Chương IV

MẢNG VÀ CON TRỎ

4.1. Mảng

Mảng là một tập hợp hữu hạn các phần tử có cùng kiểu dữ liệu

Các phần tử của mảng được lưu trữ trong một khối gồm các ô nhớ liên tục nhau, có cùng tên (cũng là tên của mảng) nhưng phân biệt với nhau ở chỉ số. Chỉ số này xác định vị trí của nó trong mảng.

Mảng

Mảng được chia ra làm 2 loại: mảng một chiều và mảng nhiều chiều (mảng 2 chiều trở lên).

4.1.1 Mảng một chiều

A) Khai báo mảng:

Kiểu_dữ_liệu Tên_mảng **[**số_phần_tử**] ;**

Lưu ý: số_phần_tử (kích thước của mảng) phải được xác định ngay tại thời điểm khai báo và phải là hằng số.

Ví dụ:

```
int A[100]; //khai báo mảng số nguyên A gồm 100 phần tử.
```

4.1.1 Mảng một chiều

B) Truy xuất dữ liệu trong mảng:

Truy xuất các phần tử của mảng theo cú pháp:

Tên_mảng[chỉ_số]

Trong đó chỉ_số là số nguyên bắt đầu từ 0 đến n-1, với n là kích thước của mảng.

Ví dụ: Mảng A có 4 phần tử, phần tử thứ 2 là A[1]= - 7

Chỉ số	0	1	2	3
Mảng A	4	-7	3	2
Phần tử	A[0]	A[1]	A[2]	A[3]

4.1.1 Mảng một chiều

C) Khởi tạo giá trị cho mảng một chiều khi khai báo:

Kiểu_dữ_liệu Tên_mảng[số_phần_tử] = { giá_trị_1, giá_trị_2,…};

Trong đó, giá_trị_1, giá_trị_2,… là các giá trị tương ứng được khởi tạo cho từng phần tử của mảng theo đúng thứ tự. Số lượng các giá trị không được vượt quá kích thước của mảng.

Ví dụ 1: Khởi tạo giá trị cho tất cả các phần tử của mảng:

```
int A[5]={1,5,-8,7,0};
```

Ví dụ 2: Khởi tạo vài giá trị đầu của mảng, các phần tử sau mặc định bằng 0:

```
int A[5]={2,4,1};
```

Ví dụ 3: Khởi tạo giá trị 0 cho tất cả các phần tử:

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

Ví dụ 4: Khởi tạo mảng mà không khai báo kích thước:

```
int A[]={1,5,-8};
```

Khi đó mảng A sẽ có kích thước bằng 3 vì nó có 3 phần tử.

4.1.1 Mảng một chiều

D) Dùng mảng một chiều làm tham số hàm:

Việc truyền một phần tử đơn lẻ vào hàm thì hoàn toàn giống với truyền một biến vào hàm.

Đối với trường hợp muốn truyền toàn bộ mảng cho hàm thì ta cần phải khai báo mảng là tham số hình thước của hàm.

Ví dụ: int Max**(int** A**[12], kich_thuoc);**

Lưu ý:

- Có thể không cần ghi kích thước mảng trong phần khai báo tham số hàm, ví dụ: int Max**(int** A**[], kich_thuoc);**

- Khi gọi hàm và truyền mảng thì ta chỉ ghi tên mảng mà không có cặp ngoặc **[]** ,ví dụ**:** Gọi hàm Max ở ví dụ trên, truyền tham số là mảng int B**[**12**]** , kích_thước là 12 như sau: Max**(** B**,** 12**) ;**

Ví dụ: Dùng mảng một chiều làm tham số hàm

```
1   #include<stdio.h>
2   int Max(int A[], int kich_thuoc)
3   {
4       int max=A[0];
5       for(int i=1;i<kich_thuoc;i++)
6       if(max<A[i])max=A[i];
7       return max;
8   }
9   main()
10  {
11      int B[12]={1,2,3,4,5,3,2,8,0,-5,-3,16};
12      printf("Phan tu lon nhat cua mang B la: %d",Max(B,12));
13  }
```

Kết quả khi chạy chương trình :

```
Phan tu lon nhat cua mang B la: 16
```

4.1.2 Mảng hai chiều

Dạng đơn giản nhất và thông dụng nhất của mảng nhiều chiều là mảng hai chiều. Một mảng hai chiều là mảng chứa các mảng một chiều.

