Bài 8

Lập trình C
Mảng và Chuỗi

Nội dung

- Mảng
- Mảng một chiều
- Chuỗi kí tự
- Mảng hai chiều
- Khởi tạo mảng
- Lỗi với mảng và chuỗi
- Tên biến mảng và con trỏ
- Con trỏ chỉ mục
- Mảng các chuỗi kí tự

Mảng

- **Mảng** là tập hợp các biến **cùng kiểu** được gọi bằng một **tên chung**. Mỗi biến được gọi là một phần tử.
- Một phần tử cụ thể trong mảng được truy cập bằng chỉ mục.
- Trong ngôn ngữ C, tất cả các phần tử của mảng được lưu trữ tại các vị trí bộ nhớ liền kề nhau.
- Mảng có thể có từ một chiều đến nhiều chiều.
- Mảng phổ biến nhất là chuỗi, nó là một mảng các ký tự được kết thúc bằng một giá trị NULL.

Mảng một chiều - Khai báo

- Khai báo mảng một chiều

  <kiểu dữ liệu> <tên biến>[kích thước];

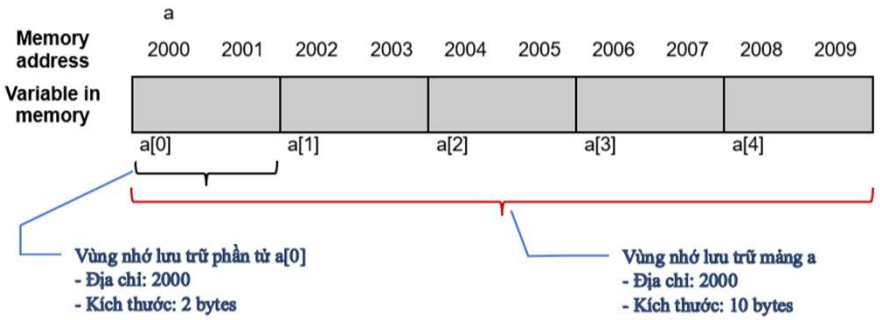
- Trong đó
  - ***<kiểu dữ liệu>*** là kiểu của mỗi phần tử trong mảng.
  - ***<kích thước>*** là một hằng nguyên xác định số lượng phần tử mà mảng sẽ chứa.

```
/* Declare
   - a 100-element array named balance of type double
   - a 50-element array named number of type int
   - a 10-element array named pointer of type int * */
double balance[100];
int number[50];
int *pointer[10];
```

Mảng một chiều - Tổ chức lưu trữ

- Tổ chức lưu trữ mảng một chiều trong bộ nhớ

```
int a[5];
```



- Mảng **a** (đối tượng kiểu **int[5]**) và phần tử **a[0]** (đối tượng **kiểu int**) có cùng giá trị địa chỉ là 2000 nhưng khác kiểu.

Mảng một chiều - Truy cập phần tử

- Các phần tử của mảng được đánh chỉ mục
  - Phần tử đầu tiên có chỉ mục là 0.
  - Mảng có kích thước là n thì phần tử cuối cùng có chỉ mục là n - 1.
- **Phép toán chỉ mục [ ]** dùng để **truy cập phần tử** của mảng

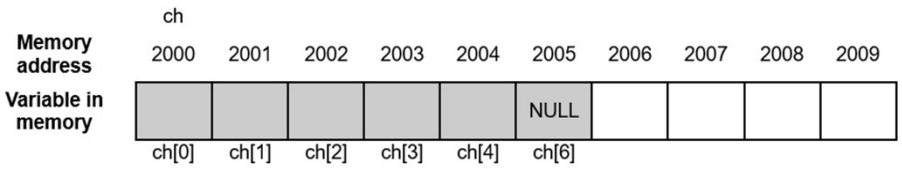
  <tên biến>[chỉ mục]

```
int a[5], t;
/* assign 0 to the 3rd element of the array a */
a[2] = 1;
/* get value of the 3rd element in the array a */
t = a[2];
```

Chuỗi kí tự (1)

