

TÀI LIỆU LÍ THUYẾT NMLT

# Mảng Một Chiều



# Nội dung

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



# Đặt vấn đề

- Ví dụ
  - Chương trình cần lưu trữ **3** số nguyên?  
=> Khai báo **3** biến **int a1, a2, a3;**
  - Chương trình cần lưu trữ **100** số nguyên?  
=> Khai báo **100** biến kiểu số nguyên!
  - Người dùng muốn nhập **n** số nguyên?  
=> Không thực hiện được!
- Giải pháp
  - Kiểu dữ liệu mới cho phép **lưu trữ một dãy** các số nguyên và **dễ dàng truy xuất**.

# Dữ liệu kiểu mảng

- Khái niệm
  - Là một **kiểu dữ liệu có cấu trúc** do người lập trình định nghĩa.
  - Biểu diễn một **dãy các biến có cùng kiểu**. Ví dụ: dãy các số nguyên, dãy các ký tự...
  - Kích thước được **xác định ngay khi khai báo** và **không bao giờ thay đổi**.
  - NNLT C luôn chỉ định **một khối nhớ liên tục** cho một biến kiểu mảng.

# Khai báo biến mảng (tường minh)

- Tường minh

```
<kiểu cơ sở> <tên biến mảng> [<số phần tử>];  
<kiểu cơ sở> <tên biến mảng> [<N1>] [<N2>] ... [<Nn>];
```

- <N1>, ..., <Nn> : số lượng phần tử của mỗi chiều.

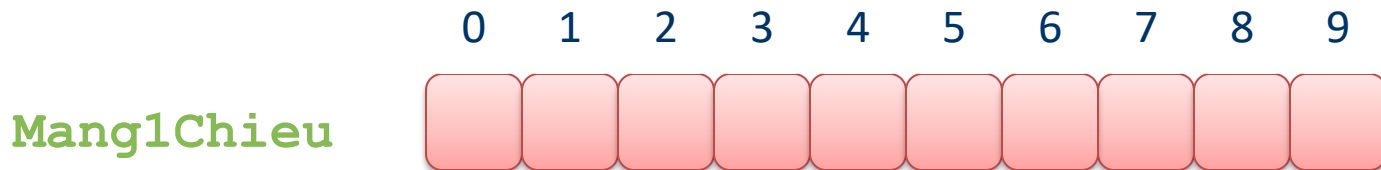
- Lưu ý

- Phải **xác định <số phần tử> cụ thể (hằng)** khi khai báo.
- Mảng nhiều chiều:  $\text{<tổng số phần tử>} = N1 * N2 * \dots * Nn$
- Bộ nhớ sử dụng =  $\text{<tổng số phần tử>} * \text{sizeof}(\text{<kiểu cơ sở>})$
- Bộ nhớ sử dụng phải **ít hơn 64KB** (65536 Bytes)
- Một dãy liên tục có chỉ số từ **0** đến **<tổng số phần tử>-1**

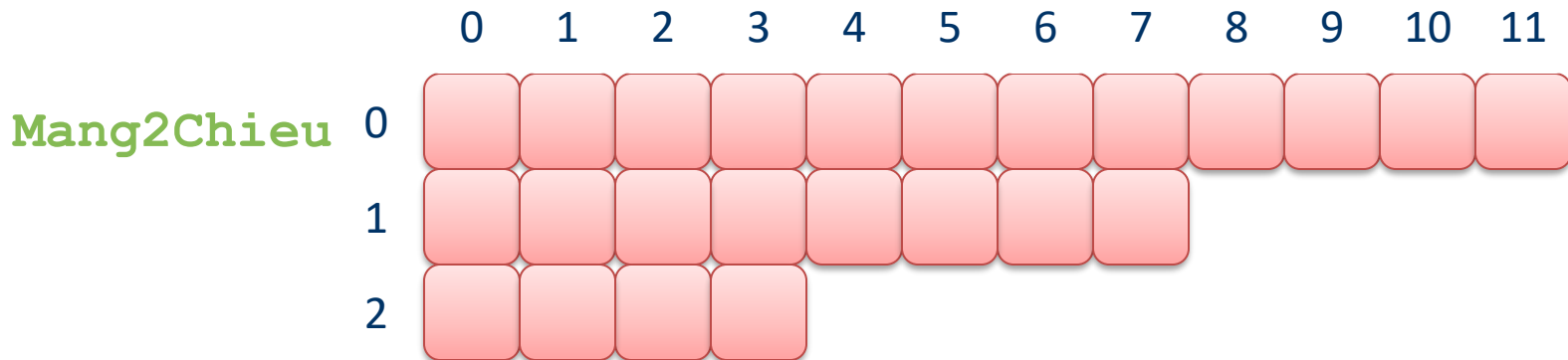
# Khai báo biến mảng (tường minh)

