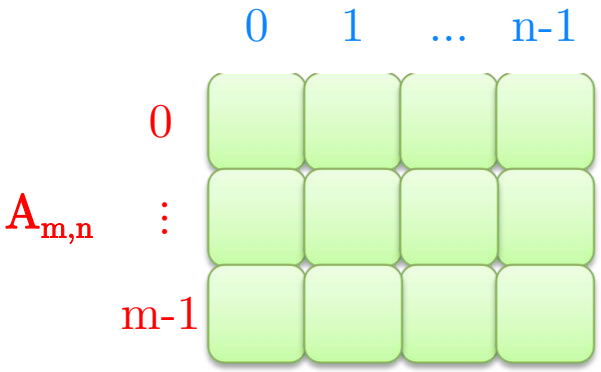
NHẬP MÔN LẬP TRÌNH

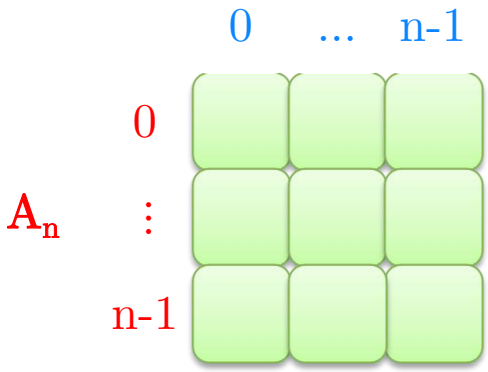
MẢNG HAI CHIỀU

Nội dung

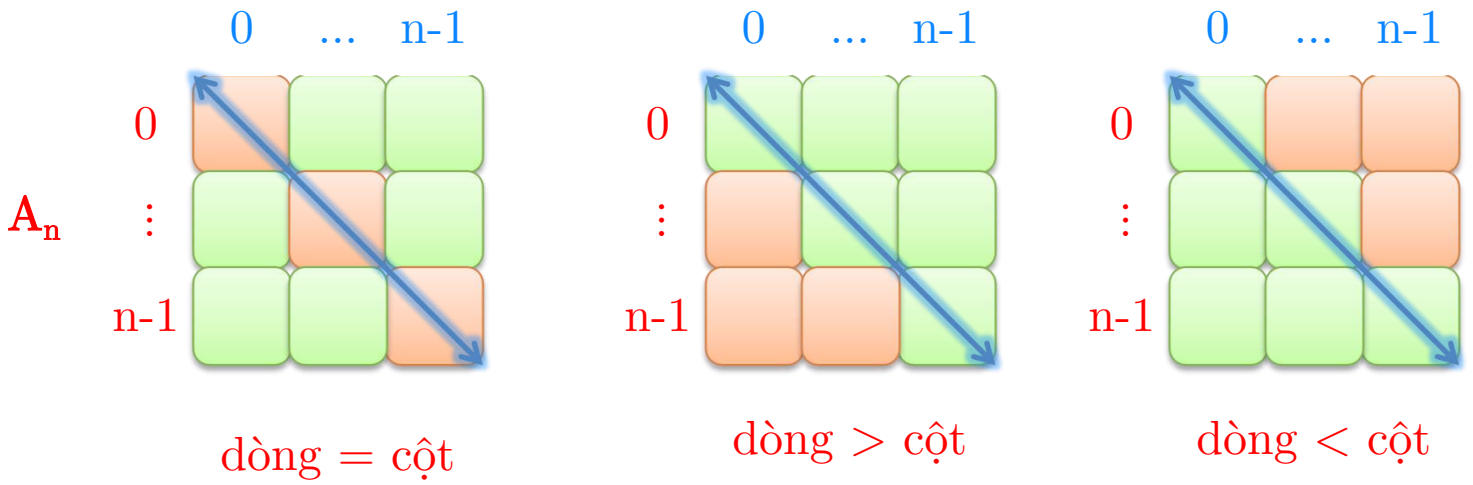
1. Khái niệm
2. Khai báo
3. Truy xuất dữ liệu kiểu mảng
4. Một số bài toán trên mảng 2 chiều

