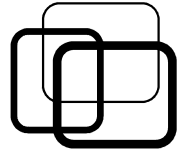


Con trỏ nâng cao

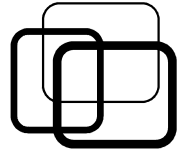
GV. Nguyễn Minh Huy

Nội dung

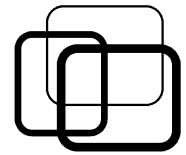


- Quản lý bộ nhớ.
- Con trỏ nhiều cấp.
- Các loại con trỏ khác.

Nội dung



- **Quản lý bộ nhớ.**
- Con trỏ nhiều cấp.
- Các loại con trỏ khác.



■ Cấp phát bộ nhớ trong C:

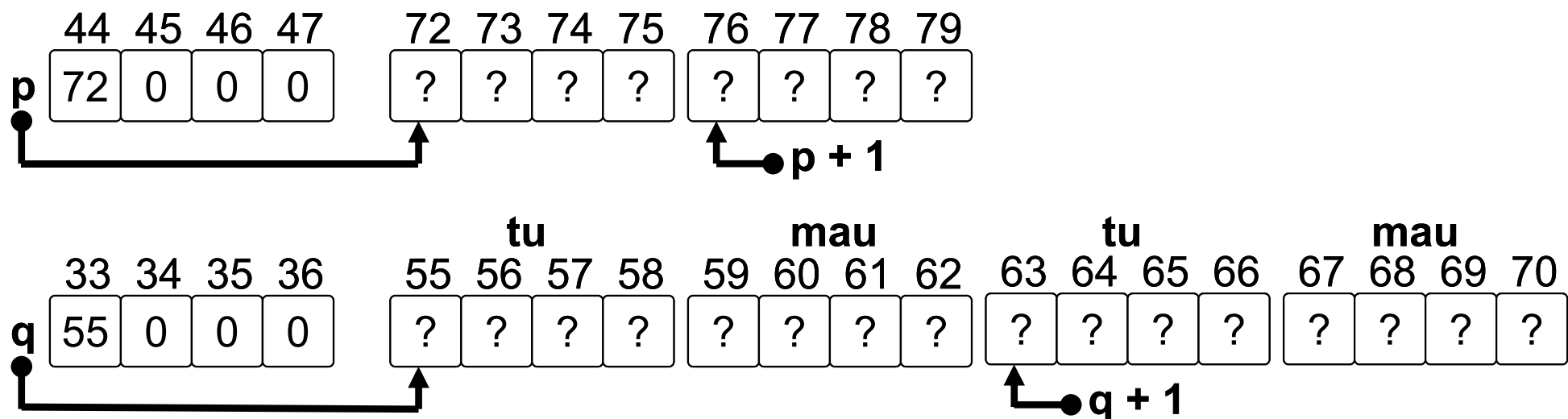
- Xin một vùng nhớ trong RAM để sử dụng.

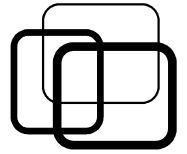
- Lệnh malloc: thư viện <malloc.h>

- Cú pháp: **malloc**(<số byte cấp phát>);

- Trả về: địa chỉ vùng nhớ được cấp.

```
int      *p = (int *) malloc( 2 * sizeof(int) );  
PhanSo  *q = (PhanSo *) malloc( 2 * sizeof(PhanSo) );
```





■ Thu hồi bộ nhớ trong C:

- Trả lại vùng nhớ đã xin của RAM.

- Quy tắc quản lý bộ nhớ của C:

 - Tự thu hồi vùng nhớ khai báo biến.

 - Không thu hồi vùng nhớ tự cấp phát.

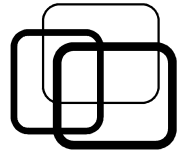
➔ Lập trình viên phải tự thu hồi vùng nhớ cấp phát.

- Quên thu hồi ➔ “Rò rỉ” bộ nhớ (memory leak).

- Lệnh free: thư viện <malloc.h>

 - Cú pháp: **free**(<Con trỏ giữ địa chỉ vùng nhớ>);

```
float      *r = (float *) malloc( 20 * sizeof(float) );  
free(r);
```



■ Cấp phát và thu hồi trong C++:

- C++ tương thích với C (hỗ trợ malloc).

