

# 程式設計與實習(二)

---

BY 孫茂勛

EMAIL:JOHN85051232@GMAIL.COM

# F5+F10+F11+中斷點

---

先隨便打個程式碼，最好包含一個function、一個loop

Ex：

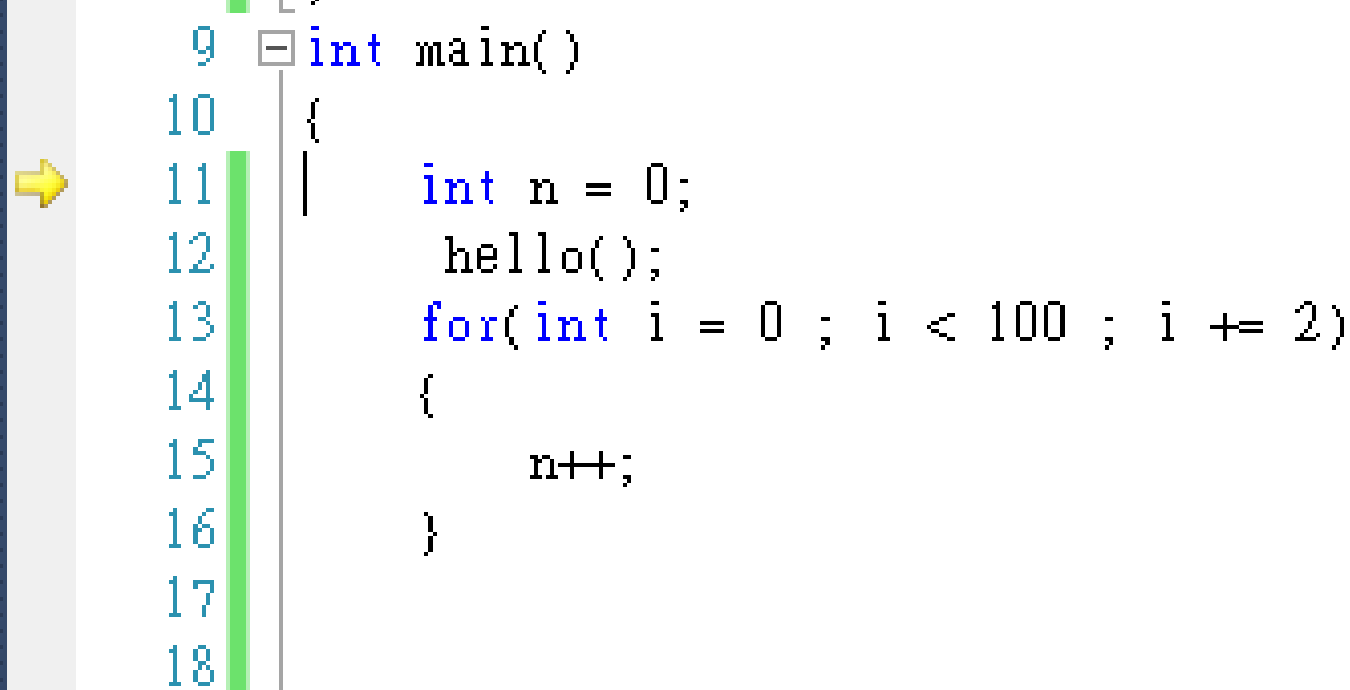
```
void hello()
{
    printf("hello");
    printf("hello1");
    printf("hello2");
}
int main()
{
    int n = 0;
    hello();
    for(int i = 0 ; i < 100 ; i += 2)
    {
        n++;
    }
}
```

# F5+F10+F11+中斷點

然後試試看分別按F10 跟F11

此時會有箭頭代表執行到哪一行

(loop跑太久不想跑可以隨時按F5一次跑完)



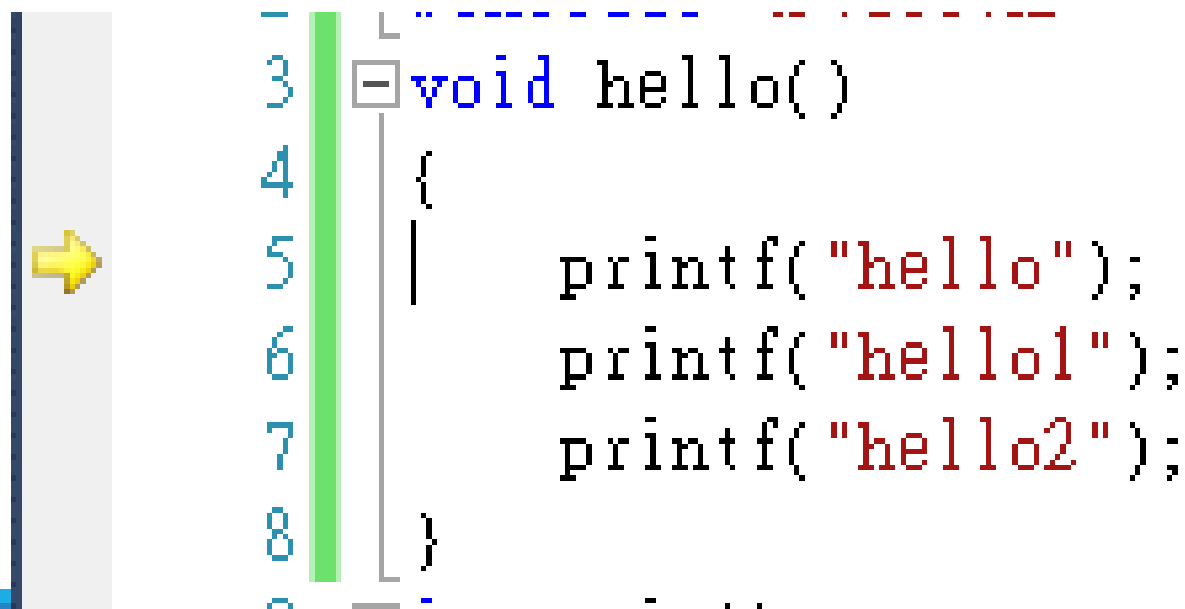
```
9 int main()  
10 {  
11 | int n = 0;  
12 | hello();  
13 | for(int i = 0 ; i < 100 ; i += 2)  
14 | {  
15 |     n++;  
16 | }  
17  
18
```

