

TRƯỜNG ĐẠI HỌC QUỐC TẾ HỒNG BÀNG  
KHOA CÔNG NGHỆ THÔNG TIN  
oOo

# **NGÔN NGỮ LẬP TRÌNH C**

(Phần Nâng Cao)

(TÀI LIỆU THAM KHẢO)

**LÊ VĂN HẠNH**  
**4 - 2018**

# MỤC LỤC

<b>PHẦN MỞ ĐẦU</b>	<b>iii</b>
<b>1 MẢNG MỘT CHIỀU</b>	<b>5</b>
1.1. Khái niệm	5
1.2. Cách khai báo mảng một chiều	5
1.3. Truy cập vào các phần tử của mảng	6
1.4. Nhập dữ liệu cho mảng một chiều	7
1.5. Xuất dữ liệu cho mảng một chiều	8
1.6. Tham số của hàm là mảng	8
1.7. Một số kỹ thuật cơ bản xử lý trên mảng	9
1.8. Phần thực hành	18
1.9. Bài tập (sinh viên tự thực hiện)	20
<b>2 CHUỖI KÝ TỰ (String)</b>	<b>33</b>
2.1. Ký tự (character)	33
2.2. Chuỗi	34
2.3. Các hàm hỗ trợ thao tác trên chuỗi ký tự	36
2.4. Phần thực hành	45
2.5. Bài tập (sinh viên tự thực hiện)	45
<b>3 MẢNG HAI CHIỀU</b>	<b>47</b>
3.1. Khái niệm	47
3.2. Cách khai báo mảng hai chiều	47
3.3. Nhập dữ liệu cho mảng hai chiều	48
3.4. Xuất dữ liệu cho mảng hai chiều	50
3.5. Ma trận vuông	52
3.6. Khai báo kiểu ma trận	54
3.7. Tham số của hàm là mảng hai chiều	54
3.8. Một số kỹ thuật cơ bản trên mảng hai chiều	55
3.9. Phần thực hành	59
3.10. Bài tập (sinh viên tự thực hiện)	60
<b>4 KIỂU DỮ LIỆU CÓ CẤU TRÚC (Struct data types)</b>	<b>72</b>
4.1. Khái niệm	72
4.2. Cách khai báo kiểu cấu trúc	72
4.3. Sử dụng biến cấu trúc	74
4.4. Nhập dữ liệu cho kiểu dữ liệu có cấu trúc	75
4.5. Xuất dữ liệu	75
4.6. Ví dụ tổng hợp	76
4.7. Sử dụng cấu trúc để trừu tượng hóa dữ liệu	77
4.8. Mảng cấu trúc	82
4.9. Tham số của hàm có kiểu cấu trúc	83
4.10. Phần thực hành	85
4.11. Bài tập (sinh viên tự thực hiện)	86
<b>5 KIỂU CON TRỎ (Pointer)</b>	<b>90</b>
5.1. Khái niệm	90
5.2. Khai báo và sử dụng biến con trỏ	91
5.3. Các phép toán trên con trỏ	94
5.4. Sử dụng con trỏ để cấp phát và thu hồi bộ nhớ động	97
5.5. Con trỏ và mảng một chiều	100
5.6. Con trỏ và mảng hai chiều	104

5.7.	Con trỏ với kiểu dữ liệu có cấu trúc (struct).....	108
5.8.	Bài thực hành có hướng dẫn .....	109
5.9.	Bài tập (sinh viên tự thực hiện) .....	114
<b>6</b>	<b>ĐỆ QUY (Recursion) .....</b>	<b>117</b>
6.1.	Khái niệm .....	117
6.2.	Phân loại hàm đệ quy.....	118
6.3.	Kỹ thuật giải bài toán bằng đệ quy .....	120
6.4.	Một số bài toán kinh điển dùng phương pháp đệ quy.....	121
6.5.	Nhận xét.....	122
6.6.	Cấu trúc lặp và đệ quy .....	122
6.7.	Phần thực hành .....	123
6.8.	Bài tập (sinh viên tự thực hiện) .....	124
<b>7</b>	<b>TẬP TIN (FILE) .....</b>	<b>127</b>
7.1.	Khái niệm .....	127
7.2.	Các thao tác trên tập tin .....	128
7.3.	Truy cập tập tin văn bản .....	130
7.4.	Truy cập tập tin nhị phân.....	132
7.5.	Bài tập (sinh viên tự thực hiện) .....	136
	<b>TÀI LIỆU THAM KHẢO .....</b>	<b>140</b>

## PHẦN MỞ ĐẦU

### 1.- Mô tả môn học

Môn học cung cấp cho sinh viên những kiến thức cơ bản và nâng cao về lập trình thông qua ngôn ngữ lập trình C. Môn học này là nền tảng để tiếp thu hầu hết các môn học khác trong chương trình đào tạo. Mặt khác, nắm vững môn này là cơ sở để phát triển tư duy và kỹ năng lập trình để giải các bài toán và các ứng dụng trong thực tế.

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

- Các kiểu dữ liệu trong C
- Các lệnh có cấu trúc
- Một số cấu trúc dữ liệu trong C
- Xử lý các bài toán trên mảng một chiều
- Xử lý các bài toán trên mảng hai chiều
- Kỹ thuật dùng con trỏ
- Biết kỹ thuật viết đệ quy và khử đệ quy
- Biết xây dựng và xử lý các bài toán trên dữ liệu có cấu trúc do người dùng định nghĩa
- Cách lưu trữ và xử lý các file trong C
- Tìm hiểu và cài đặt một số bài toán kinh điển như “Tháp Hà Nội“, “Bài toán mã đi tuần”, “bài toán tám hậu” bằng phương pháp đệ quy hay đệ quy quay lui, Phương pháp sinh dữ liệu

### 2.- NỘI DUNG MÔN HỌC

- **Mảng một chiều:** cung cấp cho sinh viên khái niệm về mảng một chiều, cách nhập, xuất, lưu trữ trên mảng một chiều với dữ liệu kiểu số, xử lý các bài toán như tính tổng các giá trị trên mảng một chiều các số nguyên, số thực, tìm phần tử nhỏ nhất, lớn nhất, thêm, xóa, sắp xếp các phần tử trên mảng dữ liệu kiểu số, ...
- **Chuỗi ký tự:** cung cấp cho sinh viên khái niệm và cách xử lý các bài toán trên dữ liệu kiểu chuỗi.
- **Mảng hai chiều:** cung cấp cho sinh viên khái niệm về mảng hai chiều, cách nhập, xuất, lưu trữ trên mảng hai chiều, xử lý các bài toán như tính tổng các giá trị trên mảng số nguyên, số thực, tìm phần tử nhỏ nhất, lớn nhất, sắp xếp các phần tử trên mảng dữ liệu kiểu số, kiểu chuỗi.
- **Kiểu dữ liệu có cấu trúc:** cung cấp cho sinh viên khái niệm cơ bản về kiểu dữ liệu có cấu trúc do người dùng định nghĩa. Biết Nhập Xuất dữ liệu có cấu trúc cho một phần tử. Biết Nhập, Xuất dữ liệu có cấu trúc và lưu trên mảng một chiều. Cách tìm kiếm và sắp xếp dữ liệu trên mảng một chiều với từng thành phần của dữ liệu. Đi sâu vào các giải thuật trên mảng một chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...

- **Kiểu con trỏ:** cung cấp cho sinh viên khái niệm cơ bản về kiểu dữ liệu 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
- **Đệ quy:** cung cấp cho sinh viên khái niệm cơ bản về kiểu lập trình bằng phương pháp đệ quy, các kiểu đệ quy. Ưu điểm và nhược điểm khi cài đặt hàm bằng phương pháp đệ quy. Giải quyết một số bài toán kinh điển bằng phương pháp đệ quy. Xử lý các giải thuật trên mảng 1 chiều bằng phương pháp đệ quy
- **Tập tin:** cung cấp cho sinh viên một số khái niệm về tập tin. Các bước thao tác với tập tin. Thao tác trên tập tin văn bản. Thao tác trên tập tin nhị phân.

### 3.- KIẾN THỨC TIỀN ĐỀ

Môn học yêu cầu sinh viên phải có nền tảng của môn lập trình cơ bản, có tư duy toán học tốt.

### 4.- YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

### 5.- CÁCH TIẾP NHẬN NỘI DUNG 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 và tóm tắt 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.

### 6.- PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- Điểm thực hành: 30%. Thi thực hành trên máy. Hình thức và nội dung do GV quyết định, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.
- Điểm quá trình: 20%. Do giảng viên lý thuyết quy định dựa trên các tiêu chí chuyên cần, điểm danh, làm bài tập trên lớp, làm bài tập về nhà...
- Điểm thi: 50%. Hình thức bài thi tự luận trong 90 phút.

# 1 MẢNG MỘT CHIỀU

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

- Hiểu được khái niệm về kiểu dữ liệu của mảng cũng như ứng dụng của nó;
- Biết cách khai báo biến kiểu mảng và các phép toán trên các phần tử của mảng;
- Đi sâu vào các giải thuật trên mảng 1 chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...

## 1.1. Khái niệm

- Mảng thực chất là một biến được cấp phát bộ nhớ liên tục và bao gồm nhiều biến thành phần, các thành phần này có cùng một kiểu dữ liệu, gọi là các phần tử. Kiểu dữ liệu của các phần tử có thể là 1 trong các kiểu dữ liệu đã biết như: ký tự, số, chuỗi ký tự..., hoặc những kiểu dữ liệu do người dùng tự định nghĩa.

Mảng là kiểu dữ liệu được sử dụng rất thường xuyên. Chẳng hạn, người ta cần quản lý một danh sách họ và tên của khoảng 100 sinh viên trong một lớp. Nhận thấy rằng mỗi họ và tên để lưu trữ ta cần 1 biến kiểu chuỗi, như vậy 100 họ và tên thì cần khai báo 100 biến kiểu chuỗi. Nếu khai báo như thế này thì đoạn khai báo cũng như các thao tác trên các họ tên sẽ rất dài dòng và rắc rối. Vì thế, kiểu dữ liệu mảng giúp ích ta trong trường hợp này; chỉ cần khai báo 1 biến, biến này có thể coi như là tương đương với 100 biến kiểu chuỗi ký tự; đó là 1 mảng mà các phần tử của nó là chuỗi ký tự. Hay như để lưu trữ các từ khóa của ngôn ngữ lập trình C, ta cũng dùng đến một mảng để lưu trữ chúng.

- Kích thước của mảng là số phần tử của mảng. Kích thước này phải được biết ngay khi khai báo mảng.
- Phân loại mảng: có thể chia mảng làm 2 loại mảng một chiều và mảng nhiều chiều.
- Chỉ số mảng (chỉ mục): là số thứ tự của các phần tử trong mảng (tính từ 0).

0	1	2	3	4	5	6	7

Cách gán chỉ số của các phần tử trong mảng gồm 8 phần tử

## 1.2. Cách khai báo mảng một chiều

### 1.2.1. Khai báo tường minh (số phần tử xác định)

- Cú pháp

**<kiểu cơ sở> <tên mảng> [<số phần tử>]**

- Ý nghĩa

- **<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] ;**
- Thông thường trong lập trình người ta thường định nghĩa trước một hằng chứa số phần tử của mảng. Điều này rất có lợi khi ta cần thay đổi số phần tử của mảng. Khi đó, ta có thể khai báo một mảng như sau:  
**const int MAX=500;**  
**int diem[MAX];**  
Hay **const int MAX=8;**  
**char A[MAX];**

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

- Cú pháp chung  
**<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};**  
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**  
Có thể sử dụng hàm **sizeof()** để biết số phần tử của mảng như sau:  
**Số phần tử = sizeof(tên mảng) / sizeof(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)**

### 1.3. 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]** là phần tử đầu tiên của mảng **a** được khai báo ở trên.
  - **Chỉ số** của phần tử mảng có thể là một hằng, một biến hay 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.
- 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;
//gán giá trị trực tiếp cho biến mảng
a [0] = 4.2;
//gán giá trị cho biến mảng thông qua 1 biến khác
a [2]=t;
/*gán giá trị cho biến mảng thông qua các biến khác trong cùng mảng */
a [i ]=(a [0] +a [2])/2;
a [2*i+1]= a [2*i]+ a [2*i+2];
/*gán giá trị cho biến mảng từ giá trị nhập từ bàn phím */
printf("\nGia tri : %f ", a[1]);
scanf("%f",&a [4]);
//gán giá trị của biến mảng cho biến khác
t= a [4];
```

## 1.4. Nhập dữ liệu cho mảng một chiều

### 1.4.1. Ví dụ 1

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 từng phần tử của mảng

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

### 1.4.2. Ví dụ 2

Khai báo mảng **a** để lưu trữ 100 phần tử là các số thực a: **float a [ 100 ]**. Khi đó máy sẽ cấp phát 400 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[ ]	7.1	3.2	9.0	4.3	5.4	....	8.0	12.0	2.1
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	....	a[n-3]	a[n-2]	a[n-1]

Nhập từng phần tử của mảng

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



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

## 1.5. Xuất dữ liệu cho mảng một chiều

### 1.5.1. Ví dụ 1

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]

Xuất từng phần tử của mảng

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

### 1.5.2. Ví dụ 2

Khai báo mảng a để lưu trữ 100 phần tử là các số thực a: `float a [ 100 ]`. Khi đó máy sẽ cấp phát 400 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[ ]	7.1	3.2	9.0	4.3	5.4	....	8.0	12.0	2.1
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	....	a[n-3]	a[n-2]	a[n-1]

Xuất từng phần tử của mảng

```
for (int i =0 ; i<n ; i++)
{
    printf ("% .1f", a[ i ] );
}
```

## 1.6. Tham số của hàm là mảng

- Khi một mảng được truyền cho một hàm thông qua tham số, địa chỉ bắt đầu của vùng nhớ dành cho mảng sẽ được truyền cho tham số vì vậy hàm được gọi sẽ tác động trực tiếp lên vùng nhớ của mảng truyền cho nó.
- Ví dụ: Xét chương trình sau:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
//Khai bao hang so MAX
const int MAX=100;
//Khai bao prototype
void TaoMangNgauNhienDuong(int A[],int n);
void XuatToanBoMang (int A[], int n);
// Ham chinh cua chuong trinh
```

```

int main()
{
    int n, A[MAX];
    srand((unsigned int) time (NULL));
    printf("Nhap so phan tu cua mang: ");
    scanf("%d",&n);
    TaoMangNgauNhiemDuong(A,n);
    printf("Mang da nhap:");
    XuatToanBoMang (A,n);
    return 0;
}

// Phan chua cac ham
void TaoMangNgauNhiemDuong(int Arr[],int n)
{
    int min=50;
    int max=100;
    for (int i=0; i<n; i++)
        Arr[i]=(rand()%(max-min+1))+min;
}

void XuatToanBoMang (int Arr[], int n)
{
    for (int i = 0; i < n; i ++)
        printf("%5d",Arr[i]);
}

```

## 1.7. Một số kỹ thuật cơ bản xử lý trên mảng

### 1.7.1. Nhập

#### 1.7.1.1. Nhập (gán) dữ liệu cho 1 phần tử

- Dùng lệnh gán, ví dụ: `A[0]='A';`
- Dùng `scanf`. Ví dụ: `scanf("%d",&A[3]);`

#### 1.7.1.2. Khởi tạo giá trị cho mảng tại thời điểm khai báo:

- Khởi tạo bằng cách đặt các giá trị này vào cặp dấu ngoặc nhọn (`{}`), các giá trị cách nhau bởi dấu phẩy (`,`).
- Ví dụ:

```

const int MAX=5;
int A[MAX]={2,95,9,6,3};

```

các giá trị khởi tạo cho mảng cũng có thể đặt trên nhiều dòng

```

const int MAX=10;
int Arr[10]= {2,6,1,7,8,
              5,3,9,10,12};

```

- Nếu các giá trị liệt kê trong `{}` ít hơn số phần tử của mảng thì chỉ có các phần tử đầu được khởi tạo tương ứng với các giá trị trong `{}` các phần tử còn lại sẽ được khởi tạo với giá trị `=0`.
- Gán cùng 1 giá trị cho tất cả các phần tử của mảng (thường dùng khi khởi tạo giá trị ban đầu cho mảng). Ví dụ

```

int A[MAX]={0};

```

### 1.7.1.3. Nhập dữ liệu lần lượt cho nhiều phần tử của mảng bằng lệnh lặp

#### 1.7.1.3.1 Người dùng tự nhập giá trị cho từng phần tử của mảng

Ví dụ:

```
void NhapTungPhantuMang(int A[],int n)
{
    for (int i=0; i<n; i++)
    {   printf("Nhap gia tri cho phan tu thu %d",i);
        scanf("%d",&A[i]);
    }
}
```

#### 1.7.1.3.2 Phát sinh giá trị ngẫu nhiên cho từng phần tử của mảng

Ví dụ:

```
void TaoMangNgauNhiemDuong(int A[],int n)
{
    int min=50;
    int max=100;
    for (int i=0; i<n; i++)
        A[i]=(rand()%(max-min+1))+min;
}

void TaoMangNgauNhiemAmDuong(int A[],int n)
{   int min=50;
    int max=100;
    for (int i=0; i<n; i++)
    {   //B1: tao dau +/-
        int dau,x;
        x=rand();
        if (x%2==1)
            dau=-1;
        else
            dau=1;
        //B2: random so X dau
        A[i]=dau*((rand()%(max-min+1))+min );
    }
}
```

## 1.7.2. Xuất (liệt kê) mảng một chiều

Lưu ý: Hầu hết các lệnh của C khi thực hiện không kiểm tra chỉ số của mảng đang dùng có nằm trong phạm vi hợp lệ hay không (không báo lỗi khi biên dịch). Giả sử ta khai báo một mảng `int A[10]`, nhưng khi ta truy xuất ta lại truy xuất đến những phần tử có chỉ số >9, điều này có thể gây ra lỗi trong quá trình xử lý của chương trình.

#### 1.7.2.1. Xuất (liệt kê) toàn bộ các phần tử của mảng

```
void XuatToanBoMang (int A[], int n)
{
    for (int i = 0; i < n; i ++ )
        printf("%5d",A[i]);
}
```

### 1.7.2.2. Xuất (liệt kê) các phần tử của mảng thỏa điều kiện cho trước

#### 1.7.2.2.1 Điều kiện lựa chọn được gán trực tiếp trong biểu thức điều kiện của phát biểu if

- Điều kiện căn cứ trên giá trị của từng phần tử trong mảng:

- Xuất các số chẵn có trong mảng

```
void XuatSoChanCoTrongMang (int A[], int n)
{
    for (int i = 0; i < n; i++)
        if(A[i]%2==0)
            printf("%5d",A[i]);
}
```

- Xuất các số âm có trong mảng

```
void XuatSoAmCoTrongMang (int A[], int n)
{
    for (int i = 0; i < n; i++)
        if(A[i]<0)
            printf("%5d",A[i]);
}
```

- Điều kiện căn cứ vào vị trí của phần tử trong mảng:

- Xuất các số tại vị trí chẵn trong mảng

```
void LietKeSoTaiViTriChan (int A[], int n)
{
    for (int vitri = 0; vitri < n; vitri++)
        if(vitri %2==0)
            printf("\nVi tri %d chua so %d", vitri,A[i]);
}
```

#### 1.7.2.2.2 Điều kiện lựa chọn là kết quả trả về của một hàm trong biểu thức điều kiện của phát biểu if

```
void LietKeSoTaiViTriChan (int A[], int n)
{
    for (int i=0; i<n; i++)
        if(LaSNT(A[i]))
            printf("\nVi tri %d chua so %d", i, A[i]);
}

bool LaSNT(int k)
{
    int dem=0;
    for(int i=1;i<=k;i++)
        if (k%i==0)
            dem++;
    if (dem==2)
        return true;
    else
        return false;
}
```

**1.7.3. Tính tổng – Tính trung bình có điều kiện***1.7.3.1. Tính tổng**1.7.3.1.1 Tính tổng toàn bộ các phần tử trong mảng*

VD: tính tổng tất cả các số có trong mảng

```

int TinhTong (int A[], int n )
{
    int tong = 0;
    for (int i = 0; i < n; i++ )
        tong = tong + A[i] ;
    return tong;
}

```

*1.7.3.1.2 Tính tổng các phần tử trong mảng thỏa điều kiện cho trước*

VD: tính tổng các số âm có trong mảng

```

int TinhTong (int A[], int n )
{
    int tong = 0;
    for (int i = 0; i < n; i++ )
        if (A[i]<0)
            tong = tong + A[i] ;
    return tong;
}

```

*1.7.3.2. Tính trung bình**1.7.3.2.1 Tính giá trị trung bình toàn bộ các phần tử trong mảng*

VD: tính tổng tất cả các số có trong mảng

```

double TrungBinhAm (int A[], int n )
{
    long tong = 0;
    for (int i = 0; i < n; i++ )
        if( A[i]<0 )
            tong = tong + A[i] ;
    return (double)tong/spt;
}

```

*1.7.3.2.2 Tính tổng các phần tử trong mảng thỏa điều kiện cho trước*

Đối với hàm tính trung bình có điều kiện phải lưu ý khi chia giá trị (có thể mảng không có phần tử nào thỏa điều kiện, nếu ta chia tức là chia cho 0).

VD: tính tổng các số âm có trong mảng

```

double TrungBinhAm (int A[], int n )
{
    long tong = 0;
    int spt=0;
    for (int i = 0; i < n; i++ )
        if( A[i]<0 )
        {
            tong = tong + A[i] ;
            spt++;
        }
    if(spt==0)

```

```

    return 0;
    return (double)tong/spt;
}

```

#### 1.7.4. Kỹ thuật đặt cờ hiệu

Kỹ thuật này thường được áp dụng cho những bài toán “**kiểm tra**” (có hay không, đúng hay sai, ...) hay “**đánh dấu**”.

##### 1.7.4.1. Dạng 1: (dạng $\forall$ )

Sử dụng khi cần kiểm tra điều kiện trên toàn bộ mảng (mọi phần tử đều phải thoả điều kiện)

##### 1.7.4.1.1 Kiểm tra giá trị từng phần tử riêng lẻ

Thường dùng trong những trường hợp kiểm tra mảng toàn dương, toàn âm, đồng nhất, ... Khi đó người ta thường dùng lượng từ tồn tại  $\exists$  để kiểm tra. Ví dụ:

<i>Yêu cầu cần kiểm tra</i>	<i>Điều kiện cần kiểm tra</i>
Tất cả các phần tử trong mảng đều là số dương	Không tồn tại phần tử trong mảng là số âm
$A[i] \geq 0; \forall i$	$\neg \exists (A[i] < 0) \Rightarrow$ là đúng ①

Lấy phủ định 2 vế của ①

$\Leftrightarrow \exists (A[i] < 0) \Rightarrow$  là sai ②

Hay nói cách khác điều kiện dùng trong phát biểu IF là điều kiện ngược với điều kiện muốn kiểm tra. Nếu điều kiện ngược này xuất hiện thì kết thúc (KHÔNG thành công).

VD: Viết hàm kiểm tra xem tất cả các phần tử có trong mảng đều là số dương hay không? (Trả về true nếu mảng toàn dương, ngược lại trả về false).

**bool KiemTraMangToanDuong (int A[ ], int n)**

```

{
    for (int i = 0; i < n; i++)
        if (A[i] < 0) //  $\Leftrightarrow \neg (A[i] \geq 0) \Rightarrow$  Vi phạm điều kiện dương
            return false;
    //Nếu đã tìm hết trên mảng nhưng vẫn không thấy số âm
    //  $\Rightarrow$  mảng chứa toàn số dương
    return true;
}

```

##### 1.7.4.1.2 Kiểm tra giá trị giữa 2 phần tử trong mảng

Thường dùng trong những trường hợp kiểm tra mảng tăng, mảng giảm, mảng đối xứng, ...

VD: Viết hàm kiểm tra xem có phải là mảng tăng hay không? (Trả về true nếu mảng tăng dần, ngược lại trả về false).

**bool KiemTraMangTangDan (int A[ ], int n)**

```

{
    for (int i = 0; i < n-1; i++)
        if (A[i] > A[i+1]) //  $\Leftrightarrow \neg (A[i] \leq A[i+1]) \Rightarrow$  Vi phạm
            return false;
    //Nếu đã duyệt hết trên mảng nhưng vẫn không thấy vi phạm
    //  $\Rightarrow$  mảng tăng
    return true;
}

```

**1.7.4.2. Dạng 2: (dạng  $\exists$ )**

Sử dụng khi cần kiểm tra có xuất hiện (hay tồn tại) một giá trị x nào đó. Khi đó điều kiện dùng trong phát biểu IF là điều kiện đúng với điều kiện muốn kiểm tra. Nếu điều kiện này xuất hiện thì kết thúc (THÀNH CÔNG).

VD: Viết hàm kiểm tra xem trong mảng các số nguyên có tồn tại số lẻ?

```
bool KiemTraMangCoChuaSoLeLonHon100 (int A[ ], int n)
{
    for (int i = 0; i < n; i ++ )
        if (A[i]%2==1)//Gặp phần tử thoả
            return true;
    /* Khi kết thúc vòng lặp for (đã duyệt hết mảng) nhưng không thấy
    phần tử thỏa điều kiện (chưa return true)⇒ mảng không chứa số Lẻ*/
    return false;
}
```

**1.7.5. Kỹ thuật đặt lính canh**

Kỹ thuật này thường được áp dụng cho những bài tập về “tìm kiếm”, “liệt kê” theo một điều kiện nhất định nào đó.

**1.7.5.1. Viết hàm tìm và trả về giá trị lớn nhất trong mảng một chiều các số nguyên.**

```
int TimMax (int A[], int n)
{
    int max;
    max = A[0];
    for (int i = 1; i < n ; i ++ )
        if ( A[i] > max )
            max = A[i] ;
    return max;
}
```

**1.7.5.2. Viết hàm tìm phần tử có giá trị x xuất hiện đầu tiên trong mảng một chiều. (Nếu tìm thấy trả về vị trí xuất hiện x, ngược lại trả về -1)**

```
int TimX (int A[], int n, int x)
{
    for (int i = 0; i < n ; i ++ )
        if ( x==A[i] )
            return i;
    return -1;
}
```

**1.7.6. Đếm số lần xuất hiện của số trong mảng**

Có thể chia các bài toán về đếm thành 3 loại:

**1.7.6.1. Đếm số lần xuất hiện của số thỏa điều kiện cho trước**

Thường dùng trong những trường hợp đếm số lượng số có giá trị = x, đếm số âm, đếm số nguyên tố, ...)

```
int DemSoChiaChanCho5 (int A[], int n )
{
    int dem = 0;
    for (int i = 0; i < n ; i++ )
        if ( A[i] % 5 == 0 )
            dem++;
    return dem;
}
```

### 1.7.6.2. Đếm số lần xuất hiện của 1 giá trị nào đó

Thường dùng trong những trường hợp đếm số lượng số có giá trị bằng với giá trị x, hay nhỏ nhất, hoặc số trung bình, ...) xuất hiện trong mảng. Tương tự như 4.2.6.1, nhưng trước khi thực hiện đếm, cần qua bước tìm số lớn nhất (số nhỏ nhất, số trung bình, ...).

```
int DemSoLanXuatHienCuaSoLonNhat (int A[], int n )
{   //Sử dụng hàm TimMax trong các ví dụ ở các phần trước
    int max=TimMax(a,n);
    int dem = 0;
    for (int i = 0; i < n ; i++ )
        if ( A[i] == max )
            dem++;
    return dem;
}
```

### 1.7.6.3. Đếm số lần xuất hiện của từng số có trong mảng

Yêu cầu: đếm số lần xuất hiện của các số có trong mảng A (chỉ chứa số nguyên dương).

Sử dụng mảng phụ (B) chứa kết quả đếm, với chỉ số của mảng phụ đại diện cho số xuất hiện trong mảng chính (A).

```
int DemSoLanXuatHienCuaCacSoTrongMang (int A[], int n )
{
    //Gọi hàm TimMax đã có trong các ví dụ trước
    int max=TimMax(a,n);
    //mảng B gồm max+1 phần tử
    int B[max+1];
    for (int i = 0; i < max+1 ; i++ )
        B[i]=0;
    //đếm các số trên mảng A. Kết quả đếm lưu vào mảng B
    for (int i = 0; i < n ; i++ )
        B[A[i]]++;
    //xuất kết quả đếm
    printf("Cac so xuat hien trong mang A:\n");
    for (int i = 0; i < max+1 ; i++ )
        if (B[i]>0)
            printf("So %d xuat hien %d lan",i,B[i]);
}
```

## 1.7.7. Sắp xếp

Viết hàm sắp xếp mảng theo thứ tự tăng dần.

```
void HoanVi (int &a, int &b)
{   int tam = a;
    a = b;
    b = tam;
}

void SapTang (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])
                HoanVi (A[i], A[j]);
}
```



**1.7.8. Xóa 1 phần tử khỏi mảng**

Các bước thực hiện:

- B1:** Duyệt mảng từ trái sang phải, trong quá trình duyệt, sẽ tìm giá trị (hoặc vị trí) cần xóa.
- B2:** Xuất phát từ vị trí cần xóa tiến hành dời lần lượt các phần tử về phía trước cho đến khi kết thúc mảng.
- B3:** Giảm kích thước mảng.

*1.7.8.1. Viết hàm xóa phần tử tại vị trí (vị trí) cho trước trong mảng*

```
void XoaTaiViTri (int A[], int &n, int vitri)
{
    //Dời sang tri từ vitri den n-1
    for (int i = vitri; i < n ; i++)
        A[i] = A[i+1];
    //Giảm n đi 1 đơn vị
    n--;
}
```

*1.7.8.2. Viết hàm xóa tất cả các phần tử trong mảng có giá trị =Z*

```
void XoaTatCaGiaTri (int A[], int &n, int SoCanXoa)
{
    for (int i = 0; i < n ; )
        /*để trống biểu thức 3 nhằm tránh bỏ sót khi có 2 phần tử cần xóa nằm liền kề nhau*/
        if (A[i] == SoCanXoa)
            XoaTaiViTri (a, n, i);
        else
            i++; // tăng biến chạy của vòng lặp
}
```

Hàm **XoaTatCaGiaTri** thường được viết lại bằng cách thay thế phát biểu FOR bằng phát biểu WHILE như sau:

```
void XoaTatCaGiaTri (int A[], int &n, int SoCanXoa)
{
    int i = 0;
    while (i < n)
        if (A[i] == SoCanXoa)
            XoaTaiViTri (A, n, i);
        else
            i++;
}
```

**1.7.9. Chèn (hay thêm) 1 phần tử vào mảng**

Các bước thực hiện:

- B1:** Duyệt mảng từ phải sang trái để tìm vị trí cần chèn.
- B2:** Xuất phát từ cuối mảng tiến hành đẩy lần lượt các phần tử về phía sau cho đến vị trí cần chèn.
- B3:** Chèn phần tử cần chèn vào vị trí chèn
- B4:** Tăng kích thước mảng.

*1.7.9.1. Thêm phần tử có giá trị X vào cuối mảng*

```
void ThemCuoi (int A[], int &n, int X)
{
    A[n]=X;
    n++;
}
```

**1.7.9.2. Chèn phần tử có giá trị X vào mảng tại vị trí cho trước**

```
void ChenXVaoViTri (int A[], int &n, int X, int vitri)
{
    /* Xuất phát từ cuối mảng tiến hành đẩy lần lượt các phần tử về phía
    sau cho đến vị trí cần chèn*/
    for (int i = n; i > vitri ; i--)
        A[i] = A[i-1] ;
    // Đưa phần tử cần chèn vào vị trí chèn
    A[vitri] = X;
    // Tăng kích thước mảng
    n++;
}
```

**1.7.9.3. Chèn phần tử có giá trị X vào sau giá trị Y đầu tiên (tính từ trái sang phải) có trong mảng. Nếu trong mảng không tồn tại giá trị Y thì thực hiện thêm X vào cuối mảng.**

```
void ChenXVaoSauY (int A[], int &n, int X, int Y)
{
    int i;
    for (i = 0; i < n ; i++)
        if (A[i] == Y)
        {
            ChenXVaoViTri(A, n, X, i);
            break;
        }
    if (i==n)//Y không tồn tại  $\Rightarrow$  thêm X vào cuối mảng
        ThemCuoi (A, n, X);
}
```

**1.7.10. Tách mảng**

- **Ví dụ 1:** Cho mảng A kích thước n. Tách mảng A thành 2 mảng B và C sao cho: B có 1/2 phần tử đầu của mảng A, 1/2 phần tử còn lại đưa vào mảng C.

Vậy nếu n là số chẵn thì số lượng 2 mảng B và C bằng nhau; ngược lại nếu n lẻ mảng C sẽ nhiều hơn mảng B 1 phần tử.

```
void TachMang(int A[], int nA, int B[], int &nB, int C[], int &nC)
{
    int k = nA/2;
    nB = nC = 0;
    for(int i=0; i<k; i++)
    {
        B[nB++]=A[i];
        C[nC++]=A[k+i];
    }
}
```

- **Ví dụ 2:** Cho mảng A kích thước n. Tách mảng A thành 2 mảng B và C sao cho: B chỉ chứa các số lẻ có trong mảng A, C chỉ chứa các số chẵn có trong mảng A.

```
void TachMang(int A[], int nA, int B[], int &nB, int C[], int &nC)
{
    nB = nC = 0;
    for(int i=0; i<nA; i++)
        if (A[i]%2==0)
            B[nB++]=A[i];
        else
            C[nC++]=A[i];
}
```

```
        C[nC++]=A[i];  
    }  
}
```

### 1.7.11. Ghép mảng

#### 1.7.11.1. Ghép tuần tự

Cho 2 mảng số nguyên A và B kích thước lần lượt là n và m. Viết chương trình nối mảng B vào cuối mảng A.

```
void NoiMang(int A[], int &n, int B[], int m)  
{  
    for(int i=0; i<m; i++)  
        A[n+i]=B[i];  
    n=n+m;  
}
```

#### 1.7.11.2. Ghép xen kẽ (đan xen)

Cho 2 mảng số nguyên A và B kích thước lần lượt là na và nb đều đã được sắp xếp tăng dần. Viết hàm nối 2 mảng A và B vào mảng C sao cho mảng C cũng được sắp xếp tăng dần.