- Ví dụ

```
int Mang1Chieu[10];
```



```
int Mang2Chieu[3][4];
```



# Khai báo biến mảng (không tường minh)

- Cú pháp

- Không tường minh (thông qua khai báo kiểu)

```
typedef <kiểu cơ sở> <tên kiểu mảng> [<số phần tử>];  
typedef <kiểu cơ sở> <tên kiểu mảng> [<N1>]... [<Nn>];  
  
<tên kiểu mảng> <tên biến mảng>;
```

- Ví dụ

```
typedef int Mang1Chieu[10];  
typedef int Mang2Chieu[3][4];  
  
Mang1Chieu m1, m2, m3;  
Mang2Chieu m4, m5;
```

# Số phần tử của mảng

- Phải xác định cụ thể số phần tử ngay lúc khai báo, không được sử dụng biến hoặc hằng thường

```
int n1 = 10; int a[n1];  
const int n2 = 20; int b[n2];
```

- Nên sử dụng chỉ thị tiền xử lý **#define** để định nghĩa số phần tử mảng

```
#define n1 10  
#define n2 20  
int a[n1];           // ⇔ int a[10];  
int b[n1][n2];       // ⇔ int b[10][20];
```



# Khởi tạo giá trị lúc khai báo

- Gồm các cách sau
  - Khởi tạo giá trị cho mọi phần tử của mảng

```
int a[4] = {2912, 1706, 1506, 1904};
```



- Khởi tạo giá trị cho một số phần tử đầu mảng

```
int a[4] = {2912, 1706};
```



# Khởi tạo giá trị lúc khai báo

- Gồm các cách sau
  - Khởi tạo giá trị **0** cho mọi phần tử của mảng

```
int a[4] = {0};
```



- Tự động xác định số lượng phần tử

```
int a[] = {2912, 1706, 1506, 1904};
```



# Truy xuất đến một phần tử

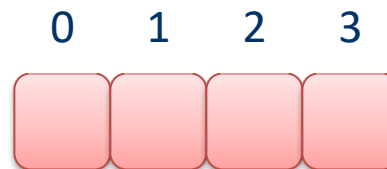
- Thông qua chỉ số

 `<tên biến mảng>[<gt cs1>][<gt cs2>]...[<gt csn>]`

- Ví dụ

– Cho mảng như sau

 `int a[4];`



– Các truy xuất

- Hợp lệ: `a[0]`, `a[1]`, `a[2]`, `a[3]`
- **Không hợp lệ**: `a[-1]`, `a[4]`, `a[5]`, ...

=> Cho kết thường không như mong muốn!

# Gán dữ liệu kiểu mảng

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

```
<biến mảng đích> = <biến mảng nguồn>; // sai  
<biến mảng đích>[<chỉ số thứ i>] = <giá trị>;
```

- Ví dụ

```
#define MAX 3  
typedef int MangSo[MAX];  
MangSo a = {1, 2, 3}, b;  
  
b = a;          // Sai  
for (int i = 0; i < 3; i++) b[i] = a[i];
```

# Một số lỗi thường gặp

- Khai báo không chỉ rõ số lượng phần tử
  - `int a[];`  $\Rightarrow$  `int a[100];`
- Số lượng phần tử liên quan đến biến hoặc hằng
  - `int n1 = 10; int a[n1];`  $\Rightarrow$  `int a[10];`
  - `const int n2 = 10; int a[n2];`  $\Rightarrow$  `int a[10];`
- Khởi tạo cách biệt với khai báo
  - `int a[4]; a = {2912, 1706, 1506, 1904};`  
 $\Rightarrow$  `int a[4] = {2912, 1706, 1506, 1904};`
- Chỉ số mảng không hợp lệ
  - `int a[4];`
  - `a[-1] = 1; a[10] = 0;`



# Truyền mảng cho hàm

- Truyền mảng cho hàm
  - Tham số kiểu mảng trong khai báo hàm **giống như khai báo biến mảng**
- 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** hoặc **sử dụng con trỏ**.
  - Mảng **có thể thay đổi nội dung** sau khi thực hiện hàm.

```
void SapXepTang(int a[]);  
void SapXepTang(int *a);
```

