Lab of Object-Oriented Programming:
Array & Pointer

黃威、陳岳紘、邱彥翔

2022

遠距網址

請登入實習課的 moodle 課程

以後遠距連線固定使用此網址

如遇不可抗力需更改時會另行通知

使用 moodle 點名

請登入實習課的 moodle 課程

點擊出缺席並完成今日的點名

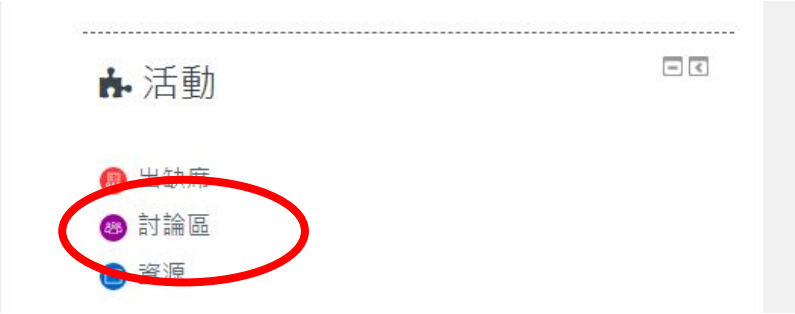
討論區發問

鼓勵大家使用實習課的討論區發問問題

如需要和助教約時間或處理其它事情再寄信

討論區發問

1.



2.

可以關閉訂閱避免同學問問題或助教回覆時一直被寄信打擾

Agenda

- Array
- Pointer
- Pass Argument
- Exercise 3
- Reminder

Array

To initialize arrays:

- type name [size];
  - type: such as int, float, ...
  - size: specify the length of the array.

Initialize Array

- int foo[5] = {7, 8, 9, 10, 11};
  int foo[] = {7, 8, 9, 10, 11};

	foo[0]	foo[1]	foo[2]	foo[3]	foo[4]
Value	7	8	9	10	11

- int foo[5] = {1}; // {1, 0, 0, 0, 0}
- int foo[5] = {1,5}; // {1, 5, 0, 0, 0}

- **int陣列的初始值為0;**
- **float陣列的初始值為 0.0;**
- **string陣列的初始值為'\0';**
- **bool陣列的初始值為false**

- **int foo[2] = {1, 2, 3}; // error**
- **int foo[]; // error**

Array Size

- Array size: sizeof(array) / sizeof(array[0])
  sizeof(array) / sizeof(*array)

```cpp
#include <iostream>
using namespace std;

int main() {
    // Initial Array
    int foo[] = {5, 1, 3};
    // Print Size of an Array
    cout << sizeof(foo) / sizeof(foo[0]) << endl; // 3
    cout << sizeof(foo) / sizeof(* foo) << endl; // 3
    cout << sizeof(foo) << endl; // 12 (4 byte for every element)
}
```

2D Array Size

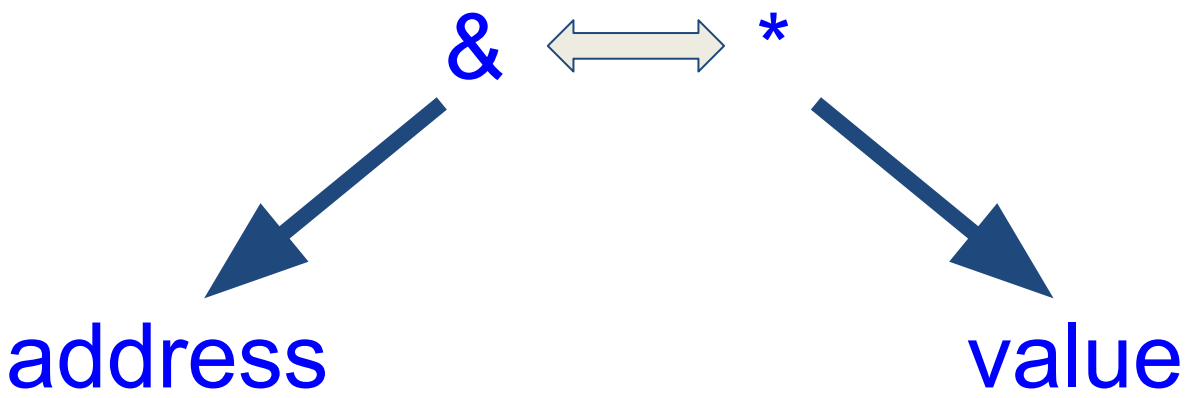
```cpp
#include <iostream>
using namespace std;

int main() {
    int foo[2][3] =
    {
            {1,5},
            {2,3,6}
    };

    for (int i = 0; i < sizeof(foo) / sizeof(foo[0]); ++i){
        for (int j = 0; j < sizeof(* foo) / sizeof(** foo); ++j){
            cout << "(" << i << ", " << j << "): " << foo[i][j] << "    ";
        }
        cout << endl;
    }

}
```