- C++ có lệnh cấp phát và thu hồi mới.

- Toán tử **new**: cấp phát bộ nhớ.

 - Cú pháp: **new** <Kiểu dữ liệu>[<Số phần tử>];

 - Trả về: địa chỉ vùng nhớ được cấp.

- Toán tử **delete**: thu hồi bộ nhớ.

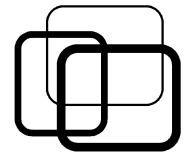
 - Cú pháp: **delete** <Con trỏ giữ địa chỉ vùng nhớ>;

```
int      *p = new int [ 10 ];
```

```
PhanSo *q = new PhanSo [ 30 ];
```

```
delete [ ]p;
```

```
delete [ ]q;
```

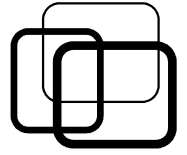


■ Mảng động một chiều:

- Không cần biết số phần tử khi khai báo.
 - Cấp phát vùng nhớ vừa đủ.
 - Thu hồi vùng nhớ khi sử dụng xong.
- Sử dụng bộ nhớ hiệu quả hơn.

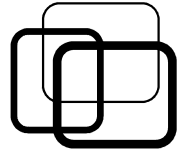
```
void nhapMang(int *&a, int &n) {  
    printf("Nhap so phan tu: ");  
    scanf("%d", &n);  
    a = new int [ n ];  
    for (int i = 0; i < n; i++) {  
        printf("Nhap phan tu %d:", i);  
        scanf("%d", &a[ i ]);  
    }  
}
```

```
void main()  
{  
    int *a;  
    int n;  
  
    nhapMang(a, n);  
    delete [ ]a;  
}
```



- Quản lý bộ nhớ.
- **Con trỏ nhiều cấp.**
- Các loại con trỏ khác.

Con trỏ nhiều cấp



■ Địa chỉ của con trỏ:

■ Mỗi biến có một địa chỉ.

- Biến int có kiểu địa chỉ int *.

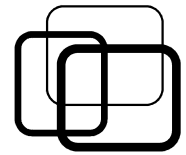
■ Biến con trỏ cũng có địa chỉ.

- Biến int * có kiểu địa chỉ là gì?

■ Khái niệm con trỏ nhiều cấp:

- Biến giữ địa chỉ của con trỏ.
- Còn gọi là con trỏ của con trỏ.
- Có kiểu là: <kiểu con trỏ> *.

Con trỏ nhiều cấp



■ Sử dụng con trỏ cấp 2 trong C:

■ Khai báo:

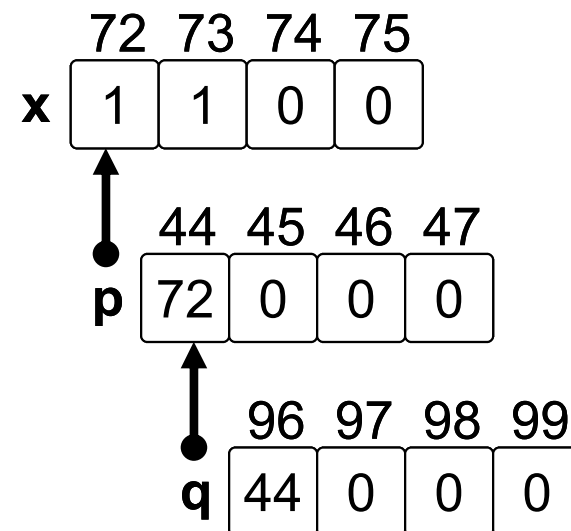
- Cách 1: dùng dấu *****.
- Cách 2: dùng từ khóa **typedef**.

■ Khởi tạo:

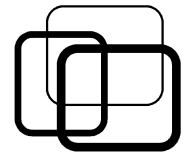
- Khởi tạo **NULL**.
- Khởi tạo bằng toán tử **&**.

```
int x = 257;  
int *p = NULL;  
int **q = NULL;
```

```
p = &x;  
q = &p;
```



Con trỏ nhiều cấp



■ Sử dụng con trỏ cấp 2 trong C:

■ Truy xuất nội dung vùng nhớ:

- Truy xuất 1 cấp: toán tử `*`.
- Truy xuất 2 cấp: toán tử `**`.

