Bài 3 Array

Mảng

Bài học này là phần tiếp theo của bài trước, nó giúp bạn hiểu thêm về cách sử dụng các pointer. Bạn cảm thấy có chút khó khăn trong việc sử dụng các pointers?

Bạn sẽ không tránh khỏi việc dùng chúng đâu! Các pointers được sử dụng thường xuyên trong C, tôi đã nói với bạn điều này!

Trong bài học này, chúng ta sẽ học cách tạo những biến số type "array" hay còn gọi là mảng. Các mảng được sử dụng thường xuyên trong C vì nó tiện lợi trong việc sắp xếp một chuỗi các giá trị.

Bài học được bắt đầu bằng vài lời giải thích về cách hoạt động của các arrays trong bộ nhớ. Tôi nhận thấy việc mở đầu với các kiến thức trong phần bộ nhớ vô cùng quan trọng: Nó giúp bạn hiểu được phương thức hoạt động. Một lập trình viên hiểu được điều họ làm, điều này sẽ đảm bảo chương trình viết ra chạy ổn định hơn, bạn nghĩ sao?

Các arrays trong bộ nhớ

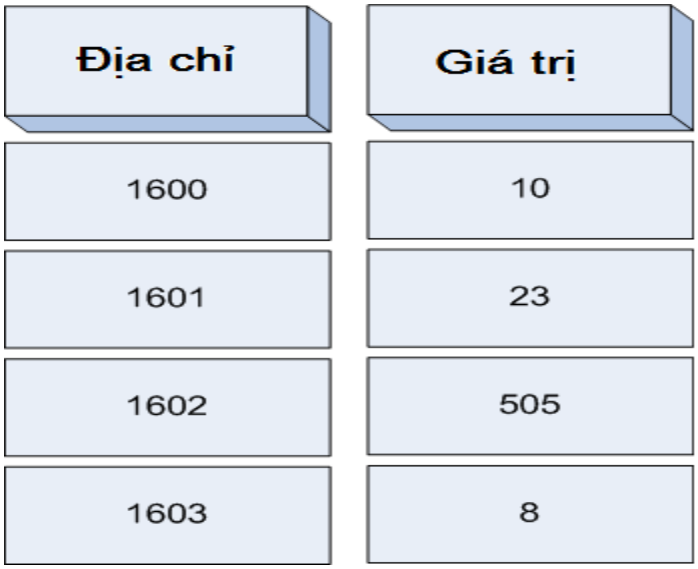
"Arrays là một dãy các biến số cùng type, chứa trong một vùng bộ nhớ liên tục."

Lời giải thích trên có vẻ giống trong từ điển phải không?

Rõ ràng hơn, mảng có thể chứa một số lượng lớn biến số cùng type (*long, int, char, double*...).

Mỗi mảng có một kích thước xác định. Nó có thể tạo bởi 2, 3, 10, 150, 2500 cases (ô, slots), tùy theo tùy chọn của bạn.

Biểu đồ sau là một mảng kích thước 4 ô trong bộ nhớ, nó bắt đầu từ địa chỉ 1600 :



Khi bạn yêu cầu tạo một mảng kích thước 4 ô trong bộ nhớ, chương trình sẽ yêu cầu hệ điều hành quyền sử dụng 4 ô bộ nhớ. 4 ô này phải nằm kề nhau, có nghĩa là ô sau sẽ kế tiếp ô trước. Giống như trên, các địa chỉ nằm nối tiếp nhau: 1600, 1601, 1602, 1603 và không có "khoảng trống" nào ở giữa.

Cuối cùng, mỗi ô trong mảng chứa một số cùng type. Nếu mảng có type int, thì mỗi ô trong mảng chứa một số type int. Không thể tạo mảng cùng lúc chứa giá trị type int và double.

Tóm lại, sau đây là những điều buộc phải ghi nhớ:

- Khi một mảng (array) được tạo ra, nó sử dụng **một vùng liên tục trong bộ nhớ**: ở đó các ô bộ nhớ sẽ nằm liên tục kề nhau.
- Tất cả các ô (case) trong mảng **phải cùng type**. Một array type int chỉ chứa các số dạng int, không thể chứa các số dạng khác.

Cách tạo một mảng (array)

Bắt đầu, chúng ta sẽ xem làm thế nào để tạo một mảng chứa 4 giá trị int:
C code:

```
int array[4] ;
```

Vậy thôi, ta chỉ cần thêm vào trong dấu ngoặc vuông [ ] số lượng ô mà bạn muốn chứa trong mảng. Không có giới hạn (tùy theo dung lượng bộ nhớ của máy tính).

