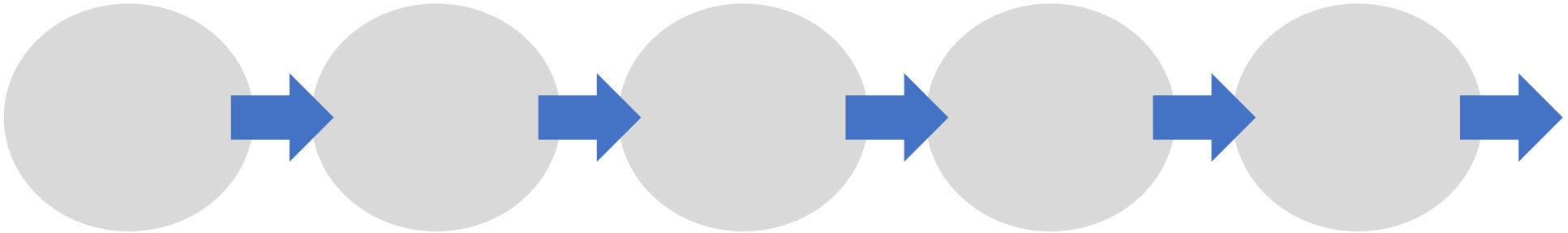


實作方式

陣列(Array)



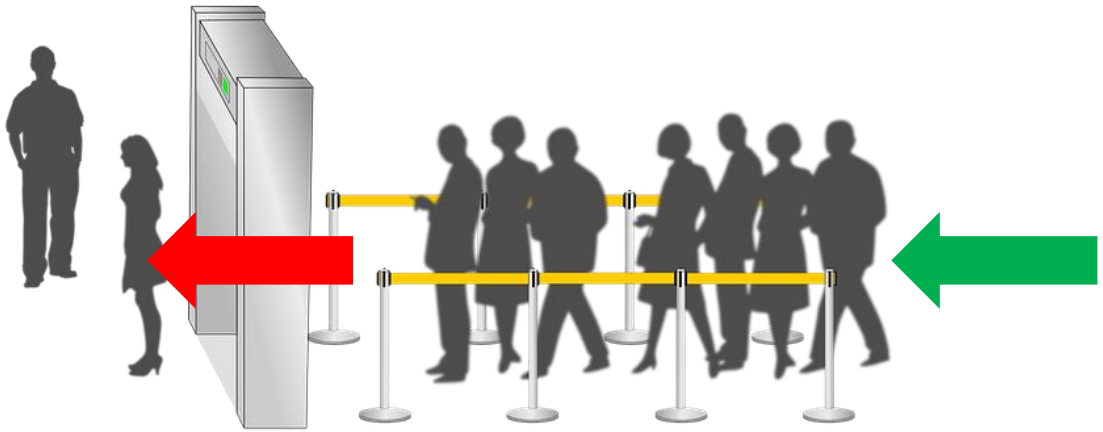
鏈結串列(Linked list)



陣列練習 stack、鏈結串列練習 queue

複雜度

- 堆疊(Stack)與佇列(Queue)
  - 新增：O(1)
  - 刪除：O(1)
  - 查詢：O(1)
- 只能在特定位置新增/刪除/查詢
  - 實務上限制更像是一種保護

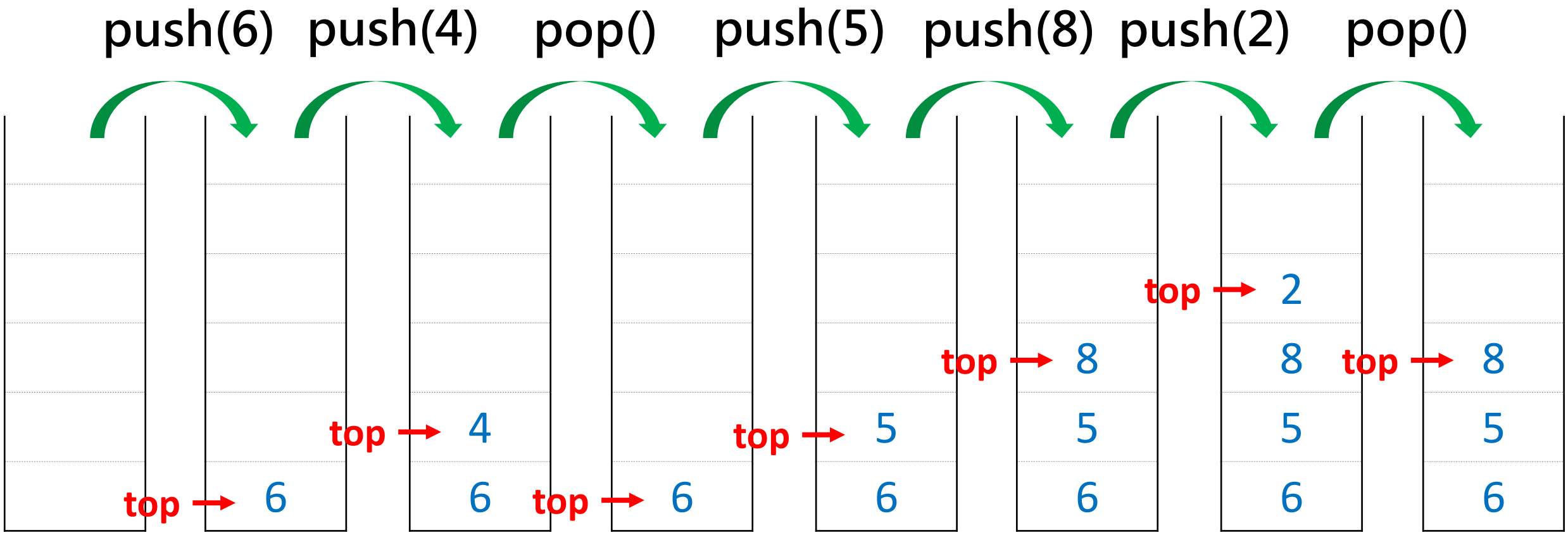


堆疊(Stack) 簡介

堆疊(Stack)

- 堆疊(Stack)
  - 插入、刪除在同側
  - last-in-first-out(LIFO)
- 常見的操作
  1. push：新增一筆資料
  2. pop：刪除一筆資料
  3. top：回傳最末端的資料
  4. empty：確認stack裡是否有資料
  5. size：回傳 stack 的資料個數

堆疊(Stack)


push(6)
push(4)
pop()
push(5)
push(8)
push(2)
pop()
top
6
top
4
6
top
6
top
5
6
top
8
5
6
top
2
8
5
6
top
8
5
6

Practice

給定 stack = {1, 2, 3}，方向為右進右出，經過以下操作後，該 stack 的最後內容為何？

1. push(4)
2. pop()
3. push(5)
4. push(6)
5. push(7)
6. pop()
7. pop()