# Truyền mảng cho hàm

- Truyền mảng cho hàm
  - Số lượng phần tử thực sự truyền qua biến khác

```
void SapXepTang(int a[100], int n);  
void SapXepTang(int a[], int n);  
void SapXepTang(int *a, int n);
```

- Lời gọi hàm

```
voidNhapMang(int a[], int &n);  
voidXuatMang(int a[], int n);  
void main()  
{  
    int a[100], n;  
    NhapMang(a, n);  
    XuatMang(a, n);  
}
```

# Một số bài toán cơ bản

- Viết hàm thực hiện từng yêu cầu sau
  - Nhập mảng
  - Xuất mảng
  - Tìm kiếm một phần tử trong mảng
  - Kiểm tra tính chất của mảng
  - Tách mảng / Gộp mảng
  - Tìm giá trị nhỏ nhất/lớn nhất của mảng
  - Sắp xếp mảng giảm dần/tăng dần
  - Thêm/Xóa/Sửa một phần tử vào mảng



# Một số quy ước

- Số lượng phần tử

```
#define MAX 100
```

- Các hàm

- 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

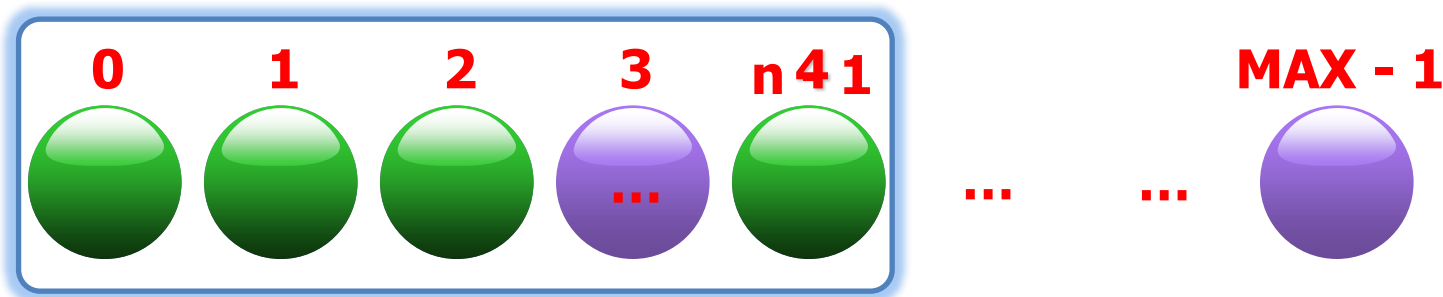
```
void HoanVi(int &x, int &y)
{
    int tam = x; x = y; y = tam;
}
```

```
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 mảng

- Yêu cầu
  - Cho phép nhập mảng **a**, số lượng phần tử **n**
- Ý tưởng
  - Cho trước một mảng có số lượng phần tử là **MAX**.
  - Nhập **số lượng phần tử thực sự n** của mảng.
  - Nhập từng phần tử cho mảng từ chỉ số **0** đến **n - 1**.



# Hàm Nhập Mảng

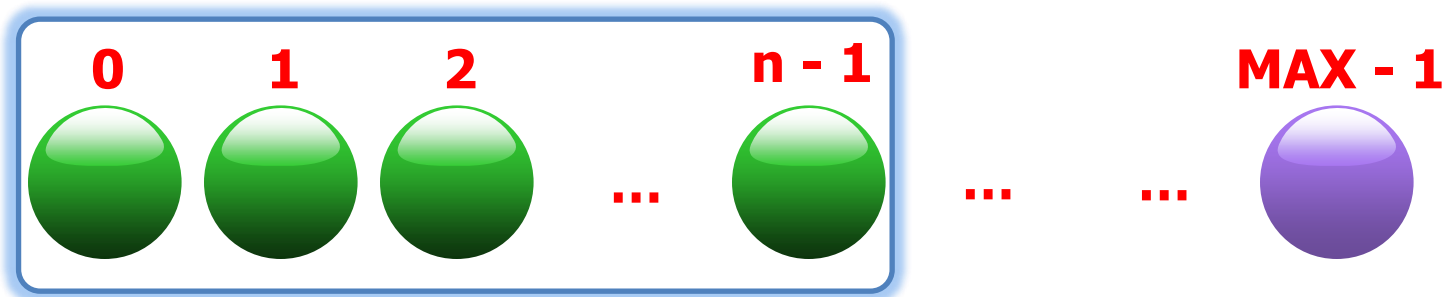
```
void NhapMang(int a[], int &n)
{
    printf("Nhap so luong phan tu n: ");
    scanf("%d", &n);

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



# Xuất mảng

- Yêu cầu
  - Cho trước mảng **a**, số lượng phần tử **n**. Hãy xuất nội dung mảng **a** ra màn hình.
- Ý tưởng
  - Xuất giá trị từng phần tử của mảng từ chỉ số **0** đến **n-1**.



