

**BỘ GIÁO DỤC VÀ ĐÀO TẠO  
ĐẠI HỌC CÔNG NGHỆ TP.HCM**

# **KỸ THUẬT LẬP TRÌNH**

**Biên Soạn:**

ThS. Trương Thị Minh Châu

**[www.hutech.edu.vn](http://www.hutech.edu.vn)**

KỸ THUẬT LẬP TRÌNH



★ 1 . 2 0 2 1 . C M P 1 6 4 ★

---

*Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập:  
tailieuhoctap@hutech.edu.vn*

# MỤC LỤC

MỤC LỤC .....	I
HƯỚNG DẪN .....	IV
BÀI 1: MẢNG .....	1
1.1 KHÁI NIỆM.....	1
1.2 MẢNG MỘT CHIỀU .....	1
1.2.1 Khai báo .....	1
1.2.2 Truy cập vào các phần tử của mảng .....	3
1.2.3 Nhập dữ liệu cho mảng một chiều .....	4
1.2.4 Xuất dữ liệu cho mảng một chiều .....	4
1.2.5 Một vài thuật toán trên mảng một chiều.....	5
1.2.6 Truyền tham số mảng một chiều cho hàm.....	7
1.3 CHUỖI KÝ TỰ (MẢNG MỘT CHIỀU CÁC KÝ TỰ) .....	7
1.3.1 Cách khai báo chuỗi.....	8
1.3.2 Lỗi khi tạo một chuỗi .....	8
1.3.3 Nhập, xuất chuỗi.....	9
1.4 MẢNG HAI CHIỀU.....	9
1.4.1 Khai báo .....	9
1.4.2 Truy cập vào các phần tử của mảng .....	10
1.4.3 Nhập dữ liệu cho mảng hai chiều .....	11
1.4.4 Xuất dữ liệu cho mảng hai chiều .....	13
1.4.5 Truyền tham số mảng 2 chiều cho hàm.....	13
CÂU HỎI ÔN TẬP .....	15
BÀI 2: KIỂU CON TRÒ.....	17
2.1 KHÁI NIỆM VỀ ĐỊA CHỈ Ô NHỚ VÀ CON TRÒ .....	17
2.2 KHAI BÁO VÀ SỬ DỤNG BIẾN CON TRÒ .....	18
2.2.1 Khai báo biến con trò.....	18
2.2.2 Các thao tác trên con trò.....	18
2.3 CÁC PHÉP TOÁN TRÊN CON TRÒ.....	21
2.3.1 Phép gán.....	21
2.3.2 Phép tăng giảm địa chỉ.....	22
2.3.3 Phép truy nhập bộ nhớ.....	22
2.3.4 Phép so sánh .....	23
2.4 SỬ DỤNG CON TRÒ ĐỂ CẤP PHÁT VÀ THU HỒI BỘ NHỚ ĐỘNG .....	23
2.4.1 Các hàm cấp phát vùng nhớ .....	24
2.4.2 Toán tử new và delete (trong C++) .....	26
2.4.3 Thu hồi bộ nhớ động.....	27
2.4.4 Toán tử sizeof: .....	27
2.5 CON TRÒ VÀ MẢNG MỘT CHIỀU.....	28

2.5.1 Truy cập các phần tử mảng theo dạng con trỏ .....	28
2.5.2 Truy cập từng phần tử đang được quản lý bởi con trỏ theo dạng mảng.....	29
2.5.3 Bài toán minh họa .....	30
<b>2.6 CON TRỎ VÀ MẢNG HAI CHIỀU .....</b>	<b>32</b>
2.6.1 Bài toán minh họa .....	32
2.6.2 Cách 1.....	32
2.6.3 Cách 2.....	34
<b>2.7 CON TRỎ VÀ CHUỖI KÝ TỰ .....</b>	<b>36</b>
2.7.1 Cách khai báo chuỗi .....	36
2.7.2 Cách nhập/ xuất chuỗi ký tự .....	36
2.7.3 Một số hàm xử lý chuỗi (trong <string.h> ).....	37
<b>2.8 CON TRỎ VỚI KIỂU DỮ LIỆU CÓ CẤU TRÚC (STRUCT).....</b>	<b>44</b>
2.8.1 Ví dụ 1.....	44
2.8.2 Ví dụ 2.....	44
2.8.3 Truyền tham số kiểu cấu trúc cho hàm .....	46
<b>TÓM TẮT .....</b>	<b>48</b>
<b>CÂU HỎI ÔN TẬP.....</b>	<b>48</b>
<b>BÀI 3: ĐỆ QUY.....</b>	<b>52</b>
3.1 KHÁI NIỆM.....	52
3.2 KỸ THUẬT GIẢI BÀI TOÁN BẰNG ĐỆ QUY .....	53
3.3 MỘT SỐ NHẬN XÉT VỀ HÀM ĐỆ QUY .....	54
3.4 SO SÁNH CẤU TRÚC LẶP VÀ ĐỆ QUY .....	54
3.5 PHÂN LOẠI HÀM ĐỆ QUY.....	55
3.5.1 Đệ quy tuyến tính .....	55
3.5.2 Đệ quy nhị phân .....	55
3.5.3 Đệ quy phi tuyến.....	56
3.5.4 Đệ quy hỗ trợ .....	57
<b>CÂU HỎI ÔN TẬP.....</b>	<b>58</b>
<b>BÀI 4: TẬP TIN (FILE).....</b>	<b>59</b>
4.1 KHÁI NIỆM.....	59
4.2 CÁC THAO TÁC TRÊN TẬP TIN .....	60
4.2.1 Khai báo biến tập tin .....	60
4.2.2 Mở tập tin .....	61
4.2.3 Đóng tập tin .....	62
4.2.4 Kiểm tra đến cuối tập tin hay chưa?.....	62
4.2.5 Di chuyển con trỏ tập tin về đầu tập tin - Hàm rewind().....	62
4.3 TRUY CẬP TẬP TIN VĂN BẢN.....	63
4.3.1 Ghi dữ liệu lên tập tin văn bản .....	63
4.3.2 Đọc dữ liệu từ tập tin văn bản.....	64
4.4 TRUY CẬP TẬP TIN NHỊ PHÂN .....	66
4.4.1 Ghi dữ liệu lên tập tin nhị phân - Hàm fwrite().....	66

4.4.2 Đọc dữ liệu từ tập tin nhị phân - Hàm fread()	66
<b>CÂU HỎI ÔN TẬP</b>	<b>71</b>
<b>BÀI 5: TÌM KIẾM</b>	<b>72</b>
5.1 GIỚI THIỆU VỀ BÀI TOÁN TÌM KIẾM	72
5.2 TÌM KIẾM TUYẾN TÍNH	73
5.3 TÌM KIẾM NHỊ PHÂN	74
<b>CÂU HỎI ÔN TẬP</b>	<b>76</b>
<b>BÀI 6: SẮP XẾP</b>	<b>77</b>
6.1 GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP	77
6.1.1 Giới thiệu	77
6.1.2 Mô tả	77
6.2 ĐỘ PHỨC TẠP CỦA GIẢI THUẬT	78
6.2.1 Khái niệm về độ phức tạp của giải thuật	78
6.2.2 Cách tính độ phức tạp của giải thuật	79
6.3 CÁC GIẢI THUẬT SẮP XẾP	80
6.3.1 Giải thuật Bubble Sort	80
6.3.2 Giải thuật Interchange Sort	82
6.3.3 Giải thuật Selection Sort	83
6.3.4 Giải thuật Insertion Sort	85
6.3.5 Giải thuật Shell Sort	87
6.3.6 Giải thuật Quick Sort	89
6.3.7 Giải thuật Merge Sort	91
6.3.8 Giải thuật Radix Sort	93
<b>CÂU HỎI ÔN TẬP</b>	<b>95</b>
<b>TÀI LIỆU THAM KHẢO</b>	<b>96</b>

# **HƯỚNG DẪN**

## **MÔ TẢ MÔN HỌC**

Học phần này trang bị cho sinh viên kiến thức, kỹ thuật viết chương trình, một số kiểu dữ liệu nâng cao để giải quyết bài toán thực tế bằng ngôn ngữ C/C++. Ngoài ra, môn học này còn củng cố tư duy, kỹ năng lập trình, hình thành phong cách lập trình chuyên nghiệp.

Học xong môn này, sinh viên phải nắm được các vấn đề sau:

- Ôn lại và nâng cao một số kỹ thuật thao tác trên mảng một chiều, chuỗi ký tự.
- Khái niệm mảng hai chiều và các thao tác trên mảng hai chiều
- Khái niệm và cách sử dụng biến con trỏ.
- Hiểu được khái niệm đệ quy và cách viết hàm đệ quy.
- Biết cách thao tác, truy cập dữ liệu trên tập tin.
- Làm quen với các kỹ thuật tìm kiếm, sắp xếp cơ bản và nâng cao.

## **NỘI DUNG MÔN HỌC**

- Bài 1: MẢNG
- Bài 2: KIỂU CON TRỎ
- Bài 3: ĐỆ QUY
- Bài 4: TẬP TIN (FILE)
- Bài 5: TÌM KIẾM
- Bài 6: SẮP XẾP

## **YÊU CẦU MÔN HỌC**

Có tư duy tốt về logic và kiến thức toán học cơ bản.

## CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Kỹ thuật lập trình là môn học nối tiếp của môn Lập trình C, là môn học tiếp theo giúp sinh viên làm quen với phương pháp lập trình và ngôn ngữ lập trình và các kỹ thuật lập trình cơ bản trên máy tính. Môn học giúp sinh viên có khái niệm cơ bản về cách tiếp cận và giải quyết các bài toán trong thực tế, giúp sinh viên có khả năng tiếp cận với cách tư duy của người lập trình, và là tiền đề để tiếp cận với các học phần quan trọng còn lại của ngành Công nghệ Thông tin. Vì vậy, yêu cầu người học phải dự học đầy đủ các buổi lên lớp, làm bài tập đầy đủ ở nhà, nghiên cứu tài liệu trước khi đến lớp và gạch chân những vấn đề không hiểu khi đọc tài liệu để đến lớp trao đổi. Để nâng cao hiệu quả học tập, người học cần chủ động tham khảo thêm nhiều nguồn tài liệu khác liên quan đến nội dung môn học như sách tham khảo, tài liệu online, ... và làm thêm nhiều bài tập mở rộng, nâng cao.

## PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Để học tốt môn này, người học cần ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, người học đọc trước mục tiêu bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người đọc trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, người đọc làm các bài tập.

Điểm đánh giá: gồm 2 phần

1. Điểm quá trình (chuyên cần + trung bình cộng các bài kiểm tra)
2. Điểm cuối kỳ (bài thi trên giấy - tự luận).





# BÀI 1: MẢNG

Học xong bài này, sinh viên sẽ nắm được các vấn đề sau:

- Ôn lại khái niệm mảng 1 chiều, chuỗi ký tự
- Khái niệm về kiểu dữ liệu mảng 2 chiều cũng như ứng dụng của nó.
- Cách khai báo biến kiểu mảng 2 chiều và các phép toán trên các phần tử của mảng 2 chiều.
- Đi sâu vào các giải thuật trên mảng 1 chiều và 2 chiều.

## 1.1 KHÁI NIỆM

Mảng là một tập hợp các phần tử cố định có cùng một kiểu dữ liệu, gọi là kiểu phần tử. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự...; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng (trong trường hợp này ta gọi là mảng của mảng hay mảng nhiều chiều).

Ta có thể chia mảng làm 2 loại: mảng một chiều và mảng nhiều chiều.

Kích thước của mảng là số phần tử của mảng. Kích thước này phải được biết ngay khi khai báo mảng.

## 1.2 MẢNG MỘT CHIỀU

### 1.2.1 Khai báo

❖ **Khai báo tường minh (số phần tử xác định):**

Cú pháp:                    <kiểu cơ sở> <tên mảng> [<số phần tử >];

Trong đó:

- <Tên mảng>: được đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên của biến mảng.

- [<Số phần tử>]: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi nó là kích thước của mảng).
- <Kiểu cơ sở>: là kiểu dữ liệu của mỗi phần tử của mảng.

Ví dụ:

Khai báo mảng một chiều có tên là **songuyen** gồm 10 phần tử và kiểu cơ sở là int:

```
int songuyen [10];
```

Khai báo mảng một chiều có tên là **sothuc** gồm 15 phần tử và kiểu cơ sở là float:

```
float sothuc [15];
```

Khai báo mảng một chiều có tên là **daykytu** gồm 30 phần tử và kiểu cơ sở là char:

```
char daykytu [30];
```

#### ❖ Khai báo không tường minh (số phần tử không xác định)

Cú pháp:                    <kiểu cơ sở> <tên mảng> [ ];

Khi khai báo, không cho biết rõ số phần tử của mảng, kiểu khai báo này thường được áp dụng trong các trường hợp: vừa khai báo vừa gán giá trị, hoặc khai báo mảng là tham số hình thức của hàm.

Cách 1. Vừa khai báo vừa gán giá trị

Cú pháp:

```
<Kiểu> <Tên mảng> [] = {Các giá trị cách nhau bởi dấu phẩy}
```

Nếu vừa khai báo vừa gán giá trị thì mặc nhiên C sẽ hiểu số phần tử của mảng là số giá trị mà chúng ta gán cho mảng trong cặp dấu {}.

Ví dụ: `float x[] = {12.1, 7.23, 5.0, 27.6, 87.9, 9.31};`

Chúng ta có thể sử dụng hàm `sizeof()` để biết số phần tử của mảng như sau:

$$\text{Số phần tử} = \text{sizeof}(\text{tên mảng}) / \text{sizeof}(\text{kiểu})$$

Cách 2. Khai báo mảng là tham số hình thức của hàm, trong trường hợp này ta không cần chỉ định số phần tử của mảng là bao nhiêu.

Ví dụ: `void nhapmang (int a[ ], int n);`

### 1.2.2 Truy cập vào các phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua Tên biến mảng theo sau là chỉ số nằm trong cặp dấu ngoặc vuông [ ].

Lưu ý: phần tử đầu tiên trong mảng có chỉ số là 0.

Chẳng hạn `a[0]` là phần tử đầu tiên của mảng `a` được khai báo ở trên.

Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

Với cách truy xuất theo kiểu này, Tên biến mảng[Chỉ số] có thể coi như là một biến có kiểu dữ liệu là kiểu được chỉ ra trong khai báo biến mảng.

Chỉ số của mảng có thể là một hằng, một biến hay một biểu thức đại số.

Một phần tử của mảng là một biến có kiểu dữ liệu là kiểu cơ sở nên các thao tác trên các biến cũng được áp dụng như trên các phần tử của mảng.

Ví dụ: Khai báo mảng số thực có 5 phần tử:

```
float a [5];
```

Khi đó: Mảng số thực trên có các phần tử là:

`a[0]`, `a[1]`, `a[2]`, `a[3]`, `a[4]` và là những biến kiểu `float`

Và ta có thể thực hiện các phép toán:

```
float t = 10.0;
```

```
int i = 1;
```

```
a[0] = 4.2;
```

```
a[2] = t;
```

```
a[i] = (a[0] + a[2]) / 2;
```

```
printf("\nGia tri: %f ", a[1]);
```

```
scanf("%f", &a[4]);
```

```
t = a[4];
```

```
a[2 * i + 1] = a[2 * i] + a [2 * i + 2];
```

Ta có thể khởi gán giá trị cho mảng:

```
float x[6] = {12.1, 7.23, 5.0, 27.6, 87.9, 9.31};
```

khi đó:  $x[0]=12.1$ ,  $x[1]=7.23$ ,  $x[2]=5.0$ ,  $x[3]=27.6$ ,  $x[4]=87.9$ ,  $x[5]=9.31$

### 1.2.3 Nhập dữ liệu cho mảng một chiều

Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên: `int a [100]` khi đó máy sẽ cấp phát 200 byte để lưu trữ mảng a

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ:

Vị trí	0	1	2	3	4	....	n-3	n-2	n-1
Giá trị a[ ]	7	3	9	4	5	....	8	12	2
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	....	a[n-3]	a[n-2]	a[n-1]

Nhập dữ liệu cho các phần tử:

```
a[ 0 ] = 7    scanf (" %d", & a[0 ]);
```

```
a[ 1 ] = 3    scanf (" %d", & a[1 ]);
```

```
a[ 2 ] = 9    scanf (" %d", & a[2 ]);
```

.....

```
a[ n-1 ] = 2    scanf (" %d", & a[n-1 ]);
```

Ví dụ: Hàm nhập từng phần tử cho mảng a, gồm n phần tử:

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

### 1.2.4 Xuất dữ liệu cho mảng một chiều

Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên:

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

khi đó máy sẽ cấp phát 200 bytes để lưu trữ mảng a.

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ:

Vị trí	0	1	2	3	4	....	n-3	n-2	n-1
Giá trị a[ ]	7	3	9	4	5	....	8	12	2
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	....	a[n-3]	a[n-2]	a[n-1]

Xuất dữ liệu cho từng phần tử:

```
a [0] = 7    printf("%4d", a[0]);
```

```
a [1] = 3    printf("%4d", a[1]);
```

```
a [2] = 9    printf("%4d", a[2]);
```

```
a [3] = 4    printf("%4d", a[3]);
```

.....

```
a [n-1] = 2    printf("%4d", a[n-1]);
```

Ví dụ: Hàm xuất từng phần tử của mảng a, gồm n phần tử:

```
void XuatMang (int a[], int n){
    for (int i =0 ; i<n ; i++){
        printf ("%4d ", a[i]) ;
    }
}
```

### 1.2.5 Một vài thuật toán trên mảng một chiều

Bài toán 1: Tính tổng các phần tử trong mảng một chiều các số nguyên.

Các bước thực hiện:

- ❖ Khai báo mảng a để lưu trữ 100 phần tử là các số nguyên: `int a[100]`

Khi đó máy sẽ cấp phát 200 bytes để lưu trữ mảng a

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	....	n-3	n-2	n-1
Giá trị a[ ]	7	3	9	4	5	....	8	12	2
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	....	a[n-3]	a[n-2]	a[n-1]

❖ Tính tổng: khai báo một biến S để lưu trữ tổng

$$S = a[0] + a[1] + a[2] + \dots + a[n - 1]$$

Giải thuật:

Đi từ đầu mảng đến cuối mảng

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

Cộng dồn các phần tử a[i] vào biến S

$$S = S + a[i] ;$$

Hàm cài đặt

```
long TinhTong (int a[ ], int n) {
    long s = 0;
    for (int i = 0; i < n; i++)
        s = s + a[i];
    return s;
}
```

Bài toán 2: Tìm phần tử âm đầu tiên có trong mảng một chiều các số nguyên.