# F5+F10+F11+中斷點

---

F10：會跳過function

F11：會進入function

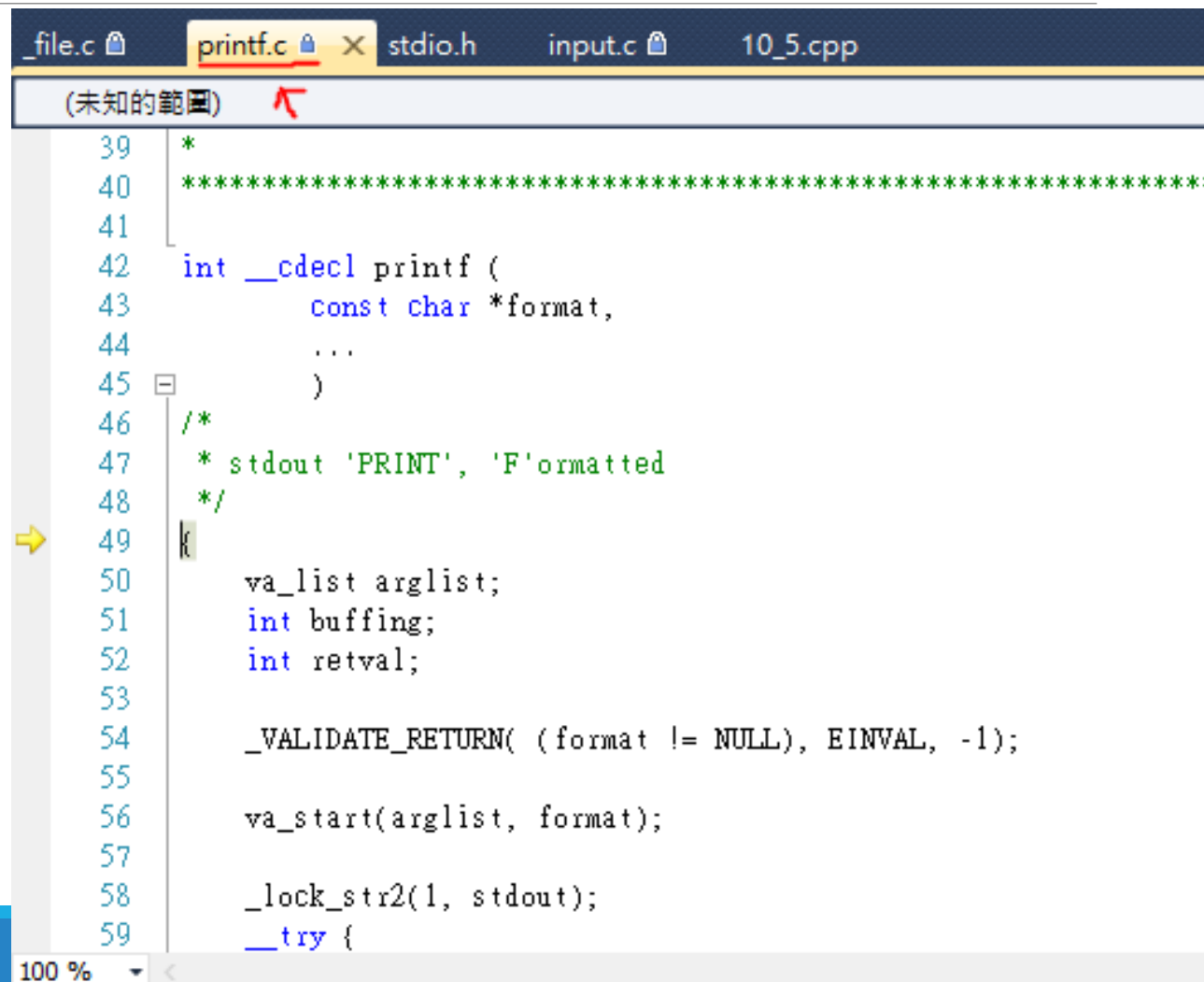


The image shows a code editor window with a function definition. A yellow arrow points to the start of the function body on line 5. The code is as follows:

```
3 void hello()  
4 {  
5     printf("hello");  
6     printf("hello1");  
7     printf("hello2");  
8 }
```

# F5+F10+F11+中斷點

F11進入printf內部了???  
多按幾次F10就會回來了



```
39  *
40  ****
41
42  int __cdecl printf (
43      const char *format,
44      ...
45  )
46  /*
47   * stdout 'PRINT', 'F'ormatted
48   */
49  {
50      va_list arglist;
51      int buffering;
52      int retval;
53
54      _VALIDATE_RETURN( (format != NULL), EINVAL, -1);
55
56      va_start(arglist, format);
57
58      _lock_str2(1, stdout);
59      __try {
```