Ma Trận



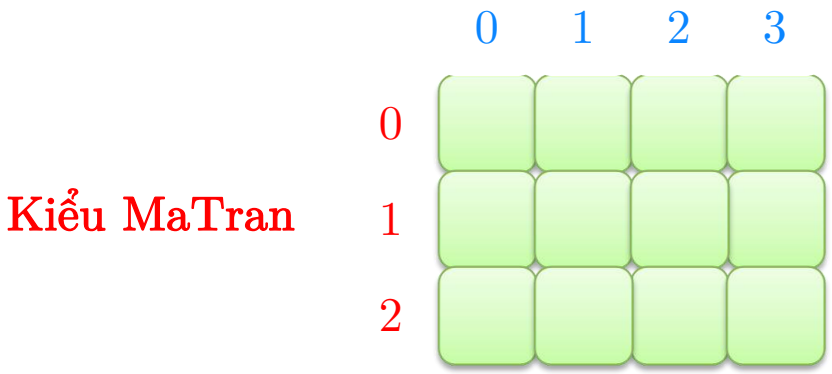


Ma Trận


0 ... n-1
0
A_n ⋮
n-1
dòng = cột
0 ... n-1
0
⋮
n-1
dòng > cột
0 ... n-1
0
⋮
n-1
dòng < cột
0 ... n-1
0
A_n ⋮
n-1
dòng + cột = n-1
0 ... n-1
0
⋮
n-1
dòng + cột > n-1
0 ... n-1
0
⋮
n-1
dòng + cột < n-1

Khai báo kiểu mảng 2 chiều

- Cú pháp
  - `typedef <kiểu cơ sở> <tên kiểu>[<N1>][<N2>];`
    - N1, N2: số lượng phần tử mỗi chiều
- Ví dụ
  - `typedef int MaTran[3][4];`



Khai báo biến mảng 2 chiều

Cú pháp

- Tường minh

```
<kiểu cơ sở> <tên biến>[<N1>][<N2>];
```

- Không tường minh (thông qua kiểu)

```
typedef <kiểu cơ sở> <tên kiểu>[<N1>][<N2>];

<tên kiểu> <tên biến>;
<tên kiểu> <tên biến 1>, <tên biến 2>;
```

Khai báo biến mảng 2 chiều

- Ví dụ
  - Tường minh

```
int a[10][20], b[10][20];
int c[5][10];
int d[10][20];
```

  - Không tường minh (thông qua kiểu)

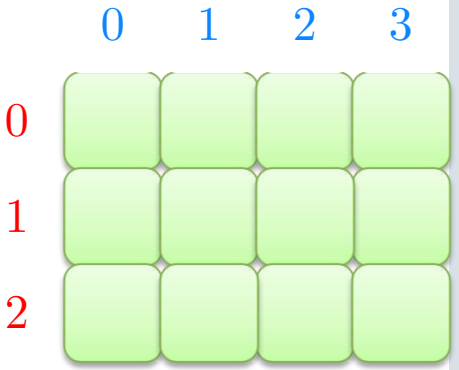
```
typedef int MaTran10x20[10][20];
typedef int MaTran5x10[5][10];

MaTran10x20 a, b;
MaTran11x11 c;
MaTran10x20 d;
```

Truy xuất đến một phần tử

- Thông qua chỉ số

```
<tên biến mảng>[<giá trị cs1>][<giá trị cs2>]
```

- Ví dụ
  - Cho mảng 2 chiều như sau




```
int a[3][4];
```

  - Các truy xuất
    - Hợp lệ: a[0][0], a[0][1], ..., a[2][2], a[2][3]
    - Không hợp lệ: a[-1][0], a[2][4], a[3][3]

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
<biến mảng đích>[<giá trị cs1>][giá trị cs2] =
                                        <giá trị>;
```

- Ví dụ

```
int a[5][10], b[5][10];

b = a;          // Sai
int i, j;
for (i = 0; i < 5; i++)
      for (j = 0; j < 10; j++)
            b[i][j] = a[i][j];
```

Truyền mảng cho hàm

Truyền mảng cho hàm

- Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
void NhapMaTran(int a[50][100]);
```

- Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng
  - Có thể bỏ số lượng phần tử chiều thứ 2 hoặc con trỏ.
  - Mảng có thể thay đổi nội dung sau khi thực hiện hàm.

```
void NhapMaTran(int a[][100]);
void NhapMaTran(int (*a)[100]);
```