Outer:
foo: 4 * 2 * 3 = 24(bytes)
foo[0] == *foo: 4 * 3 = 12(bytes)
Outer size: 24 / 12 = 2

Inner:
foo[0] == *foo: 4 * 3 = 12(bytes)
foo[0][0] == **foo = 4(bytes)
Inner size: 12 / 4 = 3

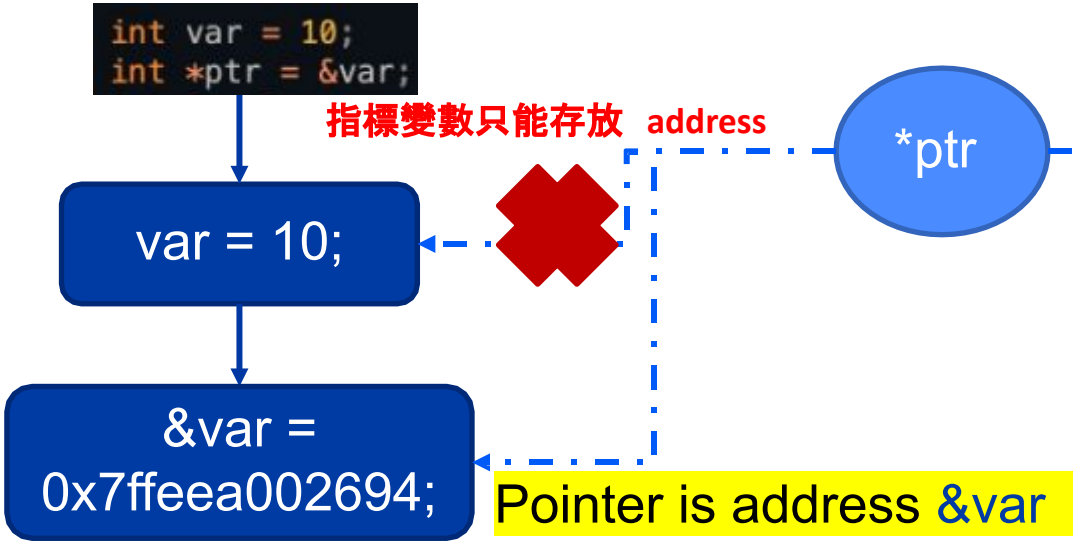
```
(0, 0): 1       (0, 1): 5       (0, 2): 0
(1, 0): 2       (1, 1): 3       (1, 2): 6
```

Pointer



Pointer

- &:取址運算子:取得變數的位址。
- *:反參考運算子:取得某個記憶體的內容。



cout << *ptr;
output : 10

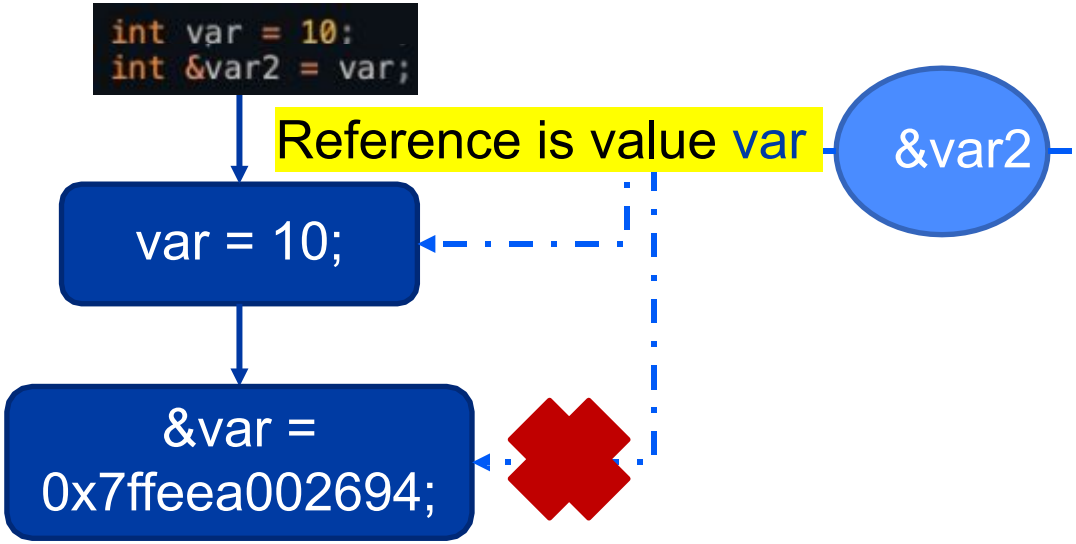
cout << ptr;
output : 0x7ffee897f694

cout << &ptr;
output : 0x7ffee897f688
(different from &var)