Cách thực hiện:

**B1:** Đưa lần lượt từng phần tử của mảng A và mảng B vào mảng C, tăng chỉ số tương ứng.

**B2:** Nếu một trong hai mảng hết trước thì chép tất cả các phần tử còn lại của mảng chưa hết vào mảng C.

Đặt **iA** là chỉ số của mảng A; **iB**: chỉ số của mảng B và **iC** là chỉ số của mảng C.

```
void NoiMang(int A[],int nA, int B[], int nB, int C[], int &nC)  
{  
    int iA=0, iB=0;  
    nC=0;  
    while ( (iA<nA) && (iB<nB) )  
    {  
        if (A[iA]<B[iB])  
            C[nC++]=A[iA++];  
        else  
            C[nC++]=B[iB++];  
    }  
    //nếu mảng A còn thì đưa hết vào C  
    while(iA<nA)  
        C[nC++]=A[iA++];  
    //nếu mảng B còn thì đưa hết vào C  
    while(iB<nB)  
        C[nC++]=B[iB++];  
}
```

## 1.8. Phần thực hành

### 1.8.1. Mục tiêu

Sinh viên phải biết cách nhập, xuất, lưu trữ trên mảng một chiều với các dữ liệu kiểu số, xử lý các bài toán như tính tổng các giá trị trên mảng một chiều các số nguyên, số thực, tìm kiếm các phần tử nhỏ nhất, lớn nhất, chẵn đầu, lẻ cuối...thêm, xóa, sắp xếp các phần tử trên mảng dữ liệu kiểu số.

### 1.8.2. Yêu cầu

- (i). Viết hàm nhập số phần tử của mảng ( $0 < n \leq 100$ ).
  - (ii). Viết hàm nhập vào mảng một chiều các số nguyên gồm  $n$  phần tử
  - (iii). Viết hàm xuất mảng số nguyên  $n$  phần tử vừa nhập ở trên .
  - (iv). Viết hàm main () kết 3 hàm trên . chạy ổn định rồi mới viết tiếp hàm khác
  - (v). Viết hàm tính tổng các phần tử có trong mảng .
  - (vi). Kết hàm tính tổng trong hàm main () ....
- Chú ý: Sau khi hoàn tất các hàm nhập xuất, thực hiện kết 3 hàm trên vào hàm main(). Khi hàm main đã chạy ổn định mới viết tiếp hàm khác. Viết xong hàm nào kết ngay hàm đó trong hàm main().

Ví dụ:

- (i). Viết hàm nhập số phần tử của mảng ( $0 < n \leq 100$ ).

```
void NhapN (int &n)
{
    do
    {
        printf ("Nhap so phan tu cua mang (tu 1-100): ");
        scanf("%d",&n);
        if ( (n<=0) || (n>100) )
            printf ("Chi nhan cac so tu 1 den 100.");
    }while ( (n<=0) || (n>100) );
}
```

- (ii).Viết hàm nhập vào mảng một chiều các số nguyên 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]);
    }
}
```

- (iii). Viết hàm xuất mảng số nguyên  $n$  phần tử vừa nhập ở trên

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

- (iv). Viết hàm main () kết 3 hàm trên. Khi đã chạy ổn định rồi mới viết tiếp hàm khác

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int a[100] ,n ;
    NhapN (n);
    NhapMang(a , n);
```

```
printf (" mang vua nhap : ");
XuatMang(a , n) ;
getch() ;
}
```

## 1.9. Bài tập (sinh viên tự thực hiện)

### 1.9.1.1. Thao tác trên một mảng

#### 1.9.1.1.1 Nhập mảng

##### Nhập giá trị các phần tử của mảng từ bàn phím

- (1)- Tạo mảng A gồm n phần tử ( $n > 0$ ), với yêu cầu giá trị của các phần tử của mảng thỏa điều kiện đều là số dương:

##### Mở rộng:

- Các trường hợp số âm, số chẵn, số vừa âm vừa lẻ, ...
- Nằm trong khoảng từ 50 đến 100.
- Nhỏ hơn 10 hoặc lớn hơn hay bằng 50.

- (2)- Tạo mảng A gồm n phần tử ( $n > 0$ ), với yêu cầu những vị trí (hay chỉ số mảng) là số lẻ chỉ nhận giá trị nhập vào là số lẻ, và những vị trí (hay chỉ số mảng) là số chẵn chỉ nhận giá trị nhập vào là số chẵn.

##### Mở rộng cho trường hợp ngược lại: vị trí lẻ chỉ nhận số chẵn; vị trí chẵn chỉ nhận số lẻ.

- (3)- Tạo mảng A gồm n phần tử ( $n > 0$ ), với yêu cầu giá trị của phần tử nhập sau trong mảng phải lớn hơn hoặc bằng phần tử liền trước (sau khi nhập hoàn tất, ta thu được mảng được sắp xếp tăng dần).

VD1:    2    4    7    7    8    11     $\Rightarrow$  Hợp lệ

VD2:    2    4    3     $\Rightarrow$  không hợp lệ  $\Rightarrow$  yêu cầu nhập lại

- (4)- Tạo mảng A gồm n phần tử ( $n > 0$ ), với yêu cầu chỉ cho người dùng nhập giá trị của các phần tử của mảng là số nguyên tố

##### Mở rộng cho các trường hợp số hoàn thiện, số chính phương, ...

##### Tạo mảng với giá trị của các phần tử được phát sinh ngẫu nhiên

- (5)- Lần lượt viết các hàm phát sinh ngẫu nhiên mảng 1 chiều các số nguyên dương gồm n phần tử trong từng trường hợp sau (mỗi hàm xử lý cho 1 trường hợp):

- a. Với  $5 < n < 200$ , và giá trị của các phần tử trong khoảng từ 0 đến 99.
- b. Với  $5 < n < 50$ , và cho giá trị của các phần tử trong khoảng từ -100 đến 100.
- c. Với  $5 < n < 50$ , và các số nguyên sao cho giá trị phát sinh ngẫu nhiên phải toàn là số lẻ.

##### Mở rộng cho các trường hợp số chẵn, số vừa âm vừa lẻ, số nguyên tố, số hoàn thiện, số hoàn hảo, ...

- (6)- Viết chương trình phát sinh ngẫu nhiên mảng 1 chiều các số nguyên gồm n phần tử, sao cho số phát sinh sau phải lớn hơn hay bằng số phát sinh liền trước đó (các phần tử của mảng được sắp xếp tăng dần).

##### Mở rộng cho trường hợp số giảm dần.

- (7)- Viết chương trình phát sinh ngẫu nhiên mảng 1 chiều các số nguyên gồm n phần tử, với yêu cầu những vị trí (hay chỉ số mảng) là số lẻ chỉ nhận giá trị nhập vào là số lẻ, và những vị trí (hay chỉ số mảng) là số chẵn chỉ nhận giá trị nhập vào là số chẵn.
- ✎ Mở rộng cho trường hợp ngược lại: vị trí lẻ chỉ nhận số chẵn; vị trí chẵn chỉ nhận số lẻ.

#### 1.9.1.1.2 Xuất mảng

- (8)- Xuất toàn bộ các phần tử có trong mảng.

##### ✎ Xuất theo giá trị phần tử có trong mảng

- (9)- Liệt kê các phần tử trong mảng có giá trị lẻ.
- ✎ Mở rộng: giá trị chẵn, giá trị âm, giá trị dương, ...
- (10)- Liệt kê các phần tử trong mảng có giá trị là số chẵn và nhỏ hơn 20.
- ✎ Mở rộng: giá trị chẵn và là số âm, giá trị lẻ và là số dương, ...
- (11)- Viết hàm (ngoài các tham số cần thiết) nhận tham số là số x, và in ra tất cả các số có giá trị lớn hơn x.
- ✎ Mở rộng: khác X, nhỏ hơn hay bằng X, bội số của X, ước số của X, ...
- (12)- Liệt kê các giá trị là số nguyên tố có trong mảng.
- ✎ Mở rộng cho các trường hợp số hoàn thiện, số hoàn hảo, ...
- (13)- Liệt kê các số trong mảng có giá trị thuộc [x,y] cho trước (x và y là tham số của hàm)
- ✎ Mở rộng: liệt kê các số chẵn trong mảng nguyên thuộc [x,y].

##### ✎ Xuất theo vị trí (hay chỉ số) của mảng

- (14)- Liệt kê các phần tử nằm tại vị trí lẻ.
- ✎ Mở rộng cho trường hợp: vị trí chẵn, vị trí là số hoàn thiện, số hoàn hảo, ...
- (15)- Liệt kê các phần tử trong mảng có vị trí là số chẵn và nhỏ hơn 20.
- ✎ Mở rộng cho trường hợp: vị trí từ p đến q (thay vì từ 0 đến n-1 như thường dùng), vị trí từ 0 đến p hoặc vị trí từ q đến n-1, ...
- (16)- Viết hàm (ngoài các tham số cần thiết) nhận tham số là số x, và in ra tất cả các số có vị trí lớn hơn x.
- ✎ Mở rộng: khác X, nhỏ hơn hay bằng X, bội số của X, ước số của X, ...).

##### ✎ Xuất dựa trên kết hợp giữa vị trí và giá trị

- (17)- In các phần tử có giá trị là số nguyên tố nằm tại những vị trí chẵn trong mảng.
- (18)- \* Liệt kê tất cả các cặp giá trị (a,b) trong mảng thỏa điều kiện  $a \geq b$  và vị trí (chỉ số) chứa số a < vị trí (chỉ số) chứa số b.
- (19)- Viết hàm nhận tham số vào 1 số K (với  $-999 \leq K \leq 999$ ). In ra cách đọc chữ số tương ứng (sử dụng mảng chứa các chuỗi “không”, “một”, “hai”, ...).

Ví dụ: nhập -456 in ra: Am bon nam sau.

✎ Mở rộng: cho trường hợp giá trị của  $K \leq \pm 2.000.000.000$ .

- (20)- Viết chương trình nhập vào năm. In ra tên của năm âm lịch tương ứng.



(25)- Kiểm tra xem mảng đã cho có phải là mảng tăng hay không? (mảng tăng là mảng có các phần tử sau luôn lớn hơn hay bằng phần tử trước nó)..

✎ Mở rộng: trường hợp mảng giảm, mảng đối xứng, mảng cấp số cộng, ...

#### 1.9.1.1.4 *Kỹ thuật đặt lính canh (áp dụng cho dạng bài toán “tìm kiếm một giá trị”, “tìm kiếm một vị trí”)*

##### ✎ Tìm vị trí

(26)- Tìm vị trí của phần tử đầu tiên có giá trị bằng x. Nếu trong mảng không chứa giá trị x, hàm trả về giá trị -1.

✎ Mở rộng:

- Tìm vị trí cuối cùng phần tử có giá trị x.
- Tìm vị trí đầu tiên chứa giá trị nhỏ hơn x, vị trí cuối cùng chứa giá trị nhỏ hơn x.
- Tìm vị trí phần tử có giá trị x xuất hiện cuối cùng trong mảng.

(27)- Viết hàm tìm vị trí của phần tử số âm đầu tiên có trong mảng. Nếu trong mảng không có số âm, hàm trả về giá trị -1.

✎ Mở rộng:

- Tìm vị trí của phần tử có giá trị là số lẻ/chẵn đầu tiên
- tìm vị trí của phần tử lớn nhất trong mảng
- Tìm vị trí các phần tử nguyên tố trong mảng
- In vị trí các phần tử là số nguyên tố có giá trị lớn hơn 23.
- Tìm số chính phương đầu tiên trong mảng.
- Trả về vị trí cuối cùng (thay vì tìm vị trí đầu tiên).

(28)- Viết hàm tìm vị trí của phần tử nhỏ nhất trong mảng. Nếu có nhiều giá trị cùng nhỏ nhất, hàm trả về vị trí đầu tiên của giá trị nhỏ nhất.

✎ Mở rộng: Tìm vị trí của phần tử lớn nhất trong mảng

(29)- Viết hàm tìm vị trí của phần tử Am lớn nhất đầu tiên trong mảng. *Nếu trong mảng không có số âm, hàm trả về giá trị -1.*

✎ Mở rộng:

- Tìm vị trí của phần tử có giá trị dương nhỏ nhất.
- Tìm vị trí các phần tử nguyên tố lớn/nhỏ nhất trong mảng
- Tìm vị trí chứa số chính phương lớn/nhỏ nhất trong mảng.
- Tìm vị trí đầu tiên của phần tử có giá trị là số hoàn thiện, số chính phương, số Armstrong.
- Tìm vị trí đầu tiên của phần tử có giá trị là số hoàn thiện nhỏ nhất, số chính phương nhỏ nhất, số Armstrong nhỏ nhất.

##### ✎ Tìm giá trị

(30)- Tìm giá trị của phần tử trong mảng “xa giá trị x nhất”

✎ Mở rộng:

- Tìm giá trị của phần tử trong mảng “gần giá trị x nhất”.
- Tìm cả giá trị và vị trí của phần tử trong mảng “xa/gần giá trị x nhất”.

(31)- Tìm đoạn [a,b] sao cho đoạn này chứa tất cả các giá trị trong mảng

(32)- Viết hàm nhận tham số là mảng các số nguyên (A), số lượng phần tử của mảng (n). Tìm giá trị x sao cho đoạn [-x,x] chứa tất cả các giá trị trong mảng.

(33)- Tìm giá trị đầu tiên nằm trong khoảng (x,y) cho trước. Nếu không có trả về giá trị -1.



(34)- Tìm giá trị đầu tiên trong mảng có dạng  $2^k$ . Nếu không có thì trả về giá trị -1.

✎ **Mở rộng:** Tìm giá trị đầu tiên trong mảng có dạng  $3^k, 4^k, \dots$

(35)- Tìm số chẵn lớn nhất nhỏ hơn mọi giá trị lẻ có trong mảng nguyên.

(36)- Tìm số nguyên tố nhỏ nhất lớn hơn mọi giá trị còn lại (không phải số nguyên tố) trong mảng.

(37)- Tìm ước chung lớn nhất của tất cả phần tử trong mảng nguyên dương.

(38)- Tìm bội số chung nhỏ nhất cho tất cả các phần tử trong mảng nguyên.

(39)- Tìm vị trí trong mảng số nguyên thỏa điều kiện giá trị tại vị trí đó lớn hơn giá trị có trong 2 vị trí liền kề. Nếu không có trả về -1. Bỏ qua (không xét) vị trí đầu và cuối mảng.

✎ **Mở rộng:**

- Giá trị tại vị trí đó bằng tổng 2 giá trị có trong vị trí kề cận

$$(A[i] = A[i-1] + A[i+1]).$$

- Giá trị tại vị trí đó bằng tích 2 giá trị có trong vị trí kề cận

$$(A[i] = A[i-1] * A[i+1]).$$

- Xét cả 2 vị trí đầu và cuối mảng (chỉ có 1 phần tử kề cận).

(40)- Hãy tìm giá trị đầu tiên trong mảng một chiều các số nguyên có chữ số đầu tiên là chữ số lẻ. Nếu trong mảng không tồn tại giá trị như vậy hàm sẽ trả về giá trị 0 (ví dụ: 110)

(41)- Tìm giá trị toàn là chữ số lẻ và lớn nhất trong những số thỏa điều kiện. Không có trả về -1.

(42)- Cho mảng một chiều các số nguyên hãy viết hàm tìm giá trị đầu tiên thỏa tính chất số gánh (ví dụ giá trị 12321).

(43)- Tìm 1 giá trị có số lần xuất hiện nhiều nhất trong mảng

(44)- Tìm chữ số xuất hiện nhiều nhất trong mảng

VD: mảng gồm 3 phần tử 15, 42, 14. Chữ số xuất hiện nhiều nhất là chữ số 1 và chữ số 4.

✎ **Mở rộng:** Tìm chữ số xuất hiện ít nhất trong mảng

(45)- Tìm 2 giá trị gần nhau nhất trong mảng

(46)- Viết hàm nhận tham số là mảng các số nguyên (A), số lượng phần tử của mảng (n) và số nguyên x. Tìm giá trị trong mảng các số nguyên “xa giá trị x nhất” (xanhat)

Ví dụ: cho mảng A

24	45	23	13	43	-12
----	----	----	----	----	-----

Với giá trị  $x = 15$ , Khoảng cách từ x tới các phần tử khác trong mảng là:

9	30	8	2	28	27
---	----	---	---	----	----

Giá trị trong mảng xa giá trị x nhất là: 45

✎ **Mở rộng:** Tìm phần tử đầu tiên trong mảng “gần giá trị x nhất”.

Ví dụ: cho mảng A

24	45	23	13	43	-12
----	----	----	----	----	-----

Với giá trị  $x = 15$ , Khoảng cách từ x tới các phần tử khác trong mảng là:

9	30	8	2	28	27
---	----	---	---	----	----

Giá trị trong mảng gần giá trị x nhất là: 13

#### 1.9.1.1.5 Tính tổng

(47)- Viết hàm tính tổng các phần tử nằm ở vị trí chẵn trong mảng.

✎ Mở rộng: Viết hàm tính tổng các phần tử nằm ở vị trí chia hết cho 5, vị trí là số nguyên tố.

(48)- Tổng các phần tử có chữ số đầu là chữ số lẻ

✎ Mở rộng:

- Tổng các phần tử có chữ số đầu là chẵn,...
- Tổng các phần tử có chữ số hàng chục là 5

(49)- Tổng các phần tử lớn hơn phần tử đứng liền trước nó

(50)- Tổng các phần tử lớn hơn trị tuyệt đối của phần tử liền sau nó

(51)- Tổng các phần tử lớn hơn phần tử liền kề

(52)- Tổng các phần tử đối xứng

(53)- Viết hàm tính tổng của từng dãy con giảm có trong mảng.

(54)- Tính tổng các phần tử cực đại trong mảng các số nguyên (phần tử cực đại là phần tử lớn hơn các phần tử liền kề).

Ví dụ:            1 5 2 6 3 5 1 8 6      $\Rightarrow$  in ra 24

✎ Mở rộng: Tính tổng các phần tử cực tiểu trong mảng.

#### 1.9.1.1.6 Đếm

(55)- Đếm số lần xuất hiện của giá trị x trong mảng.

✎ Mở rộng:

- Đếm số lần xuất hiện của giá trị dương, giá trị âm, số chẵn, số lẻ, bội số của 5, bội số của X, ước số của 7, ước số của X, ...
- Đếm số lần xuất hiện của giá trị là số nguyên tố, là số hoàn thiện, là số chính phương, số Armstrong, ...

(56)- Cho biết sự tương quan giữa số lượng chẵn và lẻ trong mảng (bao nhiêu phần trăm số lẻ, bao nhiêu phần trăm là số chẵn)

(57)- Đếm các phần tử có chữ số đầu là chữ số lẻ

✎ Mở rộng:

- Đếm các phần tử có chữ số đầu là chẵn, ...
- Đếm các phần tử có chữ số hàng chục là 5
- Đếm các phần tử có chữ số hàng đơn vị là 1 trong các số 3, 6, 9

(58)- Đếm số đối xứng trong mảng

(59)- Đếm số lượng phần tử lớn hơn các phần tử liền kề, thực hiện cho 2 trường hợp:

- KHÔNG xét 2 phần tử đầu và cuối mảng vì không đủ 2 phần tử liền kề).
- CÓ xét 2 phần tử đầu và cuối mảng (2 phần tử này chỉ có 1 phần tử liền kề trước hoặc sau).

✎ Mở rộng:

- Đếm số lượng phần tử kề nhau mà cả 2 trái dấu.
- Đếm số lượng phần tử kề nhau mà cả 2 đều chẵn.
- Đếm số lượng phần tử kề nhau mà số đứng sau cùng dấu số đứng trước và có giá trị tuyệt đối lớn hơn.

#### 1.9.1.1.7 Trung bình

(60)- Trung bình cộng các số dương

✎ Mở rộng:

- Trung bình cộng các số âm, số chẵn, số lẻ.
- Trung bình cộng các số lớn hơn x, nhỏ hơn x
- Trung bình cộng các số nguyên tố, các số hoàn thiện, các số chính phương, ...
- Trung bình nhân các số dương, số âm, số chẵn, số lẻ

(61)- (\*) Khoảng cách trung bình giữa các giá trị trong mảng

(62)- Viết hàm tính giá trị trung bình của các số hoàn thiện trong mảng.

✎ Mở rộng:

- Tính giá trị trung bình của các phần tử có giá trị âm, các phần tử có giá trị là số lẻ/số chẵn,...
- Tính giá trị trung bình của các phần tử là bội của 3 và 5 trong mảng.
- Tính giá trị trung bình của các phần tử có giá trị là số nguyên tố, là số hoàn thiện, là số chính phương, ...

#### 1.9.1.1.8 Kỹ thuật thêm

(63)- Chèn thêm giá trị x vào vị trí k trong mảng một chiều nguyên (x, k là tham số đầu vào của hàm).

(64)- Giả sử các phần tử trong mảng đã được sắp xếp tăng dần. Viết hàm thêm giá trị x vào mảng sao cho mảng vẫn tăng dần (không sắp xếp).

(65)- Thêm giá trị y vào sau các phần tử có giá trị x trong mảng một chiều nguyên (x, y là tham số đầu vào của hàm).

(66)- Viết hàm chèn phần tử có giá trị X vào vị trí đầu tiên của mảng.

(67)- Viết hàm chèn phần tử có giá trị X vào phía sau phần tử có giá trị lớn nhất trong mảng.

(68)- Viết hàm chèn phần tử có giá trị X vào trước phần tử có giá trị là số nguyên tố đầu tiên trong mảng.

(69)- Viết hàm chèn phần tử có giá trị X vào phía sau tất cả các phần tử có giá trị chẵn trong mảng.

#### 1.9.1.1.9 Kỹ thuật xóa

(70)- Xóa phần tử có chỉ số k trong mảng một chiều nguyên.

✎ Mở rộng: chỉ số là số chẵn, số lẻ, số nguyên tố, bội số của m, ...

(71)- Xóa tất cả phần tử có giá trị bằng X.

✎ Mở rộng: giá trị nhỏ/lớn hơn X, số chẵn, số lẻ, số âm, giá trị là số nguyên tố, ...

(72)- Xóa tất cả các số (dương) lớn nhất trong mảng một chiều nguyên.

✎ Mở rộng: cho các giá trị âm lớn nhất, chẵn lớn nhất, lẻ lớn nhất, nguyên tố lớn nhất, bội số lớn nhất (hay ước số lớn nhất) của số k, ...

(73)- Xóa tất cả các phần tử có giá trị xuất hiện nhiều hơn một lần trong mảng một chiều nguyên.

(74)- Xóa phần tử tại vị trí lẻ trong mảng.

(75)- Nhập vào giá trị X. Viết hàm xóa phần tử có giá trị gần X nhất.

#### 1.9.1.1.10 Kỹ thuật xử lý mảng

- (76)- Đảo ngược thứ tự mảng một chiều các số nguyên.
- (77)- Đảo ngược thứ tự các giá trị chẵn trong mảng một chiều nguyên.
- (78)- Chuyển các phần tử có giá trị chẵn về đầu mảng.
- (79)- Chuyển các phần tử có giá trị âm về cuối mảng.
- (80)- Dịch trái xoay vòng 1 lần trên mảng một chiều.

✎ Mở rộng:

- Dịch phải xoay vòng 1 lần trên mảng một chiều.
- Dịch phải/trái k lần; ...

#### 1.9.1.1.11 Sắp xếp mảng

- (81)- Sắp xếp bằng phương pháp đổi chỗ trực tiếp.
- (82)- Sắp xếp sao cho số âm giảm dần, số dương tăng dần.
- (83)- Sắp xếp các phần tử chẵn nằm bên trái theo thứ tự tăng dần còn các phần tử lẻ bên phải theo thứ tự giảm dần.
- (84)- Sắp xếp các phần tử âm giảm dần từ trái sang phải, phần tử dương tăng dần từ phải sang trái.
- (85)- Sắp xếp sao cho số lẻ tăng dần, số dương giữ nguyên vị trí.

✎ Mở rộng: Sắp xếp mảng theo thứ tự tăng dần của các phần tử là số nguyên tố, số chính phương (các phần tử khác giữ nguyên vị trí)

#### 1.9.1.1.12 Kỹ thuật mảng con

- (86)- Liệt kê các mảng con ( $> 2$  phần tử) tăng trong mảng một chiều.

✎ Mở rộng:

- Liệt kê các mảng con toàn dương.
- Liệt kê mảng con tăng có tổng giá trị các phần tử là lớn nhất.
- Liệt kê mảng con tăng có số lượng phần tử nhiều nhất (dài nhất). Nếu có 2 dãy con dài bằng nhau thì xuất mảng con đầu tiên.

Ví dụ: Nhập mảng 1 4 2 3 1 2 6 8 3 5 7

Mảng con dài nhất là: 1 2 6 8

📖 Hướng dẫn:

- Khởi động các biến `DauMax`, `CuoiMax`, `DauMoi`, `CuoiMoi` = vị trí đầu mảng, `DemMax`, `DemMoi` = 1.
- Duyệt mảng:

*Nếu còn tăng và chưa hết mảng thì (`CuoiMoi` = vị trí đang xét và `DemMoi` tăng 1)*

*Ngược lại thì so sánh:*

*Nếu  $DemMoi > DemMax$*

*thì ( $DauMax = DauMoi$  và  $CuoiMax = CuoiMoi$ ).*

- (87)- Cho 2 mảng A, B các số nguyên (kích thước mảng A nhỏ hơn mảng B). Hãy kiểm tra xem A có phải là con của B hay không?

- (88)- (\*) Viết chương trình tính trung bình cộng của các tổng các mảng tăng dần có trong mảng các số nguyên.

Ví dụ:  $1\ 2\ 3\ 4\ 2\ 3\ 4\ 5\ 6\ 4\ 5\ 6 \Rightarrow TB = 15$ .

#### 1.9.1.1.13 Tổng hợp

- (89)- Nhập vào một mảng số thực kết thúc bằng việc nhập số 0 (số 0 không lưu vào mảng) hoặc khi đã nhập đủ 20 số. Kiểm tra có hay không các tính chất sau đây của mảng:
- Mảng đơn nhất? (Không có phần tử trùng nhau trong mảng)
  - Mảng đan dấu? (2 phần tử kế nhau phải khác dấu. Mảng 1 phần tử xem như đan dấu)
  - Mảng tuần hoàn? (Mảng tuần hoàn: Nếu  $A_i, A_{i+1}, A_{i+2}$  là 3 phần tử liên tiếp trong mảng thì:  $A_{i+1} \geq A_i$  và  $A_{i+1} \geq A_{i+2}$  hoặc  $A_{i+1} \leq A_i$  và  $A_{i+1} \leq A_{i+2}$ . Mảng có 2 phần tử xem như tuần hoàn).
  - Mảng tăng dần? Mảng giảm dần?
- (90)- Nhập vào một mảng số nguyên gồm n phần tử. Tạo menu và thực hiện các thao tác trên mảng:
- Xuất mảng ra màn hình.
  - Đếm số phần tử là bội số của 3 của mảng (đếm số phần tử có điều kiện).
  - Tìm phần tử lớn nhất, nhỏ nhất và tính tổng, tích các phần tử của mảng.
  - Thêm vào mảng phần tử x tại vị trí k ( $k < n$ ).
  - Xóa phần tử tại vị trí k ( $k < n$ ).
  - Tìm phần tử x trong mảng, chỉ ra vị trí xuất hiện của x.
  - Tìm cặp phần tử có tổng bình phương đúng bằng k (k nhập từ bàn phím).
  - Sắp xếp mảng tăng dần.
  - Sắp xếp các phần tử ở vị trí chẵn tăng dần, vị trí lẻ giảm dần.
- (91)- Nhập mảng số nguyên a có n phần tử ( $0 < n \leq 20$ ). Tạo menu để thực hiện các công việc:
- Sắp các phần tử vị trí lẻ tăng dần, các phần tử vị trí chẵn giảm dần
  - Sắp các phần tử dương về đầu mảng có thứ tự giảm dần, các phần tử âm cuối mảng có thứ tự tăng dần.
  - Sắp các số nguyên tố về đầu mảng có thứ tự tăng dần, các phần tử còn lại có thứ tự giảm dần.
- (92)- Viết hàm liệt kê các bộ 4 số a, b, c, d trong mảng các số nguyên (có ít nhất 4 phần tử và đôi một khác nhau) sao cho  $a + b = c + d$ .
- (93)- (\*) Cho mảng các số nguyên a gồm n phần tử ( $n \leq 30000$ ) và số dương k ( $k \leq n$ ). Hãy chỉ ra số hạng lớn thứ k của mảng.

Ví dụ: Mảng a: 6 3 1 10 11 18

k = 3

Kết quả: 10

- (94)- Viết chương trình tính tổng tất cả các phần tử xung quanh trên mảng các số nguyên. Biết rằng *Phần tử xung quanh là hai phần tử bên cạnh cộng lại bằng chính nó*; ví dụ: 1 3 2  $\rightarrow$  1,2 là hai phần tử xung quanh của 3).

Ví dụ: 1 3 2 5 3 9 6  $\rightarrow$  tổng 17

- (95)- (\*\*) Viết chương trình nhập vào hai số lớn a, b nguyên (a, b có từ 20 chữ số trở lên). Tính tổng, hiệu, tích, thương của hai số trên.

- (96)- Viết hàm tìm và xóa tất cả các phần tử trùng với x trong mảng một chiều các số nguyên, nếu không tồn tại phần tử x trong mảng thì trả về -1.
- (97)- (\*\*) Viết hàm xoá những phần tử sao cho mảng kết quả có thứ tự tăng dần và số lần xoá là ít nhất.
- (98)- Cho mảng a gồm n số nguyên có thứ tự tăng dần. Nhập vào một phần tử nguyên X, viết hàm chèn X vào mảng sao cho mảng vẫn có thứ tự tăng dần (không sắp xếp).
- (99)- Viết chương trình tìm số lẻ nhỏ nhất lớn hơn mọi số chẵn có trong mảng.
- (100)- Viết hàm tìm giá trị chẵn nhỏ nhất nhỏ hơn mọi giá trị lẻ trong mảng các số nguyên.
- (101)- Viết hàm tìm phần tử xuất hiện nhiều nhất trong mảng các số nguyên.
- (102)- Viết chương trình đếm và liệt kê các mảng con tăng dần trong mảng một chiều các số nguyên.

Ví dụ: 6 5 3 2 3 4 2 7 các mảng con tăng dần là 2 3 4 và 2 7

- (103)- Viết chương trình tìm mảng con tăng dần có tổng lớn nhất trong mảng một chiều.
- (104)- (\*) Viết chương trình nhập vào một mảng số a gồm n số nguyên ( $n \leq 100$ ). Tìm và in ra mảng con tăng dài nhất

Ví dụ: Nhập mảng a: 1 2 3 6 4 7 8 3 4 5 6 7 8 9 4 5

Mảng con tăng dài nhất: 3 4 5 6 7 8 9

- (105)- Viết chương trình nhập vào mảng số a gồm n số nguyên ( $n \leq 100$ ).
- a. Hãy đảo ngược mảng đó.

Ví dụ: Nhập a: 3 4 5 2 0 4 1

Mảng sau khi đảo: 1 4 0 2 5 4 3

- b. (\*) Hãy kiểm tra xem mảng đã cho có thứ tự chưa (mảng được gọi là thứ tự khi là mảng tăng hoặc mảng giảm).
- (106)- Cho mảng A có n phần tử hãy cho biết mảng này có đối xứng hay không.
- (107)- Cho mảng A có n phần tử. Nhập vào số nguyên k ( $k \geq 0$ ), dịch phải xoay vòng mảng A k lần.

Ví dụ: Mảng A: 5 7 2 3 1 9

Nhập k = 2

Dịch phải xoay vòng mảng A: 1 9 5 7 2 3

- (108)- (\*\*) Viết chương trình in ra tam giác Pascal (dùng mảng một chiều).
- (109)- Tìm giá trị trong mảng các số thực “ xa giá trị x nhất”

Ví dụ:

24	45	23	13	43	-12
----	----	----	----	----	-----

Giá trị x: 15

Khoảng cách từ x = 15 tới các phần tử khác trong mảng là:

9	30	8	2	28	27
---	----	---	---	----	----

Giá trị trong mảng xa giá trị x nhất là: 45

- (110)- Tìm một vị trí trong mảng một chiều các số thực mà tại vị trí đó là giá trị “gần giá trị x nhất”.

Ví dụ:

24	45	23	13	43	-12
----	----	----	----	----	-----

Giá trị x: 15

Khoảng cách từ x = 15 tới các phần tử khác trong mảng là:

9	30	8	2	28	27
---	----	---	---	----	----

Giá trị trong mảng gần giá trị x nhất là: 13

### 1.9.1.2. Thao tác trên nhiều mảng

#### 1.9.1.2.1 Nhập xuất mảng

- (111)- Cho mảng một chiều nguyên a. Viết hàm tạo mảng b từ a sao cho b chỉ chứa các giá trị lẻ của a.

✎ Mở rộng: cho số nguyên tố, số hoàn thiện, số chính phương, ...

- (112)- Cho mảng một chiều nguyên a. Viết hàm tạo mảng b từ a sao cho b chứa vị trí của các phần tử có giá trị lớn nhất trong a.

- (113)- Viết chương trình cho phép người dùng nhập vào mảng A các số nguyên gồm n phần tử ( $1 < n < 15$ ). Hãy tạo mảng B sao cho:

$$B[i] = (2 * A[i] - 1) \text{ nếu } A_i > 0$$

$$B[i] = (-2 * A[i] + 1) \text{ nếu } A_i < 0$$

$$B[i] = 1 \text{ nếu } A[i] = 0$$

- (114)- Cho mảng một chiều nguyên a. Viết hàm tạo mảng b từ a sao cho  $b[i]$  = tổng các phần tử lân cận với  $A[i]$  trong a.

#### 1.9.1.2.2 Nhập / tách mảng

- (115)- Cho mảng số nguyên A có n phần tử. Tách A thành 2 mảng: B chứa các số lẻ, C chứa các số chẵn.

Ví dụ: Mảng A : 1 3 **8** 2 7 5 9 0 **10**

Mảng B : 1 3 7 5 9

Mảng C : 8 2 10

- (116)- Có hai mảng một chiều A, B. Hai mảng này đã được sắp xếp tăng dần. Hãy viết chương trình trộn hai mảng A và B lại để được mảng C có các phần tử tăng dần.

- (117)- Cho 2 mảng số nguyên a và b kích thước lần lượt là n và m. Viết chương trình nối 2 mảng trên thành mảng c theo nguyên tắc chẵn ở đầu mảng và lẻ ở cuối mảng.

Ví dụ: Mảng a : 3 2 7 5 9

Mảng b : 1 8 10 4 12 6

Mảng c : **2 8 10 4 12 6** 3 7 5 9 1

- (118)- Cho 2 mảng số nguyên A và B kích thước lần lượt là n và m. Viết chương trình thực hiện:

a. Sắp xếp hai mảng A, B theo thứ tự tăng dần.

b. Trộn xen kẽ 2 mảng trên thành mảng c sao cho mảng c cũng có thứ tự tăng dần.

- c. Sắp xếp lại để có mảng B giảm dần (mảng A vẫn tăng dần). Trộn 2 mảng A và B thành mảng C tăng dần.

#### 1.9.1.2.3 Đếm – liệt kê

- (119)- Cho 2 mảng A, B. Đếm và liệt kê các phần tử chỉ xuất hiện 1 trong 2 mảng.  
 (120)- Cho 2 mảng A, B. Đếm và liệt kê các phần tử chỉ xuất hiện trong mảng A nhưng không xuất hiện trong mảng B.  
 (121)- Cho 2 mảng A, B. Đếm phần tử xuất hiện trong cả 2 mảng.