# F5+F10+F11+ 中斷點

---

常用搭配：F11進入function後就改用F10  
F5直接執行完剩下的部分

# F5+F10+F11+中斷點

利用逐步偵錯來查看當下狀態的變數狀態

10\_5 [偵錯] - Microsoft Visual Studio

檔案(F) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D) 小組(M) 資料(A) 工具(T) 測試(S) 視窗(W) 說明(H)

Debug Win32 &lt;

處理序: [7940] 10\_5.exe 執行緒: [9424] 主執行緒 堆疊框架: 10\_5.exe!main() 行 15

方案總管

- 方案 '10\_5' (1 專案)
  - 10\_5
    - 原始程式檔
      - 10\_5.cpp
    - 外部相依性
    - 標頭檔
    - 資源檔

(全域範圍) main()

```
4 {  
5     printf("hello");  
6     printf("hello1");  
7     printf("hello2");  
8 }  
9 int main()  
10 {  
11     int n = 0;  
12     hello();  
13     for(int i = 0 ; i < 100 ; i += 2)  
14     {  
15         n++;  
16     }  
17  
18  
19     system("PAUSE");  
20     return 0;  
21 }  
22
```

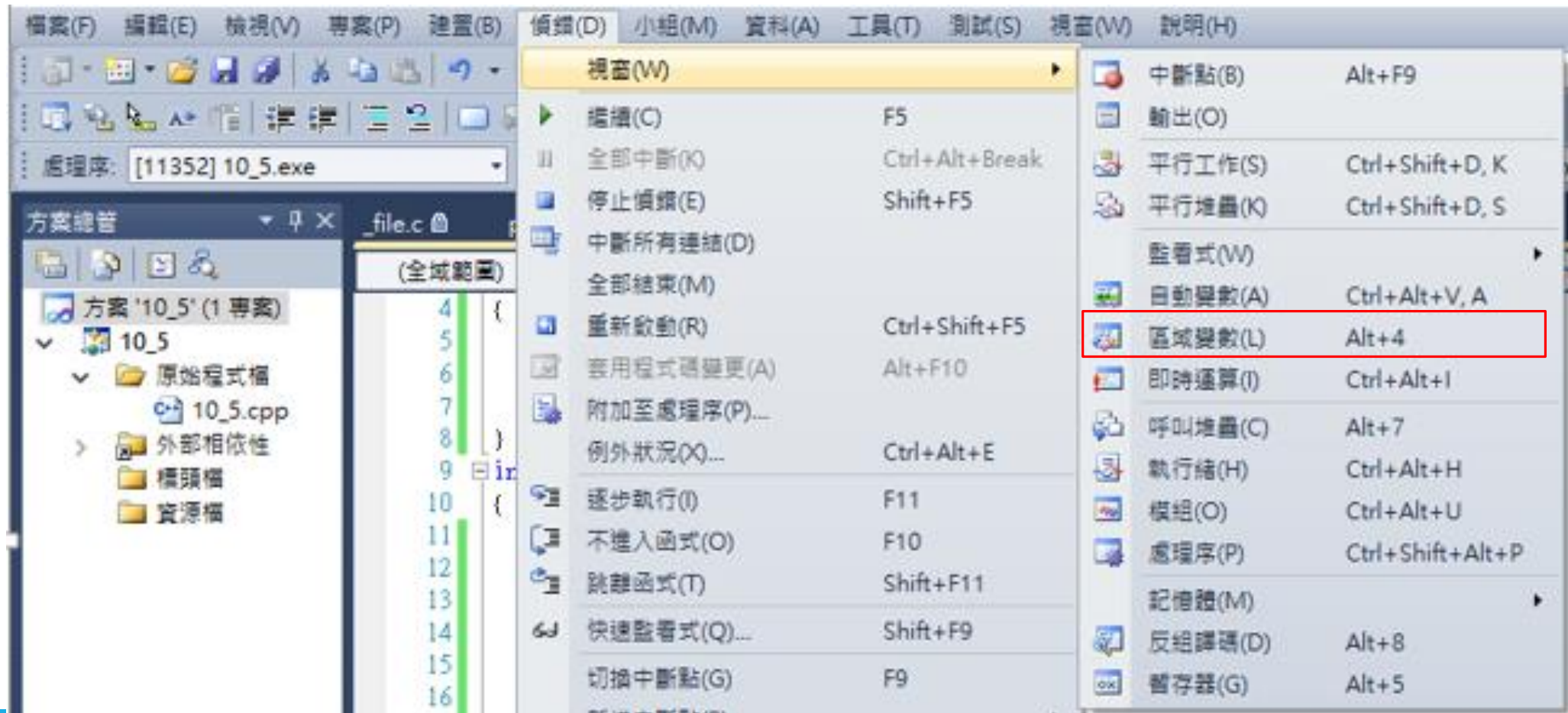
區域變數

名稱	值	型別
i	12	int
n	6	int

就緒

第 15 行 第 1 欄 字元 1 INS

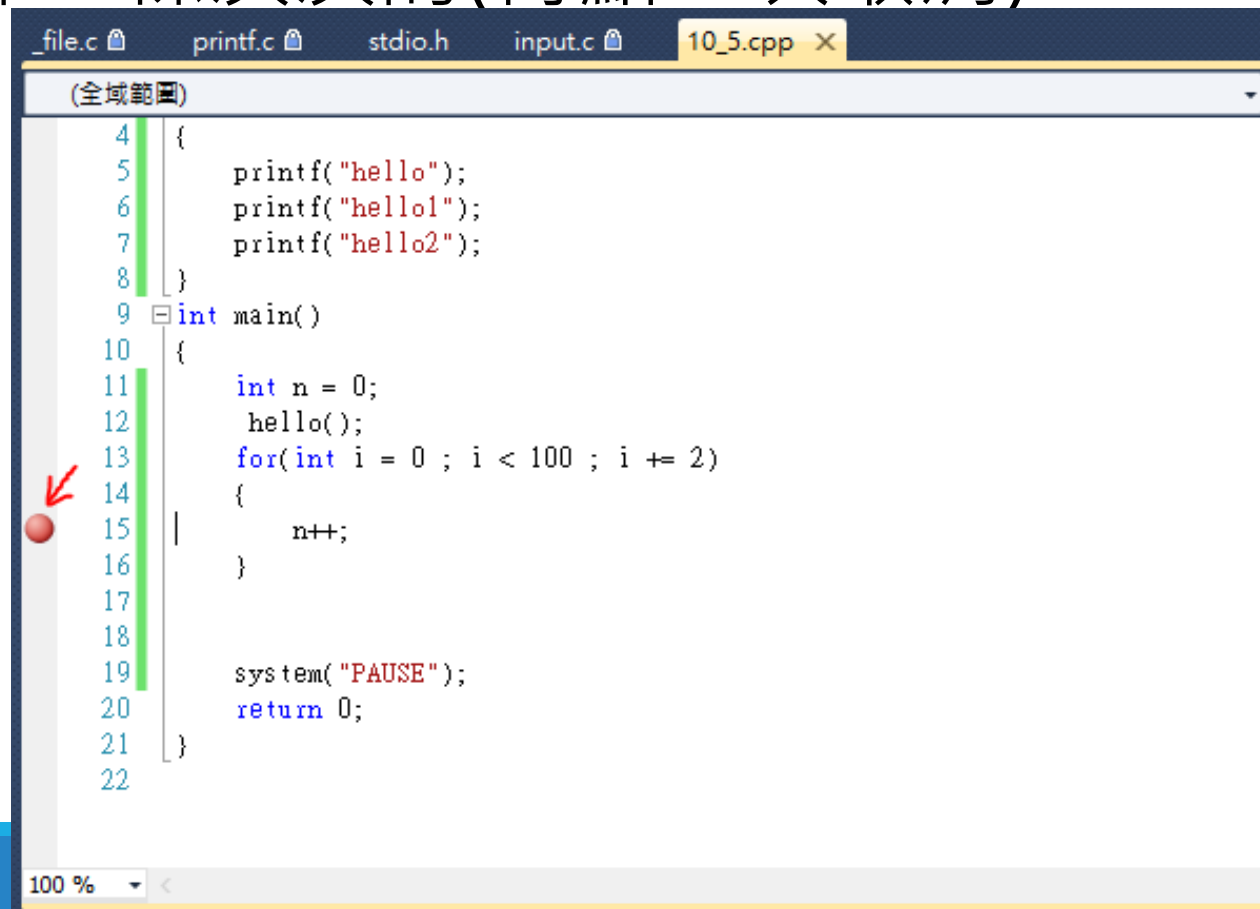
# F5+F10+F11+中斷點





# F5+F10+F11+中斷點

中斷點：點一下左邊那一條灰灰的(再點一次取消)  
在按一次F5



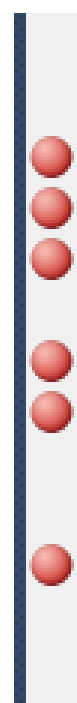
The screenshot shows a code editor window with the file `10_5.cpp` open. The code is as follows:

```
4 {  
5     printf("hello");  
6     printf("hello1");  
7     printf("hello2");  
8 }  
9 int main()  
10 {  
11     int n = 0;  
12     hello();  
13     for(int i = 0 ; i < 100 ; i += 2)  
14     {  
15         n++;  
16     }  
17  
18  
19     system("PAUSE");  
20     return 0;  
21 }  
22
```

A red circular breakpoint is placed on the left margin next to line 14. A red arrow points to this breakpoint. The IDE interface includes a tab bar at the top with files `_file.c`, `printf.c`, `stdio.h`, `input.c`, and `10_5.cpp`. A status bar at the bottom shows the zoom level as 100%.

# F5+F10+F11+中斷點

中斷點：執行到該行程式變會暫停，之後可搭配F10/F11做Debug  
你也可以有很多個中斷點...



```
9  int main()  
10 {  
11     int n = 0;  
12     hello();  