指標變數也是一個變數，有自己的記憶體位址，所以可以取址

Reference

- &:取址運算子:取得變數的位址。
- *:反參考運算子:取得某個記憶體的內容。



參考變數只能存放 value

```
cout << var2;
output : 10

cout << &var2 ;
output : 0x7ffee897f694

cout << &&var2 ;
output: error
```

reference 是現有變數的 alias，沒有實際分配記憶體位置

int &var2; // error

Pointer & Reference

	Pointer	Reference
Functionality	儲存記憶體位置	現有變數的別名
Operator	*	&
Null	允許為空 int *ptr;	不允許為空 int &ref; // error
Initialization	可以在任何時候初始化	只能在創建時被初始化

Modification	可以在任何時候指向其它物體
	int *ptr = &obj;
	ptr = &obj2;
		初始化後即**不能變更**參考的物體
		int &ref = obj;
		~~ref = obj2; // assign obj2's value to ref~~

Pointer Pratice

```cpp
#include <iostream>
using namespace std;

int main(){

    int var = 10;
    int *ptr = &var;
    cout << "var (var的值): " << var << endl;    // var的值
    cout << "&var (var的記憶體位置): " << &var << endl;   // var的記憶體位置
    cout << "ptr (ptr的值): " << ptr  << endl;    // ptr的值
    cout << "&ptr (ptr的記憶體位置): " << &ptr  << endl;  // ptr的記憶體位置
    cout << "*ptr (ptr存放的記憶體位置所指向的值): " << *ptr  << endl;  // ptr存放的記憶體位置所指向的值

}
```

```
var (var的值): 10
&var (var的記憶體位置): 0x7ffee7143694
ptr (ptr的值): 0x7ffee7143694
&ptr (ptr的記憶體位置): 0x7ffee7143688
*ptr (ptr存放的記憶體位置所指向的值): 10
```

Pointer Pratice

```cpp
#include <iostream>
using namespace std;

int main(){

    int var = 10;
    int *ptr = &var;
    int &var2 = var; // var2的記憶體位置 == var的記憶體位置

    cout << "Before" << endl;
    cout << "var: " << var << endl;      // var的值
    cout << "*ptr: " << *ptr << endl;    // *ptr的值
    cout << "var2: " << var2 << endl;    // var2的值
    cout << endl;
    // 改變var的值後，再看一次結果
    var = 20;
    cout << "After" << endl;
    cout << "var: " << var << endl;      // var的值
    cout << "*ptr: " << *ptr << endl;    // *ptr的值
    cout << "var2: " << var2 << endl;    // var2的值

}
```

等價

```
Before
var: 10
*ptr: 10
var2: 10

After
var: 20
*ptr: 20
var2: 20
```

var ==*ptr == var2
&var == ptr == &var2

Pointer Pratice

Notice:

```
int* p1, p2;     // int* p1; int p2;

                 // p1 is a pointer, but p2 is an integer.
```

```
int* p1, *p2;    // int* p1; int* p2;

                 // p1 is a pointer, and p2 is a pointer.
```

Const Pointer

- **const int* p1;** // pointer to const int

  // 只能更改p1所存放的位址(p1)

- **int const *p1;** // 同上
- **int* const p1;** // const pointer to int

  // 只能更改p1此位址所指向的內容(*p1)

- **const int* const p1;** // const pointer to const int

  // 兩者都不能改

const 在 * 前面 => 值固定
const 在 * 後面 => 位址固定

Pass Arguments

- Call by Value. (C, C++)
- Call by Address. (C, C++)
- Call by Reference. (C++)

Pass Arguments – Call by Value

- The values are copied from the variables used in the caller main() to the variables used by the function.
- After the function call, the variables will **not be changed**.
- Ex: **void swap (int, int)**;

  call the function by **swap (num1, num2)**;

```
40  #include <iostream>
41  using namespace std;
42
43  // Using XOR to swap two value
44  void swap( int num1, int num2){
45
46      num1 ^= num2; // 00000101 XOR 00001000
47      num2 ^= num1; // 00001101 XOR 00001000
48      num1 ^= num2; // 00001101 XOR 00000101
49      cout << "In function swap, (num1, num2): " << num1 << ", " << num2 << endl;
50  }
51  int main(){
52      int num1 = 5, num2 = 8;
53      cout << "Before swap, (num1, num2): " << num1 << ", " << num2 << endl;
54      swap(num1, num2);
55      cout << "After swap, (num1, num2): " << num1 << ", " << num2 << endl;
56
57
58  }
```

```
Before swap, (num1, num2): 5, 8
In function swap, (num1, num2): 8, 5
After swap, (num1, num2): 5, 8
```