#### 1.9.1.2.4 Khác

- (122)- Viết chương trình nhập vào 1 số K (với  $-999 \leq K \leq 999$ ). In ra cách đọc chữ số tương ứng (sử dụng mảng).

Ví dụ: nhập -132 in ra: Am mot tram ba muoi hai.

- (123)- Cho nhập số nguyên dương n gồm tối đa 9 chữ số. In ra số lần xuất hiện của mỗi số

Ví dụ: với  $n=12712851$ . Sẽ xuất ra màn hình: số 1 xuất hiện 3 lần

số 2 xuất hiện 2 lần

số 5 xuất hiện 1 lần

số 7 xuất hiện 1 lần

số 8 xuất hiện 1 lần

- (124)- (\*\*) Viết chương trình tách 1 mảng các số nguyên thành 2 mảng A và B. Không dùng sắp xếp, thực hiện sao cho kết quả thu được là:

- Mảng A chứa toàn số lẻ tăng dần.
- Mảng B chứa toàn số chẵn giảm dần.

Hướng dẫn: Tìm vị trí chèn thích hợp khi trích phần tử từ mảng ban đầu sang 2 mảng A và B.

Ví dụ: Mảng ban đầu : 9 3 **8 2** 7 5 1 0 **10**

Mảng A : 1 3 5 7 9

Mảng B : 10 8 2

- (125)- (\*) Cho mảng C có n phần tử ( $n < 200$ ), các phần tử là các chữ số trong hệ đếm cơ số 16 (Hexa) (điều kiện mỗi phần tử  $\leq n$ ). Hãy tách mảng C ra các mảng con theo điều kiện sau: các mảng con được giới hạn bởi hai lần xuất hiện của cùng 1 con số trong mảng.

Ví dụ1: **123A4518B23**  $\Rightarrow$  có các mảng con là 123A451, 23A4518B2, 3A4518B23

Ví dụ2: **123A4538B21**  $\Rightarrow$  có các mảng con là 123A4538B21, 23A4518B2, 3A453

Ví dụ3: 123456789  $\Rightarrow$  in ra “Khong co day con”.

- (126)- (\*\*) Cho hai số nguyên dương A, B. Hãy xác định hai số C, D tạo thành từ hai số A, B sao cho C là số lớn nhất, D là số nhỏ nhất. Khi gạch đi một số chữ số trong C (D), thì các số còn lại giữ nguyên tạo thành A, các chữ số bỏ đi giữ nguyên tạo thành B.

Ví dụ:  $A = 52568, B = 462384 \rightarrow C = 54625682384, D = 45256236884$ .

- (127)- Đếm số lần xuất hiện của giá trị lớn nhất trong mảng một chiều.

Ví dụ: Mảng: 5 6 11 4 4 5 4



*So lon nhất la 11.*

*So 11 xuất hiện 1 lần*

✎ Mở rộng:

- Đếm số lần xuất hiện của giá trị nhỏ nhất, dương nhỏ nhất, âm lớn nhất.
- Đếm số lần xuất hiện của số nguyên tố nhỏ nhất, ...).

(128)- Đếm số lượng các giá trị phân biệt có trong mảng

✎ Mở rộng: Liệt kê tần suất xuất hiện các giá trị xuất hiện trong mảng (mỗi giá trị xuất hiện bao nhiêu lần).

(129)- Liệt kê các giá trị xuất hiện trong mảng một chiều nguyên đúng 1 lần.

✎ Mở rộng:

- Liệt kê các giá trị xuất hiện trong mảng một chiều nguyên đúng k lần, nhiều hơn 1 lần
- Liệt kê các giá trị có số lần xuất hiện nhiều nhất trong mảng.

(130)- Tìm các số nguyên tố nhỏ hơn 1000 bằng giải thuật sàng Erastosthene (giải thuật sàng Erastosthene dùng phương pháp đánh dấu để loại bỏ những số không phải là số nguyên tố. Giải thuật có từ một nhận xét rằng nếu k là số nguyên tố thì các số  $2*k$ ,  $3*k$ ,...  $n*k$  sẽ không là số nguyên tố (vì đã vi phạm định nghĩa về số nguyên tố).

(131)- Viết chương trình nhập vào một mảng số a gồm n số thực ( $n \leq 100$ ), nhập vào mảng số b gồm m số thực ( $m \leq 100$ ). In ra những phần tử:

- Chỉ xuất hiện trong mảng a mà không xuất hiện trong mảng b.
- Xuất hiện ở cả hai mảng.
- Không xuất hiện ở cả 2 mảng
- Thực hiện lại yêu cầu (i) nhưng chỉ in mỗi giá trị thoả điều kiện 1 lần (do trên mảng có thể có các giá trị trùng nhau).

## 2 CHUỖI KÝ TỰ (String)

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

- Hiểu được khái niệm về kiểu dữ liệu ký tự và kiểu dữ liệu chuỗi cũng như ứng dụng của nó;
- Vận dụng kiến thức về mảng một chiều để xử lý chuỗi ký tự.
- Biết cách xử lý các chuỗi ký tự có và không có dùng các hàm có sẵn trong ngôn ngữ lập trình C.

### 2.1. Ký tự (character)

#### 2.1.1. Khái niệm

- Ký tự “in được” gồm :
  - 26 chữ thường ('a', 'b', 'c', ..., 'z'),
  - 26 chữ hoa ('A', 'B', 'C', ..., 'Z'),
  - 10 chữ số ('0', '1', '2', '3', ..., '9'),
  - Khoảng trắng, các ký tự: ! “ # \$ % & ‘ ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ { | } ~
- Các ký tự “không in được”: tab, lert (bell), newline, formfeed, ...
- Các ký tự “in được” đặc biệt: '\'', '\'', '\'
- Các ký tự “không in được” đặc biệt:
  - \n new line
  - \a bell
  - \0 null character
  - \b backspace
  - \t horizontal tab

#### 2.1.2. Nhập ký tự

##### (i). Hàm scanf

- Khai báo thư viện <stdio.h> khi sử dụng hàm
- Nhận chuỗi ký tự từ bàn phím. Đối với hàm scanf khi gặp phím space, tab, new line, Enter thì dừng, vì vậy chỉ dùng hàm scanf để nhập chuỗi không có khoảng trắng.
- Ví dụ : char ch; scanf(“%c”, &ch);

##### (ii). Hàm getch

- Nhận một ký tự từ bộ đệm bàn phím và không cho hiện ký tự này lên màn hình.
- **Cú pháp:** int getch (void)
- Hàm trả về ký tự nhận được. Ví dụ char ch; ch = getch ();
- Nếu ký tự có sẵn trong bộ đệm bàn phím thì hàm nhận một ký tự trong đó.
- Nếu bộ đệm rỗng thì máy tạm dừng cho đến khi ta gõ vào một ký tự. Ký tự gõ vào sẽ nhận được ngay, không cần phải gõ phím Enter và ký tự không được hiển thị lên màn hình.

##### (iii). Hàm getche

- Nhận một ký tự từ bộ đệm bàn phím và cho hiển thị ký tự này lên màn hình.

- **Cú pháp:** `int getche(void)`
- Ví dụ: `int ch = getche();`
- Hàm này có công dụng giống như hàm `getch`, nhưng cho hiển thị ký tự nhập vào lên màn hình.

### 2.1.3. Xuất ký tự

#### (i). Hàm *putch*

- Khai báo thư viện `<string.h>` khi sử dụng hàm
- Xuất một ký tự ra cửa sổ văn bản màn hình.
- **Cú pháp:** `int putch (int ch)`

Trong đó, đối số `ch` chứa ký tự cần hiển thị.

- Hàm trả về ký tự đã hiển thị và xuất ký tự `ch` lên cửa sổ văn bản màn hình. Ký tự sẽ được hiển thị theo màu xác định trong hàm `textcolor`.

#### (ii). Hàm *printf*

- Khai báo thư viện `<string.h>` khi sử dụng hàm
- **Cú pháp:** `printf ("%c", ch)`
- Hàm có công dụng xuất ký tự `ch` lên cửa sổ văn bản màn hình.

#### (iii). Hàm *putc (ch)*

- Khai báo thư viện `<string.h>` khi sử dụng hàm
- Hàm có công dụng xuất ký tự `ch` lên cửa sổ văn bản màn hình.

## 2.2. Chuỗi

### 2.2.1. Khái niệm

- Chuỗi là một dãy ký tự dùng để lưu trữ và xử lý văn bản như từ, tên, 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.

### 2.2.2. Cách khai báo chuỗi

#### 2.2.2.1. Khai báo chuỗi

- **Cú pháp:** `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`)

#### 2.2.2.2. Vừa khai báo vừa gán giá trị

- **Cú pháp:** `char <Biến> []=<"Hằng chuỗi">`
- Ví dụ:  

```
char chuoi [] = "Lap Trinh C"
char s [50]= "TP.Ho Chi Minh";
char name []= {'Q','u','a','n',' ',' ','3','\0'};
```

```
char ten[10]={ 'S','a','i',' ','G','o','n','\0' };
```

khi đó:

	0	1	2	3	4	5	6	7	8	9	10	11	
chuoi	'L'	'a'	'p'	' '	'T'	'r'	'i'	'n'	'h'	' '	'C'	'\0'	

s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	'T'	'p'	'.'	' '	'H'	'o'	' '	'C'	'h'	'i'	' '	'M'	'i'	'n'	'h'	'\0'

name	0	1	2	3	4	5	6
	'Q'	'u'	'a'	'n'	' '	'3'	'\0'

Ten	0	1	2	3	4	5	6	7
	'S'	'a'	'i'	' '	'G'	'o'	'n'	'\0'

### 2.2.3. Lỗi thường gặp khi tạo một chuỗi

- (i). Không sử dụng toán tử gán = để chép nội dung của một chuỗi này sang chuỗi khác.

```
char a[4]="hi";
char b[4];
b = a; // Báo lỗi khi dịch chương trình
```

- (ii). Không dùng toán tử == để so sánh nội dung hai chuỗi

```
char a[]="hi"; char b[] = "there";
if(a==b) // Báo lỗi khi dịch chương trình
{ }
```

- (iii). Phép gán trong kiểu dữ liệu chuỗi như thế này là sai

```
char ten[10];
ten = "Sai Gon" // Báo lỗi khi dịch chương trình
```

### 2.2.4. Truy xuất từng ký tự trong chuỗi

- Do chuỗi là một mảng ký tự vì vậy ta có thể truy xuất chuỗi bằng chỉ số như truy xuất mảng.
- **Ví dụ 1:** Viết chương trình đếm số nguyên âm của chuỗi được nhập từ bàn phím.

```
void main()
{
    int dem=0,i=0;
    char s[100];
    printf("Nhap chuoi");
    gets(s);
    while (s[i]!='\0')
    {
        switch(s[i])
        {
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': dem++;
        }
        i++;
    }
    printf("So luong nguyen am vua nhap la: %d",dem);
}
```

- **Ví dụ 2:** Viết chương trình cho nhập một chuỗi. Copy nội dung chuỗi thứ nhất, đổi thành chữ hoa rồi đưa vào chuỗi thứ hai.

```
void main()
{
    int dem=0,i=0;
    char s1[100],s2[100];
    printf("Nhap chuoi");
    gets(s1);
    i=0;
    while (s1[i] != '\0')
    {
        s2[i]=toupper(s1[i]);
        i++;
    }
    s2[i] = '\0'; // Dat dau ket thuc chuoi cho S2
    printf("Chuoi S2: %s",s2);
}
```

- **Ví dụ 3:** Nhập vào một chuỗi ký tự, xuất ra màn hình chuỗi bị đảo ngược thứ tự các ký tự. Giả sử nhập vào: **Sai gon**. Xuất ra màn hình: **noG iaS**

```
#define MAX 100
void DaoChuoi(char s1[MAX], char s2[MAX])
{
    int i, l=strlen(s1);
    for(i=0; i<l; i++)
        s2[i]=s1[l-i-1];
    s2[i]='\0';
}
int main()
{
    char s1[MAX], s2[MAX];
    printf("Nhap vao chuoi ky tu: ");
    gets(s1);
    DaoChuoi(s1, s2);
    printf("\nKet qua sau khi dao nguoc chuoi: %s",s2);
    return 0;
}
```

## 2.3. Các hàm hỗ trợ thao tác trên chuỗi ký tự

### 2.3.1. Thư viện <stdio.h>

#### 2.3.1.1. Hàm scanf (nhập chuỗi)

- Hàm thực hiện nhận chuỗi ký tự được nhập từ bàn phím.
- Lưu ý: do 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.
- Ví dụ: `scanf ("%s" , & Hoten);`

#### 2.3.1.2. Hàm printf (Xuất Chuỗi)

- Hàm thực hiện xuất chuỗi ra màn hình.
- Ví dụ:

```
#include <stdio.h>
#include <string.h>
void main (void)
{
    char name[20];
    printf ("Enter a name: "); // printf (xuất chuỗi, không xuống dòng)
    scanf ("%s", name); // không sử dụng & trước tên biến name
    printf ("Hello %s\n", name); // printf (xuất chuỗi kèm xuống dòng)
}
```

### 2.3.2. Thư viện <string.h>

#### 2.3.2.1. Hàm clrscr ()

- Dùng để xóa màn hình.
- Cú pháp: void clrscr (void)

#### 2.3.2.2. Hàm gets

- Hàm thực hiện nhận chuỗi ký tự được nhập từ bàn phím.
- Ví dụ: gets (Hoten);
- Tiếp nhận được các phím 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().

#### 2.3.2.3. Hàm puts

- Hàm thực hiện xuất chuỗi xong tự động xuống dòng.
- Ví dụ: puts (Hoten);

#### 2.3.2.4. Hàm kbhit

- Kiểm tra bộ đệm bàn phím.
- Cú pháp: int kbhit (void)
- Hàm trả về giá trị khác không nếu bộ đệm bàn phím khác rỗng, trả về giá trị không nếu ngược lại.

#### 2.3.2.5. Hàm gotoxy()

- Dùng để di chuyển con trỏ (màn hình) đến vị trí (x,y).
- Trong đó x là số hiệu cột có giá trị từ 1 đến 80, và y là số hiệu dòng có giá trị từ 1 đến 25.
- Cú pháp: void gotoxy(int x, int y)

#### 2.3.2.6. 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"
void main()
{
    char *s1 = "Khoa ";
```

```

char *s2 = "CNTT";
strcat(s1, s2);
printf("%s",s1); // Kết quả: Khoa CNTT
}

```

#### 2.3.2.7. Hàm strncat

- Dùng để nối n ký tự đầu tiên của chuỗi s2 vào chuỗi s1.
- Cú pháp: char \* strncat(char\* s1, char\* s2, int n)
- Ví dụ:

```

#include "stdio.h"
#include "string.h"
void main()
{
    char *s1 = "Khoa ";
    char *s2 = "CNTT";
    strncat(s1, s2, 2);
    printf("%s",s1);    // Kết quả: Khoa CN
}

```

#### 2.3.2.8. Hàm strchr

- Tìm lần xuất hiện đầu tiên của ký tự c trong chuỗi s.
- Cú pháp: char\* strchr (char\* s, char c)
- Nếu tìm thấy hàm trả về vị trí tìm thấy của ký tự c trong chuỗi s, trái lại hàm trả về giá trị NULL.
- Ví dụ:

```

#include "stdio.h"
#include "string.h"
void main()
{
    char s[15];
    char *ptr, c = 'm';
    strcpy(s, "Vi du tim ky tu");
    ptr = strchr(s, c);
    if (ptr)
        printf("Ky tu %c xuất hiện tại vị trí %d",c, ptr-s);
    else
        printf("Không tìm thấy");
    //Kết quả: Ky tu m xuất hiện tại vị trí 8
}

```

#### 2.3.2.9. Hàm strcmp

- 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 khi chuỗi s1 nhỏ hơn chuỗi s2.
  - Nếu hàm trả về giá trị =0 khi chuỗi s1 bằng chuỗi s2.
  - Nếu hàm trả về giá trị >0 khi chuỗi s1 lớn hơn chuỗi s2.
- Ví dụ:

```

#include "stdio.h"
#include "string.h"

```

```

void main()
{
    char *s1 = "abcd";
    char *s2 = "abCD";
    if(strcmp(s1, s2)==0)
        printf("Giống nhau");
    else
        printf("Khác nhau"); // Kết quả: Khác nhau
}

```

#### 2.3.2.10. Hàm strcmp

- Hàm này thực hiện việc so sánh trong n ký tự đầu tiên của 2 chuỗi s1 và s2. Không phân biệt giữa chữ thường và chữ hoa.
- Kết quả trả về tương tự như kết quả trả về của hàm strcmp()
- Cú pháp: int strcmp (const char \*s1, const char \*s2)
- Ví dụ:

```

#include "stdio.h"
#include "string.h"
void main()
{
    char *s1 = "aBcd";
    char *s2 = "Abef";
    if(strnicmp(s1, s2, 2)==0)
        printf("Giống nhau");
    else
        printf("Khác nhau");
    //Kết quả: Giống nhau
}

```

#### 2.3.2.11. Hàm strncmp

- Tương tự như hàm strcmp(), nhưng chỉ so sánh n ký tự đầu tiên của 2 chuỗi s1 và s2.
- Cú pháp: int strcmp (const char \*s1, const char \*s2)
- Ví dụ:

```

#include "stdio.h"
#include "string.h"
void main()
{
    char *s1 = "abcd";
    char *s2 = "abef";
    if(strncmp(s1, s2, 2)==0)
        printf("Giống nhau");
    else
        printf("Khác nhau");
    //Kết quả: Giống nhau
}

```

#### 2.3.2.12. Hàm strcpy

- Hàm được dùng để sao chép toàn bộ nội dung của chuỗi nguồn (Src) vào chuỗi đích (Des).
  - Cú pháp: char \*strcpy (char \*Des, const char \*Src)
  - Ví dụ:
- ```

#include "stdio.h"

```



```
#include "string.h"
void main()
{
    char s[50];
    strcpy(s,"Truong Dai hoc Quoc Te Hong Bang");
    printf("\nXuat chuoi : %s",s);
}
```

### 2.3.2.13. Hàm strncpy

- Hàm này cho phép chép n ký tự đầu tiên của chuỗi nguồn (Src) sang chuỗi đích (Des).
- Cú pháp: char \*strncpy(char \*Des, const char \*Source, size\_t n)
- Ví dụ:

```
char dest[4];
char *src = "abcdefghi";
strncpy(dest, src, 3);
printf("%s",dest); //Kết quả: abc
```

### 2.3.2.14. Hàm strlen

- Lấy chiều dài chuỗi với hàm.
- Cú pháp: int strlen (const char \*s);
- Ví dụ 1:

```
#include <stdio.h>
#include <string.h>
void main()
{
    char string [ ] = "Sai gon";
    printf("%d\n", strlen(string)); // kết quả 7
    getch ();
}

- Ví dụ 2: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.
```

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Chuoi [255];
    int Dodai;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    Dodai = strlen(Chuoi)
    printf("Chuoi vua nhap: ");
    puts(Chuoi);
    printf("Co do dai %d",Dodai);
    getch();
    return 0;
}
```

- Ví dụ 3: Gán một chuỗi vào chuỗi khác bằng cách gán từng ký tự trong chuỗi.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    clrscr() ;
    char newstr [35];
    char str[] = "Viet Nam";
    for(int i = 0; i<strlen(str); i++)
        newstr[i] = str[i];
    newstr[i] = '\0';
    getch () ;
}
```

### 2.3.2.15. 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 "dai".

```
#include<stdio.h>
#include<string.h>
int main()
{
    char *s1 = "Borland International";
    char *s2 = "nation", *ptr;
    ptr = strstr(s1, s2);
    printf("Chuoi con: %s", ptr);
    //Kết quả: Chuoi con: national
}
```

### 2.3.2.16. Hàmstrupr()

- Hàmstrupr() 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 hàmstrupr() để chuyển đổi chúng thành chuỗi chữ hoa.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
    char Chuoi[255],*s;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    s=strupr(Chuoi) ;
    printf("Chuoi chu hoa: ");
    puts(s);
}
```

```
    getch();
}
```

### 2.3.2.17. Hàm `strlwr()`

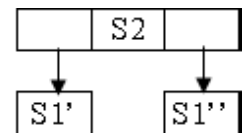
- Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàm `strlwr()`, các tham số của hàm tương tự như hàm `strupr()`
- Cú pháp: `char *strlwr (char *s)`

### 2.3.2.18. Hàm `atoi()`, `atof()`, `atol()`

- Để chuyển đổi chuỗi ra số, với kiểu dữ liệu tùy thuộc hàm được dùng.
- Cần khai báo thư viện `<stdlib.h>` khi sử dụng hàm.
- 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.

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

- Nếu `s2` có xuất hiện trong `s1`: Tách chuỗi `s1` thành hai chuỗi: `s1'`. Chuỗi đầu là những ký tự cho đến khi gặp chuỗi `s2` đầu tiên, chuỗi sau là những ký tự còn lại của `s1` sau khi đã bỏ đi chuỗi `s2` xuất hiện trong `s1`.
- Nếu `s2` không xuất hiện trong `s1` thì kết quả chuỗi tách vẫn là `s1`.
- Ví dụ :



```
#include <string.h>
#include<stdio.h>
void main()
{
    char input[16] = "abc,d";
    char *p;
    // Lay chuoai dau
    p = strtok(input, ",");
    if (p)
        printf("S11: %s-",p);
    /*Lay chuoai con lai, tham so dau la NULL*/
    p = strtok(NULL, ",");
    if (p)
        printf("S12: %s", p);
    // Kết quả: S11: abc - S12: d
}
```

### 2.3.2.20. `char* strrev(char *s)`

- Đảo ngược chuỗi.

## 2.3.3. Thư viện `<ctype.h>`

### 2.3.3.1. Hàm `toupper()`

- Hàm được dùng để chuyển đổi một ký tự thường thành ký tự hoa.
- Cú pháp: `char toupper (char c)`

### 2.3.3.2. Hàm `tolower()`

- Hàm được dùng để chuyển đổi một ký tự hoa thành ký tự thường.
- Cú pháp: `char tolower (char c)`

### 2.3.3.3. Hàm `isalpha`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một chữ cái hay không?
- Cú pháp: `int isalpha (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một chữ cái hoa
- Ví dụ: `int kq= isupper ('a');`

### 2.3.3.4. Hàm `isupper`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một chữ cái hoa hay không?
- Cú pháp: `int isupper (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một chữ cái
- Ví dụ: `int kq= isupper ('a');`

### 2.3.3.5. Hàm `islower`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một chữ cái thường hay không?
- Cú pháp: `int isupper (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một chữ cái thường
- Ví dụ: `int kq= islower ('a');`

### 2.3.3.6. Hàm `isdigit`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một ký số hay không?
- Cú pháp: `int isdigit (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một ký số
- Ví dụ: `int kq= isdigit ('a');`

### 2.3.3.7. Hàm `isascii`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một ký tự có mã ASCII < 128 hay không?
- Cú pháp: `int isascii(char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một ký tự có mã ASCII < 128.
- Ví dụ: `int kq= isascii ('a');`

### 2.3.3.8. Hàm `isspace`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một khoảng trắng hay không?
- Cú pháp: `int isspace (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là ký tự khoảng trắng
- Ví dụ: `int kq= isspace ('a');`

### 2.3.3.9. Hàm `isprint`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một ký tự có thể in được hay không?
- Cú pháp: `int isprint (char_exp)`
- Kết quả trả về:  $\leq 0$  khi `char_exp` là một ký tự có thể in được.
- Ví dụ: `int kq= isprint ('a');`

### 2.3.3.10. Hàm `isctrl`

- Hàm thực hiện kiểm tra xem `char_exp` có phải là một ký tự điều khiển hay không?
- Cú pháp: `int isctrl (char_exp)`
- Kết quả trả về:  $<0$  khi `char_exp` là một ký tự có thể in được.
- Ví dụ: 

```
int kq= isctrl ('!');
```

## 2.3.4. Thư viện `<stdlib.h>`

### 2.3.4.1. Hàm `atoi`

- Hàm thực hiện chuyển một chuỗi sang số nguyên. Việc chuyển đổi sẽ dừng khi gặp ký tự không phải là ký số
- Cú pháp: `int atoi(str_exp)`
- Ví dụ: 

```
int kq= atoi('123a45');
```

### 2.3.4.2. Hàm `atof`

- Hàm thực hiện chuyển một chuỗi sang số double. Việc chuyển đổi sẽ dừng khi gặp ký tự không thể chuyển sang dạng double được
- Cú pháp: `double atof(str_exp)`
- Ví dụ: 

```
double kq= atof('123.45');
```

### 2.3.4.3. Hàm `itoa`

- Hàm thực hiện chuyển giá trị số nguyên sang dạng chuỗi và gán vào vùng nhớ mà con trỏ `st` đang trỏ đến. `st` là một con trỏ kiểu ký tự.
- Cú pháp: `char* itoa(int value, char *st, int radix)`
- Ví dụ:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main()
{
    int a=54325;
    char buffer[20];
    itoa(a,buffer,2); // chuyển sang giá trị theo cơ số 2 (binary)
    printf("Binary value = %s\n", buffer); // Binary value = 1101010000110101

    itoa(a,buffer,10); // chuyển sang giá trị theo cơ số 10 (decimal)
    printf("Decimal value = %s\n", buffer); // Decimal value = 54325

    itoa(a,buffer,16); //chuyển sang giá trị theo cơ số 16 (Hexadecimal)
    printf("Hexadecimal value = %s\n", buffer); //Hexadecimal value = D435
}
```

## 2.4. Phần thực hành

### 2.4.1. Mục tiêu

Sinh viên phải biết cách nhập, xuất, lưu trữ trên mảng một chiều với các dữ liệu kiểu ký tự, xử lý các bài toán tìm kiếm, so sánh, cắt chuỗi, ...

### 2.4.2. Bài thực hành

- (1)- Nhập chuỗi
- (2)- Xuất chuỗi
- (3)- Nhập vào 2 chuỗi, xuất chuỗi theo thứ tự từ điển
- (4)- Đếm số ký tự 'a' có trong chuỗi
- (5)- Cắt khoảng trắng có trong chuỗi
- (6)- Đếm khoảng trắng trong chuỗi .
- (7)- Đếm số từ có trong chuỗi
- (8)- Sắp xếp chuỗi tăng dần
- (9)- Nhập 3 chuỗi, xuất chuỗi theo thứ tự từ điển.

### 2.5. Bài tập (sinh viên tự thực hiện)

- (10)- Viết hàm nhập vào một mảng một chiều các số nguyên gồm n phần tử ( $0 < n < 100$ ).
- (11)- Viết hàm nhập vào một mảng một chiều các số thực gồm n phần tử ( $0 < n < 100$ ).
- (12)- Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên .
- (13)- Viết hàm xuất mảng số thực n phần tử vừa nhập ở trên .
- (14)- Tính tổng các phần tử có trong mảng .
- (15)- Tính tổng các phần tử chẵn có trong mảng .
- (16)- Tính tổng các phần tử lẻ có trong mảng .
- (17)- Tính tổng các phần tử nguyên tố có trong mảng .
- (18)- Tìm phần tử chẵn đầu tiên có trong mảng
- (19)- Tìm phần tử lẻ đầu tiên có trong mảng
- (20)- Tìm phần tử nguyên tố đầu tiên có trong mảng
- (21)- Tìm phần tử chẵn cuối cùng có trong mảng
- (22)- Tìm phần tử chính phương cuối cùng có trong mảng
- (23)- Tìm phần tử lớn nhất có trong mảng
- (24)- Đếm số phần tử chẵn có trong mảng
- (25)- Đếm số phần tử có giá trị là x có trong mảng
- (26)- Đếm số phần tử lớn nhất có trong mảng
- (27)- In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (28)- In ra vị trí của phần tử có giá trị là x cuối cùng có trong mảng
- (29)- Thêm một phần tử vào đầu mảng.
- (30)- Thêm một phần tử vào cuối mảng.

- (31)- Thêm một phần tử vào vị trí x trong mảng.
  - (32)- Xóa phần tử chẵn đầu tiên.
  - (33)- Xóa tất cả các phần tử lớn nhất trong mảng
  - (34)- Sắp xếp mảng tăng dần
  - (35)- Thêm một phần tử có giá trị là x vào trong mảng sao cho mảng vẫn có thứ tự tăng dần.
  - (36)- Đảo ngược mảng
  - (37)- Nhập chuỗi
  - (38)- Xuất chuỗi
  - (39)- Đếm số ký tự ' a ' có trong chuỗi
  - (40)- Cắt khoảng trắng có trong chuỗi
  - (41)- Đếm khoảng trắng trong chuỗi .
  - (42)- Đếm số từ có trong chuỗi
  - (43)- Sắp xếp chuỗi tăng dần
  - (44)- Nhập 3 chuỗi, xuất chuỗi theo thứ tự từ điển.
-

## 3 MẢNG HAI CHIỀU

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

- Hiểu được các khái niệm cơ bản về Nhập, Xuất dữ liệu trên mảng 2 chiều;
- Biết cách khai báo biến kiểu mảng và các phép toán trên các phần tử của mảng;
- Đi sâu vào các giải thuật trên mảng 2 chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...

### 3.1. Khái niệm

Mảng (array) là một dãy liên tiếp các phần tử, mỗi phần tử chứa dữ liệu có cùng một kiểu dữ liệu, kiểu dữ liệu đó gọi là kiểu cơ sở của mảng.

- Kích thước của mảng là số phần tử của mảng. Kích thước này phải được biết ngay khi khai báo mảng.
- Mảng nhiều chiều là mảng có từ hai chiều trở lên. Điều đó có nghĩa là mỗi phần tử của mảng là một mảng khác.
- Người ta thường sử dụng mảng nhiều chiều để lưu các ma trận, các tọa độ 2 chiều, 3 chiều, ...
- Phần dưới đây ta chỉ nghiên cứu các vấn đề liên quan đến mảng 2 chiều; các mảng 3, 4, ... chiều thì tương tự (chỉ cần tổng quát hóa lên).
- Mảng hai chiều còn gọi là ma trận gồm m dòng và n cột.

### 3.2. Cách khai báo mảng hai chiều

#### 3.2.1. Cú pháp

##### 3.2.1.1. Chỉ thuần khai báo và không gán giá trị

**<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 khác đi nó là kích thước của mảng). Số phần tử của mảng chính bằng số dòng nhân số cột .5.
  - **Kiểu cơ sở:** là kiểu dữ liệu của mỗi phần tử của mảng.
- Ví dụ
  - Khai báo mảng hai chiều có tên là sn gồm 8 hàng, 14 cột và kiểu cơ sở là int  
`int sn [8] [14];`
  - Khai báo mảng hai chiều có tên là st gồm 10 hàng, 5 cột và kiểu cơ sở là float  
`float st [10] [5];`
  - Khai báo mảng hai chiều có tên là str gồm 12 hàng, 30 cột và kiểu cơ sở là char  
`char str[12] [30];`



**3.2.1.2. Chỉ thuần khai báo đồng thời gán giá trị**

- Ví dụ: `int A[3][4] = { {2,3,9,4} , {5,6,7,6} , {2,9,4,7} };`

Với khai báo và gán giá trị ở trên, ma trận sẽ có hình dạng như sau:

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2 | 3 | 9 | 4 |
| 1 | 5 | 6 | 7 | 6 |
| 2 | 2 | 9 | 4 | 7 |

**3.2.2. 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 dòng và chỉ số cột nằm trong cặp dấu ngoặc vuông [ ][ ].
- Chẳng hạn `a[0][0]` là phần tử đầu tiên của mảng `a` được khai báo ở trên, phần tử này nằm trên dòng 0 cột 0 của mảng hai chiều.
- 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] [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 trên các phần tử của mảng.
- Ví dụ : Cho mảng `a[4][8]`. Khi đó, mảng gồm  $4 \times 8 = 32$  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 | <code>[0][0]</code> | <code>[0][1]</code> | <code>[0][2]</code> | <code>[0][3]</code> | <code>[0][4]</code> | <code>[0][5]</code> | <code>[0][6]</code> | <code>[0][7]</code> |
| 1 | <code>[1][0]</code> | <code>[1][1]</code> | <code>[1][2]</code> | <code>[1][3]</code> | <code>[1][4]</code> | <code>[1][5]</code> | <code>[1][6]</code> | <code>[1][7]</code> |
| 2 | <code>[2][0]</code> | <code>[2][1]</code> | <code>[2][2]</code> | <code>[2][3]</code> | <code>[2][4]</code> | <code>[2][5]</code> | <code>[2][6]</code> | <code>[2][7]</code> |
| 3 | <code>[3][0]</code> | <code>[3][1]</code> | <code>[3][2]</code> | <code>[3][3]</code> | <code>[3][4]</code> | <code>[3][5]</code> | <code>[3][6]</code> | <code>[3][7]</code> |

**3.3. Nhập dữ liệu cho mảng hai chiều****3.3.1. Nhập dữ liệu cho mảng hai chiều các số nguyên****3.3.1.1. Khai báo hằng**

```
#define m 30 // m là số dòng
#define n 30 // n là số cột
```

**3.3.1.2. Khai báo mảng 2 chiều có kiểu số nguyên**

```
int a [m][n]
```

Hình ảnh mảng `a` gồm `m` dòng `n` cột lưu trữ các số nguyên được biểu diễn như sau

|       | 0  | 1  | 2  | 3  | 4  | 5  | ... | $n-1$ |
|-------|----|----|----|----|----|----|-----|-------|
| 0     | 1  | 2  | 3  | 4  | 5  | 6  | ... | 7     |
| 1     | 10 | 11 | 12 | 13 | 14 | 15 | ... | 17    |
| ...   |    |    |    |    |    |    |     |       |
| $m-1$ | 20 | 21 | 22 | 23 | 24 | 25 | ... | $m*n$ |

- Nhập dữ liệu cho dòng thứ nhất

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

```
{   printf ("Nhap so nguyen: ");
    scanf ("%d" , &a[0][col]) ;
}
```

- Nhập dữ liệu cho dòng thứ hai

```
for (int col=0 ; col<n ; col++)
{   printf ("Nhap so nguyen: ");
    scanf ("%d" , &a[1][col]) ;
}
```

...

- Nhập dữ liệu cho dòng thứ m

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

- Nhập dữ liệu cho ma trận gồm m dòng và n cột

```
void NhapMangSoNguyen (int A[][MAX], int &d, int &c)
{
    for (int row=0 ; row<m ; row++)
        for (int col=0 ; col<n ; col++)
        {
            printf ("Nhap so nguyen: ");
            scanf ("%d" , &a[row][col]) ;
        }
}
```

### 3.3.2. Nhập dữ liệu cho mảng hai chiều các số thực

#### 3.3.2.1. Khai báo hằng

```
#define m 30 // m là số dòng
#define n 30 // n là số cột
```

#### 3.3.2.2. Khai báo mảng 2 chiều có kiểu số nguyên

