



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

TIN HỌC ĐẠI CƯƠNG

Phần 2. Lập trình

Bài 5. Mảng và chuỗi ký tự

Nội dung

5.1. Mảng

5.2. Xâu kí tự

5.3. Con trỏ và địa chỉ (optional)

Nội dung

5.1. Mạng

5.1.1. Khái niệm mạng

5.1.2. Khai báo và sử dụng mạng

5.1.3. Các thao tác cơ bản trên mạng

5.1.4. Tìm kiếm trên mạng

5.1.5. Sắp xếp trên mạng

5.1.6. Một số thao tác khác

5.2. Xâu kí tự

5.1.1. Khái niệm mảng

- Tập hợp hữu hạn các phần tử cùng kiểu, lưu trữ kế tiếp nhau trong bộ nhớ.
- Các phần tử trong mảng có cùng tên (là tên mảng) nhưng phân biệt với nhau ở chỉ số cho biết vị trí của nó trong mảng.
- Ví dụ:
 - Bảng điểm của sinh viên
 - Vector
 - Ma trận

5.1.2. Khai báo và sử dụng mảng

- Khai báo mảng (một chiều)

`kiểu_dữ_liệu tên_mảng[kích_thước_mảng];`

- Trong đó

- **kiểu_dữ_liệu**: kiểu dữ liệu của các phần tử trong mảng

- **tên_mảng**: tên của mảng

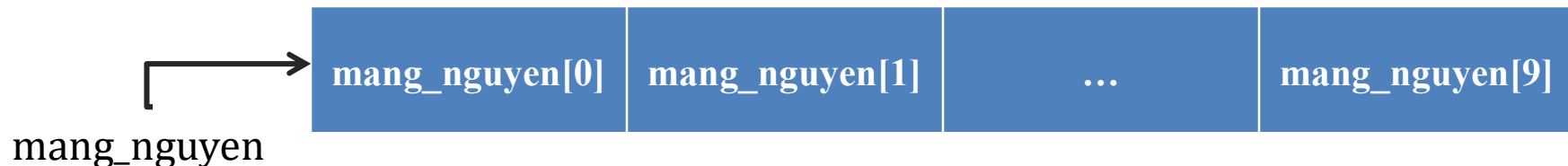
- **kích_thước_mảng**: số phần tử trong mảng

- Ví dụ

```
int mang_nguyen[10]; // khai báo mảng 10  
phần tử có kiểu dữ liệu int
```

5.1.2. Khai báo và sử dụng mảng

- Cấp phát bộ nhớ
 - Các phần tử trong mảng được cấp phát các ô nhớ kế tiếp nhau trong bộ nhớ
 - **Biến mảng** lưu trữ địa chỉ ô nhớ đầu tiên trong vùng nhớ được cấp phát
- Ngôn ngữ C đánh chỉ số các phần tử trong mảng bắt đầu từ 0
 - Phần tử thứ i trong *mang_nguyen* được xác định bởi *mang_nguyen[i-1]*



5.1.2. Khai báo và sử dụng mảng

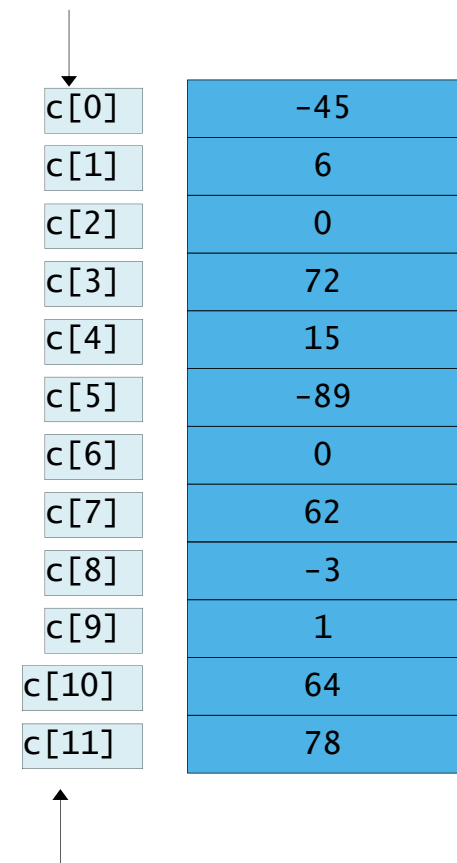
- Ví dụ khai báo mảng:

```
char c[12];
```

Khai báo một mảng:

Tên là **c**, có 12 phần tử,
c[0], c[1], ..., c[11]

Các phần tử thuộc kiểu **char**



c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	15
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	64
c[11]	78

5.1.2. Khai báo và sử dụng mảng

- Mảng một chiều và mảng nhiều chiều
 - Mỗi phần tử của mảng cũng là một mảng
=> mảng nhiều chiều
- Ví dụ
 - **int a[6][5] ;**
mảng a gồm 6 phần tử, mỗi phần tử là mảng gồm 5 số nguyên int
 - **int b[3][4][5] ;**
mảng b gồm 3 phần tử, mỗi phần tử là mảng hai chiều gồm 4 phần tử. Mỗi phần tử mảng hai chiều là mảng gồm 5 số nguyên int. b là mảng 3 chiều

5.1.2. Khai báo và sử dụng mảng

- Khai báo mảng nhiều chiều

`kiểu_dữ_liệu tên_mảng[size1][size2]...[sizek];`

Trong đó

- `sizei` là kích thước chiều thứ *i* của mảng

- Ví dụ

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

```
float b[4][5][6];
```

5.1.2. Khai báo và sử dụng mảng

- Sử dụng mảng
 - Truy cập vào phần tử thông qua **tên mảng** và **chỉ số** của phần tử trong mảng
`tên_mảng[chỉ_số_phần_tử]`
 - Chú ý: **chỉ số bắt đầu từ 0**
- Ví dụ
 - `int a[4];`
 - phần tử đầu tiên (thứ nhất) của mảng: `a[0]`
 - phần tử cuối cùng (thứ tư) của mảng: `a[3]`
 - `a[i]`: là phần tử thứ `i+1` của `a`

5.1.2. Khai báo và sử dụng mảng

- Ví dụ (tiếp)
 - `int b[3][4];`
 - phần tử đầu tiên của mảng: `b[0]` là một mảng một chiều
 - phần tử đầu tiên của mảng `b[0]`: `b[0][0]`
 - `b[i][j]`: là phần tử thứ $j+1$ của `b[i]`, `b[i]` là phần tử thứ $i+1$ của `b`

5.1.3. Các thao tác cơ bản trên mảng

a. Nhập dữ liệu cho mảng

- Khởi tạo giá trị cho mảng ngay khi khai báo

- Ví dụ:

- `int a[4] = {1, 4, 6, 2};`
- `float b[] = {40.5, 20.1, 100};`
- `char c[5] = {'h', 'e', 'l', 'l', 'o'};`
- `int b[2][3]={ {1, 2, 3}, {4, 5, 6} };`

- Số lượng giá trị khởi tạo không được lớn hơn số lượng phần tử trong mảng
- Nếu số lượng này nhỏ hơn, các phần tử còn lại được khởi tạo giá trị 0
- Nếu để trống kích thước mảng bằng số phần tử khởi tạo.

