



# IT001 - NHẬP MÔN LẬP TRÌNH

## CHƯƠNG 7.2: MẢNG HAI CHIỀU VÀ MẢNG NHIỀU CHIỀU

Mảng hai chiều trong C++ là một công cụ mạnh mẽ và linh hoạt, cho phép lập trình viên lưu trữ và quản lý dữ liệu trong dạng lưới hoặc ma trận. Sử dụng mảng hai chiều giúp việc xử lý dữ liệu trở nên trực quan hơn, đặc biệt là trong các ứng dụng liên quan đến bảng biểu, ma trận số.

Khoa Khoa học Máy tính



# NỘI DUNG

- 7.1 Giới thiệu mảng
- 7.2 Khái niệm mảng
- 7.3 Các yếu tố xác định mảng
- 7.4 Mảng một chiều
- 7.5 Mảng hai chiều
- 7.6 Mảng nhiều chiều
- 7.7 Chuỗi C-String



## 7.5 Mảng hai chiều



## 7.5 Mảng hai chiều

7.5.1 Khai báo và khởi tạo mảng 2 chiều

7.5.2 Chỉ số mảng và truy xuất phần tử mảng

7.5.3 Lấy kích thước mảng dùng toán tử sizeof

7.5.4 Lấy địa chỉ các phần tử mảng

7.5.5 Phép Gán dữ liệu kiểu mảng

7.5.6 Một số khái niệm liên quan

7.5.7 Truyền mảng cho hàm và lời gọi hàm

7.5.8 Các tác vụ trên mảng 2 chiều



## 7.5 Mảng hai chiều

### 7.5.1 Khai báo và khởi tạo mảng 2 chiều



# Khai báo mảng 2 chiều

- Cú pháp:

```
<Kiểu_dữ_liệu> <Tên_bien_mảng> [<Số_dòng>] [<Số_cột>];
```

- Trong đó:

- <Kiểu\_dữ\_liệu>: int, float, char, ...
- <Tên\_bien\_mảng>: Đặt tên theo quy tắc đặt tên/định danh
- <Số\_dòng>, <Số\_cột>:
  - Là số lượng tối đa mảng có thể lưu trữ trên dòng và cột
  - Cần xác định ngay khi khai báo
  - Phải là một hằng số



# Khai báo mảng 2 chiều

- Ví dụ:

```
int A[2][4];
```

Kiểu dữ liệu: int

Tên biến mảng: A

Mảng có 2 dòng và 4 cột

	0	1	2	3
0	10	20	30	40
1	50	60	70	80

```
float B[2][2];
```

Kiểu dữ liệu: float

Tên biến mảng: B

Mảng có 2 dòng và 2 cột

	0	1
0	1.5	2.7
1	5.3	6



# Khai báo mảng 2 chiều

- Cú pháp khai báo Không tường minh:

```
typedef <kiểu cơ sở> <tên  
kiểu>[<Số_dòng>][<Số_cột>];  
<tên kiểu> <tên biến>;
```

- Ví dụ:

```
typedef int  
MaTran10x20[10][20];
```

```
MaTran10x20 a, b;  
// Khai báo trên tương đương với khai  
báo:
```



# Khởi tạo mảng 2 chiều

- Khởi tạo giá trị cho mọi phần tử của mảng

```
int A[2][3] = {1, 2, 3, 4, 5, 6};
```

```
int A[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

```
int A[2][3] {1, 2, 3, 4, 5, 6};
```

	0	1	2
0	1	2	3
1	4	5	6

	0	1	2
0	1	2	0
1	4	0	0

	0	1	2
0	0	0	0
1	0	0	0

	0	1	2
0	1	2	3
1	4	5	6

- Khởi tạo giá trị cho một số phần tử đầu mỗi dòng

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

- Khởi tạo giá trị 0 cho mọi phần tử của mảng

```
int C[2][3] = {};
```

- Tự động xác định số lượng dòng

```
int D[][][3] = {1, 2, 3, 4, 5, 6};
```



# Khởi tạo giá trị mảng 2 chiều

- Ví dụ:

```
#include<iostream>
using namespace std;
int main() {
    int A[2][3] = { 1, 2, 3, 4, 5, 6};
    int B[2][3] = { {1, 2, 3}, {4, 5, 6} };
    int C[2][3] = { {1, 2}, {4} };
    for(int i=0; i<2; i++)
        for(int j=0; j<3; j++) cout << A[i] [j] ;
    cout << endl;
    for(int i=0; i<2; i++)
        for(int j=0; j<3; j++) cout << B[i] [j] ;
    cout << endl;
    for(int i=0; i<2; i++)
        for(int j=0; j<3; j++) cout << C[i] [j] ;
    return 0;
}
```

Kết quả thực thi:

```
123456
123456
120400
```



## 7.5 Mảng hai chiều

### 7.5.2 Chỉ số mảng và truy xuất phần tử mảng



## 7.5.2 Chỉ số mảng và truy xuất phần tử mảng

- Chỉ số mảng là một giá trị **số nguyên**.
- Chỉ số trong mảng 2 chiều gồm **chỉ số dòng** và **chỉ số cột**.
  - $0 \leq \text{chỉ số dòng} \leq \text{số dòng của mảng} - 1$
  - $0 \leq \text{chỉ số cột} \leq \text{số cột của mảng} - 1$
- Ví dụ:

`int A[2][3];`

Tên mảng: A

Kiểu dữ liệu của từng phần tử trong mảng: `int`

Số phần tử tối đa trong mảng:  $2 \text{ dòng} * 3 \text{ cột} = 6 \text{ phần tử}$

Các chỉ số được đánh số: Chỉ số Dòng: 0, 1, Chỉ số Cột: 0, 1, 2

	0	1	2
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]



## 7.5.2 Chỉ số mảng và truy xuất phần tử mảng

- Truy xuất phần tử mảng thông qua chỉ số:

[<Tên\_bien\_mảng> [<Số\_dòng>]  
[<Số\_cột>]; \_\_\_\_\_]

- Ví dụ:

```
int A[2][3];
```

Các truy xuất hợp lệ:

A[0][0], A[0][1], A[0][2],  
A[1][0], A[1][1], A[1][2]

Các truy xuất không hợp lệ: A[-1][0], A[1][4], A[2][0]

Giá trị các phần tử mảng:

A[0][0]=10

A[0][1]=20

0	1	2	
10	20	30	0
40	50	60	1

A[1][0]=40

A[1][1]=50

A[1][2]=60



# Ví dụ:

- Ví dụ: Xuất các giá trị của mảng 2 chiều

```
#include<iostream>
using namespace std;
int main() {
    int A[4][3] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int r=4, c=3;
    cout << "Gia tri mang:\n";
    for(int i=0; i<r; i++) {
        for(int j=0; j<c; j++)
            cout << A[i][j] << "\t";
        cout << endl;
    }
    return 0;
}
```

Kết quả thực thi:

Gia tri mang:

1	2	3
4	5	6
7	8	9
10	11	12



## 7.5 Mảng hai chiều

### 7.5.3 Lấy kích thước mảng dùng toán tử sizeof



## 7.5.3 Lấy kích thước mảng dùng toán tử sizeof

- Cú pháp: Cho mảng 2 chiều A có r dòng và c cột
  - `sizeof(A)`: kích thước toàn mảng
  - `sizeof(A[i][j])`: lấy kích thước của phần tử ở dòng i cột j trong mảng với ( $0 \leq i \leq r$  và  $0 \leq j \leq c$ )
  - `sizeof(A[i])`: lấy kích thước của cả dòng thứ i ( $0 \leq i \leq r$ ) trong mảng



## Ví dụ:

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

int main() {
    int A[4][3] = { {1, 2, 3}, {4, 5, 6},
                    {7, 8, 9}, {10, 11, 12} };
    int r=4, c=3;
    cout << "Kich thuoc mang: " << sizeof(A) << endl;
    cout << "Kich thuoc phan tu A[0][0]: " << sizeof(A[0][0])
<< endl;
    for(int i=0; i<r; i++)
        cout << "Kich thuoc dong " << i << ":" <<
    sizeof(A[i]) << endl;
    return 0;
}
```