```
float a [m][n]
```

Hình ảnh mảng a gồm m dòng n cột lưu trữ các số nguyên được biểu diễn như sau

|     | 0  | 1  | 2  | 3  | 4  | 5  | ... | n-1 |
|-----|----|----|----|----|----|----|-----|-----|
| 0   | 1  | 2  | 3  | 4  | 5  | 6  | ... | 7   |
| 1   | 10 | 11 | 12 | 13 | 14 | 15 | ... | 17  |
| ... |    |    |    |    |    |    |     |     |
| m-1 | 20 | 21 | 22 | 23 | 24 | 25 | ... | m*n |

- Nhập dữ liệu cho dòng thứ nhất

```
for (int col=0 ; col<n ; col++)
{   printf ("Nhap so thuc: ");
    scanf ("%f" , &a[0][col]) ;
}
```

- Nhập dữ liệu cho dòng thứ hai

```
for (int col=0 ; col<n ; col++)
{   printf ("Nhap so thuc: ");
    scanf ("%f" , &a[1][col]) ;
}
```

...

- Nhập dữ liệu cho dòng thứ m

```
for (int col=0 ; col<n ; col++)
{   printf ("Nhap so thuc: ");
    scanf ("%f" , &a[m][col]) ;
}
```

- Tổng quát việc nhập dữ liệu cho ma trận gồm m dòng và n cột

```
void NhapMangSoThuc (float A[][MAX], int &d, int &c)
{
    for (int row=0 ; row<m ; row++)
        for (int col=0 ; col<n ; col++)
        {
            printf ("Nhap so thuc: ");
            scanf ("%f" , &A[row][col]) ;
        }
}
```

### 3.4. Xuất dữ liệu cho mảng hai chiều

#### 3.4.1. Xuất dữ liệu cho mảng hai chiều các số nguyên

Hình ảnh mảng a gồm m dòng n cột lưu trữ các số nguyên được biểu diễn như sau:

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

- Xuất dữ liệu cho dòng thứ nhất

```
for (int j=0 ; j< n ; j++)
    printf (" %4d " , a[0][j]) ;
```

- Xuất dữ liệu cho dòng thứ hai

```
for (int j=0 ; j< n ; j++)
    printf (" %4d " , a[1][j]) ;
```

- ...

- Xuất dữ liệu cho dòng thứ m

```
for (int j=0 ; j< n ; j++)
    printf (" %4d " , a[m-1][j]) ;
```

- Tổng quát, xuất dữ liệu cho từng phần tử của ma trận gồm m dòng n cột lưu trữ các số nguyên

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

### 3.4.2. Xuất dữ liệu cho mảng hai chiều các số thực

Hình ảnh mảng a gồm m dòng n cột lưu trữ các số thực được biểu diễn như sau:

|     | 0   | 1   | 2   | 3   | 4   | 5   | ... | n-1 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0   | 1.1 | 2.1 | 3.2 | 4.3 | 5.0 | 6.0 | ... | 7.1 |
| 1   | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | ... | 1.7 |
| ... |     |     |     |     |     |     |     |     |
| m-1 | 2.0 | 2.1 | 2.2 | 2.3 | 2.4 | 2.5 | ... | 2.7 |

- Xuất dữ liệu cho dòng thứ nhất

```
for (int j=0 ; j< n ; j++)
    printf (" %.1f " , a [0][j]);
```

- Xuất dữ liệu cho dòng thứ hai

```
for (int j=0 ; j< n ; j++)
    printf (" %.1f " , a [1][j]);
```

- ...

- Xuất dữ liệu cho dòng thứ m

```
for (int j=0 ; j< n ; j++)
    printf (" %.1f " , a [m-1][j]);
```

- Tổng quát, xuất dữ liệu cho từng phần tử của ma trận gồm m dòng n cột lưu trữ các số thực

```
void XuatMangSoNguyen(int a[][MAX], int d, int c)
{
    for (int i =0 ; i<m ; i++)
    {
        for (int j=0; j< n ; j++)
            printf (" %.1f ", a[ i ][ j ]);
        printf (" \n");
    }
}
```

### 3.5. Ma trận vuông

#### 3.5.1. Khái niệm

Là ma trận có số dòng và số cột bằng nhau.

#### 3.5.2. Ma trận đơn vị

Ma trận đơn vị  $I_n$  là ma trận vuông kích thước  $(n \times n)$  có tất cả các phần tử nằm trên đường chéo chính bằng 1 và những phần tử còn lại bằng 0.

|   |   |   |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

#### 3.5.3. Tính chất của ma trận vuông

##### 3.5.3.1. Đường chéo chính

- Đường chéo chính: là đường chéo nối phần tử góc trái trên đến phần tử ở góc phải dưới.
- Đường chéo chính là đường chéo có:

$$\text{chỉ số dòng} = \text{chỉ số cột}$$

- Ví dụ: Cho ma trận vuông  $A_{(n \times n)}$ . Xuất các giá trị có trên đường chéo chính
- ```
void XuatDuongCheoChinh(int a[][MAX], int n)
{
    for (int i=0 ; i<n ; i++)
        printf("%5d", a[i][i]);
}
```

##### 3.5.3.2. Đường chéo phụ

- Đường chéo phụ là đường chéo có:

$$\text{chỉ số cột} + \text{chỉ số dòng} = \text{số dòng (hoặc số cột)} - 1$$

- Ví dụ: Cho ma trận vuông  $A_{(n \times n)}$ . Xuất các giá trị có trên đường chéo phụ:
- ```
void XuatDuongCheoPhu(int a[][MAX], int n)
{
    for (int i=0; i<n ; i++)
        printf("%5d", a[i][n-i-1]);
}
```

##### 3.5.3.3. Đường chéo song song với đường chéo chính hoặc với đường chéo phụ

- Số lượng đường chéo song song với đường chéo chính hoặc song song với đường chéo phụ là:  $(2*n) - 1$

##### 3.5.3.3.1 Đường chéo song song với đường chéo chính

- Tính chất: Các phần tử cùng nằm trên 1 đường chéo sẽ có:

$$\text{chỉ số dòng} - \text{chỉ số cột} = \text{hằng số}$$

|   |   |   |   |   |
|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 |
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

##### 3.5.3.3.2 Đường chéo song song với đường chéo phụ

- Tính chất: Các phần tử cùng nằm trên 1 đường chéo sẽ có:

$$\text{chỉ số dòng} + \text{chỉ số cột} = \text{hằng số}$$

|   |   |   |   |   |
|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 |
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

### 3.5.3.4. Tam giác tạo thành từ các đường chéo chính

#### 3.5.3.4.1 Đối với đường chéo chính

- Tam giác trên:

- Tính chất: các phần tử thuộc tam giác trên của đường chéo chính sẽ có:

$$\text{Chỉ số cột} > \text{Chỉ số dòng}$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

- Tam giác dưới:

- Tính chất: các phần tử thuộc tam giác trên của đường chéo chính sẽ có:

$$\text{Chỉ số cột} < \text{Chỉ số dòng}$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

#### 3.5.3.4.2 Đối với đường chéo phụ

- Tam giác trên:

- Tính chất các phần tử thuộc tam giác trên của đường chéo phụ sẽ có:

$$\text{Chỉ số cột} + \text{Chỉ số dòng} < \text{số dòng (hoặc số cột)} - 1$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

- Tam giác dưới:

- Tính chất các phần tử thuộc tam giác dưới của đường chéo phụ sẽ có:

$$\text{Chỉ số cột} + \text{Chỉ số dòng} > \text{số dòng (hoặc số cột)} - 1$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   |   |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |

### 3.5.3.5. Phần tử đối xứng qua đường chéo chính

#### 3.5.3.5.1 Đối với đường chéo chính

- Tính chất Truy xuất các phần tử đối xứng qua đường chéo chính dựa vào tính chất:

$$A[\text{dòng}][\text{cột}] \text{ đối xứng với } A[\text{cột}][\text{dòng}]$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   |   | X |   |
| 1 |   |   |   |   |
| 2 | X |   |   |   |
| 3 |   |   |   |   |

#### 3.5.3.5.2 Đối với đường chéo phụ

- Tính chất Truy xuất các phần tử đối xứng qua đường chéo phụ dựa vào tính chất:

$$A[\text{dòng}][\text{cột}] \text{ đối xứng với } A[n-1-\text{cột}][n-1-\text{dòng}]$$

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 |   | X |   |   |
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   | X |   |

### 3.6. Khai báo kiểu ma trận

- Để đơn giản trong việc khai báo ma trận, ta định nghĩa kiểu ma trận các phần tử với kiểu dữ liệu bất kỳ như sau:

- Ma trận thường (số dòng và số cột khác nhau)

```
#define ROWS 100
#define COLS 200
typedef int MATRAN[ROWS][COLS];
```
- Ma trận vuông (số dòng và số cột bằng nhau)

```
#define SIZE 100
typedef int MATRAN[SIZE][SIZE];
```

- Ví dụ: Khai báo ma trận các số nguyên A.

```
#define SIZE 100
typedef int MATRAN[SIZE][SIZE];
MATRAN A;
```

### 3.7. Tham số của hàm là mảng hai chiều

- Nguyên tắc truyền và nhận dữ liệu giữa hàm gọi và hàm được gọi cho mảng hai chiều cũng giống như hàm một chiều nghĩa là hàm được gọi sẽ tác động trực tiếp lên mảng truyền cho nó. Tuy nhiên việc khai báo có khác nhau.
- Ví dụ: chương trình sau tạo mảng hai chiều với các giá trị là ngẫu nhiên rồi in ma trận lên màn hình.

```
#define ROWS 200
#define COLS 200
typedef int MATRAN[ROWS][COLS];
void TaoMangNgauNhien(MATRAN a, int dong, int cot);
void XuatMaTran(MATRAN a, int dong, int cot);
int main()
{
    srand((unsigned int) time(NULL));
    MATRAN A;
    int dong, cot;
    printf("Nhap so dong cua ma tran: ");
    scanf("%d",&dong);
    printf("Nhap so cot cua ma tran: ");
    scanf("%d",&cot);
    TaoMangNgauNhien (A, dong,cot);
    printf("\n Ma tran vua duoc tao:\n");
    XuatMaTran(A,dong,cot);
    return 0;
}
```

**//Cách 1: Khai báo kiểu MATRAN  $\Rightarrow$  sử dụng kiểu MATRAN cho tham số**  
**void TaoMaTranNgauNhien (MATRAN a, int dong, int cot)**

```
{
    int i,j;
    for (i=0; i<dong; i++)
        for (j=0;j<cot;j++)
            a[i][j]=rand()%100;
}
```

```

void XuatMaTran(MATRAN a, int dong, int cot)
{
    int i,j;
    for (i=0; i<dong; i++)
    {
        for(j=0;j<cot;j++)
            printf("%5d",a[i][j]);
        printf("\n");
    }
}

//Cách 2: không sử dụng kiểu MATRAN => không cần khai báo số dòng
void TaoMaTranNgauNhiên2 (int a[][COLS], int dong, int cot)
{
    int i,j;
    for (i=0; i<dong; i++)
        for (j=0;j<cot;j++)
            a[i][j]=rand()%100;
}

void XuatMaTran2 (int a[][COLS], int dong, int cot)
{
    int i,j;
    for (i=0; i<dong; i++)
    {
        for(j=0;j<cot;j++)
            printf("%5d",a[i][j]);
        printf("\n");
    }
}

```

### 3.8. Một số kỹ thuật cơ bản trên mảng hai chiều

Có thể sử dụng chung các kỹ thuật sau đây cho mảng 2 chiều vuông hoặc không vuông (ma trận). Để dùng cho ma trận vuông, ta thay biến d (dòng) và c (cột) bằng n (cạnh của ma trận vuông).

#### 3.8.1. Nhập / xuất ma trận

- Nhập trực tiếp các phần tử của ma trận:

```

void NhapMaTran (MATRAN a, int dong, int cot)
{
    int i,j;
    for (i=0; i<dong; i++)
        for (j=0;j<cot;j++)
        {
            printf("Nhap gia tri cho phan tu a[%d][%d]: ",i,j);
            scanf("%d",&a[i][j]);
        }
}

```

- Tạo giá trị ngẫu nhiên cho các phần tử của ma trận:

```

void TaoMangNgauNhiên (int a[][COLS], int dong, int cot)
{
    for (int i=0; i<dong; i++)
        for (int j=0;j<cot;j++)
            a[i][j]=rand()%100;
}

```



- Xuất các phần tử của ma trận:

```
void XuatMang(int a[][COLS], int dong, int cot)
{
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            printf("%5d",a[i][j]);
        printf("\n");
    }
}
```

### 3.8.2. Tính tổng các phần tử trong ma trận

#### 3.8.2.1. Tính tổng không có điều kiện

- Khai báo mảng a để lưu trữ các phần tử là các số nguyên : int a [100][100], khi đó máy sẽ cấp phát 20000 byte để lưu trữ mảng a
- Hình ảnh mảng a gồm m dòng n cột lưu trữ các số nguyên được biểu diễn như sau

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

- Tính tổng : khai báo một biến s để lưu trữ tổng

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

- Giải thuật

- Đi từ đầu ma trận đến cuối ma trận

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

- Cộng dồn các phần tử a[i][j] vào biến s

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

- Hàm cài đặt

```
long TinhTong (int a[ ][100] , 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;
}
```

- Viết hàm tính tổng các giá trị âm trong ma trận

```
long TongAm(int A[][COLS], int dong, int cot)
{
    long s=0;
    for (int i=0; i<dong; i++)
        for(int j=0; j<cot; j++)
            if (A[i][j]< 0)
                s += A[i][j];
    return s ;
}
```

**3.8.2.2. Tính tổng theo điều kiện cho trước**

- Tính tổng chẵn : khai báo một biến s để lưu trữ tổng
- Giải thuật

- Đi từ đầu ma trận đến cuối ma trận

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

- Kiểm tra a[i][j] chẵn thì cộng dồn các phần tử a[i][j] vào biến s

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

- Hàm cài đặt

```
long TinhTongChan (int a[ ][100] , int m, int n)
{
    long s =0 ;
    for (int i = 0 ; i<m ; i++)
        for (int j=0; j< n ; j++)
            if (a[i][j] % 2 == 0)
                s=s+ a[i][j] ;
    return s;
}
```

**3.8.3. Kỹ thuật đặt cờ hiệu**

- Viết hàm kiểm tra xem trong ma trận các số nguyên có tồn tại các số nguyên lẻ lớn hơn 100 không?

```
int KiemTraLe (int a[][COLS], int dong, int cot)
{
    int flag = 0; //tra ve 1 neu co nguoc lai tra ve 0
    for (int i = 0; i<dong; i ++ )
        for (int j = 0; j<cot; j++)
            if ( (a[i][j]%2!=0) && (a[i][j]>100) )
            {
                flag = 1;
                break;
            }
    return flag;
}
```

- Viết hàm kiểm tra trong ma trận có tồn tại giá trị chẵn nhỏ hơn 100 hay không?

```
int TonTaiChan(int a[][COLS], int dong, int cot)
{
    int flag = 0;
    for (int i=0; i<dong; i++)
        for(int j=0; j<cot; j++)
            if (a[i][j] %2 ==0 && a[i][j]< 100)
                flag = 1;
    return flag;
}
```

### 3.8.4. Kỹ thuật đặt lính canh

- Viết hàm tìm phần tử nhỏ nhất trong ma trận.

```
int Min (int a[][COLS], int d, int c )
{
    int min = a[0][0];
    for ( int i = 0 ; i < d ; i ++ )
        for (int j = 0 ; j < c ; j ++ )
            if ( a[i][j] < min )
                min = a[i][j];
    return min;
}
```

- Viết hàm tìm giá trị lớn nhất trong ma trận các số thực.

```
float LonNhat (int a[][COLS], int dong, int cot)
{
    float ln = a[0][0] ;
    for (int i=0; i<dong ; i++)
        for (int j=0; j<cot ; j++)
            if ( a[i][j] > ln )
                ln=a[i][j] ;
    return ln ;
}
```

### 3.8.5. Sắp xếp: Viết hàm sắp xếp ma trận tăng dần từ trên xuống dưới và từ trái sang phải

- Không dùng mảng phụ:

```
void SortTrucTiep(int A[][COLS], int dong, int cot)
{
    for(int k=0; k<=dong*cot-2; k++)
        for(int l=k+1; l<= dong*cot-1; l++)
            if (A[k/cot][k%cot]>A[l/cot][l%cot])
            {
                int temp = A[k/cot][k%cot];
                A[k/cot][k%cot] = A[l/cot][l%cot];
                A[l/cot][l%cot] = temp;
            }
}
```

- Có dùng mảng phụ:

```
void SapXep(float a[MAX][], int dong, int cot)
{
    float b[1000];
    int k,i,j;
    // copy ma trận ra mảng một chiều b
    k=0;
    for (i=0;i<dong;i++)
        for(j=0,j<cot;j++)
            b[k++] = a[i][j];
    // Sắp xếp mảng một chiều b
    for (i=0; i< k-1; i++)
        for(j=i+1;j<k;j++)
            if (b[i] > b[j])
            {
                float temp = b[i];
```

```

        b[i] = b[j];
        b[j] = temp;
    }
    // copy ma trận ra mảng một chiều b
    k=0;
    for (i=0;i<dong;i++)
        for(j=0,j<cot;j++)
            a[i][j]= b[k++] ;
}

```

### 3.8.6. Đếm

- Viết hàm đếm các phần tử chẵn trong ma trận.

```

int DemChan (MATRAN a, int dong, int cot)
{
    int dem = 0;
    for (int i = 0 ; i<dong ; i++)
        for (int j = 0 ; j<cot ; j++)
            if (a[i][j] % 2 == 0 )
                dem ++;
    return dem;
}

```

- Viết hàm đếm số lượng số nguyên tố trong ma trận các số nguyên?

