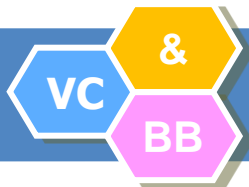


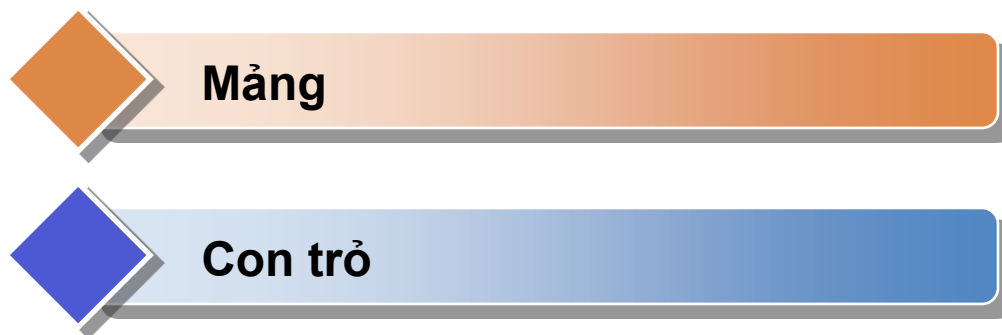
TIN HỌC CƠ SỞ 2

CHƯƠNG 5: CẤU TRÚC DỮ LIỆU KIỂU MẠNG





Nội dung



❖ 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**.

1

Khái niệm

2

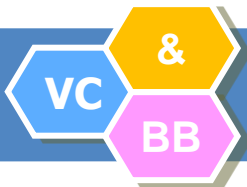
Khai báo mảng

3

Duyệt mảng

4

Nhập xuất mảng



Mảng 1 chiều

Khái niệm

Mảng là một tập hợp nhiều phần tử có cùng kiểu giá trị và chung một tên. Mỗi phần tử của mảng biểu diễn được 1 giá trị

Ví dụ: Mảng A có 5 phần tử, các phần tử là kiểu số nguyên
 $A = \{3, 5, 10, 9, 1\}$

Phần tử:

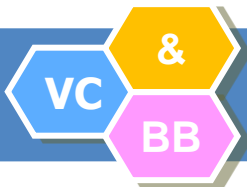
$A[0] = 3$

$A[1] = 5$

$A[2] = 10$

$A[3] = 9$

$A[4] = 1$



Khai báo mảng 1 chiều

Khai báo tường minh

`<Kiểudl> <tên mảng> [số lượng phần tử];`

Ví dụ: `int A[10];`

`float X[20];`

Khai báo không tường minh

`typedef <kiểudl> <tên kiểu mảng> [<số phần tử>];`

`<tên kiểu mảng> <tên biến mảng>;`

Ví dụ: `typedef int Mang1Chieu[10];`

`Mang1Chieu m1, m2, m3;`

❖ Thông qua chỉ số

 <tên biến mảng> [<chỉ số>]

❖ Ví dụ

- Cho mảng như sau

 `int a[4];`

- Các truy xuất
 - Hợp lệ: `a[0]`, `a[1]`, `a[2]`, `a[3]`





Khai báo số phần tử của mảng

❖ Khai báo ở ngoài chương trình:

Cách 1: `const int n2 = 20; int b[n2];`

Cách 2: `#define n1 10`
`int a[n1];` `// ⇔ int a[10];`

Khởi tạo giá trị ngẫu nhiên cho mảng

- Cú pháp:

```
int rand ( )
```

- Nếu muốn tạo số n có giá trị từ a đến b dùng cú pháp:

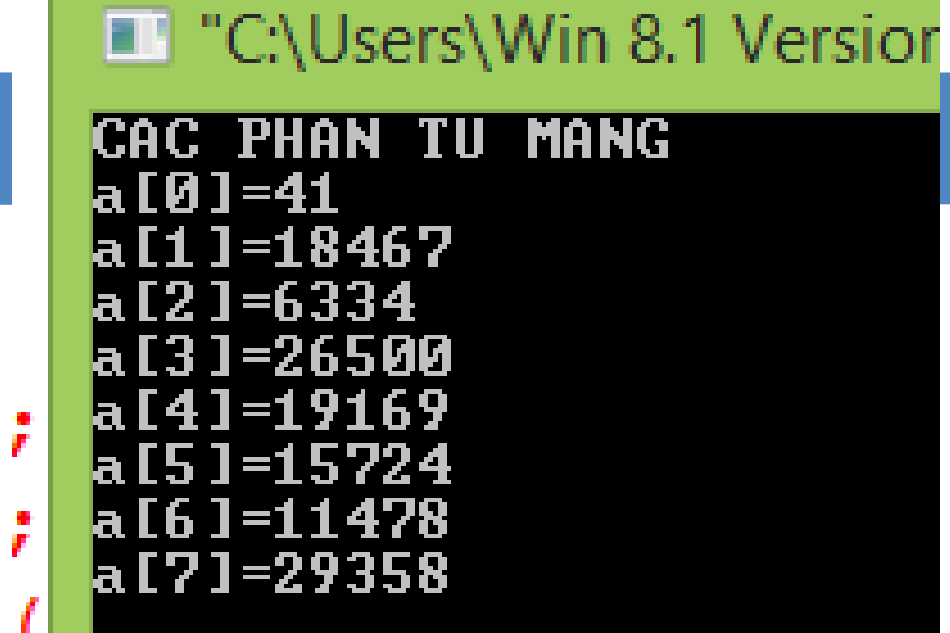
```
int n = rand() % (b - a + 1) + a;
```

- Ví dụ tạo số ngẫu nhiên từ 1 đến 50:

```
int a = rand() % 50 + 1;
```

- Để sử dụng các hàm trên thì trong chương trình phải khai báo thư viện **<stdlib.h>**

```
int i,n;
    int a[200];
scanf("%d",&n);
for( i=0;i<n;i++)
    a[i]=rand();
printf("CAC PHAN TU MANG\n");
for( i=0;i<n;i++)
{
    printf("a[%d]=%d\n",i,a[i]) ;
}
```



```
"C:\Users\Win 8.1 Version"
CAC PHAN TU MANG
a[0]=41
a[1]=18467
a[2]=6334
a[3]=26500
a[4]=19169
a[5]=15724
a[6]=11478
a[7]=29358
```

NHẬP MẢNG 1 CHIỀU

Cách 1: Đọc các phần tử mảng từ bàn phím

```
for(int i=0;i<n;i++)  
{  
    printf("Nhap phan tu thu %d: ",i+1);  
    //Cach 2: printf("a[%d]= ",i);  
    scanf("%d",&a[i]);  
}
```

Cách 2: Khai báo và gán giá trị cho mảng

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a[6]={0,5,9,8,7,3};
    printf("\n=====CAC PHAN TU
CUA MANG VUA NHAP=====");
    for(int i=0;i<6;i++)
        printf("%d\t",a[i]);
    getch();
}
```

Cách 3: Sinh các số ngẫu nhiên cho mảng

```
main()
{
    int a[50];
    int n;
    printf("So phan tu mang n= ");scanf("%d",&n);
    //sinh cac so ngau nhien
    for(int i=0;i<n;i++)
        a[i]=rand()%50+1;
    printf("\n=====CAC PHAN TU CUA MANG VUA NHAP=====\n");
    for(int i=0;i<n;i++)
        printf("%d\t",a[i]);
    getch();
}
```

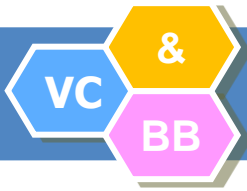
Xuất mảng một chiều

Xuất xuôi:

```
for( i=0;i<n;i++)  
{  
    printf("a[%d]=%d\n",i,a[i]) ;  
}
```

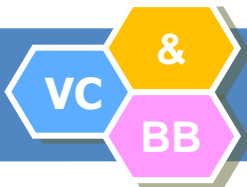
Xuất ngược:

```
for( i=n-1;i>=0;i--)  
{  
    printf("a[%d]=%d\n",i,a[i]) ;  
}
```



Một số lỗi thường gặp

- ❖ Khai báo không chỉ rõ số lượng phần tử
 - `int a[]; => int a[100];`
- ❖ Số lượng phần tử liên quan đến biến hoặc hằng
 - `int n1 = 10; int a[n1]; => int a[10];`
 - `const int n2 = 10; int a[n2]; => int a[10];`
- ❖ Khởi tạo cách biệt với khai báo
 - `int a[4]; a = {2912, 1706, 1506, 1904};`
`=> int a[4] = {2912, 1706, 1506, 1904};`



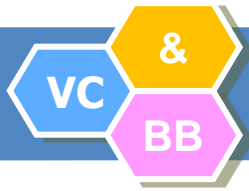
Ví dụ

- Nhập vào một mảng gồm n số nguyên
- Xuất ra giá trị trung bình cộng của các phần tử trong mảng vừa nhập



Mảng hai chiều

- 1 Khái niệm
- 2 Khai báo mảng
- 3 Duyệt mảng
- 4 Nhập xuất mảng



Khái niệm

- ❖ Một mảng nhiều chiều là một mảng mà những phần tử của nó được xác định bằng nhiều chỉ số.
- ❖ Mảng 2 chiều là mảng nhiều chiều đơn giản và sử dụng nhiều nhất.
- ❖ Mảng 2 chiều giống như một bảng, gồm nhiều dòng và nhiều cột

Giải thích mảng hai chiều

Mảng 2 chiều:
Một nhóm các phần tử có cùng kiểu, chung tên.

Các phần tử được xác định bằng số dòng và số cột.

chỉ số
dòng

Chỉ số
cột

	0	1	2	3	4
0	5	7	9	2	3
1	1	5	7	4	6
2	1	8	9	0	3
3	2	5	6	3	4

m[2][3]

i và j là số nguyên

Khai báo mảng hai chiều

❖ Khai báo mảng hai chiều – khai báo tường minh

<kiểu> arrayName[rows][columns];

- **rows**: số dòng
- **columns**: số cột

❖ Ví dụ: Khai báo mảng số nguyên 3 dòng 4 cột

int a[3][4]

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

Khai báo biến mảng 2 chiều

- Không tường minh (thông qua kiểu)

```
typedef <kiểu cơ sở> <tên kiểu> [<N1>] [<N2>] ;  
<tên kiểu> <tên biến 1>, <tên biến 2>;
```

Ví dụ:

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

```
MaTran10x20 a, b;  
MaTran5x10 c;  
MaTran10x20 d;
```

NHẬP MẢNG 2 CHIỀU

Cách 1: Đọc các phần tử mảng từ bàn phím

```
for(int i=0;i<d;i++)  
for(int j=0;j<c;j++)  
{  
    printf("Doc phan tu % d, %d: ",i,j);  
    //a[i][j]  
    scanf("%d",&a[i][j]);  
}
```

Cách 2: Khai báo và gán giá trị cho mảng

Khởi tạo = khai báo + gán giá trị cho mảng

```
<kiểu> arrayName[][columns] = {  
    {value1,value2,...,valueN},  
    {value1,value2,...,valueN},  
    {...},  
    {value1,value2,...,valueN}};
```

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


Cách 3: Sinh các số ngẫu nhiên cho mảng

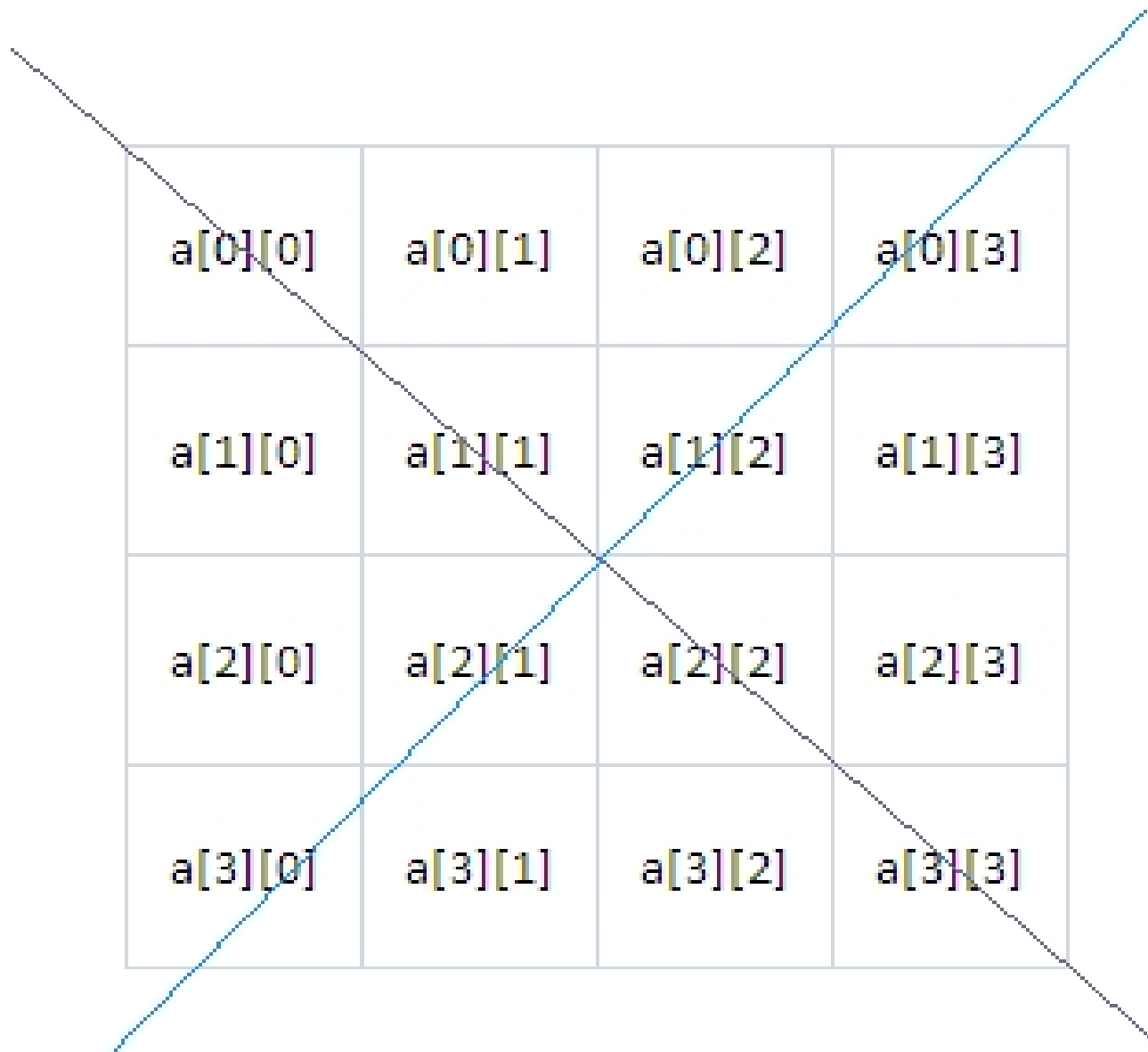
```
main(){
    int a[50][50];
    int n,m;
    printf("So phan tu mang n= ");scanf("%d",&n);
    //sinh cac so ngau nhien
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            a[i][j]=rand()%50+1;
    printf("\nCAC PHAN TU CUA MANG VUA NHAP\n");
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++){
            printf("%d\t",a[i][j]);
            printf("\n");
        }
}
```

Duyệt mảng hai chiều