Để cho dễ hiểu người ta thường biểu diễn mảng hai chiều dưới dạng một ma trận gồm các hàng và các cột. Tuy nhiên, về mặt vật lý thì các phần tử của mảng hai chiều vẫn được lưu trong một khối nhớ liên tục nhau.

A) Khai báo mảng hai chiều:

```
Kiểu_dữ_liệu  Tên_mảng[số_hàng][số_cột] ;
```

Khi đó kích thước mảng sẽ là tích (số_hàng*số_cột)

Ví dụ:

float A[3][4]; */*Mảng số thực A gồm 12 phần tử được chia thành 3 hàng, 4 cột*/*

4.1.2 Mảng hai chiều

B) Truy xuất các phần tử của mảng:

Mỗi phần tử của mảng có dạng:

Tên_mảng[chỉ_số_hàng][chỉ_số_cột]

Trong đó, chỉ_số_hàng có giá trị từ 0 đến (số_hàng -1)và chỉ_số_cột có giá trị từ 0 đến (số_cột - 1).

Ví dụ: Mảng int A[3][2] được minh họa như hình dưới:

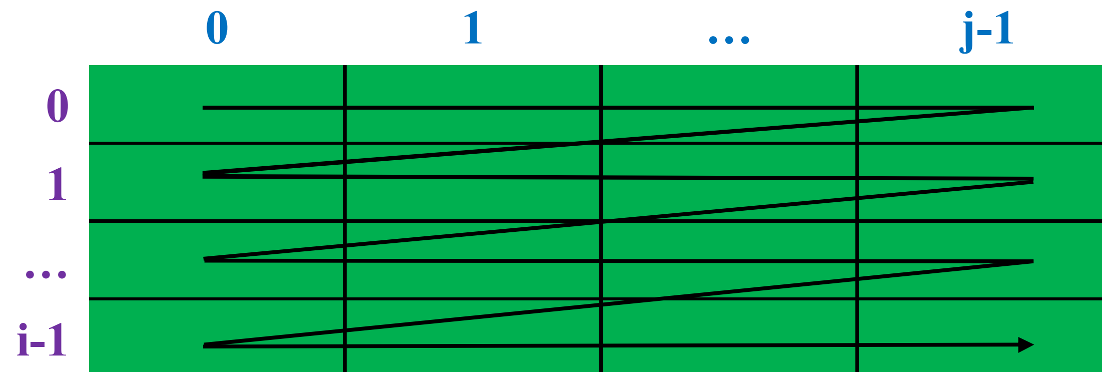
Chỉ số	**0, 0**	**0, 1**	**1, 0**	**1, 1**	**2, 0**	**2, 1**
Mảng A						
Phần tử	**[0][0]**	**[0][1]**	**[1][0]**	**[1][1]**	**[2][0]**	**[2][1]**

	0	**1**
0	**6**	**2**
1	**-9**	**0**
2	**1**	**5**

4.1.2 Mảng hai chiều

C) Khởi tạo giá trị cho mảng hai chiều khi khai báo:

Thứ tự các phần tử của mảng A[i][j] sẽ được khởi tạo theo thứ tự như sau:



Ví dụ 1: Khởi tạo mảng số nguyên A[3][2] giống mảng 1 chiều:

int A[3][2] = {2, 7, 9, 0, 4, -3};