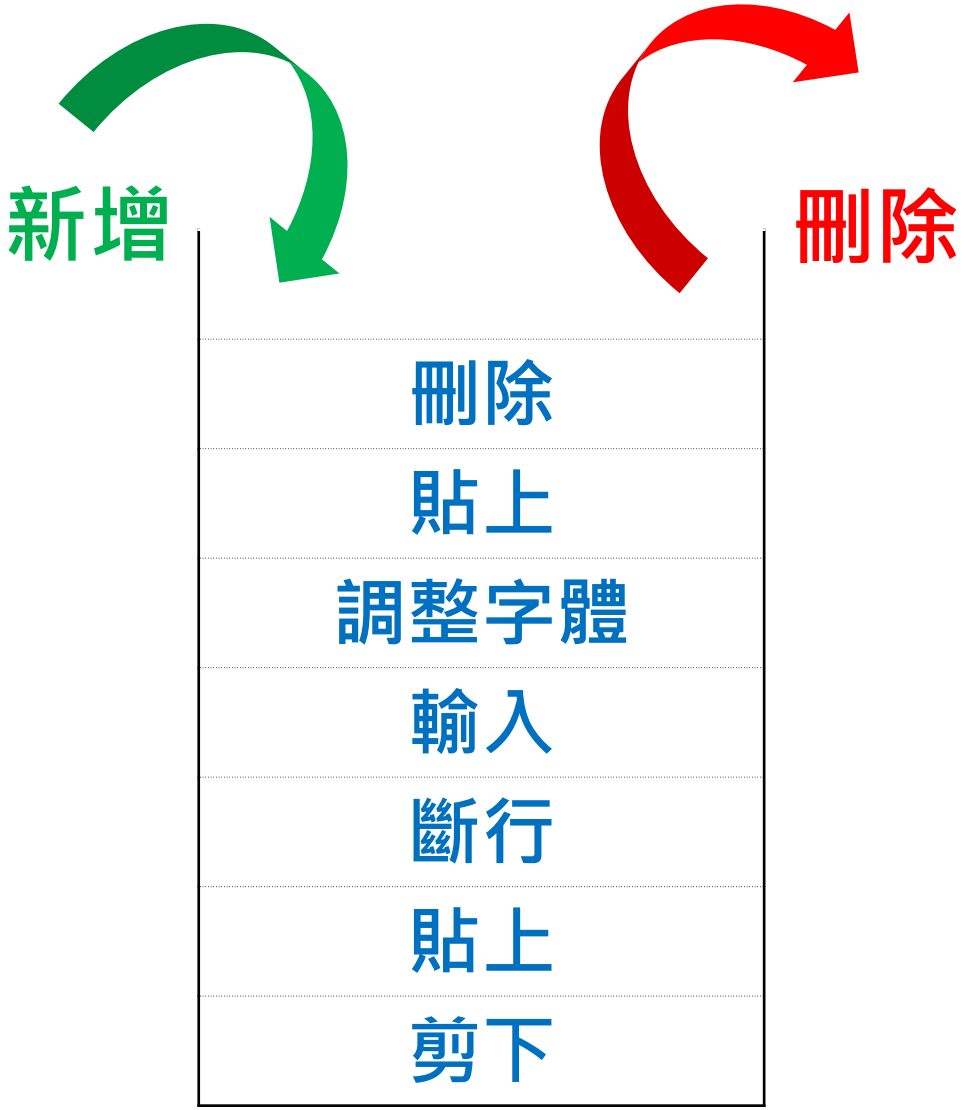
Practice

給定 stack = {1, 2, 3}，方向為右進右出，經過以下操作後，該 stack 的最後內容為何？

1. push(4)： {1, 2, 3, 4}
2. pop()： {1, 2, 3}
3. push(5)： {1, 2, 3, 5}
4. push(6)： {1, 2, 3, 5, 6}
5. push(7)： {1, 2, 3, 5 ,6 ,7}
6. pop()： {1, 2, 3, 5, 6}
7. pop()： {1, 2, 3, 5}

堆疊(Stack)的用途

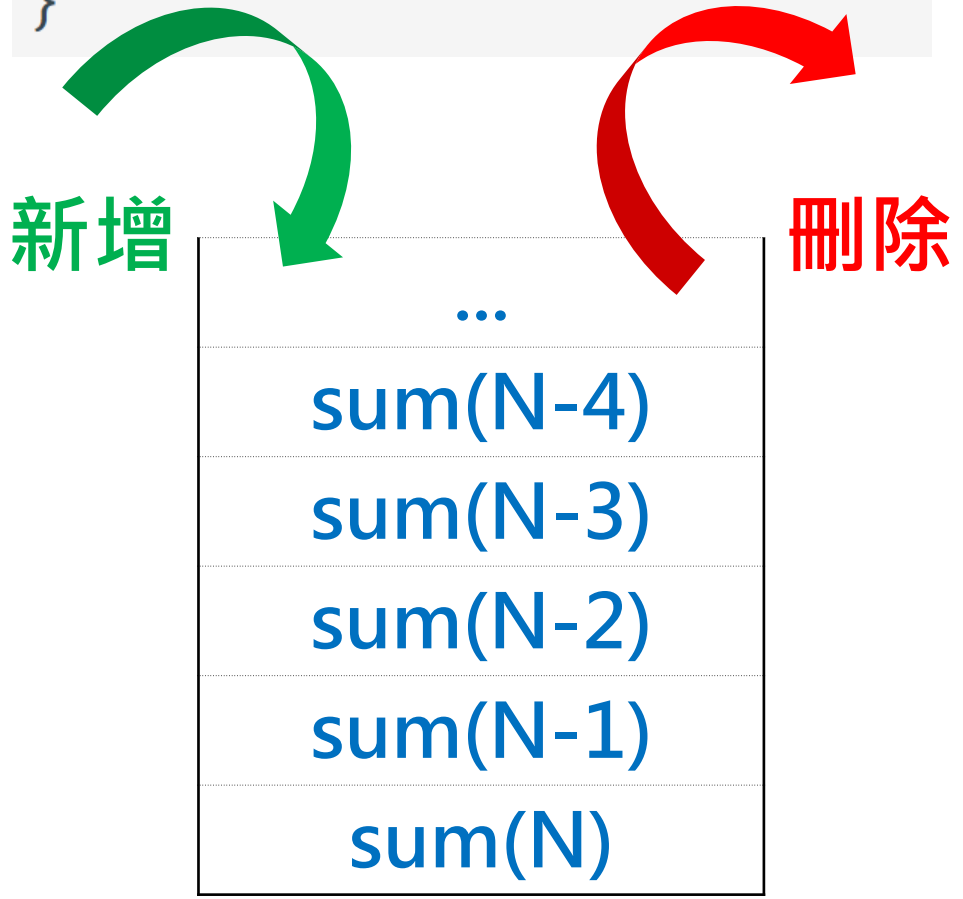
- 依序紀錄先前的資訊
  - 常用來回復到先前的狀態
    1. 瀏覽器回到上一頁
    2. 編輯器復原
    3. 編譯器的解析(parse)
    4. 函式呼叫(遞迴)
- 迷宮探索、河內塔、發牌
  - Depth-First Search (演算法)
- 無法得知 stack 裡有哪些資料
  - 只能以 pop() 一個個把資料拿出來



Stackoverflow

- 呼叫函式要記錄當下位置以便之後返回
  - 最後呼叫的函式會優先返回
  - 作業系統會以 stack 處理函式呼叫
- 呼叫太多層函式 → Stack Overflow

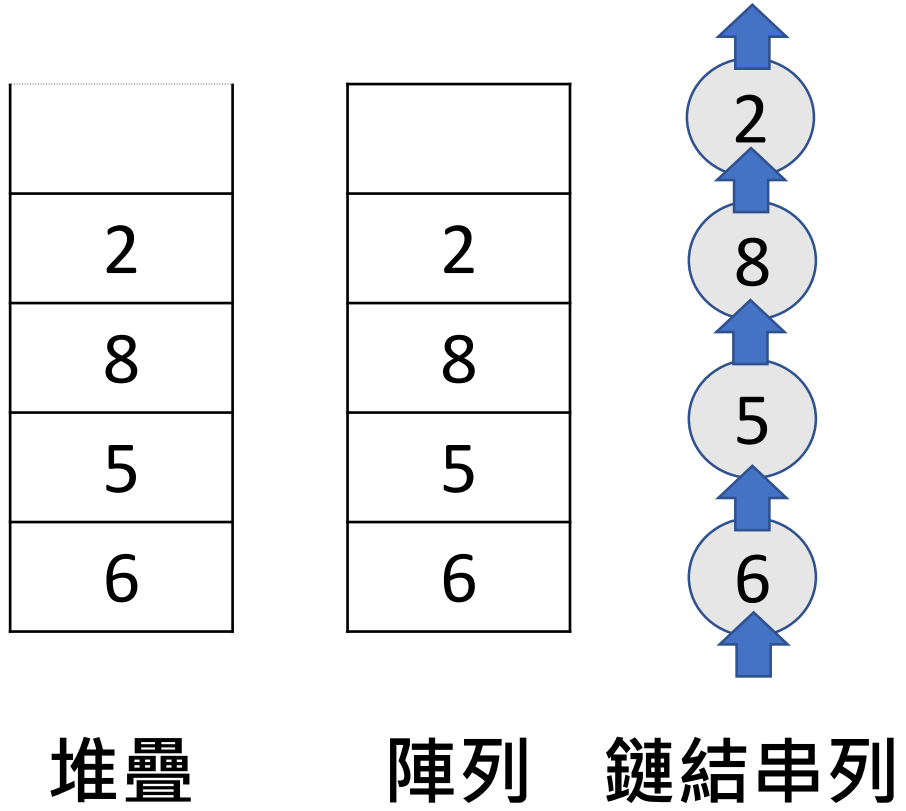

```
int sum(int N){
    return sum(N-1) + N;
}
```



堆疊(Stack) 實作

堆疊實作

- 堆疊(Stack)
  - 可以以陣列或鏈結串列來實作
    1. 以陣列示範堆疊 (Stack)
    2. 再以鏈結串列練習佇列 (Queue)
  - 這裡的堆疊有大小限制
    1. 解決方式：realloc
    2. 或是直接使用 vector



堆疊實作

- 資料成員
  1. 空間大小(Capacity)
     - 容器能容納的上限
  2. 最上層位置(Top_Index)
     - 容器內最上層資料的位置
  3. 指標(Pointer)
     - 指向儲放資料的空間
- 函式成員
  1. 新增(Push)
  2. 刪除(Pop)
  3. 大小(Size)
  4. 空(Empty)
  5. 取值(Top)

