回課程 切換章節

10802_程式設計（二）-【Data Structure】Stack、Queue

切換章節 回課程

以下 stack, queue 皆以陣列做講解

Stack

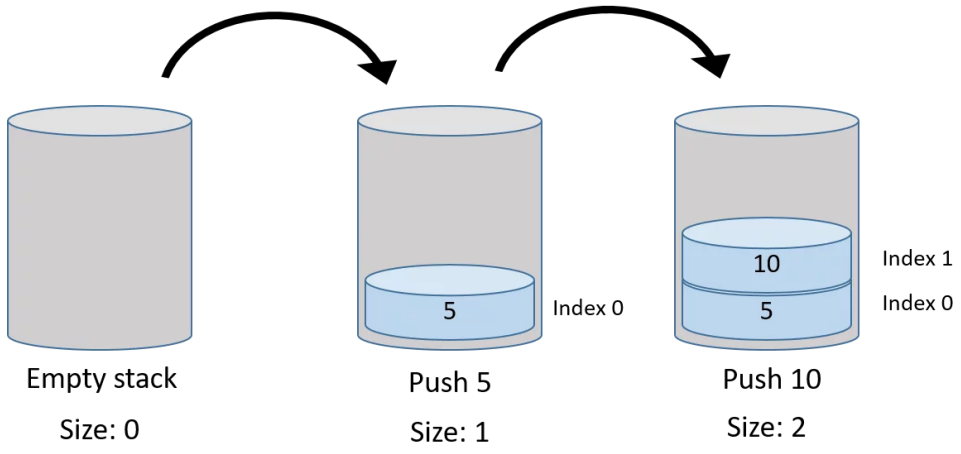
堆疊， Last In First Out (LIFO)。

> 把東西疊上去，東西從最上面拿。
>
> 會有一個 size 變數代表 stack 的內容到哪裡

- Empty 情況
  stack 的 size 等於 0

- Full 情況
  stack 的 size 等於陣列最大的 size

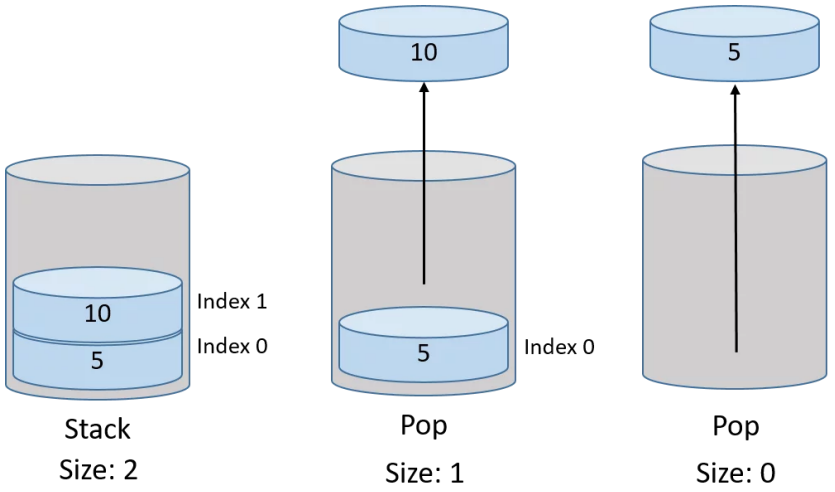
- Push Data




```
1  /*   @param: stack 為陣列，模擬 stack 行為
2   *           stack_size 為 stack 存放資料數量，因為會修改該值所以以 pointer 傳入
3   *           data 為欲放入的資料
4   *   @return: 沒有回傳值。可以自行設定回傳值
5   */
```

```c
void push_stack(int * stack, int * stack_size, const int data)
{
    if(stack == NULL || stack_size == NULL || *stack_size < 0) return;
    stack[*stack_size] = data
    (*stack_size) += 1;
}
```

- Pop Data




```c
/*   @param: stack 為陣列，模擬 stack 行為
 *           stack_size 為 stack 存放資料數量，因為會修改該值所以以 pointer 傳入
 *   @return: 回傳 pop 出來的值
 */
int pop_stack(int * stack, int * stack_size)
{
    if(stack == NULL || stack_size == NULL || *stack_size < 0) return -1; //return error num

    int data;
    data = stack[(*stack_size - 1)];
    (*stack_size) -= 1;

    return data;
}
```