5.1.3. Các thao tác cơ bản trên mảng

a. Nhập dữ liệu cho mảng

- Nhập dữ liệu từ bàn phím bằng hàm scanf
 - `int a[10];`
 - Nhập dữ liệu cho `a[1]`: `scanf("%d", &a[1]);`
 - Nhập dữ liệu cho toàn bộ phần tử của mảng `a`
=> Sử dụng vòng lặp for
- Lưu ý
 - Tên mảng là một hằng (hằng con trỏ) do đó không thể thực hiện phép toán với tên mảng như phép gán sau khi đã khai báo

5.1.3. Các thao tác cơ bản trên mảng

```
#include <stdio.h>

#define MONTHS 12

int main()
{
    int rainfall[MONTHS], i;
    for ( i=0; i < MONTHS; i++ )
    {
        printf("Nhap vao phan tu thu %d:", i+1);
        scanf("%d", &rainfall[i]);
    }
    return 0;
}
```

5.1.3. Các thao tác cơ bản trên mảng

a. Nhập dữ liệu cho mảng

- Lưu ý

- Nếu số phần tử của mảng được nhập từ bàn phím và chỉ biết trước số phần tử tối đa => khai báo mảng với kích thước tối đa và sử dụng biến lưu số phần tử thực sự của mảng.
- Ví dụ: Khai báo mảng số nguyên a có tối đa 100 phần tử. Nhập từ bàn phím số phần tử trong mảng và giá trị các phần tử đó....

5.1.3. Các thao tác cơ bản trên mảng

```
#include <stdio.h>
```

```
void main() {  
    int a[100];  
    int n, i;  
    do {  
        printf("\n Cho so phan tu cua mang:");  
        scanf("%d", &n);  
    } while (n > 100 || n <= 0);
```


5.1.3. Các thao tác cơ bản trên mảng

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

5.1.3. Các thao tác cơ bản trên mảng

b. Xuất dữ liệu trong mảng

- Dùng hàm **printf()**
- Để hiển thị tất cả các phần tử: dùng vòng **for**

- Ví dụ

- Hiển thị một phần tử bất kì
- Hiển thị tất cả các phần tử, mỗi phần tử trên một dòng
- Hiển thị tất cả các phần tử trên một dòng, cách nhau 2 vị trí
- Hiển thị từng k phần tử trên một dòng

5.1.3. Các thao tác cơ bản trên mảng

```
#include <stdio.h>
#define MONTHS 12
int main() {
    int rainfall[MONTHS], i;
    for (i = 0; i < MONTHS; i++) {
        printf("Nhap phan tu thu %d:", i+1);
        scanf("%d", &rainfall[i]);
    }
    printf("Luong mua hang thang la:\n");
    for (i = 0; i < MONTHS; i++)
        printf("%5d ", rainfall[i]);
    return 0;
}
```

5.1.3. Các thao tác cơ bản trên mảng

c. Tìm giá trị lớn nhất, nhỏ nhất

- Tìm giá trị lớn nhất
 - Giả sử phần tử đó là phần tử đầu tiên
 - Lần lượt so sánh với các phần tử còn lại
 - Nếu lớn hơn hoặc bằng => so sánh tiếp
 - Nếu nhỏ hơn => coi phần tử này là phần tử lớn nhất và tiếp tục so sánh
 - Cách làm?
- Tìm giá trị nhỏ nhất: tương tự

5.1.3. Các thao tác cơ bản trên mảng

```
max = rainfall[0];
```

```
for (i = 1; i < MONTHS; i++)
```

```
    if (max < rainfall[i])
```

```
        max = rainfall[i];
```

```
printf("\n Luong mua nhiều nhất là:  
%d", max);
```

Ví dụ

Lập trình nhập vào mảng N số nguyên, in ra các số vừa nhập, tìm phần tử (chỉ số + giá trị) có giá trị tuyệt đối lớn nhất.

5.1.4. Tìm kiếm trên mảng

- Bài toán
 - Cho mảng dữ liệu a và một giá trị k
 - Tìm các phần tử trong mảng a có giá trị bằng (giống) với k . Nếu có in ra vị trí (chỉ số) các phần tử này. Ngược lại thông báo không tìm thấy
- Cách làm
 - Duyệt toàn bộ các phần tử trong mảng
 - Nếu $a[i]$ bằng (giống) k thì lưu lại chỉ số i
 - Sử dụng một biến để xác định tìm thấy hay không tìm thấy

5.1.4. Tìm kiếm trên mảng

- Phân tích
 - Duyệt toàn bộ các phần tử
 - Vòng lặp for (while, do while)
 - Lưu lại i nếu a[i] bằng (giống) k
 - Sử dụng mảng lưu chỉ số
 - Biến xác định tìm thấy hay không tìm thấy
 - Biến nhận giá trị 0 hoặc 1
 - Biến nhận giá trị 0 hoặc ≥ 1 (tìm thấy thì tăng giá trị)

5.1.4. Tìm kiếm trên mảng

```
#include <stdio.h>
```

```
void main() {  
    int a[100], chi_so[100];  
    int n;        // n là số phần tử của mảng  
    int i, k;  
    int dem; //đếm số phần tử đã tìm thấy  
    printf("Nhap vao so phan tu cua mang:");  
    scanf("%d", &n);  
    printf("Nhap vao giá trị tìm kiếm");  
    scanf("%d", &k);  
}
```

5.1.4. Tìm kiếm trên mảng

```
//Nhập các phần tử cho mảng a .....
```

```
.....
```

```
//Phần xử lý tìm kiếm
```

```
dem = 0; //vì chưa tìm thấy p.tử nào
```

```
// Duyệt qua tất cả các phần tử
```

```
for (i = 0; i < n; i++)
```

```
    if (a[i] == k) { //tìm ra a[i] như mong đợi
```

```
        chi_so[dem] = i; //lưu chỉ số
```

```
        dem++; //ghi nhận đã tìm thêm
```

```
    }
```

5.1.4. Tìm kiếm trên mảng

```
if (dem > 0) {  
    printf("Trong mang co %d phan tu co gia tri bang %d", dem, k);  
    printf("\nChi so cua cac phan tu la:");  
    for (i = 0; i < dem; i++)  
        printf("%3d", chi_so[i]);  
} else  
    printf("\n Trong mang khong co phan tu nao co gia tri bang %d", k);  
}
```

Ví dụ

- Lập trình nhập vào N số, liệt kê các phần tử khác nhau có trong mảng.
- Ví dụ: 1 6 2 3 2 4 2 6 5 => Có 6 phần tử khác nhau là 1 6 2 3 4 5

5.1.5. Sắp xếp mảng

- Bài toán
 - Cho mảng a gồm n phần tử. Sắp xếp các phần tử của mảng a theo thứ tự tăng dần/giảm dần