Hình ảnh mảng a gồm n phần tử được lưu trong bộ nhớ

Vị trí	0	1	2	3	4	....	n-3	n-2	n-1
a[ ]	7	3	-9	4	5	....	8	12	2

Giải thuật:

Đi từ đầu mảng đến cuối mảng

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

Kiểm tra phần tử a[i] < 0 đầu tiên

```
if (a[i] < 0)
```

Nếu gặp thì xuất giá trị `a[i]` và dừng chương trình.

```
return a[i];
```

Hàm cài đặt

```
int AmDau (int a[ ], int n) {  
    for (int i = 0; i < n; i++)  
        if (a[i] < 0)  
            return a[i];  
    return 1;  
}
```

### 1.2.6 Truyền tham số mảng một chiều cho hàm

Ta có thể truyền tham số cho hàm là mảng 1 chiều, khi đó phải khai báo hàm có dạng như sau:

Kiểudl Tên\_hàm(kiểudl tên\_mảng[], số\_phần\_tử, [tham số khác]);

Ví dụ 1: Hàm tính tổng các phần tử trong mảng một chiều các số nguyên

```
long TinhTong (int a[ ], int n);
```

Ví dụ 2: Hàm tính phần tử x trong mảng một chiều các số nguyên

```
int TimX (int a[ ], int n, int x);
```

## 1.3 CHUỖI KÝ TỰ (MẢNG MỘT CHIỀU CÁC KÝ TỰ)

Chuỗi là một dãy ký tự dùng để lưu trữ và xử lý văn bản như từ, cụm từ, câu. Trong ngôn ngữ C không có kiểu chuỗi và chuỗi được thể hiện bằng mảng các ký tự (có kiểu cơ sở `char`), được kết thúc bằng ký tự `'\0'` (còn được gọi là ký tự NULL trong bảng mã ASCII).

Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép `" "`.

Chú ý: Chuỗi được khai báo là một mảng các ký tự nên các thao tác trên mảng có thể áp dụng đối với chuỗi ký tự.

### 1.3.1 Cách khai báo chuỗi

❖ **Khai báo chuỗi:**

**char** < tên biến > [chiều dài tối đa chuỗi];

- Ví dụ:

```
char Hoten [20];
```

Khai báo như trên là khai báo 1 chuỗi chứa tối đa 19 ký tự (còn 1 ký tự cuối của chuỗi chứa NULL)

❖ **Vừa khai báo vừa gán giá trị:**

**char** <Biến> [] = <"Hàng chuỗi">;

Ví dụ:      **char** chuỗi[ ] = " Kỹ thuật lập trình ";

```
char chuoi[50] = "CONG HOA XA HOI CHU NGHIA VIET NAM";
```

```
char name [] = {'K','C','N','T','T','\0'};
```

```
char ten[10] = {'h','o','a','h','o','n','g','\0'};
```

khi đó:

```
ten[0]= 'h'; ten[1]= 'o'; ten[2]= 'a'; ten[3]= 'h'; ten[4]= 'o';
```

```
ten[5]= 'n'; ten[6]= 'g'; ten[7]= '\0';
```

### 1.3.2 Lỗi khi tạo một chuỗi

1. **Chú ý:** Không sử dụng toán tử gán = để chép nội dung của một chuỗi này sang chuỗi khác.

Ví dụ:

```
char a[4] = "hi";
```

```
char b[4];
```

```
b = a;     // Lỗi
```



2. Không dùng toán tử so sánh như `==`, `!=`, `<`, `>` để so sánh nội dung hai chuỗi.

Ví dụ:

```
char a[]="hi";  
char b[] = "there";  
if(a==b) // Không so sánh nội dung hai chuỗi  
{  
    //.....  
}
```

### 1.3.3 Nhập, xuất chuỗi

#### ❖ Nhập chuỗi:

Cách 1: `gets(biến chuỗi);`

- Nhận các ký tự nhập từ phím cho đến khi nhấn phím Enter và đưa vào biến chuỗi.

Ví dụ:

```
char s[30];  
  
fflush(stdin); printf("Nhap chuoi: "); gets(s);
```

Cách 2: `scanf("%s", biến chuỗi);`

Nhận các ký tự nhập từ phím cho đến khi gặp phím space, tab, new line, Enter thì dừng, cho nên chỉ dùng hàm `scanf` để nhập chuỗi không có khoảng trắng.

#### ❖ Xuất chuỗi:

Cách 1: `puts (biến chuỗi);` //Xuất chuỗi xong tự động xuống dòng

Cách 2: `printf ("%s", biến chuỗi);`

## 1.4 MẢNG HAI CHIỀU

### 1.4.1 Khai báo

Cú pháp: `<kiểu cơ sở> <tên mảng> [<số dòng >] [<số cột >];`

Ý nghĩa:

- Tên mảng: Được đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên biến mảng.
- Số dòng: là một hằng số nguyên, cho biết số lượng dòng tối đa trong mảng là bao nhiêu.
- Số cột: là một hằng số nguyên, cho biết số lượng cột tối đa trong mảng là bao nhiêu.
- Số phần tử: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói cách khác, nó là kích thước của mảng). Số phần tử của mảng chính bằng số dòng \* số cột.
- Kiểu cơ sở: là kiểu dữ liệu của mỗi phần tử của mảng.

Ví dụ:

- Khai báo mảng hai chiều có tên là a gồm 8 hàng, 14 cột và kiểu cơ sở là int  
`int a [8][14];`
- Khai báo mảng hai chiều có tên là b gồm 10 hàng, 5 cột và kiểu cơ sở là float  
`float b [10][5];`

### 1.4.2 Truy cập vào các phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua Tên biến mảng theo sau là chỉ số nằm trong cặp dấu ngoặc vuông [ ][ ].

Chẳng hạn `a[0][0]` là phần tử đầu tiên của mảng a được khai báo ở trên.

Chỉ số của phần tử mảng có giá trị là kiểu số nguyên.

Chỉ số của mảng có thể là một hằng, một biến hay một biểu thức đại số.

Một phần tử của mảng là một biến có kiểu dữ liệu là kiểu cơ sở nên các thao tác trên các biến cũng được áp dụng trên các phần tử của mảng.

Ví dụ: Cho mảng `a[4][8]`,

Khi đó, mảng gồm  $4 * 8 = 32$  phần tử, vị trí các phần tử của mảng theo dòng và cột như hình sau:

	0	1	2	3	4	5	6	7
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]

### 1.4.3 Nhập dữ liệu cho mảng hai chiều

Ví dụ minh họa trên mảng số nguyên, đối với mảng kiểu khác thực hiện tương tự.

Khai báo hằng:

```
#define d 30 // d là số dòng
```

```
#define c 30 // c là số cột
```

Khai báo mảng số nguyên 2 chiều:

```
int a[d][c];
```

Hình ảnh mảng a gồm d dòng c cột lưu trữ các số nguyên được biểu diễn như ma trận gồm d hàng, c cột như sau:

	0	1	2	3	4	5	...	c-1
0	1	2	3	4	5	6	...	7
1	10	11	12	13	14	15	....	17
...								
d-1	20	21	22	23	24	25	...	27

- Nhập dữ liệu cho dòng thứ nhất:

```
a[0][0] = 1    scanf ("%d", &a[0][0]);
```

```
a[0][1] = 2    scanf ("%d", &a[0][1]);
```

```
a[0][2] = 3    scanf ("%d", &a[0][2]);
```

```
.....
```

```
a[0][c-1] = 2    scanf ("%d", &a[0][c-1]);
```

Như vậy, ta dùng một lệnh lặp để thực hiện việc nhập dữ liệu cho  $c$  phần tử của dòng thứ nhất:

```
for (int j = 0; j < c; j++)
    scanf ("%d ", & a[0][j]);
```

- Nhập dữ liệu cho dòng thứ hai:

```
a [ 1 ][ 0 ] = 1    scanf ("%d", & a [1][0]);
a [ 1 ][ 1 ] = 2    scanf ("%d", & a [1][1]);
a [ 1 ][ 2 ] = 3    scanf ("%d", & a [1][2]);
.....
a [ 1 ][ c-1 ] = 2   scanf ("%d", & a[1][c-1 ]);
```

Như vậy, ta dùng một lệnh lặp để thực hiện việc nhập dữ liệu cho  $c$  phần tử của dòng thứ hai:

```
for (int j = 0; j < c; j++)
    scanf ("%d", & a[1][j]) ;
.....
```

- Nhập dữ liệu cho dòng thứ  $d$ :

```
a [ d-1 ][ 0 ] = 1    scanf ("%d", & a [d-1 ][0]);
a [ d-1 ][ 1 ] = 2    scanf ("%d", & a [d-1 ][1]);
a [ d-1 ][ 2 ] = 3    scanf ("%d", & a [d-1 ][2]);
.....
a [ d-1 ][ c-1 ] = 2   scanf ("%d", & a[d-1][c-1 ]);
```

Như vậy, ta dùng một lệnh lặp để thực hiện việc nhập dữ liệu cho  $c$  phần tử của dòng  $d$ :

```
for (int j = 0; j < c; j++)
    scanf ("%d", & a[d-1][j]) ;
```

Tóm lại, nhập từng phần tử cho ma trận các số nguyên gồm d dòng c cột:

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

Ta có thể viết hàm Nhập ma trận cho ma trận a gồm d dòng, c cột như sau:

```
void NhapMatran (int a[][10], int d, int c){
    for (int i = 0; i < d; i++)
        for (int j = 0; j < c; j++){
            printf ("Nhap a[%d][%d]: ", i, j) ;
            scanf ("%d", &a[ i ][ j ]);
        }
}
```

#### 1.4.4 Xuất dữ liệu cho mảng hai chiều

Tương tự, ta có hàm Xuất ma trận cho ma trận a gồm d dòng, c cột như sau:

```
void XuatMatran (int a[][10], int d, int c){
    for (int i = 0; i < d; i++) {
        for (int j = 0; j < c; j++)
            printf ("%4d ", a[ i ][ j ]);
        printf ("\n");
    }
}
```

#### 1.4.5 Truyền tham số mảng 2 chiều cho hàm

Ta có thể truyền tham số cho hàm là mảng 2 chiều, khi đó phải khai báo hàm có dạng như sau:

```
#define MAX_SIZE 10

typedef Kiểudl MATRAN[MAX_SIZE][MAX_SIZE];
```

Kiểu `dl_Tên_hàm(MATRAN tên_mảng,[int số_dòng,int số_cột,tham số khác]);`

Ví dụ 1: Hàm tính tổng các phần tử trong ma trận `d` dòng, `c` cột

```
#define MAX_SIZE 10
typedef int MATRAN[MAX_SIZE][MAX_SIZE];
long TongMatran (MATRAN a, int d, int c){
    long s = 0;
    for (int i = 0; i < d; i++)
        for (int j = 0; j < c; j++)
            s = s + a[i][j];
    return s;
}
```

Ví dụ 2: Hàm tính tổng các phần tử trong hàng thứ `k` của ma trận `d` dòng, `c` cột

```
#define MAX_SIZE 10
typedef int MATRAN[MAX_SIZE][MAX_SIZE];
long TongHang (MATRAN a, int c, int k){
    long s = 0;
    for (int j = 0; j < c; j++)
        s = s + a[k][j];
    return s;
}
```

## CÂU HỎI ÔN TẬP

**Câu 1:** Viết chương trình thực hiện:

- a. Nhập mảng số nguyên gồm  $n$  phần tử ( $0 < n \leq 100$ ).
- b. Xuất mảng số nguyên.
- c. Tìm vị trí phần tử dương đầu tiên. Xuất ra vị trí và giá trị phần tử dương đầu tiên tìm được nếu có.
- d. Tìm vị trí phần tử dương cuối cùng. Xuất ra vị trí và giá trị phần tử dương cuối cùng tìm được nếu có.
- e. Tìm giá trị phần tử lớn nhất.
- f. Đếm số phần tử lớn nhất.
- g. Xuất ra vị trí của các phần tử lớn nhất.
- h. Thêm 1 phần tử mới vào đầu mảng.
- i. Thêm 1 phần tử mới vào vị trí  $k$  trong mảng ( $k$  do người dùng nhập,  $0 < k \leq \text{MAX}$ , với  $\text{MAX}$  là kích thước cấp phát mảng).
- j. Xóa phần tử đầu mảng
- k. Xóa phần tử tại vị trí  $k$  ( $k$  do người dùng nhập,  $0 < k < \text{MAX}$ , với  $\text{MAX}$  là kích thước cấp phát mảng).
- l. Kiểm tra mảng có chứa số lẻ không?

**Câu 2:** Viết chương trình:

- a. Nhập vào mảng  $A$  gồm  $n$  phần tử, trong quá trình nhập kiểm tra các phần tử nhập vào không được trùng, nếu trùng thông báo và yêu cầu nhập lại.
- b. Xuất mảng.
- c. Xuất ra màn hình các phần tử là số chính phương nằm tại những vị trí lẻ trong mảng.
- d. Xuất ra vị trí của các phần tử có giá trị lớn nhất.

- e. Tìm phần tử âm lớn nhất / phần tử dương nhỏ nhất
- f. Tính tổng các phần tử nằm ở vị trí chẵn trong mảng.
- g. Viết hàm sắp xếp mảng theo thứ tự tăng dần.

**Câu 3:** Đổi các từ ở đầu câu sang chữ hoa và những từ không phải đầu câu sang chữ thường.

Ví dụ: nGuYen vAN an đổi thành: Nguyễn Văn An

**Câu 4:** Viết chương trình gồm các hàm sau:

- a. Nhập ma trận gồm d dòng và c cột (d, c nhập từ bàn phím)
- b. Xuất ma trận
- c. Tính tổng các phần tử của ma trận
- d. Tính trung bình cộng các phần tử trong ma trận
- e. Tính trung bình cộng các phần tử dương trong ma trận
- f. Xuất các phần tử nằm trên dòng k (k do người dùng nhập) trong ma trận
- g. Tính tổng các phần tử nằm trên cột k (k do người dùng nhập) trong ma trận
- h. Tìm phần tử lớn nhất trong ma trận



## BÀI 2: KIỂU CON TRỎ

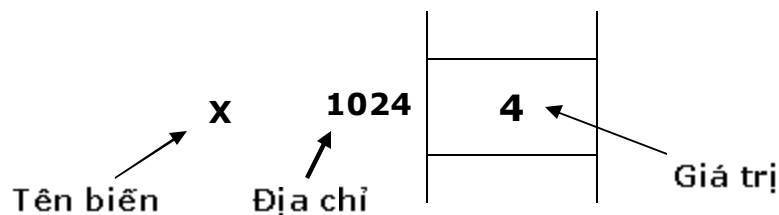
Sau khi học xong bài này, học viên có thể

- Hiểu khái niệm về con trỏ;
- Biết cách khai báo và sử dụng biến kiểu con trỏ;
- Biết xử lý các phép toán trên mảng một chiều theo kiểu con trỏ;
- Biết xử lý các phép toán trên mảng hai chiều theo kiểu con trỏ;
- Đi sâu vào các giải thuật trên mảng 1 chiều, 2 chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...theo kiểu con trỏ;
- Con trỏ với kiểu dữ liệu có cấu trúc.

### 2.1 KHÁI NIỆM VỀ ĐỊA CHỈ Ô NHỚ VÀ CON TRỎ

Các biến chúng ta đã biết và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là biến tĩnh. Khi khai báo biến tĩnh, một lượng ô nhớ cho các biến này sẽ được cấp phát mà không cần biết trong quá trình thực thi chương trình có sử dụng hết lượng ô nhớ này hay không. Mặt khác, các biến tĩnh dạng này sẽ tồn tại trong suốt thời gian thực thi chương trình dù có những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ.

Ví dụ: `int x=4;`



Một số hạn chế có thể gặp phải khi sử dụng các biến tĩnh:

1. Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
2. Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

Để tránh những hạn chế trên, ngôn ngữ C cung cấp cho ta một loại biến đặc biệt gọi là biến động với các đặc điểm sau:

1. Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
2. Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
3. Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên các biến động không có địa chỉ nhất định nên ta không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp cho ta một loại biến đặc biệt nữa để khắc phục tình trạng này, đó là biến con trỏ (pointer) với các đặc điểm:

1. Biến con trỏ không chứa dữ liệu mà chỉ chứa địa chỉ của dữ liệu hay chứa địa chỉ của ô nhớ chứa dữ liệu.
2. Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

## 2.2 KHAI BÁO VÀ SỬ DỤNG BIẾN CON TRỎ

---

### 2.2.1 Khai báo biến con trỏ

Cú pháp: <Kiểu> \* <Tên con trỏ>

Ý nghĩa: Khai báo một biến có tên là Tên con trỏ dùng để chứa địa chỉ của các biến có kiểu Kiểu.

Ví dụ 1: Khai báo 2 biến a, b có kiểu int và 2 biến pa, pb là 2 biến con trỏ

```
int a, b, *pa, *pb;
```

Ví dụ 2: Khai báo biến f kiểu float và biến pf là con trỏ float

```
float f, *pf;
```

### 2.2.2 Các thao tác trên con trỏ

#### 2.2.2.1 Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

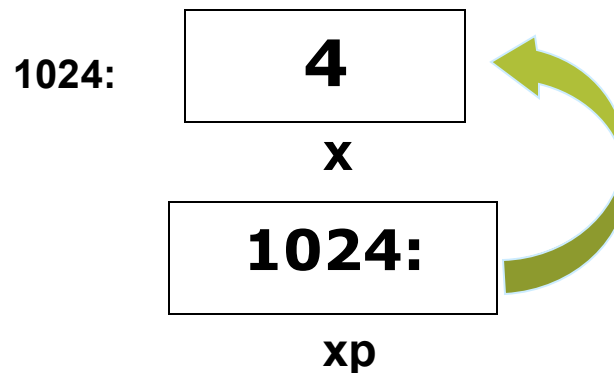
Cú pháp: `<Tên biến con trỏ> = &<Tên biến>`

Giải thích: Ta gán địa chỉ của biến Tên biến cho con trỏ Tên biến con trỏ.

Ví dụ: khai báo biến `int *xp, x = 4;`

Gán địa chỉ của biến `x` cho con trỏ `xp`. `xp = &x`

Lúc này, hình ảnh của các biến trong bộ nhớ được mô tả:



### 2.2.2.2 Lấy giá trị của biến con trỏ chỉ tới

Để truy cập đến nội dung của ô nhớ mà con trỏ chỉ tới, ta sử dụng cú pháp:

`*<Tên biến con trỏ>`

Với cách truy cập này thì `*<Tên biến con trỏ>` có thể coi là một biến có kiểu được mô tả trong phần khai báo biến con trỏ.

Ví dụ sau đây cho phép khai báo, gán địa chỉ cũng như lấy nội dung vùng nhớ của biến con trỏ:

```
int x = 100, *ptr;
```

```
ptr = &x;
```

```
int y = *ptr;
```

Lưu ý: Khi gán địa chỉ của một biến cho một biến con trỏ, mọi sự thay đổi trên nội dung ô nhớ con trỏ chỉ tới sẽ làm giá trị của biến thay đổi theo (thực chất nội dung ô nhớ và biến chỉ là một).