- Trong ngôn ngữ C, **chuỗi kí tự** là một **mảng một chiều kiểu char** với **phần tử cuối cùng là NULL** (bằng 0).
- Do đó, khi khai báo một mảng kiểu char để lưu trữ một chuỗi kí tự, ta **cần phải khai báo kích thước của mảng lớn hơn độ dài của chuỗi 1 kí tự**.
- Ví dụ, để khai báo một mảng kiểu char để lưu trữ một chuỗi 5-kí tự, phải viết:

```
char ch[6];
```



Chuỗi kí tự (2)

- **Hằng chuỗi** là một dãy các kí tự được đặt trong cặp dấu **“”.** Ví dụ: **“Hello world”**.
  - Không cần thiết thêm giá trị NULL vào cuối một hằng chuỗi, trình biên dịch sẽ tự thực hiện.
- **string.h** cung cấp các hàm tiện ích để thao tác với chuỗi kí tự
  - strcopy(s1, s2) - sao chép chuỗi s2 vào s1
  - strcat(s1, s2) - nối chuỗi s2 vào cuối chuỗi s1
  - strlen(s1) - trả về độ dài của chuỗi s1
  - strcmp(s1, s2) - trả về giá trị bằng 0 nếu s1 ≡ s2; nhỏ hơn 0 nếu s1 < s2; lớn hơn 0 nếu s1 > s2
  - strchr(s1, c) - trả về con trỏ trỏ tới vị trí xuất hiện lần đầu của kí tự c trong chuỗi s1
  - strstr(s1, s2) - trả về con trỏ trỏ tới vị trí xuất hiện lần đầu của chuỗi s2 trong chuỗi s1

Mảng hai chiều - Khai báo

- **Mảng hai chiều** là một mảng một chiều với mỗi phần tử là một mảng một chiều có cùng kích thước và cùng kiểu dữ liệu.

	Col 1	Col 2	Col 3	Col 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

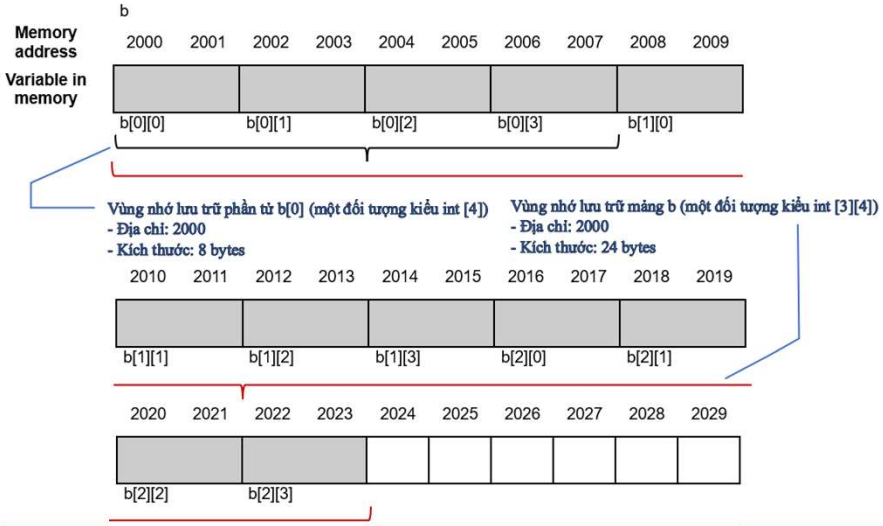
- Khai báo mảng hai chiều

  <kiểu dữ liệu> <tên biến>[kích thước 1][kích thước 2]

```
/* declare a 2D integer array of size 3x4
   (3 rows, 4 columns)*/
int b[3][4];
```