```

int DemNguyenTo(int a[][100], int dong, int cot)
{
    int dem = 0;
    for (int i=0; i<dong; i++)
        for(int j=0; j<cot; j++)
            if (LaSNT(a[i][j])==1)
                dem ++ ;
    return dem;
}

```

## 3.9. Phần thực hành

### 3.9.1. Mục tiêu

- Sinh viên phải biết cách nhập, xuất, lưu trữ trên mảng hai chiều với các dữ liệu kiểu số, xử lý các bài toán như tính tổng các giá trị trên mảng hai chiều các số nguyên, số thực, tìm kiếm các phần tử nhỏ nhất, lớn nhất, trên mảng, trên dòng , xoá dòng, xoá cột, xoay ma trận
- Sinh viên phải biết cách nhập, xuất, lưu trữ trên mảng hai chiều với các dữ liệu kiểu ký tự, xử lý các bài toán tìm kiếm, so sánh, cắt chuỗi....

### 3.9.2. Bài thực hành

- 3.10.2\_1.- Viết hàm nhập vào số dòng, số cột ( $0 < m, n < 100$ ) .
- 3.10.2\_2.- Viết hàm nhập vào mảng hai chiều các số nguyên gồm m dòng, n cột ( $0 < m, n < 100$ ).
- 3.10.2\_3.- Viết hàm xuất mảng hai chiều các số nguyên vừa nhập ở trên.
- 3.10.2\_4.- Tính tổng các phần tử có trong mảng .
- 3.10.2\_5.- Tính tổng các phần tử chẵn có trong mảng .
- 3.10.2\_6.- Tính tổng các phần tử nguyên tố có trong mảng .
- 3.10.2\_7.- Tính tổng các phần tử nằm trên đường chéo chính có trong mảng .

**3.10.2\_8.-** Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng .

**3.10.2\_9.-** Tính tổng các phần tử nằm trên đường biên.

**3.10.2\_10.-** Tìm phần tử lớn nhất có trong mảng

**3.10.2\_11.-** Đếm số phần tử lẻ có trong mảng

**3.10.2\_12.-** Đếm số phần tử lớn nhất có trong mảng

**3.10.2\_13.-** In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng

**3.10.2\_14.-** Tính tổng các phần tử nằm trên một dòng .

**3.10.2\_15.-** Tìm dòng có tổng lớn nhất

**3.10.2\_16.-** Xoá dòng

**3.10.2\_17.-** Xoá cột

**3.10.2\_18.-** Sắp xếp mảng tăng dần

**3.10.2\_19.-** Xoay mảng về trái .

**3.10.2\_20.-** Xoay mảng về phải

### **3.9.3. Hướng dẫn**

- Viết hàm nhập số dòng, số cột của mảng ( $0 < n, m \leq 100$ ).
- Viết hàm nhập vào mảng gồm m dòng n cột
- Viết hàm xuất mảng số nguyên m x n phần tử vừa nhập ở trên .
- Viết hàm main () kết 3 hàm trên . chạy ổn định rồi mới viết tiếp hàm khác Viết hàm tính tổng các phần tử có trong mảng .
- Kết hàm tính tổng trong hàm main () ....
- Chú ý: Viết xong hàm nào kết ngay hàm đó trong hàm main()

### **3.10. Bài tập (sinh viên tự thực hiện)**

#### **3.10.1. Nhập (hoặc tạo) giá trị cho các phần tử trong ma trận**

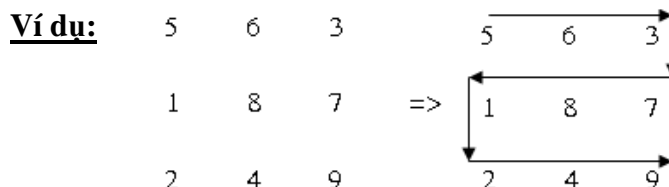
1. Viết chương trình khởi tạo giá trị các phần tử là ngẫu nhiên cho ma trận các số nguyên kích thước n X m.
2. Viết hàm nhập ma trận các số nguyên dương (nhập sai báo lỗi và cho nhập lại).
3. Viết hàm tạo ma trận a các số nguyên gồm n dòng m cột. Trong đó phần tử  $a[i][j] = i * j$ .
4. Viết hàm tạo giá trị ngẫu nhiên cho các phần tử trong 1 ma trận vuông.
5. Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột ( $5 < n, m < 200$ ), sao cho giá trị của các phần tử trong khoảng từ 0 đến 99.
6. Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột ( $5 \leq n, m < 100$ ), sao cho giá trị của các phần tử phải toàn là số lẻ.
  - ☞ Mở rộng cho các trường hợp số chẵn, số vừa âm vừa lẻ, ...
7. Viết chương trình phát sinh ngẫu nhiên mảng 1 chiều A (gồm n phần tử, với  $5 < n < 50$ ) các số nguyên sao cho giá trị phát sinh ngẫu nhiên phải toàn là số nguyên tố.
  - ☞ Mở rộng cho các trường hợp số hoàn thiện, số hoàn hảo, ...).
8. Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột ( $5 \leq n, m < 100$ ), sao cho giá trị của các phần tử trong khoảng từ -100 đến 100.

9. Viết hàm phát sinh ngẫu nhiên ma trận vuông ( $n \times n$ ) các số nguyên dương, sao cho số phát sinh sau phải lớn hơn hay bằng số phát sinh liền trước đó (các phần tử của ma trận được sắp xếp tăng dần từ trái sang phải và từ trên xuống dưới)  
 ➤ *Mở rộng* cho trường hợp số giảm dần.
10. Giả sử gọi vị trí của 1 phần tử trong ma trận = chỉ số dòng + chỉ số cột của phần tử đó. Viết hàm phát sinh ngẫu nhiên ma trận vuông ( $n \times n$ ) các số nguyên dương, sao cho những vị trí là số lẻ chỉ nhận giá trị nhập vào là số lẻ, và những vị trí là số chẵn chỉ nhận giá trị nhập vào là số chẵn.  
 ➤ *Mở rộng*: cho trường hợp ngược lại: vị trí lẻ chỉ nhận số chẵn; vị trí chẵn chỉ nhận số lẻ.
11. Cho ma trận các số thực  $A(m \times n)$ . Hãy xây dựng ma trận  $B(m \times n)$  từ ma trận  $A$  sao cho  $B[i][j] = \text{abs}(A[i][j])$ .
12. Cho ma trận các số thực  $A(m \times n)$ . Hãy xây dựng ma trận  $B(m \times n)$  từ ma trận  $A$  sao cho  $B[i][j] = \text{lớn nhất của dòng } i \text{ và cột } j \text{ trong ma trận } A$ .
13. Cho ma trận các số thực  $A(m \times n)$ . Hãy xây dựng ma trận  $B(m \times n)$  từ ma trận  $A$  sao cho  $B[i][j] = \text{số lượng phần tử dương xung quanh } (A[i][j]) \text{ trong ma trận } A$  ( $B[i][j]$  tối đa là 8 và nhỏ nhất là 0).
14. (\*) Xây dựng ma phương bậc  $n$ . Một ma trận được gọi là ma phương khi tổng các phần tử trên các dòng, các cột và hai đường chéo chính (dạng 1 và dạng 2) đều bằng nhau.

### 3.10.2. Xuất

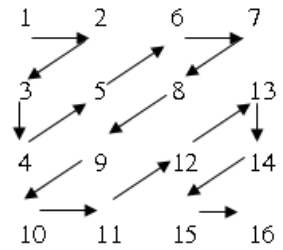
15. Cho ma trận vuông ( $n \times n$ ) các số nguyên. Viết hàm in ra:
  - 15.1. Các phần tử có giá trị là số lẻ sẽ được in giá trị, ngược lại các phần tử có giá trị là số chẵn sẽ được in thay thế bằng ký tự "X".
  - 15.2. Các phần tử nằm trên 2 đường chéo chính và phụ, các phần tử khác in khoảng trắng.
  - 15.3. Các phần tử **không** nằm trên 2 đường chéo chính và phụ, các phần tử nằm trên 2 đường chéo chính và phụ sẽ được in khoảng trắng (ngược lại với bài 15.2).
16. Xuất theo hình cho trước:
  - 16.1. In các giá trị các phần tử của ma trận theo thứ tự của đường chéo song song với đường chéo phụ của ma trận.  
 Ví dụ: Ma trận nhập vào:  
 Kết quả in ra: 7.0; 8.5; 12; 9.5; 5.5; 8.0; 10; 6.2; 3.3; 9.5; 15; 8.0; 1.0; 12; 9; 13;
  - 16.2. In các giá trị các phần tử của ma trận theo đường zigzag ngang.

|     |     |     |    |
|-----|-----|-----|----|
| 7.0 | 8.5 | 9.5 | 10 |
| 12  | 5.5 | 6.2 | 15 |
| 8.0 | 3.3 | 8.0 | 12 |
| 9.5 | 1.0 | 9.0 | 13 |



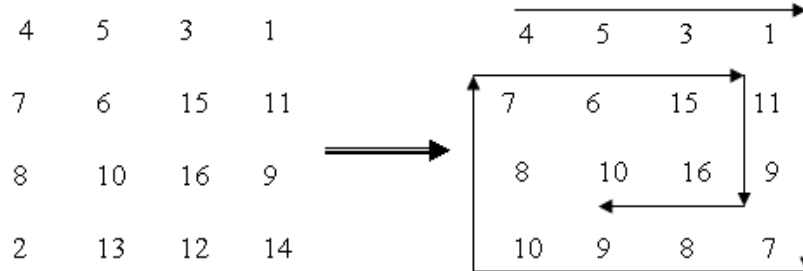
16.3. In các giá trị các phần tử của ma trận theo đường zigzag chéo

**Ví dụ:**



16.4. Lần lượt in các giá trị các phần tử của ma trận theo đường xoắn ốc từ ngoài vào trong theo chiều kim đồng hồ và ngược chiều kim đồng hồ.

**Ví dụ :**



### 3.10.3. Tính tổng/trung bình

#### 3.10.3.1. Thao tác trên toàn bộ các phần tử có trong ma trận

Viết hàm thực hiện những yêu cầu sau (mỗi yêu cầu được viết thành 1 hàm riêng):

1. Tính tổng tất cả các số
2. Tính tổng các số chẵn.

✎ *Mở rộng:* cho các trường hợp: tổng các số lẻ, tổng các số dương ( $\geq 0$ ), tổng các số âm.

3. Tính giá trị trung bình cộng
4. Tính giá trị trung bình nhân
5. Tính tổng các số nguyên tố.
6. Tính tổng các số hoàn thiện. Biết rằng số hoàn thiện là số (n) có tổng các ước số không kể n bằng chính n. Ví dụ: 6 là số hoàn thiện vì  $1 + 2 + 3 = 6$

#### 3.10.3.2. Thao tác trên dòng có trong ma trận

Viết hàm thực hiện những yêu cầu sau (mỗi yêu cầu được viết thành 1 hàm riêng):

7. Tính tổng tất cả các số trên mỗi dòng. Sau đó in ra giá trị tổng dòng lớn nhất.
8. Tính tổng các số chẵn trên mỗi dòng. Sau đó in ra giá trị tổng các số chẵn lớn nhất.
9. Tính tổng các số âm ( $< 0$ ) trên mỗi dòng. Sau đó in ra giá trị tổng âm lớn nhất.
10. Tính giá trị trung bình cộng trên mỗi dòng. Sau đó in ra giá trị tổng trung bình lớn nhất.
11. Tính giá trị trung bình nhân trên mỗi dòng. Sau đó in ra giá trị trung bình nhân lớn nhất.
12. Tính tổng các số nguyên tố trên mỗi dòng. Nếu dòng không có số nguyên tố, xem như tổng của dòng đó là 0.
13. (\*) In ra số nguyên tố lớn nhất trên mỗi dòng. Nếu dòng không có số nguyên tố, xem như số nguyên tố lớn nhất của dòng đó là 0.

**Từ bài 7 đến 13 thực hiện in ra màn hình theo dạng:****Ma trận ban đầu**

|   |   |   |   |
|---|---|---|---|
| 4 | 2 | 9 | 7 |
| 2 | 2 | 9 | 4 |
| 7 | 8 | 3 | 4 |
| 4 | 3 | 5 | 6 |

Kết quả in ra màn hình ⇨

|   |   |   |   |             |    |
|---|---|---|---|-------------|----|
| 4 | 2 | 9 | 7 | Tổng dòng = | 22 |
| 2 | 2 | 9 | 4 | Tổng dòng = | 17 |
| 7 | 8 | 3 | 4 | Tổng dòng = | 22 |
| 4 | 3 | 5 | 6 | Tổng dòng = | 18 |

Tổng dòng lớn nhất là 22, dòng lớn nhất là dòng 0,2

14. Giả sử định nghĩa dãy tăng trên mỗi dòng là dãy thỏa 2 điều kiện sau:

14.1. Có ít nhất 3 số liên tiếp nhau.

14.2. Giá trị của số đi trước phải nhỏ hơn hay bằng số đi sau liền kề.

In ra các dãy tăng có trên mỗi dòng theo dạng của ví dụ sau:

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

**Ma trận ban đầu**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | X | X |
| 2 | 2 | 9 | X | 6 | 7 | 8 | X |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| X | 3 | 5 | 6 | X | 2 | 7 | 9 |

**Kết quả in ra màn hình****3.10.3.3. Thao tác trên các cột có trong ma trận:**

Thực hiện tương tự như các yêu cầu có trong bài tập 7.2.3.2, nhưng thay dòng thành cột.

**3.10.3.4. Thao tác trên đường chéo chính của ma trận**

Lần lượt thực hiện các yêu cầu sau cho cả 2 dạng đường chéo chính và đường chéo phụ.

15. Tính tổng tất cả các số trên đường chéo
16. Tính tổng các số lẻ trên đường chéo
17. Tính tổng các số dương ( $\geq 0$ ) trên đường chéo
18. Tính giá trị trung bình cộng trên đường chéo
19. Tính giá trị trung bình nhân trên đường chéo
20. Tính tổng các số nguyên tố trên đường chéo

**3.10.3.5. Thao tác trên từng đường chéo phụ của ma trận**

Lần lượt thực hiện các yêu cầu trong bài tập 7.2.3.4 cho cả 2 dạng đường chéo chính và đường chéo phụ.

**3.10.3.6. Thao tác trên các phần tử thuộc tam giác (không tính đường chéo chính) của ma trận**

Lần lượt thực hiện các yêu cầu trong bài tập 7.2.3.4 cho cả 2 dạng đường chéo chính và đường chéo phụ.

21. Tính tổng tất cả các số
22. Tính tổng các số lẻ
23. Tính tổng các số dương ( $\geq 0$ )
24. Tính giá trị trung bình nhân
25. Tính tổng các số hoàn thiện



### *3.10.3.7. Thao tác trên các phần tử nằm trên đường biên trong ma trận*

Thực hiện tương tự như các yêu cầu có trong bài tập 7.2.3.6.

### **3.10.4. Tìm kiếm**

Cho ma trận các số nguyên A gồm d dòng và c cột. Viết hàm thực hiện các yêu cầu tìm kiếm sau:

#### *3.10.4.1. Tìm giá trị trong toàn ma trận*

26. Lớn nhất trong ma trận
27. Phần tử chẵn dương và nhỏ nhất trong ma trận.
28. Am lẻ lớn nhất
29. Nhỏ nhất trong ma trận.
30. Tìm số đầu tiên nhỏ hơn x (với x là tham số được truyền cho hàm).
31. Số nguyên tố xuất hiện đầu tiên (tính theo chiều từ trên xuống dưới và từ trái sang phải).
32. Số nguyên tố xuất hiện cuối cùng.
33. Chẵn cuối cùng trong ma trận.
34. Số hoàn thiện đầu tiên trong ma trận.
35. Số hoàn thiện lớn nhất trong ma trận.

#### *3.10.4.2. Tìm vị trí*

Thực hiện các yêu cầu tương tự như 10 bài tập trong phần 7.2.4.1 nhưng thay thế yêu cầu tìm giá trị bằng tìm vị trí (chỉ số)

##### *3.10.4.2.1 Tìm trên phạm vi có trong yêu cầu*

Thực hiện các yêu cầu tương tự như 10 bài tập trong phần 2.4.1 nhưng thay vì thao tác trên toàn ma trận, hàm được xây dựng chỉ thao tác trên phạm vi có trong yêu cầu

36. Tìm giá trị trên đường chéo chính (dấu huyền).
37. Tìm giá trị trên đường chéo phụ (dấu sắc).
38. Tìm giá trị trong tam giác trên của đường chéo chính.
39. Tìm giá trị trong tam giác dưới của đường chéo chính.
40. Tìm giá trị trong tam giác trên của đường chéo phụ.
41. Tìm giá trị trong tam giác dưới của đường chéo phụ.

##### *3.10.4.2.2 Tìm kiếm khác*

42. Viết hàm tìm cột có tổng nhỏ nhất trong ma trận.
43. Viết hàm tìm dòng có tổng lớn nhất trong ma trận.

### **3.10.5. Kỹ thuật đặt lính canh**

**Yêu cầu chung:** Các hàm cần trả về giá trị và vị trí (dòng, cột) của các phần tử thỏa yêu cầu.

#### *3.10.5.1. Tìm kiếm trên toàn ma trận*

44. Tìm số chẵn xuất hiện đầu tiên trong ma trận số nguyên.
45. Tìm giá trị dương lớn nhất trong ma trận.
46. Tìm giá trị âm lớn nhất trong ma trận.

47. Tìm giá trị lớn thứ nhì trong ma trận.
48. Tìm giá trị dương đầu tiên trong ma trận.
49. Tìm giá trị nguyên tố lớn nhất trong ma trận vuông các số nguyên
50. Tìm số nguyên tố đầu tiên trong ma trận các số nguyên.
51. Tìm số chẵn lớn nhất trong ma trận các số nguyên.
52. Tìm giá trị dương nhỏ nhất trong ma trận các số thực.
53. Tìm số nguyên tố lớn nhất trong ma trận các số nguyên.
54. Tìm một chữ số xuất hiện nhiều nhất trong ma trận các số nguyên.
55. Tìm giá trị xuất hiện nhiều nhất trong ma trận số thực.
56. Tìm số hoàn thiện nhỏ nhất trong ma trận các số nguyên.
57. Tìm số xuất hiện nhiều nhất trong ma trận các số nguyên.

Ví dụ: cho ma trận

|    |    |    |    |    |
|----|----|----|----|----|
| 34 | 45 | 23 | 24 | 52 |
| 78 | 47 | 45 | 31 | 34 |
| 94 | 34 | 22 | 76 | 74 |

Số xuất hiện nhiều nhất: 34; xuất hiện 3 lần

58. Tìm số chính phương lớn nhất trong ma trận các số nguyên.
59. Viết hàm thay thế những phân tử có giá trị x thành phân tử có giá trị y trong ma trận (x , y nhập từ bàn phím).

### 3.10.5.2. Tìm kiếm trên dòng/cột/đường chéo của ma trận

60. Dùng mảng phụ, viết hàm xuất
  - 60.1. Các giá trị chẵn theo thứ tự giảm dần.
  - 60.2. Số nguyên tố trong ma trận theo thứ tự tăng dần.
  - 60.3. Các giá trị âm theo thứ tự giảm dần.
61. Cho ma trận các số nguyên. Viết hàm thực hiện liệt kê theo các yêu cầu sau:
  - 61.1. Liệt kê các dòng có chứa giá trị chẵn liên tiếp nhau.
  - 61.2. Liệt kê các dòng có chứa giá trị âm.
  - 61.3. Liệt kê các dòng có chứa số nguyên tố.
  - 61.4. Liệt kê các dòng có chứa giá trị chẵn.
  - 61.5. Liệt kê các cột có chứa số chính phương.
  - 61.6. Liệt kê các dòng toàn âm.
  - 61.7. Liệt kê chỉ số của các dòng chứa toàn giá trị chẵn.
  - 61.8. Liệt kê các dòng có giá trị giảm dần trong ma trận số thực.
  - 61.9. Liệt kê các dòng có giá trị tăng dần trong ma trận số thực.
  - 61.10. Liệt kê các dòng trong ma trận các số thực thỏa mãn đồng thời các điều kiện sau: dòng có chứa giá trị âm, giá trị dương và giá trị 0 (phần tử trung hòa).
62. Tìm các dòng có chứa giá trị lớn nhất của ma trận trong ma trận các số thực.

63. Liệt kê các dòng có tổng dòng lớn nhất trong ma trận.
64. Liệt kê các cột có tổng cột nhỏ nhất trong ma trận.
65. Liệt kê các dòng có nhiều số chẵn nhất trong ma trận số nguyên.
66. Liệt kê các dòng có nhiều số nguyên tố nhất trong ma trận số nguyên.
67. Liệt kê các dòng có nhiều số hoàn thiện nhất trong ma trận số nguyên.
68. Tìm giá trị lớn nhất trên một dòng i trong ma trận các số thực.
69. Tìm giá trị nhỏ nhất trên một cột j trong ma trận các số thực.
70. Tìm giá trị lớn nhất trong tam giác trên của đường chéo dạng 1
71. Tìm giá trị nhỏ nhất trong tam giác dưới của đường chéo dạng 2
72. Tìm giá trị lớn nhất trên đường chéo chính trong ma trận vuông các số thực
73. Tìm giá trị lớn nhất trên đường chéo phụ trong ma trận vuông các số thực
74. (\*)Tìm hai giá trị gần nhau nhất trong ma trận
75. (\*) Liệt kê các cột nhiều chữ số nhất trong ma trận các số nguyên (lietkecot).
76. (\*) Tìm ma trận con có tổng lớn nhất.
77. (\*) Cho ma trận vuông A ( $n \times n$ ) các số thực. Hãy tìm ma trận vuông B ( $k \times k$ ) sao cho tổng các giá trị trên ma trận vuông này là lớn nhất

### 3.10.6. Đếm

#### 3.10.6.1.Đếm giá trị trong toàn ma trận

78. Số lượng số dương
79. Số lượng số âm
80. Số lượng số chẵn và dương.
81. Số lượng số Am và lẻ
82. Nhỏ nhất trong ma trận.
83. Số lượng số x xuất hiện trong ma trận(với x là tham số được truyền cho hàm).
84. Số lượng số nhỏ hơn x (với x là tham số được truyền cho hàm).
85. Số lượng số nguyên tố.
86. Số lượng số nguyên tố nhỏ hơn x (với x là tham số được truyền cho hàm).
87. Đếm số lượng số dương trên biên ma trận trong ma trận các số thực.

Ví dụ: cho ma trận

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| -34 | 45  | -23 | -24 | 52  |
| 12  | -11 | 21  | -56 | -75 |
| 78  | 47  | 45  | 31  | -34 |
| -94 | -34 | 22  | 76  | 74  |

Số lượng giá trị dương nằm trên biên: 7

88. (\*) Đếm số lượng phần tử cực đại trong ma trận các số thực. Một phần tử được gọi là cực đại khi nó lớn hơn các phần tử xung quanh.

89. (\*) Đếm số lượng phần tử cực trị trong ma trận các số thực. Một phần tử được gọi là cực trị khi nó lớn hơn các phần tử xung quanh hoặc nhỏ hơn các phần tử xung quanh.
90. (\*) Đếm số lượng giá trị có trong ma trận các số thực. Lưu ý: Nếu có k phần tử ( $k \geq 1$ ) trong ma trận bằng nhau thì ta chỉ tính là 1.
91. (\*) Tính tổng các phần tử cực trị trong ma trận các số thực. Một phần tử được gọi là cực trị khi nó lớn hơn các phần tử xung quanh hoặc nhỏ hơn các phần tử xung quanh.
92. (\*) Đếm số lượng giá trị “Yên Ngựa” trên ma trận. Một phần tử được gọi là Yên Ngựa khi nó lớn nhất trên dòng và nhỏ nhất trên cột.
93. (\*) Đếm số lượng giá trị “Hoàng Hậu” trên ma trận. Một phần tử được gọi là Hoàng Hậu khi nó lớn nhất trên dòng, trên cột và hai đường chéo đi qua nó.

### 3.10.6.2. Đếm trên phạm vi có trong yêu cầu

Thực hiện yêu cầu tương tự như các bài tập trong phần 7.2.6.1 nhưng thay vì thao tác trên toàn ma trận, hàm được xây dựng chỉ thao tác trên phạm vi có trong yêu cầu

94. Đếm giá trị trên đường chéo chính.
95. Đếm giá trị trên đường chéo phụ
96. Đếm giá trị trong tam giác trên của đường chéo chính
97. Đếm giá trị trong tam giác dưới của đường chéo chính
98. Đếm giá trị trong tam giác trên của đường chéo phụ
99. Đếm giá trị trong tam giác dưới của đường chéo phụ
100. Đếm giá trị có trên từng dòng
101. Đếm giá trị có trên từng cột

### 3.10.6.3. Đếm khác

102. Đếm số lượng cặp giá trị đối xứng nhau qua đường chéo chính
103. Đếm số lượng dòng giảm trong ma trận
104. Đếm số lượng phần tử cực đại trong ma trận
105. Đếm số lượng giá trị nhỏ nhất trong ma trận các số thực.
106. Đếm số lượng giá trị chẵn nhỏ nhất trong ma trận các số nguyên.

## 3.10.7. Sắp xếp

### 3.10.7.1. Sắp xếp trên toàn ma trận

107. Viết hàm sắp xếp ma trận theo thứ tự tăng dần từ trên xuống dưới và từ trái qua phải theo phương pháp DÙNG mảng phụ.
108. Viết hàm sắp xếp ma trận theo thứ tự tăng dần từ trên xuống dưới và từ trái qua phải theo phương pháp KHÔNG DÙNG mảng phụ.
109. Viết hàm sắp xếp tất cả các cột trong ma trận sao cho các phần tử trên mỗi cột giảm dần từ trên xuống dưới (vị trí cột của các phần tử không đổi)
110. Viết hàm sắp xếp các phần tử trong ma trận theo yêu cầu sau:
  - Cột có chỉ số chẵn giảm dần từ trên xuống.
  - Cột có chỉ số lẻ tăng dần từ trên xuống.
111. Viết hàm sắp xếp các phần tử trong ma trận theo yêu cầu sau:

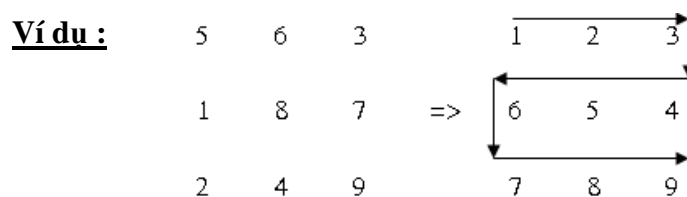
- Dòng có chỉ số chẵn tăng dần.
  - Dòng có chỉ số lẻ giảm dần.
112. Sắp xếp các giá trị âm của ma trận tăng dần, các giá trị dương giảm và các phần tử có giá trị bằng 0 giữ nguyên vị trí.
  113. Sắp xếp các giá trị chẵn của ma trận số nguyên tăng dần, các giá trị lẻ giảm dần.
  114. Sắp xếp các giá trị dương nằm trên biên ma trận tăng dần theo chiều kim đồng hồ.
  115. Sắp xếp các giá trị dương nằm trên biên ma trận các số thực tăng dần theo chiều kim đồng hồ.
  116. Cho ma trận vuông chứa các số thực A cạnh là n (với  $n \geq 3$ ). Sắp xếp các giá trị trong ma trận tam giác trên (không tính đường chéo) tăng dần từ trên xuống dưới và từ trái sang phải
  117. Sắp xếp các phần tử dương trong ma trận các số thực tăng dần theo hàng và cột bằng phương pháp sử dụng mảng phụ và không dùng mảng phụ (các số âm giữ nguyên giá trị và vị trí).

### 3.10.7.2. Sắp xếp dòng/cột/đường chéo của ma trận

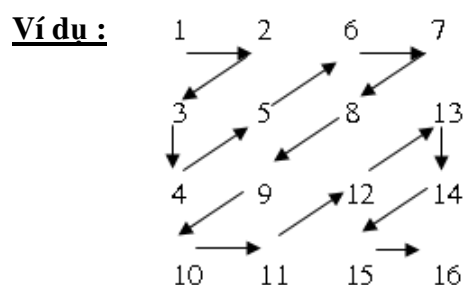
118. Viết hàm sắp xếp các dòng trên ma trận theo thứ tự tăng dần.
119. Viết hàm sắp xếp các cột trên ma trận theo thứ tự giảm dần.
120. Viết hàm sắp xếp các phần tử trên một dòng tăng dần từ trái sang phải.
121. Viết hàm sắp xếp các phần tử trên một dòng giảm dần từ trái sang phải.
122. Viết hàm sắp xếp các phần tử trên một cột tăng dần từ trên xuống dưới.
123. Hãy sắp xếp các dòng trong ma trận theo tiêu chuẩn sau: dòng có tổng dòng nhỏ hơn nằm ở trên và dòng có tổng dòng lớn hơn bằng nằm ở dưới.
124. Sắp xếp các phần tử trên đường chéo chính tăng dần (từ trên xuống dưới)
125. Sắp xếp các phần tử trên đường chéo phụ giảm dần (từ trên xuống dưới)
126. Đưa các giá trị chẵn về đầu ma trận vuông các số nguyên.

### 3.10.7.3. Sắp xếp ma trận theo hình cho trước

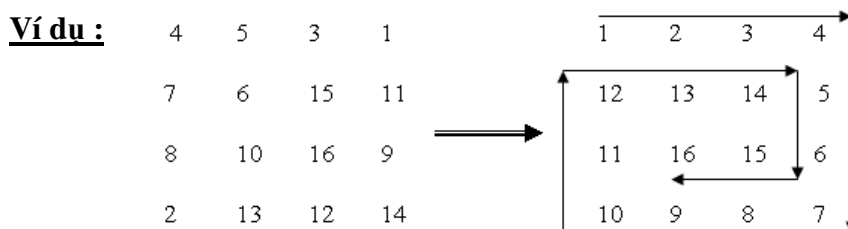
127. Viết hàm sắp xếp ma trận theo đường ziczắc ngang.



128. Viết hàm sắp xếp ma trận theo đường ziczắc chéo



129. Viết hàm sắp xếp ma trận theo đường xoắn ốc từ ngoài vào trong theo chiều kim đồng hồ.



### 3.10.7.4. Hoán vị – Xoay vòng dòng/cột

130. Hoán vị hai dòng r1, r2 trong ma trận.  
 131. Hoán vị hai cột c1, c2 trong ma trận.  
 132. Viết hàm hoán vị dòng có tổng lớn nhất với dòng có tổng nhỏ nhất.  
 133. Dịch xuống xoay vòng các hàng trong ma trận.

Ví dụ:

Ma trận ban đầu

|     |    |    |    |
|-----|----|----|----|
| 87  | 75 | 62 | 54 |
| 46  | 40 | 33 | 28 |
| 20  | 18 | 15 | 10 |
| -20 | 67 | 53 | 23 |

Kết quả dịch xuống xoay vòng các hàng

|     |    |    |    |
|-----|----|----|----|
| -20 | 67 | 53 | 23 |
| 87  | 75 | 62 | 54 |
| 46  | 40 | 33 | 28 |
| 20  | 18 | 15 | 10 |

134. Dịch lên xoay vòng các hàng trong ma trận.  
 135. Dịch trái xoay vòng các cột trong ma trận.

Ví dụ:

Ma trận ban đầu

|     |    |    |    |
|-----|----|----|----|
| 87  | 75 | 62 | 54 |
| 46  | 40 | 33 | 28 |
| 20  | 18 | 15 | 10 |
| -20 | 67 | 53 | 23 |

Kết quả dịch trái xoay vòng các cột

|    |    |    |     |
|----|----|----|-----|
| 75 | 62 | 54 | 87  |
| 40 | 33 | 28 | 46  |
| 18 | 15 | 10 | 20  |
| 67 | 53 | 23 | -20 |

136. Dịch phải xoay vòng các cột trong ma trận.  
 137. Dịch phải xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên ma trận.  
 138. Dịch trái xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên ma trận.  
 139. Xoay ma trận một góc 90 độ theo 2 cách thuận chiều kim đồng hồ và ngược chiều kim đồng hồ.  
 140. Xoay ma trận một góc 180 độ.  
 141. Xoay ma trận một góc 270 độ.  
 142. Sắp xếp “các dòng” tăng dần theo tổng dòng

### **3.10.8. Thêm – Xoá dòng/cột**

143. Viết hàm xoá một dòng r trên ma trận.
144. Viết hàm xoá một cột c trên ma trận.
145. Viết hàm xoá dòng có tổng lớn nhất trên ma trận.
146. Viết hàm tìm và thay thế các phần tử chẵn trong ma trận bằng ước số nhỏ nhất của nó.
147. Viết hàm chèn một dòng chứa toàn giá trị X vào vị trí r trên ma trận.
148. Viết hàm chèn một cột chứa toàn giá trị X vào vị trí c trên ma trận.

### **3.10.9. Kỹ thuật xử lý khác trên ma trận**

149. Hãy biến đổi ma trận bằng cách thay các phần tử chứa giá trị âm bằng giá trị tuyệt đối của chính phần tử đó.
150. Hãy biến đổi ma trận các số thực bằng cách thay các giá trị bằng giá trị nguyên gần nó nhất (làm tròn số).
151. Chiều gương ma trận theo trục ngang theo 2 trường hợp:
  - Chiều nửa trên xuống nửa dưới (bỏ giá trị cũ của nửa dưới).
  - Chiều nửa dưới lên nửa trên (bỏ giá trị cũ của nửa trên).
152. Chiều gương ma trận theo trục dọc.
  - Chiều nửa trái sang nửa phải (bỏ giá trị cũ của nửa trái).
  - Chiều nửa phải sang nửa trái (bỏ giá trị cũ của nửa phải).

### **3.10.10. Các phép toán trên ma trận**

153. Tính tổng hai ma trận
154. Tính hiệu hai ma trận
155. tính tích hai ma trận  $A(n \times m)$  và  $B(m \times l)$ . In kết quả ma trận tích  $C(n \times l)$ .
156. (\*)Tìm ma trận nghịch đảo
157. (\*) Tính định thức của ma trận
158. (\*) Tạo ma phương bậc  $(n \times n)$ .

### **3.10.11. Game**

159. Viết chương trình bắn tàu với máy.
160. Viết chương trình Puzzle 15 số.
161. Viết trò chơi logic. Quy tắc trò chơi như sau: Máy sẽ tạo ngẫu nhiên một số gồm bốn chữ số (không cho ta biết), trong đó không có chữ số nào được lặp lại. Sau đó máy sẽ yêu cầu ta đoán. Nếu trong số giả định của ta có chữ số nào trùng với chữ số mà máy đã tạo thì: nếu vị trí chữ số đó trùng với vị trí mà máy đã tạo ra thì nó sẽ cho 1 dấu +, nếu chỉ trùng chữ số nhưng khác vị trí thì máy sẽ cho dấu -. Nếu trong số giả định có chữ số không có trong nhóm chữ máy tạo ra thì nó không cho dấu gì hết.
162. Viết chương trình minh họa kỹ thuật sắp xếp mảng.
163. Viết chương trình demo vẽ hộp trên màn hình (tương tự Screen saver).
164. Viết trò chơi “tung đồng xu”. Quy tắc trò chơi như sau: Có hai người A,B mỗi người có 10 đồng tiền. Đến lượt người nào đi sẽ tung đồng tiền lên, nếu nó sấp(1) thì người đó thắng và người kia phải mất một đồng, nếu nó ngửa(0) thì người đó phải mất một đồng. Sau đó đổi người.Trò chơi sẽ kết thúc nếu có một người hết tiền.



### 3.10.12. Khác

165. Nhập vào ma trận vuông số thực cấp  $n$ . Kiểm tra các tính chất sau đây của ma trận:
  - Ma trận tam giác trên hay ma trận tam giác dưới ? ma trận đơn vị hay không phải (ma trận tam giác trên (dưới) là ma trận có ít nhất một phần tử trên đường chéo chính khác 0 và toàn bộ các phần tử dưới (trên) đường chéo đều bằng 0. Ma trận đơn vị có tất cả các phần tử trên đường chéo chính bằng 1 và các phần tử khác bằng 0.
  - Ma trận đối xứng qua đường chéo chính hay đối xứng tâm hay không phải. (Ma trận đối xứng qua đường chéo chính :  $a[i][j] = a[j][i]$  với mọi  $i, j$ . Còn với ma trận đối xứng tâm :  $a[i][j] = a[n-i-1][n-j-1]$ )
  - Ma trận lõm hay ma trận lõm hay không phải (Ma trận lõm có các phần tử bên trong nhỏ hơn các phần tử trên 4 cạnh, ngược lại là lõm)
166. Nhập vào ma trận số thực cấp  $n \times m$ . Tạo menu thực hiện các thao tác trên ma trận:
  - Tìm phần tử nhỏ nhất, lớn nhất (chỉ ra vị trí)
  - Tìm hàng có tổng lớn nhất, cột có tổng nhỏ nhất. In vị trí tìm thấy và tổng của nó
  - In các phần tử nằm trên đường chéo song song và cách đường chéo chính khoảng cách là  $k-1$  ( $k$  nhập từ bàn phím. Có 2 đường chéo cách đường chéo chính  $k-1$  là :  $\{a[i][i \pm k]\}$ )
167. Nhập vào ma trận số thực cấp  $n \times m$ . Tạo menu thực hiện các thao tác trên ma trận:
  - Xóa hàng  $k$  (nhập từ bàn phím)
  - Xóa cột  $k$  (nhập từ bàn phím)
  - Xóa hàng và cột chứa phần tử nhỏ nhất
  - Nhập số nguyên dương  $k$  ( $k < n$ ) và dãy có  $n$  số thực, chèn dãy đó vào hàng  $k$ .
  - Nhập số nguyên dương  $k$  ( $k < m$ ) và dãy có  $m$  số thực, chèn dãy đó vào cột  $k$ .
  - Hoán vị các hàng sao cho tổng giá trị của từng hàng giảm từ trên xuống.
168. Nhập ma trận ký tự, vuông cấp  $n$ . Thực hiện các thao tác:
  - Đổi các ký tự alphabet thành ký tự hoa.
  - Cho biết hàng nào có nhiều nguyên âm nhất (nếu có 2 hàng thì in hàng phía trên)
169. Cho mảng 2 chiều  $A$  gồm các số nguyên dương  $n$  dòng,  $m$  cột ( $0 < m, n < 20$ ). Viết chương trình thực hiện các yêu cầu sau:
  - Tính tổng các phần tử là số nguyên tố trên dòng thứ  $i$  của  $A$ .
  - Tìm phần tử lớn nhất, xoá dòng, cột chứa phần tử lớn nhất đó.
  - In ra những dòng nào có tổng giá trị các phần tử là số nguyên tố bằng tổng giá trị các phần tử còn lại.
  - Xoá cột có nhiều số nguyên tố nhất (nếu có 2 cột có cùng số lượng số nguyên tố thì xoá cột đầu tiên).
170. Viết chương trình nhập/ xuất một ma trận vuông số thực  $n \times n$  ( $n \leq 10$ ). Sau đó thực hiện các yêu cầu sau:
  - Thiết kế hàm xét cột thứ  $j$  của ma trận có thuộc dạng hội tụ hay không. Cột  $j$  được gọi là hội tụ khi thoả điều kiện: càng về cuối ma trận trị tuyệt đối của độ sai biệt giữa hai phần tử kế nhau trong cột càng giảm.
  - Vận dụng câu a) để khảo sát xem ma trận có phải là hội tụ hay không? (nghĩa là mọi cột của ma trận đều hội tụ).



## 4 KIỂU DỮ LIỆU CÓ CẤU TRÚC (Struct data types)

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

- Hiểu được các khái niệm cơ bản về kiểu dữ liệu có cấu trúc;
- Biết Nhập Xuất dữ liệu có cấu trúc cho một phần tử ;
- Biết Nhập, Xuất dữ liệu có cấu trúc và lưu trên mảng một chiều;
- Cách tìm kiếm và sắp xếp dữ liệu có cấu trúc trên mảng một chiều;
- Đi sâu vào các giải thuật trên mảng một chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử theo từng thành phần của kiểu dữ liệu có cấu trúc.

### 4.1. Khái niệm

Kiểu cấu trúc (*Structure*) là một kiểu dữ liệu do người dùng tự định nghĩa, là kiểu dữ liệu bao gồm nhiều thành phần, mỗi thành phần có một kiểu dữ liệu khác nhau, mỗi thành phần được gọi là một trường (field).

Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng có cùng kiểu dữ liệu còn các phần tử của kiểu cấu trúc có thể có các kiểu dữ liệu khác nhau.

Hình ảnh của kiểu cấu trúc được minh họa:

*Minh họa kiểu Cấu trúc gồm 5 thành phần  
(có thể cùng hoặc khác kiểu dữ liệu)*

| 1       | 2       | 3       | 4       | 5       |
|---------|---------|---------|---------|---------|
| Field 1 | Field 2 | Field 3 | Field 4 | Field 5 |

*Minh họa kiểu Mảng gồm 5 phần tử luôn có cùng kiểu dữ liệu*

| 0    | 1    | 2    | 3    | 4    |
|------|------|------|------|------|
| a[0] | a[1] | a[2] | a[3] | a[4] |

### 4.2. Cách khai báo kiểu cấu trúc

#### 4.2.1. Cú pháp

```
struct <Tên cấu trúc>
{
    <Kiểu> <Trường 1> ;
    <Kiểu> <Trường 2> ; ...
    <Kiểu> <Trường n> ;
};
typedef struct <Tên cấu trúc> < tên mới > ;
```

Trong đó:

- **<Tên cấu trúc>**: là một tên được đặt theo quy tắc đặt tên của danh biểu; tên này mang ý nghĩa sẽ là tên kiểu cấu trúc.
- **<Kiểu> <Trường i>** (i=1..n): mỗi trường trong cấu trúc có dữ liệu thuộc kiểu gì (tên của trường phải là một tên được đặt theo quy tắc đặt tên của danh biểu).

#### 4.2.2. Ví dụ 1

Khai báo một kiểu struct có tên **NgayThang** có các thành phần là: ngày, tháng, năm, và đặt tên mới là **Date**.

```

struct NgayThang
{
    int ngay;
    int thang;
    int nam;
};
typedef struct NgayThang Date ;

```

#### 4.2.3. Ví dụ 2

Khai báo một kiểu struct có tên **DiemThi** có các thành phần là: Mã số sinh viên, Mã số môn học, Lần thi, điểm thi.

##### 4.2.3.1. Cách 1

```

struct DiemThi
{
    char masv[8]; /* mã số sinh viên */
    char mamh[5]; /* mã số môn học */
    int lanthi; /* lần thi */
    float diem; /* điểm thi */
};
typedef struct DiemThi Diem ;

```

##### 4.2.3.2. Cách 2

Khai báo 3 biến **x, y, z** có kiểu **DiemThi**

```

struct DiemThi
{
    char masv[8]; /* mã số sinh viên */
    char mamh[5]; /* mã số môn học */
    int lanthi; /* lần thi */
    float diem; /* điểm thi */
} x, y, z ;

```

#### 4.2.4. Khởi tạo giá trị cho biến cấu trúc

- Cách khởi tạo giá trị cho biến cấu trúc giống như khởi tạo cho mảng.
- Ví dụ 3: `DATE birth = {12, 10, 2000};`

#### 4.2.5. Kiểu dữ liệu có cấu trúc có thể lồng vào nhau.

Ví dụ 4: Định nghĩa kiểu dữ liệu của học sinh HOCSINH gồm:

- Mã số học sinh (MSHS) : chuỗi có tối đa 5 ký tự.
- Họ tên (hoten) : chuỗi có tối đa 30 ký tự.
- Ngày tháng năm sinh (ngaysinh) : kiểu DATE.
- Địa chỉ (diachi) : chuỗi có tối đa 50 ký tự.
- Giới tính (phai) : chuỗi có tối đa 3 ký tự.
- Điểm trung bình (diemtb) : số thực.

Ta định nghĩa kiểu HOCSINH như sau:

```

struct DATE
{
    char thu[5];
    unsigned char ngay;
    unsigned char thang;
    int nam;
};

```

```
typedef struct HOCSINH
{
    char          MSHS[6];
    char          hoten[31];
    struct DATE   ngaysinh;
    char          diachi[51];
    unsigned char phai[4];
    float         diemtb;
};
```

☞ Khi định nghĩa kiểu dữ liệu struct lồng nhau, ta cần lưu ý: Kiểu dữ liệu cấu trúc được sử dụng phải khai báo trước (đặt ở phía trên).

### 4.3. Sử dụng biến cấu trúc

#### 4.3.1. Khai báo

Khi định nghĩa kiểu dữ liệu tức là ta có một kiểu dữ liệu mới, muốn sử dụng ta phải khai báo biến. Cú pháp khai báo biến của kiểu dữ liệu mới định nghĩa cũng giống như cách khai báo của các kiểu dữ liệu chuẩn.

```
struct < tên cấu trúc > < tên biến > ;
```

Ví dụ:     

```
struct DATE x ;     // Khai báo biến x có kiểu dữ liệu DATE
```

Tuy nhiên nếu khi định nghĩa struct có dùng từ khoá typedef thì ta có thể khai báo trực tiếp mà không cần từ khoá struct.

Ví dụ:     

```
DATE x ;     // Khai báo biến x có kiểu DATE
```

#### 4.3.2. Truy cập vào từng phần tử của cấu trúc

##### 4.3.2.1. Cú pháp

Thông qua dấu chấm để truy cập đến từng thành phần của struct .

**<tên biến struct>.<tên thành phần của struct>**

##### 4.3.2.2. Ví dụ 1

Để xuất ra màn hình tên của sinh viên sv1, ta thực hiện câu lệnh sau :

```
printf("%s", sv1.ten);
```

##### 4.3.2.3. Ví dụ 2

Định nghĩa cấu trúc dữ liệu cho phân số

```
struct phanso
{
    int tu;
    int mau ;
};
typedef struct phanso PS;
```

Khi đó Nếu ta khai báo một biến a có kiểu dữ liệu là phân số thì a gồm 2 thành phần: a.tu, a.mau

**PS a**

|             |              |
|-------------|--------------|
| <b>a.tu</b> | <b>a.mau</b> |
|-------------|--------------|

#### 4.4. Nhập dữ liệu cho kiểu dữ liệu có cấu trúc

##### 4.4.1. Nhập dữ liệu cho một phân số

```
struct phanso
{
    int tu;
    int mau ;
};
typedef struct phanso PS;
// hàm cài đặt
void nhapphanso (ps &x)
{
    printf("cho biet tu :");
    scanf("%d",&x.tu);
    printf("cho biet mau :");
    scanf("%d",&x.mau);
}
```

##### 4.4.2. Nhập vào điểm của một sinh viên

```
struct DiemThi
{
    char masv[8];
    char mamh[5];
    int lanthi;
    float diem;
};
typedef struct DiemThi Diem ;
void nhapdiem (Diem &sv)
{
    flushall () ;
    printf("\nNhap diem cho sinh vien:");
    printf("\nMa so sinh vien: ");
    gets(sv.masv);
    printf("Ma mon hoc: ");
    gets(sv.mamh);
    printf("Lan thi: ");
    scanf("%d",&sv.lanthi);
    printf("Diem thi: ");
    float t; scanf("%f",&t);
    sv.diem=t;
}
```

**Ghi nhớ:** Với kiểu dữ liệu thành phần là số thực, khi nhập, ta không nhập trực tiếp mà phải thông qua biến tạm.

```
float t;
scanf("%f",&t);
sv.diem=t;
```

#### 4.5. Xuất dữ liệu

##### 4.5.1. Xuất dữ liệu cho một phân số

```
void xuatphanso (ps x)
```

```
{  
    printf ("phan so :%d / %d ", x.tu , x.mau);  
}
```

#### 4.5.2. Xuất dữ liệu điểm

```
void xuatdiem (Diem sv)  
{  
    printf("\nXuat diem: ");  
    printf("\nMa so sinh vien: %s",sv.masv);  
    printf("\nMa mon hoc: %s",sv.mamh);  
    printf("\nLan thi: %d",sv.lanthi);  
    printf("\nDiem thi: %.1f ",sv.diem);  
}
```

#### 4.6. Ví dụ tổng hợp

Viết chương trình nhập vào tọa độ hai điểm trong mặt phẳng và tính khoảng cách giữa 2 điểm


```
#include <conio.h>  
#include <stdio.h>  
#include <math.h>  
typedef struct DIEM    //khai bao kieu du lieu DIEM gom toa do x, y  
{  
    double x;  
    double y;  
};  
void Nhap (DIEM &d, char c)  
{  
    printf("\nNhap vao toa do diem %c:\n",c);  
    printf("Tung do: ");  
    scanf("%f",&d.x);  
    printf("Hoanh do: ");  
    scanf("%f",&d.y);  
}  
void Xuat (DIEM d, char c)  
{  
    printf("\nToa do diem %c: x= %.2f, y= %.2f: ",c,d.x,d.y);  
}  
DIEM ToaDoTrungDiemAB (DIEM d1,DIEM d2)  
{  
    DIEM Temp;  
    Temp.x = (d1.x + d2.x)/2 ;  
    Temp.y = (d1.y + d2.y)/2 ;  
    return Temp;  
}  
double KhoangCach2Diem (DIEM d1,DIEM d2)  
{  
    return sqrt(pow(d1.x - d2.x,2)+pow(d1.y - d2.y,2));  
}  
int main ()  
{  
    DIEM A , B;    //khai bao 2 diem A, B  
    Nhap (A, 'A');  
    Xuat (A, 'A');  
    Nhap (B, 'B');  
    Xuat (B, 'B');
```

```

printf("\n Chieu dai canh AB: %5.2f",KhoangCach2Diem(A, B));
return 0;
}

```

#### 4.7. Sử dụng cấu trúc để trừu tượng hóa dữ liệu

 Phương pháp luận giải quyết bài toán bằng phương pháp trừu tượng hóa dữ liệu: (sử dụng phương pháp lập trình hướng thủ tục hàm)

- Bước 1: Xác định các kiểu dữ liệu. Trong bước này ta phải xác định xem trong bài toán (vấn đề) phải làm việc với những kiểu dữ liệu nào. Kiểu dữ liệu nào đã có sẵn, kiểu dữ liệu nào phải định nghĩa mới.
- Bước 2: Thiết kế hàm. Trong bước này ta phải xác định xem trong bài toán mà ta giải quyết cần phải có bao nhiêu hàm, tên của các hàm ra sao, các tham số đầu vào, kiểu dữ liệu trả như thế nào. Quan trọng hơn nữa là các tham số thì tham số nào là tham trị và tham số nào là tham biến.
- Bước 3: Định nghĩa hàm. Trong bước này ta sẽ tiến hành định nghĩa các hàm đã thiết kế và khai báo trong bước 2. Hơn nữa ta phải định nghĩa hàm main với các khai báo biến và thực hiện các lời gọi hàm cần thiết cho chương trình.

Lưu ý: Bước 1 và Bước 2 là hai bước quan trọng nhất.

##### 4.7.1. Phân số

Khai báo kiểu dữ liệu biểu diễn khái niệm phân số trong toán học và định nghĩa các hàm nhập, xuất, tính tổng cho kiểu dữ liệu này.