■ Truyền tham số:

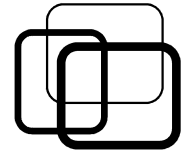
- Truyền tham trị.
- Truyền tham chiếu.

➔ Những giá trị nào thay đổi trong hàm `foo`?

```
void foo(int **g, int **&h)
{
    (**g)++; (*g)++; g++;
    (**h)++; (*h)++; h++;
}
```

```
void main()
{
    int a[10];
    int *p = a;
    int **q = &p;
    int **r = &p;

    foo(q, r);
}
```



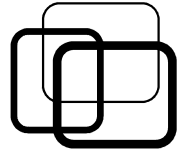
■ Mảng đông nhiều chiều:

■ Mảng con trỏ:

- Con trỏ cấp 1 là mảng động một chiều.
- Con trỏ cấp 2 là mảng động các con trỏ cấp 1.
- ➔ Dùng làm mảng động nhiều chiều.

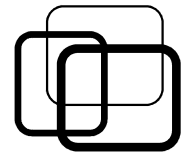
```
void nhapMang(int **&a, int &dong, int &cot) {  
    scanf("%d %d", &dong, &cot);  
    a = new int * [ dong ];  
    for (int i = 0; i < dong; i++) {  
        a[ i ] = new int [ cot ];  
        for (int j = 0; j < cot; j++)  
            scanf("%d", &a[ i ][ j ]);  
    }  
}
```

```
void main()  
{  
    int **a;  
    int dong, cot;  
  
    nhapMang(a, dong, cot);  
    delete [ ]a;  
}
```



- Quản lý bộ nhớ.
- Con trỏ nhiều cấp.
- **Các loại con trỏ khác.**

Các loại con trỏ khác



■ Hằng con trỏ (constant pointer):

- Con trỏ suốt đời giữ 1 địa chỉ (khi khởi tạo).
- Khai báo: `<Kiểu dữ liệu> * const <Tên con trỏ>;`

```
int x = 5, y = 6;  
int * const p = &x;  
p = &y;    // Sai.
```

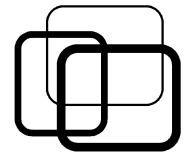
- Tất cả mảng tĩnh trong C đều là hằng con trỏ.

■ Con trỏ hằng (pointer to constant):

- Nội dung vùng nhớ con trỏ giữ địa chỉ là hằng.
- Khai báo: `const <Kiểu dữ liệu> * <Tên con trỏ>;`

```
int x = 5;  
const int *p = &x;  
*p = 6;    // Sai.
```

Các loại con trỏ khác



■ Con trỏ void:

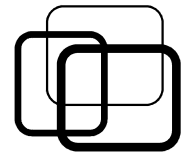
- Giữ địa chỉ kiểu bất kỳ.
- Có kiểu void *.
- Ép về kiểu địa chỉ mong muốn khi dùng.

```
void inByteVungNho(void *p, int size)
{
    char *q = ( unsigned char * ) p;
    for ( int i = 0; i < size; i++ )
        printf( "%d ", q[ i ] );
}
```

```
void main()
{
    int    x = 1057;
    double y = 1.25;

    inByteVungNho(&x, 4);
    inByteVungNho(&y, 8);
}
```

Các loại con trỏ khác



■ Con trỏ hàm:

■ Địa chỉ hàm:

- Hàm được lưu trong bộ nhớ như biến.
- Mỗi hàm có một địa chỉ.

■ Con trỏ hàm là biến giữ địa chỉ hàm.

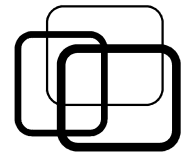
■ Khai báo:

```
<Kiểu trả về> (* <Tên con trỏ>) (<DS tham số>);  
typedef <Kiểu trả về> (* <Tên thay thế>) (<DS tham số>);  
<Tên thay thế> <Tên con trỏ>;
```

■ Các hàm có cùng kiểu địa chỉ khi:

- Cùng kiểu trả về.
- Cùng danh sách tham số.