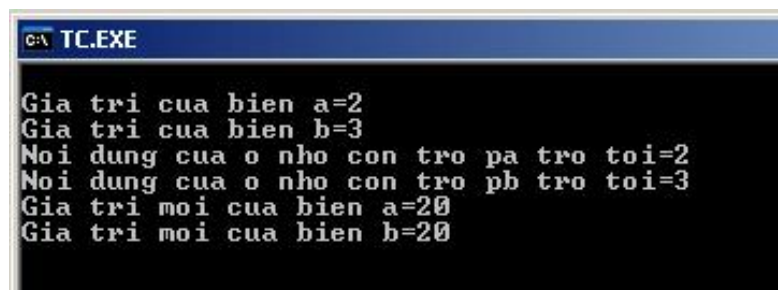
Ví dụ 1: Đoạn chương trình sau thấy rõ sự thay đổi này:

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

```
#include <conio.h>
```

```
int main() {  
    int a, b, *pa, *pb;  
  
    a = 2;  
    b = 3;  
  
    printf("\nGia tri cua bien a = %d\nGia tri cua bien b = %d", a, b);  
  
    pa = &a;  
    pb = &b;  
  
    printf("\nNoi dung cua o nho con tro pa tro toi = %d", *pa);  
    printf("\nNoi dung cua o nho con tro pb tro toi = %d ", *pb);  
  
    *pa = 20; // Thay doi gia tri cua *pa  
    *pb = 20; // Thay doi gia tri cua *pb  
  
    printf("\nGia tri moi cua bien a = %d", a); // a=20  
    printf("\nGia tri moi cua bien b = %d", b); // b=20  
  
    return 0;  
}
```

Kết quả thực hiện chương trình:



```
C:\ TC.EXE  
Gia tri cua bien a=2  
Gia tri cua bien b=3  
Noi dung cua o nho con tro pa tro toi=2  
Noi dung cua o nho con tro pb tro toi=3  
Gia tri moi cua bien a=20  
Gia tri moi cua bien b=20
```

Ví dụ 2:

```
int x, y, *px, *py;  
  
x = 95;    // Truy nhập trực tiếp tới biến x  
  
px = &x;   // Gán địa chỉ của biến x vào biến con trỏ px  
  
py = px;   // Biến con trỏ py trỏ tới biến x
```

```
y = *px;    // Gán y=x
```

```
*py =17;    // Gán số 17 vào nơi py trỏ tới (x=17)
```

Như vậy để truy nhập tới biến ta có thể :

1. Truy nhập trực tiếp tới biến.
2. Truy nhập gián tiếp thông qua biến con trỏ.

## 2.3 CÁC PHÉP TOÁN TRÊN CON TRỎ

Có bốn phép toán liên quan đến con trỏ và địa chỉ:

1. Phép gán
2. Phép tăng giảm địa chỉ
3. Phép truy nhập bộ nhớ
4. Phép so sánh

### 2.3.1 Phép gán

Chỉ nên thực hiện phép gán cho các con trỏ cùng kiểu

Ví dụ:

```
int x=1, *pi, *qi;
```

```
pi = &x;
```

```
qi = pi;
```

Câu lệnh thứ 3 sao chép nội dung của pi vào qi, nhờ vậy mà pi và qi trỏ đến cùng một đối tượng (ở đây là biến x).

Muốn gán các con trỏ khác kiểu phải dùng phép ép kiểu.

Ví dụ:

```
int x;
```

```
char *pc;
```

```
pc = (char*) (&x); // ép kiểu
```

### 2.3.2 Phép tăng giảm địa chỉ

Ta có thể cộng (+), trừ (-) 1 con trỏ với 1 số nguyên N nào đó; kết quả trả về là 1 con trỏ. Con trỏ này chỉ đến vùng nhớ cách vùng nhớ của con trỏ hiện tại N phần tử.

Ví dụ:

```
float x[30], *px;
```

```
px = &x[10];
```

Cho biết px là con trỏ float trỏ đến phần tử x[10]

Khi đó:

px++ trỏ đến phần tử x[11];

px+i trỏ đến phần tử x[10+i]

px-i trỏ đến phần tử x[10-i] .

Phép trừ 2 con trỏ cùng kiểu sẽ trả về 1 giá trị nguyên (int).

Đây chính là khoảng cách (số phần tử) giữa 2 con trỏ đó.

Chẳng hạn

trong ví dụ trên pc-pa=4.

Con trỏ NULL: là con trỏ không chứa địa chỉ nào cả. Ta có thể gán giá trị NULL cho 1 con trỏ có kiểu bất kỳ.

Lưu ý: Ta không thể cộng 2 con trỏ với nhau.

### 2.3.3 Phép truy nhập bộ nhớ

Nguyên tắc: con trỏ float truy nhập 4 byte. Con trỏ int truy nhập 2 byte. Con trỏ char truy nhập 1 byte.

Ví dụ:

```
float *pf;
```

```
int *pi;
```

```
char *pc;
```

Khi đó:

- Nếu `pf` trỏ đến byte thứ 10001, thì `*pf` biểu thị vùng nhớ 4 byte liên tiếp từ byte 10001 đến byte 10004.
- Nếu `pi` trỏ đến byte thứ 10001, thì `*pi` biểu thị vùng nhớ 2 byte liên tiếp từ byte 10001 đến byte 10002
- Nếu `pc` trỏ đến byte thứ 10001 thì `*pc` biểu thị vùng nhớ 1 byte là byte 10001.

Chú ý: 2 phép toán trên không dùng cho con trỏ kiểu void

### 2.3.4 Phép so sánh

1. Cho phép so sánh 2 con trỏ cùng kiểu
2. `p1 < p2` nếu địa chỉ `p1` trỏ tới thấp hơn địa chỉ `p2` trỏ tới
3. `p1 = p2` nếu địa chỉ `p1` trỏ tới bằng địa chỉ `p2` trỏ tới
4. `p1 > p2` nếu địa chỉ `p1` trỏ tới cao hơn địa chỉ `p2` trỏ tới.

## 2.4 SỬ DỤNG CON TRỎ ĐỂ CẤP PHÁT VÀ THU HỒI BỘ NHỚ ĐỘNG

Để cấp phát bộ nhớ động, ta sử dụng các hàm trong thư viện `stdlib.h` hoặc `alloc.h`.

1. `malloc`
2. `calloc`
3. `realloc`
4. toán tử `new`

Lưu ý : khi cấp phát bộ nhớ cho 1 biến con trỏ, nếu chúng ta sử dụng các hàm `malloc`, `calloc`, `realloc`, thì khi giải phóng bộ nhớ, ta phải sử dụng hàm `free`.

Nếu chúng ta sử dụng toán tử `new` để xin cấp phát bộ nhớ, thì khi giải phóng bộ nhớ ta phải dùng toán tử `delete`

## 2.4.1 Các hàm cấp phát vùng nhớ

### 2.4.1.1 Hàm malloc

Cú pháp: `void *malloc(size_t n);`

Hàm xin cấp phát vùng nhớ cho `n` phần tử, mỗi phần tử có kích thước là `size_t`. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp phát. Khi không đủ vùng nhớ để cấp phát hàm trả về trị `NULL`.

Ví dụ: dùng hàm malloc

```
#include <stdio.h>

#include <string.h>

#include <alloc.h>

#include <process.h>

int main( ) {

    char *str;

    /* allocate memory for string */

    str = (char *) malloc(10);

    if(str==NULL){

        printf("Not enough memory to allocate buffer\n");

        exit(1); /* terminate program if out of memory */

    }

    strcpy(str, "Hello"); /* copy "Hello" into string */

    printf("String is %s\n", str); /* display string */

    free(str); /* free memory */

    return 0;

}
```



### 2.4.1.2 Hàm calloc

Cú pháp: `void *calloc(size_t nItems, size_t size);`

Cấp phát vùng nhớ `nItems*size` byte. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp phát. Khi không đủ bộ nhớ để cấp phát hàm trả về giá trị NULL.

Ví dụ: dùng hàm calloc

```
#include <stdio.h>

#include <alloc.h>

#include <string.h>

int main() {

    char *str = NULL;

    /* allocate memory for string */

    str = (char *) calloc(10, sizeof(char));

    strcpy(str, "Hello"); /* copy "Hello" into string */

    printf("String is %s\n", str); /* display string */

    free(str); /* free memory */

    return 0;

}
```

Lưu ý:

1. Hàm calloc cấp phát vùng nhớ và khởi tạo tất cả các bit trong vùng nhớ mới cấp phát về 0.
2. Hàm malloc chỉ cấp phát vùng nhớ.

### 2.4.1.3 Hàm realloc

Cú pháp: `void* realloc(void *ptr, unsigned size);`

Trong đó:

- ptr: trỏ đến vùng nhớ đã được cấp phát trước đó.

- size: là số byte cần cấp phát lại.

Hàm thay đổi kích thước vùng nhớ đã cấp phát trước đó. Vùng nhớ mới có thể có địa chỉ khác so với vùng nhớ cũ. Phần dữ liệu trên vùng nhớ cũ được chuyển đến vùng nhớ mới.

Ví dụ: `int a, *pa;`

```
pa = (int*) malloc (10) ; /*Xin cấp phát vùng nhớ có kích thước (10 x 2 ) byte*/
```

```
pa = realloc (pa, 16); /* Xin cấp phát lại vùng nhớ có kích thước 16 x2 byte*/
```

### 2.4.2 Toán tử new và delete (trong C++)

Mục này không phải kiến thức của C chuẩn.

Để cấp phát động trong C++, ta sử dụng toán tử new

Cú pháp:

```
Kiểu_dữ_liệu biến_con_trỏ = new kiểu_dữ_liệu ;
```

Ở đây, kiểu\_dữ\_liệu có thể là bất kỳ kiểu dữ liệu có sẵn nào ví dụ như các kiểu cơ sở (int, float...), mảng hoặc các kiểu dữ liệu tự định nghĩa như cấu trúc.

Đầu tiên, chúng ta xét các kiểu dữ liệu cơ sở có sẵn. Ví dụ, chúng ta có thể định nghĩa một con trỏ tới kiểu double và sau đó yêu cầu bộ nhớ được cấp phát tại thời gian thực thi. Chúng ta có thể làm điều này bởi sử dụng toán tử new trong C/C++ với các lệnh sau:

```
double *p = NULL;
```

```
p = new double;
```

Bộ nhớ có thể chưa được cấp phát thành công, do vậy ta có thể kiểm tra nếu toán tử new trả về con trỏ NULL và đưa ra thông báo lỗi như sau:

```
double *p = NULL;
```

```
if (!(p = new double)) {
```

```
    printf("Lỗi cấp phát bộ nhớ!");
```

```
    exit(1);
```

```
}
```

Tại bất kỳ thời điểm nào, khi ta thấy một biến đã được cấp phát động là không cần thiết nữa, ta có thể giải phóng bộ nhớ mà nó đã chiếm giữ trong phần bộ nhớ với toán tử `delete` trong C/C++, như sau:

Cú pháp:

```
delete biến_con_trỏ ;
```

### 2.4.3 Thu hồi bộ nhớ động

1. Khi sử dụng con trỏ để xin cấp phát bộ nhớ thì người lập trình bắt buộc phải trả lại bộ nhớ.
2. Để thu hồi bộ nhớ ta có thể dùng hàm `free` hoặc toán tử `delete`.

### 2.4.4 Toán tử `sizeof`:

Toán tử `sizeof` cho ta biết kích thước (tính theo byte) của một kiểu dữ liệu hay một đối tượng dữ liệu.

Kiểu dữ liệu có thể là kiểu chuẩn (`int`, `float`, ...) hay kiểu dữ liệu được định nghĩa trong chương trình (`typedef`, `enum`, `struct`, `union`, ...).

Đối tượng dữ liệu bao gồm tên biến, tên mảng, biến `struct`, ...

Khai báo:

```
sizeof (<đối tượng dữ liệu> )
```

```
sizeof (<kiểu dữ liệu> )
```

Ví dụ:

```
typedef float KieuThuc;
```

```
struct DiemThi {
```

```
    char masv[8];
```

```
    char mamh[5];
```

```
    int lanthi;
```

```
    float diem;
```

```
} x;
```

```
sizeof (int)           cho trị 2
```

```
sizeof (KieuThuc)     cho trị 4
```

```
sizeof (x)            cho trị 19
```

## 2.5 CON TRỎ VÀ MẢNG MỘT CHIỀU

---

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

### 2.5.1 Truy cập các phần tử mảng theo dạng con trỏ

Ta có các quy tắc sau:

`&<Tên mảng>[0]` tương đương với `<Tên mảng>`

`&<Tên mảng> [<Vị trí>]` tương đương với `<Tên mảng> + <Vị trí>`

`<Tên mảng>[<Vị trí>]` tương đương với `*(<Tên mảng> + <Vị trí>)`

Ví dụ: Cho 1 mảng 1 chiều các số nguyên a có n phần tử, truy cập các phần tử theo kiểu mảng và theo kiểu con trỏ.

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

```
#include <conio.h>
```

```
/* Cách 1: Nhập mảng bình thường*/
```

```
void NhapMang (int a[], int n)
```

```
{
```

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

```
    {
```

```
        printf("Phan tu thu %d: ", i);
```

```
        scanf("%d", &a[i]);
```

```
    }
```

```

}

/*Cách 2: Nhập mảng theo dạng con trỏ*/
void NhapContro (int *a, int n)
{
    for(i=0; i<n; i++)
    {
        printf("Phan tu thu %d: ", i);
        scanf("%d", a+i);
    }
}

```

### 2.5.2 Truy cập từng phần tử đang được quản lý bởi con trỏ theo dạng mảng

<Tên biến>[<Vị trí>] tương đương với \*(<Tên biến> + <Vị trí>)  
 &<Tên biến>[<Vị trí>] tương đương với (<Tên biến> + <Vị trí>) Trong đó  
 <Tên biến> là biến con trỏ,  
 <Vị trí> là 1 biểu thức số nguyên.

Ví dụ: Giả sử có khai báo:

```

#include <stdio.h>
#include <alloc.h>
#include <conio.h>

int main()
{
    int *a;

    a = (int*)malloc(10);

    for(i=0; i<10; i++)

```

```

        a[i] = 2*i;

printf("Truy cap theo kieu mang: ");

for(i=0; i<10; i++)

    printf("%d ",a[i]);

printf("\nTruy cap theo kieu con tro: ");

for(i=0; i<10; i++)

    printf("%d ",*(a+i));

return 0;

}

```

Kết quả chương trình:



```

CA TC.EXE
Truy cap theo kieu mang: 0 2 4 6 8 10 12 14 16 18
Truy cap theo kieu con tro: 0 2 4 6 8 10 12 14 16 18

```

### 2.5.3 Bài toán minh họa

Dùng phương pháp con trỏ viết chương trình thực hiện các yêu cầu sau:

1. Nhập vào một mảng số nguyên gồm  $n$  phần tử.
2. Xuất ra mảng số nguyên gồm  $n$  phần tử.
3. Tính tổng các phần tử có trong mảng số nguyên gồm  $n$  phần tử.

Cách thực hiện:

```

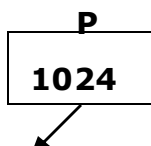
int *p; // khai báo biến con trỏ p để quản lý mảng n phần tử

p = (int *) malloc (n); // xin cấp phát (n x 2) byte cho biến p quản lý.

Hoặc p = new int[n];

```

Hình ảnh mảng số nguyên lưu trong bộ nhớ do con trỏ  $p$  quản lý



P	P +1	P +2	P +3	P +4	P +5	P +6	P +7	....	P+n-1
1	2	3	4	5	6	7	8		n
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]		a[n-1]

Cài đặt:

1. Hàm nhập mảng dùng phương pháp con trỏ:

```
void Nhapmang (int *a, int n)
```

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

2. Hàm xuất mảng dùng phương pháp con trỏ:

```
void Xuatmang ( int *a, int n)
```

```
{
    for (int i=0; i<n; i++)
        printf ("%4d", *(a+i));
}
```

3. Hàm tính tổng các phần tử trong mảng dùng phương pháp con trỏ:

```
long Tongmang ( int *a, int n)
```

```
{
    long s = 0;
    for (int i=0; i<n; i++)
        s = s + *(a+i));
    return s;
}
```

4. Hàm main gọi thực hiện các hàm trên:

```

int main()
{
    int *p;

    p = (int*)malloc (100);

    // hoac p = new int[100];

    //... gọi thực hiện các hàm

    return 0;
}

```

## 2.6 CON TRỎ VÀ MẢNG HAI CHIỀU

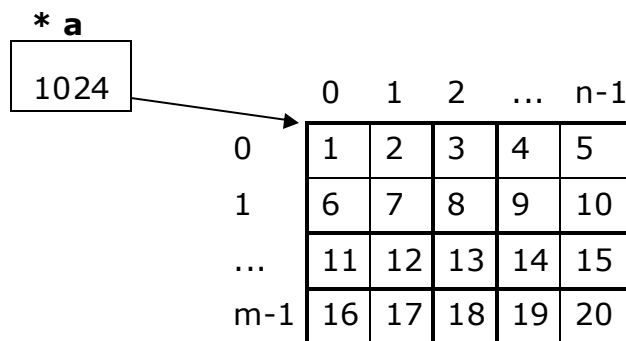
### 2.6.1 Bài toán minh họa

Dùng phương pháp con trỏ viết chương trình thực hiện các yêu cầu sau:

1. Nhập vào một ma trận số nguyên gồm m dòng, n cột có mxn phần tử (m, n ≤ 100).
2. Xuất ma trận số nguyên.
3. Tính tổng các phần tử có trong ma trận.

### 2.6.2 Cách 1

int \*a; // khai báo con trỏ a để quản lý ma trận m dòng, n cột



`a = (int *) malloc (m*n)) // xin cấp phát bộ nhớ để quản lý ma trận`

hoặc: `a = new int[m][n];`



Khi đó

Hình ảnh ma trận  $m \times n$  số nguyên lưu trong bộ nhớ do con trỏ  $a$  quản lý

$a$	$a + 1$	$a + 2$	$a + 3$	$a + 4$	$a + 5$	$a + 6$	...	$a + m \times n - 1$
1	2	3	4	5	6	7		$m \times n$
$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$	$a[0][4]$	$a[0][5]$	$a[0][6]$		$a[m-1][n-1]$

Để truy cập đến phần tử  $a[i][j]$  của mảng  $a$ , ta dùng công thức sau :

$*(a + i * n + j);$

Cài đặt:

1. Hàm nhập ma trận:

`void NhapMang (int * a, int m, int n)`

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

2. Hàm xuất ma trận:

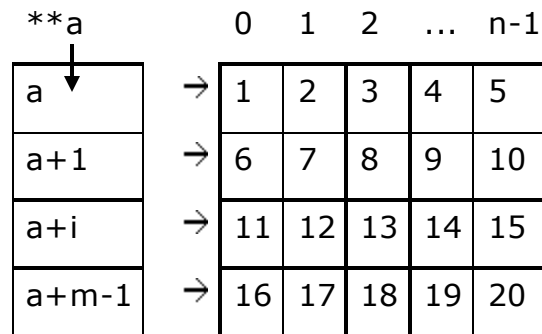
`void XuatMang (int * a, int m, int n)`

```
{
    for (int i=0; i<m; i++)
    {
        for (int j=0; j<n; j++)
            printf ("%4d", *(a+i *n +j));
        printf("\n");
    }
}
```

3. Hàm tính tổng các phần tử trong ma trận:

```
long TinhTong (int * a, int m, int n)
{
    for (int i=0; i<m; i++)
        for (int j=0; j<n; j++)
            s = s+ *(a+i *n +j);
}
```

### 2.6.3 Cách 2



1. Con trỏ \*a quản lý các phần tử dòng 0:  $a[0][0]$ ,  $a[0][1]$ ,  $a[0][2]$ , ...,  $a[0][n-1]$

*a →	$a[0][0]$	$a[0][1]$	...	$a[0][n-1]$
------	-----------	-----------	-----	-------------

2. Con trỏ \*(a+1) quản lý các phần tử dòng 1 :  $a[1][0]$ ,  $a[1][1]$ ,  $a[1][2]$ , ...,  $a[1][n-1]$

*(a+1) →	$a[1][0]$	$a[1][1]$	...	$a[1][n-1]$
----------	-----------	-----------	-----	-------------

3. Con trỏ \*(a+i) quản lý các phần tử dòng i :  $a[i][0]$ ,  $a[i][1]$ ,  $a[i][2]$ , ...,  $a[i][n-1]$

*(a+i) →	$a[i][0]$	$a[i][1]$	...	$a[i][n-1]$
----------	-----------	-----------	-----	-------------

4. Con trỏ \*(a+m-1) quản lý các phần tử dòng m-1:  $a[m-1][0]$ ,  $a[m-1][1]$ ,  $a[m-1][2]$ , ...,  $a[m-1][n-1]$

*(a+m-1) →	$a[m-1][0]$	$a[m-1][1]$	...	$a[m-1][n-1]$
------------	-------------	-------------	-----	---------------

Xin cấp phát cho con trỏ \*\*a để quản lý mảng con trỏ : \*a, \*(a+1), \*(a+2) ..., \*(a+m-1).

