

## 6 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.

### 6.1. Ký tự (character)

#### 6.1.1. 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

#### 6.1.2. 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 6.1
 

```
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)`
- Ví dụ 6.2
 

```
char ch;
printf("Nhap 1 ky tu: ");
ch = getch();
```

```
printf("Ma ASCII cua ky tu %c la %d", ch,ch);
```

- Hàm trả về ký tự nhận được.
  - Nếu ký tự có sẵn trong bộ đệm bàn phím thì hàm nhận một ký tự trong đó.
  - Nếu bộ đệm rỗng thì chương trình 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 vừa nhập không được hiển thị lên màn hình.

(iii). **Hàm getche**

- Nhận một ký tự từ bộ đệm bàn phím (tương tự như hàm getch) và cho hiển thị ký tự này lên màn hình.
- Cú pháp: `int getche(void)`
- Ví dụ 6.3

```
printf("Nhap 1 ky tu: ");  
int ch = getche();  
printf("Ma ASCII cua ky tu %c la %d", ch,ch);
```

(iv). **Hàm getchar**

- Hàm cho chương trình dừng lại và chờ người dùng nhập 1 ký tự và nhấn enter.
- Cú pháp: `int getchar(void)`
- Ví dụ 6.4

```
printf("Nhap 1 ky tu: ");  
unsigned char ch = getchar();  
printf("Ma ASCII cua ky tu %c la %d", ch,ch);
```

### 6.1.3. Xuất ký tự

(i). **Hàm putchar**

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

## 6.2. Chuỗi

### 6.2.1. 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.

### 6.2.2. Cách khai báo chuỗi

#### 6.2.2.1. Khai báo chuỗi

- Cú pháp: `char < tên biến > [ chiều dài tối đa chuỗi ]`
- Ví dụ 6.5 `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)

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

- Cú pháp: `char <Biến> []=<“Hằng chuỗi”>`
- Ví dụ 6.6: các cách khai báo và gán giá trị sau đây là tương đương nhau, chỉ khác nhau ở kích thước mảng được cấp:

```
char s1 [] = "Sai Gon";
char s2 [12] = "Sai Gon";
char s3 [] = {'S','a','i',' ','G','o','n','\0'};
char s4 [12]={'S','a','i',' ','G','o','n','\0'};
```

khi đó:

	0	1	2	3	4	5	6	7				
<b>s1</b>	'S'	'a'	'i'	' '	'G'	'o'	'n'	'\0'				

	0	1	2	3	4	5	6	7	8	9	10	11
<b>s2</b>	'S'	'a'	'i'	' '	'G'	'o'	'n'	'\0'				

	0	1	2	3	4	5	6	7				
<b>s3</b>	'S'	'a'	'i'	' '	'G'	'o'	'n'	'\0'				

	0	1	2	3	4	5	6	7	8	9	10	11
<b>s4</b>	'S'	'a'	'i'	' '	'G'	'o'	'n'	'\0'				

### 6.2.3. 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
```

#### 6.2.4. 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 6.7: 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 6.8: 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 6.9: 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;
}
```

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

#### 6.3.1. Thư viện <stdio.h>

##### 6.3.1.1. 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 6.10     scanf ("%s" , & Hoten);

##### 6.3.1.2. Hàm printf (xuất chuỗi)

- Hàm thực hiện xuất chuỗi ra màn hình.
- Ví dụ 6.11

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

#### 6.3.2. Thư viện <string.h>

##### 6.3.2.1. Hàm gets

- Hàm thực hiện nhận chuỗi ký tự được nhập từ bàn phím.
- Ví dụ 6.12     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()`.

#### 6.3.2.2. Hàm *puts*

- Hàm thực hiện xuất chuỗi xong tự động xuống dòng.
- Ví dụ 6.13 `puts (Hoten);`

#### 6.3.2.3. 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.

#### 6.3.2.4. Hàm *gotoxy*

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

#### 6.3.2.5. 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 6.14

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

#### 6.3.2.6. 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 6.15