Mảng hai chiều - Tổ chức lưu trữ

- Tổ chức lưu trữ mảng hai chiều trong bộ nhớ



Mảng hai chiều - Truy cập phần tử

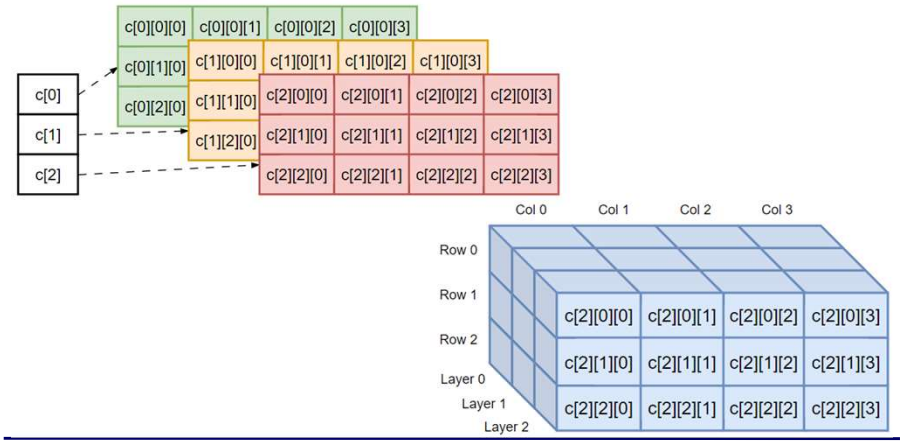
- Các phần tử của mảng hai chiều được đánh chỉ mục theo hàng và cột.
  - Hàng và cột đầu tiên có chỉ mục là 0.
  - Mảng có kích thước m x n thì hàng cuối có chỉ mục là (m - 1) và cột cuối có chỉ mục là (n - 1).
- Dùng phép toán [ ] với hai chỉ mục hàng và cột

  <tên biến>[chỉ mục hàng][chỉ mục cột]

```
int b[3][4], t;
/* assign 0 to the element at the 2nd row and
   the 3th column in the array b */
b[1][2] = 1;
/* get value of the element at the 2nd row and
   the 3th column in the array b */
t = b[1][2];
```


Mảng nhiều chiều

- **Mảng ba chiều** là một mảng một chiều với mỗi phần tử là một mảng hai chiều có cùng kích thước và cùng kiểu dữ liệu.



Mảng nhiều chiều - Khai báo

- Khai báo **mảng ba chiều**

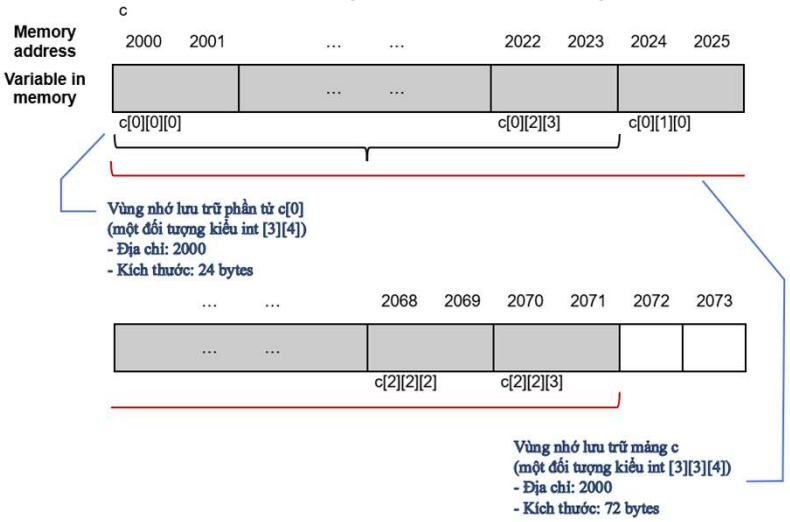
  <kiểu dữ liệu> <tên biến>[kích thước 1][kích thước 2] [kích thước 3];

- Các phần tử của mảng ba chiều được đánh chỉ mục theo tầng, hàng và cột.
  - Tầng, hàng và cột đầu tiên có chỉ mục là 0.
  - Mảng có kích thước (*l x m x n*) thì tầng cuối có chỉ mục là *(l - 1)*, hàng cuối có chỉ mục là *(m - 1)* và cột cuối có chỉ mục là *(n - 1)*.
- Dùng phép toán [ ] với ba chỉ mục tầng, hàng và cột

  <tên biến>[chỉ mục tầng][chỉ mục hàng][chỉ mục cột]

Mảng nhiều chiều - Tổ chức lưu trữ

- Tổ chức lưu trữ mảng ba chiều trong bộ nhớ



Mảng nhiều chiều - Truy cập phần tử

- Dạng khai báo cho **mảng nhiều hơn ba chiều** là ta thêm kích thước tương ứng với chiều và đặt trong cặp dấu **[ ]**.