Additional: std::bitset<8>(...) ☐ bit pattern

```
62  #include <iostream>
63  using namespace std;
64
65  void swap( int num1, int num2){
66      cout << "Initial, (num1, num2): " << std::bitset<8>(num1) << ", " << std::bitset<8>(num2) << endl;
67      num1 ^= num2;
68      cout << "step1, (num1, num2): " << std::bitset<8>(num1) << ", " << std::bitset<8>(num2) << endl;
69      num2 ^= num1;
70      cout << "step2, (num1, num2): " << std::bitset<8>(num1) << ", " << std::bitset<8>(num2) << endl;
71      num1 ^= num2;
72      cout << "In function swap, (num1, num2): " << std::bitset<8>(num1) << ", " << std::bitset<8>(num2) << endl;
73  }
74  int main(){
75      int num1 = 5, num2 = 8;
76      swap(num1, num2);
77  }
```

```
Initial, (num1, num2): 00000101, 00001000
step1, (num1, num2): 00001101, 00001000
step2, (num1, num2): 00001101, 00000101
In function swap, (num1, num2): 00001000, 00000101
```

Pass Arguments – Call by Address (Pointer)

- The addresses of the actual arguments are passed to the pointers used by the function.
- The function will affect the actual arguments if there are **changes in values** pointed by formal arguments.
- Ex: **void swap (int*, int*)**;

  call the function by **swap (&num1, &num2)**;

```
80  #include <iostream>
81  using namespace std;
82
83  void swap( int *num1, int *num2){
84      *num1 ^= *num2;
85      *num2 ^= *num1;
86      *num1 ^= *num2;
87      cout << "In function swap, (*num1, *num2): " << *num1 << ", " << *num2 << endl;
88  }
89  int main(){
90      int num1 = 5, num2 = 8;
91      cout << "Before swap, (num1, num2): " << num1 << ", " << num2 << endl;
92      swap(&num1, &num2);
93      cout << "After swap, (num1, num2): " << num1 << ", " << num2 << endl;
94
95  }
```

```
Before swap, (num1, num2): 5, 8
In function swap, (*num1, *num2): 8, 5
After swap, (num1, num2): 8, 5
```

Pass Arguments – Call by Reference

- Only in **C++**.
- The references to the actual arguments are passed to the formal arguments used by the function.
- The function will affect the actual arguments if there are changes in formal arguments.
- Ex: **void swap (int&, int&)**;

  call the function by **swap (num1, num2)**;

```
97  #include <iostream>
98  using namespace std;
99
100 void swap( int &num1, int &num2){
101     num1 ^= num2;
102     num2 ^= num1;
103     num1 ^= num2;
104     cout << "In function swap, (num1, num2): " << num1 << ", " << num2 << endl;
105 }
106 int main(){
107     int num1 = 5, num2 = 8;
108     cout << "Before swap, (num1, num2): " << num1 << ", " << num2 << endl;
109     swap(num1, num2);
110     cout << "After swap, (num1, num2): " << num1 << ", " << num2 << endl;
111
112 }
```

```
Before swap, (num1, num2): 5, 8
In function swap, (num1, num2): 8, 5
After swap, (num1, num2): 8, 5
```

Summary

- **Call by value:** The values are copied from the variables used in the caller main() to the variables used by the function. After the function call, the variables will **not be changed**.

- **Call by address:** The addresses of the actual arguments are passed to the pointers used by the function. The function will **affect the actual arguments** if there are changes in values pointed by formal.

- **Call by reference:** The references to the actual arguments are passed to the formal arguments used by the function. The function will **affect the actual arguments** if there are changes in formal arguments.