Vậy bây giờ, làm cách nào đưa giá trị vào mỗi case trong mảng? Rất đơn giản, chỉ cần viết array[số_thứ_tự_của_case]


Chú ý: một mảng bắt đầu từ số 0! Mảng chứa 4 giá trị int có các ô với số thứ tự 0, 1, 2, và 3. Không có ô số 4 trong array 4 cases! Các bạn hay nhầm lẫn ở đây, nhớ kĩ.

Nếu tôi muốn thêm vào mảng các giá trị giống như trong biểu đồ trên, tôi sẽ viết:

C code:

```
int array[4] ;
array[0] = 10;
array[1] = 23;
array[2] = 505;
array[3] = 8;
```


Vậy mối quan hệ giữa mảng và pointer là gì ?

Nếu bạn chỉ viết là array thì đó chính là pointer. Đó là một pointer chỉ vào ô đầu tiên của mảng. Test :

C code:

```
int array[4] ;
printf ("%d", array);
```

Kết quả, ta nhận được ô địa chỉ đầu tiên của mảng:

Console:

```
1600
```

Nếu bạn ghi thứ tự của ô trong mảng vào ngoặc vuông [ ], bạn sẽ nhận được giá trị của ô đó:

C code:

```
int array[4];
printf ("%d", array[0]);
```

Console:

```
10
```

Tương tự với các ô khác. Nhắc lại là nếu bạn viết array, nó sẽ là một pointer, chúng ta có thể sử dụng kí tự * để có được giá trị của ô đầu tiên:

C code:

```
int array[4];
printf ("%d", *array);
```

Console:

```
10
```

Và có thể nhận giá trị của ô tiếp theo với *(array+1) (địa chỉ của array+1). Cả 2 dòng code sau hoạt động tương tự nhau:

C code:

```
array[1] // Cho gia tri cua o thu 2 (o dau tien viet la [0])
*(array+ 1) // Tuong tu: cho gia tri o thu 2
```

Nói rõ hơn, nếu bạn viết array[0], cũng giống như bạn yêu cầu giá trị tìm thấy ở địa chỉ array + 0 (ở ví dụ là 1600). Nếu bạn viết array[1], bạn sẽ nhận được giá trị ở địa chỉ array + 1 (1601 trong ví dụ). Và tương tự với những trường hợp còn lại.

Dynamic array(mảng động)

Ngôn ngữ C tồn tại rất nhiều versions. Version trước đây, gọi là C99, cho phép tạo các dynamic array, có nghĩa là mảng với kích thước được khai báo bởi một biến số:

C code:

```
int kichThuoc = 5;
int array[kichThuoc];
```

Cách viết này không được thông dụng lắm đối với các compiler, nhiều khi chương trình chạy đến ở dòng thứ 2 sẽ dừng lại. Từ đầu đến giờ, tôi hướng dẫn bạn ngôn ngữ C89 nên chúng ta sẽ tuyệt đối không dùng dòng code thứ 2

Vậy là, các bạn không được viết một biến số đặt trong ngoặc vuông để khai báo kích thước một mảng, kể cả khi biến số này là hằng số (constant)! Một mảng phải có kích thước xác định, có nghĩa là khi khai báo bạn phải ghi rõ ràng kích thước của mảng đó bằng một con số:

C code:

```
int array[5] ;
```

Vậy tại sao lại cấm tạo một mảng với kích thước phụ thuộc vào một biến số?

Tôi bảo đảm với bạn: điều này là có thể thực hiện! ngay cả trong C89.

Nhưng để thực hiện điều này, chúng ta sẽ dùng một kĩ thuật khác (chắc chắn, đảm bảo có thể chạy được trên mọi điều kiện) gọi là phân bổ động (dynamic allocation). Chúng ta sẽ được học về sau.

Liệt kê các giá trị trong mảng (array)

Bây giờ tôi muốn hiển thị giá trị mỗi ô trong mảng. Tôi có thể sử dụng số lượng printf bằng với số ô trong mảng. Nhưng tôi phải viết nhiều lần printf, điều này thật nhàm chán (hãy tưởng tượng đến trường hợp mảng chứa 8000 giá trị)

Vì vậy, tốt hơn là sử dụng vòng lặp. Và vòng lặp *for* rất tiện lợi trong việc này:

C code:

```
int main (int argc, char *argv[ ])
{
   int array[4], i = 0;
   array[0] = 10;
   array[1] = 23;
   array[2] = 505;
   array[3] = 8;
   for (i = 0 ; i < 4 ; i++)
   {
      printf ("%d\n", array[i]);
   }
   return 0;
}
```