  <kiểu dữ liệu> <tên biến>[kích thước 1]
  [kích thước 2][kích thước 3]
  …[kích thước n];

- Các mảng nhiều hơn ba chiều hiếm khi được sử dụng, do
  - Yêu cầu bộ nhớ lớn.
  - Thời gian truy cập các phần tử chậm hơn.

Khởi tạo mảng (1)

- Khởi tạo giá trị cho các phần tử của mảng khi khai báo mảng bằng cách liệt kê các giá trị khởi tạo đặt trong dấu **{ }**.
- Khởi tạo với kích thước xác định

```
/* Initialize variables
   - iNum1 is a 5-elements array of type int
   - iNum2, iNum3 are 2x3 arrays of type float. */
int iNum1[5] = {1, -2, 0, 5, 7};
int iNum2[2][3] = {{1, 2, 3}, {4, 5, 6}};
int iNum3[2][3] = {1, 2, 3, 4, 5, 6};
```

Khởi tạo mảng (2)

- Khởi tạo với kích thước xác định nhưng thiếu giá trị, các phần tử khuyết giá trị sẽ được gán bằng 0.

```
/* Initialize a 5-elements array with a list of 3 values */
float fNum1[5] = {1.1, 2.2, 3.3};
```

```
1.100000    2.200000    3.300000    0.000000    0.000000
```

```
/* Initialize a 2x3 array of type float */
float fNum2[2][3] = {{1.0, 2.0}, {3.0, 4.0}};
```

```
1.000000    2.000000    0.000000
3.000000    4.000000    0.000000
```

```
/* Initialize a 2x3 array of type float */
float fNum3[2][3] = {1.0, 2.0, 3.0, 4.0};
```

```
1.000000    2.000000    3.000000
4.000000    0.000000    0.000000
```


Khởi tạo mảng (3)

- Khởi tạo với kích thước không xác định. Trường hợp mảng nhiều chiều, ngoại trừ kích thước thứ nhất (ngoài cùng bên trái), các kích thước còn lại phải được xác định.

```
/* Initialize iArr1 using a list of 5 integers */
int iArr1[] = {1, 2, 3, 4, 5};
```

```
Size of the array: 5
    1    2    3    4    5
```

```
/* Initialize two [ ]x3 arrays of type int */
int iArr2[][3] = {{1, 2},  {4, 5}};
```

```
Size of array: 2 x 3
    1    2    0
    4    5    0
```

```
int iArr3[][3] = {1, 2, 3, 4};
```

```
Size of array: 2 x 3
    1    2    3
    4    0    0
```

Khởi tạo mảng (4)

- Khởi tạo **mảng một chiều *kiểu char*** thì giá trị của phần tử cuối cùng phải là kí tự null (bằng **'\0'** hoặc **0**). Do đó kích thước của mảng phải lớn hơn số kí tự thực sự có ý nghĩa sử dụng 1 đơn vị.

```
/* Initialize the variable cArr1 of the array of type char */
char cArr1[6] = {'H', 'e', 'l', 'l', 'o', 0};
```

Nếu khởi tạo bằng một hằng chuỗi thì không cần xác định kích thước.

```
/* Initialize the variable cArr2 of using a string constant */
char cArr2[] = "Hello";
```


Lỗi với mảng và chuỗi (1)

- **Lỗi thiếu kí tự kết thúc chuỗi**

```
/* Wrong - Initialize the variable cArr1 using a list of 5 characters
   without the terminating null character '\0' */
char cArr1[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                  'a', 'b', 'c', 'd', 'e', 'f'};
printf("Size of cArr1: %zd (bytes);\tcArr1 = %s\n", sizeof(cArr1), cArr1);

char cArr2[16], cArr3[16];
/* Wrong - copy full characters to cArr2, cArr3
   without leaving room for the terminating null character '\0' */
strncpy(cArr2, cArr1, sizeof(cArr2));
strncpy(cArr3, cArr1, sizeof(cArr3));
printf("Size of cArr2: %zd (bytes);\tcArr2 = %s\n", sizeof(cArr2), cArr2);
printf("Size of cArr3: %zd (bytes);\tcArr3 = %s\n", sizeof(cArr3), cArr3);

char cArr4[16];
/* copy the value of cArr2 to cArr4 */
strcpy(cArr4, cArr2);
printf("Size of cArr4: %zd (bytes);\tcArr4 = %s\n", sizeof(cArr4), cArr4);
```