# Hàm Xuất Mảng

```
void XuatMang(int a[], int n)
{
    printf("Noi dung cua mang la: ");

    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);

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

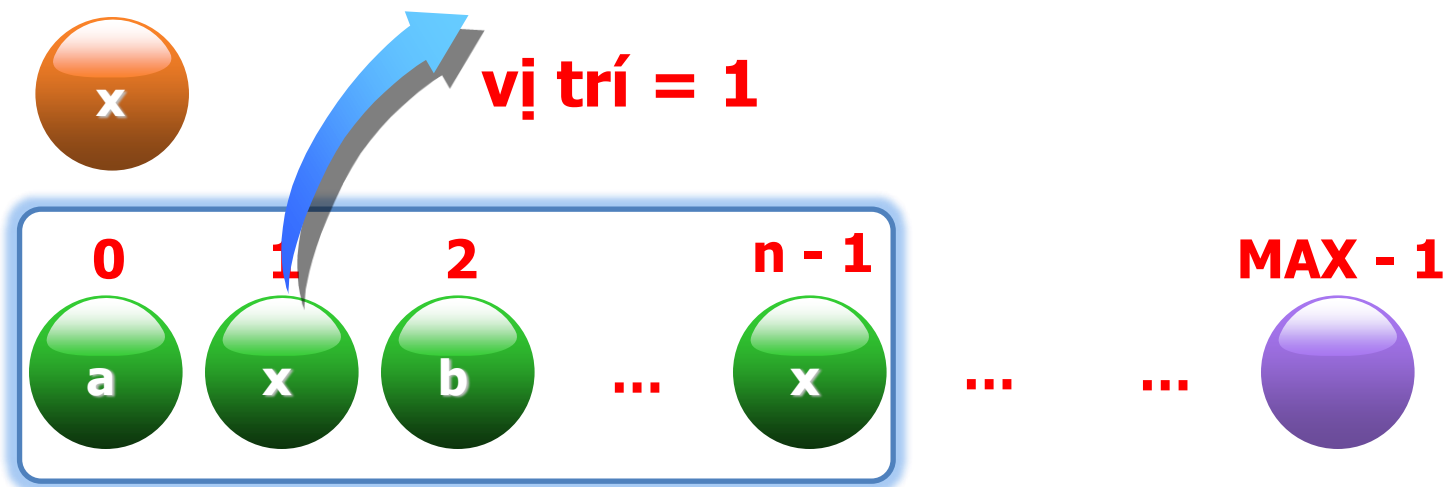
# Tìm kiếm một phần tử trong mảng

- Yêu cầu

- Tìm xem phần tử **x** có nằm trong mảng **a** kích thước **n** hay không? Nếu có thì nó nằm ở vị trí đầu tiên nào.

- Ý tưởng

- Xét từng phần của mảng **a**. Nếu phần tử đang xét bằng **x** thì trả về vị trí đó. Nếu không tìm được thì trả về **-1**.



# Hàm Tìm Kiếm (dùng while)

```
int TimKiem(int a[], int n, int x)
{
    int vt = 0;

    while (vt < n && a[vt] != x)
        vt++;

    if (vt < n)
        return vt;
    else
        return -1;
}
```



# Hàm Tìm Kiếm (dùng for)

```
int TimKiem(int a[], int n, int x)
{
    for (int vt = 0; vt < n; vt++)
        if (a[vt] == x)
            return vt;

    return -1;
}
```

# Kiểm tra tính chất của mảng

- Yêu cầu

- Cho trước mảng **a**, số lượng phần tử **n**. Mảng a có phải là mảng toàn các số nguyên tố hay không?

- Ý tưởng

- Cách 1: **Đếm số lượng số nguyên tố của mảng**. Nếu số lượng này **bằng đúng n** thì mảng toàn nguyên tố.
- Cách 2: **Đếm số lượng số không phải nguyên tố của mảng**. Nếu số lượng này **bằng 0** thì mảng toàn nguyên tố.
- Cách 3: **Tìm xem có phần tử nào không phải số nguyên tố không**. Nếu có thì mảng không toàn số nguyên tố.

# Hàm Kiểm Tra (Cách 1)