```
#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
}
```

### 6.3.2.7. 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 6.16

```
#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
}
```

### 6.3.2.8. 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 6.17

```
#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
}
```

### 6.3.2.9. Hàm stricmp

- 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 stricmp (const char *s1, const char *s2)`

- Ví dụ 6.18

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

### 6.3.2.10. 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 stricmp(const char \*s1, const char \*s2)
- Ví dụ 6.19

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

### 6.3.2.11. 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 6.20

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

### 6.3.2.12. 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 6.21

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

### 6.3.2.13. Hàm strlen

- Lấy chiều dài chuỗi với hàm.
- *Cú pháp:*     **int strlen (const char \*s);**
- Ví dụ 6.22

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

- Ví dụ 6.23: 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);
    return 0;
}
```

- Ví dụ 6.24: 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()
{
    char newstr [35];
    char str[] = "Viet Nam";
    for(int i = 0; i<strlen(str); i++)
        newstr[i] = str[i];
    newstr[i] = '\0';
}
```

#### 6.3.2.14. 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 6.25 Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi "nation".

```
#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
}
```

#### 6.3.2.15. Hàm strupr

- Hàm strupr() được dùng để chuyển đổi chuỗi chữ thường thành chuỗi chữ hoa, kết quả trả về của hàm là một con trỏ chỉ đến địa chỉ chuỗi được chuyển đổi.
- *Cú pháp:* char \*strupr(char \*s)
- Ví dụ 6.26: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm strupr() để chuyển đổi chúng thành chuỗi chữ hoa.

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

#### 6.3.2.16. 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)

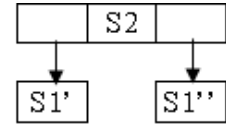
#### 6.3.2.17. Hàm atoi, atof, atol

- 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.

#### 6.3.2.18. Hàm `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 6.27



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

#### 6.3.2.19. Hàm `char* strrev(char *s)`

- Đảo ngược chuỗi.

### 6.3.3. Thư viện `<ctype.h>`

#### 6.3.3.1. Hàm `toupper`

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

#### 6.3.3.2. Hàm `tolower`

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

#### 6.3.3.3. 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: `<>0` khi `char_exp` là một chữ cái
- Ví dụ 6.28 `int kq= isalpha('a');`

#### 6.3.3.4. 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: <>0 khi *char\_exp* là một chữ cái
- Ví dụ 6.29 

```
int kq= isupper('a');
```

#### 6.3.3.5. 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: <>0 khi *char\_exp* là một chữ cái thường
- Ví dụ 6.30 

```
int kq= islower('a');
```

#### 6.3.3.6. 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: <>0 khi *char\_exp* là một ký số
- Ví dụ 6.31 

```
int kq= isdigit ('a');
```

#### 6.3.3.7. 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: <>0 khi *char\_exp* là một ký tự có mã ASCII<128.
- Ví dụ 6.32 

```
int kq= isascii('a');
```

#### 6.3.3.8. 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: <>0 khi *char\_exp* là ký tự khoảng trắng
- Ví dụ 6.33 

```
int kq= isspace ('a');
```

#### 6.3.3.9. 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: <>0 khi *char\_exp* là một ký tự có thể in được.
- Ví dụ 6.34 

```
int kq= isprint ('a');
```

#### 6.3.3.10. 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 6.35 

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

### 6.3.4. Thư viện <stdlib.h>

#### 6.3.4.1. 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 6.36 `int kq= atoi('123a45');`

#### 6.3.4.2. 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 6.37 `double kq= atof('123.45');`

#### 6.3.4.3. 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 6.38