Lỗi với mảng và chuỗi (2)

```
Size of cArr1: 16 (bytes);      cArr1 = 0123456789abcdef♦♦♦♦♦♦뿐♦m♦♦
Size of cArr2: 16 (bytes);      cArr2 = 0123456789abcdef0123456789abcdef
Size of cArr3: 16 (bytes);      cArr3 = 0123456789abcdef
Press any key to continue . . .
```

- **Lỗi vượt giới hạn chỉ mục mảng**
  - Trình biên dịch không tự động kiểm tra giới hạn chỉ mục của mảng. Chương trình vẫn được biên dịch và thực thi bình thường.
  - Có thể gây ra sự lưu trữ đè lên vùng nhớ ngoài phạm vi của mảng có thể đè lên dữ liệu của biến khác hoặc mã đối tượng của chương trình.

```
int count[10], i;
/* this causes the array count to be overrun */
for (i = 0; i < 100; i++) count[i] = i;
```


Tên biến mảng và con trỏ

- Ngôn ngữ C qui định ***<tên biến>*** mảng là **địa chỉ bộ nhớ** của **phần tử đầu tiên**
  - Được xác định khi lệnh khai báo được thi hành.
  - **Không thể thay đổi** khi mảng còn tồn tại.
  - ***<tên biến>*** mảng là một **hằng con trỏ**.
- **Biểu thức địa chỉ** và **giá trị tại địa chỉ** của **các thành phần** của mảng một chiều:

	int a[N] - mảng một chiều a (đối tượng kiểu int [N])	**a[0] - phần tử đầu tiên (đối tượng kiểu int)**	**a[i] - phần tử thứ i (đối tượng kiểu int)**
Biểu thức địa chỉ	&a	&a[0] ≅ a	&a[i] ≅ &a[0] + i ≅ a + i
Truy cập giá trị	*(&a)	*(&a[0]) ≅ *a	*(&a[i]) ≅ *(&a[0] + i) ≅ *(a + i)
Kết quả	Địa chỉ của a[0] ≅ &a[0]	a[0]	a[i]

Con trỏ trỏ tới mảng (1)

- Có thể tạo **con trỏ trỏ tới mảng** hoặc **con trỏ trỏ tới phần tử đầu tiên của mảng**.
- Khai báo **con trỏ trỏ tới mảng một chiều**

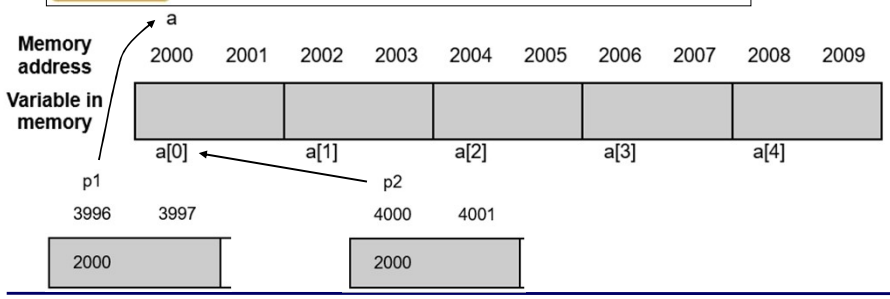
  <kiểu dữ liệu> (*<tên biến>)[kích thước];
- Tạo **con trỏ trỏ tới mảng một chiều** bằng cách gán địa chỉ của biến mảng cho con trỏ **(*)[ ]** cùng kiểu cơ sở với biến mảng.

```
int a[5];
int (*p1)[5]; // declare a pointer of type int [5]
p1 = &a;
```