Summary

	Function argument	Function parameter	Will change the value of num1 and num2?	C or C++?
Call by Value	num1 num2	num1 num2	Not change in actual value	C, C++
Call by Address	&num1 &num2	*num1 *num2	Change in actual value	C, C++
Call by Reference	num1 num2	&num1 &num2	Change in actual value	C++

Exercise3

回『基礎題庫』

a273: 數獨驗證

標籤：
通過比率：75% (6 人 / 8 人) (非即時)
評分方式： Tolerant
最近更新：2020-09-28 19:28

公開題

Waiting | </> | 6人/8人 (75%)

內容 ：
數獨是一種源自18世紀末的瑞士數學家歐拉(Leonhard Euler)所創造的拉丁方塊游戲。

在9格寬×9格高的大九宮格中有9個3格寬×3格高的小九宮格，已經有一些數字在裡面了(但並非一定採用數字，例如採用字母a,b,c...)，根據這些數字，運用你的邏輯和推理,在其他的空格上填入1到9的數字，但是要注意了，每個數字在每個小九宮格內不能重複，每個數字在每行、每列也不能出現一樣的數字。如下圖所示：

	9							
1	2	3	4	5	6	7	8	9
	7							
	6		3	2	1			
	5		6	8	4			
	4		7	9	5			
	3							
	8							
	1							

Exercise3

現在請寫一個程式來判斷一個九宮格數字是不是一個數獨的正解。

＊若出現 1 ~ 9 之外的數字，請輸出Invalid

輸入說明

輸入第一行N為數獨矩陣的數量，接下來的每一組測試資料均為 9 × 9 的矩陣，且全部為 1~9 的數字，每兩組九宮格之間以「x」作為分隔

輸出說明

依照數獨矩陣數量輸出每一行：Valid 或 Invalid

測資資訊：

記憶體限制： 64 MB
公開 測資點#0 (25%): 1.0s , <1K
公開 測資點#1 (35%): 1.0s , <1M
公開 測資點#2 (40%): 1.0s , <1M

範例輸入

```
2
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 1
3 4 5 6 7 8 9 1 2
4 5 6 7 8 9 1 2 3
5 6 7 8 9 1 2 3 4
6 7 8 9 1 2 3 4 5
7 8 9 1 2 3 4 5 6
8 9 1 2 3 4 5 6 7
9 1 2 3 4 5 6 7 8
x
1 9 3 2 6 5 4 7 8
7 8 2 3 1 4 9 5 6
4 5 6 9 7 8 1 3 2
2 3 4 8 5 1 6 9 7
9 6 5 4 3 7 2 8 1
8 7 1 6 9 2 3 4 5
3 1 9 5 8 6 7 2 4
5 2 7 1 4 3 8 6 9
6 4 8 7 2 9 5 1 3
```

範例輸出

```
Invalid
Valid
```

Assign1 - 評分標準

評分標準

項目	配分
有交（含屍體）	20
有無錯誤判斷（>52張牌）	15
亂數處理	25
五張一排	15
印牌 （有無顏色、印不出來、格式跑掉）	25

Reminder - vim and Compile

vim 的指令(一般模式下)

- G 跳到最後一行
- gg 跳到第一行
- Ctrl + U: 往上捲動
- Ctrl + D: 往下捲動
- U: 復原
- Ctrl + R: 重做

cmd/terminal Compile

- Using g++ to compile your cpp code.
  - g++ a.cc b.cc (默認輸出檔名 a.out)
  - g++ -o test.out a.cc b.cc (指定輸出檔名為 test.out 而非 a.out)
- compile all .cc file at once
  - g++ *.cc
  - g++ -o test.out *.cc (指定輸出檔名為 test.out 而非 a.out)

Reminder - scp Command

scp 指令和 cp 類似，不同點是 scp 可以在不同主機間進行複制

scp 指令是在 local 端使用，scp [from] [to]

- 從 oop 工作站(server 端)複制 Cards.cc 檔案到 local 端桌面
  - scp s110xx@oop.cs.nccu.edu.tw:~s110xx/oop/assign1/Cards.cc "C:\Users*ACCOUNT_NAME*\Desktop"
- 從 oop 工作站(server 端)複制 assign1 資料夾到 local 端桌面
  - scp **-r** s110xx@oop.cs.nccu.edu.tw:~s110xx/oop/assign1 "C:\Users*ACCOUNT_NAME*\Desktop"
- 從 local 到 server 端，只是將參數調換
  - scp "C:\Users*ACCOUNT_NAME*\Desktop\Cards.cc" s110xx@oop.cs.nccu.edu.tw:~s110xx/oop/assign1
  - scp **-r** "C:\Users*ACCOUNT_NAME*\Desktop\assign1" s110xx@oop.cs.nccu.edu.tw:~s110xx/oop

如果目標路徑下有相同的檔案，會被複制過來的新檔案覆蓋，且不會有提示

請確保最終的作業檔案在工作站中編譯 (make)成功

More…

More about Linux command: https://kinsta.com/blog/linux-commands

More about vim command: https://www.computerhope.com/unix/vim.htm

oop_note: https://github.com/kebwlmbhee/oop_note

Any questions?