Các loại con trỏ khác



■ Con trỏ hàm:

```
typedef int (*PhepTinh)(int a, int b);
```

```
int cong(int u, int v)
{
    return u + v;
}
```

```
int nhan(int u, int v)
{
    return u * v;
}
```

```
int  tinhToan(int u, int v, PhepTinh p)
{
    //  $u^3$  operator  $v^2$ .
    return p(u*u*u, v*v);
}
```

```
void main()
{
```

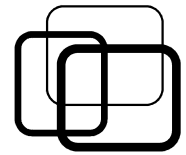
```
    int x = 5;
    int y = 6;
```

```
    PhepTinh p = cong;
    int kq1 = p(x, y);
```

```
    p = nhan;
    int kq2 = p(x, y);
```

```
    int kq3 =  tinhToan(x, y, cong);
```

```
}
```



■ Con trỏ đến vùng nhớ kích thước cố định:

■ Địa chỉ của mảng một chiều tĩnh:

- Mảng một chiều tĩnh có kiểu địa chỉ thế nào?

```
int a[ 10 ];  
int *p = a      // p và a là địa chỉ của a[ 0 ].  
??? q = &a;
```

■ Con trỏ đến vùng nhớ kích thước cố định:

- Biến giữ địa chỉ của mảng 1 chiều tĩnh.

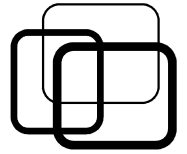
- Khai báo:

<Kiểu dữ liệu mảng> (*<Tên con trỏ>)[<Số phần tử mảng>];

```
int a[ 10 ];
```

```
int ( *p )[ 10 ] = &a;  // p chỉ trỏ đến vùng nhớ  
                        // có kích thước 10 phần tử.
```

Các loại con trỏ khác

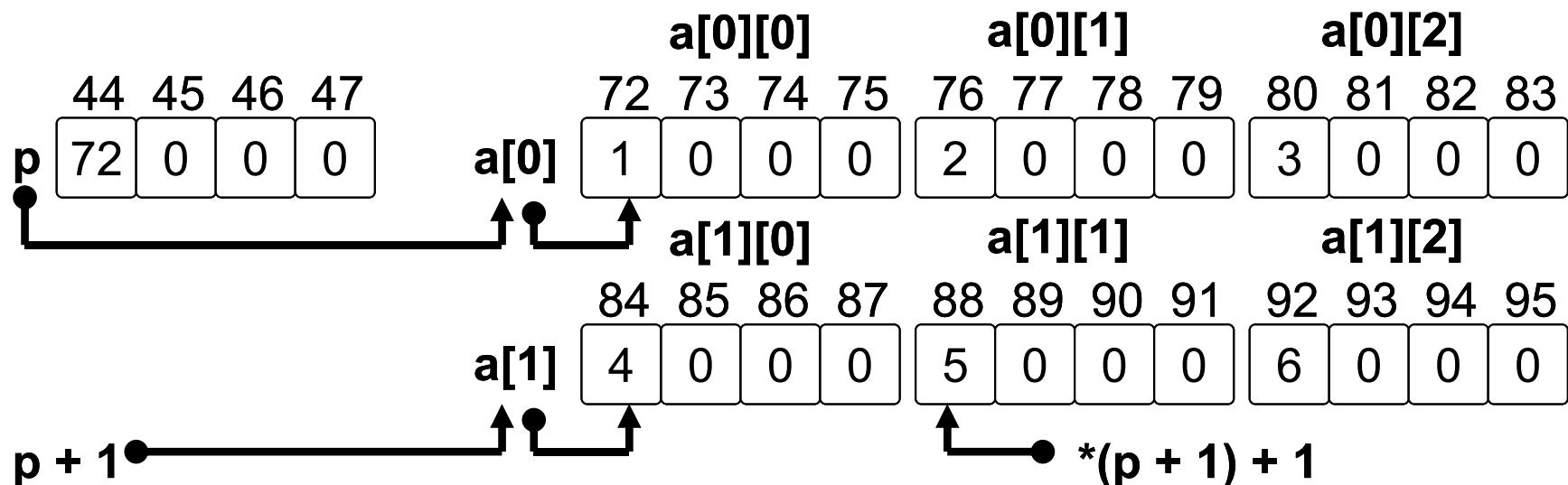


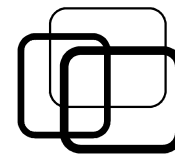
■ Con trỏ đến vùng nhớ kích thước cố định:

■ Bản chất mảng hai chiều tĩnh trong C:

- Là con trỏ giữ địa chỉ mảng 1 chiều tĩnh.
- Giữ địa chỉ dòng đầu tiên.

```
int a[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };  
int (*p)[3] = a; // a = &a[0].  
printf("%d\n", *( *(p + 1) + 1 ) );
```





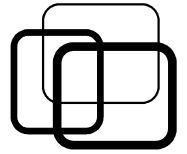
■ Con trỏ đến vùng nhớ kích thước cố định:

■ Truyền tham số mảng hai chiều tĩnh:

- Không phải truyền tất cả mảng.
- Chỉ truyền địa chỉ dòng đầu tiên.

```
void xuatMang(int a[ ][20], int dong, int cot) { // truyền &a[ 0 ].
    for (int i = 0; i < dong; i++) {
        for (int j = 0; j < cot; j++)
            printf("%d ", a[ i ][ j ] );
        printf("\n");
    }
}

void main() {
    int a[10][20];
    xuatMang(a, 10, 20);
}
```



■ Quản lý bộ nhớ:

■ Cấp phát:

- Xin một vùng nhớ trong RAM.
- Các lệnh: malloc, calloc, toán tử new.

■ Thu hồi:

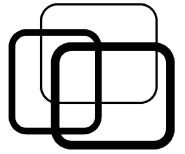
- Trả lại một vùng nhớ cho RAM.
- Các lệnh: free, toán tử delete.

■ Con trỏ cấp 1 là mảng động 1 chiều.

■ Các loại con trỏ:

- Nhiều kiểu địa chỉ → nhiều loại con trỏ.
- Mỗi loại con trỏ giữ 1 kiểu địa chỉ.



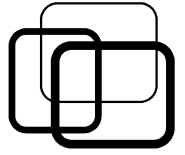


■ Các loại con trỏ:

- Địa chỉ con trỏ → con trỏ nhiều cấp.
- Hằng địa chỉ → hằng con trỏ.
- Địa chỉ hằng → con trỏ hằng.
- Địa chỉ kiểu bất kỳ → con trỏ void.
- Địa chỉ hàm → con trỏ hàm.
- Địa chỉ mảng tĩnh → con trỏ đến vùng nhớ cố định.



Bài tập



■ Bài tập 3.1:

Mảng nhiều chiều `m` được khai báo như sau:

```
int m[4][6];
```

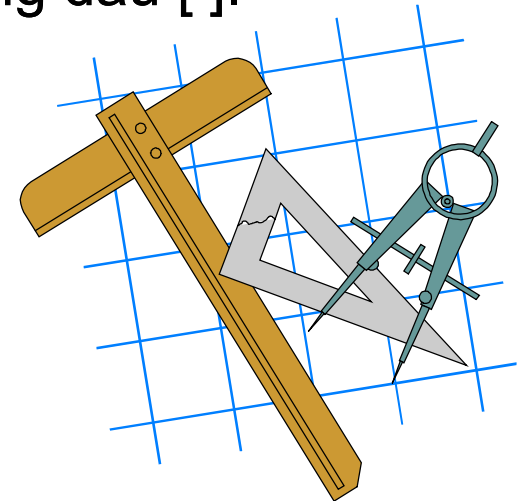
Hãy cho biết kiểu địa chỉ của:

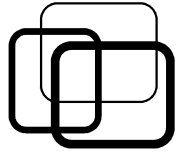
a) `m[1][3]`.

b) `m[0]`.

c) `m`.

Hãy cho biết câu lệnh truy xuất `m[2][4]` mà không dùng dấu `[]`.