堆疊實作

- 以陣列實作堆疊的類別
  1. 先用 malloc 準備好一個陣列
  2. Top_Index (即索引值)初始化為 -1
  3. 再分別完成下列函式
     a) Top
     b) Empty
     c) Size
     d) Double_Capacity
     e) Push
     f) Pop
     g) Print_Stack (正常不會有)

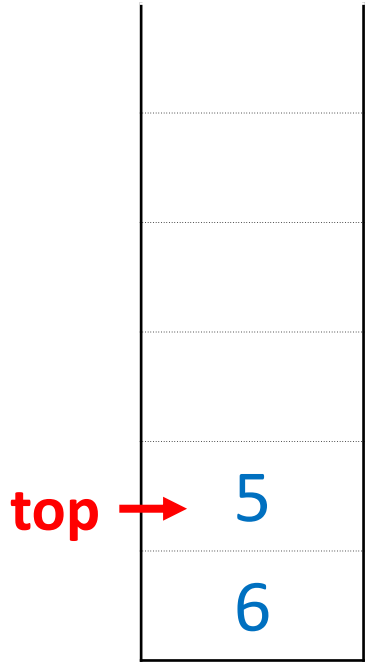
```
template<typename T>
class Stack_Array{
    private:
        int Capacity;
        int Top_Index;
        T* Pointer;
        void Double_Capacity();
    public:
        Stack_Array(int=0);
        bool Empty();
        int Size();
        T Top();
        void Push(T);
        void Pop();
        void Print_Stack();
};
```

堆疊實作

- Top→回傳最末端的資料
  1. 確認 Stack 不為空
  2. 回傳 Top_Index 指到的資料

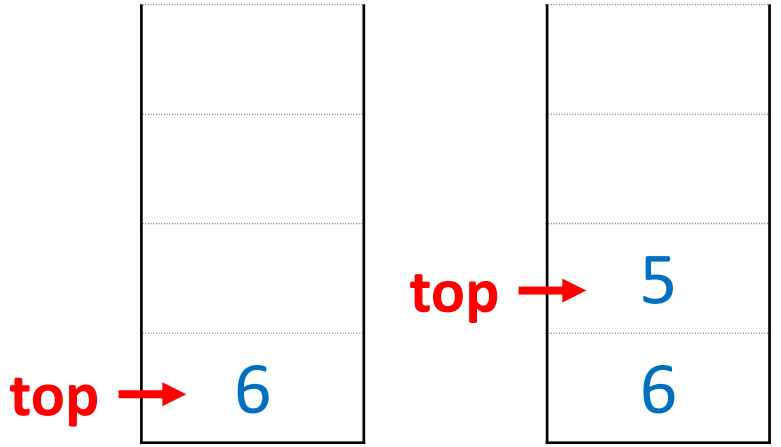
- Empty→確認 Stack 裡是否有資料
  1. 確認 Top_Index 是否 ≥ 0

- Size→回傳 Stack 內的資料個數
  1. 回傳 Top_Index + 1



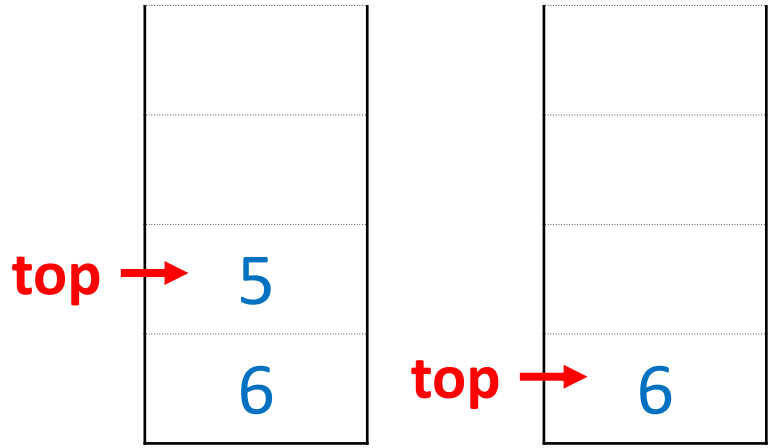
堆疊實作

- Push→新增一筆資料
  1. 確認空間足夠，空間不夠時用 realloc 重新配置空間 (Capacity = Capacity * 2)
  2. Top_Index 往後移一格 (Top_Index++)
  3. 把資料 assign 給 Top_Index 指到的位置



- Pop→刪除一筆資料
  1. 確認 Stack 不為空
  2. Top_Index 往前移一格(Top_Index --)

不需要刪除資料！



Example Code

Mission

初始化一個堆疊，並完成：

1. 建構式
2. Print_Stack()

```
template<typename T>
class Stack_Array{
    private:
        int Capacity;
        int Top_Index;
        T* Pointer;
        void Double_Capacity();
    public:
        Stack_Array(int=0);
        bool Empty();
        int Size();
        T Top();
        void Push(T);
        void Pop();
        void Print_Stack();
};
```

Practice

Mission

完成以下函式：

1. Empty()
2. Size()
3. Double_Capacity()
4. Top()
5. Push()
6. Pop()

```
template<typename T>
class Stack_Array{
    private:
        int Capacity;
        int Top_Index;
        T* Pointer;
        void Double_Capacity();
    public:
        Stack_Array(int=0);
        bool Empty();
        int Size();
        T Top();
        void Push(T);
        void Pop();
        void Print_Stack();
};
```

C++ STL 的堆疊

Container(容器)

- 常見的容器：
  - Container adapter
    - ✓ 提供特殊的介面/資料存取順序
    - ✓ stack
    - ✓ queue
    - ✓ priority_queue

STL 中的 stack

- C++
  - stack 是 stack
  - queue 是 queue
  - priority_queue 是 priority queue
  - deques 是 double-ends queue
    - ✓ 插入、搜尋、刪除：$O(1)$
    - ✓ 但只能在特定位置！
- Python
  - list 治百病！

STL 中的 stack

- stack 與 queue 的使用
  - 引用函式庫

```
#include <stack>
#include <queue>
```

  - 宣告

```
stack<datatype> stack_name;
queue<datatype> queue_name;
```

STL 中的
stack 與 queue 沒有 iterator

stack 的操作

- stack 的操作
  - 新增一筆資料
    stack.push(value);
  - 刪除一筆資料
    stack.pop();
  - 回傳一筆資料
    stack.top();
  - 判斷 stack 是否為空
    stack.empty();
  - 回傳 stack 長度
    stack.size();

stack 的操作

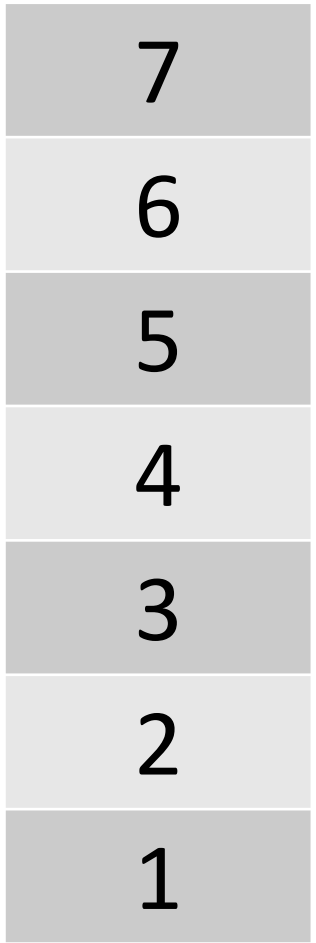
```
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    stack<int> data;
    for(int i=0;i<10;i++)
        data.push(i);
    // 0 1 2 3 4 5 6 7 8 9
    cout << data.top() << endl;
    // 9
    data.pop();
    data.pop();
    // 0 1 2 3 4 5 6 7
    cout << data.top() << endl;
    // 7
    return 0;
}
```

stack 的操作

印出 stack 內的資料

```
void print_stack(stack<int>& s){
    if(s.empty())
        return ;
    int data = s.top();
    s.pop();
    print_stack(s);
    cout << data << " ";
    s.push(data);
}
```