13     for(int i = 0 ; i < 100 ; i += 2)  
14     {  
15         n++;  
16     }  
17  
18  
19     system("PAUSE");  
20     return 0;  
21 }
```



# 用Linked list做資料新增、刪除

C:\Users\user\Desktop\10\_5\Debug\10\_5.exe

選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：2  
輸入要新增的資料：10  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：2  
輸入要新增的資料：20  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：2  
輸入要新增的資料：30  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：1  
目前linked list 的資料有：10->20->30

選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：3  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：1  
目前linked list 的資料有：10->20

選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：3  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：1  
目前linked list 的資料有：10

選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：3  
選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：1  
目前linked list 的資料有：

選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：

微軟注音 半：

# struct建立、main宣告

---

```
struct node
{
    int num;
    node *next;
};
```

# struct建立、main宣告

```
int main()
{
    node *head = new node;
    head->num = -1;
    head->next = NULL;
    int choose = 0;
    int num = 0;
    ...
    system("PAUSE");
    return 0;
}
```

head紀錄串列的開頭，  
習慣上不會拿來進行操  
作，而是當作一個代表  
節點。



# switch

等下的操作都用function來實現

```
int choose = 0;
int num = 0;
while(1)
{
    printf("選擇要進行的操作：1.印出 2.插入尾端節點 3.刪除尾端節點 4.離開：");
    scanf("%d",&choose);
    switch(choose)
    {
        case 1:
            print(head);
            break;
        case 2:
            printf("輸入要新增的資料：");
            scanf("%d",&num);
            insert(head,num);
            break;
        case 3:
            del(head);
            break;
        case 4:
            exit(0);
            break;
    }
}
system("PAUSE");
```