23	78	45	8	32	100
----	----	----	---	----	-----



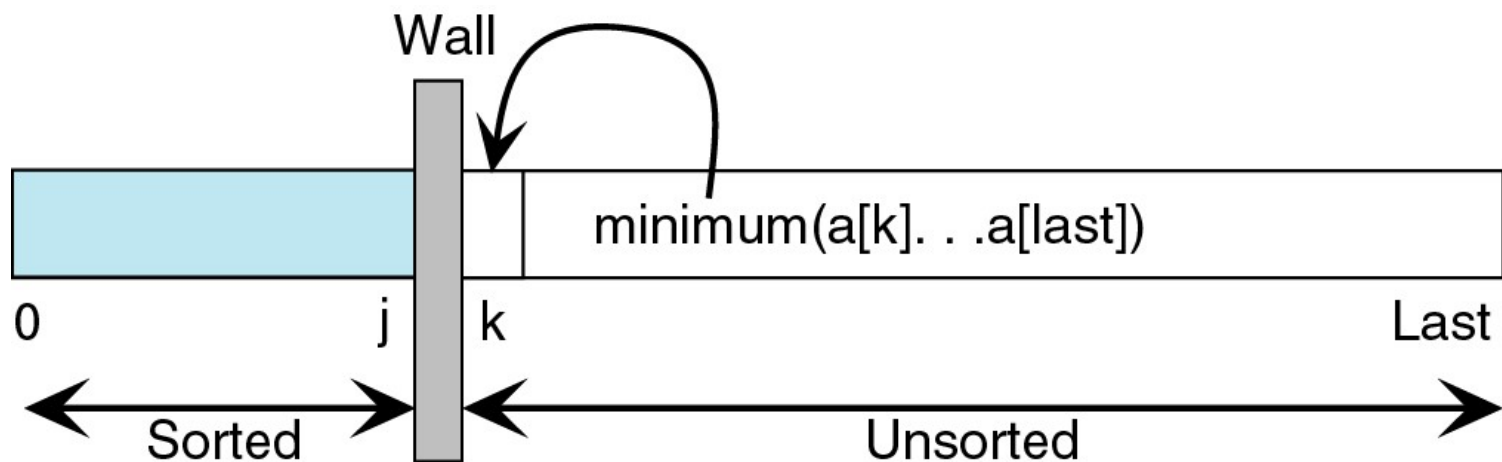
8	23	32	45	78	100
---	----	----	----	----	-----

5.1.5. Sắp xếp mảng

- Giải thuật sắp xếp
 - Sắp xếp thêm dần (insertion sort)
 - **Sắp xếp lựa chọn (selection sort)**
 - **Sắp xếp nổi bọt (bubble sort)**
 - Sắp xếp vun đống (heap sort)
 - Sắp xếp nhanh (quick sort)
 - Sắp xếp trộn (merge sort)
 -

5.1.5. Sắp xếp mảng

- Giải thuật sắp xếp lựa chọn
 - Tìm phần tử nhỏ nhất chưa được sắp xếp trong mảng
 - Đổi chỗ nó với phần tử đầu tiên trong phần chưa được sắp

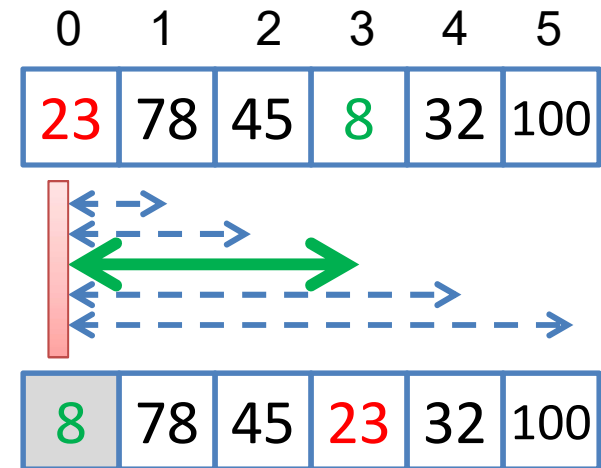


5.1.5. Sắp xếp mảng

- Ý tưởng

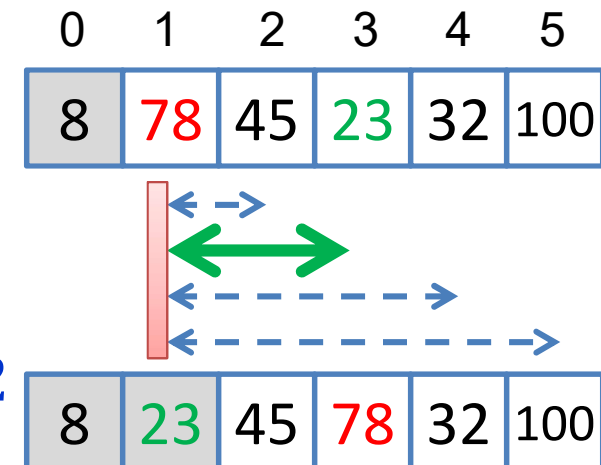
- Lần sắp xếp thứ 1

- So sánh $a[0]$ với các $a[i]$, $i = 1..n-1$
 $a[0] > a[i] \Rightarrow$ đổi chỗ $a[0]$ và $a[i]$
 - Thu được $a[0]$ là phần tử nhỏ nhất



- Lần sắp xếp thứ 2

- So sánh $a[1]$ với các $a[i]$, $i = 2..n-1$
 $a[1] > a[i] \Rightarrow$ đổi chỗ $a[1]$ và $a[i]$
 - Thu được $a[1]$ là phần tử nhỏ thứ 2

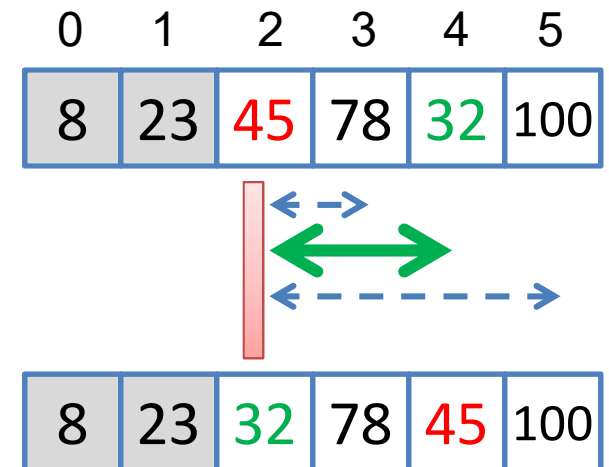


5.1.5. Sắp xếp mảng

- Ý tưởng

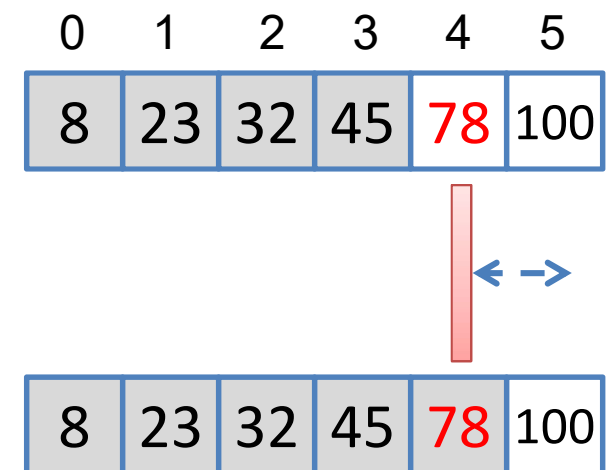
- Lần sắp xếp thứ k

- So sánh $a[k-1]$ với các $a[i]$, $i = k..n-1$
 $a[k-1] > a[i] \Rightarrow$ đổi chỗ $a[k-1]$ và $a[i]$
 - Thu được $a[k-1]$ là phần tử nhỏ thứ k