<b>** a</b> →	<b>* a</b>	<b>* (a+1)</b>	<b>* (a+2)</b>	<b>...</b>	<b>*(a+m-1)</b>
---------------	------------	----------------	----------------	------------	-----------------

Cài đặt:

1. Hàm nhập ma trận:

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

2. Hàm xuất ma trận:

```
void XuatMang (int ** a, int m, int n) {
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++)
            printf ("%4d", *( *( a+i ) +j));
        printf("\n");
    } }
}
```

3. Hàm tính tổng các phần tử trong ma trận:

```
long TinhTong (int ** a, int m, int n) {
    long s=0;
    for (int i=0; i<m; i++)
        for (int j=0; j<n; j++)
            s = s + (*( *( a+i ) +j) );
    return s;
}
```

4. Hàm giải phóng bộ nhớ:

```
void FreeArray (int **a, int m)
{
    for (int i=0; i<m; i++)
```

```
    free (a[i]) ;  
    free (a);  
}
```

## 2.7 CON TRÒ VÀ CHUỖI KÝ TỰ

Ta đã biết, chuỗi là một dãy ký tự dùng để lưu trữ và xử lý văn bản như từ, cụm từ, câu. Trong ngôn ngữ C không có kiểu chuỗi và chuỗi được thể hiện bằng mảng các ký tự (có kiểu cơ sở char), được kết thúc bằng ký tự '\0' (còn được gọi là ký tự NULL trong bảng mã ASCII).

Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép " ".

### 2.7.1 Cách khai báo chuỗi

❖ Khai báo chuỗi:

**Cách 1:**

**char < tên biến> [chiều dài tối đa chuỗi];**

- Ví dụ:

char Hoten [20];

**Cách 2:**

**char \*< tên biến> ;**

- Ví dụ:

char \*Hoten;

Hoten = (char \*) malloc (20\*sizeof(char));

Hoặc Hoten = new char[20];

### 2.7.2 Cách nhập/ xuất chuỗi ký tự

Cách 1: Nhập xuất chuỗi dùng thư viện <stdio.h>

❖ Nhập:

**scanf("format string", argument\_list);**

Ví dụ: `scanf ("%s", hoten);`

Đối với hàm `scanf` khi gặp phím `space`, `tab`, `new line`, `enter` thì dừng, cho nên chỉ dùng hàm `scanf` để nhập chuỗi không có khoảng trắng.

❖ **Xuất:**

**`printf("format string", argument_list);`**

Ví dụ: `printf ("%s", hoten);`

Cách 2: Nhập xuất chuỗi tùy ý

❖ **Nhập:**

**`gets (string);`**

Tiếp nhận được `space`, `tab`, `new line`.

Gặp `enter` thì dừng, phải khai báo hàm xóa bộ đệm bàn phím trước khi dùng hàm `gets`: `fflush(stdin)` hay `flushall()`

Ví dụ:

`gets(hoten);`

❖ **Xuất:**

**`puts (string);`**

**Hàm này xuất chuỗi xong tự động xuống dòng.**

Ví dụ:

**`puts(hoten);`**

### 2.7.3 Một số hàm xử lý chuỗi (trong `<string.h>`)

1. Hàm `strcat`: dùng để nối hai chuỗi lại với nhau.

Cú pháp:

`char * strcat(char *s1, char *s2);`

Hàm có công dụng ghép nối hai chuỗi s1 và s2 lại với nhau; kết quả ghép nối được chứa trong s1.

Ví dụ:

```
#include "stdio.h"
#include "string.h"
int main() {
    char s1[50], s2[50];
    printf("\nNhap chuoi 1: ");
    gets(s1);
    printf("Nhap chuoi 2: ");
    gets(s2);
    strcat(s1,s2);
    printf("Xuat chuoi 1: %s",s1);
    printf("\nXuat chuoi 2: %s",s2);
    return 0;
}
```

**2. Hàm strchr:** Tìm lần xuất hiện đầu tiên của ký tự trong chuỗi.

Cú pháp:

```
char * strchr (char *ch, int kt);
```

Hàm có tác dụng tìm lần xuất hiện đầu tiên của ký tự kt trong chuỗi ch. Nếu tìm thấy hàm trả về địa chỉ của ký tự được tìm thấy trong chuỗi ch, trái lại hàm trả về giá trị NULL.

Ví dụ:

```
#include "stdio.h"
#include "string.h"
int main() {
    char s[50], ch, *p;
    printf("\nNhap chuoi: ");
    gets(s);
    printf("Nhap ky tu: ");
    ch=getche();
```

```

        p=strchr(s,ch);
        if (p!=NULL)
            printf("\nchi so cua ky tu: %d",(int)(p-s));
        else
            printf("\nKhong tim thay!");
        return 0;
    }

```

### 3. Hàm strcmp: so sánh hai chuỗi.

Cú pháp:

```
int strcmp (char *s1, char *s2);
```

Hàm có công dụng so sánh hai chuỗi s1 và s2.

- Nếu hàm trả về giá trị <0 thì chuỗi s1 nhỏ hơn chuỗi s2.
- Nếu hàm trả về giá trị 0 nếu chuỗi s1 bằng chuỗi s2.
- Nếu hàm trả về giá trị >0 thì chuỗi s1 lớn hơn chuỗi s2.

### 4. Hàm stricmp(): So sánh chuỗi

Hàm này thực hiện việc so sánh trong n ký tự đầu tiên của 2 chuỗi s1 và s2, giữa chữ thường và chữ hoa không phân biệt.

Cú pháp:

```
int stricmp (const char *s1, const char *s2);
```

Kết quả trả về tương tự như kết quả trả về của hàm strcmp()

### 5. Hàm strcpy: sao chép chuỗi.

Hàm được dùng để sao chép toàn bộ nội dung của chuỗi nguồn vào chuỗi đích.

Cú pháp:

```
char * strcpy (char *Des, const char *Source);
```

Ví dụ:

```

#include "stdio.h"
#include "string.h"
int main() {

```

```
char s[50];  
strcpy(s, "Truong Dai hoc Ky thuat");  
printf("\nXuat chuoi: %s", s);  
return 0;  
}
```

#### 6. Hàm strncpy(): Sao chép một phần chuỗi

Hàm này cho phép chép n ký tự đầu tiên của chuỗi nguồn sang chuỗi đích.

Cú pháp:

```
char * strncpy(char *Des, const char *Source, size_t n);
```

#### 7. Hàm strchr(): Trích một phần chuỗi

Để trích một chuỗi con của một chuỗi ký tự bắt đầu từ một ký tự được chỉ định trong chuỗi cho đến hết chuỗi, ta sử dụng hàm strchr().

Cú pháp:

```
char *strchr (const char *str, int c);
```

Lưu ý:

Nếu ký tự đã chỉ định không có trong chuỗi, kết quả trả về là NULL.

Kết quả trả về của hàm là một con trỏ, con trỏ này chỉ đến ký tự c được tìm thấy đầu tiên trong chuỗi str.

#### 8. Hàm strstr()

Hàm strstr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1.

Cú pháp:

```
char * strstr(const char *s1, const char *s2);
```

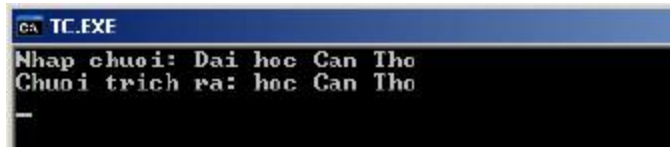
Kết quả trả về của hàm là một con trỏ chỉ đến phần tử đầu tiên của chuỗi s1 có chứa chuỗi s2 hoặc giá trị NULL nếu chuỗi s2 không có trong chuỗi s1.

Ví dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi "hoc".

```
#include <conio.h>
```



```
#include<stdio.h>
#include<string.h>
int main() {
    char Chuoi[255], *s ;
    printf("Nhap chuoi: ");gets(Chuoi);
    s=strstr(Chuoi,"hoc");
    printf("Chuoi trích ra: ");puts(s);
    return 0;
}
```



## 9. Hàm strlen(): Lấy chiều dài chuỗi

Cú pháp:

```
int strlen (const char *s);

#include <stdio.h>
#include <string.h>
int main() {
    char string [ ] = "Borland International";
    printf("%d\n", strlen(string));    // kết quả 21
    return 0;    }
```

Ví dụ: Gán một chuỗi vào chuỗi khác

Ta gán từng ký tự trong chuỗi.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main() {
    char newstr [35];
    char str[] = " Trường Đại học Công Nghệ Tp.HCM";
    for(int i = 0; i<strlen(str); i++)
        newstr[i] = str[i];
}
```

```

        newstr[i] = '\0';
        return 0;
    }

```

### 10. Đổi một ký tự thường thành ký tự hoa - Hàm toupper()

Hàm toupper() (trong ctype.h) được dùng để chuyển đổi một ký tự thường thành ký tự hoa.

Cú pháp: char toupper (char c)

### 11. Đổi chuỗi chữ thường thành chuỗi chữ hoa - Hàm strupr()

Hàm strupr() được dùng để chuyển đổi chuỗi chữ thường thành chuỗi chữ hoa, kết quả trả về của hàm là một con trỏ chỉ đến địa chỉ chuỗi được chuyển đổi.

Cú pháp: char \*strupr(char \*s)

Ví dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm strupr() để chuyển đổi chúng thành chuỗi chữ hoa.

```

#include<conio.h>

#include<stdio.h>

#include<string.h>

int main() {
    char Chuoi[255],*s;
    printf("Nhap chuoi: "); gets(Chuoi);
    s=strupr(Chuoi) ;
    printf("Chuoi chu hoa: "); puts(s);
    return 0;
}

```

### 12. Đổi chuỗi chữ hoa thành chuỗi chữ thường - Hàm strlwr()

Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàm strlwr(), các tham số của hàm tương tự như hàm strupr()

Cú pháp: char \*strlwr (char \*s)

### 13. Đổi từ chuỗi ra số, hàm atoi(), atof(), atol() (trong stdlib.h)

Để chuyển đổi chuỗi ra số, ta sử dụng các hàm trên.

Cú pháp:

`int atoi(const char *s)`: chuyển chuỗi thành số nguyên

`long atol(const char *s)`: chuyển chuỗi thành số nguyên dài float

`atof(const char *s)`: chuyển chuỗi thành số thực

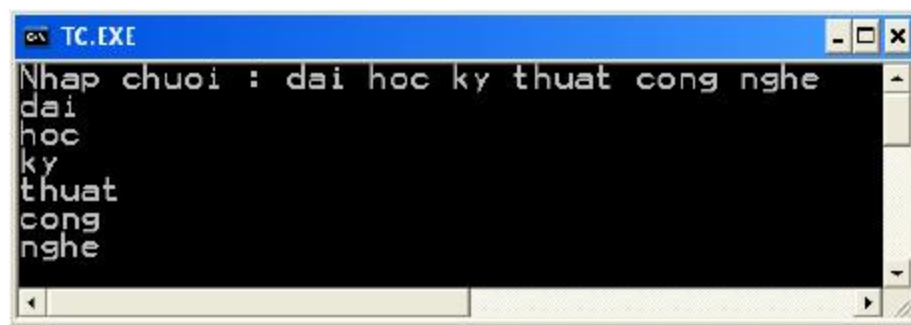
Nếu chuyển đổi không thành công, kết quả trả về của các hàm là 0.

#### 14. `char* strtok (char *s1, const char *s2)`

Xem s1 là 1 loạt chuỗi con, ngăn cách nhau bởi 1 hay nhiều ký tự có trong s2.

Ví dụ:

```
#include <string.h>
int main() {
    char s[80], *p ;
    gets(s);
    p = strtok(s, " ");
    if (p) printf("%s", p);
    while(p) {
        p = strtok(NULL, " ");
        if (p) printf("%s", p);
    }
    return 0;
}
```



#### 15. Đảo ngược chuỗi: `char* strrev(char *s)`

Ngoài ra, thư viện `string.h` còn hỗ trợ các hàm xử lý chuỗi khác, ta có thể đọc thêm trong phần trợ giúp.

## 2.8 CON TRỎ VỚI KIỂU DỮ LIỆU CÓ CẤU TRÚC (STRUCT)

---

### 2.8.1 Ví dụ 1

```
typedef struct DIEM
{
    char masv[8];
    char mamh[5];
    int lanthi;
    float diem;
};

DIEM *p, x;

/* p là con trỏ kiểu struct và x là biến struct */

P = &x;      /* con trỏ p chứa địa chỉ của biến x */
```

Để truy cập đến các thành phần của struct thông qua con trỏ ta dùng một trong hai cách sau:

Cách 1:

Cú pháp                      <tên con trỏ> -> <tên thành phần>

Ví dụ:                      printf("%f ", p->diem);

Cách 2:

Cú pháp                      (\*<tên con trỏ>).<tên thành phần>

Ví dụ :                      printf("%f ", (\*p).diem);

### 2.8.2 Ví dụ 2

Viết chương trình nhập vào điểm, và xuất kết quả ra màn hình.

```
#include "stdio.h"
```

```
#include "conio.h"
```

```
struct DIEM {  
    char masv[8];  
    char mamh[5];  
    int lanthi;  
    float diem;  
};  
  
int main() {  
    struct DIEM x,*p;  
    float tam;  
    p=&x;  
    printf("\nNhap ma so sinh vien :");  
    gets(p->masv);  
    printf("Nhap ma mon hoc : ");  
    gets(p->mamh);  
    printf("Lan thi :");  
    scanf("%d ", &p->lanthi);  
    printf("diem :");  
    scanf("%f ", &tam);  
    p->diem=tam;  
    printf("\nKet qua:");  
    printf("\nMa so sinh vien: %s", (*p).masv);  
    printf("\nMa mon hoc: %s", (*p).mamh);  
    printf("\nLan thi: %d", (*p).lanthi);  
    printf("\ndiem: %.2f ", (*p).diem);  
    return 0;  
}
```

### 2.8.3 Truyền tham số kiểu cấu trúc cho hàm

Ví dụ: Thông tin về điểm của một sinh viên bao gồm: mã số sinh viên, mã số môn học, lần thi, và điểm môn học. Viết hàm nhập và xuất điểm của một sinh viên.

```
#include "stdio.h"

#include "conio.h"

typedef struct DIEMTHI
{
    char masv[8];
    char mamh[5];
    int lanthi;
    float diem;
};

void nhap(DIEMTHI *px);
void xuat(DIEMTHI x);
int main()
{
    DIEMTHI x;
    printf("\nNhap diem thi");
    nhap(&x);
    printf("\nXuat diem thi");
    xuat(x);
    getch();
}

void xuat( DIEMTHI x)
{
    printf("\nMa sinh vien :%s", x.masv);
    printf("\nMa mon hoc:%s", x.mamh);
    printf("\nLan thi:%d", x.lanthi);
```