```
void nhapmang(int a[][50],
int n,int m)
{
    int i,j;
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        {
            printf("a[%d][%d]= ",i,j);
            scanf("%d",&a[i][j]);
        }
}
```

```
void xuatmang(int a[][50], int
n,int m)
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            printf("%d\t ",a[i][j]);
        }
        printf("\n");
    }
}
```

Các phép tính đặc trưng của ma trận



$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$
$a[1][0]$	$a[1][1]$	$a[1][2]$	$a[1][3]$
$a[2][0]$	$a[2][1]$	$a[2][2]$	$a[2][3]$
$a[3][0]$	$a[3][1]$	$a[3][2]$	$a[3][3]$

- Phần tử nằm trên đường chéo chính có chỉ số: $i = j$
- Tương tự đường chéo phụ sẽ là: $n-1 = i + j$
- Các phần tử thuộc tam giác trên: $i < j$
- Các phần tử thuộc tam giác dưới: $i > j$

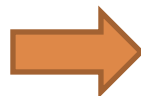
Ngoài ra còn có các bài tập như:

- Tìm tổng các phần tử trong mảng
- Max, min trong mảng.
- Đếm số nguyên tố trong mảng.
- Sắp xếp mảng.
- Tìm tần số xuất hiện của một số n bất kỳ trong mảng.
- Xóa phần tử trong mảng.

Bài tập ví dụ

- ❖ Viết chương trình nhập vào một ma trận vuông $N \times N$ với các phần tử của ma trận được tạo ngẫu nhiên. Yêu cầu:
- ❖ Xuất ma trận theo thứ tự tăng dần xoắn ốc cùng chiều kim đồng hồ
- ❖ Ví dụ nhập ma trận vuông ngẫu nhiên

10	27	30	24	32
31	49	12	35	38
16	29	44	27	49
50	12	21	40	34
11	18	17	40	15

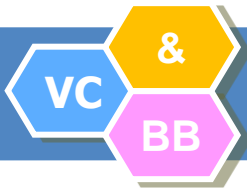


10	11	12	12	15
32	34	35	38	16
31	49	50	40	17
30	49	44	40	18
29	27	27	24	21

Bài tập ví dụ

❖ Hướng dẫn:

- Để xuất được ma trận vuông theo yêu cầu bài toán chúng ta cần chuyển về bài toán mảng 1 chiều
- Sau đó sắp mảng 1 chiều này tăng dần
- Cuối cùng là chuyển mảng một chiều này vào ma trận vuông và xuất ma trận vuông ra màn hình.
 - NhapMang
 - XuatMang
 - SapXepTang
 - Chuyen(int a[],int b[][50],int n)



Bài tập ví dụ

```
//ham nhap mang ngau nhien
void NhapMang(int a[],int n)
{
    //tao so random tu [10,50]
    //rand() % (b - a + 1) + a
    for(int i=0;i<n*n;i++)
        a[i]=rand() % 41 + 10;
}

//ham xuat mang kieu ma tran
void XuatMang(int a[],int n)
{
    int i,j;
    for (i=0;i<n;i++)
    {
        for (j=0;j<n;j++)
            printf(" %d",a[i*n+j]);
        printf("\n");
    }
}
```

```
void Sapxep(int a[],int n)
{
    int i,j,tam;
    for(i=0;i<n*n-1;i++)
        for(j=i+1;j<n*n;j++)
            if (a[i]>a[j])
            {
                tam=a[i];
                a[i]=a[j];
                a[j]=tam;
            }
}
```

Bài tập ví dụ

❖ Hướng dẫn:

- Chuyen(int a[],int b[][50],int n)
- Duyệt 4 cạnh của ma trận

```
void Chuyen(int a[],int b[][50],int n)
```

```
{ int i=0,j=0,k,l=0,tam;
```

```
  while (i<n*n)
```

```
  {
```

```
    // n = 5, l = 0
```

```
    for(j=l;j<n-l;j++)
```

```
        b[l][j]=a[i++]; //0,1 0,2 0,3 0,4
```

```
    for(k=l+1;k<n-l;k++)
```

```
        b[k][n-l-1]=a[i++]; //1,4 2,4 3,4 4,4
```

```
    for(j=n-l-2;j>=l;j--)
```

```
        b[n-l-1][j]=a[i++]; //4,3 4,2, 4,1 4,0
```

```
    for(k=n-l-2;k>l;k--)
```

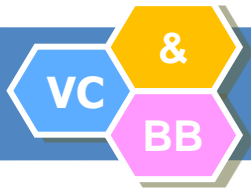
```
        b[k][l]=a[i++]; //3,0 2,0 1,0 0,0
```

```
    l++;
```

```
  }
```

```
}
```

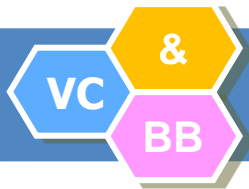
4	1	1	1	1
4	4	1	1	2
4	4		2	2
4	3	3	2	2
3	3	3	3	2



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, cann = sqrt(n), dem=0;
    for (i = 1; i <= cann; i++)
        if (n%i == 0)
            dem++;

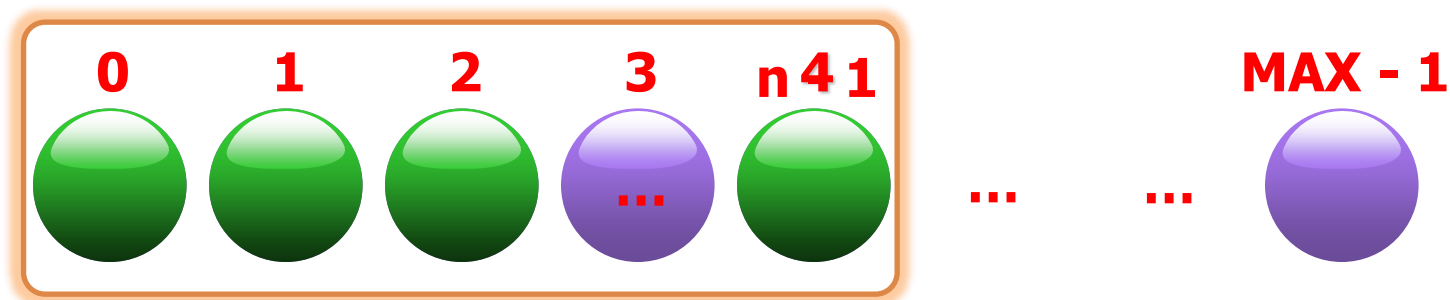
    if (dem == 2)
        return 1;
    else return 0;
}
```

❖ 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);

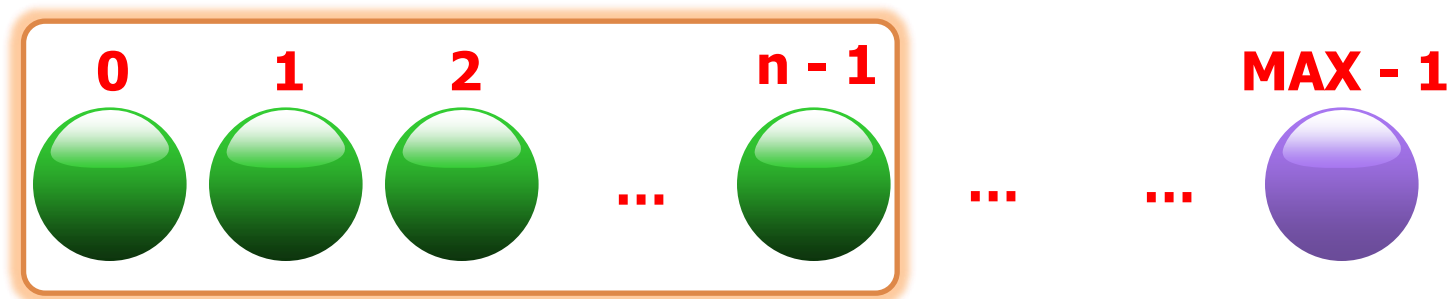
    for (int i = 0; i < n; i++)
    {
        printf("Nhap phan tu thu %d: ", i);
        scanf("%d", &a[i]);
    }
}
```

❖ 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;
}
```

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

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.