### Kết quả thực thi:

```
Kich thuoc mang: 48
Kich thuoc phan tu A[0][0]: 4
Kich thuoc dong 0: 12
Kich thuoc dong 1: 12
Kich thuoc dong 2: 12
Kich thuoc dong 3: 12
```



## 7.5 Mảng hai chiều

### 7.5.4 Lấy địa chỉ các phần tử mảng



## 7.5.4 Lấy địa chỉ các phần tử mảng

- Cú pháp:

```
& <Tên_bien_mảng>[<Chỉ_số  
dòng>] [<Chỉ_số_cột>];
```

- Ví dụ:

```
int A[2][3];
```

A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
0x14	0x18	0x22	0x26	0x30	0x34

Địa chỉ các phần tử mảng 2 chiều:

10	20	30	40	50	60
----	----	----	----	----	----

Địa chỉ các phần tử trên dòng thứ 0:

`&A[0][0]=A[0], &A[0][1], &A[0][2]`

0	1	2	
10	20	30	0

40	50	60	1
----	----	----	---

Địa chỉ các phần tử trên dòng thứ 1:

`&A[1][0]=A[1], &A[1][1], &A[1][2]`

Lưu ý: những giá trị 0x14, 0x18, 0x22, 0x26, 0x30, 0x34 là các địa chỉ giả định



# Ví dụ: Xuất địa chỉ các phần tử mảng 2 chiều

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

int main() {
    int A[4][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12};
    int r=4, c=3;
    cout << "Dia chi cac phan tu mang:\n";
    for(int i=0; i<r; i++) {
        for(int j=0; j<c; j++)
            cout << &A[i][j] << " ";
        cout << endl;
    }
    cout << "\nDia chi moi dong:\n";
    for(int i=0; i<r; i++)
        cout << A[i] << endl;

    cout << "\nDia chi mang:\n";
    cout << &A << endl;
    return 0;
}
```

	1	2
0	1	2
1	4	5
2	7	8
3	10	11
	12	

## Kết quả thực thi:

Dia chi cac phan tu mang:  
0x61fdd0 0x61fdd4 0x61fdd8  
0x61fddc 0x61fde0 0x61fde4  
0x61fde8 0x61fdec 0x61fdf0  
0x61fdf4 0x61fdf8 0x61fdfc

Dia chi moi dong:  
0x61fdd0  
0x61fddc  
0x61fde8  
0x61fdf4

Dia chi mang:  
0x61fdd0

# Ví dụ: Xuất địa chỉ các phần tử mảng 2 chiều (tt)

```
#include<iostream>
using namespace std;
int main() {
    int A[4][3] { 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12};
    int r=4, c=3;
    cout << "Dia chi cac phan tu mang:\n";
    for(int i=0; i<r; i++) {
        for(int j=0; j<c; j++)
            cout << &A[i][j] << " ";
        cout << endl;
    }
    cout << endl;
    cout << "A[0]+1: " << A[0]+1 << endl;
    cout << "A[1]+1: " << A[1]+1 << endl;
    cout << "A[2]+1: " << A[2]+1 << endl;
    cout << "A[3]+1: " << A[3]+1 << endl;
    return 0;
}
```

0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

Kết quả thực thi:

Địa chỉ các phần tử mảng:  
0x61fde0 0x61fde4 0x61fde8  
0x61fdec 0x61fdf0 0x61fdf4  
0x61fdf8 0x61fdfc 0x61fe00  
0x61fe04 0x61fe08 0x61fe0c

A[0]+1: 0x61fde4  
A[1]+1: 0x61fdf0  
A[2]+1: 0x61fdfc  
A[3]+1: 0x61fe08



## 7.5 Mảng hai chiều

### 7.5.5 Phép Gán dữ liệu kiểu mảng



## 7.5.5 Phép Gán dữ liệu kiểu mảng

- Không được sử dụng phép gán thông thường mà phải gán trực tiếp giữa các phần tử.

~~<Biến\_mảng\_đích> = <biến\_mảng\_nguồn>; → sai~~

~~<Tên\_biến\_mảng>[<Chỉ\_số\_dòng\_i>][<Chỉ\_số\_cột\_j>] = <Giá\_trị>; → đúng~~

- Ví dụ:

```
int a[3][3]={{ 1, 2 },
              2, 0,
              { 6, 7, 8 },
              7, 8,
              { 11 }};
              // row 1 = 1,
              // row 2 = 6,
              // row 3 = 11,
              0, 0
int b[3][3];
b = a;// Sai
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
```



## 7.5 Mảng hai chiều

### 7.5.6 Một số khái niệm liên quan



## 7.5.6 Một số khái niệm liên quan

- Cho ma trận A gồm 3 dòng x 3 cột như hình dưới đây:

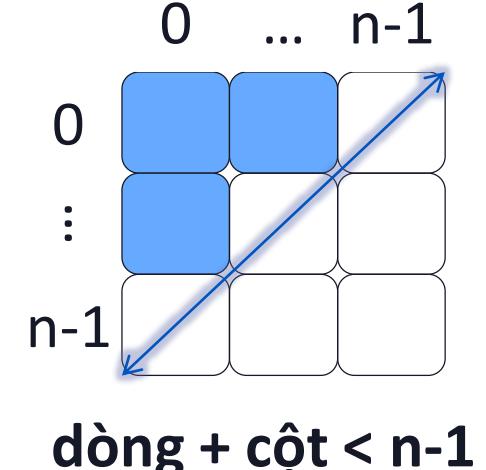
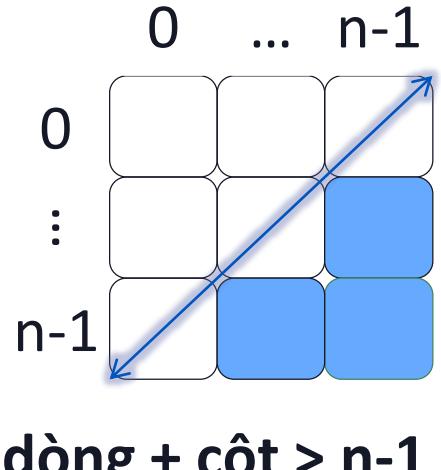
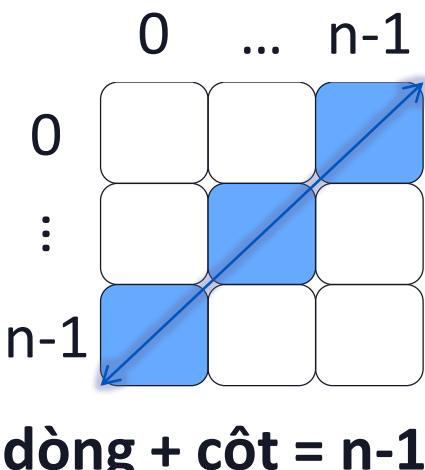
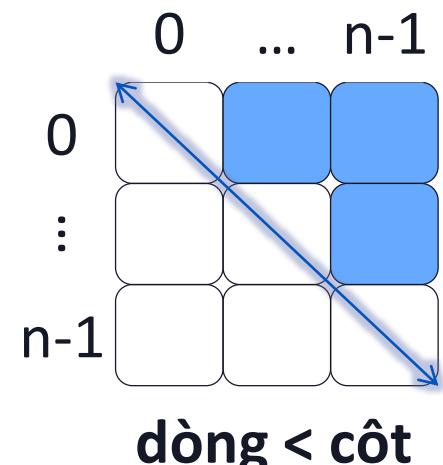
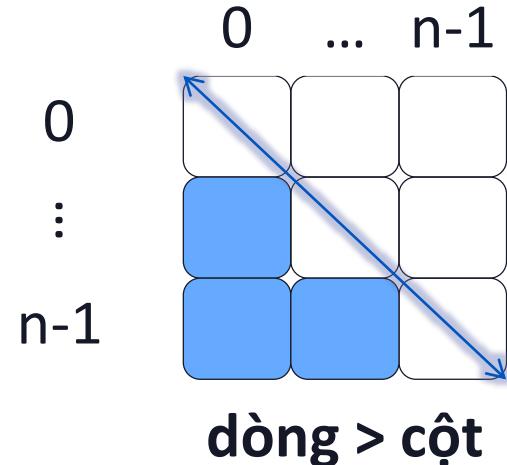
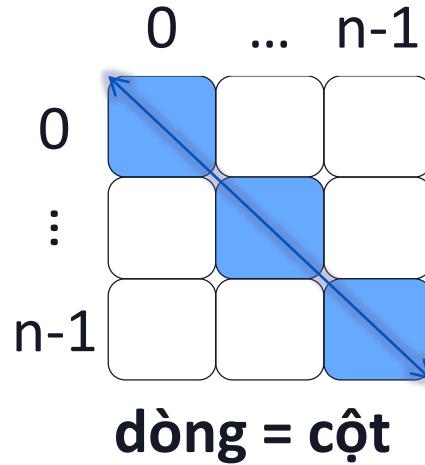
1	8	7
2	9	6
3	4	5

1	8	7
2	9	6
3	4	5

- Các phần tử nằm trên đường chéo chính là {1, 9, 5}
- Các phần tử nằm trên đường chéo phụ là {3, 9, 7}
- Các phần tử nằm nửa trên đường chéo chính là {8, 7, 6}
- Các phần tử nằm nửa dưới đường chéo chính là {2, 3, 4}



## 7.5.6 Một số khái niệm liên quan





## 7.5 Mảng hai chiều

### 7.5.7 Truyền mảng cho hàm và lời gọi hàm



## 7.5.7 Truyền mảng cho hàm và lời gọi hàm

- Cú pháp khai báo tham biến mảng:

```
<Giá_Trị_Trả_Về> <Tên_Hàm> (<Kiểu_Dữ_Liệu> <Tên_Mảng> [<Số_phần_tử_dòng>] [<Số_phần_tử_cột>],
```

<Tham số khác>, ...) ;

- Mảng có thể thay đổi nội dung sau khi thực hiện hàm.
- Có thể bỏ số lượng phần tử dòng hoặc kết hợp sử dụng con trỏ.
- Ví dụ:

```
void NhapMang(int A[3][2], int n,  
int m);  
  
void NhapMang(int A[][][2], int n,  
int m);
```



## 7.5.7 Truyền mảng cho hàm và lời gọi hàm

- Ví dụ:

```
void NhapMang(int A[3][2], int n, int m);  
void NhapMang(int A[][][2], int n, int m);  
void NhapMang(int (*A)[2], int n, int m);
```

Trong đó:

- Tên hàm: **NhapMang**
- Tham số: mảng 2 chiều số nguyên **A**  
và số lượng dòng **n**, số lượng cột **m**
- Giá trị trả về: **void**



## 7.5.7 Truyền mảng cho hàm và lời gọi hàm

- Ví dụ:

```
#include <iostream>
#define MAXR 100
#define MAXC 100
void NhapMaTran(int [] [MAXC], int&, int&);
void XuatMaTran(int [] [MAXC], int, int);
int TimKiem(int (*a)[MAXC], int n, int m, int x);
void SapXep(int [] [MAXC], int, int);
int main() {
    int a[MAXR] [MAXC], n, m;
    NhapMaTran(a, n, m);
    TimKiem(a, n, m, 10);
    SapXep(a, n, m);
    XuatMaTran(a, n, m);
}
```



## 7.5 Mảng hai chiều

### 7.5.8 Các tác vụ trên mảng 2 chiều



## 7.5.8 Các tác vụ trên mảng 2 chiều

1. Nhập mảng
2. Xuất mảng
3. Tìm kiếm một phần tử trong mảng
4. Kiểm tra tính chất của mảng
5. Tính tổng các phần tử có giá trị chẵn trong mảng
6. In các phần tử trên đường chéo chính
7. In các phần tử trên đường chéo phụ
8. Tính Tổng giá trị các phần tử trên đường chéo chính



# 1. Nhập mảng

- Yêu cầu: Nhập mảng A gồm m dòng và n cột