Console:

```
10
23
505
8
```

Vòng lặp sẽ chạy dọc các ô trong mảng nhờ vào biến số i (các nhà lập trình thường dùng i, đây là một biến số khá thông dụng để chạy dọc một mảng)

Cách này đặc biệt thông dụng, ta để một biến số trong dấu [ ]. Những biến số tuyệt đối cấm sử dụng trong việc tạo các mảng (để khai báo kích thước), nhưng nó được phép sử dụng để "di chuyển" trong mảng, có nghĩa là để hiển thị các giá trị! Trong ví dụ trên, tôi đặt biến số i, nó sẽ tăng dần từ 0, 1, 2 rồi 3. Như vậy, nó sẽ hiển thị các giá trị của array[0], array[1], array[2] và array[3]!

 Cần chú ý là không nên hiển thị giá trị của array[4]! Một array 4 ô chỉ có số thứ tự là 0, 1, 2, 3. Nếu bạn thử hiển thị giá trị của array[4], nó sẽ hiển thị một số không xác định, đây sẽ là một lỗi khá đẹp. Hệ điều hành sẽ dừng chương trình lại do nó cố ý xâm nhập vào một địa chỉ không cho phép.

Khởi tạo các giá trị trong một mảng

Bạn đã biết cách di chuyển trong mảng, điều này có nghĩa bạn có thể khởi tạo mảng với giá trị 0 ở tất cả các ô bằng việc sử dụng vòng lặp!

Bạn đã đạt được trình độ cần thiết có thể thực hiện điều này  :

C code:

```
int main (int argc, char *argv[ ])
{
   int array[4], i = 0;
   // Khoi tao cac gia tri trong array
   for (i = 0 ; i < 4 ; i++)
   {
      array[i] = 0;
   }
   // Hien thi cac gia tri de kiem tra
   for (i = 0 ; i < 4 ; i++)
   {
      printf ("%d\n", array[i]);
   }
   return 0;
}
```

Console:

```
0
0
0
0
```

Một cách khác để khởi tạo giá trị

Bạn cần biết là còn một cách khác để khởi tạo giá trị trong mảng khá thủ công trong C.
Bằng cách viết từng giá trị trong ngoặc nhọn { }, cách nhau bởi dầu phẩy ",":
array[4] = { giaTri1, giaTri2, giaTri3, giaTri4};

C code:

```
 int main (int argc, char *argv[ ])
{
   int array[4] = {0, 0, 0, 0}, i = 0;
   for (i = 0 ; i < 4 ; i++)
   {
      printf ("%d\n", array[i]);
   }
   return 0;
}
```

Console:

```
0
0
0
0
```

Mặt khác, lợi ích của cách làm này là, bạn chỉ cần khai báo giá trị những ô đầu tiên, những ô còn lại sẽ tự động nhận giá trị 0:

Cụ thể, nếu tôi viết:

C code:

```
int array[4] = {10, 23}; // Gia tri nhap vao: 10, 23, 0, 0
```

Ô đầu tiên nhận giá trị 10, ô thứ 2 nhận giá trị 23, các ô còn lại nhận giá trị 0.
Vậy làm cách nào để khai báo tất cả các ô với giá trị 0? Bạn chỉ cần khai báo ô đầu tiên giá trị 0, sau đó các ô còn lại cũng sẽ nhận giá trị 0.

C code:

```
int array[4] = {0} // Khai bao array voi tat ca cac o gia tri 0.
```

Kỹ thuật này có thể hoạt động với bất kì kích thước nào (có thể thực hiện trên array 100 ô và hơn nữa)

Chú ý, thường gặp lỗi
int array[4] = {1};
Dòng code này sẽ thêm vào các giá trị sau: 1, 0, 0, 0. Nhiều bạn nghĩ rằng dòng code trên sẽ khởi tạo mảng với tất các giá trị là 1, nhưng không phải vậy, để làm điều này các bạn cần sử dụng vòng lặp.

Tạo một function để liệt kê các giá trị trong mảng

Bạn đã biết cách hiển thị nội dung của một mảng.
Vậy tại sao bạn không thử viết một chương trình để thực hiện điều này? Như thế bạn có thể tìm hiểu cách đưa một array vào function.

Chúng ta sẽ cần gửi hai parameter vào function: đầu tiên là array (địa chỉ của array) và thứ hai sẽ là kích thước của nó! Và function của bạn có khả năng hoạt động với bất kì array nào khác đưa vào. Chúng ta sẽ cần một biến số *kichThuocArray*.