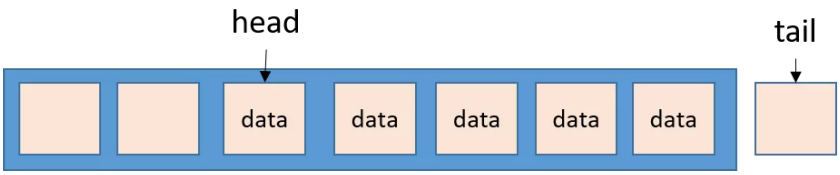
Queue

佇列，First In First Out。

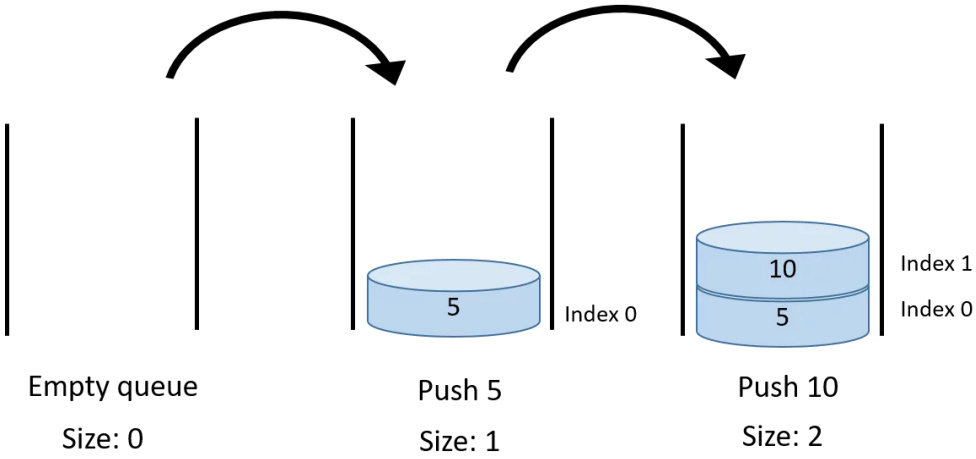
> 排排站，先來最前面。
> 會有一個 head index 代表 queue 的內容從哪開始
> 會有一個 tail index 代表 queue 的內容到哪裡結束

- Empty 情況
  queue 的 head 等於 tail

- Full 情況
  queue 的 tail 等於陣列最大的 size
  然而這情況不代表 queue 的內容數等於陣列大小 (當 head 不等於 0 時)
  會造成不必要的空間浪費
  所以通常是用 circular queue 或是以 linked list 實作



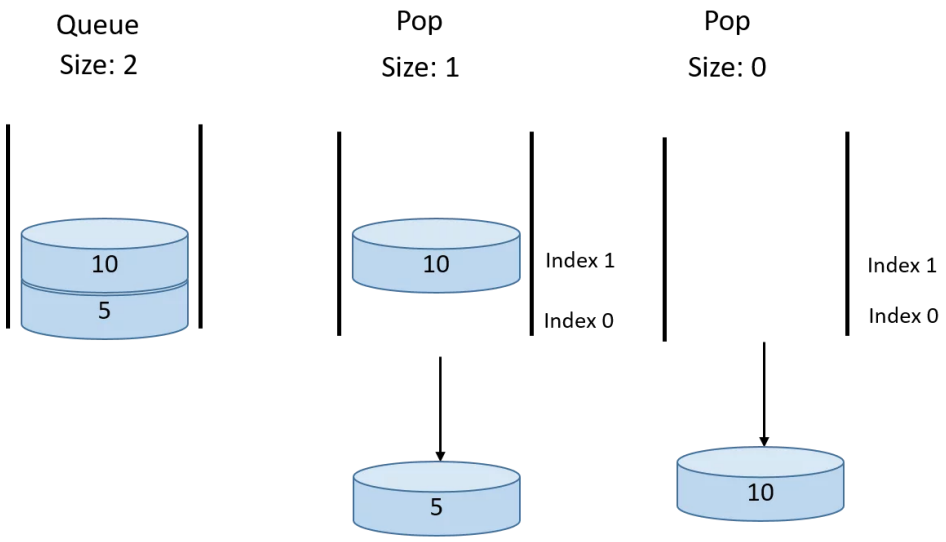
- push data




```
1  /*   @param: queue 為陣列，模擬 queue 行為
2   *           queue_tail 為 queue 存放資料位置，初始值會為 0，因為會修改該值所以以 pointer 傳入
3   *           data 為欲放入的資料
4   *   @return: 沒有回傳值。可以自行設定回傳值
5   */
6  void push_queue(int * queue, int * queue_tail, const int data)
7  {
8      if(queue == NULL || queue_tail == NULL || *queue_tail <
```

```
    0 || *queue_tail > MAX_SIZE) return;
 9      queue[*queue_tail] = data
10      (*queue_tail) += 1;
11  }
```