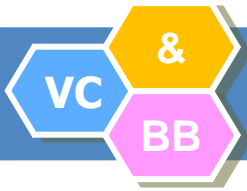
❖ Ý tưởng

- 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

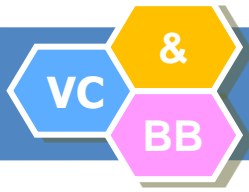
- Cho trước mảng **a**, số lượng phần tử **na**. Tách mảng **a** thành 2 mảng **b** (chứa số nguyên tố) và mảng **c** (các số còn lại).

❖ 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];  
    }  
}
```

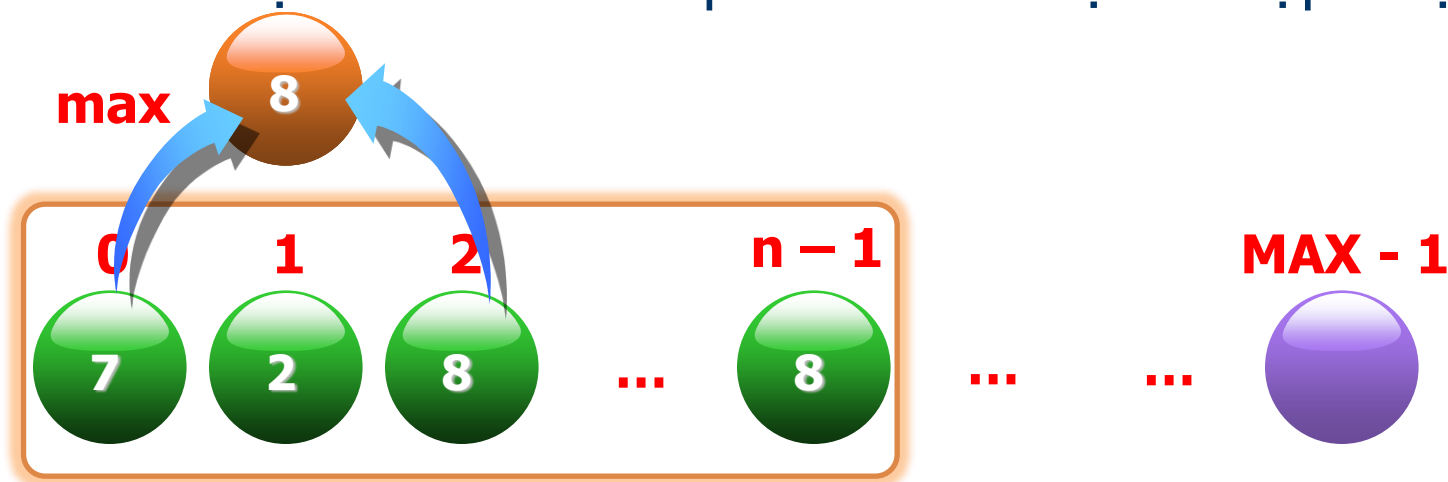
Tìm giá trị lớn nhất của mảng

❖ 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;
}
```

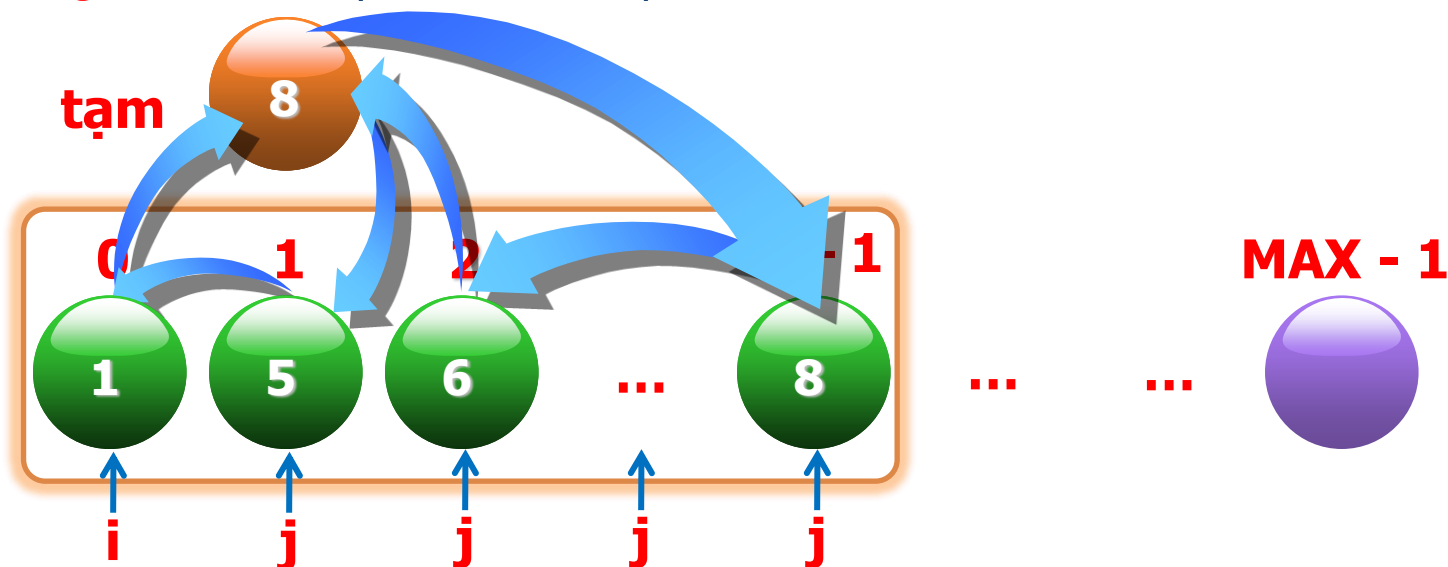
Sắp xếp mảng thành tăng dần

❖ 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]);
        }
    }
}
```

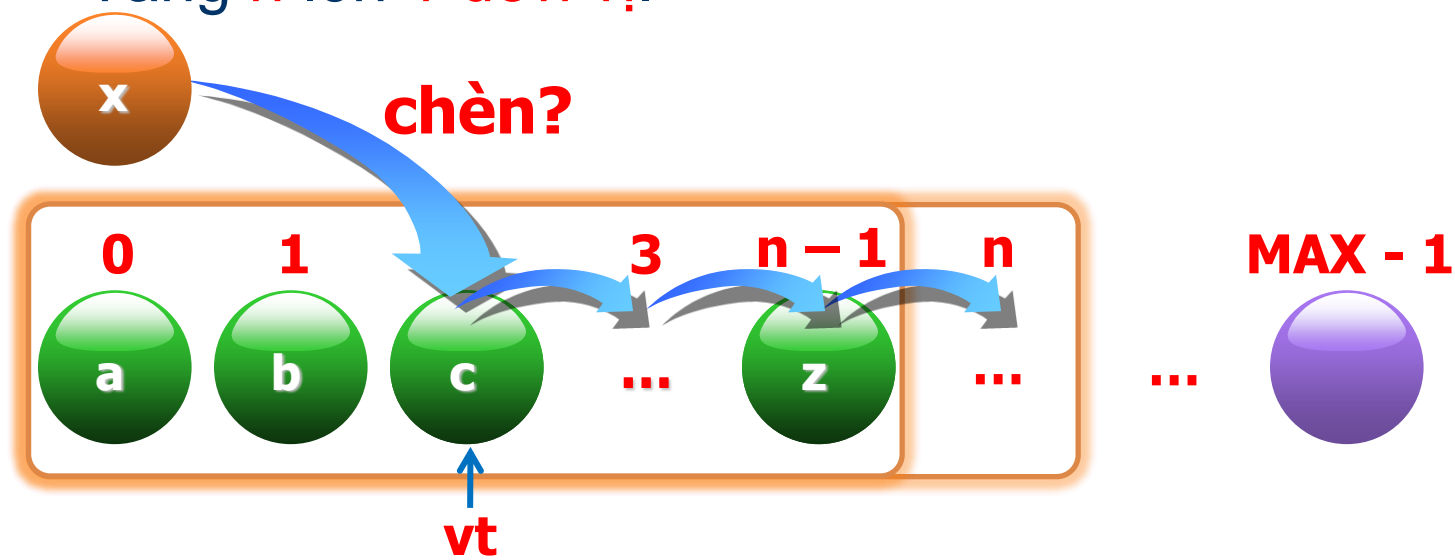

Thêm một phần tử vào mảng

❖ 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++;
    }
}
```