```
int KiemTra_C1(int a[], int n)
{
    int dem = 0;

    for (int i = 0; i < n; i++)
        if (LaSNT(a[i]) == 1) // có thể bỏ == 1
            dem++;

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

# Hàm Kiểm Tra (Cách 2)

```
int KiemTra_C2(int a[], int n)
{
    int dem = 0;

    for (int i = 0; i < n; i++)
        if (LaSNT(a[i]) == 0) // Có thể sử dụng !
            dem++;

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

# Hàm Kiểm Tra (Cách 3)

```
int KiemTra_C3(int a[], int n)
{
    for (int i = 0; i < n ; i++)
        if (LaSNT(a[i]) == 0)
            return 0;

    return 1;
}
```

# Tách các phần tử thỏa điều kiện

- Yêu cầu
  - Cho trước mảng **a**, số lượng phần tử **na**. Tách các số nguyên tố có trong mảng a vào mảng b.
- Ý tưởng
  - Duyệt từ phần tử của mảng a, nếu đó là **số nguyên tố** thì đưa vào mảng b.

# Hàm Tách Số Nguyên Tố

```
void TachSNT(int a[], int na, int b[], int &nb)
{
    nb = 0;

    for (int i = 0; i < na; i++)
        if (LaSNT(a[i]) == 1)
        {
            b[nb] = a[i];
            nb++;
        }
}
```

# Tách mảng thành 2 mảng con

- Yêu cầu

- Cho trước mảng **a**, số lượng phần tử **na**. Tách mảng **a** thành 2 mảng **b** (chứa số nguyên tố) và mảng **c** (các số còn lại).

- Ý tưởng

- Cách 1: viết 1 hàm tách các số nguyên tố từ mảng a sang mảng b và 1 hàm tách các số không phải nguyên tố từ mảng a sang mảng c.
- Cách 2: Duyệt từ phần tử của mảng a, nếu đó là **số nguyên tố thì đưa vào mảng b, ngược lại đưa vào mảng c.**



# Hàm Tách 2 Mảng

```
void TachSNT2(int a[], int na,  
              int b[], int &nb, int c[], int &nc)  
{  
    nb = 0;  
    nc = 0;  
  
    for (int i = 0; i < na; i++)  
        if (LaSNT(a[i]) == 1)  
        {  
            b[nb] = a[i]; nb++;  
        }  
        else  
        {  
            c[nc] = a[i]; nc++;  
        }  
}
```

# Gộp 2 mảng thành một mảng

- Yêu cầu
  - Cho trước mảng **a**, số lượng phần tử **na** và mảng **b** số lượng phần tử **nb**. Gộp 2 mảng trên theo thứ tự đó thành mảng **c**, số lượng phần tử **nc**.
- Ý tưởng
  - Chuyển các phần tử của mảng a sang mảng c  
 $\Rightarrow nc = na$
  - Tiếp tục đưa các phần tử của mảng b sang mảng c  
 $\Rightarrow nc = nc + nb$

# Hàm Gộp Mảng

```
void GopMang(int a[], int na, int b[], int nb,  
             int c[], int &nc)  
{  
    nc = 0;  
  
    for (int i = 0; i < na; i++)  
    {  
        c[nc] = a[i]; nc++; // c[nc++] = a[i];  
    }  
  