1 2 3 4 5 6 7



Example Code

Mission

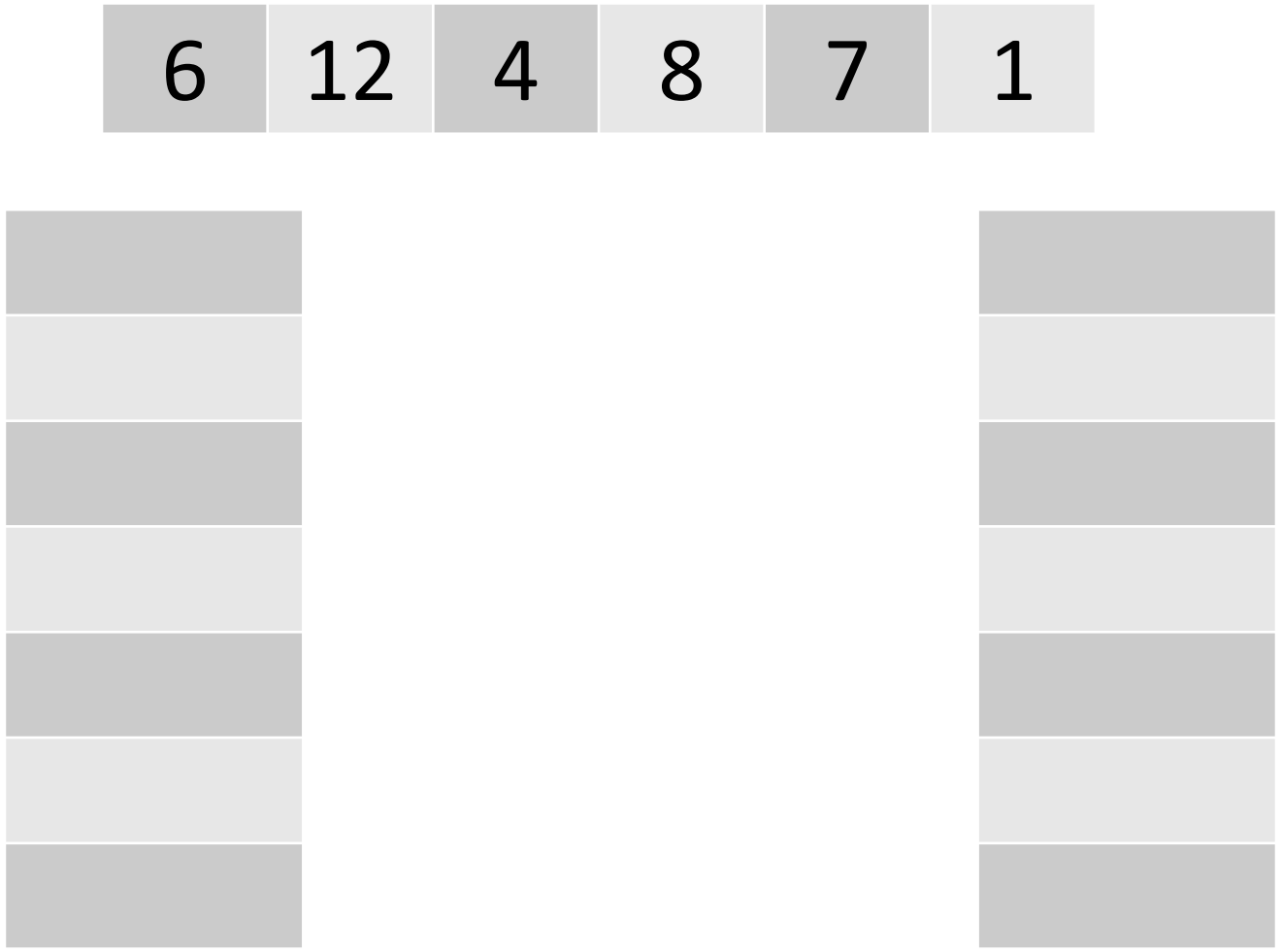
LeetCode #155 Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

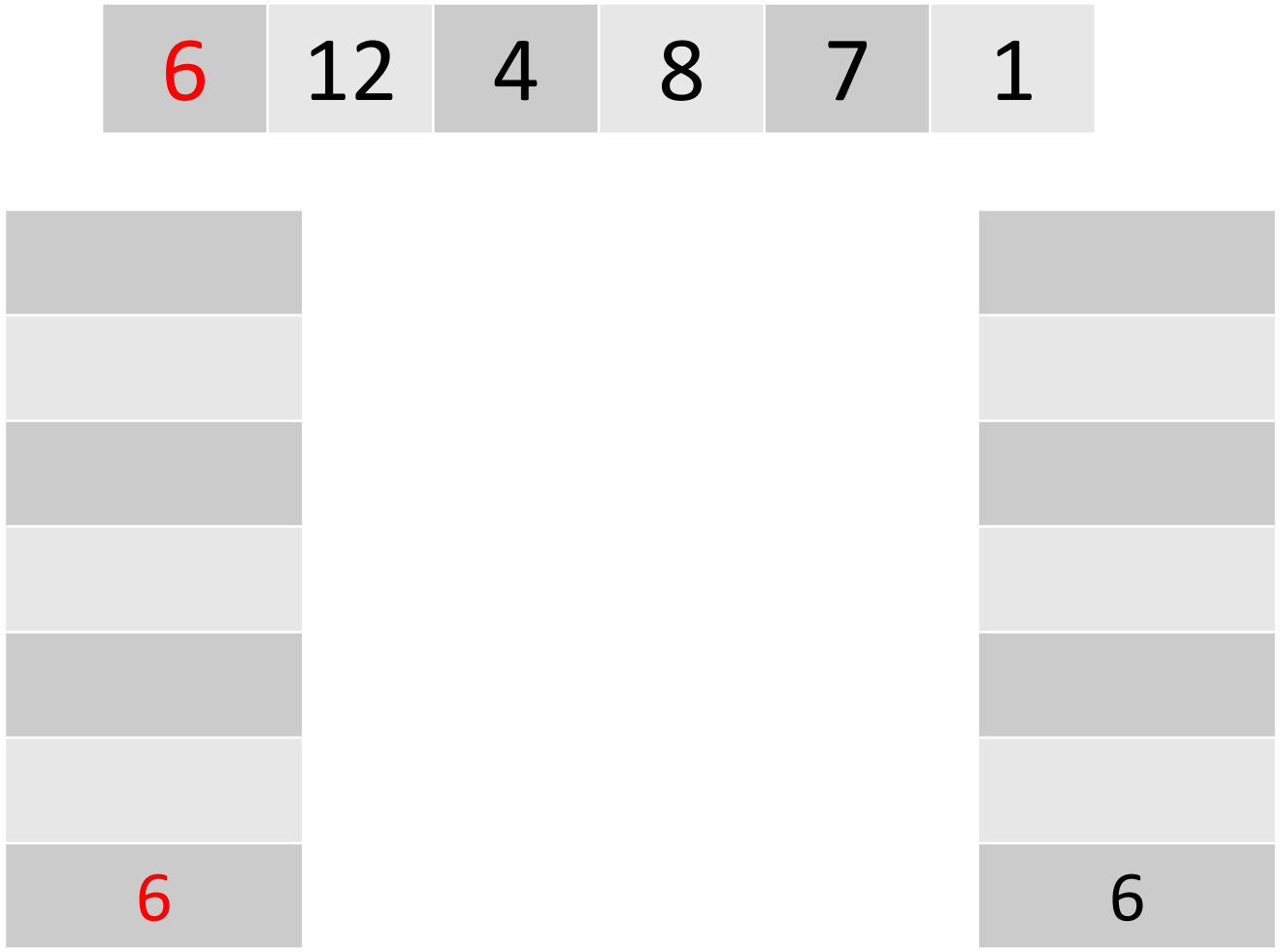
- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Ref: https://leetcode.com/problems/min-stack/