```

struct PhanSo
{
    int tu ;
    int mau ;
};
typedef struct PhanSo PHANSO;
void Nhap(PHANSO &x);
void Xuat(PHANSO x);
int TimUSCLN(int a,int b);
PHANSO Tong2PhanSo(PHANSO x, PHANSO y);
void Nhap(PHANSO &x)
{
    printf("\nNhap tu: ");
    scanf("%d",&x.tu);
    printf("Nhap mau: ");
    scanf("%d",&x.mau);
}
void Xuat(PHANSO x)
{
    printf("%d/%d",x.tu,x.mau);
}
int TimUSCLN(int a,int b)
{
    while (a!=b)
        if (a>b)
            a-=b;
        else
            b-=a;
    return a;
}

```

```

PHANSO Tong2PhanSo(PHANSO x, PHANSO y)
{
    PHANSO C;
    C.mau=x.mau*y.mau;
    C.tu= (x.tu*y.mau) + (y.tu*x.mau);
    int USC=TimUSCLN(abs(C.tu),abs(C.mau));
    C.tu=C.tu/USC;
    C.mau=C.mau/USC;
    return C;
}

int main ()
{
    PHANSO A,B,C;
    printf("Nhap phan so thu 1");
    Nhap(A);
    printf("Nhap phan so thu 2");
    Nhap(B);
    C=Tong2PhanSo(A,B);
    printf("Tong 2 phan so vua nhap la: ");
    Xuat(C) ;
    return 0;
}

```

#### 4.7.2. Hỗn số

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm hỗn số trong toán học và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Kiểu dữ liệu

```

struct HonSo
{
    int tu ;
    int mau ;
    int nguyen;
};

```

- Định nghĩa hàm nhập

```

void Nhap(HonSo &x)
{
    printf("\nNhap phan nguyen: ");
    scanf("%d",&x.nguyen);
    printf("Nhap tu:");
    scanf("%d",&x.tu);
    printf("Nhap mau: ");
    scanf("%d",&x.mau);
}

```

- Định nghĩa hàm xuất

```

void Xuat(HonSo x)
{
    printf("%d(%d/%d)",x.nguyen,x.tu,x.mau);
}

```

#### 4.7.3. Điểm trong mặt phẳng Oxy

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm điểm trong mặt phẳng Oxy và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Kiểu dữ liệu

```
struct Diem
{
    int x;
    int y;
};
typedef struct Diem DIEM;
```

- Định nghĩa hàm nhập

```
void Nhap(DIEM &p)
{
    printf("Nhap toa do x: ");
    scanf("%d",&p.x);
    printf("Nhap toa do y: ");
    scanf("%d",&p.y);
}
```

- Định nghĩa hàm xuất

```
void Xuat(DIEM p)
{
    printf("X=%d; Y=%d",p.x,p.y);
}
```

#### 4.7.4. Điểm trong không gian Oxyz

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm điểm trong không gian Oxy và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Kiểu dữ liệu

```
struct Diem3D
{
    int x;
    int y;
    int z;
};
typedef struct Diem3D DIEM3D;
```

- Định nghĩa hàm nhập

```
void Nhap(DIEM3D &p)
{
    printf("Nhap toa do x: ");
    scanf("%d",&p.x);
    printf("Nhap toa do y: ");
    scanf("%d",&p.y);
    printf("Nhap toa do z: ");
    scanf("%d",&p.z);
}
```

- Định nghĩa hàm xuất

```
void Xuat(DIEM3D p)
{
    printf("X=%d; Y=%d; Z=%d",p.x,p.y,p.z);
}
```



#### 4.7.5. Đơn thức

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm đơn thức  $P(x)=ax^2$  trong toán học và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Kiểu dữ liệu

```
struct DonThuc
{
    float a;
    int n;
};
typedef struct DonThuc DONTHUC;
```

- Định nghĩa hàm nhập

```
void Nhap(DONTHUC &p)
{
    printf("Nhap he so (a): ");
    scanf("%f",&p.a);
    printf("Nhap bac (n) cua da thuc: ");
    scanf("%d",&p.n);
}
```

- Định nghĩa hàm xuất

```
void Xuat(DONTHUC p)
{
    printf("%5.2fX^%d",p.a,p.n);
}
```

#### 4.7.6. Ngày

- Hãy khai báo kiểu dữ liệu biểu diễn ngày trong thế giới thực và định nghĩa hàm nhập , hàm xuất cho kiểu dữ liệu này.

- Kiểu dữ liệu

```
struct Ngay
{
    int d;
    int m;
    int y;
};
typedef struct ngay NGAY;
```

- Định nghĩa hàm nhập ngày

```
void Nhap(NGAY &p)
{
    printf("Nhap ngay: ");
    scanf("%d",&p.d);
    printf("Nhap thang: ");
    scanf("%d",&p.m);
    printf("Nhap nam: ");
    scanf("%d",&p.y);
}
```

- Định nghĩa hàm xuất ngày

```
void Xuat(NGAY p)
{
    printf("%d/%d/%d",p.d,p.m,p.y);
}
```

**4.7.7. Đường thẳng trong mặt phẳng Oxy**

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm đường thẳng  $ax+by+c=0$  trong mặt phẳng Oxy và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Khai báo kiểu dữ liệu
 

```
struct DuongThang
{
    float a;
    float b;
    float c;
};
typedef struct DuongThang DUONGTHANG;
```

**4.7.8. Đường tròn trong mặt phẳng Oxy**

- Hãy khai báo kiểu dữ liệu biểu diễn khái niệm đường tròn trong mặt phẳng Oxy và định nghĩa hàm nhập, hàm xuất cho kiểu dữ liệu này.

- Khai báo kiểu dữ liệu để biểu diễn đường tròn
 

```
struct Diem
{
    float x;
    float y;
};
typedef struct Diem DIEM;
struct DuongTron
{
    DIEM I;
    float R;
};
typedef struct DuongTron DUONGTRON;
```

- Định nghĩa hàm nhập điểm
 

```
void NhapDiem(DIEM &p)
{
    float temp;
    printf("Nhap toa do X: ");
    scanf("%f",&temp);
    p.x=temp;
    printf("Nhap toa do Y: ");
    scanf("%f",&temp);
    p.y=temp;
}
```

- Định nghĩa hàm nhập đường tròn
 

```
void NhapDuongTron(DUONGTRON &c)
{
    float temp;
    printf("Nhap tam cua duong tron:\n");
    NhapDiem (c.I);
    printf("Nhap ban kinh cua duong tron: ");
    scanf("%f",&temp);
    c.R=temp;
}
```

- Định nghĩa hàm xuất điểm
 

```
void XuatDiem(DIEM p)
{
    printf("X= %f; Y=%f\n",p.x,p.y);
}
```

- Định nghĩa hàm xuất đường tròn  

```
void XuatDuongTron(DUONGTRON c)
{
    printf("X= %f; Y=%f; Ban kinh=%f\n",c.I.x,c.I.y,c.R);
}
```

#### 4.8. Mảng cấu trúc

- **Khai báo**: tương tự như mảng một chiều hay ma trận (Kiểu dữ liệu bây giờ là kiểu dữ liệu có cấu trúc).
- **Truy xuất phần tử trong mảng**: tương tự như truy cập trên mảng một chiều hay ma trận. Nhưng do từng phần tử có kiểu cấu trúc nên phải chỉ định rõ cần lấy thành phần nào, tức là phải truy cập đến thành phần cuối cùng có kiểu là dữ liệu cơ bản (xem lại bảng các kiểu dữ liệu cơ bản).
- **Nguyên tắc viết chương trình có mảng cấu trúc**:

Do kiểu dữ liệu có cấu trúc thường chứa rất nhiều thành phần nên khi viết chương trình loại này ta cần lưu ý:

- Xây dựng hàm xử lý cho một kiểu cấu trúc.
- Muốn xử lý cho mảng cấu trúc, ta gọi lại hàm xử lý cho một kiểu cấu trúc đã được xây dựng bằng cách dùng vòng lặp.

##### 4.8.1. Bài toán 1

Viết chương trình

- Nhập vào một dãy phân số.
- Xuất ra dãy phân số vừa nhập.
- Tính tổng các phân số có trong dãy.
- Tìm phân số lớn nhất có trong dãy.
- Đếm số phần tử lớn nhất có trong dãy.
- Xóa tất cả các phần tử lớn nhất có trong dãy.

##### 4.8.1.1. Hình ảnh mảng một chiều các phân số

| PS A[8] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------|---|---|---|---|---|---|---|---|---|
| tu      | 9 | 6 | 1 | 5 | 1 | 1 | 1 | 8 | 3 |
| mau     | 1 | 2 | 2 | 3 | 7 | 3 | 2 | 2 | 2 |

Trong đó

$$a[0] = 9/1 \Rightarrow a[0].tu=9 \text{ và } a[0].mau= 1$$

$$a[1] = 6/2 \Rightarrow a[1].tu=6 \text{ và } a[1].mau= 2$$

...

$$a[8] = 3/2 \Rightarrow a[8].tu=3 \text{ và } a[8].mau= 2$$

#### 4.8.1.2. Nhập mảng phân số

```
// hàm cài đặt
void nhapmang (ps a[] , int n)
{
    for (int i=0 ; i<n ; i++)
        nhapphanso(a[i]);
}
```

#### 4.8.1.3. Xuất mảng phân số

```
// hàm cài đặt
void xuatmang (ps a[] , int n)
{
    for (int i=0 ; i<n ; i++)
        xuatphanso(a[i]);
}
```

### 4.8.2. Bài toán 2

Để có được danh sách điểm thi của sinh viên (có chứa tối đa 100 mẫu tin), ta khai báo một mảng các biến struct.

```
Diem sv[100];
```

#### 4.8.2.1. Nhập mảng danh sách điểm thi của sinh viên

```
//hàm cài đặt
void nhapmangdiem (Diem sv [] , int n)
{
    for (int i=0 ; i<n ; i++)
        nhapdiem(sv[i]);
}
```

#### 4.8.2.2. Xuất mảng danh sách điểm thi của sinh viên

```
// hàm cài đặt
void xuatmangdiem (Diem sv [] , int n)
{
    for (int i=0 ; i<n ; i++)
        xuatdiem(sv[i]);
}
```

### 4.9. Tham số của hàm có kiểu cấu trúc

Ví dụ: xây dựng các hàm để nhập, sắp xếp, tìm kiếm và in bảng điểm của sinh viên. Trong đó hàm **NhapToanBoTS** và **NhapMotTS** sử dụng tham chiếu, các hàm còn lại dùng tham trị (**XuatMotTS**, **XuatToanBoTS**, **SortDiemTBTang**, **TimTheoMa**).

```
#define MAX 100
struct ThiSinh
{
    char MaTS[7];
    char HoTen[31];
    float DiemLT;
    float DiemTH;
```

```
        float DiemTB;
    };
    typedef struct ThiSinh TS;
    void NhapMotTS (TS &ts,int stt)
    {
        printf("Nhap thong tin thi sinh thu %d:\n",stt+1);
        printf("Ma so (toi da 6 ky tu): ");
        gets(ts.MaTS);
        fflush(stdin);
        printf("Ho ten (toi da 30 ky tu): ");
        gets(ts.HoTen);
        fflush(stdin);
        printf("Nhap diem ly thuyet: ");
        scanf("%f",&ts.DiemLT);
        printf("Nhap diem thuc hanh: ");
        scanf("%f",&ts.DiemTH);
        ts.DiemTB=(ts.DiemLT+ts.DiemTH)/2;
        fflush(stdin);
    }
    void NhapToanBoTS (TS A[MAX],int &n)
    {
        printf("Nhap so luong thi sinh: ");
        scanf("%d",&n);
        fflush(stdin);
        for(int i=0;i<n;i++)
            NhapMotTS(A[i],i);
    }
    void XuatMotTS (TS A[MAX],int n, int vt)
    {
        printf("%6s",A[vt].MaTS) ;
        printf("%30s",A[vt].HoTen);
        printf("%5.2f",A[vt].DiemLT);
        printf("%5.2f",A[vt].DiemTH);
        printf("%5.2f",A[vt].DiemTB);
        printf("\n");
    }
    void XuatToanBoTS (TS A[MAX],int n)
    {
        for(int i=0;i<n;i++)
            XuatMotThiSinh (A, n, i);
    }
    void SortDiemTBTang(TS A[MAX],int n)
    {
        for(int i = 0 ; i < n - 1 ; i++)
            for(int j = i + 1; j<n; j++)
                if(A[i].DiemTB<A[j].DiemTB)
                {
                    TS tam= A[i];
                    A[i]=A[j];
                    A[j]=tam;
                }
    }
}
```

```

void TimTheoMa(TS A[MAX],int n)
{
    char Ma[7];
    printf("Nhap ma so can tim: ");
    gets(Ma);
    int i=0;
    while( (i<n) && (strcmp(A[i].MaTS,Ma)!=0))
        i++;
    if (i<n)
    {
        printf("Thong tin ve thi sinh co ma so %s:\n",Ma);
        XuatMotThiSinh (A, n, i);
    }
    else
        printf("KHONG TIM THAY");
}

```

#### 4.10. Phần thực hành

##### 4.10.1. Mục tiêu

- Biết Nhập Xuất dữ liệu có cấu trúc cho một phần tử ;
- Biết Nhập, Xuất dữ liệu có cấu trúc và lưu trên mảng một chiều;
- Cách tìm kiếm và sắp xếp dữ liệu có cấu trúc trên mảng một chiều ;
- Đi sâu vào các giải thuật trên mảng một chiều như tìm kiếm, sắp xếp, thêm phần tử, xóa phần tử...

##### 4.10.2. Bài thực hành số 3.1

- (1)- Viết hàm Nhập vào một phân số
- (2)- Viết hàm Nhập vào một dãy phân số
- (3)- Viết hàm xuất một phân số
- (4)- Viết hàm xuất một dãy phân số
- (5)- Viết hàm tìm phân số lớn nhất trong dãy phân số
- (6)- Viết hàm tính tổng các phân số có trong dãy
- (7)- Sắp xếp dãy phân số theo thứ tự tăng dần

#### Hướng dẫn

- Xây dựng struct cho phân số
- Viết hàm nhập 1 phân số
- Viết hàm xuất 1 phân số
- Viết hàm main () kết 2 hàm trên . chạy ổn định rồi mới viết tiếp hàm khác
- Viết hàm nhập số phần tử
- Viết hàm nhập vào một dãy phân số

- Viết hàm xuất dãy phân số vừa nhập
- Kết 3 hàm trên trong hàm main ()

**Chú ý:** Viết xong hàm nào kết ngay hàm đó trong hàm main()

#### 4.10.3. Bài thực hành số 3.2

- (1)- Viết hàm nhập dữ liệu cho một sinh viên. Thông tin về một sinh viên gồm có :
  - Họ (là chuỗi tối đa 20 ký tự);
  - Tên (là chuỗi tối đa 10 ký tự);
  - Mã số sinh viên (chuỗi 10 ký tự).
  - Ngày tháng năm sinh (theo kiểu ngày tháng năm).
  - Giới tính (Nam hoặc Nữ).
  - Lớp (chuỗi 7 ký tự trong đó 2 ký tự đầu là năm vào học, 1 ký tự tiếp là bậc học (D: Đại học, C: Cao đẳng) , 2 ký tự tiếp là ngành học (TH : Tin Học, KT : Kế Toán, QT : Điện tử , ĐT : Điện tử....)
  - Điểm toán, điểm lý, điểm tin. (Kiểu số thực)
- (2)- Viết hàm xuất dữ liệu một sinh viên với thông tin vừa nhập ở trên
- (3)- Viết hàm nhập danh sách sinh viên, lưu trên mảng một chiều.
- (4)- Viết hàm xuất danh sách sinh viên.
- (5)- Xuất thông tin của sinh viên có mã sinh viên là “ X”.
- (6)- Xuất danh sách sinh viên thuộc ngành công nghệ thông tin.
- (7)- Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin.
- (8)- Sắp xếp danh sách sinh viên theo tên
- (9)- Sắp xếp danh sách sinh viên theo MSSV
- (10)- Sắp xếp danh sách sinh viên theo điểm toán .

#### HƯỚNG DẪN

- Xây dựng struct cho Sinh viên
- Viết hàm nhập 1 sinh viên
- Viết hàm xuất 1 sinh viên
- Viết hàm main () kết 2 hàm trên . chạy ổn định rồi mới viết tiếp hàm khác
- Viết hàm nhập số phần tử
- Viết hàm nhập vào danh sách sinh viên lưu trên mảng một chiều
- Viết hàm xuất danh sách sinh viên vừa nhập
- Kết 3 hàm trên trong hàm main ()
- Xuất thông tin của sinh viên có mã sinh viên là “ X “.
- Xuất danh sách sinh viên thuộc ngành công nghệ thông tin.
- Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin.
- Sắp xếp danh sách sinh viên theo tên Sắp xếp danh sách sinh viên theo MSSV
- Sắp xếp danh sách sinh viên theo điểm toán .

#### 4.11. Bài tập (sinh viên tự thực hiện)

- (1)- Viết chương trình tính khoảng cách giữa 2 ngày.  
Ví dụ: ngay1= 15/11/2007

ngày2= 11/9/2009

in ra ngay 15/11/2007 va ngay 11/8/2009 cach nhau 26 ngay 9 thang va 1 nam

- (2)- Viết chương trình sử dụng con trỏ cấu trúc để hiển thị giờ, phút, giây ra màn hình, và tính khoảng cách giữa 2 mốc thời gian (quy ra thời gian chênh lệch tính theo giây).
- (3)- Viết chương trình khai báo kiểu dữ liệu thể hiện một số phức. Sử dụng kiểu này để viết hàm tính tổng, hiệu, tích của hai số phức.
- (4)- Viết chương trình khai báo kiểu dữ liệu để biểu diễn một phân số. Hãy viết hàm thực hiện những công việc sau:
  - Tính tổng, hiệu, tích, thương hai phân số.
  - Rút gọn phân số.
  - Quy đồng hai phân số.
  - So sánh hai phân số.
- (5)- Viết chương trình khai báo kiểu dữ liệu để biểu diễn một hỗn số. Hãy viết hàm thực hiện những công việc sau:
  - Đổi hỗn số sang phân số
  - Tính tổng, tích hai hỗn số
- (6)- Viết chương trình tạo một mảng các phân số. Hãy viết hàm thực hiện các công việc sau:
  - Tính tổng tất cả các phân số (kết quả dưới dạng phân số tối giản)
  - Tìm phân số lớn nhất, phân số nhỏ nhất.
  - Sắp xếp mảng tăng dần.
- (7)- Tổ chức dữ liệu để quản lý sinh viên bằng cấu trúc mẫu tin trong một mảng N phần tử, mỗi phần tử có cấu trúc như sau:

|                 |                                        |
|-----------------|----------------------------------------|
| - Mã sinh viên. | - Năm sinh.                            |
| - Tên.          | - Điểm toán, lý, hoá, điểm trung bình. |

Viết chương trình thực hiện những công việc sau:

  - Nhập danh sách các sinh viên cho một lớp học.
  - Xuất danh sách sinh viên ra màn hình.
  - Tìm sinh viên có điểm trung bình cao nhất.
  - Sắp xếp danh sách lớp theo thứ tự tăng dần của điểm trung bình.
  - Sắp xếp danh sách lớp theo thứ tự giảm dần của điểm toán.
  - Tìm kiếm và in ra các sinh viên có điểm trung bình lớn hơn 5 và không có môn nào dưới 3.
  - Tìm sinh viên có tuổi lớn nhất.
  - Nhập vào tên của một sinh viên. Tìm và in ra các thông tin liên quan đến sinh viên đó (nếu có).
- (8)- Tổ chức dữ liệu quản lý danh mục các bộ phim VIDEO, các thông tin liên quan đến bộ phim này như sau:

|                                               |                           |
|-----------------------------------------------|---------------------------|
| - Tên phim (tựa phim).                        | - Tên diễn viên nữ chính. |
| - Thể loại (3 loại : hình sự, tình cảm, hài). | - Năm sản xuất.           |



- Tên đạo diễn.
- Hãng sản xuất
- Tên diễn viên nam chính.

Viết chương trình thực hiện những công việc sau:

- Nhập vào bộ phim mới cùng với các thông tin liên quan đến bộ phim này.
- Nhập một thể loại: In ra danh sách các bộ phim thuộc thể loại này.
- Nhập một tên nam diễn viên. In ra các bộ phim có diễn viên này đóng.
- Nhập tên đạo diễn. In ra danh sách các bộ phim do đạo diễn này dàn dựng.

(9)- Một thư viện cần quản lý thông tin về các đầu sách. Mỗi đầu sách bao gồm các thông tin sau: MaSach (mã số sách), TenSach (tên sách), TacGia (tác giả), SL (số lượng các cuốn sách của đầu sách). Viết chương trình thực hiện các chức năng sau:

- Nhập vào một danh sách các đầu sách (tối đa là 100 đầu sách)
- Nhập vào tên của quyển sách. In ra thông tin đầy đủ về các sách có tên đó, nếu không có thì tên của quyển sách đó thì báo là :Không Tìm Thấy.
- Tính tổng số sách có trong thư viện.

(10)- Viết chương trình tạo một mảng danh sách các máy tính của một cửa hàng, thông tin của một máy tính bao gồm:

- Loại máy                      - Nơi sản xuất                      - Thời gian bảo hành
- Viết hàm nhập một dãy các loại máy tính có thông tin như trên.
- Hãy viết hàm thống kê xem có bao nhiêu máy có thời gian bảo hành là 1 năm.
- In ra danh sách các máy tính có xuất xứ từ Mỹ.

(11)- Để lắp ráp một máy vi tính hoàn chỉnh cần phải có tối thiểu 10 linh kiện loại A và có thể lắp bổ sung thêm vào khoảng tối đa 8 linh kiện loại B. Tại một cửa hàng vi tính cần quản lý bán hàng các loại linh kiện tại cửa hàng. Thông tin về một loại linh kiện gồm có: Tên linh kiện, quy cách, loại, đơn giá loại 1 (chất lượng tốt – số nguyên), đơn giá loại 2 (chất lượng thường – số nguyên). Viết chương trình thực hiện những công việc sau:

- Nhập vào thông tin về các linh kiện có ở cửa hàng.
- Xuất danh sách các linh kiện đã nhập theo thứ tự tăng dần của loại linh kiện và tên linh kiện.
- Cho biết đã có đủ 10 linh kiện loại A cần thiết lắp ráp máy hay chưa?

(12)- Một cửa hàng cần quản lý các mặt hàng, thông tin một mặt hàng bao gồm:

- Mã hàng.                      - Số lượng.                      - Số lượng tồn.
- Tên mặt hàng.                      - Đơn giá.                      - Thời gian bảo hành (tính theo tháng).
- Hãy nhập vào một danh sách các mặt hàng.
- Tìm mặt hàng có số lượng tồn nhiều nhất.
- Tìm mặt hàng có số lượng tồn ít nhất.
- Tìm mặt hàng có giá tiền cao nhất.
- In ra những mặt hàng có thời gian bảo hành lớn hơn 12 tháng.
- Sắp xếp các mặt hàng theo thứ tự tăng dần của số lượng tồn.

(13)- Viết chương trình quản lý hồ sơ nhân viên trong một công ty, chương trình thực hiện những công việc sau:

- Họ và tên.                      - Ngày sinh.                      - Lương cơ bản.                      - Thưởng.
- Phái.                              - Địa chỉ.                              - Bảo hiểm xã hội.                      - Phạt.
- Lương thực lĩnh = lương cơ bản + thưởng – BH xã hội – phạt.
- Nhập vào hồ sơ của các nhân viên trong công ty.
- Xuất danh sách các nhân viên theo lương thực lĩnh giảm dần bằng 2 cách sau:
  - Cấp phát vùng nhớ tĩnh.
  - Cấp phát vùng nhớ động.

---

## 5 KIỂU CON TRỎ (Pointer)

Sau khi học xong bài này, sinh viên có thể

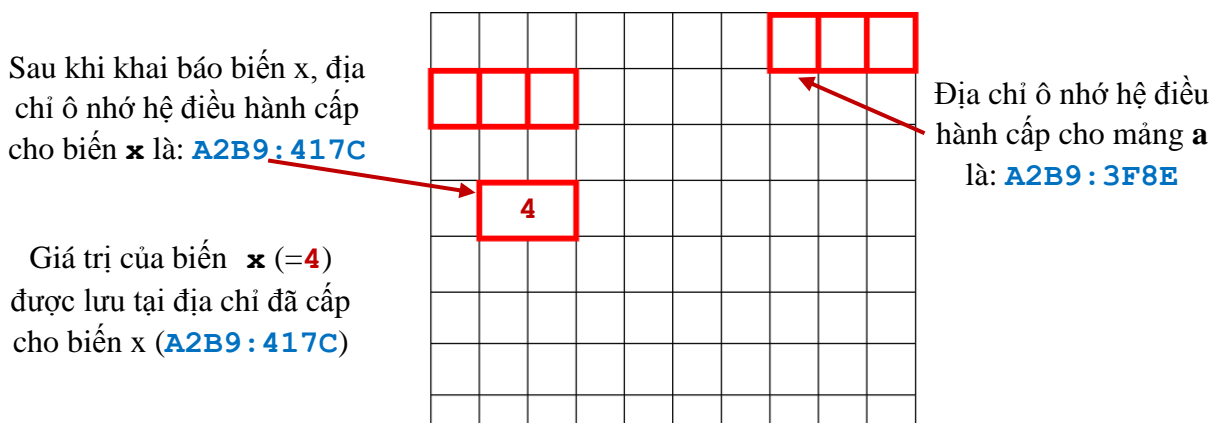
- Hiểu khái niệm về con trỏ;
- Biết cách khai báo và sử dụng biến kiểu con trỏ;
- Biết xử lý các phép toán trên mảng một chiều theo kiểu con trỏ;
- Biết xử lý các phép toán trên mảng hai chiều theo kiểu con trỏ;
- Đi sâu vào các giải thuật trên mảng 1 chiều, 2 chiều như tìm kiếm, sắp xếp, thêm phần tử,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

### 5.1. Khái niệm

#### 5.1.1. Biến tĩnh

- Các biến đã biết và sử dụng trước đây đều có 3 chi tiết liên quan:
  - Kiểu dữ liệu của biến.
  - Giá trị lưu trong biến.
  - Địa chỉ lưu trữ của biến trong bộ nhớ.
- Việc đặt tên biến giúp cho chương trình dễ hiểu nhờ tên của biến nói lên ý nghĩa sử dụng của biến trong chương trình. Mỗi tên biến như vậy sẽ tương ứng với một vị trí nào đó trong ô nhớ, và việc xác định sự tương ứng này sẽ do trình biên dịch và máy tính hoàn tất mỗi khi chương trình được thực hiện.
- 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; char a[6];`



- Khảo sát chương trình sau:
 

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num = 22;
    printf("Gia tri luu trong bien num la: %d\n", num);
    printf("So byte bo nho danh cho bien nay la: %d\n", sizeof(num));
    printf("Dia chi cua o nho danh cho bien num la: %d", &num);
}
```

- Chương trình trên khi thực hiện sẽ in lên màn hình:
 

```
Gia tri luu trong bien num la: 22
So byte trong bo nho danh cho bien nay la: 2
Dia chi cua o nho danh cho bien num la: 2FFDBC
```
- Trong ví dụ trên:
  - Địa chỉ của biến *num* có thể thay đổi khi thực hiện trên máy tính khác hoặc ở những lần thực hiện sau đó.
  - Số byte có thể là 4 nếu sử dụng Visual Studio .NET
- Một số hạn chế có thể gặp phải khi sử dụng các biến tĩnh:
  - Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
  - Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

### 5.1.2. Biến động

- 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:
  - Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
  - Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
  - 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:
  - 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.
  - Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

## 5.2. Khai báo và sử dụng biến con trỏ

### 5.2.1. Khai báo biến con trỏ

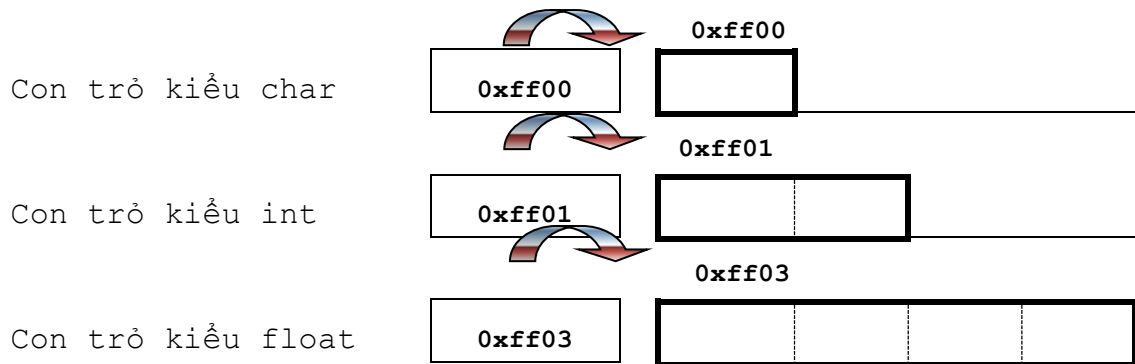
- Cú pháp: **<Kiểu\_dữ\_liệu> \* <Tên\_con\_trỏ>**
- Ý nghĩa: Khai báo một biến có tên là **Tên\_con\_trỏ** dùng để chứa địa chỉ của các biến có thuộc **Kiểu\_dữ\_liệu**.
- Ví dụ 1: Khai báo 2 biến **a, b** có kiểu **int** và 2 biến **pa, pb** là 2 biến con trỏ
 

```
int a, b, *pa, *pb;
```
- Ví dụ 2: Khai báo biến **f** kiểu **float** và biến **pf** là con trỏ **float**

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

  - Khai báo `int *px` chỉ ra ba điều:
    - `px` là một biến con trỏ.
    - vùng nhớ được lưu trong `px` phải là vùng nhớ lưu trữ một số nguyên kiểu `int` (hay biến mà `px` trỏ đến phải là một biến kiểu `int`).
    - nếu `px` xuất hiện trong ngữ cảnh `*px` thì nó cũng được coi là tương đương với việc dùng biến có kiểu `int`.
  - Một biến con trỏ có kích thước hai byte và phụ thuộc vào khai báo ban đầu nó sẽ chứa địa chỉ của vùng nhớ 1byte, 2 byte, 4byte... Trong thực tế dù là con trỏ kiểu `char`, `int`, `float`, hay `double`

thì con trỏ cũng chỉ chứa ô nhớ đầu tiên của vùng nhớ mà nó chỉ tới, nhưng nhờ vào sự khai báo này máy tính sẽ biết cần phải lấy bao nhiêu ô nhớ khi truy xuất tới con trỏ này. Xem hình minh họa sau:



### 5.2.2. Biến tham chiếu và biến con trỏ

Biến tham chiếu là một biến khi khai báo có thêm dấu **&** phía trước. Biến này cũng là một loại con trỏ nhưng có nhiều hạn chế so với con trỏ.

- Biến tham chiếu dùng để tham chiếu tới địa chỉ của một biến và chỉ một mà thôi (địa chỉ lưu trong biến tham chiếu không thể thay đổi được).
- Địa chỉ mà biến này tham chiếu tới phải được khởi tạo ngay tại thời điểm khai báo ngay từ đầu.
- Sau khi đã tham khảo tới một biến thì việc sử dụng biến tham chiếu giống như một biến thông thường và những lệnh tác động lên biến tham chiếu này sẽ ảnh hưởng tới biến mà nó tham chiếu tới.

- Ví dụ:

```
int main()
{
    int n=123, &x=n;
    printf("Gia tri trong vung nho ma x tham chieu toi: %d",x); //123
    x=100;
    printf("\nGia tri cua n hien tai la: %d",n);                //100
    return 0;
}
```

Với biến tham chiếu `x` sau khi đã khởi tạo ta chỉ có thể truy xuất đến giá trị lưu tại vùng nhớ mà nó tham chiếu tới và trong trường hợp này ta cũng không cần ghi dấu **\*** bên cạnh tên biến.

### 5.2.3. Các thao tác trên con trỏ

#### 5.2.3.1. 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:**

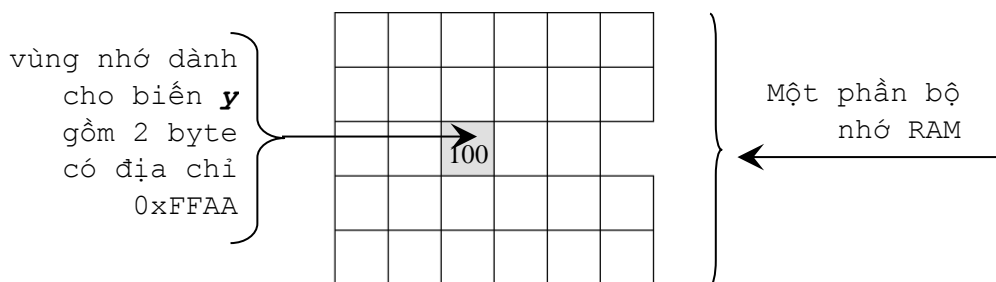
```
void main()
{
    int x=100;
    int *px;
    printf("Bien x=%d, duoc luu tru tai dia chi: %x\n", x, &x);
    px=&x;
    printf("Bien px duoc luu tru tai dia chi: %x\n",px);
}
```

```
printf("Biến px đang tham chiếu đến địa chỉ: %x\n", &px);
printf("Giá trị đang lưu tại địa chỉ %x: %d", px, *px);
}
```

- **Các kiểu dùng sai:** Phép toán **&** chỉ áp dụng được cho các biến và phần tử mảng, do đó các trường hợp dùng sau đây là không hợp lệ:
  - Kết cấu kiểu **&(x+1)** và **&3**.
  - Lấy địa chỉ của biến register cũng là sai.

### 5.2.3.2. Lấy giá trị của biến con trỏ chỉ tới

- Khi toán tử này đi trước một tên biến con trỏ (ví dụ như **\*numaddr**, **\*chrpoint**) có nghĩa là nói đến giá trị đang lưu tại vùng nhớ có địa chỉ đang lưu trong biến con trỏ.
- Ví dụ: giả sử có một biến con trỏ **y**, giá trị của biến hiện là 0xFFAA (địa chỉ của ô nhớ thứ FFAA trong RAM) giá trị đang lưu trữ tại ô nhớ này là 100. Hình minh họa bộ nhớ lúc này như sau:



như vậy lúc này: giá trị của **y** là 0xFFAA.

giá trị của **\*y** là 100 (giá trị lưu tại vùng nhớ 0xFFAA).

Đây là một cách lấy giá trị gián tiếp, theo cách này máy tính phải dò đến hai địa chỉ thì mới nhận được giá trị 100. (đầu tiên dò đến địa chỉ chứa trong **y**, sau đó từ địa chỉ này mới dò đến ô nhớ có địa chỉ tương ứng để nhận địa chỉ).

