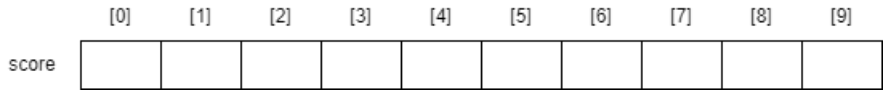
5.1 一維陣列

陣列(Array)的結構

陣列這種資料結構是用來儲存許多相同型別的資料用的。如果我們把變數想像成是一個可以放東西的箱子，那麼陣列就是一堆箱子的集合，而且每個箱子都有一個連續編號的索引值(index)。

例如：我們要儲存 10 個學生的成績(都是整數)，我們可以使用這樣一個內含 10 個元素(element)/項目(item)的陣列。



宣告陣列

在程式中我們可以這樣宣告這個陣列 score。

```
int score[10];
```

其語法為

```
型別 陣列名稱[元素數量];
```

其中陣列名稱的命名規則與一般變數的命名規則相同。

要特別注意的是，陣列的索引值是由 0 開始，所以宣告大小為 10 的陣列 score。可以使用的元素是 `score[0]` ~ `score[9]`。

給定初值

如同變數可以在宣告同時給定初值，陣列也可以。

如果只宣告，但不給定初值，則陣列內各元素的值會無法預測(會是分配到的記憶體當下的值)。

```
int a[5] = {1, 3, 5, 7, 9};

for(int i=0; i<5; i++)
{
    cout << a[i] << " ";
}
```

```
1 3 5 7 9
```

初值不給足

如果陣列有 5 個元素，但是初值只給 2 個，剩下 3 個的值會是什麼？

```
int a[5] = {1, 3};

for(int i=0; i<5; i++)
{
    cout << a[i] << " ";
}
```

```
1 3 0 0 0
```

觀察執行結果，可以發現它們被設為 0。

所以對於整數陣列，我們常用這樣的技巧來宣告並指定其初值皆為 0。

```
int a[5] = {0};

for(int i=0; i<5; i++)
{
    cout << a[i] << " ";
}
```

```
0 0 0 0 0
```

讓編譯器幫你算數量

我們可以在宣告時給初值但不指定陣列大小，編譯器會幫你算好填入。

```
int a[] = {1, 3, 5, 7};  // 相當於 int a[4] = {1, 3, 5, 7};
```

存取陣列中指定元素的值

原則上存取陣列 a 裡索引為 i 的元素值，和一般變數一樣，只要用 a[i] 來表示要存取的元素即可。

練習：讀取學生成績，並接受查詢

讀取使用者輸入的 1~10 號學生成績，並接受以座號查詢其成績。輸入 0 結束程式。

```
int score[10] = {0};

for(int i=0; i<10; i++)
{
    cin >> score[i];
}

while(true)
{
    cout << "輸入座號查詢成績: ";
    int id;
    cin >> id;
    if(id==0)
        break;
    cout << id << " 號的成績為 " << score[id-1] << endl;  // 想一想，為什麼索引值是 id-1，而不是 id？
}
```

```
11 22 33 44 55 66 77 88 99 100
輸入座號查詢成績:3
3 號的成績為 33
輸入座號查詢成績:6
6 號的成績為 66
輸入座號查詢成績:0
```

因為陣列的索引值是由 0 開始編號，和我們一般生活中由 1 開始編號的情境不同。所以也有人會選擇「浪費一個元素」來讓程式寫起來比較直覺。

```
int score[11] = {0};  // 宣告 11 個，索引 0 那個不用

for(int i=1; i<=10; i++)  // i 由 1~10，而不是 0~9
{
    cin >> score[i];
}

while(true)
{
    cout << "輸入座號查詢成績: ";
    int id;
```

```
    cin >> id;
    if(id==0)
        break;
    cout << id << " 號的成績為 " << score[id] << endl;  // id 不用減 1 了
}
```

陣列大小在宣告後無法改變

陣列大小在宣告後無法改變，所以通常我們會宣告「足夠」的大小。例如：在班級成績儲存時，若班級人數不超過 50 人，我們會宣告大小為 50 的陣列。

練習：讀取學生成績，並接受查詢(n 人版)

讀取使用者輸入的 1~n 號學生成績，並接受以座號查詢其成績。輸入 0 結束程式。 班級人數不超過 50 人。

輸入說明：

- 輸入的第一行為正整數 n，表示接下來有 n 個整數，分別代表 1~n 號學生的成績。

```
int score[50] = {0};  // 足夠的大小
int n;
cin >> n;

for(int i=0; i<n; i++)
{
    cin >> score[i];
}

while(true)
{
    cout << "輸入座號查詢成績: ";
    int id;
    cin >> id;
    if(id==0)
        break;
    cout << id << " 號的成績為 " << score[id-1] << endl;
}
```