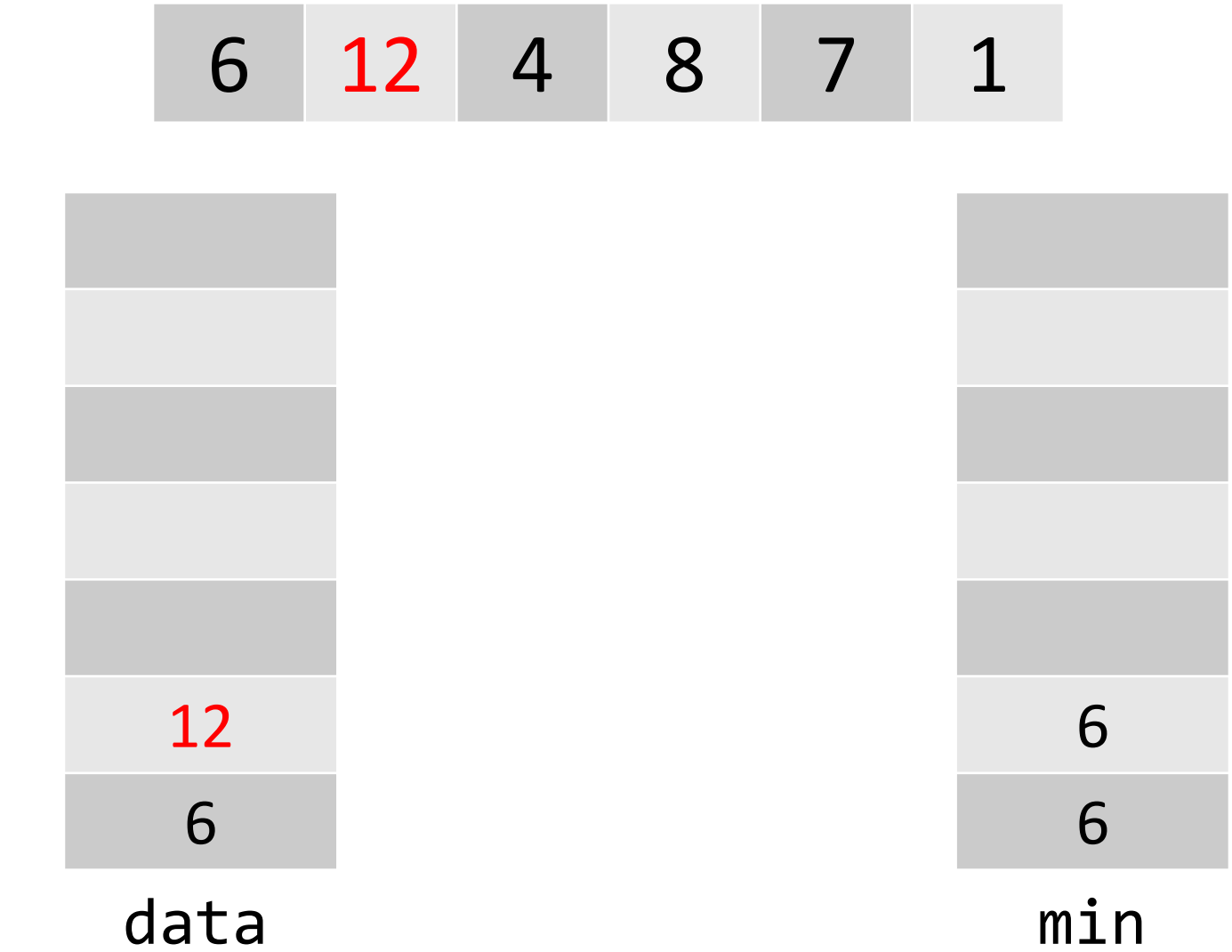
Example Code



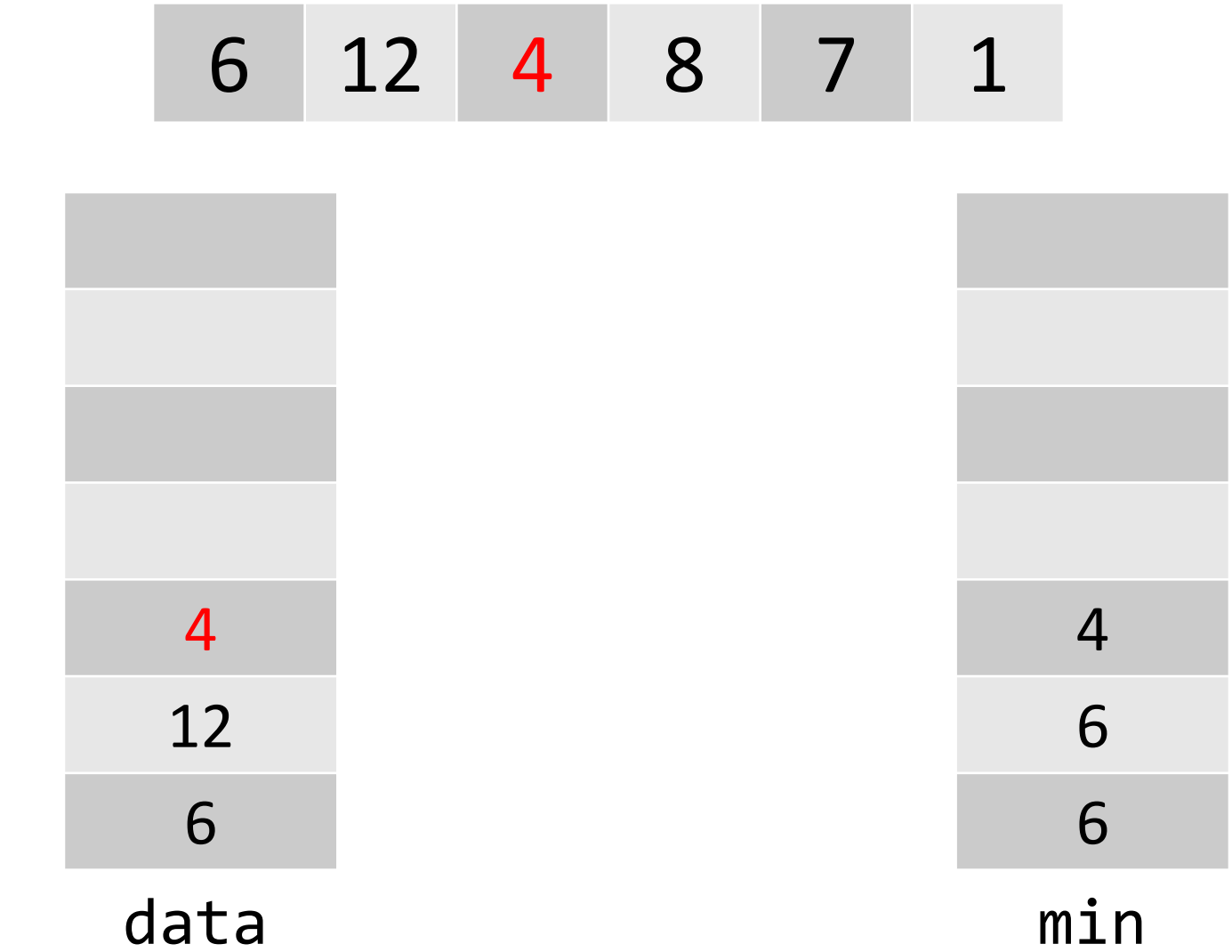
Example Code



Example Code



Example Code



Example Code

6	12	4	8	7	1

Example Code

6	12	4	8	7	1

Example Code

6	12	4	8	7	1

1
7
8
4
12
6

data

1
4
4
4
6
6

min

Practice

Mission

- 利用堆疊 (Stack) 撰寫除錯工具
  - ➢ 該除錯工具可以確認某字串中的括號是否有成對，例如：
    1. {[(3+6)]*8}+9 → True
    2. {[(3+6]*8}+9 → False
    3. {(3+6)]*8}+9 → False

Practice

Mission

LeetCode #20. Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Ref: https://leetcode.com/problems/valid-parentheses/

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

- 試著利用堆疊 (Stack) 撰寫除錯工具，該除錯工具可以確認某字串中的括號是否有成對

{[(3+6)]*8}+9

Practice

Mission

#143. Reorder List

You are given the head of a singly linked-list. The list can be represented as:

- L0 → L1 → … → Ln - 1 → Ln

Reorder the list to be on the following form:

- L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

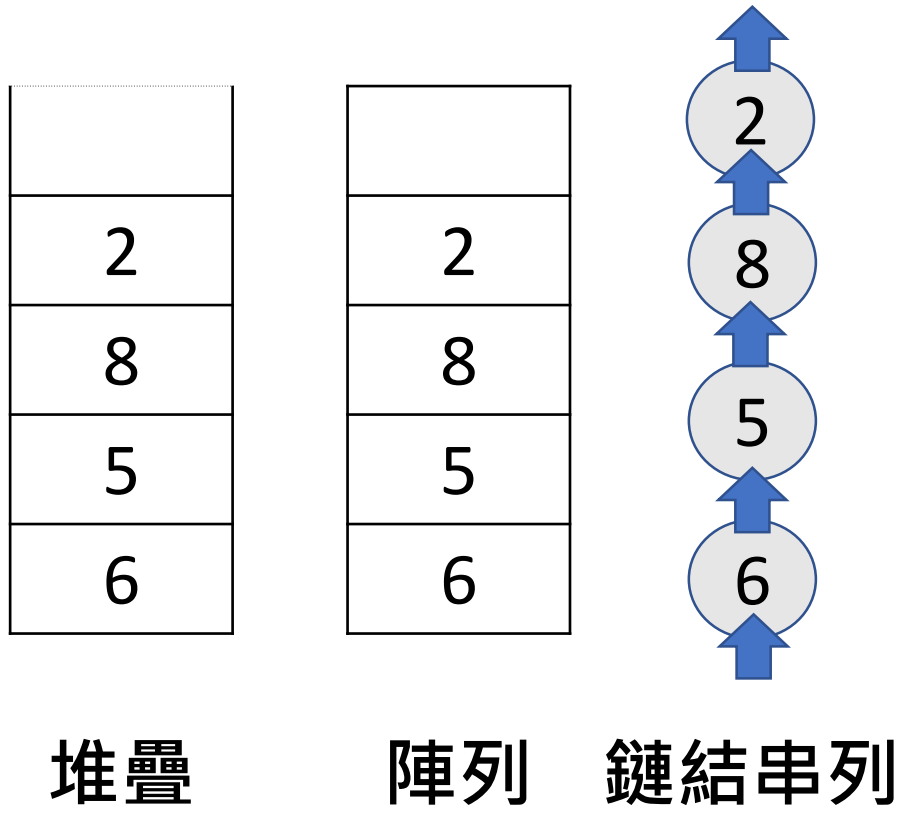
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Ref: https://leetcode.com/problems/reorder-list/