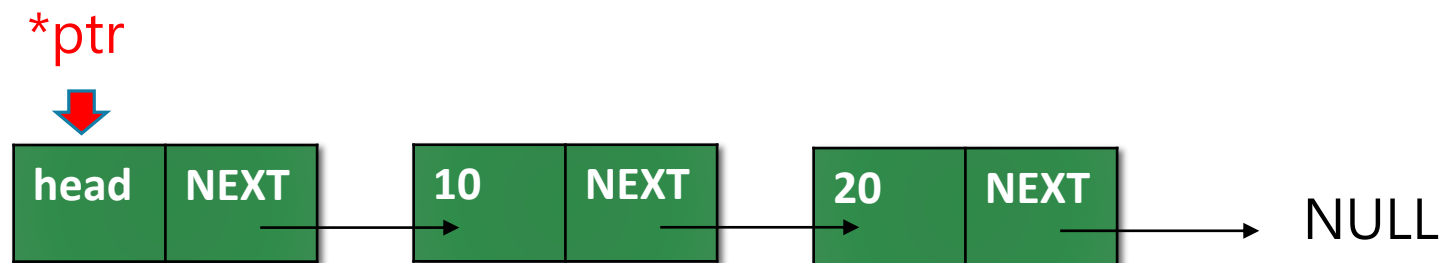
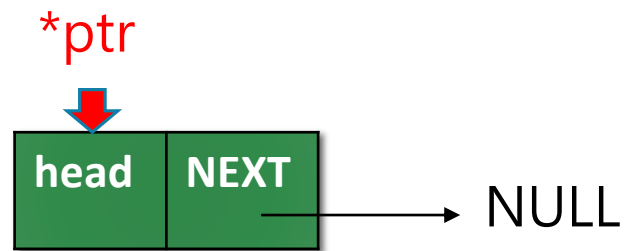
# print()

```
void print(node *head)//傳入開頭節點的位置
{
    node *ptr = head;//不能更改開頭節點，所以另外新增一個指標來操作
    printf("目前linked list 的資料有:");
    while(ptr->next != NULL)//如果有資料
    {
        ptr = ptr->next;
        if(ptr->next == NULL) printf("%d",ptr->num);
        else
        {
            printf("%d->",ptr->num);
        }
    }
    printf("\n\n");
}
```