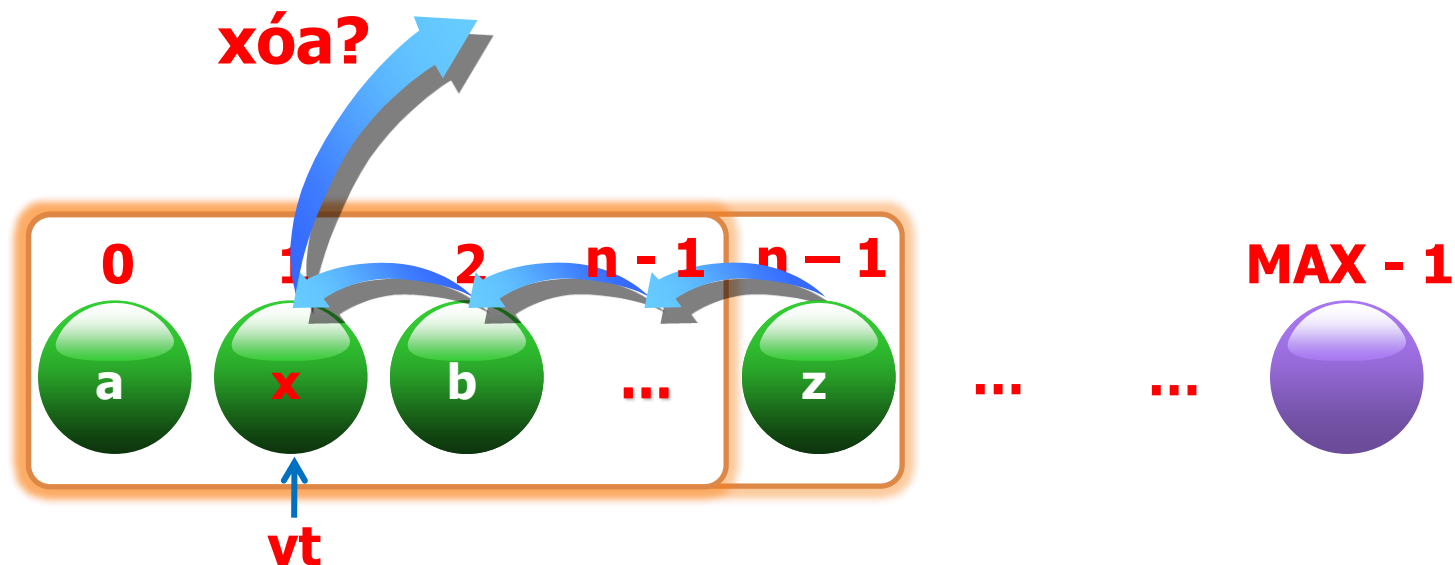
Xóa một phần tử trong mảng

❖ Yêu cầu

- Xóa một phần tử trong mảng **a** kích thước **n** tại vị trí **vt**

❖ Ý tưởng

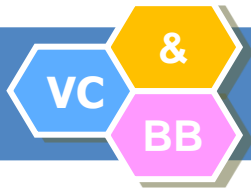
- “Kéo” các phần tử bên phải vị trí **vt** sang trái **1** vị trí.
- Giảm **n** xuống **1** đơn vị.



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

❖ BTVN ngày 08/04 trên code.ptit.edu.vn

5.2. CON TRỎ

1

Địa chỉ của biến

2

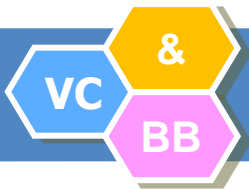
Khái niệm con trỏ

3

Khai báo con trỏ

4

Toán tử con trỏ



Địa chỉ của biến

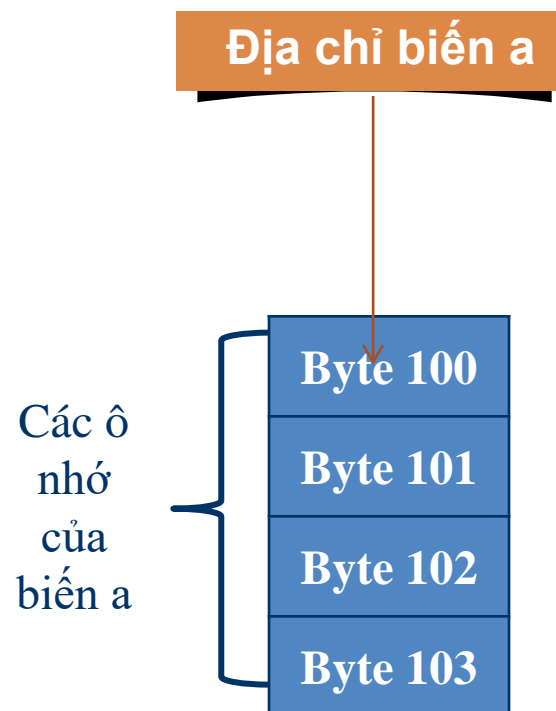
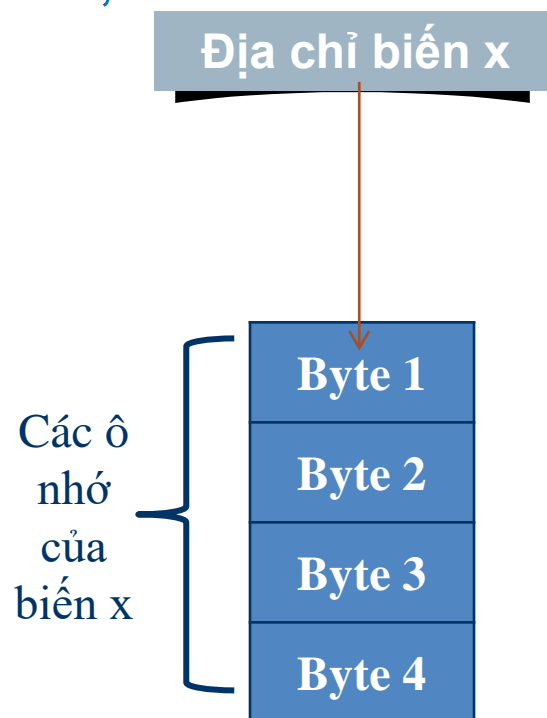
- ❖ Thông tin của một biến bao gồm:
 - Tên biến
 - Kiểu dữ liệu của biến
 - Giá trị của biến
- ❖ Mỗi biến sẽ được lưu trữ tại một vị trí xác định trong ô nhớ, nếu kích thước của biến có nhiều byte thì máy tính sẽ cấp phát một dãy các byte liên tiếp nhau, địa chỉ của biến sẽ **lưu byte đầu tiên** trong dãy các byte này

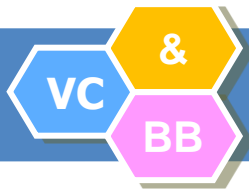
Địa chỉ của biến

❖ Ví dụ khai báo:

```
int x;
```

```
int a;
```





Địa chỉ của biến

❖ Địa chỉ của biến luôn luôn là một số nguyên (hệ thập lục phân) dù biến đó chứa giá trị là số nguyên, số thực hay ký tự, ...