Truyền mảng cho hàm

Truyền mảng cho hàm

- Số lượng phần tử thực sự truyền qua biến khác

```
void XuatMaTran(int a[50][100], int m, int n);
void XuatMaTran(int a[][100], int m, int n);
void XuatMaTran(int (*a)[100], int m, int n);
```

Lời gọi hàm

```
void NhapMaTran(int a[][100], int &m, int &n);
void XuatMaTran(int a[][100], int m, int n);
void main()
{
        int a[50][100], m, n;
        NhapMaTran(a, m, n);
        XuatMaTran(a, m, n);
}
```

Một số bài toán cơ bản

- Viết chương trình con thực hiện các yêu cầu sau
  - Nhập mảng
  - Xuất mảng
  - Tìm kiếm một phần tử trong mảng
  - Kiểm tra tính chất của mảng
  - Tính tổng các phần tử trên dòng/cột/toàn ma trận/đường chéo chính/nửa trên/nửa dưới
  - Tìm giá trị nhỏ nhất/lớn nhất của mảng
  - ...

Một số quy ước

- Kiểu dữ liệu

```
#define MAXD 50
#define MAXC 100
```

- Các chương trình con
  - Hàm void HoanVi(int x, int y): hoán vị giá trị của hai số nguyên.
  - Hàm int LaSNT(int n): kiểm tra một số có phải là số nguyên tố. Trả về 1 nếu n là số nguyên tố, ngược lại trả về 0.

Thủ tục HoanVi & Hàm LaSNT

```
void HoanVi(int &x, int &y)
{
	int tam = x; x = y; y = tam;
}

int LaSNT(int n)
{
	int i, dem = 0;
	for (i = 1; i <= n; i++)
		if (n%i == 0)
			dem++;

	if (dem == 2)
		return 1;
	else return 0;
}
```

Nhập Ma Trận

- Yêu cầu
  - Cho phép nhập mảng a, m dòng, n cột
- Ý tưởng
  - Cho trước một mảng 2 chiều có dòng tối đa là MAXD, số cột tối đa là MAXC.
  - Nhập số lượng phần tử thực sự m, n của mỗi chiều.
  - Nhập từng phần tử từ [0][0] đến [m-1][n-1].

Hàm Nhập Ma Trận

```
void NhapMaTran(int a[][MAXC], int &m, int &n)
{
        printf("Nhap so dong, so cot cua ma tran: ");
        scanf("%d%d", &m, &n);

        int i, j;
        for (i=0; i<m; i++)
                for (j=0; j<n; j++)
                {
                        printf("Nhap a[%d][%d]: ", i, j);
                        scanf("%d", &a[i][j]);
                }
}
```

Xuất Ma Trận

- Yêu cầu
  - Cho phép nhập mảng a, m dòng, n cột
- Ý tưởng
  - Xuất giá trị từng phần tử của mảng 2 chiều từ dòng có 0 đến dòng m-1, mỗi dòng xuất giá giá trị của cột 0 đến cột n-1 trên dòng đó.

Hàm Xuất Ma Trận

```
void XuatMaTran(int a[][MAXC], int m, int n)
{
        int i, j;
        for (i=0; i<m; i++)
        {
                for (j=0; j<n; j++)
                        printf("%d ", a[i][j]);

                printf("\n");
        }
}
```

Tìm kiếm một phần tử trong Ma Trận

- Yêu cầu
  - Tìm xem phần tử x có nằm trong ma trận a kích thước mxn hay không?
- Ý tưởng
  - Duyệt từng phần của ma trận a. Nếu phần tử đang xét bằng x thì trả về có (1), ngược lại trả về không có (0).

Hàm Tìm Kiếm

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

Kiểm tra tính chất của mảng

- Yêu cầu
  - Cho trước ma trận a kích thước mxn. Ma trận a có phải là ma trận toàn các số nguyên tố hay không?
- Ý tưởng
  - Cách 1: Đếm số lượng số ngtố của ma trận. Nếu số lượng này bằng đúng mxn thì ma trận toàn ngtố.
  - Cách 2: Đếm số lượng số không phải ngtố của ma trận. Nếu số lượng này bằng 0 thì ma trận toàn ngtố.
  - Cách 3: Tìm xem có phần tử nào không phải số ngtố không. Nếu có thì ma trận không toàn số ngtố.