```
        printf("\nDiem thi: %.2f", x.diem);
    }
    void nhap(DIEMTHI *px)
    {
        float tam;
        printf("\nMa sinh vien :");
        gets(px->masv);
        printf("Ma mon hoc:");
        gets(px->mamh);
        printf("Lan thi:");
        scanf("%d%c", &px->lanthi);
        printf("Diem thi:");
        scanf("%f%c", &tam);
        px->diem=tam;
    }
```

## TÓM TẮT

*Bài này cung cấp cho học viên*

- *Khái niệm về con trỏ;*
- *Cách khai báo và sử dụng biến kiểu con trỏ;*
- *Mảng và các phép toán trên mảng một chiều theo kiểu con trỏ;*
- *Mảng và các phép toán trên mảng hai chiều theo kiểu con trỏ;*
- *Đi sâu vào các giải thuật trên mảng 1 chiều , 2 chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...theo kiểu con trỏ;*
- *Con trỏ với kiểu dữ liệu có cấu trúc.*

## CÂU HỎI ÔN TẬP

Dùng phương pháp con trỏ làm các bài tập sau:

**Câu 1:** Viết hàm thao tác trên mảng một chiều các số nguyên:

- Viết hàm nhập vào một mảng một chiều các số nguyên gồm  $n$  phần tử ( $0 < n < 100$ )
- Viết hàm xuất mảng số nguyên  $n$  phần tử vừa nhập ở trên
- Tính tổng các phần tử có trong mảng
- Tính tổng các phần tử chẵn có trong mảng
- Tính tổng các phần tử lẻ có trong mảng
- Tính tổng các phần tử nguyên tố có trong mảng
- Tìm phần tử chẵn đầu tiên có trong mảng
- Tìm phần tử lẻ đầu tiên có trong mảng
- Tìm phần tử nguyên tố đầu tiên có trong mảng
- Tìm phần tử chẵn cuối cùng có trong mảng
- Tìm phần tử chính phương cuối cùng có trong mảng



- l. Tìm phần tử lớn nhất có trong mảng
- m. Đếm số phần tử chẵn có trong mảng
- n. Đếm số phần tử lớn nhất có trong mảng
- o. In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- p. Thêm một phần tử vào đầu mảng.
- q. Thêm một phần tử vào cuối mảng.
- r. Thêm một phần tử vào vị trí x trong mảng.
- s. Xóa phần tử chẵn đầu tiên.
- t. Xóa tất cả các phần tử lớn nhất trong mảng
- u. Sắp xếp mảng tăng dần

**Câu 2:** Viết hàm thao tác trên mảng hai chiều các số nguyên:

- a. Viết hàm nhập vào một mảng hai chiều các số thực gồm m dòng, n cột ( $0 < m, n < 100$ )
- b. Viết hàm xuất mảng hai chiều các số nguyên  $m \times n$  phần tử vừa nhập ở trên
- c. Viết hàm xuất mảng hai chiều các số thực  $m \times n$  phần tử vừa nhập ở trên
- d. Tính tổng các phần tử có trong mảng
- e. Tính tổng các phần tử chẵn có trong mảng
- f. Tính tổng các phần tử lẻ có trong mảng
- g. Tính tổng các phần tử nguyên tố có trong mảng
- h. Tính tổng các phần tử nằm trên đường chéo chính có trong mảng
- i. Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng
- j. Tính tổng các phần tử nằm trên đường biên có trong mảng
- k. Tìm phần tử chẵn đầu tiên có trong mảng
- l. Tìm phần tử lẻ đầu tiên có trong mảng
- m. Tìm phần tử nguyên tố đầu tiên có trong mảng

- n. Tìm phần tử chẵn cuối cùng có trong mảng
- o. Tìm phần tử chính phương cuối cùng có trong mảng
- p. Tìm phần tử lớn nhất có trong mảng
- q. Đếm số phần tử chẵn có trong mảng
- r. Đếm số phần tử lớn nhất có trong mảng
- s. In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- t. Tính tổng các phần tử nằm trên một dòng
- u. Tìm dòng có tổng lớn nhất
- v. Sắp xếp mảng tăng dần

**Câu 3:** Viết các hàm sau:

- a. Nhập vào một dãy phân số
- b. Xuất một phân số
- c. Xuất một dãy phân số
- d. Tìm phân số lớn nhất trong dãy phân số
- e. Tính tổng các phân số có trong dãy

**Câu 4:** Viết chương trình thực hiện:

- a. Nhập vào 2 chuỗi dữ liệu s1 và s2.
- b. Xuất ra màn hình hai chuỗi vừa nhập
- c. Xuất ra màn hình độ dài của các chuỗi vừa nhập.
- d. So sánh hai chuỗi s1 và s2
- e. Nối chuỗi s2 vào chuỗi s1
- f. Kiểm tra chuỗi s1 có chứa chuỗi s2 hay không?

Kiểm tra chuỗi s2 có chứa chuỗi s1 hay không?

**Câu 5:** Thông tin về một sinh viên gồm có: Họ tên, mã số sinh viên, ngày tháng năm sinh, giới tính, lớp, điểm toán, điểm lý, điểm tin.

- a. Viết hàm nhập dữ liệu cho một sinh viên.
- b. Viết hàm xuất dữ liệu một sinh viên.
- c. Viết hàm nhập danh sách sinh viên.
- d. Viết hàm xuất danh sách sinh viên.
- e. Xuất thông tin của sinh viên trong danh sách có mã sinh viên nhập từ bàn phím.
- f. Xuất danh sách sinh viên thuộc ngành công nghệ thông tin
- g. Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin
- h. Sắp xếp danh sách sinh viên theo tên
- i. Sắp xếp danh sách sinh viên theo MSSV
- j. Sắp xếp danh sách sinh viên theo điểm toán

## BÀI 3: ĐỆ QUY

Sau khi học xong bài này, học viên có thể

- Hiểu khái niệm về đệ quy, các kiểu đệ quy;
- Biết ưu điểm và nhược điểm khi cài đặt hàm bằng phương pháp đệ quy;
- Biết cách khai báo và viết hàm theo kiểu đệ quy;
- Biết xử lý các giải thuật trên mảng 1 chiều bằng phương pháp đệ quy.

### 3.1 KHÁI NIỆM

---

Đệ quy là một thuật toán dùng để đơn giản hóa những bài toán phức tạp bằng cách phân nhỏ phép toán đó thành nhiều phần đồng dạng.

Qua việc giải những bài toán được phân nhỏ này, những lời giải sẽ được kết hợp lại để giải quyết bài toán lớn hơn.

Một hàm được gọi là đệ quy nếu bên trong thân hàm có lệnh gọi đến chính nó.

Ví Dụ: Tính tổng  $S(n) = 1 + 2 + 3 + \dots + n$ .

Ta có:

$$S(n) = 1 + 2 + 3 + 4 + \dots + (n-1) + n = S(n-1) + n;$$

Cách 1: Phương pháp thứ nhất là dùng vòng lặp (không đệ quy)

long TinhTong (int n)

```
{  
    long s = 0;  
    for(int i = 1; i <= n; i++)  
        s = s + i;  
    return s;  
}
```

Cách 2: Phương pháp thứ hai là dùng hàm đệ quy:

```
long TinhTong (int n) {
    if (n == 0)
        return 0;
    return TinhTong (n - 1) + n;
}
```

## 3.2 KỸ THUẬT GIẢI BÀI TOÁN BẰNG ĐỆ QUY

1. Tham số hóa bài toán.
2. Tìm các điều kiện biên (chặn, dừng), tìm giải thuật cho các tình huống này.
3. Tìm giải thuật tổng quát theo hướng đệ quy lui dần về tình huống bị chặn.

Ví dụ: Tính tổng 1 mảng a gồm n phần tử

- Tham số hóa: int a[ ], int n
- Điều kiện biên: Mảng 0 phần tử thì tổng bằng 0.
- Giải thuật chung:

$$\text{Sum}(a, n) = a[0] + a[1] + \dots + a[n-3] + a[n-2] + a[n-1]$$

$\underbrace{\hspace{10em}}_{\text{Sum}(a, n-1)}$

$$\text{Sum}(a, n) = \begin{cases} 0 & , n = 0 \\ a[n-1] + \text{Sum}(a, n-1), n > 0 \end{cases}$$

Cài đặt:

```
long TongMang (int a[ ], int n) {
    if (n==0) return 0;
    return a[n-1] + TongMang (a, n-1);
}
```

Lưu ý: Với các thuật toán đệ quy trên mảng, ta nên giảm dần số phần tử của mảng.

### 3.3 MỘT SỐ NHẬN XÉT VỀ HÀM ĐỆ QUY

---

1. Hàm đệ quy là hàm mà trong thân hàm lại gọi chính nó.
2. Giải thuật đệ quy đẹp (gọn gàng, dễ chuyển thành chương trình).
3. Tuy nhiên HÀM ĐỆ QUY: Vừa tốn bộ nhớ vừa chạy chậm
4. Nhiều ngôn ngữ không hỗ trợ giải thuật đệ quy (Fortran).
5. Nhiều giải thuật rất dễ mô tả dạng đệ quy nhưng lại rất khó mô tả với giải thuật không-đệ-quy.
6. Hàm đệ quy kém hiệu quả: tốn bộ nhớ và gọi hàm quá nhiều lần. Tuy nhiên viết hàm đệ quy rất ngắn gọn vì vậy tùy từng bài toán cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không (có những trường hợp không dùng đệ quy thì không giải quyết được bài toán).

### 3.4 SO SÁNH CẤU TRÚC LẶP VÀ ĐỆ QUY

---

1. Lập trình đệ quy sử dụng cấu trúc lựa chọn
2. Phương pháp lặp sử dụng cấu trúc lặp.
3. Cả 2 phương pháp đều liên quan đến quá trình lặp, tuy nhiên phương pháp lặp sử dụng vòng lặp một cách tường minh, còn phương pháp đệ quy có được quá trình lặp bằng cách sử dụng liên tục lời gọi hàm.
4. Cả 2 phương pháp đều phải kiểm tra khi nào thì kết thúc.
  - a. Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai.
  - b. Phương pháp đệ quy kết thúc khi đến điều kiện biên.
  - c. Phương pháp lặp thay đổi biến đếm trong vòng lặp cho đến khi nó làm cho điều kiện lặp sai.
  - d. Còn đệ quy làm cho các lời gọi hàm đơn giản dần cho đến khi đơn giản đến điều kiện biên.
5. Cả 2 phương pháp đều có thể dẫn đến trường hợp chạy vô hạn mãi.
  - a. Lặp sẽ không thoát ra được khi điều kiện lặp không bao giờ sai.

- b. Còn đệ qui không thoát ra được khi các bước đệ qui không làm cho bài toán đơn giản hơn và cuối cùng hội tụ về điều kiện biên.
- c. Tuy nhiên đệ qui tồi hơn vì nó liên tục đưa ra lời gọi hàm làm tốn thời gian của bộ vi xử lý và không gian nhớ.

## 3.5 PHÂN LOẠI HÀM ĐỆ QUY

Tùy thuộc cách diễn đạt tác vụ đệ quy mà có các loại đệ quy sau.

- (1) Đệ quy tuyến tính.
- (2) Đệ quy nhị phân.
- (3) Đệ quy phi tuyến
- (4) Đệ quy hỗ tương.

### 3.5.1 Đệ quy tuyến tính

- ❖ Thân hàm gọi 1 lần chính nó
- ❖ Ví dụ:

$$U_n = \begin{cases} a & , n=1 \text{ (trị thứ } n \text{ của cấp số cộng)} \\ r + U_{n-1} & , n>1 \end{cases}$$

Cài đặt hàm như sau:

```
double U (int n, double a, double r)
{
    if (n==1)
        return a;
    return r + U(n-1,a,r);
}
```

### 3.5.2 Đệ quy nhị phân

- ❖ Thân hàm gọi 2 lần chính nó.

❖ Ví dụ: Chuỗi số Fibonacci: 1 1 2 3 5 8 13 ...

$$U_n = \begin{cases} 1 & , n=1, 2 \\ U_{n-2} + U_{n-1} & , n>2 \end{cases}$$

Cài đặt hàm như sau:

```
long Fibo (int n)
```

```
{
    if (n<=2)
        return 1;
    return Fibo(n-2) + Fibo(n-1);
}
```

### 3.5.3 Đệ quy phi tuyến

❖ Thân hàm lặp gọi 1 số lần chính nó

❖ Ví dụ:

$$U_n = \begin{cases} n & , n < 6 \\ U_{n-5} + U_{n-4} + U_{n-3} + U_{n-2} + U_{n-1} & , n > 6 \end{cases}$$

Cài đặt hàm như sau:

```
long U (int n)
```

```
{
    if (n<6)
        return n;

    long S= 0;
    for (int i = 5; i>0; i--)
        S += U(n-i);

    return S;
}
```



### 3.5.4 Đệ quy hỗ trợ

❖ 2 hàm đệ quy gọi nhau

❖ Ví dụ:

$$U_n = \begin{cases} n & , n < 5 \\ U_{n-1} + G_{n-2} & , n \geq 5 \end{cases}$$

$$G_n = \begin{cases} n-3 & , n < 8 \\ U_{n-1} + G_{n-2} & , n \geq 8 \end{cases}$$

Cài đặt hàm như sau:

```
long G(int n);
```

```
long U (int n)
```

```
{
```

```
    if (n<5)
```

```
        return n;
```

```
    return U(n-1) + G(n-2);
```

```
}
```

```
long G(int n)
```

```
{
```

```
    if (n<8)
```

```
        return n-3;
```

```
    return U(n-1) + G(n-2);
```

```
}
```

## CÂU HỎI ÔN TẬP

Dùng phương pháp đệ quy giải các bài toán sau

**Câu 1:** Viết hàm nhập vào một mảng một chiều các số nguyên gồm  $n$  phần tử ( $0 < n < 100$ )

**Câu 2:** Viết hàm xuất mảng số nguyên  $n$  phần tử vừa nhập ở trên

**Câu 3:** Tính tổng các phần tử có trong mảng

**Câu 4:** Tính tổng các phần tử chẵn có trong mảng

**Câu 5:** Tính tổng các phần tử lẻ có trong mảng

**Câu 6:** Tính tổng các phần tử nguyên tố có trong mảng

**Câu 7:** Tìm phần tử chẵn đầu tiên có trong mảng

**Câu 8:** Tìm phần tử lẻ đầu tiên có trong mảng

**Câu 9:** Tìm phần tử nguyên tố đầu tiên có trong mảng

**Câu 10:** Tìm phần tử chẵn cuối cùng có trong mảng

**Câu 11:** Tìm phần tử lớn nhất có trong mảng

**Câu 12:** Đếm số phần tử chẵn có trong mảng

**Câu 13:** Đếm số phần tử lớn nhất có trong mảng

# BÀI 4: TẬP TIN (FILE)

*Sau khi học xong bài này, học viên có thể*

- *Hiểu một số khái niệm về tập tin;*
- *Biết các bước thao tác với tập tin;*
- *Biết sử dụng một số hàm truy xuất đến tập tin văn bản;*
- *Biết sử dụng một số hàm truy xuất đến tập tin nhị phân.*

## 4.1 KHÁI NIỆM

Đối với các kiểu dữ liệu ta đã biết như kiểu số, kiểu mảng, kiểu cấu trúc thì dữ liệu được tổ chức trong bộ nhớ trong (RAM) của máy tính nên khi kết thúc việc thực hiện chương trình thì dữ liệu cũng bị mất; khi cần chúng ta bắt buộc phải chạy lại chương trình. Điều đó vừa mất thời gian vừa không giải quyết được các bài toán với số liệu lớn cần lưu trữ. Để giải quyết vấn đề, người ta đưa ra kiểu tập tin (file) cho phép lưu trữ dữ liệu ở bộ nhớ ngoài. Khi kết thúc chương trình thì dữ liệu vẫn còn do đó chúng ta có thể sử dụng nhiều lần. Một đặc điểm khác của kiểu tập tin là kích thước lớn với số lượng các phần tử không hạn chế (chỉ bị hạn chế bởi dung lượng của bộ nhớ ngoài).

Có 3 loại dữ liệu kiểu tập tin:

- 1. Tập tin văn bản (Text File):** là loại tập tin dùng để ghi các ký tự lên đĩa, các ký tự này được lưu trữ dưới dạng mã ASCII. Điểm đặc biệt là dữ liệu của tập tin được lưu trữ thành các dòng, mỗi dòng được kết thúc bằng ký tự xuống dòng (new line), ký hiệu '\n'; ký tự này là sự kết hợp của 2 ký tự CR (Carriage Return - Về đầu dòng, mã ASCII là 13) và LF (Line Feed - Xuống dòng, mã ASCII là 10). Mỗi tập tin được kết thúc bởi ký tự EOF (End Of File) có mã ASCII là 26 (xác định bởi tổ hợp phím Ctrl + Z).

Tập tin văn bản chỉ có thể truy xuất theo kiểu tuần tự.

2. Tập tin định kiểu (Typed File): là loại tập tin bao gồm nhiều phần tử có cùng kiểu: char, int, long, cấu trúc... và được lưu trữ trên đĩa dưới dạng một chuỗi các byte liên tục.
3. Tập tin không định kiểu (Untyped File): là loại tập tin mà dữ liệu của chúng gồm các cấu trúc dữ liệu mà người ta không quan tâm đến nội dung hoặc kiểu của nó, chỉ lưu ý đến các yếu tố vật lý của tập tin như độ lớn và các yếu tố tác động lên tập tin mà thôi.

Biến tập tin: là một biến thuộc kiểu dữ liệu tập tin dùng để đại diện cho một tập tin. Dữ liệu chứa trong một tập tin được truy xuất qua các thao tác với thông số là biến tập tin đại diện cho tập tin đó.

Con trỏ tập tin: Khi một tập tin được mở ra để làm việc, tại mỗi thời điểm, sẽ có một vị trí của tập tin mà tại đó việc đọc/ghi thông tin sẽ xảy ra. Người ta hình dung có một con trỏ đang chỉ đến vị trí đó và đặt tên nó là con trỏ tập tin.

Sau khi đọc/ghi xong dữ liệu, con trỏ sẽ chuyển dịch thêm một phần tử về phía cuối tập tin. Sau phần tử dữ liệu cuối cùng của tập tin là dấu kết thúc tập tin EOF (End Of File).

## 4.2 CÁC THAO TÁC TRÊN TẬP TIN

Muốn thao tác trên tập tin, ta phải lần lượt làm theo các bước:

1. Khai báo biến tập tin.
2. Mở tập tin bằng hàm `fopen()`.
3. Thực hiện các thao tác xử lý dữ liệu của tập tin bằng các hàm đọc/ghi dữ liệu.
4. Đóng tập tin bằng hàm `fclose()`.

Ở đây, ta thao tác với tập tin nhờ các hàm được định nghĩa trong thư viện `stdio.h`.

### 4.2.1 Khai báo biến tập tin

Cú pháp: `FILE <Danh sách các biến con trỏ>`

Các biến trong danh sách phải là các con trỏ và được phân cách bởi dấu phẩy (,).

Ví dụ: `FILE *f1, *f2;`

### 4.2.2 Mở tập tin

Cú pháp: `FILE *fopen(char *Path, const char *Mode)`

Trong đó:

- Path: chuỗi chỉ đường dẫn đến tập tin trên đĩa.
- Type: chuỗi xác định cách thức mà tập tin sẽ mở. Các giá trị có thể của Mode:

Chế độ	Ý nghĩa
r	Mở tập tin văn bản để đọc
w	Tạo ra tập tin văn bản mới để ghi
a	Nối vào tập tin văn bản
rb	Mở tập tin nhị phân để đọc
wb	Tạo ra tập tin nhị phân để ghi
ab	Nối vào tập tin nhị phân
r+	Mở một tập tin văn bản để đọc/ghi
w+	Tạo ra tập tin văn bản để đọc ghi
a+	Nối vào hay tạo mới tập tin văn bản để đọc/ghi
r+b	Mở ra tập tin nhị phân để đọc/ghi
w+b	Tạo ra tập tin nhị phân để đọc/ghi
a+b	Nối vào hay tạo mới tập tin nhị phân

Hàm `fopen` trả về một con trỏ tập tin. Chương trình của ta không thể thay đổi giá trị của con trỏ này. Nếu có một lỗi xuất hiện trong khi mở tập tin thì hàm này trả về con trỏ `NULL`.

Ví dụ: Mở một tập tin tên `C:\\TEST.txt` để ghi.