4.1.2 Mảng hai chiều

Ví dụ 2: Khởi tạo theo từng dòng (hàng), các phần tử của mỗi dòng sẽ nằm trong một cặp dấu {}:

```
int A[3][2]={{2, 7}, {9, 0}, {4, 3}};
```

Ví dụ 3: Khởi tạo giá trị cho vài phần tử ở đầu mảng, các phần tử còn lại sẽ tự động nhận giá trị 0:

```
int A[3][2]={{2, 7}, {9}, {4, 3}};
```

Ví dụ 4: Khởi tạo giá trị 0 cho tất cả các phần tử của mảng:

```
int A[3][2]={0};
```

Ví dụ 5: Khởi tạo giá trị cho mảng mà không bao gồm khai báo chiều thứ nhất hay số hàng (chiều thứ hai hay số cột bắt buộc phải có):

```
int A[][2]={{2, 7}, {9, 0}, {4, 3}};
```

4.1.2 Mảng hai chiều

D) Mảng hai chiều làm tham số cho hàm:

Tương tự như mảng một chiều, mảng hai chiều cũng có thể được dùng làm tham số cho hàm.

Khi khai báo một tham số là mảng hai chiều, cần phải chỉ rõ số phần tử của các chiều, tuy nhiên số phần tử của chiều đầu tiên có thể vắng.

Ví dụ về mảng hai chiều

```
 1  #include<stdio.h>
 2  void xuat_mang(int a[3][3])//Ham xuat mang
 3  {
 4      for(int i=0;i<3;i++)//Chi so hang
 5      {
 6          for(int j=0;j<3;j++)//Chi so cot
 7          printf("%-4d",a[i][j]);//Du lieu can le trai
 8          printf("\n");
 9      }
10  }
11  main()
12  {
13      int A[3][3] ={1,2,3,4,5,6,7,8,9};
14      printf("Mang A la: \n");
15      xuat_mang(A);/*Goi ham xuat_mang*/
16  }
```

Kết quả chạy chương trình:

```
Mang A la:
1   2   3
4   5   6
7   8   9
```

A) Khái niệm

Con trỏ là một biến lưu trữ địa chỉ của một biến khác. Nghĩa là, giá trị của con trỏ là địa chỉ của một ô nhớ trong bộ nhớ.

Con trỏ thường được sử dụng trong những trường hợp:

- Được sử dụng bên trong hàm để truy xuất đến giá trị của các biến nằm bên ngoài hàm.

- Truyền mảng và chuỗi từ một hàm đến một hàm khác.

- Cần cấp phát vùng nhớ động giúp quản lí và sử dụng bộ nhớ một cách hiệu quả.

- Xây dựng các cấu trúc dữ liệu như cây, danh sách liên kết,…

B) Khai báo và sử dụng con trỏ

Cú pháp khai báo:

Kiểu_dữ_liệu *Tên_con_trỏ ;

Trong đó, **Kiểu_dữ_liệu** là kiểu của biến mà con trỏ muốn trỏ đến. Dấu '*' là kí hiệu con trỏ chỉ áp dụng duy nhất cho biến nằm ngay sau nó.

Trước khi sử dụng con trỏ chúng ta phải chỉ rõ địa chỉ của ô nhớ mà nó trỏ đến bằng cách gán như sau:

Tên_con_trỏ = &tên_biến **;**

Với '**&**' là toán tử lấy địa chỉ của biến được đặt trước tên biến. Ngoài ra, có thể khởi tạo cho con trỏ bằng giá trị 0 (hay NULL). Một con trỏ có giá trị NULL không trỏ tới ô nhớ nào cả, nó khác với con trỏ chưa khởi tạo.

4.2 Con trỏ

Lưu ý:

- Địa chỉ của một ô nhớ là một số nguyên không dấu, nên kiểu dữ liệu của biến con trỏ sẽ là kiểu số nguyên

- Con trỏ được sử dụng trong hai trường hợp: sử dụng chính giá trị của con trỏ và sử dụng giá trị của ô nhớ mà con trỏ đang trỏ tới.