■ Bài tập 3.2:

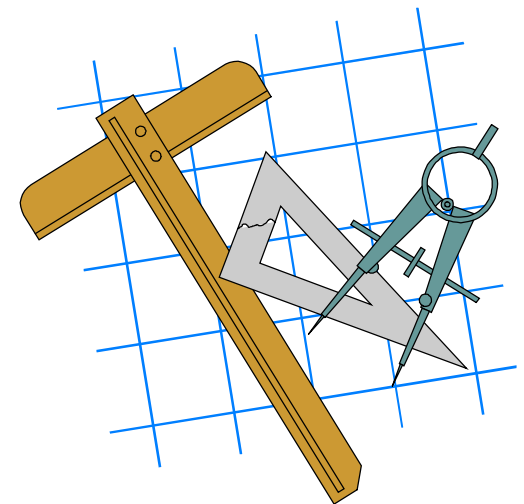
Xét đoạn chương trình C sau:

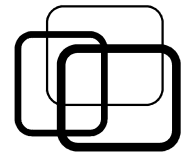
```
void khoiTao(double **p)
{
    for (int i = 0; i < 5; i++)
        *(p + i) = new double[ i + 1 ];
}
```

Hãy cho biết:

- Ở mỗi dòng trong hàm main có bao nhiêu byte được cấp phát.
- Sau hàm **khoiTao**, các phần tử của mảng **p** được cấp phát thế nào.
- Hàm **thuHoi** được viết ra sao để để không gây rò rỉ bộ nhớ.

```
void main()
{
    double *p[10];
    khoiTao(p + 3);
    thuHoi(p);
}
```





■ Bài tập 3.3:

Hãy khai báo kiểu địa chỉ cho những hàm sau (dùng typedef):

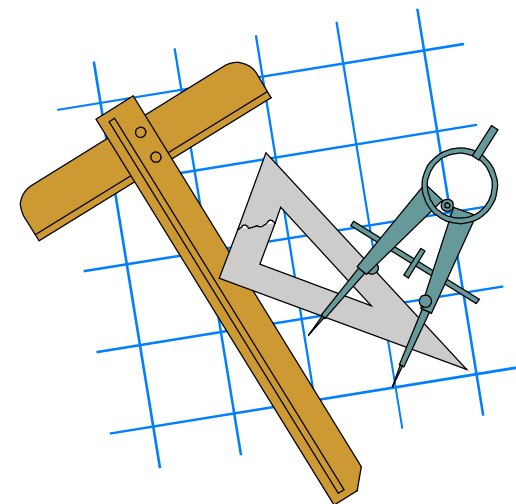
```
void xuly();
```

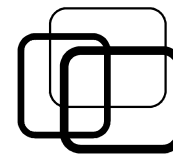
```
int luyThua(int x, int n);
```

```
int * nhapMang(int &n);
```

```
void xuatMang(int a[ ], int n);
```

```
PhanSo cong(PhanSo p1, PhanSo p2);
```

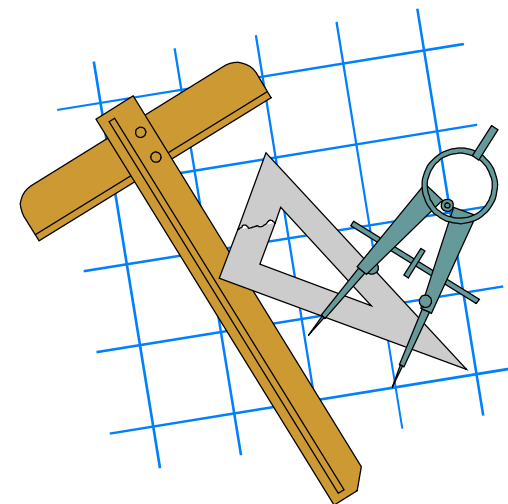


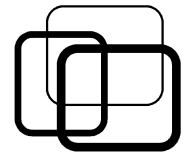


■ Bài tập 3.4:

Viết chương trình C (dùng mảng động) thực hiện những việc sau:

- Nhập vào từ bàn phím ma trận số nguyên $M \times N$.
- Trích ra danh sách các số nguyên tố trong ma trận.
- Xuất danh sách trích được ra màn hình.





■ Bài tập 3.5 (*):

Viết chương trình C thực hiện sắp xếp mảng N số nguyên theo thứ tự bất kỳ do người dùng lựa chọn.

Gợi ý: dùng con trỏ hàm làm điều kiện sắp xếp.