- pop data




```
1  /*    @param: queue 為陣列，模擬 queue 行為
2   *            queue_head 為 queue 存放起始位置，因為 pop 會修改該
    值所以以 pointer 傳入
3                queue_tail 為 queue 存放尾端
4   *    @return: 回傳 pop 出來的值
5   */
6  int pop_queue(int * queue, int * queue_head, const int
    queue_tail)
7  {
8      if(queue == NULL || queue_head == NULL ||
9         *queue_head < 0  || *queue_head >= queue_tail )
    return -1; //return error num
10
11     int data;
12     data = queue[(*queue_head)];
13     (*queue_head) += 1;
14
15     return data;
16 }
```

使用陣列實作 circular queue

其實操作概念與上方雷同，僅有在 index 到達陣列長度時需做調整。

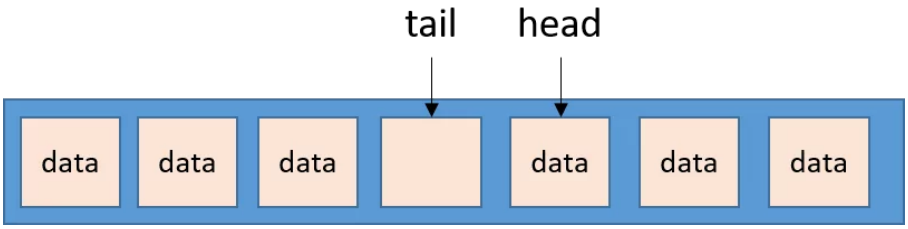
前面有說，若 tail 已經達到陣列尾端時，就會判斷為 full 。然而 head 前面還會有空間，所以這時則需要把 tail 取餘數繼續放即可。

- Empty 情況

  queue 的 head 等於 tail

- Full 情況

  queue 的 tail +1 等於 head



> 使用陣列實做 circular queue 要小心的是會有一個空值，來確保 tail + 1 == head 為滿的情況 tail == head 為空的情況

- push data

```
/*    @param: queue 為陣列，模擬 queue 行為
 *            queue_tail 為 queue 存放資料數位置，初始值會為 0，因為會修改該值所以以 pointer 傳入
 *            queue_head 為 queue 的起始位置。
 *            data 為欲放入的資料
 *    @return: 沒有回傳值。可以自行設定回傳值
 */
void push_circular_queue(int * queue, int * queue_tail, const int queue_head, const int data)
{
    if(queue == NULL || queue_tail == NULL || *queue_tail < 0 || (*queue_tail + 1) % MAX_SIZE == queue_head) return;
    queue[*queue_tail] = data
    (*queue_tail) = (*queue_tail + 1) % MAX_SIZE;
}
```

- pop data

```
/*    @param: queue 為陣列，模擬 queue 行為
 *            queue_head 為 queue 存放資料數量，因為會修改該值所以以 pointer 傳入
 *            queue_tail 為 queue 存放資料尾端
 *    @return: 回傳 pop 出來的值
 */
int pop_queue(int * queue, int * queue_head, const int queue_tail)
```

```
7  {
8      if(queue == NULL || queue_head == NULL ||
9         *queue_head < 0  || *queue_head == queue_tail )
   return -1;    // return error num
10
11     int data;
12     data = queue[(*queue_head)];
13     (*queue_head) = (*queue_head + 1) % MAX_SIZE;
14
15     return data;
16 }
```

第一次上機考優質題庫 ›

第一次上機考優質題庫

課後練習

#	題目	難度	通過率	解題人次	配分
1	Stack Practice	未分級	78%	27	-
2	Circular Queue Practice	未分級	59%	17	-