- Lần sắp xếp thứ n-1

- So sánh $a[n-2]$ và $a[n-1]$
 $a[n-2] > a[n-1] \Rightarrow$ đổi chỗ $a[n-2]$ và $a[n-1]$
 - Thu được $a[n-2]$ là phần tử nhỏ thứ n-1
 \Rightarrow còn lại $a[n-1]$ là phần tử nhỏ thứ n (lớn nhất)



5.1.5. Sắp xếp mảng

- $A = \{ 12, 5, 3, 4 \};$

	Lượt 1	Lượt 2	Lượt 3
12	3	3	3
5	12	4	4
3	5	12	5
4	4	5	12

5.1.5. Sắp xếp mảng

//Khai bao cac bien

```
int a[100];
```

```
int i, j, tmp;
```

//Sap xep

```
for (i = 0; i < n-1; i++)
```

```
    for (j = i+1; j < n ; j++)
```

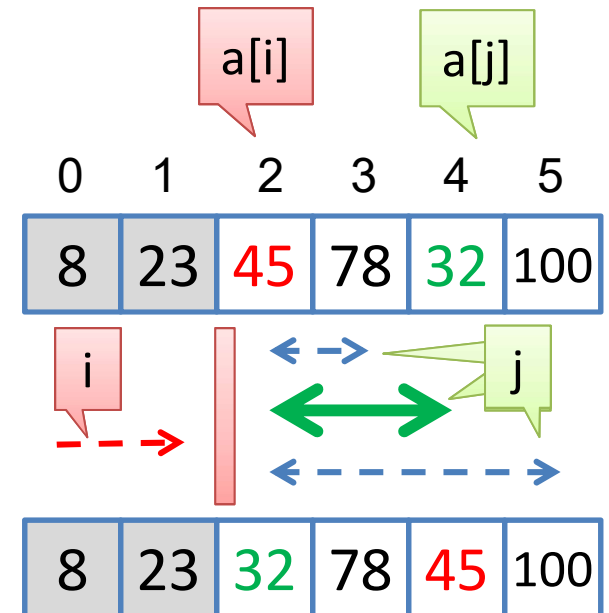
```
        if ( a[i] > a[j]) {
```

```
            tmp= a[i];
```

```
            a[i]= a[j];
```

```
            a[j] = tmp;
```

```
        }
```



5.1.5. Sắp xếp mảng

- Ví dụ
 - Nhập vào từ bàn phím một mảng số nguyên m trong đó số phần tử cũng được nhập từ bàn phím
 - Hiển thị các phần tử vừa được nhập vào
 - Sắp xếp mảng m theo thứ tự tăng dần trong đó có hiển thị các phần tử trong mỗi lượt sắp xếp.

5.1.5. Sắp xếp mảng

```
#include <stdio.h>
```

```
void main() {  
    int m[100];  
    int n;    // n là số phần tử trong mảng  
    int i, j, k, temp;  
    // Nhập giá trị dữ liệu cho mảng m  
    printf("Cho biết số phần tử của mảng: ");  
    scanf("%d", &n);
```

5.1.5. Sắp xếp mảng

```
// nhập giá trị cho các phần tử
for (i = 0; i < n; i++){
    printf("\n Nhập gia tri cua m[%d]=", i);
    scanf("%d", &m[i]);
}
// Hiển thị mảng vừa nhập vào
printf("Mang truoс khi sap xep\n");
for (i = 0; i < n; i++)
    printf("%3d", m[i]);
```

5.1.5. Sắp xếp mảng

```
for (i = 0; i < n-1; i++) {  
    for (j = i + 1; j < n; j++)  
        if (m[j] < m[i]) {  
            temp = m[j];  
            m[j] = m[i];  
            m[i] = temp;  
        }  
    printf("\nMang o luot sap xep thu %d", i+1);  
    for (k = 0; k < n; k++)  
        printf("%3d", m[k]);  
}  
  
}
```

5.1.5. Sắp xếp mảng

- Thuật toán sắp xếp nổi bọt
- Xem lại thuật toán ở phần 2

5.1.6. Một số thao tác khác

- Tính tổng, tích các phần tử của mảng
- Đếm các phần tử của mảng thỏa mãn điều kiện
- Thêm phần tử vào mảng
- Xóa phần tử khỏi mảng

Tính tổng, tích các phần tử

```
// Tinh tong
s = 0;
for (i = 0; i < n; i++)
    s += m[i];
printf("Tong cua cac phan tu: %d\n", s);
```

```
// Tinh tich
p = 1;
for (i = 0; i < n; i++)
    p *= m[i];
printf("Tich cua cac phan tu: %d\n", p);
```

Ví dụ

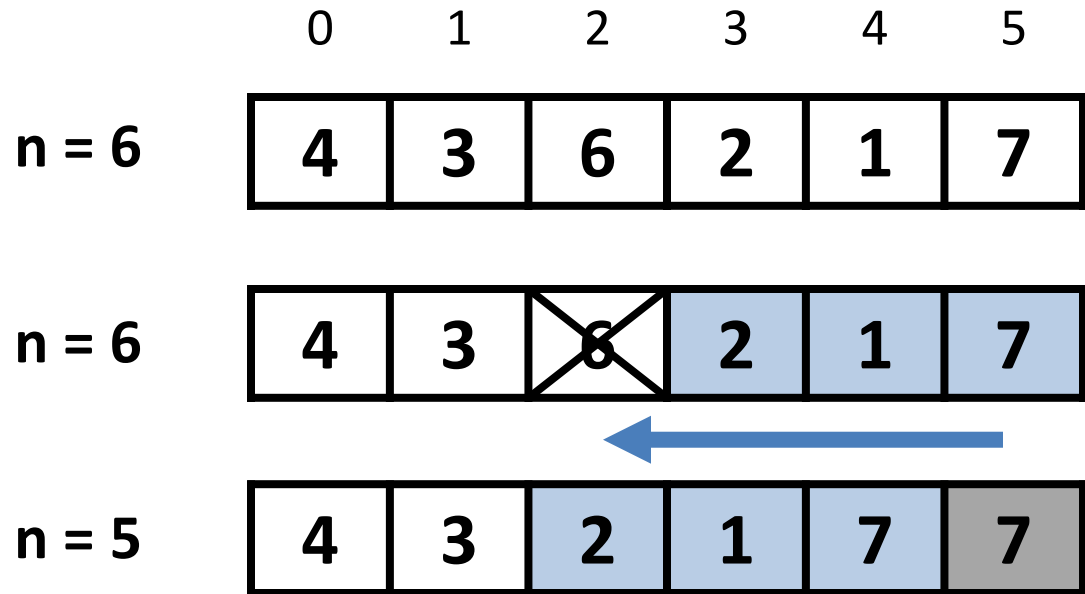
1. Lập trình nhập vào N số, tính tổng các số chẵn và tổng các số lẻ.
2. Lập trình nhập vào N số, tính trung bình cộng của các số dương.