Con trỏ trỏ tới mảng (2)

- Tạo một **con trỏ trỏ tới phần tử đầu tiên của mảng một chiều** bằng cách gán ***<tên biến>*** mảng cho một con trỏ * cùng kiểu cơ sở với biến mảng.

```
int a[5];
int *p2; // declare a pointer of type int
p2 = a;
```



Con trỏ chỉ mục

- **Một con trỏ trỏ tới phần tử đầu tiên** của mảng dùng để truy cập các phần tử của mảng thông qua chỉ mục của phần tử được gọi là **con trỏ chỉ mục**.
- **Con trỏ chỉ mục** và **biểu thức truy xuất** phần tử của mảng được xác định theo bảng sau:

Con trỏ chỉ mục	<kiểu> a[N]	<kiểu> a[M][N]	<kiểu> a[L][M][N]
<kiểu> *p	- Tạo con trỏ: p = a - Chỉ mục: p[i] - Gián tiếp: *(p + i)	- Tạo con trỏ: p = (<kiểu> *)a - Chỉ mục: p[i * N + j] - Gián tiếp: *(p + i * N + j)	- Tạo con trỏ: p = (<kiểu> *)a - Chỉ mục: p[i * M * N + j * N + k] - Gián tiếp: *(p + i * M * N + j * N + k)
<kiểu> (*p)[N]		- Tạo con trỏ: p = a - Chỉ mục: p[i][j] - Gián tiếp: *(*(p + i) + j)	- Tạo con trỏ: p = (<kiểu> (*)[N])a - Chỉ mục: p[i * M + j][k] - Gián tiếp: *(*(p + i * M + j) + k)
<kiểu> (*p)[M][N]			- Tạo con trỏ: p = a - Chỉ mục: p[i][j][k] - Gián tiếp: *(*(*(p + i) + j) + k)

Con trỏ chỉ mục - Mảng một chiều

```
int a[5], *p;

/* indexing pointer p points to the element a[0] */
p = a;

/* assign a value to a[i] by p with 1 indexing operator */
p[i] = 1;

/* get the value a[i] through p */
sum += p[i];
```

Con trỏ chỉ mục - Mảng hai chiều

```
int b[3][4], (*p1)[4], *p2;

/* indexing pointer p1 points to the element b[0] */
p1 = b;
/* indexing pointer p2 points to the element b[0][0] */
p2 = (int *)b;

/* assign a value to b[i][j] by p1 with 2 indexing operators
                              or p2 with 1 indexing operator */
p1[i][j] = 1;
p2[i * 4 + j] = 2;

/* get the value b[i][j] through p1 or p2 */
sum += (p1[i][j] + p2[i * 4 + j]);
```


Con trỏ chỉ mục - Mảng ba chiều

```
int c[2][3][4], (*p1)[3][4], (*p2)[4], *p3;

/* indexing pointer p1 points to the element c[0] */
p1 = c;
/* indexing pointer p2 points to the element c[0][0] */
p2 = (int (*)[4])c;
/* indexing pointer p3 points to the element c[0][0][0] */
p3 = (int *)c;

/* assign a value to c[i][j][k] by p1 with 3 indexing operators
                                 or p2 with 2 indexing operator
                                 or p3 with 1 indexing operator */
p1[i][j][k] = 1;
p2[i * 3 + j][k] = 2;
p3[(i * 3 * 4) + (j * 4) + k] = 3;

/* get the value c[i][j][k] through p1, p2 or p3 */
sum += (p1[i][j][k] + p2[i * 3 + j][k] + p3[(i * 3 * 4) + (j * 4) + k]);
```

Mảng các chuỗi kí tự

- **Mảng một chiều các con trỏ kiểu char** hoặc **mảng hai chiều kiểu char** được dùng để lưu trữ mảng các chuỗi kí tự.

```
char *errs[4];
errs[0] = "Cannot Open File";
errs[1] = "Read Error";
errs[2] = "Write Error";
errs[3] = "Media Failure";

char msgs[4][21] = {"Cannot Open File",
                    "Read Error",
                    "Write Error",
                    "Media Failure"};
```