C++

陣列 - 一維陣列

資料結構

- 資料結構（data structure）是電腦中儲存、組織資料的方式
- 資料量會變多、變多樣
- 靜態資料結構：
- 靜態資料結構：在程式執行期間的大小並不會改變
  - 陣列：相同類型的相關資料項目所組成(本單元探討陣列)
  - 結構：不同類型的相關資料項目所組成
- 動態資料結構
  - 串列
  - 柱列
  - 堆疊
  - 樹

爲什麼要使用陣列

當資料量變大時

- 前面幾章的學習，我們已經可以處理許多複雜的問題
- 只是當資料量變多時，依靠前面的知識是不夠的
- 即使簡單的問題也需要不同的方法來解決

輸入5個整數，
並且印出5個整數。

輸入50個整數，
並且印出50個整數。

宣告50個變數是不能解決問題的

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

int main(){

    int a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,...,a48,a49,a50;

    return 0;
}
```

使用迴圈輸入50個整數，且印出50個整數。

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

int main(){
    int a[50],i;
    for(i=0;i<50;i++){
        cin>>a[i];
        cout<<a[i]<<endl;
    }
    return 0;
}
```

- 使用迴圈，改善了輸出入的效率
- 但是
  - 在記憶體中，並沒有紀錄每一個資料

陣列才是輸入大量相同類型變數的正解

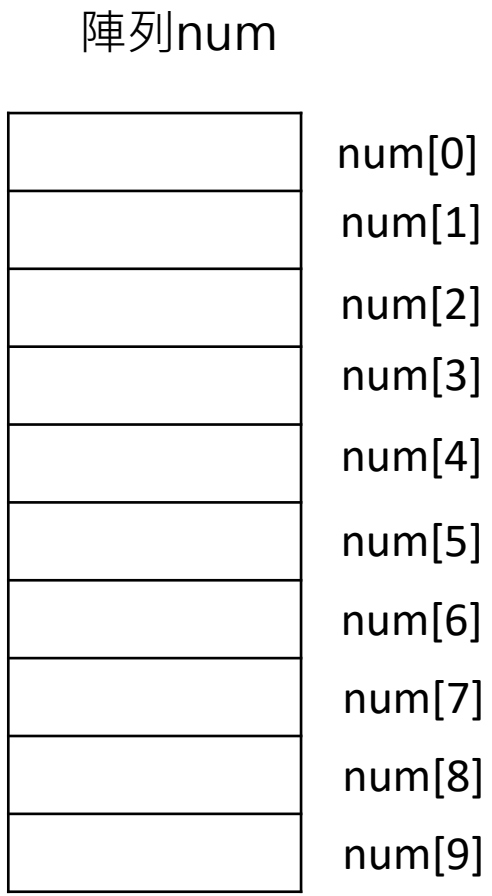
```cpp
#include <iostream>
using namespace std;

int main(){
    int a[50],i;
    for(i=0;i<50;i++){
        cin>>a[i];
        cout<<a[i]<<endl;
    }
    return 0;
}
```

陣列的定義

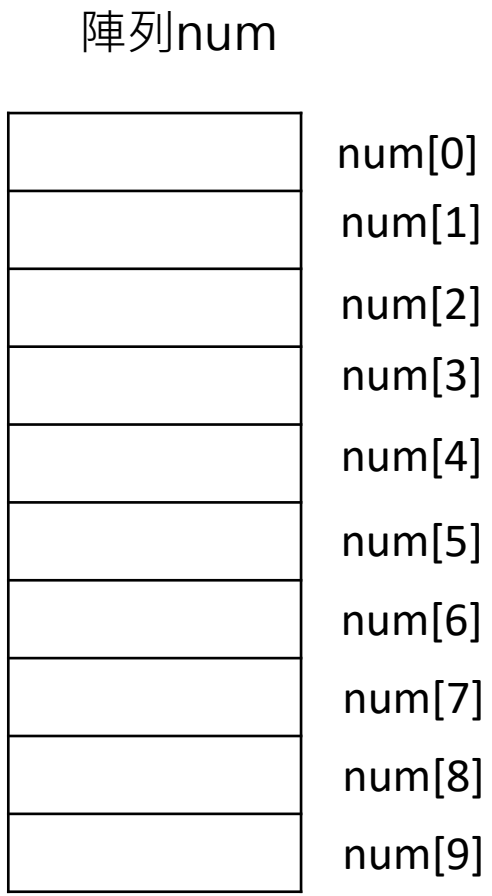
陣列

- 一群具有相同名稱以及相同類型的記憶體位置，此陣列所有的元素都具有相同名稱num
- 若要引用陣列的某個位置或元素，必須指定陣列名稱，以及元素在陣列中的位置編號
- 右圖表示一個名為num的整數陣列，陣列含有10個元素，在程式的宣告方法，如右上第01行：
  num[10]



陣列元素的引用

- 每一個陣列中的第一個元素均是第0個元素
- 陣列num的引用
  - 第1個元素：num[0]
  - 第2個元素：num[1]
- 一般化的表示式：
  - 第i個元素：num[i-1]
- 陣列的命名原則如同變數
- 中跨號中的位置編號，正式名稱為**下標**