- Biến con trỏ có thể được sử dụng trong các biểu thức giống như các biến thông thường.

- Trong trường hợp muốn truy xuất đến giá trị tại ô nhớ mà con trỏ đang trỏ tới, ta sử dụng toán tử '*' đặt trước biến con trỏ(trừ trường hợp con trỏ có giá trị NULL để tránh lỗi khi chạy chương trình).

C) Các phép toán số học trên con trỏ

C chỉ cho phép một số phép toán số học được thực hiện trên con trỏ gồm: +, -, ++,--, +=, -=.

Bên cạnh các phép toán số học, chúng ta còn có thể thực hiện các phép so sánh giữa hai con trỏ như: ==, !=, >, >=, <, <=

D) Cấp phát vùng nhớ động cho con trỏ

- Bằng các hàm trong *<alloc.h>* hay *<stdlib.h>*

- void ***malloc(S);** Cấp phát vùng nhớ có kích thước S bytes và trả về địa chỉ của vùng nhớ đó. *Nếu không đủ bộ nhớ hàm trả về giá trị NULL.*

- void ***calloc(** N**,** S **);** Cấp phát mảng động có N phần tử, mỗi phần tử có kích thước S bytes và trả về địa chỉ của vùng nhớ đó.

- void ***realloc(**void *ptr**,** S**);** Cấp lại vùng nhớ với kích thước S bytes cho biến con trỏ ptr, đồng thời chép dữ liệu vào vùng nhớ mới.

- Bằng toán tử new :

BiếnConTrỏ **=** **new** KiểuDL**[**Số_phần_tử**];**

4.2 Con trỏ

Ví dụ: int *p1, *p2;

- Cấp động 1 biến kiểu int :

```
p1 = (int *) malloc( sizeof(int) );
p1 = new int;
```

- Cấp động 1 mảng 10 phần tử:

```
p2 = (int*) calloc(10, sizeof(int) );
p2 = new int[10];
```

4.2 Con trỏ

E) Xóa biến động

Giải phóng vùng nhớ động đã cấp phát cho biến con trỏ:

- Dùng hàm: **free (**Biếncontrỏ**);**
- Dùng toán tử: **delete** Biếncontrỏ**;**

Ví dụ:

```
free(p1);
delete p1;
```

Ví dụ 1: Minh họa về con trỏ

```
1  #include <stdio.h>
2  main()
3  {
4       int x = 7;
5       int *px = &x;//Gan dia chi bien x cho con tro px
6       printf("Dia chi cua x: px = %u, &x = %u \n", px, &x);
7       printf("Gia tri ban dau cua x: ");
8       printf("*p = %d, x = %d\n",*px, x);
9       *px = 12; /*Thay doi gia tri cua x thong qua bien px*/
10      printf("Gia tri cua x sau khi thay doi: ");
11      printf("*p = %d, x = %d\n",*px,x);
12 }
```

Kết quả khi chạy chương trình:

```
Dia chi cua x: px = 2686744, &x = 2686744
Gia tri ban dau cua x: *p = 7, x = 7
Gia tri cua x sau khi thay doi: *p = 12, x = 12
```

Ví dụ 2: Sử dụng biến con trỏ để hàm trả về nhiều giá trị

```
 1 #include <stdio.h>
 2 void init(int *px, int *py);
 3 main()
 4 {
 5     int x=9, y=10;
 6     init(&x,&y);
 7     //truyen dia chi 2 bien x,y cho ham init
 8     printf("x = %d, y = %d\n", x, y);
 9 }
10 void init(int *px, int *py)
11 {
12     *px = 3; //gan 3 cho noi dung cua px
13     *py = 5; //gan 5 cho noi dung cua py
14 }
```