Đếm các phần tử mảng

```
// Dem cac so le va so chan
c1 = 0;
c2 = 0;
for (i = 0; i < n; i++)
    if (m[i] % 2 == 1)
        c1++;
    else
        c2++;

printf("So cac phan tu le: %d\n", c1);
printf("So cac phan tu chan: %d\n", c2);
```

Xóa phần tử khỏi mảng



Giả sử vị trí cần xóa là k

1. Cập nhật lại giá trị các phần tử từ k trở đi $a[i] = a[i+1]$
2. Cập nhật lại số phần tử mảng

Xóa phần tử khỏi mảng

```
// Nhập vị trí mảng cần xóa
printf("Nhập vị trí cần xóa: ");
scanf("%d", &k);

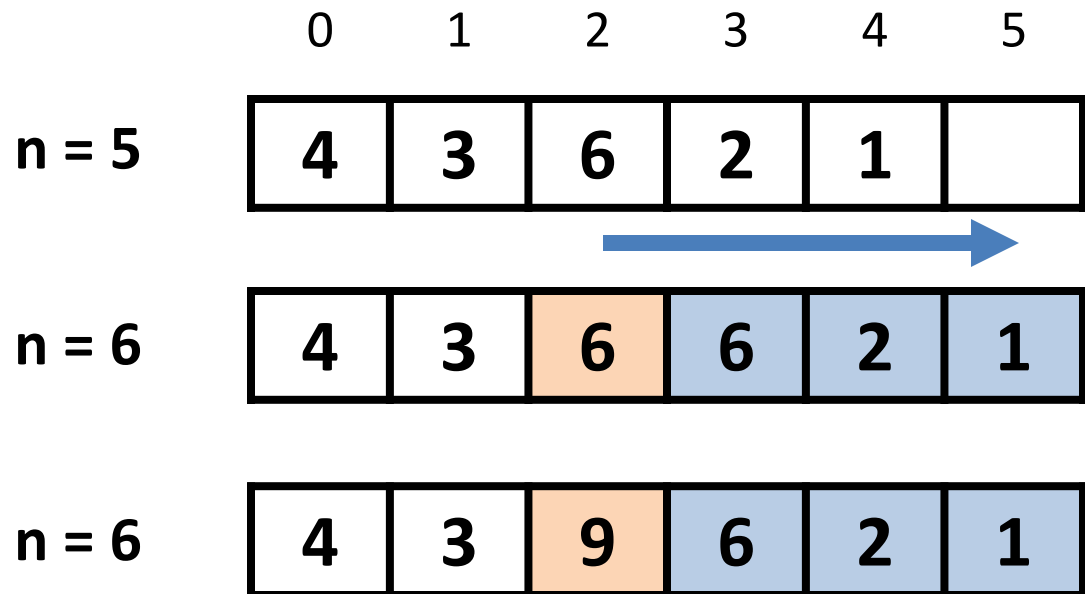
// Xóa phần tử khỏi mảng
if (k < n - 1)
    for (i = k; i < n - 1; i++)
        m[i] = m[i+1];
n--;

// In ra nội dung mảng sau khi xóa phần tử
printf("Nội dung của mảng: \n");
for (i = 0; i < n; i++)
    printf("%5d", m[i]);
```

Ví dụ

- Lập trình nhập vào mảng N số nguyên, sau đó xóa tất cả các số chẵn khỏi mảng.

Thêm phần tử vào mảng



Chú ý: số phần tử của mảng nhỏ hơn số phần tử tối đa

Giả sử vị trí cần thêm là k

1. Cập nhật lại giá trị các phần tử từ k + 1 trở đi $a[i+1] = a[i]$
2. Cập nhật giá trị mới tại vị trí k
3. Cập nhật lại số phần tử mảng

Thêm phần tử vào mảng

```
// Nhập vị trí và giá trị cần chèn vào mảng
printf("Nhập vị trí cần chèn: ");scanf("%d", &k);
printf("Nhập giá trị cần chèn: ");scanf("%d", &p);

// Chèn phần tử vào mảng
if (k < n - 1)
    for (i = n - 1; i >= k; i--)
        m[i+1] = m[i];
m[k] = p;
n++;

// In ra nội dung mảng sau khi xóa phần tử
printf("Nội dung của mảng: \n");
for (i = 0; i < n; i++)
    printf("%5d", m[i]);
```

Bài tập

1. Nhập vào N số thực, kiểm tra xem dãy số có tăng dần hoặc giảm dần không?

2. Nhập N số nguyên dương từ bàn phím. Lập trình chuyển các số lẻ lên đầu dãy, các số chẵn về cuối dãy.

Ví dụ: 1 3 2 7 4 6 5 \Rightarrow 1 3 7 5 2 4 6

3. Nhập N số nguyên dương từ bàn phím. Lập trình sắp xếp các số lẻ tăng dần, các số chẵn giảm dần (giữ nguyên thứ tự đan xen).

Ví dụ: 1 3 2 7 4 6 5 \Rightarrow 1 3 6 5 4 2 7

4. Cho mảng a gồm N phần tử đã được sắp xếp theo thứ tự tăng dần. Lập trình nhập và chèn số X vào mảng sao cho mảng vẫn giữ nguyên thứ tự.

Nội dung

5.1. Mảng

5.2. Xâu kí tự

5.2.1. Khái niệm xâu kí tự

5.2.2. Khai báo và sử dụng xâu

5.2.3. Các hàm xử lý kí tự

5.2.4. Các hàm xử lý xâu

5.2.1. Khái niệm chuỗi ký tự

- Chuỗi ký tự (string) là một dãy các ký tự viết liên tiếp nhau
 - Độ dài chuỗi là số ký tự có trong chuỗi
 - Chuỗi rỗng là chuỗi không có ký tự nào
 - Ví dụ: “Tin hoc” là một chuỗi ký tự gồm 7 ký tự: ‘T’, ‘i’, ‘n’, dấu cách (‘ ’), ‘h’, ‘o’, và ‘c’.
- Lưu trữ: kết thúc chuỗi bằng ký tự ‘\0’ hay NUL (mã ASCII là 0)

‘T’	‘i’	‘n’	‘ ’	‘h’	‘o’	‘c’	‘\0’
-----	-----	-----	-----	-----	-----	-----	------

5.2.1. Khái niệm chuỗi ký tự

- So sánh

- Chuỗi ký tự và mảng ký tự ?

- Giống nhau: Tập hợp các ký tự viết liên tiếp nhau
 - Khác nhau: Chuỗi ký tự có ký tự kết thúc chuỗi, mảng ký tự không có ký tự kết thúc chuỗi
 - Chuỗi là một mảng ký tự một chiều có ký tự kết thúc là NUL ('\0')

- Chuỗi ký tự "A" và ký tự 'A'?

- 'A' là 1 ký tự
 - "A" là 1 chuỗi ký tự, ngoài ký tự 'A' còn có ký tự '\0' => gồm 2 ký tự

- Chuỗi rỗng "" có chứa ký tự kết thúc chuỗi không?