# print()

---

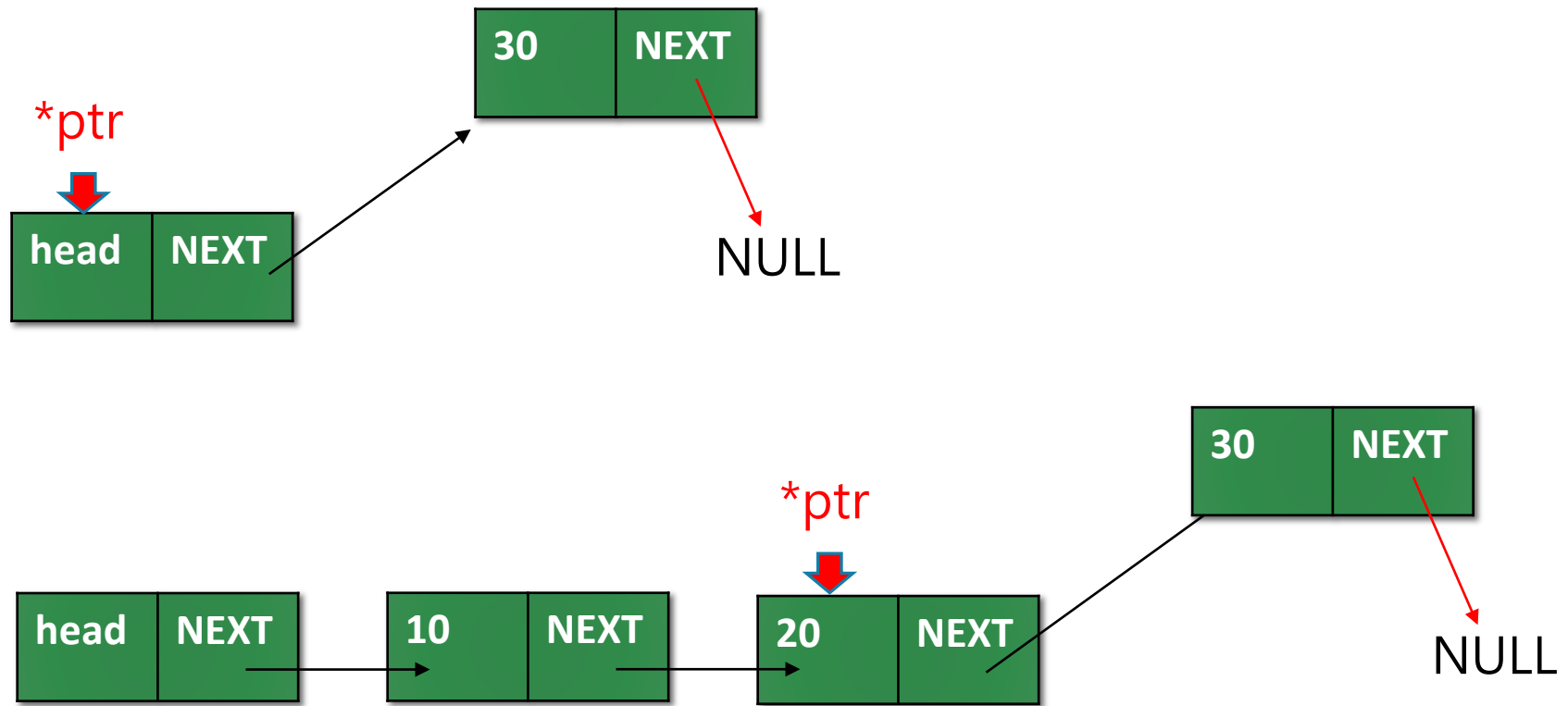


# insert()

```
void insert(node *head, int data) // 插入 linked list 尾端
{
    node *newPtr = new node;
    newPtr->num = data;
    newPtr->next = NULL;

    node *ptr = head;
    while(ptr->next != NULL) // 走到 NULL 前一個節點
    {
        ptr = ptr->next;
    }
    ptr->next = newPtr;
}
```

# insert()

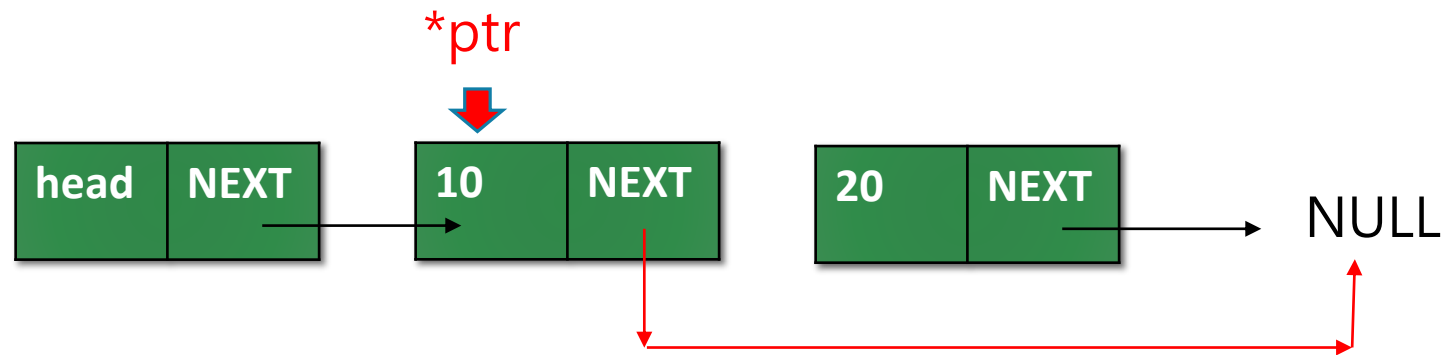
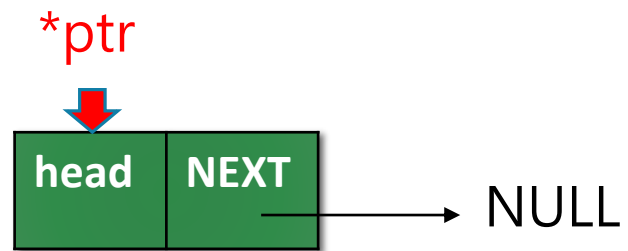


# del()

```
void del(node *head)
{
    node *ptr = head;
    node *deletePtr = NULL;
    if(ptr->next == NULL) printf("沒有資料可刪除!!\n");
    else
    {
        while(ptr->next->next != NULL)//走到NULL前兩個節點
        {
            ptr = ptr->next;
        }
        deletePtr = ptr->next;
        ptr->next = ptr->next->next;
        delete deletePtr;
    }
}
```

# del()

---



# 如果用陣列來寫？

---

必須假設資料量很大，宣告一個足夠容納所有變數的陣列（還是有可能不夠用）。

Linked list的好處在於可以隨著資料的增加/減少作變動，比起陣列可能需要一次宣告足夠大的空間來說更加方便。



# Stack

---

堆疊(Stack)：

- 資料結構的一種
- LIFO(Last In,First Out)
- Ex：疊盤子的時候，一定是從最上層(最後放的)開始拿
- 程式的呼叫順序就是一種Stack
- 遞迴的原理也是Stack