```
#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
}
```

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

### 6.4.1. Mục tiêu

Sinh viên biết tổ chức các hàm chức năng để thao tác trên chuỗi như: 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, ...

### 6.4.2. Các yêu cầu (mỗi yêu cầu được viết riêng thành 1 hàm chức năng)

- (1)- Nhập chuỗi
- (2)- Xuất chuỗi
- (3)- Đếm số ký tự 'a' có trong chuỗi

- (4)- Đếm khoảng trắng trong chuỗi.  
 (5)- Đổi tất cả các ký tự có trong chuỗi thành chữ thường (không dùng hàm `strlwr`)

➤ Mở rộng: đổi tất cả thành chữ hoa/chữ in).

- (6)- Viết hàm đảo ngược các ký tự trong chuỗi.

Ví dụ: nhập ABCDE, chuỗi sau khi đảo ngược là:EDCBA

- (7)- Một từ được gọi là *palindrome* khi các ký tự trong từ đối xứng nhau. Một số từ *palindrome* là: rotor, radar, madam, abba, php, www. Viết hàm kiểm tra xem từ có phải là từ đối xứng hay không? Kết quả trả về *true* hoặc *false*.

➤ Mở rộng: Một chuỗi được gọi là *palindrome* khi các ký tự trong chuỗi đối xứng nhau.

Ví dụ: chuỗi sau là chuỗi *palindrome*: ABLE WAS I ERE I SAW ELBA

- (8)- So sánh 2 chuỗi S1 và S2. Kết quả trả về 0 nếu S1 và S2 giống nhau; trả về 1 nếu S1 lớn hơn S2 và trả về -1 nếu S2>S1.

Ví dụ:

S1	S2	Kết quả trả về	Giải thích
Sai Gon	Sai Gon	0	
Sai Gon	SAi Gon	1	Vì S1 có 'a' = 97 > A= '65' của S2
SAi Gon	Sai Gon	-1	Vì S1 có A= '65' < 'a' = 97 của S2

- (9)- Sắp xếp các ký tự trong chuỗi tăng dần.  
 (10)- Đếm số từ có trong chuỗi (các từ cách nhau bởi khoảng trắng)  
 (11)- Đổi tất cả ký tự đầu của từ thành chữ hoa (chữ in), các ký tự còn lại là chữ thường.  
 (12)- Cho nhập 1 số nguyên dương n (n>0). Chương trình thực hiện loại bỏ 1 số trong n sao cho giá trị các số còn lại trong n là nhỏ nhất. <sup>1</sup>

Ví dụ: n = 21      ⇒ bỏ số 2 ⇒ n = 1  
 n = 132      ⇒ bỏ số 3 ⇒ n = 12  
 n = 104      ⇒ bỏ số 1 ⇒ n = 4  
 n = 23198      ⇒ bỏ số 3 ⇒ n = 2198

- (13)- Giả sử chuỗi s được xem là hợp lệ khi giữa các từ chỉ có đúng 1 khoảng trắng và chuỗi không chứa các ký tự khoảng trắng ở đầu và cuối chuỗi. Viết hàm cắt khoảng trắng thừa đang có trong chuỗi.  
 (14)- Nhập 3 chuỗi, xuất 3 chuỗi theo thứ tự từ điển.

Ví dụ: nhập s1= "ABE", s2= "BAC", s3= "ABC". Sẽ in ra theo thứ tự "ABC", "ABE", "BAC".

- (15)- Cho nhập chuỗi S chỉ chứa các ký số 0 hoặc 1. Cho biết chuỗi có chứa chuỗi con có từ 5 ký tự trở lên giống nhau (cùng bằng 0 hoặc 1).

Ví dụ: S = "010100000010111" ⇒ có  
 S = "0101000111101" ⇒ không

<sup>1</sup> Trích trong đề thi ICPC Asia Regionals 2019 Online Preliminary Round (<https://www.codechef.com/ICPCIN19>)

(16)- Viết chương trình cho người dùng nhập 1 số nguyên ( $n$ ). In ra các số từ 1 đến  $n$  dưới dạng số nhị phân.