堆疊(Stack) 應用

堆疊應用

1. 河內塔
2. 深度優先搜尋：老鼠走迷宮
3. 中序轉後序運算

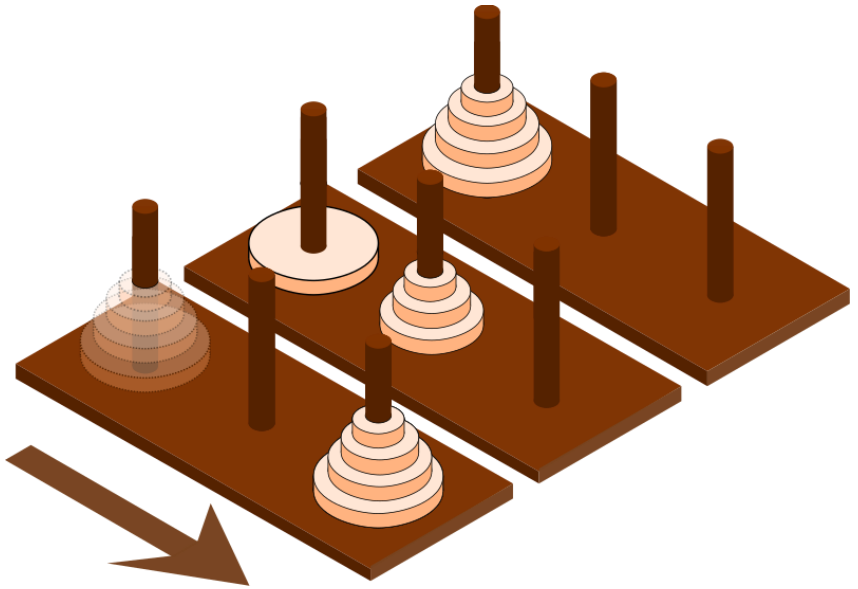


河內塔

河內塔

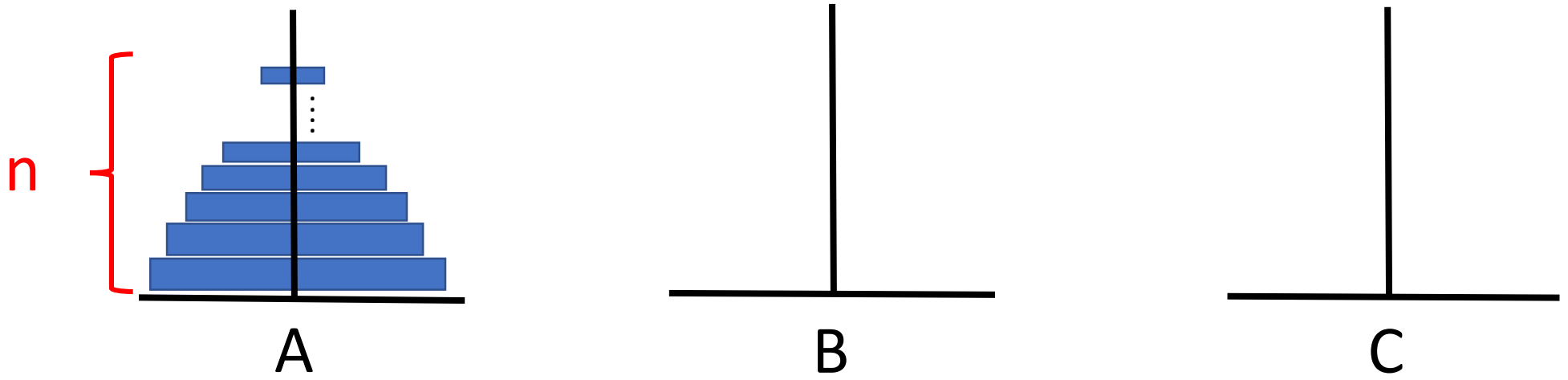
讓使用者輸入一正整數 N，請輸出要把 N 層河內塔從一根棍子移到另一根棍子的所有過程。過程必須上面盤子較下面盤子小的規則。