# Stack

---

基本操作：

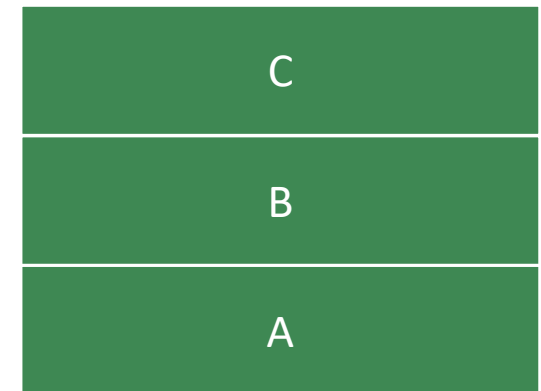
- push
- pop
- top
- size
- empty

# Stack

---

基本操作：

- push
- push A -> push B -> push C

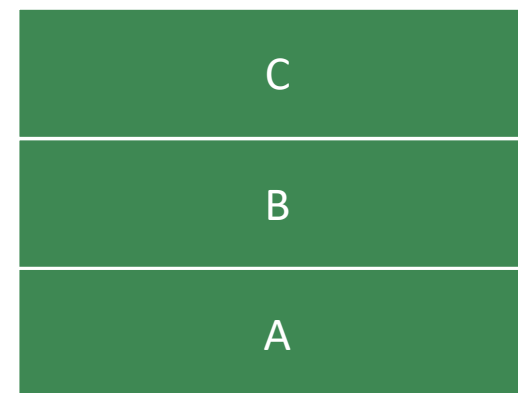


# Stack

---

基本操作：

- pop
- pop -> pop -> pop

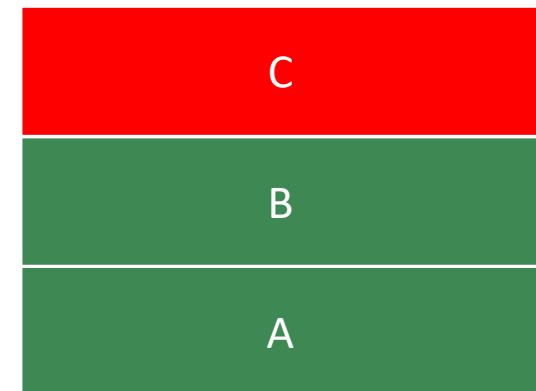


# Stack

---

基本操作：

- top

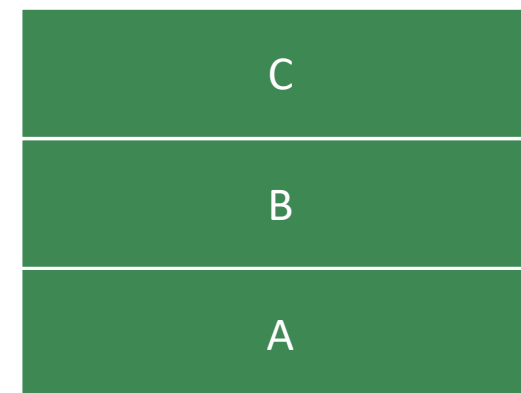


# Stack

---

基本操作：

- size = 3



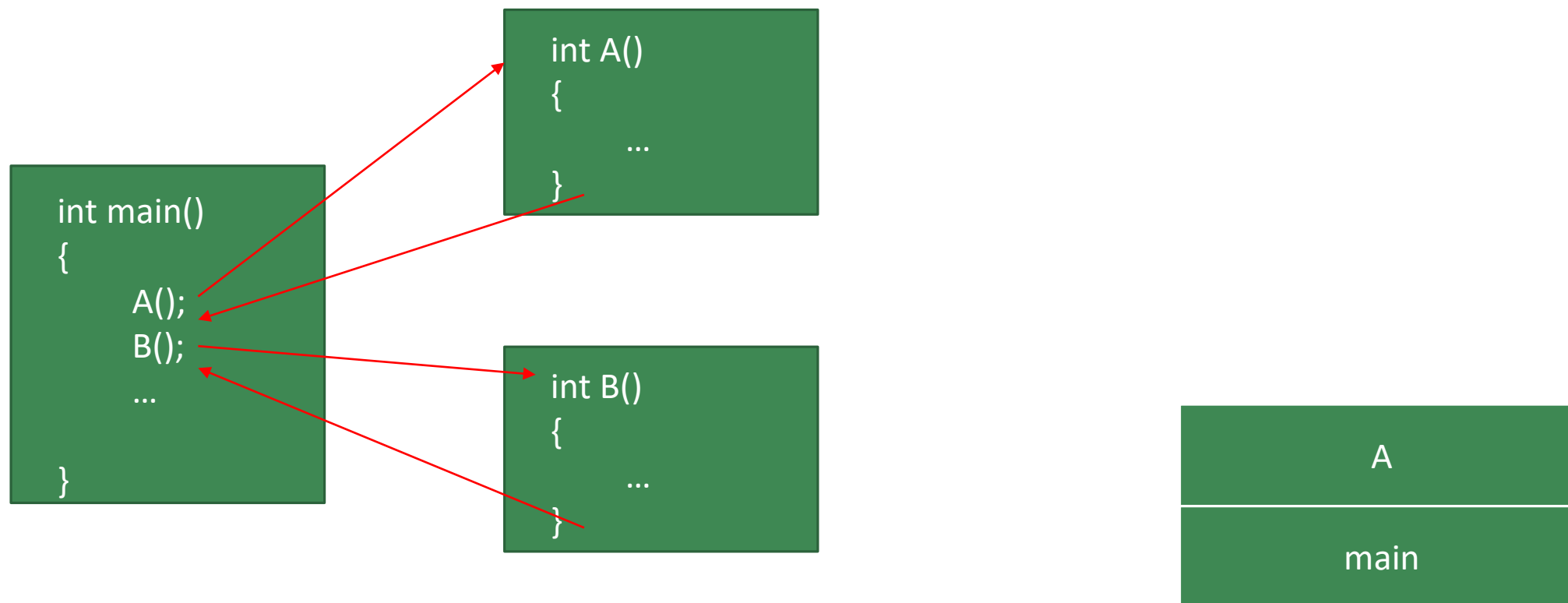
# Stack

---

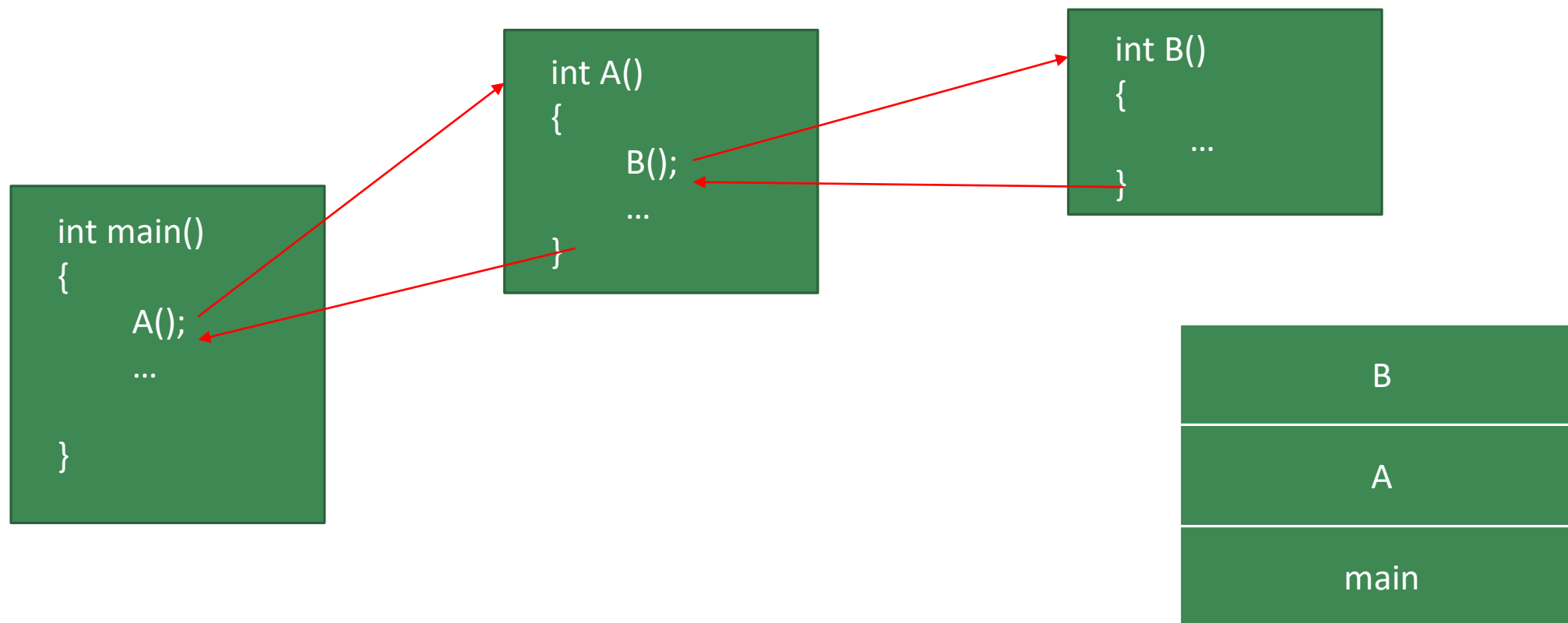
基本操作：

- empty

# 程式的執行順序



# 程式的執行順序





# Queue

---