5.2.2. Khai báo và sử dụng chuỗi

a. Khai báo chuỗi

- Cú pháp

```
char tên_xâu[số_kí_tự_tối_đa];
```

- Lưu ý:

- Để lưu trữ một chuỗi có n kí tự chúng ta cần một mảng có kích thước $n+1$

- Ví dụ

- Để lưu trữ chuỗi “Tin học” chúng ta phải khai báo chuỗi có số phần tử tối đa ít nhất là 8

```
char str[8];
```

5.2.2. Khai báo và sử dụng chuỗi

b. Truy cập vào một phần tử của chuỗi

- Cú pháp:

tên_xâu[chỉ_số_của_kí_tự]

- Ví dụ

```
char quequan[10];
```

```
quequan = "Ha noi" ;//xâu này có nội dung là "Ha noi"
```

⇒ quequan[0]	lưu trữ	'H'
quequan[1]		'a'
quequan[5]		'i'
quequan[6]		'\0'

5.2.3. Các hàm xử lý kí tự

- Tập tiêu đề sử dụng: **ctype.h**
- **int toupper(int ch)**: chuyển kí tự thường thành kí tự hoa

toupper('a') => 'A'

- **int tolower(int ch)**: chuyển kí tự hoa thành kí tự thường

tolower('B') => 'b'

5.2.3. Các hàm xử lý kí tự

- `int isalpha(int ch):` kiểm tra xem kí tự có phải chữ cái hay không ('a'...'z','A','..','Z')
- `int isdigit(int ch):` kiểm tra chữ số ('0','1','..'','9')
- `int islower(int ch):` kiểm tra chữ thường
- `int isupper(int ch):` kiểm tra chữ hoa
- `int iscntrl(int ch):` kiểm tra kí tự điều khiển (0-31)
- `int isspace(int ch):` kiểm tra kí tự dấu cách (mã 32), xuống dòng ('\n' 10), đầu dòng ('\r' 13), tab ngang ('\t' 9), tab dọc ('\v' 11)
- trả về khác 0 nếu đúng, ngược lại trả về 0

Bảng mã ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

5.2.3. Các hàm xử lý kí tự

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
void main() {
```

```
    char ch;
```

```
    printf("Nhap vao mot ki tu: ");
```

```
    scanf("%c", &ch);
```

5.2.3. Các hàm xử lý kí tự

```
if (isupper(ch)) {  
    printf("Ki tu nay la chu hoa\n");  
    printf("Ki tu chu thuong tuong  
        ung %c\n", tolower(ch));  
} else if (islower(ch)) {  
    printf("Ki tu nay la chu thuong\n");  
    printf("Ki tu chu hoa tuong ung %c\n",  
        toupper(ch));  
}  
  
}
```

5.2.3. Các hàm xử lý chuỗi ký tự

Vào ra chuỗi ký tự

- Tập tiêu đề: `stdio.h`
- Nhập chuỗi ký tự
 - `gets(tên_xâu) ;`
 - `scanf("%s", tên_xâu) ;`
- Hiển thị chuỗi ký tự
 - `puts(tên_xâu) ;`
 - `printf("%s", tên_xâu) ;`
- Sự khác nhau giữa `gets` và `scanf`?

5.2.4. Các hàm xử lý chuỗi ký tự

Tập tiêu đề: **string.h**

- **size_t strlen(char* tên_xâu)**: trả về độ dài chuỗi
- **char* strcpy(char* chuỗi_đích, char* chuỗi_nguồn)**: sao chép chuỗi
- **int strcmp(char* chuỗi_thứ_nhất, char* chuỗi_thứ_hai)**: so sánh hai chuỗi
 - giá trị 0 : hai chuỗi giống nhau
 - giá trị >0: chuỗi thứ nhất lớn hơn chuỗi thứ hai
 - giá trị <0: chuỗi thứ nhất nhỏ hơn chuỗi thứ hai
- **char* strcat(char* chuỗi_đích, char* chuỗi_nguồn)**: ghép nối chuỗi nguồn vào ngay sau chuỗi đích

5.2.4. Các hàm xử lý chuỗi ký tự

Tập tiêu đề: **stdlib.h**

- **int atoi(char* str):** chuyển một chuỗi ký tự thành một số nguyên tương ứng
- **int atol(char* str):** chuyển thành số long int
- **float atof(char* str):** chuyển thành số thực
- Không thành công cả 3 hàm: trả về 0

5.2.4. Các hàm xử lý chuỗi ký tự

```
#include <stdio.h>
#include <string.h>
```

```
void main() {
    char str1[10] = "abc";
    char str2[10] = "def";
    printf(" str1: %s",str1);
    printf("\n str2: %s",str2);
    printf("\n strcmp(str1,str2)= %d",
           strcmp(str1,str2));
}
```


5.2.4. Các hàm xử lý chuỗi ký tự

```
printf("\n strcpy(str1,str2) = %s",
        strcpy(str1,str2));
printf(" str1: %s",str1);
printf("\n str2: %s",str2);
strcpy(str1,"ab");    // str1 ← "ab"
strcpy(str2,"abc");   // str2 ← "abc"
printf(" str1: %s",str1);
printf("\n str2: %s",str2);
printf("\n strcmp(str1,str2) = %d",
        strcmp(str1,str2));
}
```

Bài tập

Viết chương trình yêu cầu người sử dụng nhập vào 1 xâu ký tự:

1. Đếm số ký tự chữ hoa, chữ thường và chữ số trong xâu.
2. Đếm số từ có trong câu. Giả sử giữa các từ chỉ có 1 dấu cách.

Ví dụ: người sử dụng nhập vào

“Xin chao cac ban”

Kết quả in ra là: 4 từ

3. Chuyển các chữ cái đầu các từ thành chữ hoa, các chữ cái còn lại thành chữ thường.

Ví dụ: người sử dụng nhập vào

“Xin cHao caC bAn”

Kết quả in ra là:

“Xin Chao Cac Ban”

Bài tập

4. In ra xâu theo chiều ngược lại.

Ví dụ: người sử dụng nhập vào

“Xin chao cac ban”

Chương trình sẽ in ra

“nab cac oahc niX”

5. Xóa các dấu cách dư thừa

Ví dụ: người sử dụng nhập vào

“ Xin chao cac ban ”

Chương trình sẽ in ra

“Xin chao cac ban”

5.3. Con trỏ và địa chỉ

- 5.3.1. Tổng quan về con trỏ
- 5.3.2. Các phép toán làm việc với con trỏ
- 5.3.3. Sử dụng con trỏ làm việc với mảng

5.3.1. Tổng quan về con trỏ

- a. Địa chỉ và giá trị của một biến
 - Bộ nhớ như một dãy các byte nhớ.
 - Các byte nhớ được xác định một cách duy nhất qua một *địa chỉ*.
 - Biến được lưu trong bộ nhớ.
 - Khi khai báo một biến
 - Chương trình dịch sẽ cấp phát cho biến đó một số ô nhớ liên tiếp đủ để chứa nội dung của biến. Ví dụ một biến số nguyên (int) được cấp phát 2 byte.
 - Địa chỉ của một biến chính là địa chỉ của byte đầu tiên trong số đó.

5.3.1. Tổng quan về con trỏ

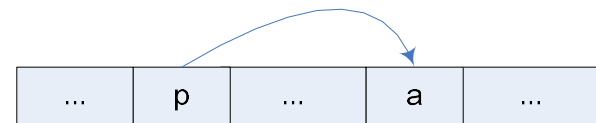
- a. Địa chỉ và giá trị của một biến (tiếp)
 - Một biến luôn có hai đặc tính:
 - Địa chỉ của biến.
 - Giá trị của biến.
 - Ví dụ:
 - `int i, j;`
 - `i = 3;`
 - `j = i + 1;`

Biến	Địa chỉ	Giá trị
i	FFEC	3
	FFED	0
j	FFEE	4
	FFEF	0

5.3.1. Tổng quan về con trỏ

- b. Khái niệm và khai báo con trỏ
 - Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ.

- Khai báo con trỏ:



- Cú pháp khai báo một con trỏ như sau:

kieu_du_lieu * **ten_bien_con_tro**;

- Ví dụ

- `int i = 3;`
- `int * p;`
- `p = &i;`

Biến	Địa chỉ	Giá trị
i	FFEC	3
p	FFEE	FFEC

- Một con trỏ chỉ có thể trỏ tới một đối tượng cùng kiểu.

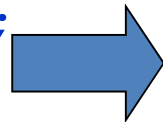
5.3.1. Tổng quan về con trỏ

- Toán tử **&** và *****

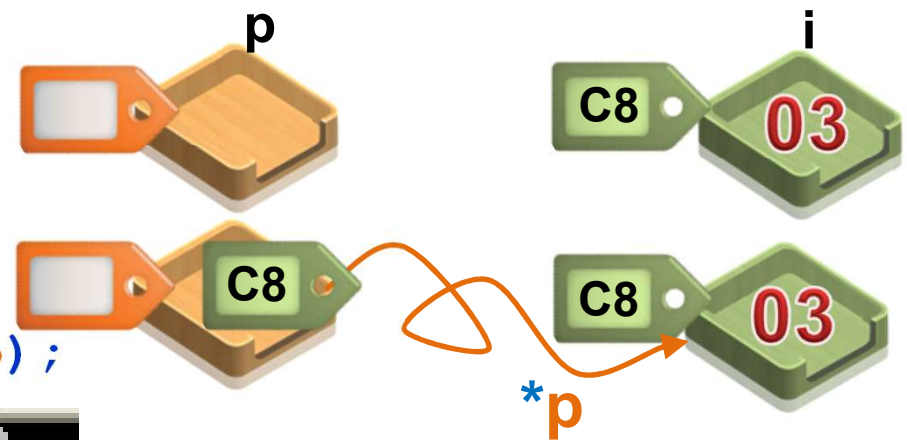
- Toán tử **&**: Trả về địa chỉ của biến.
- Toán tử *****: Trả về giá trị chứa trong vùng nhớ được trỏ bởi giá trị của biến con trỏ.
- Cả hai toán tử ***** và **&** có độ ưu tiên cao hơn tất cả các toán tử số học ngoại trừ toán tử đảo dấu.

- Ví dụ:

```
void main()  
{  
    int i = 3; int *p;  
  
    p = &i;  
    printf("*p = %d \n", *p);  
    getch();  
}
```



```
*p = 3
```

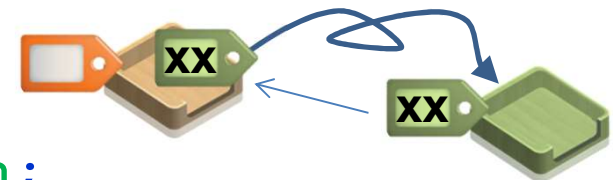


5.3.1. Tổng quan về con trỏ

- c. Sử dụng biến con trỏ:
 - Một biến con trỏ có thể được gán bởi:

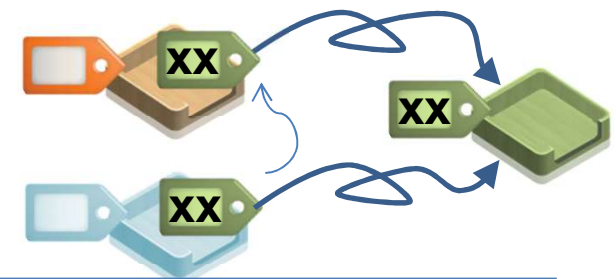
- Địa chỉ của một biến khác:

```
ten_bien_con_tro = &ten_bien;
```



- Giá trị của một con trỏ khác (tốt nhất là cùng kiểu):

```
ten_bien_con_tro2 =  
ten_bien_con_tro1;
```



- Giá trị NUL (số 0):

```
ten_bien_con_tro = 0;
```



- Gán giá trị cho biến(vùng nhớ) mà biến con trỏ trỏ tới:

- `*ten_bien_con_tro = 10;`



5.3.1. Tổng quan về con trỏ

- Ví dụ 1:

?

```
int i = 3, j = 6;
```



```
int *p1, *p2;
```



?

```
p1 = &i;
```



```
p2 = &j;
```



?

```
*p1 = *p2;
```



5.3.1. Tổng quan về con trỏ

- Ví dụ 1:**

```
main()
```

```
{
```

```
    int i = 3, j = 6;
```

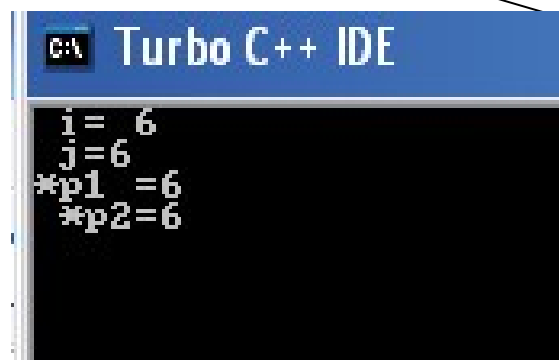
```
    int *p1, *p2;
```

```
    p1 = &i;
```

```
    p2 = &j;
```

```
    *p1 = *p2;
```



```
}
```



```

Turbo C++ IDE
i= 6
j=6
*p1 =6
*p2=6

```



biến	địa chỉ	giá trị
i	FFEC	3
j	FFEE	6
p1	FFDA	FFEC
p2	FFDC	FFEE



biến	địa chỉ	giá trị
i	FFEC	6
j	FFEE	6
p1	FFDA	FFEC
p2	FFDC	FFEE

5.3.1. Tổng quan về con trỏ

- Ví dụ 2:**

```
main()
```

```
{
```

```
    int i = 3, j = 6;
```

```
    int *p1, *p2;
```

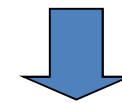
```
    p1 = &i;
```