提示：最下面/最大的盤子必須先移動到目標



河內塔

最大的盤子要先從 A 移到 C

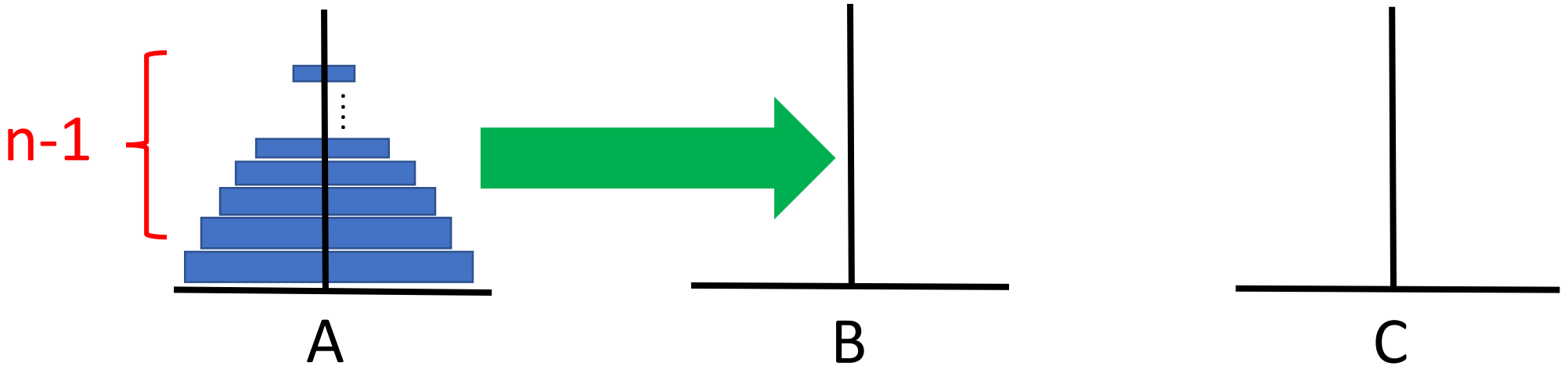


$$Hanoi(n) =$$

河內塔

最大的盤子要先從 A 移到 C

→須淨空最大盤子上的所有盤子

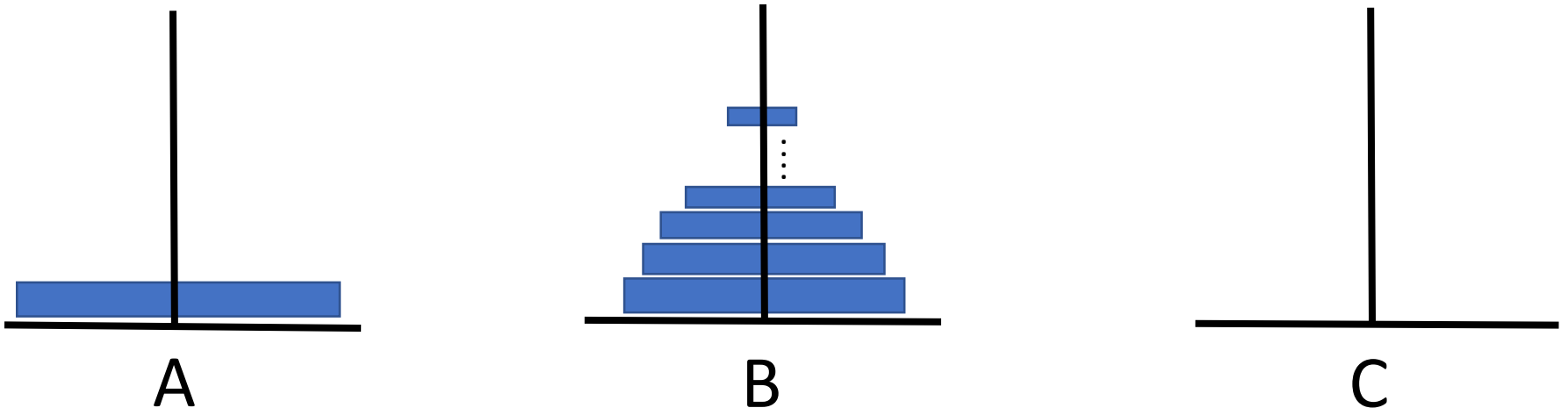


$$Hanoi(n) =$$

河內塔

最大的盤子要先從 A 移到 C

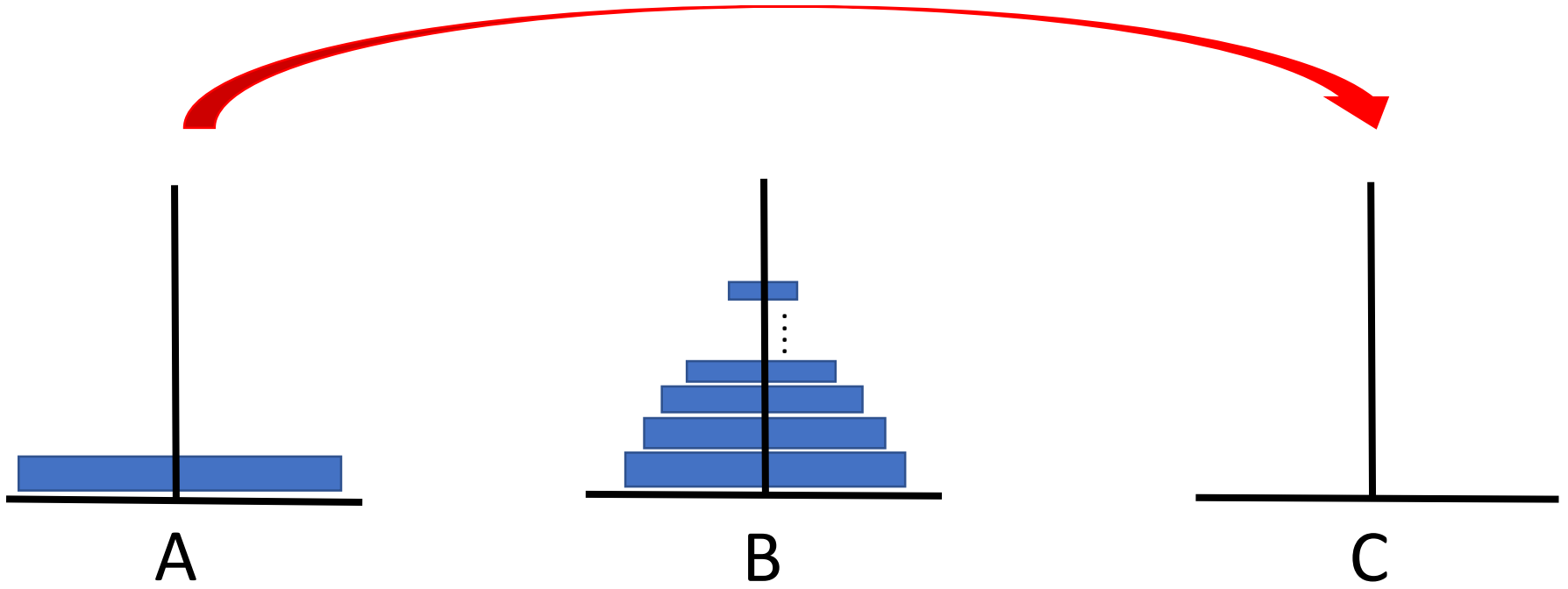
→須淨空最大盤子上的所有盤子



$$Hanoi(n) = Hanoi(n-1)$$

河內塔

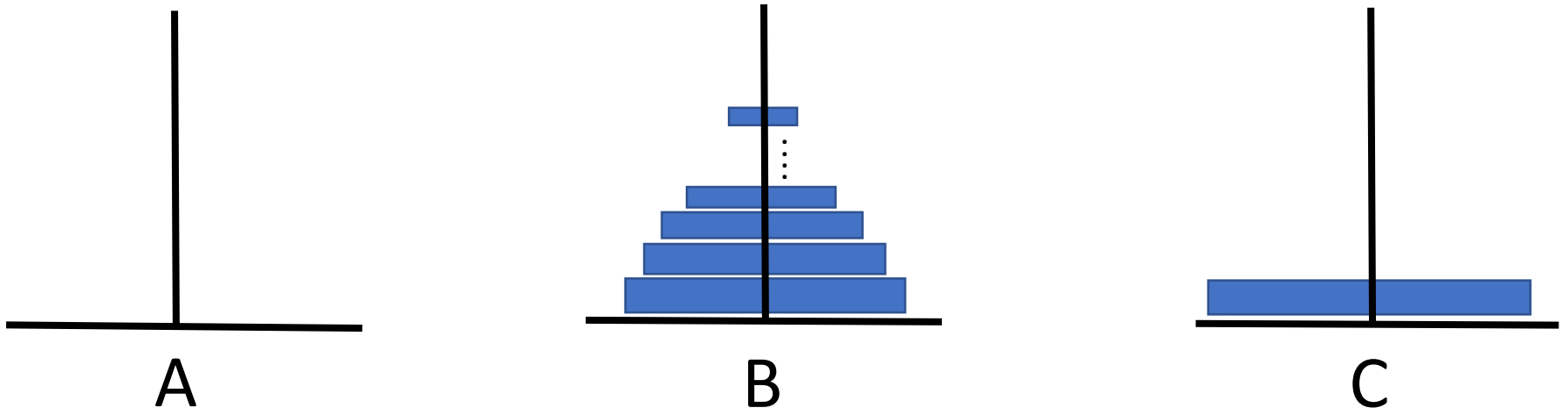
最大的盤子移動到目的地



$$Hanoi(n) = Hanoi(n-1)$$

河內塔

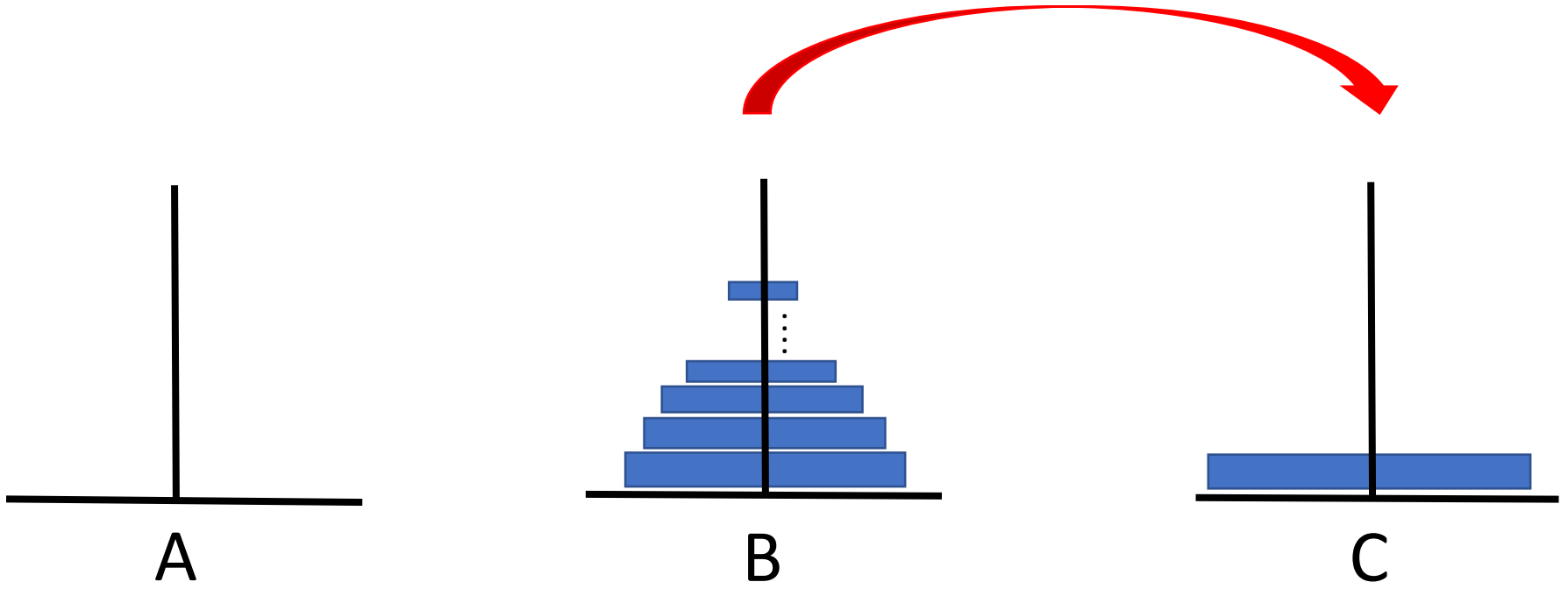
最大的盤子移動到目的地



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1)$$

河內塔

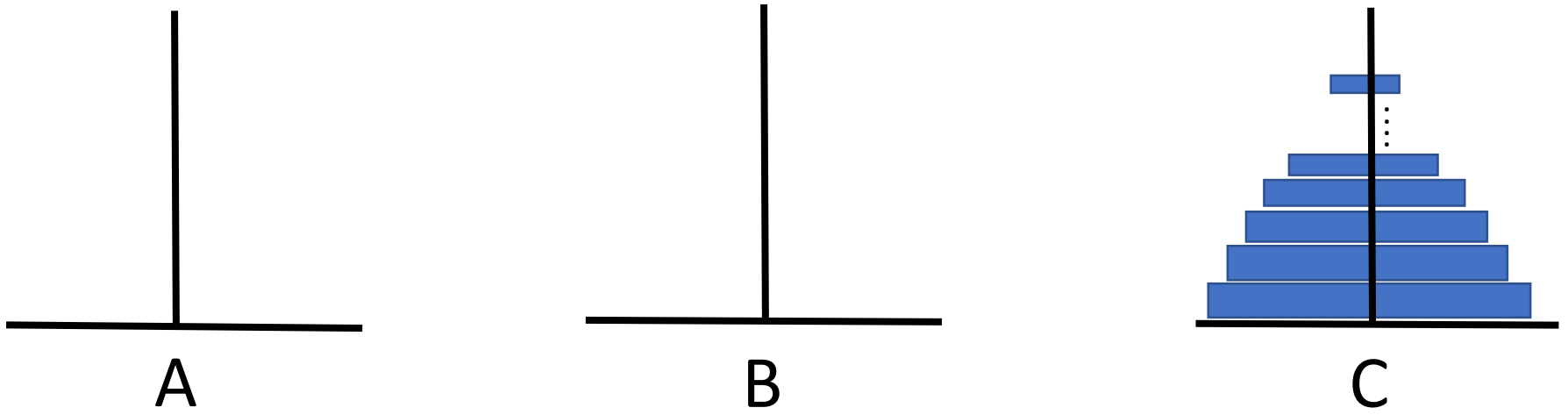
移動 n-1 個盤子到目的地



$$Hanoi(n) = Hanoi(n-1) + Hanoi(1)$$

河內塔

移動 n-1 個盤子到目的地

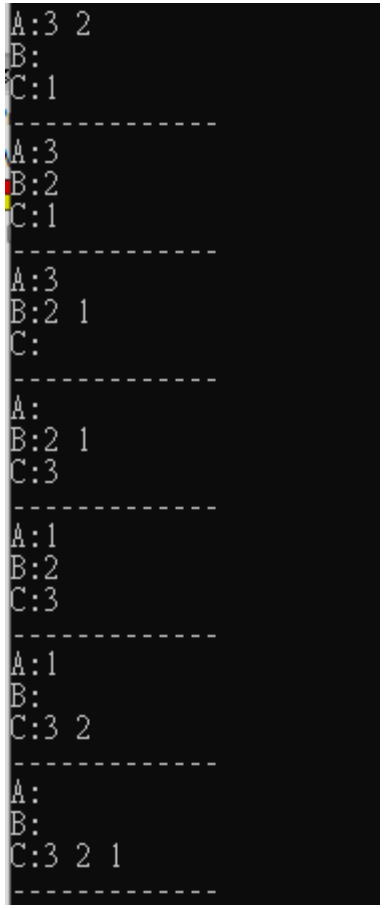


$$Hanoi(n) = Hanoi(n-1) + Hanoi(1) + Hanoi(n-1)$$

Example Code

Mission

輸入 N 層河內塔，印出搬動過程中的所有河內塔的圖形



堆疊應用

老鼠走迷宮

右邊的二維陣列中，1 代表牆壁，0 代表可以走的路徑，起點是 (1,1)，終點是 (8,10)，請找出一條路徑可以從起點到終點，並把可行的路徑用 2 表示。

```
int MAZE[10][12]={
    {1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,1,1,1,1,1,1,1,1},
    {1,1,1,0,1,1,0,0,0,0,1,1},
    {1,1,1,0,1,1,0,1,1,0,1,1},
    {1,1,1,0,1,0,0,1,1,0,1,1},
    {1,1,1,0,1,1,0,1,1,0,1,1},
    {1,1,1,0,0,0,0,1,1,0,1,1},
    {1,1,1,1,1,1,0,1,1,0,1,1},
    {1,1,0,0,0,0,0,0,1,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1}
};
```

堆疊應用

老鼠走迷宮

提示：利用 Stack 儲存目前走過的路徑，開另外一個二維陣列紀錄已經嘗試過的路徑，不斷往下搜索直至無路可走為止後回退到分岔口找別的路徑。

```
int MAZE[10][12]={
    {1,1,1,1,1,1,1,1,1,1,1,1},
    {1,2,2,2,1,1,1,1,1,1,1,1},
    {1,1,1,2,1,1,2,2,2,2,1,1},
    {1,1,1,2,1,1,2,1,1,2,1,1},
    {1,1,1,2,2,2,2,1,1,2,1,1},