- Công dụng của phép toán **\***
  - Làm việc trên các phần tử của mảng.
  - Tạo và xóa các biến động khi chương trình đang thực hiện.
- Để truy nhập tới biến ta có thể thực hiện 1 trong 2 cách:
  - Truy nhập trực tiếp tới biến.
  - Truy nhập gián tiếp thông qua biến con trỏ.
- **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("\nGiá trị của biến a=%d \nGiá trị của biến 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 đổi giá trị của biến a (hiện con trỏ pa đang chỉ đến)*/
*pb=20; /*Thay đổi giá trị của biến b (hiện con trỏ pb đang chỉ đến)*/
printf("\nGia tri moi cua bien a=%d \n
Gia tri moi cua bien b=%d ",a,b); /* a,b thay đổi theo*/
getch();
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)

```

#### - Ví dụ 3

```

void main()
{
    int x=7, y=5;
    int *px, *py;
    px=&x;
    printf("y= %d\n",y); //y=5
    printf("px chua gia tri %d\n", *px); //7
    printf("px= %x\n", px); //px=9ffbe8
    y=*px; //y sẽ chứa giá trị đang
    //luu giữ tại địa chỉ mà biến con trỏ px chỉ đến (= 7)*/
    printf("Dia chi y= %d\n",y); //aafb8
    printf("y= %d\n",y); //7
}

```

#### 5.2.3.3. Độ ưu tiên của các phép toán một ngôi & và \*

Các phép toán một ngôi & và \* có mức ưu tiên cao hơn các phép toán số học.

### 5.3. Các phép toán trên con trỏ

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

- Phép gán
- Phép tăng giảm địa chỉ
- Phép truy nhập bộ nhớ
- Phép so sánh

#### 5.3.1. Phép gán

Con trỏ dùng để lưu địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán địa chỉ cho con trỏ chỉ có thể thực hiện được khi kiểu địa chỉ phù hợp với kiểu con trỏ.

##### 5.3.1.1. Lưu trữ địa chỉ của một biến vào một biến con trỏ đã được khai báo phù hợp

```
int m;
```

```
int *d= &m;          // lưu trữ địa chỉ biến m vào con trỏ d
float a[30], *pa, (*pm)[30];
pa=a;                // hợp lệ
pm=&a;                // hợp lệ
pm=a;                // KHÔNG hợp lệ  $\Rightarrow$  bo lỗi khi biên dịch
```

#### 5.3.1.2. Xuất giá trị

```
int *px, x=5;
px=&x;
printf("%5d", *px);
```

#### 5.3.1.3. Gán giá trị cho biến từ biến con trỏ: (biến con trỏ xuất hiện bên về phải của biểu thức)

```
int x=4;
int y=*px+1;        //  $\Leftrightarrow y = 4+1=5$ 
d=sqrt((double) *px);

/*  $\Leftrightarrow d = \sqrt{x} = \sqrt{4} = 2$ . Do hàm sqrt chỉ nhận tham số kiểu double nên
phải thực hiện ép kiểu */
```

#### 5.3.1.4. Gán giá trị cho biến con trỏ: (biến con trỏ xuất hiện bên về trái của biểu thức)

```
*px=0;              // x = 0.
*px+=1;             // tăng giá trị của biến x lên thêm 1 đơn vị (x=1)
(*px)++;            // tăng giá trị của biến x lên thêm 1 đơn vị (x=2)
```

Lưu ý: Các dấu ngoặc đơn ở câu lệnh cuối là cần thiết, nếu không thì biểu thức sẽ tăng px thay cho tăng ở biến mà nó trỏ tới vì phép toán một ngôi như \* và ++ được tính từ phải sang trái.

#### 5.3.1.5. Gán giá trị giữa 2 biến con trỏ của cùng kiểu dữ liệu: (biến con trỏ xuất hiện ở cả 2 vế của biểu thức)

```
int *px, *py;
py=px;              /* lệnh này sẽ sao nội dung của px vào py, nghĩa là làm
cho py trỏ tới nơi mà px trỏ */
```

#### 5.3.1.6. Ép kiểu

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

### 5.3.2. Phép tăng giảm địa chỉ

- Ta có thể cộng (+) hoặc 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, *pa, *pc;
px = &x[10];
pa = &x[0];
pc = &x[4];
```

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.

- **Ví dụ 1:** (dùng trên mảng 1 chiều)

```
float x[30], *px;
px=&x[10]; // px trỏ tới phần tử x[10]
```

cho con trỏ px là con trỏ float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ 4 byte, nên các phép tăng giảm địa chỉ được thực hiện trên 4 byte. Vì thế:

```
px+i // px trỏ tới phần tử x[10+i]
px-i // pxtrỏ tới phần tử x[10-i]
```

- **Ví dụ 2:** (dùng trên mảng 2 chiều) Giả sử ta có khai báo : `float b[40][50];`

Khai báo trên cho ta một mảng b gồm các dòng 50 phần tử kiểu số thực. Kiểu địa chỉ của b là  $50 \times 4 = 200$  byte.

Do vậy:

b trỏ tới đầu dòng thứ nhất ( phần tử b[0][0]).

b+1 trỏ tới đầu dòng thứ hai ( phần tử b[1][0]).

...

b+i trỏ tới đầu dòng thứ i ( phần tử b[i][0]).

### 5.3.3. 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

### 5.3.4. Phép so sánh

Dùng cho phép so sánh 2 con trỏ cùng kiểu

- $p1 < p2$  nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới
- $p1 = p2$  nếu địa chỉ p1 trỏ tới bằng địa chỉ p2 trỏ tới
- $p1 > p2$  nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

### 5.3.5. Một số ví dụ

#### 5.3.5.1. Ví dụ 1: Phân biệt giữa địa chỉ và giá trị của biến

```
int main()
{
    int *addr, n=158, m=22;
    addr=&n;
    printf("Dia chi bien addr dang tro toi: %u\n",addr);
    printf("Gia tri tai vung nho addr dang tro toi: %d\n", *addr);
    addr=&m;
    printf("Dia chi bien addr dang tro toi: %u\n",addr);
    printf("Gia tri tai vung nho addr dang tro toi: %d\n", *addr);
    return 0;
}
```

#### 5.3.5.2. Ví dụ 2: Đoạn chương trình tính tổng các số thực dùng phép so sánh con trỏ

```
int main()
{
    float a[5]={2,4,6,8,10}, *p, *pcuoi, tong=0.0;
    int s=0, n=5;

    pcuoi=a+n-1; //pcuoi chỉ đến phần tử cuối của mảng
    for (p=a ; p<=pcuoi ; p++)
        s+=*p;
    printf("Tong cac phan tu co trong mang= %d",s);
    return 0;
}
```

#### 5.3.5.3. Ví dụ 3: Dùng con trỏ char để tách các byte của một biến nguyên

```
int main()
{
    unsigned int n=16689; /* 01000001 00110001*/
    char *pc; //A 1
    pc=(char*) (&n);
    printf("%c", *pc); // 1
    pc++;
    printf("%c", *pc); //A
    return 0;
}
```

### 5.4. 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**.
  - (i). malloc
  - (ii). calloc
  - (iii). realloc
  - (iv). new
- Khi sử dụng con trỏ để xin cấp phát bộ nhớ thì khi dùng xong người lập trình bắt buộc phải trả lại bộ nhớ. Để thu hồi bộ nhớ ta có thể dùng hàm **free** hoặc toán tử **delete**.
- Lưu ý khi giải phóng bộ nhớ:
  - Sử dụng hàm **free**: khi sử dụng các hàm **malloc**, **calloc**, **realloc** cấp phát bộ nhớ cho 1 biến con trỏ.

- Sử dụng toán tử **delete**: sử dụng toán tử **new** để xin cấp phát bộ nhớ.

### 5.4.1. Các hàm cấp phát vùng nhớ

#### 5.4.1.1. 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;
}
```

#### 5.4.1.2. Hàm calloc

- **Cú pháp:** `void *calloc(size_t nItems, size_t size);`
- **Giải thích:** 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(void)
{
    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**

(i). Hàm **calloc** cấp phát vùng nhớ và khởi tạo tất cả các bit trong vùng nhớ mới cấp phát về 0.

(ii). Hàm **malloc** chỉ cấp phát vùng nhớ.

#### 5.4.1.3. 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 x 2) byte*/
```

#### 5.4.2. Toán tử thu hồi bộ nhớ động

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

#### 5.4.3. Toán tử sizeof

- Toán tử **sizeof** cho ta biết kích thước (tính theo byte) của một kiểu dữ liệu chuẩn hay một đối tượng dữ liệu.
- Kiểu dữ liệu có thể là kiểu chuẩn (như **int**, **float**, ...) hay kiểu dữ liệu được định nghĩa trong chương trình (như **typedef**, **enum**, **struct**, **union**, ...). Đối tượng dữ liệu bao gồm tên biến, tên mảng, biến **struct**, ...

- Khai báo: `sizeof (<đối tượng dữ liệu>)`  
hoặc `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

```

## 5.5. Con trỏ và mảng một chiều

### 5.5.1. Nhận xét

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.

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

|                 |          |             |      |      |      |      |      |      |     |        |  |
|-----------------|----------|-------------|------|------|------|------|------|------|-----|--------|--|
|                 | <b>p</b> | <b>1024</b> |      |      |      |      |      |      |     |        |  |
| STT phần tử     | 0        | 1           | 2    | 3    | 4    | 5    | 6    | 7    | 8   | n-1    |  |
| Sử dụng mảng    | a[0]     | a[1]        | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | ... | a[n-1] |  |
| Sử dụng con trỏ | p        | p+1         | p+2  | p+3  | p+4  | p+5  | p+6  | p+7  | ... | p+n-1  |  |

### 5.5.2. 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>
/* 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]);
    }
}
/* 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);
    }
}

```

### 5.5.3. 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,

**<Vi tri>** là 1 biểu thức số nguyên.

- **Ví dụ 1:** Giả sử có khai báo:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
int main()
{
    int *a;
    clrscr();
    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));
    getch();
    return 0;
}
```

Kết quả chương trình:



- Cũng có thể áp dụng các phép toán tăng/giảm (++/--) một ngôi trên biến con trỏ.

**Ví dụ 2:**

```
int main()
{
    const int SIZE=5;
    int i, *p, Arr[SIZE]={98,87,76,65,54};
    p=&Arr[0]; // point=Arr
    printf("DUYET MANG THEO NHIEU CACH KHAC NHAU");
    printf("\n(a).-Dua tren ten mang:");
    for (i=0;i<SIZE;i++)
        printf ("%5d",*(Arr+i));
    printf("\n(b).-Dua tren dia chi co dinh ma bien con tro
  p dang giu:");
    for (i=0;i<SIZE;i++)
        printf ("%5d",*(p+i));
    printf("\n(c).-Dua tren viec thay doi dia chi trong bien
  con tro p qua moi lan lap");
    for (i=0;i<SIZE;i++)
        printf ("%5d",*p++);
    return 0;
}
```

- **Ví dụ 3:** tính tổng các giá trị có trong mảng

- Cách 1:

```
int main()
```

```

{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    for (i=0;i<SIZE;++i)
        tong+=A[i];
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}

```

- Cách 2:

```

int main()
{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    p=A;
    for (i=0;i<SIZE;++i)
        tong+=p[i];
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}

```

- Cách 3

```

int main()
{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    p=A;
    for (i=0;i<SIZE;++i)
        tong+=*(p+i);
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}

```

- Chú ý: Mảng một chiều và con trỏ tương ứng phải cùng kiểu.

#### 5.5.4. Minh họa các thao tác trên mảng một chiều bằng phương pháp con trỏ

##### Yêu cầu:

- Viết chương trình nhập vào một mảng số nguyên gồm n phần tử ( $n \leq 100$ ).
- Xuất ra mảng số nguyên vừa nhập.
- Tính tổng các phần tử có trong mảng.

```

int *p // khai báo biến con trỏ p để quản lý mảng n phần tử
p = (int*) malloc (n); //xin cấp phát (n*2) byte cho biến p quản lý.

```

```

// Hàm cài đặt
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
// hàm nhập số phần tử của mảng
void nhapsopt (int &n)
{
    do
    { printf (" nhap so phan tu cua mang : ");
      scanf(" %d ", &n);
      if (n<=0)

```

```

        printf (" nhập sai, nhập lại ");
    } while(n<=0);
}

// hàm xin cấp phát bộ nhớ cho con trỏ p
void capphat (int *p , int n)
{
    p = (int*) calloc (n , sizeof (int));
    if (p==NULL)
    {   printf(" ko du bo nho");
        getch();
        exit(1);
    }
}

// hàm nhập mảng dùng phương pháp con trỏ
void nhapmang (int * a , int n)
{
    for (int i=0 ; i<n ; i++)
    { printf(" nhập a[%d]:", i);
        scanf (" %d ", (a+ i));
    }
}

// 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));
}

// hàm main () kết các hàm đã định nghĩa ở trên
void main()
{
    int *a, n;
    nhapsopt (n);
    capphat(a,n);
    nhapmang(a , n);
    xuatmang(a , n);
    free(a);
    getch();
}

```



## 5.6. Con trỏ và mảng hai chiều

### 5.6.1. Nhận xét

- Giả sử, với  $m=3$  và  $n=4$ , ta có số thứ tự các phần tử sẽ có dạng như sau:

|                   |   |   |   |    |    |
|-------------------|---|---|---|----|----|
| <b>*a</b><br>1024 |   | 0 | 1 | 2  | 3  |
| 0                 | → | 0 | 1 | 2  | 3  |
| 1                 |   | 4 | 5 | 6  | 7  |
| 2                 |   | 8 | 9 | 10 | 11 |

Do mảng hai chiều được lưu trữ liên tục nhau trong bộ nhớ và theo từng dòng từ trái qua phải và từ trên xuống dưới. Vì vậy nếu sử dụng con trỏ để truy xuất ma trận thì xem như đang thao tác trên mảng 1 chiều.

|                 |         |         |         |         |         |         |         |         |         |         |         |         |
|-----------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| STT phần tử     | 0       | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      | 11      |
| Sử dụng mảng    | a[0][0] | a[0][1] | a[0][2] | a[0][3] | a[1][0] | a[1][1] | a[1][2] | a[1][3] | a[1][0] | a[1][1] | a[1][2] | a[1][3] |
| Sử dụng con trỏ | *a      | *(a+1)  | *(a+2)  | *(a+3)  | *(a+4)  | *(a+5)  | *(a+6)  | *(a+7)  | *(a+8)  | *(a+9)  | *(a+10) | *(a+11) |

- Tổng quát, số thứ tự các phần tử trong ma trận  $m$  dòng  $\times$   $n$  cột được quy ước như sau:

|                   |   |               |               |               |     |                   |
|-------------------|---|---------------|---------------|---------------|-----|-------------------|
| <b>*a</b><br>1024 |   | 0             | 1             | 2             | ... | $n-1$             |
| 0                 | → | $(0*n)+0$     | $(0*n)+1$     | $(0*n)+2$     | ... | $(0*n)+(n-1)$     |
| 1                 |   | $(1*n)+0$     | $(1*n)+1$     | $(1*n)+2$     | ... | $(1*n)+(n-1)$     |
| 2                 |   | $(2*n)+0$     | $(2*n)+1$     | $(2*n)+2$     | ... | $(2*n)+(n-1)$     |
| ...               |   | ...           | ...           | ...           | ... | ...               |
| $m-1$             |   | $((m-1)*n)+0$ | $((m-1)*n)+1$ | $((m-1)*n)+2$ | ... | $((m-1)*n)+(n-1)$ |

Tương tự, hình ảnh ma trận  $m*n$  lưu trong bộ nhớ do con trỏ **a** quản lý sẽ là:

|                 |         |         |         |     |            |            |     |                |          |     |              |
|-----------------|---------|---------|---------|-----|------------|------------|-----|----------------|----------|-----|--------------|
| STT phần tử     | 0       | 1       | 2       | ... | $n-1$      | $n$        | ... | $2*n-1$        | $2*n$    | ... | $m*n-1$      |
| Sử dụng mảng    | a[0][0] | a[0][1] | a[0][2] | ... | a[0][n-1]  | a[1][0]    | ... | a[1][n-1]      | a[2][0]  | ... | a[m-1][n-1]  |
| Sử dụng con trỏ | *(a)    | *(a+1)  | *(a+2)  | ... | *(a+(n-1)) | *(a+(n)+0) | ... | *(a+(n)+(n-1)) | *(a+2*n) | ... | *(a+(m*n)-1) |

Vậy, để truy cập đến phần tử a [row][col] của mảng hai chiều a, ta dùng công thức sau :

$$*(a + (row*n) + col);$$

### 5.6.2. Minh họa thao tác trên mảng hai chiều bằng phương pháp con trỏ

- Viết chương trình nhập vào một ma trận số nguyên gồm  $m$  dòng,  $n$  cột có  $m \times n$  phần tử ( $m, n \leq 100$ ).
- Xuất ra ma trận số nguyên vừa nhập.
- Tính tổng các phần tử có trong ma trận.
- Sắp xếp mảng 2 chiều tăng dần từ trái sang phải và từ trên xuống dưới.

#### 5.6.2.1. Cách 1: Sử dụng con trỏ

```
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
//Hàm cài đặt
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
// hàm nhập dòng và cột
```

```
void nhapdongcot (int &m , int &n)
{ do
    { printf (" nhap so dong: ");
      scanf(" %d ", &m);
      if (m<=0 || m >100)
          printf (" nhap sai, nhap lai ");
    } while(m<=0 || m>100);
  do
  {
      printf (" nhap so cot: ");
      scanf (" %d ", &n);
      if(n<=0 || n>100)
          printf (" nhap sai, nhap lai ") ;
    } while(n<=0 || n>100);
}

// hàm xin cấp phát bộ nhớ
void KhoiTaoMang (int *a , int m ,int n)
{
    a = (int*) calloc (m * n , sizeof (int));
    if(a==NULL)
    {
        printf(" khong du bo nho");
        getch();
        exit(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));
        }
}

// 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");
    }
}

// hàm tính tổng các phần tử trong ma trận
long tinh tong (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) ;
}
void sort(int a[][COLS], int m, int n)
{
    int i,j,tam, sl=m*n-1;
    int *p;
    p=&a[0][0]; // p chỉ vào phần tử đầu tiên của ma trận;
    for (i=0; i<sl; i++)
        for (j=i+1; j<sl; j++)
            if (*(p+i)>*(p+j))
            {
                tam    =*(p+i);
                *(p+i) =*(p+j);
                *(p+j) =tam;
            }
}

```

### 5.6.2.2. Cách 2: Sử dụng mảng các con trỏ

- Nhận xét:

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

|                  |           |           |     |             |
|------------------|-----------|-----------|-----|-------------|
| $*a \Rightarrow$ | $a[0][0]$ | $a[0][1]$ | ... | $a[0][n-1]$ |
|------------------|-----------|-----------|-----|-------------|

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

|                      |           |           |     |             |
|----------------------|-----------|-----------|-----|-------------|
| $*(a+1) \Rightarrow$ | $a[1][0]$ | $a[1][1]$ | ... | $a[1][n-1]$ |
|----------------------|-----------|-----------|-----|-------------|

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

|                      |           |           |     |             |
|----------------------|-----------|-----------|-----|-------------|
| $*(a+i) \Rightarrow$ | $a[i][0]$ | $a[i][1]$ | ... | $a[i][n-1]$ |
|----------------------|-----------|-----------|-----|-------------|

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

|                        |             |             |     |               |
|------------------------|-------------|-------------|-----|---------------|
| $*(a+m-1) \Rightarrow$ | $a[m-1][0]$ | $a[m-1][1]$ | ... | $a[m-1][n-1]$ |
|------------------------|-------------|-------------|-----|---------------|

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

|                 |               |        |          |          |     |        |
|-----------------|---------------|--------|----------|----------|-----|--------|
| <b>**a</b>      |               | 0      | 1        | 2        | ... | n-1    |
| <b>*(a)</b>     | $\Rightarrow$ | 0      | 1        | 2        | ... | n-1    |
| <b>*(a+1)</b>   | $\Rightarrow$ | N      | N+1      | N+2      | ... | 2n-1   |
| <b>*(a+2)</b>   | $\Rightarrow$ | 2n     | 2n+1     | 2n+2     | ... | 3n-1   |
| ...             |               | ...    | ...      | ...      | ... | ...    |
| <b>*(a+m-1)</b> | $\Rightarrow$ | (m-1)n | (m-1)n+1 | (m-1)n+2 | ... | (mn)-1 |

// hàm xin cấp phát bộ nhớ

```

void KhoiTao (int **a , int m ,int n)
{
    a = (int**) calloc (m, sizeof (int*));
    if (a==NULL)
    {
        printf(" ko du bo nho"); getch();
    }
}

```

```
        exit(1);
    }
    for(int i=0;i<m;i++)
    { a[ i] = (int *) calloc (n, sizeof (int));
      if (a[i] == NULL)
      { printf(" ko du bo nho");
        getch();
        exit(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));
        }
}

// 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");
    }
}

// hàm tính tổng các phần tử trong ma trận
long tinhhtong (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;
}

// hàm giải phóng bộ nhớ
void myfree (int **a , int m)
{
    for (int i=0 ; i<m ; i++)
        free (a[ i ] ) ;
    free (a);
}
```

## 5.7. Con trỏ với kiểu dữ liệu có cấu trúc (struct)

### 5.7.1. Truy cập đến các thành phần của struct

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

#### 5.7.1.1. Cách 1: cách thông thường

- **Cú pháp** <tên con trỏ> -> <tên thành phần>
- **Ví dụ:** printf("%f ", p->diem);

#### 5.7.1.2. Cách 2: dùng con trỏ

- **Cú pháp** (\*<tên con trỏ>).<tên thành phần>
- **Ví dụ:** printf("%f ", (\*p).diem);

### 5.7.2. Ví dụ

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;
};
void main()
{
    p=&x;
    struct 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 */
    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);
    getch();
}
```

### 5.7.3. Truyền structure sang 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);
void main()
{
    DIEMTHI x, *px;
    printf("\nNhap diem thi");
    nhap(px);
    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;
}
```

## 5.8. Bài thực hành có hướng dẫn

### 5.8.1. Bài thực hành 5.1

*5.8.1.1. Dùng phương pháp con trỏ làm các bài thực hành sau:*

- (i). Viết hàm nhập mảng một chiều các số nguyên gồm n phần tử ( $0 < n \leq 100$ )
- (ii). Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên
- (iii). Tính tổng các phần tử có trong mảng
- (iv). Tính tổng các phần tử chẵn có trong mảng

- (v). Tính tổng các phần tử lẻ có trong mảng
- (vi). Tính tổng các phần tử nguyên tố có trong mảng
- (vii). Tìm phần tử chẵn đầu tiên có trong mảng
- (viii). Tìm phần tử lẻ đầu tiên có trong mảng
- (ix). Tìm phần tử nguyên tố đầu tiên có trong mảng
- (x). Tìm phần tử chẵn cuối cùng có trong mảng
- (xi). Tìm phần tử chính phương cuối cùng có trong mảng
- (xii). Tìm phần tử lớn nhất có trong mảng
- (xiii). Đếm số phần tử chẵn có trong mảng
- (xiv). Đếm số phần tử lớn nhất có trong mảng
- (xv). In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (xvi). Thêm một phần tử vào đầu mảng.
- (xvii). Thêm một phần tử vào cuối mảng.
- (xviii). Thêm một phần tử vào vị trí x trong mảng.
- (xix). Xóa phần tử chẵn đầu tiên.
- (xx). Xóa tất cả các phần tử lớn nhất trong mảng
- (xxi). Sắp xếp mảng tăng dần

#### 5.8.1.2. Hướng dẫn

- Viết hàm nhập số phần tử của mảng ( $0 < n \leq 100$ ).

```
void nhapsopt (int &n)
{
    do
    {
        printf (" nhap so phan tu cua mang : ");
        scanf(" %d ", &n);
        if (n<=0 || n>100)
            printf (" nhap sai, nhap lai ");
    } while(n<=0 || n>100);
}
```

- Viết hàm xin cấp phát bộ nhớ cho con trỏ p

```
void capphat (int *p , int n)
{
    p = (int*) calloc (n , sizeof (int));
    if (p==NULL)
    {
        printf(" ko du bo nho");
        getch();
        exit(1);
    }
}
```

- Viết 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));
    }
}
```

- Viết 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));
}
```

- Hàm main () kết các hàm đã định nghĩa ở trên

```
void main()
{
    clrscr();
    int *a, n;
    nhapsort (n);
    capphat(a,n)
    nhapmang(a , n);
    xuatmang(a , n);
    free(a);
    getch();
}
```

- Viết hàm tính tổng các phần tử có trong mảng .
- Kết hàm tính tổng trong hàm main () ....

## 5.8.2. Bài thực hành số 5.2

### 5.8.2.1. Dùng phương pháp con trỏ làm các bài thực hành sau:

- (i). Nhập chuỗi
- (ii). Xuất chuỗi
- (iii). Nhập vào 2 chuỗi, xuất chuỗi theo thứ tự từ điển
- (iv). Đếm số ký tự ‘ a’ có trong chuỗi
- (v). Cắt khoảng trắng có trong chuỗi
- (vi). Đếm khoảng trắng trong chuỗi .
- (vii). Đếm số từ có trong chuỗi
- (viii). Sắp xếp chuỗi tăng dần
- (ix). Nhập 3 chuỗi, xuất chuỗi theo thứ tự từ điển.

### 5.8.2.2. Hướng dẫn

## 5.8.3. Bài thực hành số 5.3

### 5.8.3.1. Dùng phương pháp con trỏ làm các bài tập sau:

- (i). Viết hàm nhập vào số dòng, số cột ( $0 < m, n < 100$ ) .
- (ii). Viết hàm nhập vào mảng hai chiều các số nguyên gồm m dòng, n cột ( $0 < m, n < 100$ ) .
- (iii). Viết hàm xuất mảng hai chiều các số nguyên vừa nhập ở trên.
- (iv). Tính tổng các phần tử có trong mảng .
- (v). Tính tổng các phần tử chẵn có trong mảng .



- (vi). Tính tổng các phần tử nguyên tố có trong mảng .
- (vii). Tính tổng các phần tử nằm trên đường chéo chính có trong mảng .
- (viii). Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng .
- (ix). Tính tổng các phần tử nằm trên đường biên.
- (x). Tìm phần tử lớn nhất có trong mảng
- (xi). Đếm số phần tử lẻ có trong mảng
- (xii). Đếm số phần tử lớn nhất có trong mảng
- (xiii). In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (xiv). Tính tổng các phần tử nằm trên một dòng .
- (xv). Tìm dòng có tổng lớn nhất
- (xvi). Xoá dòng
- (xvii). Xoá cột
- (xviii). Sắp xếp mảng tăng dần
- (xix). Xoay mảng về trái .
- (xx). Xoay mảng về phải

### 5.8.3.2. Hướng dẫn

- Hàm nhập dòng và cột

```
void nhapdongcot (int &m , int &n)
{
    do
    {
        printf (" nhap so dong: ");
        scanf(" %d ", &m);
        if (m<=0 || m >=100)
            printf (" nhap sai, nhap lai ");
    } while(m<=0 || m>=100);
    do
    {
        printf (" nhap so cot: ");
        scanf (" %d ", &n);
        if(n<=0 || n>=100)
            printf (" nhap sai, nhap lai ") ;
    } while(n<=0 || n>100);
}
```

- Hàm xin cấp phát bộ nhớ

```
void caphat (int **a , int m ,int n)
{
    a = (int**) calloc (m, sizeof (int*));
    if(a==NULL)
    {
        printf(" ko du bo nho");
        getch();
        exit(1);
    }
    for(int i=0;i<m;i++)
    {
```

- ```

        a[ i] = (int *) calloc (n, sizeof (int));
        if (a[i] == NULL)
        {
            printf(" ko du bo nho"); getch();
            exit(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));
        }
}

```
  - 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");
    }
}

```
  - Hàm tính tổng các phần tử trong ma trận
 

```

long tinhcong (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;
}

```
  - Hàm giải phóng bộ nhớ
 

```

void myfree (int **a , int m)
{
    for (int i=0 ; i<m ; i++)
        free (a[ i ]) ;
    free (a);
}

```

#### 5.8.4. Bài thực hành số 5.4

##### 5.8.4.1. Dùng phương pháp con trỏ làm các bài tập sau:

- (i). Viết hàm nhập dữ liệu cho một sinh viên. Thông tin về một sinh viên gồm có :
  - Họ (là chuỗi tối đa 20 ký tự);

- Tên (là chuỗi tối đa 10 ký tự);
- Mã số sinh viên (chuỗi 10 ký tự).
- ngày tháng năm sinh (theo kiểu ngày tháng năm).
- Giới tính (Nam hoặc Nữ).
- Lớp (chuỗi 7 ký tự trong đó 2 ký tự đầu là năm vào học, 1 ký tự tiếp là bậc học (D: Đại học, C: Cao đẳng) , 2 ký tự tiếp là ngành học (TH : Tin Học, KT : Kế Toán, QT : Điện tử , ĐT : Điện tử....)
- điểm toán, điểm lý, điểm tin. (Kiểu số thực)

(ii). Viết hàm xuất dữ liệu một sinh viên với thông tin vừa nhập ở trên

(iii). Viết hàm nhập danh sách sinh viên, lưu trên mảng một chiều.

(iv). Viết hàm xuất danh sách sinh viên.

(v). Xuất thông tin của sinh viên có mã sinh viên là “ X “.

(vi). Xuất danh sách sinh viên thuộc ngành công nghệ thông tin.

(vii). Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin.

(viii). Sắp xếp danh sách sinh viên theo tên

(ix). Sắp xếp danh sách sinh viên theo MSSV

(x). Sắp xếp danh sách sinh viên theo điểm toán .

#### 5.8.4.2. Hướng dẫn

- Xây dựng struct cho Sinh viên
- Viết hàm nhập 1 sinh viên
- Viết hàm xuất 1 sinh viên
- Viết hàm main () kết 2 hàm trên . chạy ổn định rồi mới viết tiếp hàm khác
- Viết hàm nhập số phần tử
- Viết hàm nhập vào danh sách sinh viên lưu trên mảng một chiều
- Viết hàm xuất danh sách sinh viên vừa nhập
- Kết 3 hàm trên trong hàm main ()
- Xuất thông tin của sinh viên có mã sinh viên là “ X “.
- Xuất danh sách sinh viên thuộc ngành công nghệ thông tin.
- Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin.
- Sắp xếp danh sách sinh viên theo tên Sắp xếp danh sách sinh viên theo MSSV
- Sắp xếp danh sách sinh viên theo điểm toán .

### 5.9. Bài tập (sinh viên tự thực hiện)

Dùng phương pháp con trỏ làm các bài tập sau:

- (1)- Viết hàm nhập vào một mảng một chiều các số nguyên gồm n phần tử ( $0 < n < 100$ )
- (2)- Viết hàm nhập vào một mảng một chiều các số thực gồm n phần tử ( $0 < n < 100$ )
- (3)- Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên

- (4)-Viết hàm xuất mảng số thực n phần tử vừa nhập ở trên
- (5)-Tính tổng các phần tử có trong mảng
- (6)-Tính tổng các phần tử chẵn có trong mảng
- (7)-Tính tổng các phần tử lẻ có trong mảng
- (8)-Tính tổng các phần tử nguyên tố có trong mảng
- (9)-Tìm phần tử chẵn đầu tiên có trong mảng
- (10)-Tìm phần tử lẻ đầu tiên có trong mảng
- (11)-Tìm phần tử nguyên tố đầu tiên có trong mảng
- (12)-Tìm phần tử chẵn cuối cùng có trong mảng
- (13)-Tìm phần tử chính phương cuối cùng có trong mảng
- (14)-Tìm phần tử lớn nhất có trong mảng
- (15)-Đếm số phần tử chẵn có trong mảng
- (16)-Đếm số phần tử lớn nhất có trong mảng
- (17)-In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (18)-Thêm một phần tử vào đầu mảng.
- (19)-Thêm một phần tử vào cuối mảng.
- (20)-Thêm một phần tử vào vị trí x trong mảng.
- (21)-Xóa phần tử chẵn đầu tiên.
- (22)-Xóa tất cả các phần tử lớn nhất trong mảng
- (23)-Sắp xếp mảng tăng dần
- (24)-Viết hàm nhập vào một mảng hai chiều các số nguyên gồm m dòng, n cột ( $0 < m, n < 100$ )
- (25)-Viết hàm nhập vào một mảng hai chiều các số thực gồm m dòng, n cột ( $0 < m, n < 100$ )
- (26)-Viết hàm xuất mảng hai chiều các số nguyên mxn phần tử vừa nhập ở trên
- (27)-Viết hàm xuất mảng hai chiều các số thực mxn phần tử vừa nhập ở trên
- (28)-Tính tổng các phần tử có trong mảng
- (29)-Tính tổng các phần tử chẵn có trong mảng
- (30)-Tính tổng các phần tử lẻ có trong mảng
- (31)-Tính tổng các phần tử nguyên tố có trong mảng
- (32)-Tính tổng các phần tử nằm trên đường chéo chính có trong mảng
- (33)-Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng
- (34)-Tính tổng các phần tử nằm trên đường biên có trong mảng
- (35)-Tìm phần tử chẵn đầu tiên có trong mảng
- (36)-Tìm phần tử lẻ đầu tiên có trong mảng
- (37)-Tìm phần tử nguyên tố đầu tiên có trong mảng
- (38)-Tìm phần tử chẵn cuối cùng có trong mảng
- (39)-Tìm phần tử chính phương cuối cùng có trong mảng
- (40)-Tìm phần tử lớn nhất có trong mảng
- (41)-Đếm số phần tử chẵn có trong mảng
- (42)-Đếm số phần tử lớn nhất có trong mảng
- (43)-In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng

- (44)- Tính tổng các phần tử nằm trên một dòng
  - (45)- Tìm dòng có tổng lớn nhất
  - (46)- Sắp xếp mảng tăng dần
  - (47)- Nhập chuỗi
  - (48)- Xuất chuỗi
  - (49)- Đếm số ký tự ‘ a ’ có trong chuỗi
  - (50)- Cắt khoảng trắng có trong chuỗi
  - (51)- Đếm khoảng trắng trong chuỗi
  - (52)- Đếm số từ có trong chuỗi
  - (53)- Sắp xếp chuỗi tăng dần
  - (54)- Nhập 3 chuỗi, xuất chuỗi theo thứ tự từ điển
  - (55)- Viết hàm Nhập vào một phân số
  - (56)- Viết hàm Nhập vào một dãy phân số
  - (57)- Viết hàm xuất một phân số
  - (58)- Viết hàm xuất một dãy phân số
  - (59)- Viết hàm tìm phân số lớn nhất trong dãy phân số
  - (60)- Viết hàm tính tổng các phân số có trong dãy
  - (61)- Viết hàm nhập dữ liệu cho một sinh viên. 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.
  - (62)- Viết hàm xuất dữ liệu một sinh viên với thông tin vừa nhập ở trên
  - (63)- Viết hàm nhập danh sách sinh viên, lưu trên mảng một chiều
  - (64)- Viết hàm xuất danh sách sinh viên
  - (65)- Xuất thông tin của sinh viên có mã sinh viên là “ X “
  - (66)- Xuất danh sách sinh viên thuộc ngành công nghệ thông tin
  - (67)- Xuất danh sách sinh viên Nữ thuộc ngành công nghệ thông tin
  - (68)- Sắp xếp danh sách sinh viên theo tên
  - (69)- Sắp xếp danh sách sinh viên theo MSSV
  - (70)- Sắp xếp danh sách sinh viên theo điểm toán
-

## 6 ĐỆ QUY (Recursion)

Sau khi học xong bài này, sinh 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 cách giải quyết một số bài toán kinh điển bằng phương pháp đệ quy; - Biết xử lý các giải thuật trên mảng 1 chiều bằng phương pháp đệ quy.