Ví dụ: tùy thuộc giá trị của  $n$ , kết quả in ra màn hình sẽ có dạng như sau:

	n=4	n=7	n=8
Kết quả xuất ra màn hình	1	1	1
	10	10	10
	11	11	11
	100	100	100
		101	101
		110	110
		111	111
			1000

#### **Lucky Number**

- (Theo Wikipedia) *Lucky Number* (số may mắn) là số được định nghĩa theo quá trình sau: bắt đầu với số nguyên dương  $x$  và tính tổng bình phương  $y$  các chữ số của  $x$ , sau đó tiếp tục tính tổng bình phương các chữ số của  $y$ . Quá trình này lặp đi lặp lại cho đến khi thu được kết quả là 1 thì dừng (tổng bình phương các chữ số của số 1 chính là 1) hoặc quá trình sẽ kéo dài vô tận. Số mà quá trình tính này kết thúc bằng 1 gọi là số may mắn. Số có quá trình tính kéo dài vô tận là *số không may mắn* hay còn gọi là *sad number* (số đen đui)

Ví dụ: 7 là số may mắn vì

$$7^2 = 49$$

$$4^2 + 9^2 = 97$$

$$9^2 + 7^2 = 130$$

$$1^2 + 3^2 + 0^2 = 10$$

$$1^2 + 0^2 = 1$$

- Những số may mắn dưới 500 là: 1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, 103, 109, 129, 130, 133, 139, 167, 176, 188, 190, 192, 193, 203, 208, 219, 226, 230, 236, 239, 262, 263, 280, 291, 293, 301, 302, 310, 313, 319, 320, 326, 329, 331, 338, 356, 362, 365, 367, 368, 376, 379, 383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496.

#### **Số Palindrome (hoặc Palindromic)**

- (Theo Wikipedia) Là một số vẫn giữ nguyên giá trị khi các chữ số của nó được đảo ngược. Hay nói cách khác là số đối xứng.
- 30 số thập phân *palindrome* đầu tiên là: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 202, ...

#### **Số nguyên tố Palindrome**

- (Theo Wikipedia) Là số nguyên tố vẫn giữ nguyên giá trị khi các chữ số của nó được đảo ngược.
- Các số nguyên tố *Palindrome* dưới 20000 gồm: 2, 3, 5, 7, 11, 101, 131, 151, 181, 191, 313, 353, 373, 383, 727, 757, 787, 797, 919, 929, 10301, 10501, 10601, 11311, 11411, 12421, 12721, 12821, 13331, 13831, 13931, 14341, 14741, 15451, 15551, 16061, 16361, 16561, 16661, 17471, 17971, 18181, 18481, 19391, 19891, 19991.

### Số strobogrammatic

- Là một số có giá trị không đổi khi xoay số đó 180 độ (180°). Nhận xét: các số *strobogrammatic* chỉ chứa các số sau đây: 0, 1, 6, 8, 9. Trong đó các số 0, 1, 8 không bị thay đổi giá trị sau khi xoay, còn số 6 và 9 bị thay đổi (6 chuyển thành 9 và 9 chuyển thành 6).
- Các số *strobogrammatic* đầu tiên là 0, 1, 8, 11, 69, 88, 96, 101, 111, 181, 609, 619, 689, 808, 818, 888, 906, 916, 986, 1001, 1111, 1691, 1881, 1961, 6009, 6119, 6699, 6889, 6969, 8008, 8118, 8698, 8888, 8968, 9006, 9116, 9696, 9886, 9966, ...

### Số nguyên tố strobogrammatic

- Là số vừa là số nguyên tố, vừa là số *strobogrammatic*.
- Các số nguyên tố *strobogrammatic* đầu tiên là: 11, 101, 181, 619, 16091, 18181, 19861, 61819, 116911, 119611, 160091, 169691, 191161, 196961, 686989, 688889, ...