❖ Cách lấy địa chỉ của biến

&tênbiến

❖ Ví dụ:

```
int x=7;
```

```
float y=10.5;
```

```
printf("Dia chi cua bien x =",&x);
```

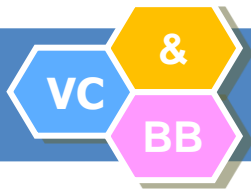
```
printf("Dia chi cua bien y =",&y);
```

Khái niệm con trỏ

❖ Con trỏ là 1 biến chứa một địa chỉ của biến khác



Con trỏ trỏ đến vùng nhớ của biến a



Khai báo con trỏ

❖ **Cú pháp:**

<Type> *Tên biến con trỏ;

type: xác định kiểu dữ liệu của biến mà con trỏ trỏ đến

Sau khai báo pointer variable trỏ tới NULL (chưa trỏ tới 1 đối tượng).

Ví dụ:

int *a;  a

Các biểu tượng được sử dụng trong con trỏ

Biểu tượng	Tên	Mô tả
&	Địa chỉ của toán tử	Xác định địa chỉ của một biến.
*	Toán tử liên kết.	Truy cập đến giá trị của địa chỉ.

Toán tử địa chỉ (pointer operators)

- ❖ **Toán tử &** là toán tử 1 ngôi, trả về địa chỉ của một biến
- ❖ Toán tử & dùng để gán địa chỉ của biến cho biến con trỏ
- ❖ Cú pháp:

<Tên biến con trỏ> = &<Tên biến>

- ❖ Ví dụ:

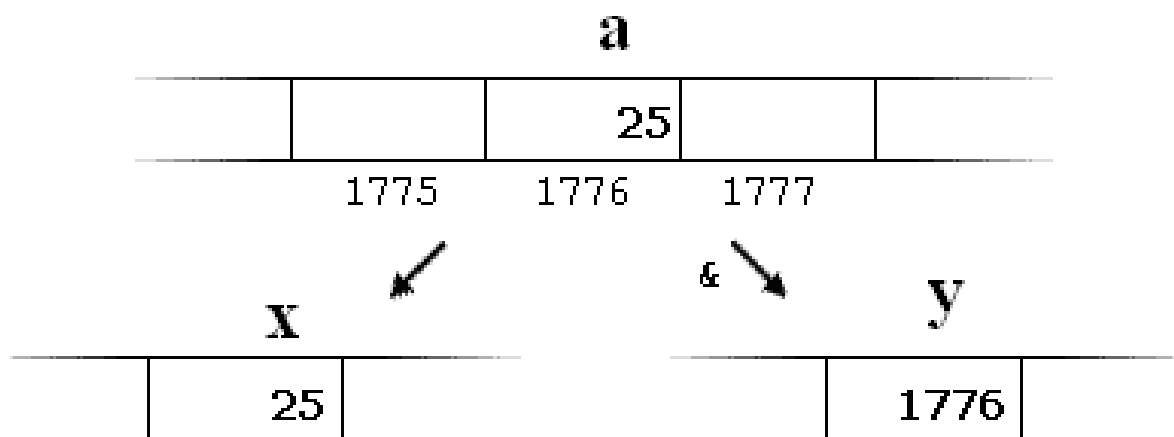
int a = 25, x;

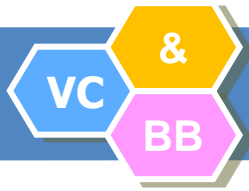
int *y;

x = a;

y = &a;

~~y = a;~~





Toán tử liên kết

❖ **Toán tử *** là toán tử một ngôi trả về giá trị tại địa chỉ con trỏ trỏ đến.

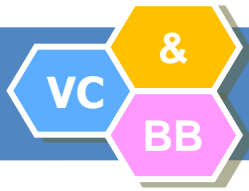
❖ Cú pháp:

* <Tên biến con trỏ>

Ví dụ:

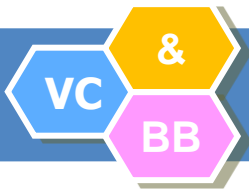
a=*p ;

a=p;//sai



CÁC THAO TÁC TRÊN CON TRỎ

- ❖ Phép gán
- ❖ Phép tăng giảm địa chỉ



Lệnh gán con trỏ

- ❖ Có thể dùng phép gán để gán giá trị của một con trỏ cho một con trỏ khác có cùng kiểu

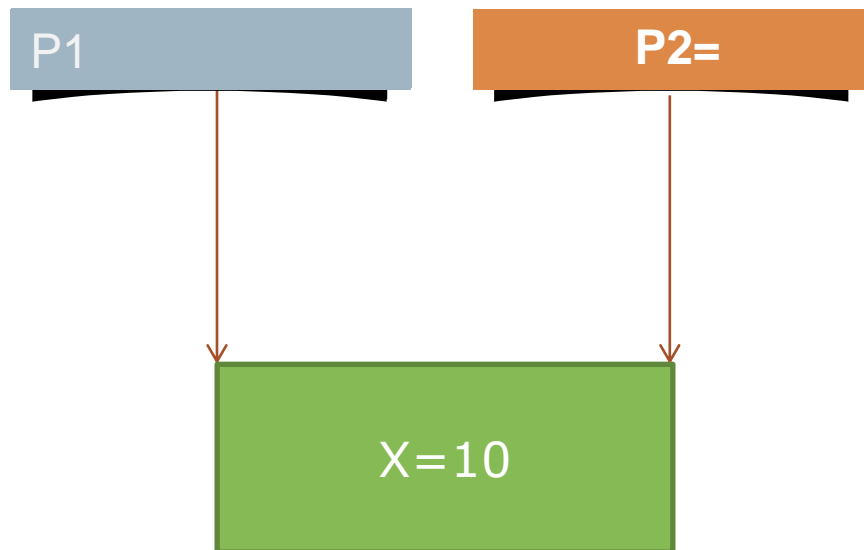
Ví dụ:

```
int x=10;  
int *p1, *p2;  
p1 = &x;  
p2 = p1;
```

- Sau khi đoạn lệnh trên được thực hiện, cả hai p1 và p2 cùng trỏ đến biến x.

Lệnh gán con trỏ

❖ Ví dụ



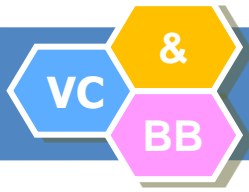
```
int x=10;  
int *p1, *p2;  
p1 = &x;  
p2 = p1;
```

```
int main()  
{  
    int x=10;  
    int *p1,*p2;  
    p1=&x;  
    p2=p1;  
    printf("x= %d\n",x);  
    printf("*p1= %d\n",x);  
    printf("*p2= %d\n",x);  
    return 0;  
}
```

- ❖ `Int *p;`
- ❖ `P=NULL;` // gán con trỏ cho NULL
- ❖ Muốn gán cho con trỏ khác kiểu phải dùng ép kiểu

– Ví dụ:

```
■ int x;  
■ char *pc;  
■ pc = (char*) (&x);
```



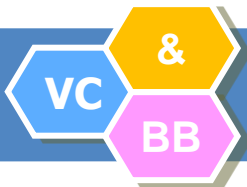
Phép tăng giảm địa chỉ

❖ `float x[30]; *px;`

❖ `px = &x[10];`

- `px++` trở tới phần tử `x[11]`
- `px--` trở tới phần tử `x[9]`
- `px + i` trở tới phần tử `x[10 + i]`
- `px - i` trở tới phần tử `x[10 - i]`

- ❖ **Lưu ý:** cả hai toán tử tăng (++) và giảm (--) đều có quyền ưu tiên lớn hơn toán tử *
- ❖ **Ví dụ:** *p++;
 - Lệnh *p++ tương đương với *(p++) : thực hiện là tăng p (địa chỉ ô nhớ mà nó trỏ tới chứ không phải là giá trị trỏ tới).