佇列(Queue)：

- 資料結構的一種
- **FIFO(First In,First Out)**
- Ex：排隊的時候，最先排的最先離開
- CPU排程、廣度優先演算法(BFS)

# Queue

---

基本操作：

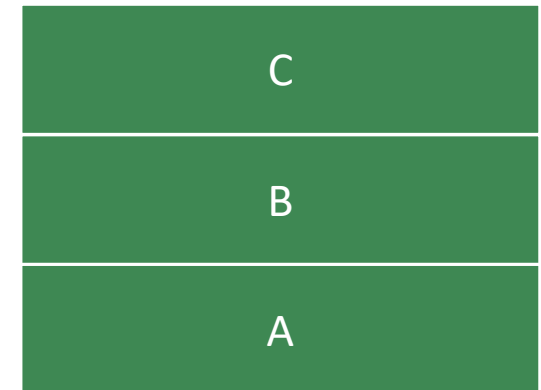
- push
- pop
- front
- size
- empty

# Queue

---

基本操作：

- push
- push A -> push B -> push C

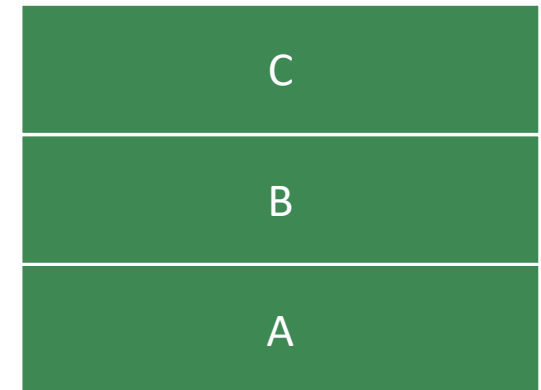


# Queue

---

基本操作：

- pop
- pop -> pop -> pop



# Queue

---

基本操作：

- front

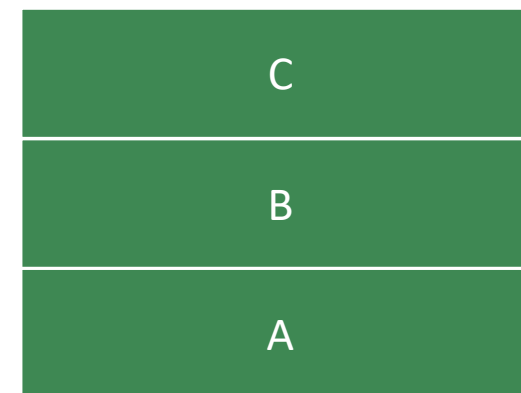


# Queue

---

基本操作：

- `size = 3`



# Queue

---

基本操作：

- empty

# 之後會用Linked List實作這些資料結構

---

可以回去先練習看看  
( 用陣列也可以做得出來 )