```
    p2 = &j;
```

```
    p1 = p2;
```

```
}
```

biến	địa chỉ	giá trị
i	FFEC	3
j	FFEE	6
p1	FFDA	FFEC
p2	FFDC	FFEE



biến	địa chỉ	giá trị
i	FFEC	3
j	FFEE	6
p1	FFDA	FFEE
p2	FFDC	FFEE

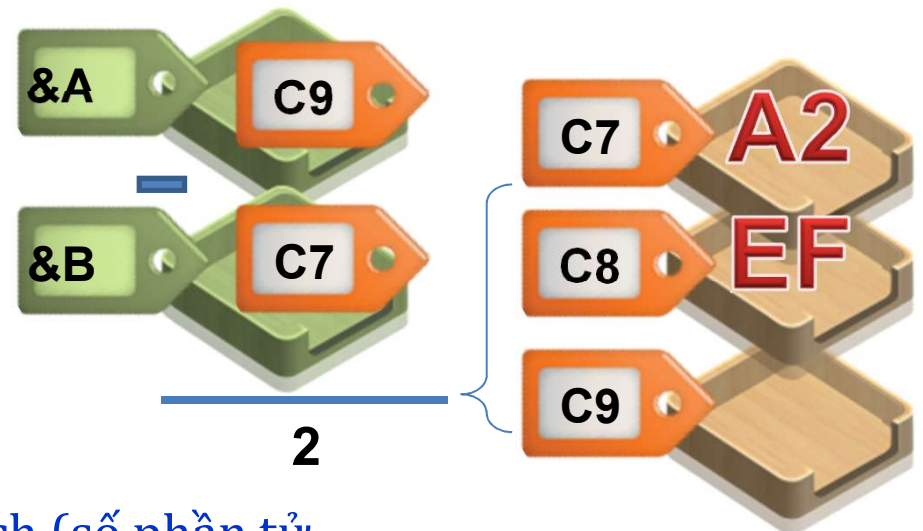
5.3.1. Tổng quan về con trỏ

- d. Con trỏ void
 - `void *ten_bien_con_tro;`
 - Con trỏ đặc biệt, không có kiểu,
 - Có thể nhận giá trị là địa chỉ của một biến thuộc bất kỳ kiểu dữ liệu nào.
 - Ví dụ:
 - `void *p, *q;`
 - `int x = 21;`
 - `float y = 34.34;`
 - `p = &x; q = &y;`

5.3.2. Các phép toán làm việc với con trỏ

- Cộng/trừ con trỏ với một số nguyên (**int**, **long**) → Kết quả là một con trỏ cùng kiểu
 - `ptr--`; //ptr trỏ đến vị trí của phần tử đứng trước nó.
 - Ví dụ: (p2 trỏ đến số nguyên nằm ngay sau x trong bộ nhớ)

```
int x, *p1, *p2;  
p1 = &x;  
p2 = p1 + 1;
```



- Trừ hai con trỏ cho nhau
 - Kết quả là một số nguyên
 - Kết quả này nói lên khoảng cách (số phần tử thuộc kiểu dữ liệu của con trỏ) ở giữa hai con trỏ.
- Các phép toán: nhân, chia, lấy số dư trên con trỏ là không hợp lệ.

5.3.3. Sử dụng con trỏ làm việc với mảng

- Khi khai báo:

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

⇒ a là **hằng con trỏ**, trỏ vào địa chỉ &a[0]

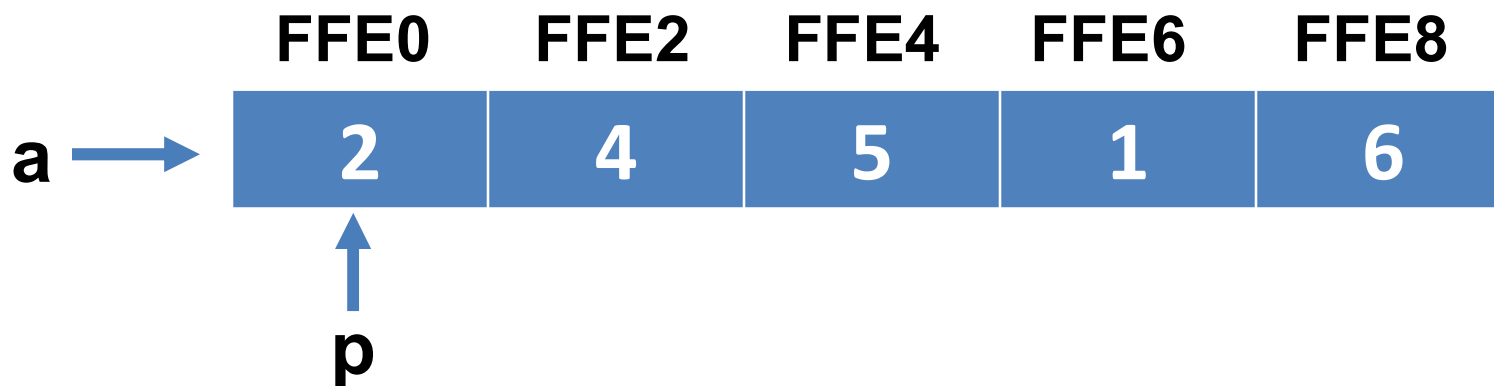
- Khai báo con trỏ

```
int *p;
```

```
p = a;
```

⇒ p là **biến con trỏ**, p và a có cùng giá trị

Khi đó $p = a = \&a[0]$ và $*p = a[0]$



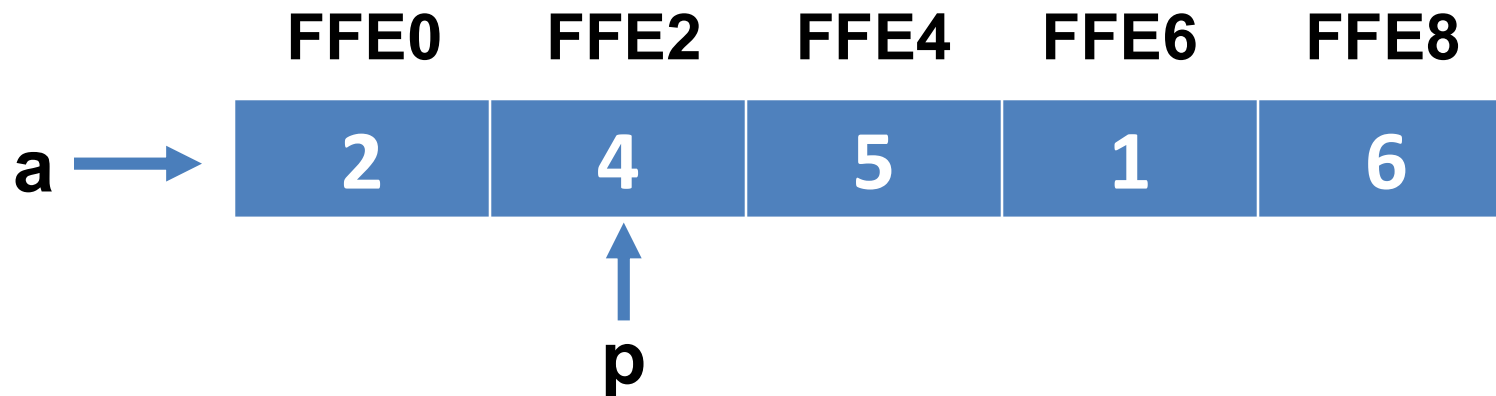
5.3.3. Sử dụng con trỏ làm việc với mảng

- Khi thay đổi giá trị của p

p++ ;

⇒ p trỏ tới phần tử tiếp theo trong mảng

Khi đó $p = \&a[1]$ và $*p = a[1]$



Ví dụ

- Sử dụng con trỏ thực hiện 2 hàm strcmp() và strcpy()

```
p1 = str1;
p2 = str2;

while (*p1 == *p2)
{
    if (*p1 == 0 || *p2 == 0) break;
    p1++;
    p2++;
}

if (*p1 < *p2)
    printf("str1 < str2");
else if (*p1 > *p2)
    printf("str1 > str2");
else
    printf("str1 = str2");
```

Ví dụ

- Sử dụng con trỏ thực hiện 2 hàm strcmp() và strcpy()

```
p1 = str1;  
p2 = str2;  
  
while (*p1 != 0)  
    *p2++ = *p1++;  
*p2 = 0;
```