    for (int i = 0; i < nb; i++)  
    {  
        c[nc] = b[i]; nc++; // c[nc++] = b[i];  
    }  
}
```

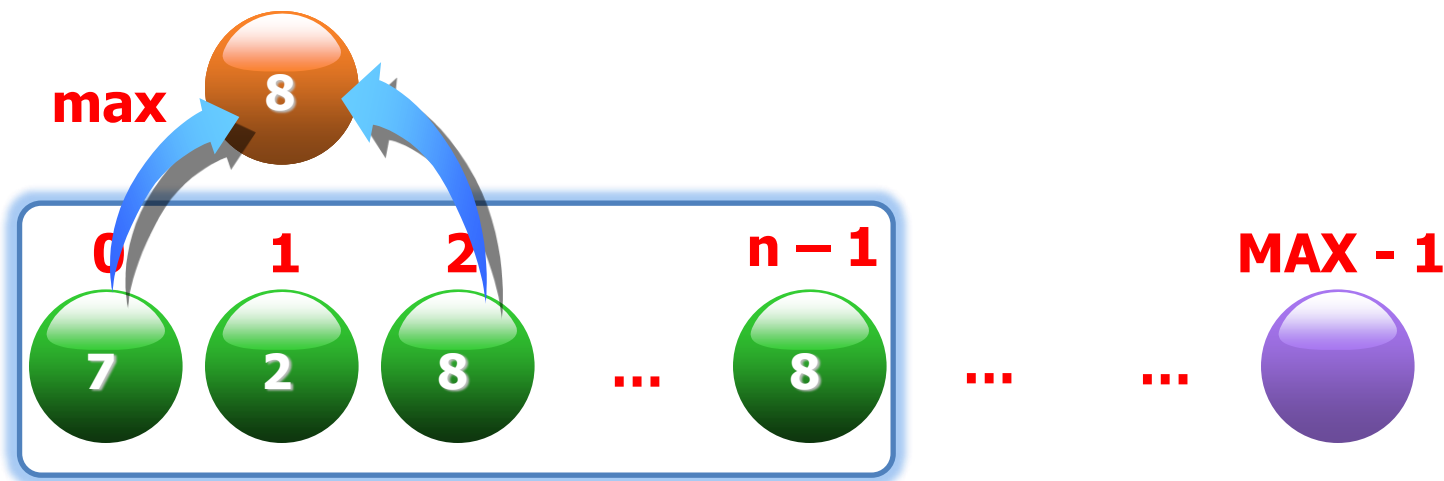
# Tìm giá trị lớn nhất của mảng

- Yêu cầu

- Cho trước mảng **a** có **n** phần tử. Tìm giá trị lớn nhất trong **a** (gọi là **max**)

- Ý tưởng

- Giả sử giá trị **max hiện tại** là giá trị phần tử đầu tiên **a[0]**
- Lần lượt kiểm tra các phần tử còn lại để cập nhật **max**.



# Hàm tìm Max

```
int TimMax(int a[], int n)
{
    int max = a[0];

    for (int i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];