### Số strobogrammatic mở rộng

Khi xoay các số dạng digital, ta thấy các số sau đây vẫn có ý nghĩa là 0, 1, 2, 5, 6, 8, 9. Trong đó các số 0, 1, 2, 5, 8 không bị thay đổi giá trị sau khi xoay, còn số 6 và 9 bị thay đổi (6 chuyển thành 9 và 9 chuyển thành 6)

### Thực hiện các bài tập về số sau với kiểu dữ liệu cần xử lý là kiểu chuỗi

#### (17)- *Lucky number*

Giả sử để xác định quá trình tìm *Lucky Number* có kéo dài vô tận hay không, người ta cho lặp việc tìm này tối đa 100 lần. Do đó nếu quá trình lặp để tìm vượt trên 100 lần thì chương trình có thể kết luận số đó không là *Lucky number*.

- Viết chương trình cho nhập số nguyên dương  $n$ . In ra số *Lucky Number* **nhỏ** hơn hay bằng và gần với  $n$  nhất.
- Viết chương trình cho nhập số nguyên dương  $n$ . In ra số *Lucky Number* **lớn** hơn hay bằng và gần với  $n$  nhất.
- Viết chương trình in ra các số *Lucky Number* từ 1 đến 1000000.

#### (18)- *Palindrome number*

- Lần lượt viết các hàm tự phát sinh số Palindrome gồm 2, 3, 4, 5 ký số.
- Viết chương trình in ra các số Palindrome từ 0 đến 100000.
- Viết chương trình in ra các số nguyên tố Palindrome từ 0 đến 100000.

#### (19)- *Strobogrammatic number*

- Lần lượt viết các hàm tự phát sinh số strobogrammatic gồm 2, 3, 4, 5 ký số.
- In ra các số *strobogrammatic* nhỏ hơn 1 triệu (1000000).
- In ra các số nguyên tố *strobogrammatic* nhỏ hơn 1 triệu (1000000).
- In ra các số *strobogrammatic* **mở rộng** nhỏ hơn 1 triệu (1000000).
- In ra các số nguyên tố *strobogrammatic* **mở rộng** nhỏ hơn 1 triệu (1000000).
- In ra các số nhỏ hơn 1 triệu (1000000) không phải là số strobogrammatic và không phải là số nguyên tố nhưng những số này sau khi xoay 180 độ lại là số nguyên tố.



(20)- Trong mật mã học, mật mã Caesar (*Caesar Cipher*, còn được gọi là mật mã dịch chuyển - *Shift Cipher*, mã của Caesar hoặc dịch chuyển Caesar), là một trong những kỹ thuật mã hóa đơn giản và được biết đến rộng rãi nhất. Nó là một loại mật mã thay thế, trong đó mỗi chữ cái trong bản rõ (plain text) được thay thế bằng một chữ cái một số vị trí cố định trong bảng chữ cái dựa trên khóa  $k$  (key). Ví dụ với dịch chuyển trái 3, A sẽ được thay thế bằng D, B sẽ trở thành E, v.v. Phương pháp này được đặt theo tên của Julius Caesar, người đã sử dụng nó trong thư tín riêng của mình.

Thông điệp được mã hóa bằng cách dịch chuyển xoay vòng từng ký tự đi  $k$  vị trí trong bảng chữ cái. Trường hợp với  $k=3$  gọi là phương pháp mã hóa Caesar.

☞ Minh họa: Cho *plain text* =  $x_1, x_2, x_3, x_4, x_5, x_6$  = 'ATTACK',  $k=17$ , chiều dịch chuyển của ký tự là từ trái sang phải và bảng chữ cái được quy ước đánh số như sau:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