```
void NhapMaTran(int A[][][MAXC], int & n, int  
& m){  
    cin >> n >> m;  
  
    for (int i = 0; i < n; i++)  
        for (int j = 0; j < m; j++)  
            cin >> a[i][j];  
}
```



## 2. Xuất mảng

- **Yêu cầu:** Xuất mảng A gồm m dòng và n cột.

```
void XuatMaTran(int A[][] [MAXC], int & n, int  
& m){  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < m; j++)  
            cout << a[i][j] << " ";  
        cout << endl;  
    }  
}
```



### 3. Tìm kiếm 1 phần tử trong mảng

- **Yêu cầu:** Tìm xem phần tử x có nằm trong ma trận a kích thước (m x n) hay không?

```
int TimKiem(int (*a)[MAXC], int n, int m, int x) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (a[i][j] == x)
                return 1;
    return 0;
}
```



## 4. Kiểm tra tính chất của mảng

- **Yêu cầu**
  - Cho trước ma trận  $a$  kích thước  $(n \times m)$ . Ma trận  $a$  có phải là ma trận toàn các số nguyên tố hay không?
- **Ý tưởng**
  - YT 1: Đếm số lượng số nguyên tố của ma trận. Nếu số lượng này bằng đúng  $(n \times m)$  thì ma trận toàn số nguyên tố.
  - YT 2: Đếm số lượng số không phải số nguyên tố của ma trận. Nếu số lượng này bằng 0 thì ma trận toàn số nguyên tố.
  - YT 3: Tìm xem có phần tử nào không phải số số nguyên tố không. Nếu có thì ma trận không toàn số nguyên tố.



## 4. Kiểm tra tính chất của mảng

```
bool LaSNT(int n) {  
    if (n < 2)  return 0; // Khong la SNT  
    for (int i = 2; i <= sqrt(n); i++)  
        if (n % i == 0) return 0; // Khong la SNT  
    return 1; // SNT  
}  
  
int KiemTra_YT1(int a[][] [MAXC], int n, int m) {  
    int i, j, dem = 0;  
    for (i = 0; i < n; i++)  
        for (j = 0; j < m; j++)  
            if (LaSNT(a[i][j]) == 1) dem++;  
    if (dem == n * m) return 1;  
    return 0;  
}
```



## 4. Kiểm tra tính chất của mảng

```
int KiemTra_YT2(int a[][][MAXC], int n, int m) {  
    int i, j, dem = 0;  
  
    for (i = 0; i < n; i++)  
        for (j = 0; j < m; j++)  
            if (LaSNT(a[i][j]) == 0)  
                dem++;  
  
    if (dem == 0) return 1;  
    return 0;  
}
```



## 4. Kiểm tra tính chất của mảng

```
int KiemTra_YT3(int a[][][MAXC], int n, int m) {  
    int i, j, dem = 0;  
  
    for (i = 0; i < n; i++)  
        for (j = 0; j < m; j++)  
            if (LaSNT(a[i][j]) == 0)  
                return 0;  
  
    return 1;  
}
```



## 5. Tính tổng các phần tử có giá trị chẵn

```
int TongChan(int A[][] [MAXC], int n, int m) {  
    int TC = 0;  
    for (int i = 0; i < n; i++)  
        for (int j = 0; j < m; j++)  
            if (A[i][j] % 2 == 0)  
                TC = TC + A[i][j];  
  
    return TC;  
}
```



## 6. In các phần tử trên đường chéo chính

```
void InDuongCheoChinh(int A[][][MAXC], int n) {  
    int sum=0;  
    for (int i = 0; i < n; i++)  
        std::cout << A[i][i] << " ";  
}
```



## 7. In các phần tử trên đường chéo phụ

