IM1003: Programming Design, Spring 2017
Lab 04

Jhih-Bang Hsieh
bigelephant29

National Taiwan University

Outline

- HW2 Code Review
- Array
- Function
- Practice

HW2-3

```cpp
#include <iostream>
using namespace std;
int main ()
{
    int num = 0, k = 0, a = 0 ;
    cin >> num >> k ;
    if (k == 1)
        {  //盡量不要過度縮排，一個 block 一個縮排。
            for (int i = 1 ; i <= num ; i++ )
                {
                    if (num % i == 0 )
                        cout << (a == 0 ? "" : " ") << i ;
                    a = 1 ;
                }
        }
    if (k == 2)
        {
            // 省略
        }
    return 0 ;
}
```

HW2-3

```
1   for(t; t <= n; t++)
2   {
3
4   }
5
6   for(; t <= n; t++)
7   {
8
9   }
10
11  for(;;)
12  {
13      //infinity loop
14  }
```

HW2-3

```
1   if(number % i == 0) // 建議加上大括號，增加可讀性。
2       for(int j = 2; j<=i; j++)
3       {
4           if(i % j == 0 && i != j)
5               break;
6
7           if(i == j)
8           {
9               if(k != 1)
10                  cout << " ";
11              cout << i;
12              k++;
13          }
14      }
```

HW2-4

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {     //請注意縮排對齊
5        int num = 0, sum = 0;
6        while(cin >> num)
7        {
8          //縮排距離請保持一致
9          for(int i=2; i <= num; i++)
10         {
11                 while(num % i == 0)
12                 {
13                     if(num % (i * i)==0)
14                     {
15                         //中間省略
16                     }
17                 }
18         }
19
20         cout << "\n";
21
22         } //這行與下一行應該要對齊
23     return 0;
24 }
```

Array

- 陣列分成靜態陣列、動態陣列。
- 前者在編譯時期決定大小，存在 stack 裡面。
- 後者在執行時期決定大小，存在 heap 裡面。

Static & Dynamic Array

- 靜態陣列速度很快。
- 動態陣列大小彈性，控制記憶體用量較容易。
- 非全域靜態陣列受 stack size 限制。

Declare a static array

```
1  int my_array[256];
```

- 這樣宣告了一個名字叫做 my_array 的靜態陣列，長度為 256。
- index: my_array[0] - my_array[255]
- 大小無法改變，宣告時無法用變數指定大小。

Declare a dynamic array

```
1  int n = 256;
2  int *my_array = new int[n];
3
4  delete [] my_array;
```

- 這樣宣告了一個名字叫做 my_array 的指標。
- 該指標指向一段長度為 256 的動態陣列。
- index: my_array[0] - my_array[255]
- 大小無法改變，但是宣告時可以用變數指定大小。
- 陣列使用完畢之後，須自行指定釋放記憶體（delete）。

Notice

```
1  int arr[] = {}; // wrong
2
3  int n;
4  cin >> n;
5  int arr[n]; // wrong
```

- 兩種寫法都是錯的。
- 靜態陣列須在編譯時期決定大小，所以不能讓使用者輸入時決定（執行時期）。

Notice

```
1  #include<iostream>
2  using namespace std;
3  int main() {
4      int arr[10000000];
5      for(int i = 0 ; i < 10000000 ; i++)
6          arr[i] = i;
7      cout << arr[9999999] << endl;
8      return 0;
9  }
```

- 非全域的靜態陣列存在 stack 裡面，受到 stack size 限制。
- stack size 通常不大，所以開一個很大的靜態陣列在 function 內是很危險的。
- 將這段程式碼抓到電腦上跑跑看，應該會出問題才對！
- 可以在編譯參數中將 stack 調大避免這個問題。
- 用動態陣列才是最好的方法！

Function

Function 就像是一個功能機，可以幫助你達成某些特定的功能。

例如：
有一個做飲料的 Function Drink，
傳入紅茶、牛奶，得到奶茶，Drink(black tea, milk)，
傳入綠茶、牛奶，得到奶綠，Drink(green tea, milk)。

當我們把任務妥善的包裝好之後，Function 就可以簡潔有力的幫我們處理一些繁複的工作。

*Practice G - 2D Rotation

給你一個正方形的二維矩陣，請輸出其順時針旋轉 90 度、180 度、270 度、360 度的結果。
輸入、輸出請見 PDOGS。

1 2 3
4 5 6
7 8 9

旋轉 90 度：

7 4 1
8 5 2
9 6 3

*Practice G - 2D Rotation

思考一下，如果我們將「把一個二維陣列旋轉 90 度」包裝成一個 Function，是不是就容易很多了呢？

延伸：為什麼不直接用迴圈去做重複的工作就好？

*Practice G - 2D Rotation

Function 比迴圈還具有更大的彈性，除了在任何地方都可以直接呼叫之外，Function 還可以帶有其他參數。

以這題來說，我們可以讓 Function 額外帶一個「度數」的參數，這樣旋轉時就不一定只受限於 90 度了。