```
FILE *f;
f = fopen(" C:\\TEST.txt", "w");
if (f != NULL) {
    /* Các câu lệnh để thao tác với tập tin*/
    /* Đóng tập tin*/
}
```

Trong ví dụ trên, ta có sử dụng câu lệnh kiểm tra điều kiện để xác định mở tập tin có thành công hay không?

Khi mở tập tin để ghi (chế độ "w"), nếu tập tin đã tồn tại rồi thì nội dung của tập tin sẽ bị xóa và một tập tin mới được tạo ra.

Nếu ta muốn ghi nối dữ liệu, ta phải sử dụng chế độ "a".

Khi mở với chế độ đọc, tập tin phải tồn tại rồi, nếu không một lỗi sẽ xuất hiện.

### 4.2.3 Đóng tập tin

Hàm `fclose()` được dùng để đóng tập tin được mở bởi hàm `fopen()`.

Hàm này sẽ ghi dữ liệu còn lại trong vùng đệm vào tập tin và đóng lại tập tin.

Cú pháp: `int fclose (FILE *f)`

Trong đó `f` là con trỏ tập tin được mở bởi hàm `fopen()`.

Giá trị trả về của hàm là 0 báo rằng việc đóng tập tin thành công.

Hàm trả về EOF nếu có xuất hiện lỗi.

Ngoài ra, ta còn có thể sử dụng hàm `fcloseall()` để đóng tất cả các tập tin lại.

Cú pháp: `int fcloseall()`

Kết quả trả về của hàm là tổng số các tập tin được đóng lại.

Nếu không thành công, kết quả trả về là EOF.

### 4.2.4 Kiểm tra đến cuối tập tin hay chưa?

Cú pháp: `int feof (FILE *f)`

Ý nghĩa: Kiểm tra xem đã chạm tới cuối tập tin hay chưa và trả về EOF nếu cuối tập tin được chạm tới, ngược lại trả về 0.

### 4.2.5 Di chuyển con trỏ tập tin về đầu tập tin - Hàm `rewind()`

Khi ta đang thao tác một tập tin đang mở, con trỏ tập tin luôn di chuyển về phía cuối tập tin. Muốn cho con trỏ quay về đầu tập tin như khi mở nó, ta sử dụng hàm `rewind()`.

Cú pháp: `void rewind (FILE *f)`

## 4.3 TRUY CẬP TẬP TIN VĂN BẢN

### 4.3.1 Ghi dữ liệu lên tập tin văn bản

#### ❖ Hàm `putc()`

Hàm này dùng để ghi một ký tự lên một tập tin văn bản đang được mở để làm việc.

Cú pháp: `int putc (int c, FILE *f)`

Trong đó, tham số `c` chứa mã Ascii của một ký tự nào đó. Mã này được ghi lên tập tin liên kết với con trỏ `f`. Hàm này trả về EOF nếu gặp lỗi.

#### ❖ Hàm `fputs()`

Hàm này dùng để ghi một chuỗi ký tự chứa trong vùng đệm lên tập tin văn bản.

Cú pháp: `int puts (const char *buffer, FILE *f)`

Trong đó, `buffer` là con trỏ có kiểu `char` chỉ đến vị trí đầu tiên của chuỗi ký tự được ghi vào. Hàm này trả về giá trị 0 nếu `buffer` chứa chuỗi rỗng và trả về EOF nếu gặp lỗi.

#### ❖ Hàm `fprintf()`

Hàm này dùng để ghi dữ liệu có định dạng lên tập tin văn bản.

Cú pháp: `fprintf (FILE *f, const char *format, varexpr)`

Trong đó:

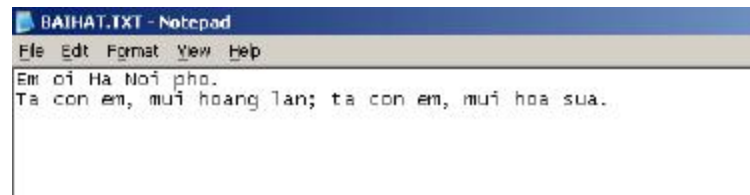
- `format`: chuỗi định dạng (giống với các định dạng của hàm `printf()`).
- `varexpr`: danh sách các biểu thức, mỗi biểu thức cách nhau dấu phẩy (,).

Định dạng	Ý nghĩa
<code>%d</code>	Ghi số nguyên
<code>%.số chữ số thập phân] f</code>	Ghi số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
<code>%o</code>	Ghi số nguyên hệ bát phân
<code>%x</code>	Ghi số nguyên hệ thập lục phân
<code>%c</code>	Ghi một ký tự
<code>%s</code>	Ghi chuỗi ký tự
<code>%e</code> hoặc <code>%E</code> hoặc <code>%g</code> hoặc <code>%G</code>	Ghi số thực dạng khoa học (nhân 10 mũ x)

Ví dụ: Viết chương trình ghi chuỗi ký tự lên tập tin văn bản D:\\Baihat.txt

```
#include<stdio.h>
#include<conio.h>
int main () {
    FILE *f;
    f = fopen ("D:\\Baihat.txt", "r+");
    if (f != NULL) {
        fputs("Em oi Ha Noi pho.\n", f);
        fputs("Ta con em, mui hoang lan; ta con em, mui hoa sua.", f);
        fclose(f);
    }
    return 0;
}
```

Nội dung tập tin Baihat.txt khi được mở bằng trình soạn thảo văn bản Notepad.



### 4.3.2 Đọc dữ liệu từ tập tin văn bản

#### ❖ Hàm `getc()`

Hàm này dùng để đọc dữ liệu từ tập tin văn bản đang được mở để làm việc.

Cú pháp: `int getc (FILE *f)`

Hàm này trả về mã ASCII của một ký tự nào đó (kể cả EOF) trong tập tin liên kết với con trỏ f.

#### ❖ Hàm `fgets()`

Cú pháp: `char *fgets (char *buffer, int n, FILE *f)`

Hàm này được dùng để đọc một chuỗi ký tự từ tập tin văn bản đang được mở ra và liên kết với con trỏ f cho đến khi đọc đủ n ký tự hoặc gặp ký tự xuống dòng '\n' (ký tự này cũng được đưa vào chuỗi kết quả) hay gặp ký tự kết thúc EOF (ký tự này không được đưa vào chuỗi kết quả).



Trong đó:

- **buffer** (vùng đệm): con trỏ có kiểu char chỉ đến vùng nhớ đủ lớn chứa các ký tự nhận được.
- **n**: giá trị nguyên chỉ độ dài lớn nhất của chuỗi ký tự nhận được.
- **f**: con trỏ liên kết với một tập tin nào đó.
- Ký tự NULL ('\0') tự động được thêm vào cuối chuỗi kết quả lưu trong vùng đệm.

Hàm trả về địa chỉ đầu tiên của vùng đệm khi không gặp lỗi và chưa gặp ký tự kết thúc EOF. Ngược lại, hàm trả về giá trị NULL.

### ❖ Hàm **fscanf()**

Hàm này dùng để đọc dữ liệu từ tập tin văn bản vào danh sách các biến theo định dạng.

Cú pháp: `fscanf (FILE *f, const char *format, varlist)`

Trong đó:

**format**: chuỗi định dạng (giống hàm `scanf()`);

**varlist**: danh sách các biến mỗi biến cách nhau dấu phẩy (,).

Ví dụ: Viết chương trình chép tập tin `D:\Baihat.txt` ở trên sang tập tin `D:\Baica.txt`.

```
#include<stdio.h>
#include<conio.h>
int main() {
    FILE *f1,*f2 ;
    f1=fopen("D:\\Baihat.txt", "rt");
    f2=fopen("D:\\Baica.txt", "wt");
    if (f1 != NULL && f2 != NULL) {
        int ch=fgetc (f1);
        while (! feof (f1)) {
            fputc(ch,f2);
            ch=fgetc(f1);
        }
    }
```

```
        fcloseall();  
    }  
    return 0;  
}
```

## 4.4 TRUY CẬP TẬP TIN NHỊ PHÂN

### 4.4.1 Ghi dữ liệu lên tập tin nhị phân - Hàm fwrite()

Cú pháp: `size_t fwrite(const void *ptr, size_t size, size_t n, FILE*f)`

Trong đó:

- `ptr`: con trỏ chỉ đến vùng nhớ chứa thông tin cần ghi lên tập tin.
- `n`: số phần tử sẽ ghi lên tập tin.
- `size`: kích thước của mỗi phần tử.
- `f`: con trỏ tập tin đã được mở.

Giá trị trả về của hàm này là số phần tử được ghi lên tập tin. Giá trị này bằng `n` trừ khi xuất hiện lỗi.

### 4.4.2 Đọc dữ liệu từ tập tin nhị phân - Hàm fread()

Cú pháp: `size_t fread (const void *ptr, size_t size, size_t n, FILE *f)`

Trong đó:

- `ptr`: con trỏ chỉ đến vùng nhớ sẽ nhận dữ liệu từ tập tin.
- `n`: số phần tử được đọc từ tập tin.
- `size`: kích thước của mỗi phần tử.
- `f`: con trỏ tập tin đã được mở.

Giá trị trả về của hàm này là số phần tử đã đọc được từ tập tin. Giá trị này bằng `n` hay nhỏ hơn `n` nếu đã chạm đến cuối tập tin hoặc có lỗi xuất hiện.

Ví dụ 1: Viết chương trình ghi lên tập tin `CacSo.Dat` 3 giá trị số (thực, nguyên, nguyên dài). Sau đó đọc các số từ tập tin vừa ghi và hiển thị lên màn hình.

```
#include<stdio.h>
#include<conio.h>
int main() {
    FILE *f;
    f=fopen ("D:\\CacSo.txt", "wb");
    if (f != NULL) {
        double d=3.14;
        int i=101;
        long l=54321;
        fwrite (&d, sizeof(double), 1, f);
        fwrite(&i, sizeof(int), 1, f);
        fwrite(&l, sizeof(long), 1, f);
        /* Doc tu tap tin*/
        rewind(f);
        fread (&d, sizeof(double), 1, f);
        fread(&i, sizeof(int), 1, f);
        fread(&l, sizeof(long), 1, f);
        printf("Cac ket qua la: %f %d %ld", d, i, l);
        fclose(f);
    }
    return 0;
}
```

Ví dụ 2: Mỗi sinh viên cần quản lý ít nhất 2 thông tin: mã sinh viên và họ tên. Viết chương trình cho phép lựa chọn các chức năng: nhập danh sách sinh viên từ bàn phím rồi ghi lên tập tin SinhVien.dat, đọc dữ liệu từ tập tin SinhVien.dat rồi hiển thị danh sách lên màn hình, tìm kiếm họ tên của một sinh viên nào đó dựa vào mã sinh viên nhập từ bàn phím.

Ta nhận thấy rằng mỗi phần tử của tập tin SinhVien.Dat là một cấu trúc có 2 trường: mã và họ tên. Do đó, ta cần khai báo cấu trúc này và sử dụng các hàm đọc/ghi tập tin nhị phân với kích thước mỗi phần tử của tập tin là chính kích thước cấu trúc đó.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
typedef struct {
    char Ma[10];
    char HoTen[40];
} SinhVien;
//-----
void WriteFile (char *FileName) {
    FILE *f;
    int n,i;
    SinhVien sv;
    f=fopen(FileName,"ab");
    printf("Nhap bao nhieu sinh vien? ");
    scanf("%d",&n);
    fflush(stdin);
    for(i=1;i<=n;i++) {
        printf("Sinh vien thu %i\n",i);
        printf(" - MSSV: ");
        gets(sv.Ma);
        printf(" - Ho ten: ");
        gets(sv.HoTen);
        fwrite(&sv,sizeof(sv),1,f);
        fflush(stdin);
    }
    fclose(f);
    printf("Bam phim bat ky de tiep tuc");
    getch();
}
```

```
//-----  
void ReadFile(char *FileName) {  
    FILE *f;  
    SinhVien sv;  
    f=fopen(FileName,"rb");  
    printf("    MSSV    |    Ho va ten\n");  
    fread (&sv,sizeof(sv),1,f);  
    while (!feof(f)) {  
        printf("    %s    |    %s\n",sv.Ma,sv.HoTen);  
        fread(&sv,sizeof(sv),1,f);  
    }  
    fclose(f);  
    printf("Bam phim bat ky de tiep tuc!!!");  
    getch();  
}  
//-----  
void Search(char *FileName) {  
    char MSSV[10] ;  
    FILE *f ;  
    int Found=0;  
    SinhVien sv;  
    fflush(stdin);  
    printf("Ma so sinh vien can tim: ");  
    gets(MSSV);  
    f=fopen(FileName,"rb");  
    while (!feof(f) && Found==0) {  
        fread(&sv,sizeof(sv),1,f);  
        if (strcmp(sv.Ma,MSSV)==0)  
            Found=1;  
    }
```

```
    }
    fclose(f);
    if (Found == 1)
        printf("Tim thay SV co ma %s. Ho ten la: %s",sv.Ma,sv.HoTen);
    else
        printf("Tim khong thay sinh vien co ma %s",MSSV);
    printf("\nBam phim bat ky de tiep tuc!!!");
    getch();
}
//-----
int main() {
    int c;
    for (;;) {
        printf("1. Nhap DSSV\n");
        printf("2. In DSSV\n");
        printf("3. Tim kiem\n");
        printf("4. Thoat\n");
        printf("Ban chon 1, 2, 3, 4: ");
        scanf("%d",&c);
        if(c==1)
            WriteFile("d:\\SinhVien.Dat");
        else if (c==2)
            ReadFile("d:\\SinhVien.Dat");
        else if (c==3) Search("d:\\SinhVien.Dat");
        else break;
    }
    return 0 ;
}
```

## CÂU HỎI ÔN TẬP

**Câu 1:** Viết chương trình quản lý một tập tin văn bản theo các yêu cầu:

- Nhập từ bàn phím nội dung một văn bản sau đó ghi vào đĩa.
- Đọc từ đĩa nội dung văn bản vừa nhập và in lên màn hình.
- Đọc từ đĩa nội dung văn bản vừa nhập, in nội dung đó lên màn hình và cho phép nối thêm thông tin vào cuối tập tin đó.

**Câu 2:** Cho file TXT có cấu trúc như sau:

Dòng đầu lưu giá trị của 1 số nguyên dương  $n$ . Dòng còn lại lưu giá trị của 1 dãy gồm  $n$  số nguyên

- Viết chương trình đọc file trên, lưu vào mảng 1 chiều.
- Xuất dãy đọc được ra màn hình.
- Ghi các số nguyên tố có trong mảng vào file đã cho (ghi tiếp theo, không xoá nội dung cũ).

**Câu 3:** Cho file TXT có cấu trúc như sau:

Dòng đầu lưu giá trị của 2 số nguyên dương  $d, c$ . Các dòng còn lại lưu giá trị của 1 ma trận  $d$  dòng và  $c$  cột là các số nguyên.

- Viết chương trình đọc file ma trận trên.
- Xuất ma trận đó ra màn hình
- Nếu ma trận vuông thì hãy ghi các phần tử trên đường chéo chính của ma trận vào file đã cho (ghi tiếp theo, không xoá nội dung cũ).

# BÀI 5: TÌM KIẾM

*Sau khi học xong bài này, học viên có thể:*

- *Nắm được các phương pháp tìm kiếm.*
- *Cài đặt được các giải thuật tìm kiếm.*
- *Vận dụng giải thuật vào bài toán tìm kiếm trên dữ liệu thực tế.*

## 5.1 GIỚI THIỆU VỀ BÀI TOÁN TÌM KIẾM

---

Tìm kiếm là thao tác quan trọng & thường xuyên trong tin học. Ví dụ: tìm kiếm một nhân viên trong danh sách nhân viên, tìm một sinh viên trong danh sách sinh viên của một lớp ...

Tìm kiếm là quá trình xác định một đối tượng nào đó trong một tập các đối tượng. Kết quả trả về là đối tượng tìm được (nếu có) hoặc chỉ số (nếu có) xác định vị trí của đối tượng trong tập đó.

Việc tìm kiếm dựa theo một trường nào đó của đối tượng, trường này là khóa (key) của việc tìm kiếm.

Ví dụ: Tìm sinh viên có họ tên  $x$  trong danh sách sinh viên, với các thông tin {Mã SV, họ tên, địa chỉ, ...}. Khi đó khóa tìm kiếm là họ tên. Kết quả trả về là: thông tin của sinh viên đó hoặc số thứ tự của sinh viên đó trong danh sách.

Mô tả bài toán:

Tập dữ liệu được lưu trữ là dãy  $a_0, a_1, \dots, a_{n-1}$ . Giả sử chọn cấu trúc dữ liệu mảng để lưu trữ dãy số này trong bộ nhớ chính, có khai báo: `int a[n];`

Khóa cần tìm là  $x$ , có kiểu nguyên: `int x;`

Phương pháp:

Có hai phương pháp tìm kiếm: tìm kiếm tuyến tính (tuần tự) và tìm kiếm nhị phân.



- Tìm kiếm tuyến tính áp dụng trên tập dữ liệu bất kỳ.
- Tìm kiếm nhị phân áp dụng trên tập dữ liệu đã sắp xếp.

## 5.2 TÌM KIẾM TUYẾN TÍNH

Ý tưởng:

Duyệt tuần tự từ phần tử đầu tiên, lần lượt so sánh khóa tìm kiếm với khoá tương ứng của các phần tử trong danh sách. Cho đến khi gặp phần tử cần tìm hoặc đến khi duyệt hết danh sách.

Các bước tiến hành như sau:

```
B1: i = 0;
B2: So sánh a[i] với x:
    Nếu a[i] = x: Tìm thấy. Dừng
    Nếu a[i] ≠ x: Sang bước 3
B3: i = i + 1 // xét phần tử kế tiếp trong mảng
    Nếu i ≥ n: Hết mảng, không tìm thấy. Dừng
    Nếu i < n: Quay lại bước 2.
```

Cài đặt:

```
/* Trả về: vị trí xuất hiện đầu tiên của x trong mảng a
   Trả về: -1 nếu x không có trong mảng a */
int Search(int a[], int n, int key)
{
    int i = 0;
    while (i < n && key != a[i])
        i++;
    if (i < n)
        return i; // tìm thấy tại vị trí i
    return -1; // tìm không thấy
}
```

Nhận xét:

Giải thuật tìm kiếm tuyến tính không phụ thuộc vào thứ tự của các phần tử trong mảng, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy bất kỳ.

Một thuật toán có thể được cài đặt theo nhiều cách khác nhau, kỹ thuật cài đặt ảnh hưởng nhiều đến tốc độ thực hiện. Ví dụ như thuật toán Search cải tiến sẽ chạy nhanh hơn thuật toán trước do vòng lặp while chỉ so sánh một điều kiện...

## 5.3 TÌM KIẾM NHỊ PHÂN

Phép tìm kiếm nhị phân được áp dụng trên dãy khóa đã có thứ tự:  $k[1] \leq k[2] \leq \dots \leq k[n]$ .

Ý tưởng:

Giả sử ta cần tìm trong đoạn  $a[\text{left}, \dots, \text{right}]$  với khóa tìm kiếm là  $x$ ;

Chia đôi phạm vi tìm kiếm  $\text{mid} = (\text{left} + \text{right}) / 2$ .

Xét phần tử giữa là  $a[\text{mid}]$ :

Nếu  $a[\text{mid}] = x$ : Tìm thấy tại vị trí  $\text{mid}$ .

Nếu  $a[\text{mid}] < x$ : Đoạn  $a[\text{left}, \dots, \text{mid}]$  chứa các phần tử  $< x$

Tìm  $x$  trong đoạn  $a[\text{mid} + 1, \dots, \text{right}]$

Nếu  $a[\text{mid}] > x$ : Đoạn  $a[\text{mid}, \dots, \text{right}]$  chứa các phần tử  $> x$

Tìm  $x$  trong đoạn  $a[\text{left}, \dots, \text{mid} - 1]$

Quá trình tìm kiếm thất bại nếu  $\text{left} > \text{right}$

Các bước tiến hành:

B1:  $\text{left} = 0, \text{right} = n - 1$  // tìm kiếm trên tất cả phần tử

B2:  $\text{mid} = (\text{left} + \text{right}) / 2$  // lấy mốc so sánh

So sánh  $a[\text{mid}]$  với  $X$

$a[\text{mid}] = x$ : Tìm thấy. Dừng

$a[\text{mid}] < x$ : // tìm tiếp trong dãy  $a[\text{mid} + 1] \dots a[\text{right}]$

$\text{left} = \text{mid} + 1;$

$a[\text{mid}] > x$ : // tìm tiếp trong dãy  $a[\text{left}] \dots a[\text{mid} - 1]$

$\text{right} = \text{mid} - 1;$

B3:

Nếu  $\text{left} \leq \text{right}$ : //còn phần tử  $\rightarrow$  tìm tiếp

Lặp bước 2

Ngược lại: Dừng //đã xét hết các phần tử

Cài đặt:

```
int BinarySearch(int a[], int n, int x){
    int left = 0, right = n-1, mid;
    while (left <= right){
        mid = (left + right)/ 2; // lấy điểm giữa
        if (a[mid] == x)         // nếu tìm được
            return mid;
        if (a[mid] < x)          // tìm đoạn bên phải mid
            left = mid+1;
        else
            right = mid-1;       // tìm đoạn bên trái mid
    }
    return -1;                  // không tìm được
}
```

Nhận xét:

Thuật giải nhị phân dựa vào quan hệ giá trị của các phần tử trong mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được với dãy đã có thứ tự.