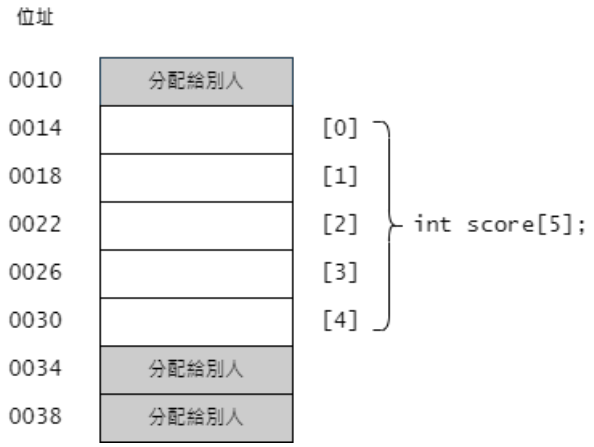
為什麼陣列的大小不能在程式執行中動態改變呢？

這可能跟陣列的特性有關，陣列有以下的特點：

1. 所有元素都是相同型別
2. 所有元素在記憶體中相鄰緊密排列
3. 可以依索引值快速隨機存取(無需循序)任一內部元素

其中 3 是因為 1, 2 才有辦法做到的。

以下面這個陣列為例：



因為每個元素都是 int，也就是佔記憶體的大小都是 4 byte。所以只要有陣列的開頭位址 $ 0014 $，和索引 $ i $，就可以知道 score[i] 在記憶體裡的位址為 $ 0014+i*4 $。

如果我們可以在記憶體中另外找 5 個 int 大小的空間給 score 來讓它的 size 由 5 變成 10。則這兩塊不連續的空間便無法再擁有原來設計的高速隨機存取優勢。

C99 的可變長度陣列(VLA)

上一個練習題，你會不會很想要這樣寫呢？

```
int n;
cin >> n;  // 先知道 n 的值

int score[n] = {0};  // 再宣告大小剛好為 n 的陣列

for(int i=0; i<n; i++)
{
    cin >> score[i];
}

......
```

實際寫下去執行，會發現還真的可以成功，這是為什麼呢？

C 語言的 C99 標準，支援可變長度陣列 (VLA, variable-length array) 。所以我們可以像上面那樣在執行中宣告一個以變數指定大小的陣列(但是之後就固定那個大小)。

由於我們使用的 C++ 編譯器 gcc 使用 extention 支援了 VLA，所以也可以做到。但是這個並不是 C++ 標準裡的東西，也就是並非所有的 C++ 編譯器會支援，你的程式碼可能在別的環境下會編譯失敗。

安全性問題

看一下以下的程式碼，預測他的輸出結果。

```
int numberOfStudent = 6;
int score[6];

for(int i=1; i<=6; i++) {  // 依序輸入 10 20 30 40 50 60
    cin >> score[i];
}
cout << numberOfStudent << endl;
```

使用 Code::Blocks 預設的 32 位元編譯器來編譯執行後，令人意外的，我們在程式裡根本沒有動到 numerOfStudent，但是輸出時卻發現 numerOfStudent 已經由 6 變成 60 了，Why？

因為程式裡宣告了 `int score[6];`，理應只使用 `score[0]`~`score[5]`，但是我們誤操作為 `score[1]`~`score[6]`。而 `score[6]` 推算出來的位址正好是 numberOfStudent 所在的位置。

編譯器不會提出警告，因為這是合法的操作(雖然在這情境下不合理)。程式設計師要自己負責做這種邊界檢查。

這讓 C++ 的程式變得容易有安全弱點，所以近年來有人提議使用會自己做記憶體管理和邊界檢查的程式語言。但是相對的就要付出一定的性能做為代價。

競賽可能遇到的問題

在競賽時為了搶時間求效能，我們常會宣告一個很大的陣列，而不是在那斤斤計較的省記憶體。

下面個程式可以成功編譯，但是執行後就直接 crash，連第一行 cout 都沒執行到。

```
#include <iostream>

using namespace std;

int main()
{
    int dat[200000001];  // 宣告在區域(local)

    cout << "Input a integer:";
    cin >> dat[200000000];
    cout << dat[200000000] << endl;

    return 0;
}
```

結束時返回的狀態碼是 **Process terminated with status -1073741571** (stack over flow)。在比賽時你可能會得到的訊息是 **segmentation fault**。

如果不要把它宣告在 main 函數內，而是宣告存全域區，讓它成為全域變數則可以成功執行。

```
#include <iostream>

using namespace std;

int dat[200000001];  // 宣告在全域區(global)

int main()
{
    cout << "Input a integer:";
    cin >> dat[200000000];
    cout << dat[200000000] << endl;

    return 0;
}
```

差別在哪裡呢？宣告在 local 的話，會用 stack 裡的記憶體來配置給它，而預設的 stack 都不大，可能只有幾 MB。而宣告在 global，則會配置在 data segment 裡，有更大的空間可用。

所以在比賽時，如果沒有變數污染的顧慮，宣告在 global 會比較好。

Revision #21
★Created 8 April 2024 02:28:33 by huihui
Updated 8 December 2025 07:04:14 by huihui