### 6.1. 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 1:** Người ta định nghĩa giai thừa của một số nguyên dương  $n$  như sau:

$$n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n \quad (\text{với } 0! = 1)$$

Như vậy, để tính  $n!$  ta thấy nếu  $n=0$  thì  $n!=1$  ngược lại thì  $n!=n * (n-1)!$

Với định nghĩa trên thì hàm đệ quy tính  $n!$  được viết:

```
#include <stdio.h>
#include <conio.h>
/*Hàm tính n! bằng đệ quy */
int giaithua_dequy(int n)
{
    if (n==0)
        return 1;
    else
        return n*giaithua_dequy(n-1);
}
/*Hàm tính n! không đệ quy*/
int giaithua_khongdequy(int n)
{
    int kq,i;
    kq=1;
    for (i=2;i<=n;i++)
        kq=kq*i;
    return kq;
}
int main()
{
    int n;
    printf("\n Nhap so n can tinh giai thua ");
    scanf("%d",&n);
    printf("\nGoi ham de quy: %d != %d ",n,giaithua_dequy(n));
```

```

        printf("\nGoi ham khong de quy:  %d  !=  %d  ",  n,
giaithua_khongdequy(n));
        getch();
        return 0;
    }

```

**Ví Dụ 2 :** 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: dùng vòng lặp (không đệ quy)

```

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

```

//Cách 2: dùng hàm đệ quy

```

long tinh tong (int n)
{
    if (n == 0)
        return 0;
    long s= tinh tong (n - 1);
    return s + n ;
}

```

## 6.2. Phân loại hàm đệ quy

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

- Đệ quy tuyến tính.
- Đệ quy nhị phân.
- Đệ quy phi tuyến .
- Đệ quy hỗ tương .

### 6.2.1. Đệ quy tuyến tính

- Một hàm được gọi là đệ quy tuyến tính khi bên trong thân hàm có duy nhất một lời gọi hàm lại chính nó
- Các hàm đệ quy tuyến tính có dạng

```

void < tên >
{
    if<điều kiện dừng>
    {
        /* trả về giá trị hay kết thúc công việc*/
    } else
    {
        /* làm một số công việc...*/
        /* gọi đệ quy đến hàm < tên> */
    }
}

```

- Ví dụ : Tính tổng  $S = 2 + 4 + 6 + \dots + 2n$ .

Ta có:  $S(n) = 2 + 4 + 6 + \dots + 2(n-1) + 2n = S(n-1) + 2n$  ;

```
// Hàm cài đặt
long tongchan(int n)
{
    if (n==1)
        return 2;
    return 2*n + tongchan(n-1);
}
```

### 6.2.2. Đệ quy nhị phân

- Một hàm được gọi là đệ quy nhị phân khi bên trong thân hàm có 2 lời gọi hàm gọi lại chính nó một cách tường minh.
- Chúng ta thường dùng để cài đặt thuật toán chia để trị hay duyệt cây nhị phân.
- Các hàm đệ quy nhị phân có dạng sau:

```
void <tên>
{
    if<điều kiện dừng>
    {
        /*trả về giá trị hay kết thúc công việc*/
    }
    else
    {
        /*làm một số công việc ... */
        /*gọi đệ quy đến hàm <tên> để giải quyết vấn đề nhỏ hơn*/
        /*gọi đệ quy đến hàm <tên> để giải quyết vấn đề còn lại*/
    }
}
```

- Ví dụ: Viết chương trình tính số hạng thứ n của chuỗi Fibonacci. Chuỗi số Fibonacci: 1 1 2 3 5 8 13 ...

Dãy Fibonacci được Định nghĩa truy hồi như sau:

- $F_0 = F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$  nếu  $n \geq 2$

```
// hàm cài đặt
long fibo (int n)
{
    if (n <= 1)
        return 1;
    else
        return fibo(n-1) + fibo(n-2);
}
```

### 6.2.3. Đệ quy phi tuyến

- Trong các chương trình đệ quy phi tuyến, việc gọi đệ quy sẽ được thực hiện bên trong vòng lặp.
- Các hàm đệ quy phi tuyến có dạng sau:

```
void < TÊN>
{
    for (int i =1 ; i < =n ; i++)
    {
        // làm một số công việc ...
        if<điều kiện dừng>
```



```

        {
            // làm một số công việc ...
        }
    else
    {
        // gọi đệ quy đến hàm <TÊN>
    }
}
}

```

- Ví Dụ: Cho dãy  $X_n$  được xác định theo công thức truy hồi như sau:

$$\begin{cases} X_0 = 1 \\ X_n = 1^2 X_0 + 2^2 X_1 + \dots + n^2 X_{n-1} \end{cases}$$

```

long Xn (int n)
{
    long tp= 0;
    if (n == 0)
        return 1;
    for (int i =0 ; i < n ; i++)
        tp = tp + (i+1) * (i+ 1) * Xn (i);
    return tp;
}

```

#### 6.2.4. Đệ quy hỗ tương

- Hai hàm được gọi là đệ quy hỗ tương khi bên trong thân hàm này có lời gọi hàm tới hàm kia, và bên trong thân hàm kia có lời gọi hàm tới hàm này.
- Ví Dụ: Viết hàm tính số hạng thứ n của 2 dãy số sau:

$$\begin{cases} x_0=1 \\ x_n=x_{n-1}+y_{n-1} \quad (n \geq 1) \end{cases}$$

$$\begin{cases} y_0=1 \\ y_n=2x_{n-1}+3y_{n-1} \quad (n \geq 1) \end{cases}$$

// hàm cài đặt

```

long tinhx (int n)
{
    if (n == 0)
        return 1;
    return tinhx (n-1) + tinhx (n-1) ;
}
long tinhx (int n)
{
    if (n == 0)
        return 1;
    return 2 * tinhx (n-1)+ 3 * tinhx (n-1);
}

```

#### 6.3. Kỹ thuật giải bài toán bằng đệ quy

Bước 1. Thông số hóa bài toán.

Bước 2. Tìm các điều kiện biên (chặn , dừng), tìm giải thuật cho các tình huống này.

Bước 3. Tìm giải thuật tổng quát theo hướng đệ quy lui dần về tình huống bị chặn.

Ví dụ Tính tổng 1 mảng a, n phần tử

- Thông số hóa: int a [ ] , int n

- Điều kiện biên: Mảng 0 phần tử thì tổng bằng 0.
- Giải thuật chung:

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

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

// hàm 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.

## 6.4. Một số bài toán kinh điển dùng phương pháp đệ quy

### 6.4.1. Bài toán Tháp Hà Nội

Truyền thuyết kể rằng: Một nhà toán học Pháp sang Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa quý gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo cách:

- Mỗi lần chỉ chuyển một đĩa
- Khi chuyển có thể dùng một cột trung gian B
- Trong suốt quá trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé được đặt trên đĩa có kích thước lớn).

Khi hỏi các vị sư cho biết khi nào chuyển xong chồng đĩa ?

Các vị trả lời : Đến ngày tận thế.

Vì với n đĩa cần  $2^n - 1$  lần chuyển đĩa

Với  $n=64$   $T=(2^{64} - 1) \cdot t$ .

giả sử  $t=1/100s$  thì  $T = 5.8$  tỷ năm.

```

BC.EXE
File Edit Search Run Compile Debug Project Options
// Hanoi.cpp - Bai toan Thap Ha Noi
#include <stdio.h>
#include <conio.h>
void ChuyenDia<int n, char X, char Z>
{ printf("Chuyen dia %d tu cot %c sang cot %c\n", n,X, Z);
}
void ThapHaNoi<int n, char X, char Y, char Z>
{ if <n>0>
{ ThapHaNoi<n-1,X,Z,Y>;
  ChuyenDia<n,X,Z>;
  ThapHaNoi<n-1,Y,X,Z>;
}
}
void main<>
{ clrscr<>;
  ThapHaNoi<3,'A','B','C'>;
  getch<>;
}

```

### 6.4.2. Phương pháp Chia để trị (*Divide and Conquer*)

- Giải thuật phân rã vấn đề thành những vấn đề con, giải những vấn đề con này và kết hợp những lời giải của những vấn đề con thành lời giải cho vấn đề nguyên thủy.
- Chiến lược này bao gồm 3 bước sau đây ở mỗi cấp đệ quy:
  - **Phân chia** (*divide*) đầu vào thành các bài toán con
  - **Đệ quy** (*recur*): giải quyết các bài toán con bằng gọi đệ quy.
  - **Trị** (*Conquer, combine*): Kết hợp các giải pháp tìm được để giải quyết bài toán.
- Chú ý: độ phức tạp giải thuật thường là:  $(\log_n \times (\text{divide}(n) + \text{combine}(n)))$ .
- Áp dụng giải bài toán sắp xếp trên mảng

// Hàm cài đặt

```
void MergeSort (int a[], int Left, int Right)
{
    if (Left < Right) //Mảng có nhiều hơn 1 phần tử
    {
        int Mid = (Left+Right)/2;
        // Sắp xếp mảng bên trái
        MergeSort (a, Left, Mid);
        // Sắp xếp mảng bên phải
        MergeSort (a, Mid+1, Right);
        // Trộn 2 mảng lại với nhau
        Merge (a, Left, Mid, Right);
    }
}
```

### 6.5. Nhận xét

- Hàm đệ quy là hàm mà trong thân hàm lại gọi chính nó.
- Giải thuật đệ quy đẹp (gọn gàng, dễ chuyển thành chương trình).
- Đặc điểm của *hàm đệ quy*:
  - Nhiều ngôn ngữ không hỗ trợ giải thuật đệ quy (Fortran).
  - Nhiều giải thuật rất dễ mô tả dạng đệ quy nhưng lại rất khó mô tả với giải thuật không đệ quy.
  - Vừa tốn bộ nhớ vừa chạy chậm (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).

### 6.6. Cấu trúc lặp và đệ quy

- Lập trình đệ quy sử dụng cấu trúc lựa chọn
- Phương pháp lặp sử dụng cấu trúc lặp.
- 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.
- Cả 2 phương pháp đều phải kiểm tra khi nào thì kết thúc.
  - Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai.

- Phương pháp đệ quy kết thúc khi đến trường cơ sở.
  - Phương pháp lặp thay đổi biến đếm trong vòng lặp cho đến khi nó làm cho điều kiện lặp sai.
  - Còn đệ quy làm cho các lời gọi hàm đơn giản dần cho đến khi đơn giản đến trường cơ sở.
- Cả 2 phương pháp đều có thể dẫn đến trường hợp chạy vô hạn mãi.
- Lặp sẽ không thoát ra được khi điều kiện lặp không bao giờ sai.
  - Còn đệ quy không thoát ra được khi các bước đệ quy không làm cho bài toán đơn giản hơn và cuối cùng hội tụ về trường cơ sở.
  - Tuy nhiên đệ quy tồi hơn vì nó liên tục đưa ra lời gọi hàm làm tốn thời gian của bộ vi xử lý và không gian nhớ.

## 6.7. Phần thực hành

### 6.7.1. Bài thực hành số 6.1

Làm lại các ví dụ trong phần lý thuyết

### 6.7.2. Bài thực hành số 6.2

Dùng phương pháp đệ quy làm các bài tập sau:

- (1)- Viết hàm nhập mảng một chiều các số nguyên gồm n phần tử ( $0 < n \leq 100$ )
- (2)- Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên
- (3)- Tính tổng các phần tử có trong mảng
- (4)- Tính tổng các phần tử chẵn có trong mảng
- (5)- Tính tổng các phần tử nguyên tố có trong mảng
- (6)- Tìm phần tử chẵn đầu tiên có trong mảng
- (7)- Tìm phần tử lẻ đầu tiên có trong mảng
- (8)- Tìm phần tử chẵn cuối cùng có trong mảng
- (9)- Tìm phần tử chính phương cuối cùng có trong mảng
- (10)- Tìm phần tử lớn nhất có trong mảng
- (11)- Đếm số phần tử chẵn có trong mảng
- (12)- Đếm số phần tử lớn nhất có trong mảng
- (13)- In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (14)- Sắp xếp mảng tăng dần

### Hướng dẫn

Viết hàm nhập số phần tử của mảng ( $0 < n \leq 100$ ).

```
void nhapsopt (int &n)
{
    do
    {
        printf ("nhap so phan tu cua mang : ");
        scanf ("%d", &n);
        if (n <= 0 || n > 100)
            printf ("nhap sai, nhap lai ");
    } while (n <= 0 || n > 100);
}
```

## 6.8. Bài tập (sinh viên tự thực hiện)

Dùng phương pháp đệ quy giải các bài toán sau

### 6.8.1. Tổng/tích một dãy số

(1)- Tính  $P(n) = 1.3.5 \dots (2n+1)$ , với  $n \geq 0$

(2)- Tính  $S(n) = 1 + 3 + 5 + \dots + (2 \times n + 1)$ , với  $n \geq 0$

(3)- Tính  $S(n) = 1 - 2 + 3 - 4 + \dots + (-1)^{n+1}n$ , với  $n > 0$

(4)- Tính  $S(n) = 1 + 1.2 + 1.2.3 + \dots + 1.2.3 \dots n$ , với  $n > 0$

(5)- Tính  $S(n) = 1^2 + 2^2 + 3^2 + \dots + n^2$ , với  $n > 0$

(6)- Tính  $S(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$ , với  $n > 0$

(7)- Tính  $S(n) = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \dots + \frac{1}{1+2+3+\dots+n}$ , với  $n > 0$

(8)- Tính  $P(x, y) = x^y$

(9)- Tính  $S(n) = 1 + (1+2) + (1+2+3) + \dots + (1+2+3+\dots+n)$ , với  $n > 0$

(10)- Tính  $S(n) = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{6} \dots + \frac{1}{2n}$

(11)- Tính  $S(n) = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \dots + \frac{1}{2n+1}$

(12)- Tính  $S(n) = \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \frac{1}{3 \times 4} + \dots + \frac{1}{n \times (n+1)}$

(13)- Tính  $S(n) = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} \dots + \frac{n}{n+1}$

(14)- Tính  $S(n) = \frac{1}{2} + \frac{3}{4} + \frac{5}{6} \dots + \frac{2n+1}{2n+2}$

(15)- Tính  $S(n) = x + x^2 + x^3 + \dots + x^n$

(16)- Tính  $S(n) = x^2 + x^4 + \dots + x^{2n}$ .

(17)- Tính  $S(n) = x + x^3 + x^5 + \dots + x^{2n+1}$ .

(18)- Tính  $S(n) = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \frac{1}{1+2+3+4} + \dots + \frac{1}{1+2+3+\dots+n}$

(19)- Tính  $S(n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$

(20)- Tính  $S(n)$ . Biết  $S(n)$  có  $n$  dấu phân số

$$S(n) = \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{\dots}{1 + \frac{\dots}{1 + \frac{1}{1 + \frac{1}{1}}}}}}}$$

(21)- Tính  $S(n) = \sqrt{2 + \sqrt{2 + \sqrt{2 + \dots \sqrt{2 + \sqrt{2}}}}}$  có n dấu căn.

(22)- Tính  $S(n) = \sqrt{n + \sqrt{n-1 + \sqrt{n-2 + \dots \sqrt{2 + \sqrt{1}}}}}$  có n dấu căn.

(23)- Tính  $S(n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + \dots \sqrt{n-1 + \sqrt{n}}}}}$  có n dấu căn.

### 6.8.2. Thao tác trên 1 số nguyên

(24)- Đếm số lượng chữ số của số nguyên dương n. VD:  $n=29730 \Rightarrow 29730$  gồm 5 chữ số

(25)- Tính tổng các chữ số của số nguyên dương n.

(26)- Tính tích các chữ số của số nguyên dương n.

(27)- Đếm số lượng chữ số lẻ của số nguyên dương n.

(28)- Tính tổng các chữ số chẵn của số nguyên dương n.

(29)- Tính tích các chữ số lẻ của số nguyên dương n.

(30)- Cho số nguyên dương n. Tìm chữ số đầu tiên của n.

(31)- Tìm chữ số đảo ngược của số nguyên dương n.

(32)- Tìm chữ số lớn nhất của số nguyên dương n.

(33)- Tìm chữ số nhỏ nhất của số nguyên dương n.

(34)- Kiểm tra số nguyên dương n có toàn chữ số lẻ hay không?

(35)- Tìm ước số lẻ lớn nhất của số nguyên dương n. Ví dụ  $n=100$  ước số lẻ lớn nhất của 100 là 25.

(36)- Kiểm tra số nguyên dương n có toàn chữ số chẵn hay không?

### 6.8.3. Ước số chung lớn nhất

(37)- Tìm ước số chung lớn nhất của hai số nguyên dương a và b.

### 6.8.4. Mảng một chiều

(38)- Tính tổng tất cả các phần tử trong mảng

(39)- Tính tổng các số lẻ có trong mảng.

✎ Mở rộng: cho trường hợp: số nguyên tố, số chính phương, ...

(40)- Tìm giá trị là số lẻ cuối cùng có trong mảng. Nếu không có số lẻ trả về -1.

✎ Mở rộng: cho trường hợp giá trị cuối cùng là số nguyên tố, giá trị cuối cùng là số chính phương, ...

(41)- Tìm vị trí chứa số lẻ cuối cùng có trong mảng. Nếu không có số lẻ trả về -1.

➤ *Mở rộng*: cho trường hợp: vị trí cuối cùng chứa số nguyên tố, vị trí cuối cùng số chính phương, ...

(42)- Kiểm tra xem mảng có được sắp xếp tăng dần hay không?

(43)- Kiểm tra xem mảng có chứa số nguyên tố hay không?

#### 6.8.5. Mảng hai chiều

(44)- Tính tổng tất cả các phần tử trong mảng 2 chiều.

(45)- Tính tổng các số lẻ có trong mảng 2 chiều.

➤ *Mở rộng*: cho trường hợp: số nguyên tố, số chính phương, ...

(46)- Tìm giá trị là số lẻ cuối cùng có trong mảng 2 chiều. Nếu không có số lẻ trả về -1.

➤ *Mở rộng*: cho trường hợp giá trị cuối cùng là số nguyên tố, giá trị cuối cùng là số chính phương, ...

(47)- Tìm vị trí chứa số lẻ cuối cùng có trong mảng 2 chiều. Nếu không có số lẻ trả về -1.

➤ *Mở rộng*: cho trường hợp: vị trí cuối cùng chứa số nguyên tố, vị trí cuối cùng số chính phương, ...

(48)- Cho ma trận A ( $n \times m$ ) chỉ chứa các giá trị 0 và -1. Giả sử có định nghĩa về “thành phần liên thông” như sau: một nhóm các ô liên tục (chỉ tính 4 phía trên, dưới, trái, phải) cùng chứa giá trị -1 được xem là 1 “thành phần liên thông”. Viết hàm đếm số “thành phần liên thông” có trong ma trận.

Ví dụ:

0	-1	-1	0	0	0	-1
0	-1	-1	0	0	0	0
0	-1	0	0	0	0	0
0	-1	-1	0	0	0	-1
0	0	0	0	0	0	-1
0	-1	0	-1	-1	-1	-1
0	-1	0	0	0	0	-1
<b>Có 4 thành phần liên thông</b>						

0	0	-1	0
0	-1	-1	-1
0	0	-1	0
0	0	0	0
-1	0	0	0
0	-1	0	0
0	0	0	0
<b>Có 3 TPLT</b>			

#### 6.8.6. Một số bài toán đề quy thông dụng

(49)- Bài toán tháp Hà Nội: Có 3 chồng đĩa đánh số 1, 2 và 3. đầu tiên chồng 1 có  $n$  đĩa được xếp sao cho đĩa lớn hơn nằm bên dưới và 2 chồng còn lại không có đĩa nào. Yêu cầu: chuyển tất cả các đĩa từ chồng 1 sang chồng 3, mỗi lần chuyển 1 đĩa và được phép sử dụng chồng 2 làm trung gian. Hơn nữa trong quá trình chuyển đĩa phải bảo đảm quy tắc đĩa lớn nằm bên dưới.

(50)- Bài toán phát sinh hoán vị: Cho tập hợp A có  $n$  phần tử được đánh số  $1, 2, \dots, n$ . Một hoán vị của A là một dãy  $a_1, a_2, \dots, a_n$ . Trong đó  $a_i \in A$  và chúng đôi một khác nhau. Hãy viết hàm phát sinh tất cả các hoán vị của tập hợp A.

(51)- Bài toán Tám Hậu: Cho bàn cờ vua kích thước  $(8 \times 8)$ . Hãy sắp 8 quân hậu vào bàn cờ sao cho không có bất kỳ 2 quân hậu nào có thể ăn nhau.

(52)- Bài toán Mã Đi Tuần: Cho bàn cờ vua kích thước  $(8 \times 8)$ . Hãy di chuyển quân mã trên khắp bàn cờ sao cho mỗi ô đi đúng 1 lần.



## 7 TẬP TIN (FILE)

Sau khi học xong bài này, sinh 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.

### 7.1. 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 nhập lại từ bàn phím. 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 (đĩa). 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).

#### 7.1.1. Phân loại kiểu tập tin

Có 3 loại dữ liệu kiểu tập tin:

##### 7.1.1.1. 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.

##### 7.1.1.2. 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.

##### 7.1.1.3. 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.

#### 7.1.2. 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.

#### 7.1.3. 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).

## 7.2. Các thao tác trên tập tin

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

- Khai báo biến tập tin.
- Mở tập tin bằng hàm `fopen()`.
- Thực hiện các thao tác xử lý dữ liệu của tập tin bằng các hàm đọc/ghi dữ liệu.
- 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`.

### 7.2.1. 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;`

### 7.2.2. Mở tập tin

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

Trong đó:

- Path: chuỗi chỉ đường dẫn đến tập tin trên đĩa.
- Mode: 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.

### 7.2.3. Đóng tập tin

#### 7.2.3.1. 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.

#### 7.2.3.2. 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.

### 7.2.4. 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.

### 7.2.5. Di chuyển con trỏ tập tin về đầu tập tin

- 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)**

### 7.3. Truy cập tập tin văn bản

#### 7.3.1. Ghi dữ liệu lên tập tin văn bản

##### 7.3.1.1. 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.

##### 7.3.1.2. 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.

##### 7.3.1.3. 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
%d	Ghi số nguyên
%[.số chữ số thập phân] f	Ghi số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
%o	Ghi số nguyên hệ bát phân
%x	Ghi số nguyên hệ thập lục phân
%c	Ghi một ký tự
%s	Ghi chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	Ghi số thực dạng khoa học (nhân 10 mũ x)

##### 7.3.1.4. Ví dụ

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

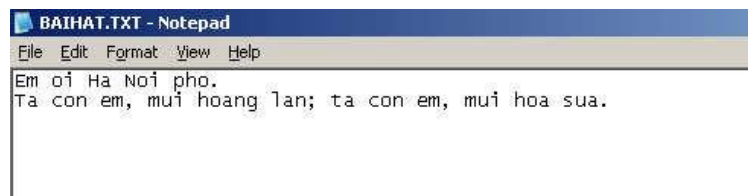
```
#include<stdio.h>
#include<conio.h>
int main ()
{
    FILE *f;
    clrscr ();
    f = fopen ("D:\\Baihat.txt", "r+");
```

```

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);
}
getch();
return 0;
}

```

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



### 7.3.2. Đọc dữ liệu từ tập tin văn bản

#### 7.3.2.1. 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**.

#### 7.3.2.2. 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**.

#### 7.3.2.3. 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 1:** 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;
    clrscr();
    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();
    }
    getch(); return 0;
}
```

## 7.4. Truy cập tập tin nhị phân

### 7.4.1. Ghi dữ liệu lên tập tin nhị phân

- 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.

### 7.4.2. Đọc dữ liệu từ tập tin nhị phân

- 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.

### 7.4.3. Di chuyển con trỏ tập tin

- Việc ghi hay đọc dữ liệu từ tập tin sẽ làm cho con trỏ tập tin dịch chuyển một số byte, đây chính là kích thước của kiểu dữ liệu của mỗi phần tử của tập tin.
- Khi đóng tập tin rồi mở lại, con trỏ luôn ở vị trí ngay đầu tập tin.
- Nhưng nếu ta sử dụng kiểu mở tập tin là “a” để ghi nối dữ liệu, con trỏ tập tin sẽ di chuyển đến vị trí cuối cùng của tập tin này.
- Ta cũng có thể điều khiển việc di chuyển con trỏ tập tin đến vị trí chỉ định bằng hàm `fseek()`.
- Cú pháp: **`int fseek (FILE *f, long offset, int whence)`**

Trong đó:

- `fseek` di chuyển con trỏ `f` đến vị trí `offset` theo mốc `whence`
- `f` : con trỏ tập tin đang thao tác.
- `offset`: số byte cần dịch chuyển con trỏ tập tin kể từ vị trí trước đó. Phần tử đầu tiên là vị trí 0.
- `whence`: vị trí bắt đầu để tính `offset`, ta có thể chọn điểm xuất phát là:
  - `#define SEEK_SET 0` //tính từ đầu tập tin
  - `#define SEEK_CUR 1` //tính từ vị trí hiện hành của con trỏ
  - `#define SEEK_END 2` // tính từ cuối tập tin
- `fseek` trả về:
  - = 0 nếu thành công.
  - <>0 nếu di chuyển có lỗi.

- **Ví dụ 2:** 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);
    }
}
```

```

    }
    getch();
    return 0;
}

```

- **Ví dụ 3:** 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 tục"); 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 tục!!!");
        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 tục!!!");
        getch();
    }
    int main()
    {
        int c;
        for (;;)
        {
            clrscr();
            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 ;
    }
}

```



### 7.5. Bài tập (sinh viên tự thực hiện)

- (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.
- (2)- Viết chương trình cho phép thống kê số lần xuất hiện của các ký tự là chữ ('A'..'Z','a'..'z') trong một tập tin văn bản.
- (3)- Viết chương trình đếm số từ và số dòng trong một tập tin văn bản.
- (4)- Viết chương trình nhập từ bàn phím và ghi vào 1 tập tin tên là DMHH.TXT với mỗi phần tử của tập tin là 1 cấu trúc bao gồm các trường:
- Ma (mã hàng: char[5]).
  - Ten (Tên hàng: char[20]).

Kết thúc việc nhập bằng cách gõ ENTER vào Ma.

- (5)- Viết chương trình cho phép nhập từ bàn phím và ghi vào 1 tập tin tên DSHH.TXT với mỗi phần tử của tập tin là một cấu trúc bao gồm các trường:
- mh (mã hàng: char[5]).
  - sl (số lượng: int).
  - dg (đơn giá: float).
  - st (Số tiền: float).

#### Yêu cầu cài đặt:

- Mỗi lần nhập một cấu trúc
- Trước tiên nhập mã hàng (mh), đưa mh so sánh với Ma trong tập tin DMHH.TXT đã được tạo ra bởi bài tập 146, nếu mh=ma thì in tên hàng ngay bên cạnh mã hàng.
- Nhập số lượng (sl).
- Nhập đơn giá (dg).
- Tính số tiền = số lượng \* đơn giá.

Kết thúc việc nhập bằng cách đánh ENTER vào mã hàng. Sau khi nhập xong yêu cầu in toàn bộ danh sách hàng hóa có sự giải mã về tên hàng.

- (6)- Viết chương trình đọc một chuỗi tối đa 100 ký tự từ bàn phím. Lưu các ký tự là nguyên âm vào tập tin "NguyenAm.txt". Đọc các ký tự từ tập tin này và hiển thị lên màn hình console.
- (7)- 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 n các số nguyên

- Viết chương trình đọc file trên, lưu vào mảng 1 chiều
- Xuất ra màn hình
- Tính tổng các số chẵn có trong mảng
- Tìm phần tử lớn nhất

- (v). Đếm số phần tử lớn nhất
  - (vi). Sắp xếp mảng tăng dần
- (8)- Cho file TXT có cấu trúc như sau:
- Dòng đầu lưu giá trị của 2 số nguyên dương m,n. Các dòng còn lại lưu giá trị của 1 ma trận m dòng và n cột là các số nguyên.
- (i). Viết chương trình đọc file ma trận trên
  - (ii). Xuất ma trận đó ra màn hình
  - (iii). Tính tổng các số chẵn có trong ma trận
  - (iv). Tìm phần tử lớn nhất có trong ma trận
  - (v). Đếm số phần tử lớn nhất có trong ma trận
  - (vi). Sắp xếp mảng tăng dần
- (9)- Viết chương trình tạo mới 1 file chứa dữ liệu của một ma trận 10 x10 với các phần tử của ma trận là số nguyên (0-100) được tạo ngẫu nhiên. Sau khi đóng file vừa tạo, mở lại file, đọc và in ma trận ra màn hình.
- (10)- Lập chương trình tạo một tập tin chứa các số nguyên có giá trị ngẫu nhiên. Sắp xếp chúng theo thứ tự tăng dần và lưu trữ sang tập tin khác.
- (11)- Viết chương trình gồm 2 hàm sau:
- Hàm TaoFile:
    - Nhận tham số là số dòng, số cột cần có của 1 ma trận.
    - File cần tạo có tên MaTran.txt. Cấu trúc của file có dạng như sau:
      - Dòng 1: chứa số dòng của ma trận
      - Dòng 2: chứa số cột của ma trận
      - Dòng 3 đến dòng cuối: mỗi dòng chứa giá trị các phần tử có trên từng dòng của ma trận. Biết rằng giá trị các phần tử này được phát sinh ngẫu nhiên và có giá trị trong khoảng từ 1-100.
  - Hàm XoaDong:
    - Nhận tham số là số dòng (row) cần xóa.
    - Thực hiện đọc file MaTran.txt. Tiến hành xóa dòng row. Lưu kết quả lại vào file MaTran.txt.
- (12)- Viết chương trình đọc và hiện một tập tin văn bản. Sau đó trình bày những thống kê sau: số ký tự, số từ, số dòng của nó.
- (13)- Viết chương trình tạo ra một tập văn bản chứa tên, tuổi, địa chỉ (mỗi thông tin chiếm một dòng). Sau đó chương trình sẽ đọc lại tập tin này và chép sang một tập tin khác nhưng trên một dòng cho mỗi người.
- (14)- Viết chương trình mã hóa một file bằng phép XOR.
- (15)- Viết chương trình tính số lần xuất hiện ký tự chữ cái trong một tập tin văn bản DOS.
- (16)- Viết chương trình tính số từ có trong một tập tin văn bản DOS.

- (17)- Viết chương trình nối hai tập tin văn bản lại với nhau thành một tập tin mới.
- (18)- Viết chương trình lưu lại nội dung của màn hình vào tập tin SCREEN.DAT (tương tự lệnh COPY CON của DOS).
- (19)- Viết chương trình hiện chương trình được lưu trong SCREEN.DAT (tương tự lệnh TYPE của DOS).
- (20)- Tương tự bài trên nhưng những dòng trống sẽ được bỏ qua.
- (21)- \*Tương tự như bài trên, nhưng cho phép người xem cuộn lên, cuộn xuống màn hình.
- (22)- Viết chương trình in ra các giá trị bit của một mảng các giá trị nguyên dương gồm 23 phần tử. Những vị trí tương ứng với giá trị bit bằng 1 sẽ in dấu\*, bit bằng 0 sẽ in dấu -.
- (23)- Viết thủ tục tìm và thay thế một chuỗi con bằng một chuỗi khác.
- (24)- Bài toán ANCA. Viết chương trình tìm những hình chữ nhật có chiều dài gấp đôi chiều rộng và diện tích=chu vi.
- (25)- Viết chương trình tạo tập tin văn bản chứa 1 dãy số nguyên bất kỳ.
- (26)- Viết chương trình tạo tập tin nhị phân chứa 10000 số nguyên bất kỳ ghi vào file SONGUYEN.INP. Mỗi dòng 10 số, sau đó viết chương trình đọc file SONGUYEN.INP, sắp xếp theo thứ tự tăng dần và lưu kết quả vào file SONGUYEN.OUT.
- (27)- Viết chương trình tạo một file chứa 10000 số nguyên ngẫu nhiên đôi một khác nhau trong phạm vi từ 1 đến 32767 và đặt tên là "SONGUYEN.INP".
- (28)- Viết chương trình tạo một file chứa các số nguyên có tên SONGUYEN.INP. Sau đó đọc file SONGUYEN.INP và ghi các số chẵn vào file SOCHAN.OUT và những số lẻ vào file SOLE.OUT.
- (29)- Viết chương trình ghi vào tập tin SOCHAN.DAT các số nguyên chẵn từ 0 đến 100.
- (30)- Viết chương trình đọc tập tin SOCHAN.DAT và xuất ra màn hình.
- (31)- Viết chương trình tạo file văn bản có tên là "MATRIX.INP" có cấu trúc như sau:
- Dòng đầu ghi hai số m, n.
  - Trong m dòng tiếp theo mỗi dòng ghi n số và các số cách nhau một khoảng cách.
- Hãy kiểm tra xem trong file đó có bao nhiêu số nguyên tố.
- Kết quả cần ghi vào file "MATRIX.OUT" có nội dung là một số nguyên đó là số lượng các số nguyên tố trong file "MATRIX.INP".
- (32)- Cho mảng các số nguyên, hãy sắp xếp mảng theo thứ tự tăng dần.
- Dữ liệu vào : tập tin văn bản ARRAY.INP gồm 2 dòng
    - Dòng 1 chứa số nguyên n (  $n \leq 100$  ).
    - Dòng 2 chứa n số nguyên.
  - Kết quả : Đưa ra tập tin văn bản ARRAY.OUT gồm hai dòng
    - Dòng 1 chứa n phần tử của mảng các số nguyên.
    - Dòng 2 chứa n số nguyên được xếp tăng dần.
- (33)- Cho mảng các số nguyên, tìm phần tử lớn nhất của mảng.
- Dữ liệu vào: tập tin văn bản ARRAY.INP gồm hai dòng:
    - Dòng 1 chứa số nguyên n (  $n \leq 100$  ).

- Dòng 2 chứa n số nguyên.
  - Kết quả: Đưa ra tập tin văn bản ARRAY.OUT gồm 1 dòng ghi 2 giá trị x, y trong đó x là giá trị lớn nhất, y là vị trí của x trong mảng.
-

## **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 2003
- [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.
- [9] Hoàng Kiếm – Giai một bài toán trên máy tính như thế nào – Tập 1 & Tập 2- Nhà Xuất Bản Giáo Dục -2001
- [10] Dương Anh Đức – Trần Hạnh Nhi – Giáo trình cấu trúc dữ liệu - Trường Đại Học Khoa Học Tự Nhiên Tp.Hồ Chí Minh – 2003
- [11] Trần Đan Thư – Giáo trình lập trình C – Tập 1& Tập 2 – Nhà Xuất Bản Đại Học Quốc Gia – 2001
- [12] Lê Hoài Bắc – Lê Hoàng Thái – Nguyễn Tấn Trần Minh Khang – Nguyễn Phương Thảo – Giáo trình lập trình C – Nhà Xuất Bản Đại Học Quốc Gia Tp Hồ Chí Minh – 2003
- [13] Nguyễn Tiến Huy – Bài giảng Kỹ thuật lập trình – 2003.