Hàm Kiểm Tra (Cách 1)

```
int KiemTra_C1(int a[][MAXC], int m, int n)
{
      int i, j, dem = 0;

      for (i=0; i<m; i++)
            for (j=0; j<n; j++)
                  if (LaSNT(a[i][j]==1)
                        dem++;

      if (dem == m*n)
            return 1;
      return 0;
}
```

Hàm Kiểm Tra (Cách 2)

```
int KiemTra_C2(int a[][MAXC], int m, int n)
{
	int i, j, dem = 0;

	for (i=0; i<m; i++)
		for (j=0; j<n; j++)
			if (LaSNT(a[i][j]==0)
				dem++;

	if (dem == 0)
		return 1;
	return 0;
}
```

Hàm Kiểm Tra (Cách 2)

```
int KiemTra_C3(int a[][MAXC], int m, int n)
{
      int i, j, dem = 0;

      for (i=0; i<m; i++)
            for (j=0; j<n; j++)
                  if (LaSNT(a[i][j]==0)
                        return 0;

      return 1;
}
```

Tính tổng các phần tử

Yêu cầu

- Cho trước ma trận a, kích thước mxn. Tính tổng các phần tử trên:
  - Dòng d, cột c
  - Đường chéo chính, đường chéo phụ (ma trận vuông)
  - Nửa trên/dưới đường chéo chính (ma trận vuông)
  - Nửa trên/dưới đường chéo phụ (ma trận vuông)

Ý tưởng

- Duyệt ma trận và cộng dồn các phần tử có tọa độ (dòng, cột) thỏa yêu cầu.

Hàm tính tổng trên dòng

```
int TongDong(int a[][MAXC], int m, int n, int d)
{
      int j, tong;

      tong = 0;

      for (j=0; j<n; j++)      // Duyệt các cột
            tong = tong + a[d][j];

      return tong;
}
```

Hàm tính tổng trên cột

```
int TongCot(int a[][MAXC], int m, int c)
{
      int i, tong;

      tong = 0;

      for (i=0; i<m; i++)      // Duyệt các dòng
            tong = tong + a[i][c];

      return tong;
}
```

Hàm tính tổng đường chéo chính

```
int TongDCChinh(int a[][MAXC], int n)
{
	int i, tong;

	tong = 0;

	for (i=0; i<n; i++)
		tong = tong + a[i][i];

	return tong;
}
```

Hàm tính tổng trên đường chéo chính

```
int TongTrenDCChinh(int a[][MAXC], int n)
{
        int i, j, tong;

        tong = 0;

        for (i=0; i<n; i++)
              for (j=0; j<n; j++)
                    if (i < j)
                          tong = tong + a[i][j];

        return tong;
}
```

Hàm tính tổng dưới đường chéo chính

```
int TongTrenDCChinh(int a[][MAXC], int n)
{
        int i, j, tong;

        tong = 0;

        for (i=0; i<n; i++)
              for (j=0; j<n; j++)
                    if (i > j)
                          tong = tong + a[i][j];

        return tong;
}
```

Hàm tính tổng trên đường chéo phụ

```
int TongDCPhu(int a[][MAXC], int n)
{
      int i, tong;

      tong = 0;

      for (i=0; i<n; i++)
            tong = tong + a[i][n-i-1];

      return tong;
}
```

Tìm giá trị lớn nhất của Ma Trận

- Yêu cầu
  - Cho trước ma trận a, kích thước mxn. Tìm giá trị lớn nhất trong ma trận a (gọi là max)
- Ý tưởng
  - Giả sử giá trị max hiện tại là giá trị phần tử đầu tiên a[0][0]
  - Lần lượt kiểm tra các phần tử còn lại để cập nhật max.

Hàm tìm Max

```
int TimMax(int a[][MAXC], int m, int n)
{
        int i, j, max;

        max = a[0][0];

        for (i=0; i<m; i++)
              for (j=0; j<n; j++)
                    if (a[i][j] > max)
                          max = a[i][j];

        return max;
}
```