Kết quả khi chạy chương trình: Hàm init trả về 2 giá trị cho x và y

---

```
x = 3, y = 5
```

4.3 Con trỏ và mảng

Mảng và con trỏ luôn có mối quan hệ mật thiết với nhau. Tên mảng chứa địa chỉ của phần tử đầu tiên của mảng, tức là mảng làm việc giống như con trỏ luôn trỏ đến phần tử đầu tiên. Vì thế, *phép gán biến con trỏ cho một biến mảng là hoàn toàn hợp lệ*. Tuy nhiên biến con trỏ có thể được gán một địa chỉ khác, tức là giá trị của nó có thể thay đổi. Trong khi đó, biến mảng luôn luôn trỏ đến một vùng nhớ cố định do đó *phép gán biến mảng cho biến con trỏ là không hợp lệ*.

Ví dụ : Cộng hằng số vào mảng

```
1  #include <stdio.h>
2  #define SIZE 4
3  void add(int *ptr, int num, int a);
4  main()
5  {
6      int array[] = {2, 5, 6, 9};
7      int i, x = 10;
8      add(array, SIZE, x);
9      for (i = 0; i < SIZE; i++)
10     printf("%d ", *(array + i));
11 }
12 void add(int *ptr, int num, int a)
13 {
14     for (int j = 0; j < num; j++)
15     *(ptr) = *(ptr++) + a;
16 }
```

Kết quả khi chạy chương trình:

```
12 15 16 19
```

Bài tập luyện tập

Bài 1: Viết chương trình nhập giá trị cho dãy số thực gồm n phần tử (n<=100). Tìm và in ra chỉ số của các phần tử có giá trị nhỏ nhất trong dãy.

Bài 2: Nhập một dãy n số nguyên (n < 50) và một số nguyên x. Loại khỏi dãy những phần tử có giá trị bằng x.

Bài 3: Tính tổng bình phương các phần tử của dãy số nguyên gồm n phần tử (n<=20).

Bài 4: Sắp xếp một dãy số nguyên gồm n phần tử (n<=40) theo thứ tự tăng dần. Chèn thêm một số nguyên x nhập từ bàn phím vào dãy sao cho dãy vẫn tăng dần. Xuất dãy mới ra màn hình.

Bài tập luyện tập

Bài 5: Viết chương trình nhập dữ liệu cho mảng int A[m][n] với m, n <= 10.

a) Xuất mảng A ra màn hình.

b) Tìm giá trị lớn nhất trong A.

c) Dòng nào của A có tổng các phần tử là lớn nhất.

d) Sắp xếp từng dòng của A theo thứ tự tăng dần.

e) Sắp xếp mảng A theo thứ tự giảm dần.

Bài tập luyện tập

Bài 6: Cho hai ma trận số nguyên A và B có kích thước m hàng, n cột (m, n<=100). Viết chương trình:

a) Nhập/xuất dữ liệu cho hai ma trận.

b) Tính tổng của hai ma trận.

c) Tìm phần tử lớn nhất và nhỏ nhất của ma trận A.

d) Có bao nhiêu phần tử là số âm trong ma trận B.

e) Sắp xếp ma trận A theo thứ tự tăng dần.

f) Xuất ra màn hình ma trận chuyển vị của B

g) Tìm phần tử nhỏ nhất trong tất cả các phần tử của hai ma trận

Bài tập luyện tập

Bài 7: Cho ma trận A vuông cấp n (n<=8) với các phần tử là số nguyên. Viết chương trình theo các yêu cầu sau:

a) Nhập/xuất A .

b) Tính tổng các phần tử nằm ngoài đường chéo chính ?

c) Tìm giá trị lớn nhất trên đường chéo chính ?

d) Có bao nhiêu phần tử là số âm nằm trên đường chéo phụ ?

e) Ma trận A có đối xứng hoặc phản đối xứng hay không ?

f) Tính detA ?

g) Tính rank(A) ?