Thuật giải nhị phân tìm kiếm nhanh hơn tìm kiếm tuyến tính.

Tuy nhiên khi áp dụng thuật giải nhị phân thì cần phải quan tâm đến chi phí cho việc sắp xếp mảng. Vì khi mảng được sắp thứ tự rồi thì mới tìm kiếm nhị phân.

## CÂU HỎI ÔN TẬP

**Câu 1:** Sinh mảng ngẫu nhiên gồm  $n$  số nguyên có giá trị  $\in (-100, 100)$

Tìm phần tử có giá trị  $x$  trong mảng bằng 2 phương pháp:

- a. Tìm tuần tự
- b. Tìm nhị phân

**Câu 2:** Cho cấu trúc Sách (Mã sách: `char[10]`, Tên sách: `char[40]`, Giá: `long`). Viết chương trình thực hiện:

- a. Nhập, xuất danh sách gồm  $n$  cuốn sách.
- b. Tìm cuốn sách có mã là  $x$  bằng phương pháp tìm kiếm tuần tự ( $x$  nhập từ bàn phím).
- c. Tìm cuốn sách có tên là  $x$  bằng phương pháp tìm kiếm nhị phân. ( $x$  nhập từ bàn phím).
- d. Liệt kê thông tin các cuốn sách có giá  $> G$  ( $G$  nhập từ bàn phím).

Tìm cuốn sách có giá lớn nhất.

# BÀI 6: SẮP XẾP

Sau khi học xong bài này, học viên có thể:

- *Nắm được các phương pháp sắp xếp*
- *Biết vận dụng phương pháp sắp xếp nào cho từng bài toán cụ thể*
- *Cài đặt được các giải thuật sắp xếp trên dữ liệu thực tế.*

## 6.1 GIỚI THIỆU VỀ BÀI TOÁN SẮP XẾP

### 6.1.1 Giới thiệu

Sắp xếp các phần tử của một cấu trúc theo thứ tự tăng dần (hay giảm dần) là một công việc được thực hiện thường xuyên. Với một cấu trúc đã được sắp xếp chúng ta rất thuận tiện khi thực hiện các tác vụ trên cấu trúc như tìm kiếm, trích lọc, duyệt cấu trúc...

Để phân tích đánh giá giải thuật sắp xếp, chúng ta cần thẩm định giải thuật chiếm dụng bao nhiêu vùng nhớ, giải thuật chạy nhanh hay chạy chậm. Hai tiêu chí chính dùng để phân tích một giải thuật sắp xếp là:

- Sự chiếm dụng bộ nhớ của giải thuật.
- Thời gian thực hiện của giải thuật.

### 6.1.2 Mô tả

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng theo một thứ tự nhất định. Ví dụ:  $\{1, 2, 5, 7, 9, 12\}$ ,  $\{14, 12, 7, 5, 2, 1\}$

Dữ liệu thường được tổ chức thành mảng các mẫu tin dữ liệu. Mỗi mẫu tin thường có một số các trường dữ liệu khác nhau. Trường tham gia quá trình tìm kiếm gọi là khoá (key), việc sắp xếp sẽ được tiến hành dựa vào giá trị khoá này.

VD: Sinh viên (MaSV, Hoten, Ngaysinh). Sắp xếp sinh viên theo họ tên, khi đó khóa sắp xếp là Hoten.

Mô tả bài toán:

Cho tập gồm  $n$  phần tử có  $m$  thuộc tính, được biểu diễn dưới dạng mẫu tin. Dựa vào một (hoặc vài) thuộc tính để sắp xếp các phần tử theo trật tự mới.

Cách sắp xếp: Dựa theo khóa sắp xếp định vị lại thứ tự mẫu tin, chuyển các mẫu tin về vị trí mới. Sử dụng hai thao tác cơ bản: So sánh và hoán vị (gán).

## 6.2 ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

### 6.2.1 Khái niệm về độ phức tạp của giải thuật

Thời gian máy tính thực hiện một giải thuật không chỉ phụ thuộc vào bản thân giải thuật đó, mà còn tùy thuộc từng máy tính. Để đánh giá hiệu quả của một giải thuật, có thể xét số các phép tính phải thực hiện khi thực hiện giải thuật đó. Thông thường số các phép tính được thực hiện phụ thuộc vào cỡ của bài toán, tức là độ lớn của đầu vào. Vì thế độ phức tạp giải thuật là một hàm phụ thuộc đầu vào. Tuy nhiên trong những ứng dụng thực tiễn, chúng ta không cần biết chính xác hàm này mà chỉ cần biết một ước lượng đủ tốt của chúng.

Thời gian thực hiện chương trình:

Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu  $T(n)$  trong đó  $n$  là kích thước (độ lớn) của dữ liệu vào. Chương trình tính tổng của  $n$  số có thời gian thực hiện là  $T(n) = C*n$  trong đó  $C$  là một hằng số. Thời gian thực hiện chương trình là một hàm không âm, tức là  $T(n) \geq 0$  với mọi  $n \geq 0$ .

Đơn vị đo thời gian thực hiện:

Đơn vị của  $T(n)$  không phải là đơn vị đo thời gian bình thường như giờ, phút giây ... mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng. Khi ta nói thời gian thực hiện của một chương trình là  $T(n) = C*n$  thì có nghĩa là chương trình ấy cần  $C*n$  chỉ thị thực thi.

Thời gian thực hiện trong trường hợp xấu nhất:

Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi ta cho vào dãy chưa có thứ tự, hoặc khi ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi ta cho vào một dãy đã có thứ tự giảm. Vì vậy thường ta coi  $T(n)$  là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước  $n$ , tức là:  $T(n)$  là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước  $n$ .

Độ phức tạp của giải thuật:

Cho một hàm  $T(n)$ ,  $T(n)$  gọi là độ phức tạp  $f(n)$  nếu tồn tại các hằng số  $C$  và  $N_0$  sao cho  $T(n) \leq C \cdot f(n)$  với mọi  $n \geq N_0$  và ký hiệu là  $O(f(n))$ .

Nói cách khác độ phức tạp tính toán của giải thuật là một hàm chặn trên của hàm thời gian. Vì hằng nhân tử  $C$  trong hàm chặn trên không có ý nghĩa nên ta có thể bỏ qua vì vậy hàm thể hiện độ phức tạp có các dạng thường gặp sau:

- $O(1)$  : Nếu  $T(n)$  là hằng số ( $T(n)=C$ )
- $O(\log_2 n)$  : Độ phức tạp dạng logarit
- $O(n)$  : Độ phức tạp tuyến tính
- $O(n \log_2 n)$ : Độ phức tạp tuyến tính logarit
- $O(n^2)$ ,  $O(n^3)$ , ...,  $O(n^\alpha)$ : Độ phức tạp đa thức
- $O(n!)$ ,  $O(n^n)$ : Độ phức tạp dạng hàm mũ.

Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

### 6.2.2 Cách tính độ phức tạp của giải thuật

Quy tắc tổng: Độ phức tạp của đoạn gồm hai chương trình nối tiếp nhau là:

$$O(\max(f(n), g(n)))$$

Quy tắc nhân: Độ phức tạp của đoạn gồm hai chương trình lồng nhau là:

$$O(f(n).g(n))$$

Độ phức tạp thường được tính toán dựa vào số phép so sánh và phép gán.

Một số công thức thường dùng:

$$\sum_{i=1}^n 1 = n$$

$$\sum_{i=a}^b 1 = \sum_{i=1}^b 1 - \sum_{i=1}^a 1 = b - a + 1$$

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=a}^n i = \frac{n(n+1) - (a-1)a}{2}$$

$$\sum_{i=1}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

## 6.3 CÁC GIẢI THUẬT SẮP XẾP

Có rất nhiều giải thuật để hiện thực việc sắp xếp dữ liệu. Ở phần này ta xét các phương pháp: Bubble Sort, Selection Sort, Insertion Sort, Interchange Sort, Shell Sort, Quick Sort, Merge Sort và Radix Sort. Minh họa sắp xếp tăng dần.

### 6.3.1 Giải thuật Bubble Sort

Mô tả phương pháp:

Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn về đầu.

Sau đó ở bước tiếp theo không xét phần tử đó nữa. Do vậy lần xử lý thứ  $i$  sẽ có vị trí đầu dãy là  $i$ .

Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào được xét.

Các bước tiến hành:

B1:  $i=0$ ; // lần xử lý đầu tiên

B2:  $j=n-1$ ; //duyệt từ cuối dãy ngược về vị trí  $i$

Trong khi ( $j>i$ ) thực hiện:

Nếu  $a[j] < a[j-1]$ : Hoán đổi  $a[j]$  và  $a[j-1]$

$j = j - 1$ ;

B3:  $i = i+1$ ; // lần xử lý kế tiếp



Nếu  $i > n-2$ : Hết dãy  $\rightarrow$  Dừng

Ngược lại: quay lại B2

Ví dụ minh họa:

Dùng phương pháp Bubble Sort để sắp xếp lại danh sách A[] dưới đây.

25    55    45    40    10    90    85    35

Bảng sau đây minh họa quá trình so sánh và đổi chỗ cho lần duyệt đầu tiên (cặp phần tử in đậm là cặp đã được hoán đổi vị trí):

Ban đầu j	25	55	45	40	10	90	85	35
7	25	55	45	40	10	90	<b>35</b>	<b>85</b>
6	25	55	45	40	10	<b>35</b>	<b>90</b>	85
5	25	55	45	40	10	35	90	85
4	25	55	45	<b>10</b>	<b>40</b>	35	90	85
3	25	55	<b>10</b>	<b>45</b>	40	35	90	85
2	25	<b>10</b>	<b>55</b>	45	40	35	90	85
1	<b>10</b>	<b>25</b>	55	45	40	35	90	85

Sau lần duyệt đầu tiên, phần tử nhỏ nhất (10) được đưa về đầu dãy. Từ lần duyệt thứ hai, không xét nó nữa.

Nhận xét:

Nếu dùng phương pháp Bubble Sort để sắp xếp danh sách có n nút:

- Sau lần duyệt thứ 1, nút nhỏ nhất được định vị đúng chỗ.
- Sau lần duyệt thứ 2, nút thứ 2 được định vị đúng chỗ.
- Sau lần duyệt thứ n-1 thì n nút trong danh sách sẽ được sắp xếp thứ tự.

Sự biến đổi của danh sách qua các lần duyệt được mô tả trong bảng dưới đây.

Lần Duyệt	Dữ liệu							
Ban đầu	25	55	45	40	10	90	85	35
0	<b>10</b>	25	55	45	40	35	90	85
1	<b>10</b>	<b>25</b>	35	55	45	40	85	90
2	<b>10</b>	<b>25</b>	<b>35</b>	40	55	45	85	90
3	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	45	55	85	90
4	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	55	85	90
5	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>55</b>	85	90
6	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>55</b>	<b>85</b>	90
Kết quả	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>55</b>	<b>85</b>	<b>90</b>

Cài đặt giải thuật Bubble Sort:

```
void BubbleSort(int a[], int n)
{
    int i, j;
    for(i=0; i < n-1; i++)
        for(j=n-1; j > i; j--) //đổi chỗ cặp phần tử đứng sai
            if (a[j-1] > a[j]) //phần tử đứng trước > phần tử đứng sau
                Swap(a[j], a[j-1]); //đổi chỗ a[j] và a[j-1]
}
```

Trong đó, Swap() là hàm hoán vị giá trị của hai phần tử.

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

### 6.3.2 Giải thuật Interchange Sort

Mô tả phương pháp:

Xuất phát từ đầu dãy, lần lượt tìm những phần tử còn lại không thoả thứ tự với phần tử đang xét. Với mỗi phần tử tìm được mà không thoả thứ tự, thực hiện hoán vị để thoả thứ tự. Lặp lại tương tự với các phần tử tiếp theo.

Các bước tiến hành:

```
B1: i = 0; // bắt đầu từ đầu dãy
B2: j = i + 1; // duyệt qua các phần tử sau
B3: Chứng nào j < n thực hiện:
    Nếu a[j] < a[i] thì Đổi chỗ a[i] và a[j];
    j = j + 1;
B4: i = i + 1;
    Nếu i < n-1 thì lặp lại B2;
    Ngoài ra → Kết thúc!
```

Ví dụ minh họa:

Dùng phương pháp Interchange Sort sắp xếp danh sách sau:

25 55    45    40    10    90    85    35

Bảng sau đây minh họa quá trình so sánh và đổi chỗ cho lần duyệt đầu tiên,  $i=0$ :

<i>Ban đầu</i> <i>j</i>	<b>25</b>	<b>55</b>	<b>45</b>	<b>40</b>	<b>10</b>	<b>90</b>	<b>85</b>	<b>35</b>
<b>1</b>	25	55	45	40	10	90	85	35
<b>2</b>	25	55	45	40	10	90	85	35
<b>3</b>	25	55	45	40	10	90	85	35
<b>4</b>	<b>10</b>	55	45	40	<b>25</b>	90	85	35
<b>5</b>	<b>10</b>	55	45	40	25	<b>90</b>	85	35
<b>6</b>	<b>10</b>	55	45	40	25	90	<b>85</b>	35
<b>7</b>	<b>10</b>	55	45	40	25	90	85	<b>35</b>

Sự biến đổi của danh sách qua các lần duyệt được mô tả trong bảng dưới đây.

<i>Lần chạy i</i>	<i>Dữ liệu</i>							
Ban đầu	<b>25</b>	<b>55</b>	<b>45</b>	<b>40</b>	<b>10</b>	<b>90</b>	<b>85</b>	<b>35</b>
<b>0</b>	<b>10</b>	55	45	40	25	90	85	35
<b>1</b>	<b>10</b>	<b>25</b>	55	45	40	90	85	35
<b>2</b>	<b>10</b>	<b>25</b>	<b>35</b>	55	45	90	85	40
<b>3</b>	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	55	90	85	45
<b>4</b>	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	90	85	55
<b>5</b>	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>55</b>	85	90
<b>6</b>	<b>10</b>	<b>25</b>	<b>35</b>	<b>40</b>	<b>45</b>	<b>55</b>	<b>85</b>	90

Cài đặt giải thuật Insertion Sort:

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

### 6.3.3 Giải thuật Selection Sort

Mô tả phương pháp:

Phương pháp này lần lượt chọn nút nhỏ nhất cho các vị trí 0, 1, 2, ..., n-1. Cụ thể:

Lần chọn thứ 0:

- Dò tìm trong khoảng vị trí từ 0 đến n-1 để xác định nút nhỏ nhất tại vị trí min0.
- Đổi chỗ hai nút tại vị trí min0 và vị trí 0.

Lần chọn thứ 1:

- Dò tìm trong khoảng vị trí từ 1 đến  $n-1$  để xác định nút nhỏ nhất tại vị trí  $\text{min1}$ .
- Đổi chỗ hai nút tại vị trí  $\text{min1}$  và vị trí 1.

Lần chọn thứ  $i$ :

- Dò tìm trong khoảng vị trí từ  $i$  đến  $n-i$  để xác định nút nhỏ nhất tại vị trí  $\text{mini}$ .
- Đổi chỗ hai nút tại vị trí  $\text{mini}$  và vị trí  $i$ .

...

Lần chọn thứ  $n-2$  (lần chọn cuối cùng):

- Dò tìm trong khoảng từ vị trí  $n-2$  đến  $n-1$  để xác định nút nhỏ nhất tại vị trí  $\text{minn}-2$ .
- Đổi chỗ hai nút tại vị trí  $\text{minn}-2$  và vị trí  $n-2$ .

Các bước thực hiện:

B1:  $i = 0$ ,  $\text{min}$  lưu vị trí phần tử nhỏ nhất.

B2: Tìm phần tử nhỏ nhất  $a[\text{min}]$  trong dãy hiện hành từ  $a[i]$  đến  $a[n-1]$

B3: Hoán vị  $a[i]$  và  $a[\text{min}]$

B4: Nếu  $i < n-2$  thì  $i = i+1 \rightarrow$  Lặp B2

Ngược lại  $\rightarrow$  Dừng

Ví dụ minh họa:

Dùng giải thuật Selection Sort để sắp xếp cho danh sách sau:

25    55    45    40    10    90    85    35

Lần chọn	Dữ liệu							
Ban đầu	25	55	45	40	10	90	85	35
0	10	55	45	40	25	90	85	35
1	10	25	45	40	55	90	85	35
2	10	25	35	40	55	90	85	45
3	10	25	35	40	55	90	85	45
4	10	25	35	40	45	90	85	55
5	10	25	35	40	45	55	85	90
6	10	25	35	40	45	55	85	90

Cài đặt giải thuật Selection Sort:

```
void SelectionSort(int a[], int n){
    for(int i = 0; i < n-1; i++)    {
        int min = a[i], vmin = i;
        for(int j = i+1; j < n; j++)
            if (a[j] < min)
            {
                min = a[j];
                vmin = j;
            }
        Swap(a[vmin], a[i]);    //hoán vị a[i] và phần tử nhỏ nhất của mảng
    }
}
```

\*Cải tiến:

```
void SelectionSort(int a[], int n)
{
    for(int i = 0; i < n-1; i++){    // duyệt qua n-1 phần tử
        int vmin = i;
        for(int j = i+1; j < n; j++)
            if (a[j] < a[vmin])    vmin = j;
        Swap(a[vmin], a[i]);    //hoán vị a[i] và phần tử nhỏ nhất
    }
}
```

Nhận xét: Trong trường hợp vmin = i thì thao tác hoán vị a[vmin] và a[i] có cần thiết không? Hãy cải tiến giải thuật.

### 6.3.4 Giải thuật Insertion Sort

Mô tả phương pháp:

Phương pháp này sẽ lần lượt chèn các nút vào danh sách đã có thứ tự:

- Xem danh sách đầu tiên đã có thứ tự chỉ là 1 nút a[0].

- Lần chèn 1: chèn  $a[1]$  vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng hai nút là  $a[0]$  và  $a[1]$ .
- Lần chèn 2: chèn  $a[2]$  vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng 3 nút là  $a[0]$ ,  $a[1]$  và  $a[2]$ .
- ...
- Lần chèn  $n-1$ : chèn  $a[n-1]$  vào đúng vị trí chúng ta được danh sách cuối cùng đã có thứ tự có  $n$  nút là  $a[0]$ ,  $a[1]$ , ...,  $a[n-1]$ .

Các bước tiến hành:

B1:  $i = 1$ ; //giả sử có đoạn  $a[0]$  đã được sắp

B2:  $x = a[i]$ ;

Tìm được vị trí cần chèn  $x$  vào là  $pos$

B3: Dời chỗ các phần tử từ  $a[pos] \rightarrow a[i-1]$  sang phải một vị trí để dành chỗ cho  $a[i]$ .

B4:  $a[pos] = x$ ; // có đoạn  $a[0]...a[i]$  được sắp.