Bạn biết rằng array có thể xem như là một pointer. Vì vậy, chúng ta sẽ đưa array vào function giống như thực hiện với pointer:

C code:

```
// Prototype cua function hien thi
void hienThi (int *array, int kichThuocArray);

int main (int argc, char *argv[ ])
{
   int array[4] = {10, 15, 3};

   // Chung ta hien thi noi dung cua array
   hienThi (array, 4);

   return 0;
}

void hienThi (int *array, int kichThuocArray)
{
   int i;

   for (i = 0 ; i < kichThuocArray ; i++)
   {
      printf ("%d\n", array[i]);
   }
}
```

Console

```
10
15
3
0
```

Function này không khác nhiều so với function chúng ta được học ở bài trước. Chúng ta sẽ đưa vào một parameter pointer type int (array), sau đó là kích thước của array (rất quan trọng để biết khi nào vòng lặp dừng lại!). Nội dung của array sẽ được hiển thị qua function nhờ vào một vòng lặp.

Cần ghi thêm là còn một cách khác để đưa mảng vào function:

C code:
```
void hienThi (int array[ ], int kichThuocArray)
```

hoạt động tương tự như trên, việc sử dụng những dấu ngoặc vuông cho phép người đọc code hiểu rõ: function cần một mảng. Có thể hạn chế được nhầm lẫn là function cần một con trỏ hoặc biến cơ bản nào đó.

Tôi thường sử dụng cách viết thứ hai để đưa mảng vào function, các bạn nên thực hiện giống tôi. Và trong cách viết này chúng ta không cần khai báo kích thước trong ngoặc vuông.

Một vài bài tập thực hành!

Tôi lúc nào cũng có rất nhiều bài tập cho bạn luyện tập !
Các bạn nên tự viết các function làm việc với array.
Tôi chỉ đưa đề bài cho các bạn thực hành, về code, các bạn sẽ tự viết lấy. Hãy đặt câu hỏi, nếu có thắc mắc.

Bài tập 1
Tạo một function tongArray để tính tổng các giá trị chứa trong nó (sử dụng return để trả về giá trị). Và để giúp bạn hiểu rõ hơn, đây là prototype của function cần viết:
C code:
```
int tongArray (int array[ ], int kichthuocArray);
```

Bài tập 2
Tạo một function trungBinhArray để tính trung bình các giá trị chứa trong nó.
Prototype:
C code:
```
double trungBinhArray (int array[ ], int kichThuocArray);
```

Bài tập 3
Tạo một function copyArray để chép nội dung array này sang một array khác.
Prototype:
C code:
```
void copyArray (int array1[ ], int array2[ ], int kichThuoc);
```

Bài tập 4

Viết một function maximumArray có nhiệm vụ so sánh tất cả các giá trị chứa bên trong array với giaTriMax. Nếu có giá trị lớn hơn biến số giaTriMax đưa vào, nó sẽ chuyển thành 0.
Prototype:
C code:

```
void maximumArray (int array[ ], int kichThuoc, int giaTriMax);
```

VD: array {1,5,7,8,5,2,3} và max=5, sẽ chuyển thành {1,5,0,0,5,2,3}.

Bài tập 5

Bài tập này khó hơn hẳn các bài tập kia nhưng bạn hoàn toàn có khả năng thực hiện.
Hãy viết một function sapXepArray sắp xếp lại các giá trị bên trong theo thứ tự tăng dần.
C code:

```
void sapXepArray (int array[ ], int kichThuoc);
```

VD: array {1,5,7,8,5,2,3} sẽ chuyển thành {1,2,3,5,5,7,8}.

> Hãy viết trong file array.c chứa tất cả các functions cần thiết và file array.h chứa các prototypes của các functions đó.

Nào bắt đầu làm việc thôi !

Nếu bạn qua được bài học về pointer thì các bài khác các bạn đều có thể vượt qua. Tôi không nghĩ là bài học này sẽ gây khó khăn cho bạn.

Bạn nên nhớ hai điều quan trọng sau:

- **Đừng bao giờ quên** một array bắt đầu bằng số thứ tự 0 (không phải 1)
- Khi bạn đưa một array vào function, luôn gửi kèm theo kích thước của array.

Và tôi sẽ cho bạn một tin vui, **bạn đã có đủ trình độ để có thể làm việc với các chuỗi kí tự**.
Bạn có thể đưa một chuỗi kí tự vào bộ nhớ, ví dụ như việc yêu cầu họ tên người dùng.

Và trong bài học sau chúng ta sẽ học cách làm việc với các chuỗi kí tự !