    return max;
}
```

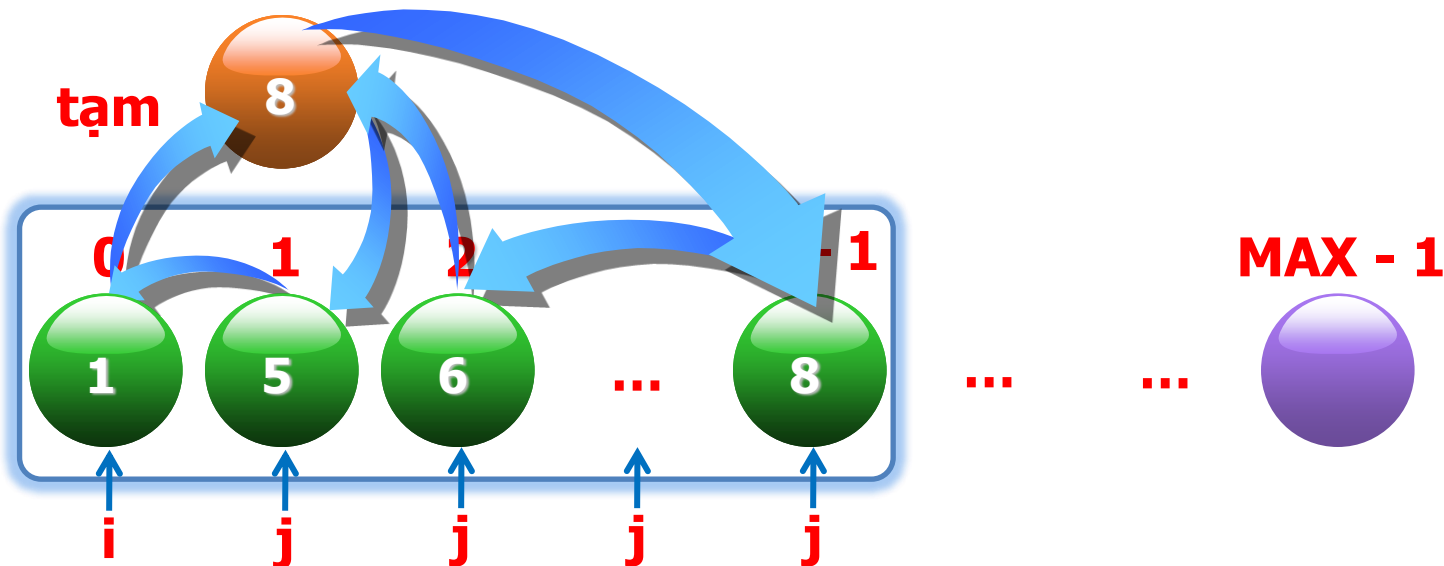
# Sắp xếp mảng thành tăng dần

- Yêu cầu

- Cho trước mảng **a** kích thước **n**. Hãy sắp xếp mảng **a** đó sao cho các phần tử có giá trị **tăng dần**.

- Ý tưởng

- Sử dụng 2 biến **i** và **j** để so sánh tất cả cặp phần tử với nhau và hoán vị các cặp **ngược thế** (sai thứ tự).



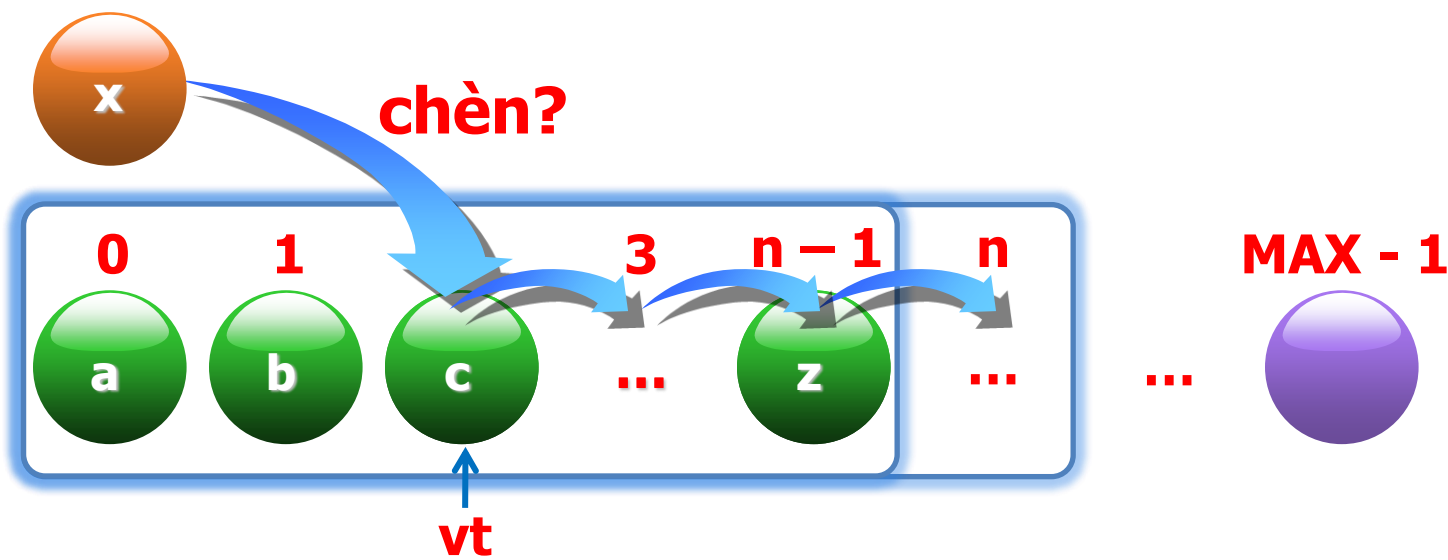
# Hàm Sắp Xếp Tăng

```
void SapXepTang(int a[], int n)
{
    int i, j;