B5:  $i = i + 1$ ;

Nếu  $i < n$ : Lặp lại B2

Ngược lại: Dừng  $\rightarrow$  Dãy đã được sắp

Ví dụ minh họa:

Dùng Insertion Sort sắp xếp danh sách:

25    55    45    40    10    90    85    35

Lần chèn	Danh sách trước lần chèn	Nút chèn vào danh sách $a[i]$	Danh sách sau khi chèn
<b>0</b>	25	55	25, <b>55</b>
<b>1</b>	25, 55	45	25, <b>45</b> , 55
<b>2</b>	25, 45, 55	40	25, <b>40</b> , 45, 55
<b>3</b>	25, 40, 45, 55	10	<b>10</b> , 25, 40, 45, 55
<b>4</b>	10, 25, 40, 45, 55	90	10, 25, 40, 45, 55, <b>90</b>
<b>5</b>	10, 25, 40, 45, 55, 90	85	10, 25, 40, 45, 55, <b>85</b> , 90
<b>6</b>	10, 25, 40, 45, 55, 85, 90	35	10, 25, <b>35</b> , 40, 45, 55, 85, 90

Cài đặt giải thuật Insertion Sort:

```
void InsertionSort(int a[], int n){
    int pos, x;           //x lưu phần tử a[i]
    for(int i=1; i < n; i++){
        x = a[i]; pos = i-1;    //xét từ vị trí i trở về trước
        while (pos ≥ 0 && a[pos] > x) {
            //kết hợp dời chỗ các phần tử đứng sau x trong dãy mới
            a[pos+1] = a[pos];
            pos--;
        }
        a[pos+1] = x; //chèn x vào dãy mới
    }
}
```

### 6.3.5 Giải thuật Shell Sort

Mô tả giải thuật:

Thuật toán Insertion Sort có hạn chế là luôn chèn 1 phần tử vào đầu dãy. Shell Sort cải tiến bằng cách chia làm nhiều dãy con và thực hiện phương pháp chèn trên từng dãy con.

Xét một dãy  $a[0]...a[n-1]$ , cho một số nguyên  $h$  ( $1 \leq h \leq n$ ), chia dãy thành  $h$  dãy con như sau:

Dãy con 1:  $a[0], a[0+h], a[0+2h]...$

Dãy con 2:  $a[1], a[1+h], a[1+2h]...$

Dãy con 3:  $a[2], a[2+h], a[2+2h]...$

...

Dãy con  $h$ :  $a[h], a[2h], a[3h]...$

Với mỗi bước  $h$ , áp dụng Insertion Sort trên từng dãy con độc lập để làm mịn dần các phần tử trong dãy chính. Tiếp tục làm tương tự đối với bước  $h \div 2...$  cho đến  $h = 1$ . Khi  $h = 1$  thực hiện Insertion Sort trên 1 dãy duy nhất là dãy chính. Kết quả được dãy phần tử được sắp.

Các bước thực hiện:

B1: chọn k khoảng cách  $h[1], h[2], \dots, h[k]$ , và  $i = 0$ ;

B2: Chia dãy ban đầu thành các dãy con có bước nhảy là  $h[i]$ . Thực hiện sắp xếp từng dãy con bằng Insertion sort.

B3:  $i = i + 1$

Nếu  $i > k$ : Dừng

Ngược lại: Đến bước 2.

Ví dụ minh họa:

Dùng giải thuật Shell Sort sắp xếp danh sách sau,  $n = 8, h = \{5, 3, 1\}$ .

10    3    7    6    2    5    4    16

Với  $h=5$ , chia được các dãy con sau:

<i>Ban đầu</i> <b>Dãy</b>	<b>10</b>	<b>3</b>	<b>7</b>	<b>6</b>	<b>2</b>	<b>5</b>	<b>4</b>	<b>16</b>
<b>1</b>	5					10		
<b>2</b>		3					4	
<b>3</b>			7					16
<b>4</b>				6				
<b>5</b>					2			
<b>Kết quả</b>	5	3	7	6	2	10	4	16

- Tiếp tục chia với  $h=3$ :

<i>Ban đầu</i> <b>Dãy</b>	<b>5</b>	<b>3</b>	<b>7</b>	<b>6</b>	<b>2</b>	<b>10</b>	<b>4</b>	<b>16</b>
<b>1</b>	4			5			6	
<b>2</b>		2			3			16
<b>3</b>			7			10		
<b>Kết quả</b>	4	2	7	5	3	10	6	16

- Với  $h=1$ , thực hiện sắp xếp chèn trên dãy:

4    2    7    5    3    10    6    16

ta được dãy sắp tăng: 2    3    4    5    6    7    10    16.

Cài đặt giải thuật:

```
// h[] chứa các bước nhảy, k là số bước nhảy
void ShellSort(int a[], int n, int h[], int k){
```



```

int step, i, pos, x, len;
for(step = 0; step < k; step++) {    //duyet qua từng bước nhảy
    len = h[step];                    // chiều dài của bước nhảy
    for(i = len; i < n; i++) {        // duyệt các dãy con
        x = a[i];                    // lưu phần tử cuối để tìm vị trí thích hợp trong dãy con
        pos = i - len;               // a[i] đứng trước a[i] trong cùng dãy con
        while ((x < a[pos]) && (pos ≥ 0)) {
            a[pos+len] = a[pos];      // dời về sau theo dãy con
            pos = pos - len;           // qua phần tử trước trong dãy con
        }
        a[pos+len] = x;              // đưa x vào vị trí thích hợp trong dãy con
    } // end for i
} // end for step

```

### 6.3.6 Giải thuật Quick Sort

Thuật toán do Hoare đề xuất. Tốc độ trung bình nhanh hơn thuật toán khác, do đó Hoare dùng “quick” để đặt tên.

Mô tả giải thuật:

Quick Sort là giải thuật rất hiệu quả, rất thông dụng và thời gian chạy của giải thuật trong khoảng  $O(n \log n)$ . Nội dung của giải thuật này như sau:

- Chọn một phần tử bất kỳ trong danh sách (giả sử là phần tử giữa) gọi là phần tử làm mốc, xác định vị trí của phần tử này trong danh sách gọi là vị trí mốc.
- Tiếp theo chúng ta phân hoạch các phần tử còn lại trong danh sách cần sắp xếp sao cho các phần tử từ vị trí 0 đến vị trí mốc -1 đều có nội dung nhỏ hơn hoặc bằng phần tử mốc, các phần tử từ vị trí mốc + 1 đến n-1 đều có nội dung lớn hơn hoặc bằng phần tử mốc.
- Quá trình lại tiếp tục như thế với hai danh sách con từ vị trí 0 đến vị trí mốc-1 và từ vị trí mốc + 1 đến vị trí n-1. Sau cùng chúng ta sẽ được danh sách đã có thứ tự.

Các bước tiến hành:

**B1:** Chọn tùy ý một phần tử  $a[k]$  trong dãy là giá trị mốc,  $\text{left} \leq k \leq \text{right}$ ,

Cho  $x = a[k]$ ,  $i = \text{left}$ ,  $j = \text{right}$ .

**B2:** Tìm và hoán vị cặp phần tử  $a[i]$  và  $a[j]$  đứng sai thứ tự.

**B2-1:** Chứng nào  $a[i] < x$  thì  $i++$ ; //Tìm  $a[i] \geq x$

B2-2: Chứng nào  $a[j] > x$  thì  $j--$ ; //Tìm  $a[j] \leq x$

B2-3: Nếu  $i < j$  thì  $\text{Swap}(a[i], a[j])$  //  $a[i], a[j]$  sai thứ tự

B3:

Nếu  $i < j$ :  $\rightarrow$  Bước 2;

Nếu  $i \geq j$ :  $\rightarrow$  Dừng.

Ví dụ minh họa:

Dùng giải thuật Quick Sort sắp xếp danh sách sau:

25 55 45 40 10 90 85 35

Phần tử làm mốc	Dữ liệu							
	25	55	45	40	10	90	85	35
40	25	35	10	40	45	90	85	55
35	25	10	35	40				
25	10	25						
90					45	55	85	90
Kết quả	10	25	35	40	45	55	85	90

Cài đặt giải thuật:

```
void QuickSort(int a[], int left, int right) {
    int i, j, x;
    x = a[(left+right)/2]; // chọn phần tử giữa làm mốc
    i = left; j = right;
    do {
        while (a[i] < x) i++; // lặp đến khi a[i] >= x
        while (a[j] > x) j--; // lặp đến khi a[j] <= x
        if (i <= j) {
            Swap(a[i], a[j]);
            i++; // qua phần tử kế tiếp
            j--; // qua phần tử đứng trước
        }
    } while (i < j);
    if (left < j) /* phân đoạn bên trái */ QuickSort(a, left, j);
    if (right > i) /* phân đoạn bên phải */ QuickSort(a, i, right);
}
```

### 6.3.7 Giải thuật Merge Sort

Mô tả giải thuật:

Là phương pháp sắp xếp bằng cách trộn hai danh sách đã có thứ tự thành một danh sách đã có thứ tự. Phương pháp Merge Sort tiến hành qua nhiều bước trộn như sau:

Bước 1:

- Xem danh sách cần sắp xếp như  $n$  danh sách con đã có thứ tự, mỗi danh sách con chỉ có 1 phần tử.
- Trộn từng cặp hai danh sách con kế cận chúng ta được  $n/2$  danh sách con đã có thứ tự, mỗi danh sách con có 2 phần tử.

Bước 2:

- Xem danh sách cần sắp xếp như  $n/2$  danh sách con đã có thứ tự, mỗi danh sách con có 2 phần tử.
- Trộn từng cặp hai danh sách con kế cận chúng ta được  $n/4$  danh sách con đã có thứ tự, mỗi danh sách con có 4 phần tử.

...

- Quá trình cứ tiếp tục diễn ra như thế cho đến khi được một danh sách có  $n$  phần tử.
- Nếu danh sách có  $n$  phần tử thì ta phải tiến hành  $\log_2 n$  bước trộn.

Cài đặt giải thuật Merge Sort:

```
void MergeSort(int a[], int n){
    int i, j, k, low1, up1, low2, up2, size;
    int dstam[MAXLIST];
    size = 1;
    while (size < n){
        low1 = 0;
        k = 0;
```

```

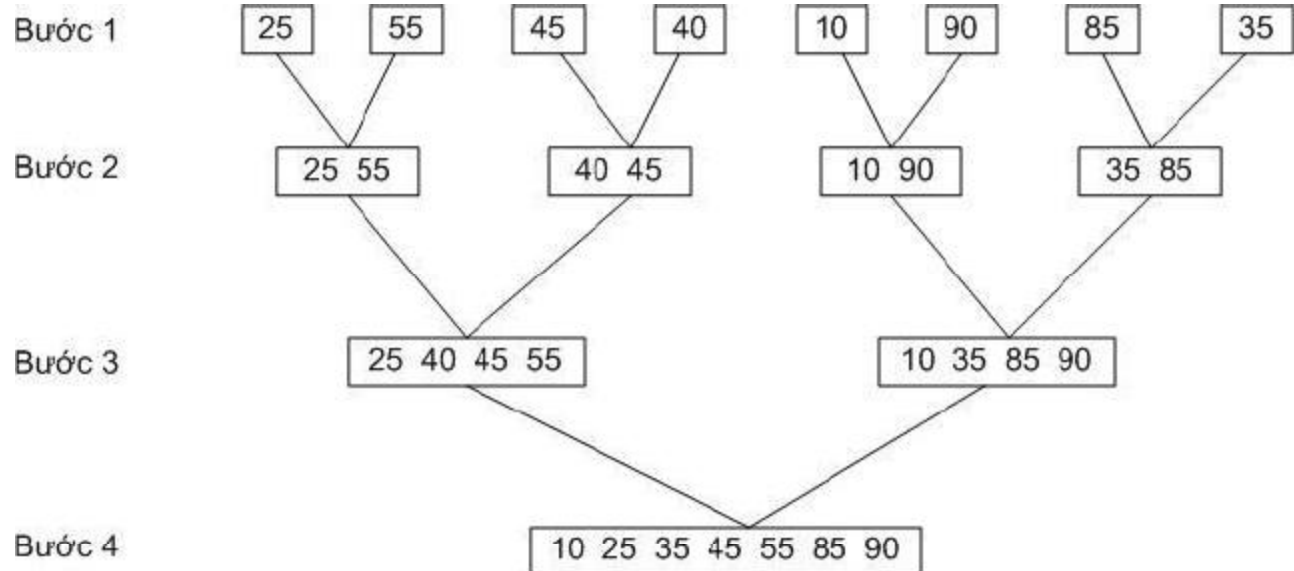
while(low1+size<n){
    low2=low1+size;
    up1=low2-1;
    up2=(low2+size-1<n)? (low2+size-1):(n-1);
    for(i=low1,j=low2;i<=up1 && j<=up2;k++)
        if(a[i]<=a[j])        dstam[k]=a[i++];
        else                  dstam[k]=a[j++];
    for(;i<=up1;k++)        dstam[k]=a[i++];
    for(;j<=up2;k++)        dstam[k]=a[j++];
    low1=up2+1;
}
for(i=low1;k<n;i++)        dstam[k++]=a[i];
for(i=0;i<n;i++)          a[i]=dstam[i];
size *=2;
}
}

```

Ví dụ minh họa:

Dùng phương pháp Merge Sort để sắp xếp danh sách như sau:

25 55 45 40 10 90 85 35



**Hình 6.1:** Minh họa giải thuật Merge Sort.

### 6.3.8 Giải thuật Radix Sort

Không quan tâm đến việc so sánh giá trị các phần tử, bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử. Còn gọi là phương pháp phân lô.

Mô tả phương pháp :

- Xem mỗi phần tử  $a[i]$  trong dãy  $a[0] \dots a[n-1]$  là một số nguyên có tối đa  $m$  chữ số
- Lần lượt phân loại các chữ số theo hàng đơn vị, hàng chục, hàng trăm ...
- Tại mỗi bước phân loại ta nối các dãy con từ danh sách đã phân loại theo thứ tự từ 0 đến 9.
- Sau khi phân loại xong ở hàng thứ  $m$  cao nhất, ta sẽ thu được danh sách các phần tử được sắp.

Các bước tiến hành:

B1:  $k = 0$ ; //  $k$  thể hiện chữ số phân loại,  $k=0$  hàng đơn vị,  $k=1$  hàng chục...

B2: //Tạo các dãy chứa các phần tử phân loại  $B[0] \dots B[9]$

Khởi tạo  $B[0] \dots B[9]$  rỗng,

$B[i]$  sẽ chứa các phần tử có chữ số thứ  $k$  là  $i$ .

B3:

For  $i=1$  to  $n$  do

Đặt  $a[i]$  vào dãy  $B[j]$  với  $j$  là chữ số thứ  $k$  của  $a[i]$ .

Nối  $B[0], B[1], \dots, B[9]$  lại theo đúng trình tự thành dãy  $a$ .

B4:

$k = k + 1$

Nếu  $k < m$ :  $\rightarrow$  Bước 2. //  $m$  là số lượng chữ số tối đa của các số

Ngược lại:  $\rightarrow$  Dừng.

Ví dụ minh họa:

Dùng Radix Sort để sắp xếp: 493 812 715 710 195 437 582 340 385

a. Phân lô theo hàng đơn vị:

Số hàng đơn vị	Dãy con
0	710 340
1	
2	812 582
3	493
4	
5	715 195 385
6	
7	437
8	
9	

Dãy có được khi phân theo hàng đơn vị:

710 340 812 582 493 715 195  
385 437

b. Phân lô dãy có được theo hàng chục:

Số hàng chục	Dãy con
0	
1	710 812 715
2	
3	437
4	340
5	
6	
7	
8	582 385
9	493 195

Dãy có được khi phân theo hàng chục:

710 812 715 437 340 582 385  
493 195

c. Sau đó, phân lô dãy có được theo hàng trăm:

Số hàng trăm	Dãy con
0	
1	195
2	
3	340 385
4	437 493
5	582
6	
7	710 715
8	812
9	

Sau khi phân lô theo hàng trăm, ta có dãy được sắp:

195 340 385 437 493 582 710 715 812

Cài đặt giải thuật:

```
void RadixSort(long a[], int n){
    int i, j, d, digit, num;
    int h = 10;          // biến để lấy các con số, bắt đầu từ hàng đơn vị
    long B[10][MAX]; //mảng hai chiều chứa các phần tử phân lô
    int Len[10];         // kích thước của từng mảng B[i]
    for(d = 0; d < MAXDIGIT; d++) { // xét lần lượt hàng
        for( i = 0; i < 10; i++) // khởi tạo kích thước các dãy B[i] là 0
```

```
        Len[i] = 0;
    for(i = 0; i < n; i++) { // duyệt qua tất cả các phần tử của mảng
        digit = (a[i] % h) / (h / 10); // lấy chữ số theo hàng h
        B[digit][Len[digit]++] = a[i];
    }
    num = 0; // chỉ số bắt đầu cho mảng a[]
    for(i = 0; i < 10; i++) // duyệt qua các dãy từ B[0] – đến B[9]
        for(j = 0; j < Len[i]; j++)
            a[num++] = B[i][j];
    h *= 10; // qua hàng kế tiếp.
}
} //end RadixSort
```

## CÂU HỎI ÔN TẬP

**Câu 1:** Viết chương trình minh họa các phương pháp sắp xếp, chương trình có các chức năng chính như sau:

- Sinh danh sách ngẫu nhiên gồm  $n$  số.
- Chọn phương pháp sắp xếp, sau khi chạy xong, chương trình có báo thời gian chạy.
- Xem danh sách sau khi đã sắp xếp.
- Kết thúc chương trình.

**Câu 2:** Viết chương trình nhập vào danh sách sinh viên, thông tin của sinh viên bao gồm: mã số sinh viên, họ tên, điểm trung bình.

- Sắp xếp danh sách sinh viên tăng dần theo họ tên bằng các phương pháp sắp xếp đã học.
- Sắp xếp danh sách sinh viên giảm dần theo điểm bằng các phương pháp sắp xếp đã học.

## TÀI LIỆU THAM KHẢO

1. Phạm Văn Ất, Kỹ thuật Lập trình C- cơ bản và nâng cao, NXB KH & KT - 2003.
2. Quách Tuấn Ngọc, Tin học căn bản, Nhà xuất bản giáo dục - 1997.
3. Hoàng Kiếm, Nguyễn Đức Thắng, Đinh Nguyễn Anh Dũng, Giáo trình Tin học Đại cương, Nhà xuất bản giáo dục - 1999.
4. Nguyễn Tấn Trần Minh Khang, Bài giảng Kỹ Thuật Lập trình, Khoa Công Nghệ Thông Tin, Đại học Khoa học Tự nhiên
5. Nguyễn Thanh Thủy (chủ biên), Nhập môn lập trình ngôn ngữ C, Nhà xuất bản Khoa học kỹ thuật – 2000.
6. Trần Minh Thái, Bài giảng và bài tập Lập trình căn bản; Khoa Công Nghệ Thông Tin, Đại học Khoa học Tự nhiên
7. Mai Ngọc Thu, Giáo trình C. Khoa Công Nghệ Thông Tin, Đại học Kỹ Thuật Công Nghệ.
8. Brain W. Kernighan & Dennis Ritchie, The C Programming Language, Prentice Hall Publisher, 1988.