```
void InDuongCheoPhu(int A[][][MAXC], int n) {  
    int sum=0;  
    for (int i = 0; i < n; i++)  
        std::cout << A[i][n-1-i] << " ";  
}
```



## 8. Tính Tổng giá trị các phần tử trên đường chéo chính

```
int TongDCchinh(int a[][][MAXC], int n, int m) {  
    int i, tong;  
    tong = 0;  
    for (i = 0; i < n; i++)  
        tong = tong + a[i][i];  
    return tong;  
}
```



# BÀI TẬP

1. Nhập mảng / Xuất mảng.
2. Tìm kiếm một phần tử trong mảng.
3. Kiểm tra mảng có đối xứng qua đường chéo chính hay không?
4. Tính tổng các phần tử trên dòng/cột/toàn mảng/đường chéo chính/nửa trên/nửa dưới.
5. Tìm giá trị nhỏ nhất/lớn nhất của mảng.
6. Tính tổng 2 ma trận (mảng được xem là ma trận).
7. Tính tích 2 ma trận (mảng được xem là ma trận).
8. Kiểm tra ma trận có phải là ma trận đơn vị không (mảng được xem là ma trận)?



## 7.6 Mảng nhiều chiều



# Khai báo mảng nhiều chiều

- Cú pháp:

```
    <Kiểu_dữ_liệu> <Tên_biến_mảng> [<size_1>] ... [<  
                                         size_n>];
```

- Trong đó:

- <Kiểu\_dữ\_liệu>: int, float, char, ...
- <Tên\_biến\_mảng>: Đặt tên theo quy tắc đặt tên/định danh
- <size\_1>, ... <size\_n>, :
  - Là số lượng tối đa các mảng chiều tương ứng
  - Cần xác định ngay khi khai báo
  - Phải là một hằng số nguyên
  - Ví dụ: int A[2][3][2]; int B[20][3][5]



# Khởi tạo giá trị mảng nhiều chiều

- `int A[2][3][2] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}`
- `int B[2][3][2] = { { {1, 2}, {3, 4}, {5, 6} }, { {7, 8}, {9, 10}, {11, 12} } }`
- `int C[2][3][4] = {1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7};`
- `int D[2][3][4] = { { {1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3} }, { {5, 5, 5, 5}, {6, 6, 6, 6}, {7, 7, 7, 7} } }`



# Kích thước mảng nhiều chiều

- Tổng số phần tử có thể được lưu trữ trong một mảng nhiều chiều có thể được tính bằng cách nhân kích thước của tất cả các kích thước.
- Ví dụ: Mảng 3 chiều `int B[20][3][5]`
  - Mảng có thể lưu trữ tổng  $(20 * 3 * 5) = 300$  phần tử.
  - Mỗi phần tử có kích thước 4 byte => Mảng 3 chiều B có kích thước  $300 * 4 = 1200$  bytes.



# Chỉ số mảng và truy xuất mảng nhiều chiều

- Truy xuất phần tử mảng nhiều chiều thông qua chỉ số:

`<Tên_b minden_mảng> [<chỉ_số_1>]`

- Ví dụ:

```
int A[2][3][2] = { { {1, 2}, {3, 4}, {5, 6} },  
                    { {7, 8}, {9, 10}, {11, 12} } };
```

$$A[0][0][0] = \quad A[1][0][0] =$$

1                      7

$$A[0][0][1] = \quad A[1][0][1] =$$

2                      8

$$A[0][1][0] = \quad A[1][1][0] =$$

3                      9

$$A[0][1][1] = \quad A[1][1][1] =$$

4                      10



# Lấy địa chỉ các phần tử mảng nhiều chiều

- Lấy địa chỉ phần tử mảng nhiều chiều:

[& <Tên\_bien\_mảng> [<Chỉ\_số\_1>  
[<Chỉ\_số\_2>], ...;

- Ví dụ:

```
int A[2][3][2] = { { {1, 2}, {3, 4}, {5, 6} },  
                    { {7, 8}, {9, 10}, {11, 12} } };
```

&A[0][0][0]: 0x61fdf0(&A[1][0][0]):

&A[0][0][1]: 0x61fdf40x61fe08

&A[0][1][0]: 0x61fdf8&A[1][0][1]:

&A[0][1][1]: 0x61fdfc0x61fe0c

&A[0][2][0]: 0x61fe00(&A[1][1][0]):

&A[0][2][1]: 0x61fe040x61fe10

A[0][0][0] A[0][0][1] A[0][1][0] A[0][1][1] A[0][2][0] A[0][2][1] A[1][0][0] A[1][0][1] A[1][1][0] A[1][1][1] A[1][2][0] A[1][2][1]  
0x61fdf0 0x61fdf4 0x61fdf8 0x61fdfc 0x61fe00 0x61fe04 0x61fe08 0x61fe0c 0x61fe10 0x61fe14 0x61fe18 0x61fe1c

&A[1][2][1]:

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----



# Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    // Mảng này có thể lưu trữ tối đa 12 phần tử (2x3x2)
    int A[2][3][2] = { { {1, 2},
                           {3, 4},
                           {5, 6} },
                        { {7, 8},
                           {9, 10},
                           {11, 12} } };
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 2; ++k) {
                cout << "A[" << i << "][" << j << "][" << k
                    << A[i][j][k] << endl;
            }
        }
    }
    return 0;
}
```

## Kết quả thực thi:

```
A[0][0][0] = 1
A[0][0][1] = 2
A[0][1][0] = 3
A[0][1][1] = 4
A[0][2][0] = 5
A[0][2][1] = 6
A[1][0][0] = 7
A[1][0][1] = 8
A[1][1][0] = 9
A[1][1][1] = 10
A[1][2][0] = 11
A[1][2][1] = 12
```



# Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    ..
    cout << A[0][0][0] << endl;
    cout << A[0][0][1] << endl;
    cout << A[0][1][0] << endl;
    cout << A[0][1][1] << endl;
    cout << A[0][2][0] << endl;
    cout << A[0][2][1] << endl;
    cout << A[1][0][0] << endl;
    cout << A[1][0][1] << endl;
    cout << A[1][1][0] << endl;
    cout << A[1][1][1] << endl;
    cout << A[1][2][0] << endl;
    cout << A[1][2][1] << endl;
    return 0;
}
```

## Kết quả thực thi:

A[0][0][0] = 1  
A[0][0][1] = 2  
A[0][1][0] = 3  
A[0][1][1] = 4  
A[0][2][0] = 5  
A[0][2][1] = 6  
A[1][0][0] = 7  
A[1][0][1] = 8  
A[1][1][0] = 9  
A[1][1][1] = 10  
A[1][2][0] = 11  
A[1][2][1] = 12



# Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    ...
    cout << &A[0][0][0] << endl;
    cout << &A[0][0][1] << endl;
    cout << &A[0][1][0] << endl;
    cout << &A[0][1][1] << endl;
    cout << &A[0][2][0] << endl;
    cout << &A[0][2][1] << endl;
    cout << &A[1][0][0] << endl;
    cout << &A[1][0][1] << endl;
    cout << &A[1][1][0] << endl;
    cout << &A[1][1][1] << endl;
    cout << &A[1][2][0] << endl;
    cout << &A[1][2][1] << endl;
    return 0;
}
```

## Kết quả thực thi:

```
&A[0][0][0]: 0x61fdf0
&A[0][0][1]: 0x61fdf4
&A[0][1][0]: 0x61fdf8
&A[0][1][1]: 0x61fdfe
&A[0][2][0]: 0x61fe00
&A[0][2][1]: 0x61fe04
&A[1][0][0]: 0x61fe08
&A[1][0][1]: 0x61fe0c
&A[1][1][0]: 0x61fe10
&A[1][1][1]: 0x61fe14
&A[1][2][0]: 0x61fe18
&A[1][2][1]: 0x61fe1c
```



# Chúc các em học tốt!