    {1,1,1,0,1,1,0,1,1,2,1,1},
    {1,1,1,0,1,1,0,1,1,2,1,1},
    {1,1,1,1,1,1,0,1,1,2,1,1},
    {1,1,0,0,0,0,0,0,1,2,2,1},
    {1,1,1,1,1,1,1,1,1,1,1,1}
};
```

堆疊應用

中序運算轉後序運算

中序 (Infix) 表示運算子在中間，是我們慣用的運算順序：

$$a + b \times c + d$$

後序 (Postfix) 表示運算子在後面，則是電腦慣用的運算順序：

$$abc \times +d +$$

之所以電腦會用後序(postfix)表達是因為運算上的考量，如果用後續運算表達的話就可以無需考慮括號造成的優先次序，每當遇到兩個數字、一個運算子符號時就可以直接進行運算。

堆疊應用

後序 (Postfix) 運算時，自左到右把資料取出，遇到運算元 (數字) 時就放入堆疊，遇到運算子就自堆疊中拿出兩個運算元加以運算。

$$Example：352\times+6+$$

1. (3) 把 3 放入堆疊，堆疊 = {3}
2. (5) 把 5 放入堆疊，堆疊 = {3,5}
3. (2) 把 2 放入堆疊，堆疊 = {3,5,2}
4. (×) 自堆疊中取出兩筆資料 5,2，運算 5× 2 後放回堆疊，堆疊 = {3,10}
5. (+) 自堆疊中取出兩筆資料 10,3，運算 10+3 後放回堆疊，堆疊 = {13}
6. (6) 把 6 放入堆疊，堆疊 = {13,6}
7. (+) 自堆疊中取出兩筆資料 13,6，運算 13+6 後放回堆疊，堆疊 = {19}
8. 結束，運算結果為 19

堆疊應用

中序運算轉後序運算

給定一個中序運算的字串，請利用堆疊 (Stack)，把中序運算轉換成後序運算。其中運算元 (數字) 保證只有一位數，且運算子間都以小括號包覆。

提示：開一個堆疊紀錄運算子，當遇到數字便直接輸出，遇到運算子就存入堆疊，當遇到右括號就自取出取出一運算子後輸出。

堆疊應用

$$\boldsymbol{Example}: ((\boldsymbol{a} + (\boldsymbol{b} \times \boldsymbol{c})) + \boldsymbol{d})$$

1. (：遇到左括號，忽略
2. (：遇到左括號，忽略
3. a：輸出 a，輸出 = a
4. + ：把 + 置入堆疊，堆疊 = {+}
5. (：遇到左括號，忽略
6. b：輸出 b，輸出 = ab
7. × ：把 × 置入堆疊，堆疊 = {+, ×}
8. c：輸出 c，輸出 = abc
9.) ：自堆疊中取出 × 後輸出，輸出 = abc×
10.) ：自堆疊中取出 + 後輸出，輸出 = abc×+
11. + ：把 + 置入堆疊，堆疊 = {+}
12. d ：輸出 d，輸出 = abc×+d
13.) ：自堆疊中取出 + 後輸出，輸出 = abc×+d+

Practice

Mission

#224. Basic Calculator

Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return the result of the evaluation.

Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

Ref: *https://leetcode.com/problems/basic-calculator/*