Cấp phát bộ nhớ động

1

Khái niệm cấp phát động

2

Cấp phát động với C

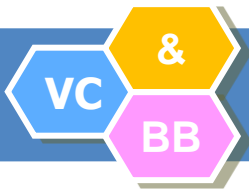
3

Con trỏ void

4

Con trỏ NULL





Khái niệm cấp phát động

- ❖ Các chương trình trước đây kích cỡ vùng nhớ khai báo là **cố định và không thể thay đổi trong thời gian chương trình chạy**.
- ❖ Tuy nhiên chúng ta cần một lượng bộ nhớ mà **kích cỡ của nó chỉ có thể được xác định khi chương trình chạy**, ví dụ như trong trường hợp chúng ta nhận thông tin từ người dùng để xác định lượng bộ nhớ cần thiết.???
- ❖ Giải pháp ở đây chính là **bộ nhớ động**
- ❖ C/C++ hỗ trợ hai hệ thống cấp phát động: một hệ thống được định nghĩa bởi C và một được định nghĩa bởi C++.

Cấp phát động bằng C

Malloc

- ❖ **void* malloc (số byte cần cấp phát cho toàn bộ chương trình);**
- ❖ **Cấp phát một vùng nhớ có kích thước là số byte**
- ❖ **Dùng thư viện: stdlib.h**

calloc

- ❖ **void* calloc (số lượng ô cần cấp phát, độ lớn của 1 ô)**
- ❖ **Cấp phát vùng nhớ đủ chứa số lượng ô**

Cấp phát động bằng C

❖ Ví dụ 1:

```
char *p;
```

```
p = (char *) malloc(1000); //cấp phát 1000 bytes
```

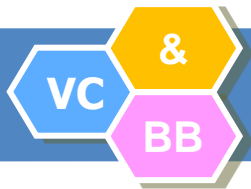
- ❖ Vì hàm malloc() trả về con trỏ kiểu void, nên phải ép kiểu nó thành con trỏ char cho phù hợp với biến con trỏ p.

❖ Ví dụ 2:

```
int *p;
```

```
p = (int *) malloc(50*sizeof(int));
```

- ❖ Toán tử **sizeof** để xác định kích thước kiểu dữ liệu int.

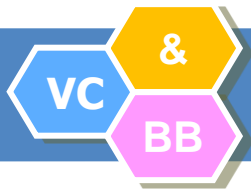


Cấp phát động bằng C

- ❖ Kiểm tra giá trị trả về của hàm malloc() để biết là bộ nhớ có được cấp phát thành công hay không.

Ví dụ:

```
p = (int *)malloc(100);  
if(p == NULL)  
{  
    printf("Khong du bo nho");  
    exit(1);  
}
```



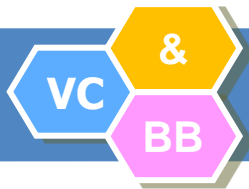
Giải phóng bộ nhớ

❖ **Hàm free():** Trả về vùng nhớ được cấp phát bởi hàm malloc().

❖ Cú pháp:

void free(void *p);

p là con trỏ đến vùng nhớ đã được cấp phát trước đó bởi hàm **malloc()**.



Con trỏ void

❖ **Con trỏ void** là một loại con trỏ đặc biệt mà có thể trỏ đến bất kỳ kiểu dữ liệu nào.

❖ Cú pháp:

```
void *pointerVariable;
```

❖ Ví dụ:

```
void *p;
```

```
p = &a; // p trỏ đến biến nguyên a
```

```
p = &f; //p trỏ đến biến thực f
```

Con trỏ void

❖ Tuy nhiên, ta cũng có thể ép kiểu con trỏ về đúng kiểu tương ứng khi dùng trong các biểu thức.

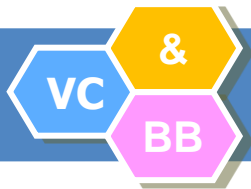
❖ Ví dụ:

- Nếu p đang trỏ đến biến nguyên a, để tăng giá trị của biến a lên 10 ta phải dùng lệnh sau:

$(\text{int}^*)^*p + 10;$

- Nếu p đang trỏ đến biến thực f, để tăng giá trị của biến f lên 10 ta phải dùng lệnh sau:

$(\text{float}^*)^*p + 10;$



Con trỏ NULL

Con trỏ NULL là một hằng số có giá trị bằng 0 được xác định trong một số thư viện chuẩn. Hãy xem xét chương trình sau -

Ví dụ

```
#include <stdio.h>

int main () {
    int *ptr = NULL;
    printf("The value of ptr is : %x\n", ptr );
    return 0;
}
```

Non-zero value



null



0



undefined



An abstract graphic featuring a central point from which several colored lines (green, orange, blue, grey) radiate outwards. The background is a light blue grid with faint binary code (0s and 1s) and a line graph with two lines (one orange, one blue) showing an upward trend. The text "CON TRỎ VÀ MẢNG" is centered in the lower half of the image.

CON TRỎ VÀ MẢNG

Con trỏ và mảng

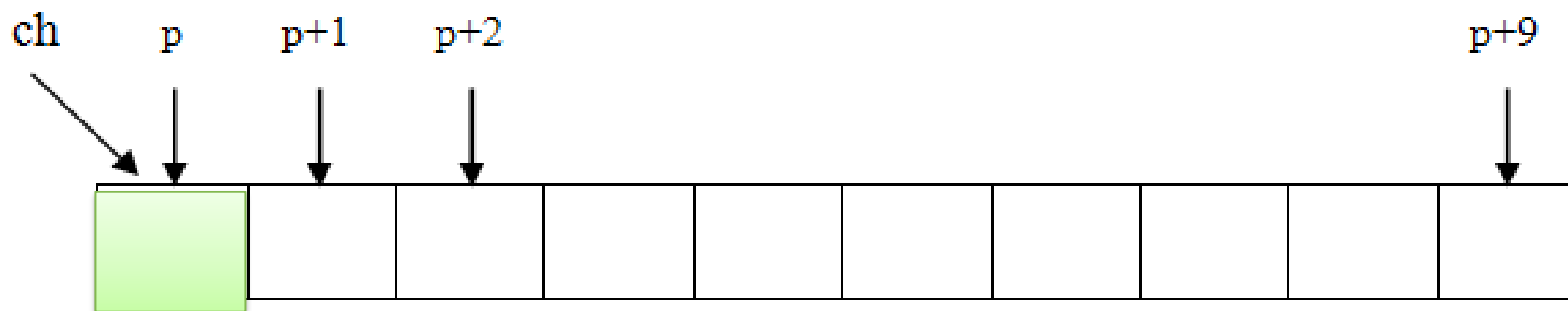
- Tên của một mảng tương đương với địa chỉ phần tử đầu tiên của nó
- Ví dụ: float a[10] thì a=&a[0] và a+i=&(a[i])