下標的使用

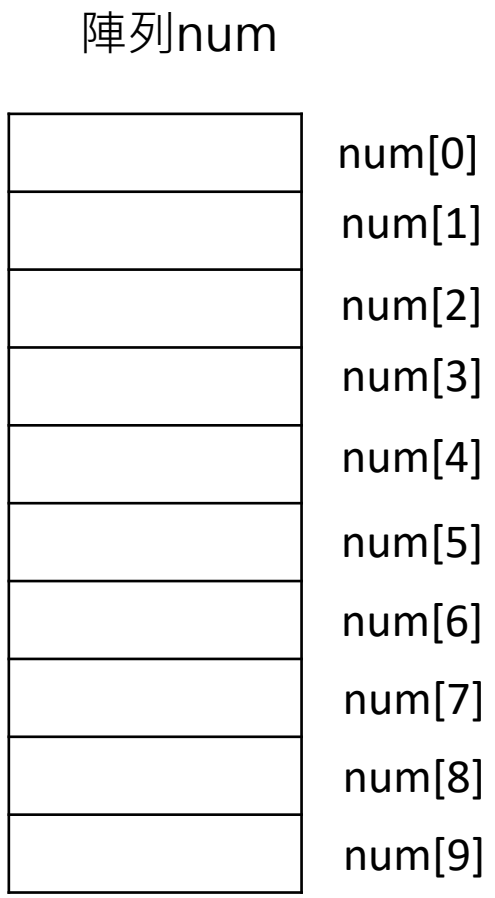
- num陣列10個元素，分別是num[0]是-45， num[1]是6，num[2]是0， num[8]是-3， num[9]是1
- 下標必須是整數或整數運算式
  - 若a=5且b=6，num[a+b]+=2
    以上會將陣列元素num[11]加2
- 前三個元素，所含的值的和
  printf(“%d”,c[0]+c[1]+c[2])
- 第6個元素除以2，結果設定給變數x
  x=c[6]/2
- 陣列中的**第6個元素**與**陣列元素6**的差異：
  前者指下標是5，後者指下標6

陣列num

-45	num[0]
6	num[1]
0	num[2]
72	num[3]
1543	num[4]
-89	num[5]
0	num[6]
62	num[7]
-3	num[8]
1	num[9]

定義陣列

- 陣列會占用記憶體空間，需要指定每一個元素型別和元素個數，電腦會依此預留適當的記憶體空間。
- 格式
  資料型態 陣列名[陣列長度];
- 例如: 若要告訴電腦預留具有10個元素的整數陣列預留空間，寫法如下：
  int num[10]
- int x[100], y[1000];
  表示x預留100個元素的位置，y則是預留1000個元素的位置
- 一個型別為char的陣列，則可以用來存放字元自串，後面章節討論



使用陣列的例子

定義一個陣列並且使用迴圈來初始化陣列元素

```
#include<stdio.h>
int main(){
   int array[10]; /*宣告一個長度為10的陣列*/
   int i;
            for(i=0;i<10;i++){
                     cin>>array[i]>>endl;
            }
            printf("Array Value\n");
            for(i=0;i<10;i++){
                     cout<<"array["<<i<<"] "<<array[i]<<endl;
            }
            return 0;
}
```

輸入資料：
10 9 8 7 6 5 4 3 2 1
輸出結果：

```
Array Value
array[0] 10
array[1] 9
array[2] 8
array[3] 7
array[4] 6
array[5] 5
array[6] 4
array[7] 3
array[8] 2
array[9] 1
```

可以在初始列表中定義全部陣列的值

- **格式**
  - 資料型態 陣列名[常量]={值1, 值2, ...}
- **例如:**
  - int num[5]={1, 2, 3, 4, 5}
- 在初始列表中可以寫出全部陣列的值，也可以寫出部分。沒寫的部分會被列爲0
  - int x[10]={0, 1, 2, 3, 4 }
- 全部初始化爲0
  - int num[5]={}

可以在初始列表中定義全部陣列的值

- 格式
  - 資料型態 陣列名[常量]={值1, 值2, ...}
- 例如：
  - int num[5]={1, 2, 3, 4, 5}
- 在初始列表中可以寫出全部陣列的值，也可以寫出部分。沒寫的部分會被列爲0
  - int x[10]={0, 1, 2, 3, 4 }
- 全部初始化爲0
  - int num[5]={}

使用陣列的越界

使用陣列時，要注意

- 陣列下標值爲正整數

- 在定義元素個數的下標範圍內使用
  int a[5]={1, 2, 3, 4, 5}

- 如果下標寫成負數或者大於陣列元素的個數時，成爲**陣列越界**
  程序編譯是不會錯的，例如：
  int a[10];
  a[-3]=5;
  a[20]=15;
  a[10]=20;

陣列越界造成的後果

- 這種類型的錯誤，常是難以捕捉
- 因爲越界本身並不一定導致程序立即出錯，可能遇到某些資料時才導致錯誤，有時由於越界，意外的改變了變量或指令，導致在編譯器裏編譯的時候，程序不按照應當的次序運行的怪現象。

問題

從標準輸入取得一系列30個整數，請你印出這30個整數。

從標準輸入取得一系列30個整數，請你印出這30個整數。

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

int main(){
    int array[30];
    int i;
    for(i=0;i<30;i++){
        cin>>array[i];
    }
    printf("Array Value\n");
    for(i=0;i<30;i++){
        cout<<"array["<<i<<"] "<<array[i]<<"\n";
    }
    return 0;
}
```

使用迴圈與陣列輸入資料

使用迴圈與陣列輸出資料