    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (a[i] > a[j])
                HoanVi(a[i], a[j]);
        }
    }
}
```

# Thêm một phần tử vào mảng

- Yêu cầu
  - Thêm phần tử **x** vào mảng **a** kích thước **n** tại vị trí **vt**.
- Ý tưởng
  - “Đẩy” các phần tử bắt đầu tại vị trí **vt** sang phải **1 vị trí**.
  - Đưa **x** vào vị trí **vt** trong mảng.
  - Tăng **n** lên **1 đơn vị**.





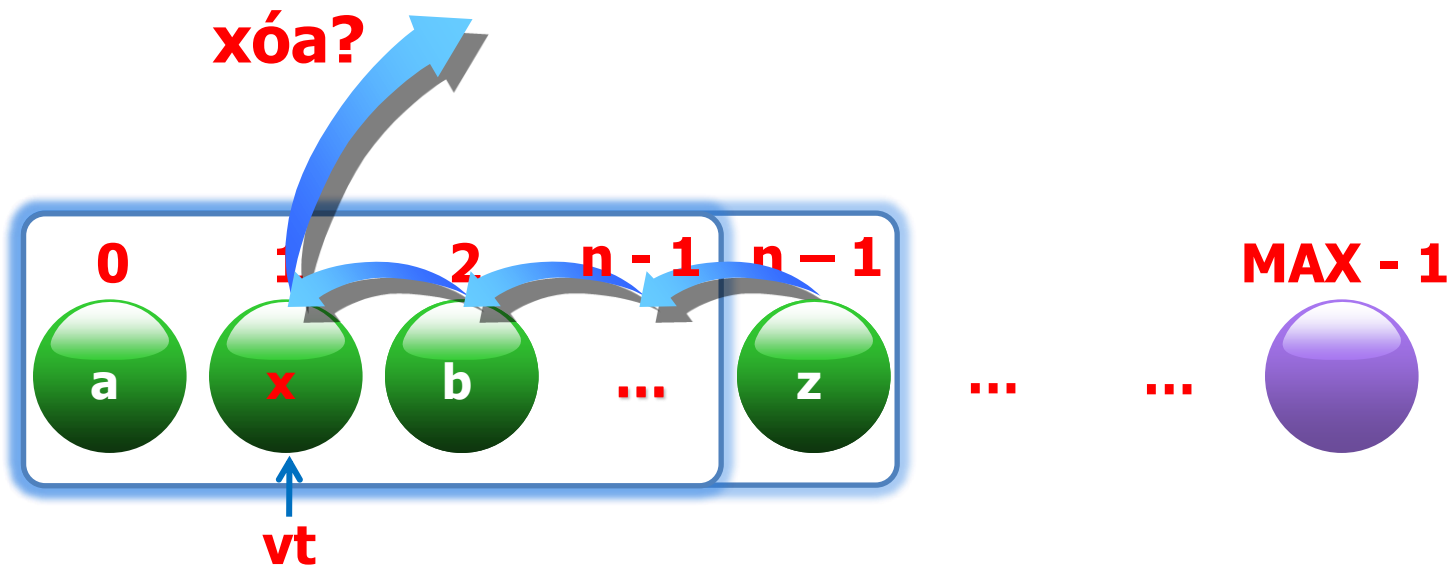
# Hàm Thêm

```
void Them(int a[], int &n, int vt, int x)
{
    if (vt >= 0 && vt <= n)
    {
        for (int i = n; i > vt; i--)
            a[i] = a[i - 1];

        a[vt] = x;
        n++;
    }
}
```

# Xóa một phần tử trong mảng

- Yêu cầu
  - Xóa một phần tử trong mảng **a** kích thước **n** tại vị trí **vt**
- Ý tưởng
  - “Kéo” các phần tử bên phải vị trí **vt** sang trái **1 vị trí**.
  - Giảm **n** xuống **1 đơn vị**.



# Hàm Xóa

```
void Xoa(int a[], int &n, int vt)
{
    if (vt >= 0 && vt < n)
    {
        for (int i = vt; i < n - 1; i++)
            a[i] = a[i + 1];

        n--;
    }
}
```



# Bài tập

## 1. Các thao tác nhập xuất

- a. Nhập mảng
- b. Xuất mảng

## 2. Các thao tác kiểm tra

- a. Mảng có phải là mảng toàn chữ
- b. Mảng có phải là mảng toàn số nguyên tố
- c. Mảng có phải là mảng tăng dần

## 3. Các thao tác tính toán

- a. Có bao nhiêu số chia hết cho 4 nhưng không chia hết cho 5
- b. Tổng các số nguyên tố có trong mảng

## 4. Các thao tác tìm kiếm

- a. Vị trí cuối cùng của phần tử  $x$  trong mảng
- b. Vị trí số nguyên tố đầu tiên trong mảng nếu có
- c. Tìm số nhỏ nhất trong mảng
- d. Tìm số dương nhỏ nhất trong mảng

## 5. Các thao tác xử lý

- a. Tách các số nguyên tố có trong mảng a đưa vào mảng b.
- b. Tách mảng a thành 2 mảng b (chứa các số nguyên dương) và c (chứa các số còn lại)
- c. Sắp xếp mảng giảm dần
- d. Sắp xếp mảng sao cho các số dương đứng đầu mảng giảm dần, kế đến là các số âm tăng dần, cuối cùng là các số 0.



# Bài tập

## 6. Các thao tác thêm/xóa/sửa

- a. Sửa các số nguyên tố có trong mảng thành số 0
- b. Chèn số 0 đằng sau các số nguyên tố trong mảng
- c. Xóa tất cả số nguyên tố có trong mảng



The End