- Ví dụ:

```
float a[10], *p, *q;  
p=a; // p trỏ tới phần tử a[0]  
q=p+5; // q trỏ vào phần tử thứ 5  
Khi đó: p+i = &a[i], q=&a[5]; a[i]=*(a+i)=*(p+i)=p[i]
```


❖ Ví dụ:

```
char ch[10], *p; p = ch;
```



ch[0] ch[1] ch[2] ch[3] ch[4] ch[5] ch[6] ch[7] ch[8] ch[9]

❖ p được gán địa chỉ của phần tử đầu tiên của mảng ch.

```
p = ch;
```

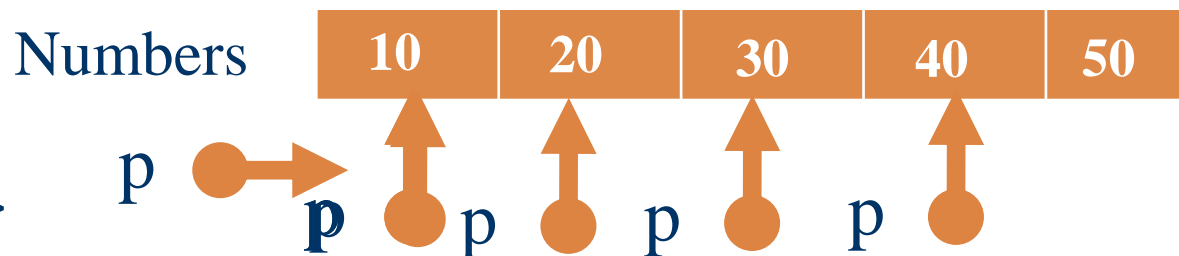
❖ Để tham chiếu phần tử thứ 3 trong mảng ch, ta dùng một trong 2 cách sau:

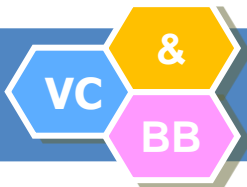
```
ch[2] hoặc *(p+2)
```

Con tr  và mảng

❖ Ví dụ:

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int numbers[5], * p;
    p = numbers; *p = 10;
    p++; *p = 20;
    p = &numbers[2]; *p = 30;
    p = numbers + 3; *p = 40;
    p = numbers; *(p+4) = 50;
    for (int n=0; n<5; n++)
        printf("%d", numbers[n] );
}
```

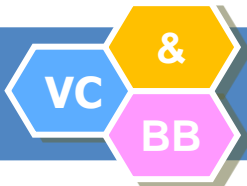




Ví dụ

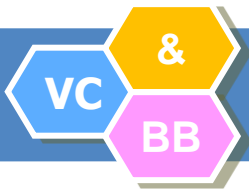
```
main()
{
    int a[3]={10,20,30};
    int *ptr;
    ptr=a;

    printf("Noi dung cua a[0]=%d\n", *ptr);
    printf("Noi dung cua a[1]=%d\n", *(ptr+1);
    printf("Noi dung cua a[1]=%d\n", *(ptr+2);
}
```



/*Minh họa việc nhập dữ liệu vào mảng bằng con trỏ*/

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i, a[5], *p;
    p=a;
    for(i=0;i<5;i++)
    {
        printf("\n a[%d]= ",i);
        scanf("%d",p+i);//Nhap vao dia chi &a[i]
    }
    for(i=0;i<5;i++)
        printf("%d\t", *(a+i));
    return 0;
}
```



Hàm và mảng đa chiều

- ❖ Mảng 2 chiều là mảng 1 chiều của mảng
- ❖ A trỏ tới phần tử $a[0][0]$
- ❖ $A+1$ trỏ tới phần tử $a[0][1]$, ...
- ❖ Khai báo: `int *pa, int n;`
Cấp phát mảng hai chiều:
`pa = (int*)malloc(số dòng*số cột*sizeof(int));`
 - truy cập phần tử $a[i][j]$, dùng công thức
 $*(pa+i*số\ cột+j)$

❖ Ví dụ;

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

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

- ❖ thực tế một mảng 2 chiều được lưu theo thứ tự chính hàng, tức là các hàng được đặt cạnh nhau.

arr[0][0]				arr[1][0]				arr[2][0]			
5000	5004	5008	5012	5016	5020	5024	5028	5032	5036	5040	5044
1	2	3	4	5	6	7	8	9	10	11	12
Row 1				Row 2				Row 3			

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int *p;
    p = (int*)malloc(2*3*sizeof(int));
    int i, j;
    //scanf("%d",&a[i][j]) &a[i][j] <=>
    ( p + socot * i + j )
    for(i = 0; i < 2; i++){
        for(j = 0; j < 3; j++){
            scanf("%d",(p + 3 * i +
j));        }
    }
```

```
printf("Mang hai chieu cua ban
sau khi nhap la : \n");
    for(i = 0; i < 2; i++){
        for(j = 0; j < 3;
j++){
            printf("%d ",*(p + 3 *
i + j));
        }
        printf("\n");
    }
    return 0;
}
```


Con trỏ và chuỗi ký tự

■ Nhắc lại:

- Khi ta khai báo một chuỗi, thì máy sẽ cung cấp một vùng nhớ cho một mảng kiểu char đủ lớn để lưu các ký tự của chuỗi và ký tự '\0'
- Khi đó: chuỗi này là một hằng địa chỉ -> chứa địa chỉ đầu của mảng lưu giữ nó.

■ Ví dụ:

- `char *p;`
- `p="Turbo C"`
- `*p='T'; *(p+1)='u'`
- `printf("%s",p);` // in chuỗi Turbo C ra màn hình

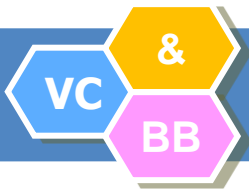
```
/* Minh hoa ve con tro chuoai */  
char s[] = "Hello";  
main()  
{  
    char *ptr;  
    ptr=s;  
    printf("Chuoai ky tu la %s. \n",s);  
    printf("Cac ky tu se la: \n");  
    printf("%c\n",*ptr);  
    printf("%c\n",*(ptr+1));  
    printf("%c\n",*(ptr+2));  
    printf("%c\n",*(ptr+3));  
    printf("%c\n",*(ptr+4));  
    printf("%c\n",*(ptr+5));  
}
```

❖ Con trỏ hàm

- Con trỏ mà trỏ tới các hàm gọi là con trỏ hàm
- <Kiểu trả về> (*tên con trỏ)(<danh sách tham số>)

Ví dụ 1 : Nhập dãy số (không dùng mảng). Sắp xếp và in ra màn hình. Trong ví dụ này chương trình xin cấp phát bộ nhớ đủ chứa n số nguyên và được trả bởi con trỏ head. Khi đó địa chỉ của số nguyên đầu tiên và cuối cùng sẽ là head và head+n-1. p và q là 2 con trỏ chạy trên dãy số này, so sánh và đổi nội dung của các số này với nhau để sắp thành dãy tăng dần và cuối cùng in kết quả.

```
main() { int *head, *p, *q, n, tam; // head trỏ đến (đánh dấu) đầu dãy
cout << "Cho biết số số hạng của dãy: "; cin >> n ;
head = new int[n] ; // cấp phát bộ nhớ chứa n số nguyên for (p=head; p<head+n; p++) // nhập dãy
{ cout << "Số thu " << p-head+1 << ": " ; cin >> *p ; } for (p=head; p<head+n-1; p++) // sắp xếp
for (q=p+1; q<head+n; q++)
if (*q < *p) { tam = *p; *p = *q; *q = tam; } // đổi chỗ for (p=head; p<head+n; p++) cout << *p ; // in kết q
```



6. Con trỏ trỏ vào con trỏ

Cú pháp: type **tên con trỏ

Ví dụ: char a;

char *b;

char **c;

a = 'z'; b = &a; c = &b;