– Thực hiện mã hóa:

$$y_1 = x_1 + k \bmod 26 = 0 + 17 \bmod 26 = 17 \Rightarrow R$$

$$y_2 = 19 + 17 \bmod 26 = 10 \Rightarrow K$$

$$y_3 = 10 \Rightarrow K$$

$$y_4 = 17 \Rightarrow R$$

$$y_5 = 2 + 17 \bmod 26 = 19 \Rightarrow T$$

$$y_6 = 10 + 17 \bmod 26 = 1 \Rightarrow B$$

Bản mã hoàn chỉnh (Cipher text)  $Y = y_1; y_2; y_3; y_4; y_5; y_6 = RKKRTB$

– Thực hiện giải mã: với  $k=17$  và cipher text ( $Y$ ) =  $y_1; y_2; : : : y_6 = RKKRTB$

$$x_1 = y_1 - k \bmod 26 = 17 - 17 \bmod 26 = 0 \Rightarrow A$$

$$x_2 = 10 - 17 \bmod 26 = -7 \bmod 26 = 19 \Rightarrow T$$

$$x_3 = 19 \Rightarrow T$$

$$x_4 = 0 \Rightarrow A$$

$$x_5 = 19 - 17 \bmod 26 = 2 \Rightarrow C$$

$$x_6 = 1 - 17 \bmod 26 = -16 \bmod 26 = 10 \Rightarrow K$$

Bản giải mã hoàn chỉnh (bản gốc - plan text)  $X = x_1; x_2; \dots; x_6 = ATTACK$

☞ Yêu cầu: viết chương trình tạo menu với các chức năng sau, trong đó khi thực hiện mã hóa hoặc giải mã, chương trình sẽ yêu cầu nhập khóa  $k$

- Nhập chuỗi
- Xuất chuỗi
- Mã hóa
- Giải mã

☞ Mở rộng:

- Văn bản bao gồm cả ký tự hoa và thường.
- Văn bản bao gồm tất cả các ký tự trong bộ mã ASCII nhưng bỏ qua các mã điều khiển (từ 0-30).

- Cho người dùng chọn chiều dịch chuyển (trái hoặc phải).

(21)- Viết chương trình cho phép chuyển đổi qua lại giữa số nguyên dương n (hệ thập phân) sang chữ số La Mã (Roman numeral). Chương trình gồm 2 hàm:

a. `char* Int_To_Roman(int n)`: để chuyển đổi số nguyên (hệ thập phân) có giá trị từ 1 đến dưới 4000 sang chữ số La Mã (Roman numeral).

Ví dụ    127    = CXXVII  
           300    = CCC  
           3321 = MMMCCCXXI

b. `int Roman_To_Int (char *s)`: để chuyển đổi số La Mã có giá trị từ 1 đến dưới 4000 sang số nguyên hệ thập phân.

Ví dụ    MMMCMLXXXVI = 3986  
           CMLIXIV       = 963  
           CDXCXL        = 530

⇒ Nhắc lại về chữ số La mã (theo wikipedia)

- Cách viết:

- Có bảy chữ số La Mã cơ bản

Ký tự	M	D	C	L	X	V	I
Giá trị	1000	500	100	50	10	5	1

- Nhiều ký hiệu có thể được kết hợp lại với nhau để chỉ các số với các giá trị khác chúng. Thông thường người ta quy định các chữ số I, X, C, M, không được lặp lại quá ba lần liên tiếp; các chữ số V, L, D không được lặp lại liên tiếp. Chính vì thế mà có 6 nhóm chữ số đặc biệt được nêu ra trong bảng sau:

Ký tự	CM	CD	XC	XL	IX	IV
Giá trị	900	400	90	40	9	4

- I chỉ có thể đứng trước V hoặc X, X chỉ có thể đứng trước L hoặc C, C chỉ có thể đứng trước D hoặc M.
- Đối với những số lớn hơn (4000 trở lên), một dấu gạch ngang được đặt trên đầu số gốc để chỉ phép nhân cho 1000. **Trong bài tập này bỏ qua các trường hợp này:**

Ký tự	$\bar{V}$	$\bar{X}$	$\bar{L}$	$\bar{C}$	$\bar{D}$	$\bar{M}$
Giá trị	5.000	10.000	50.000	100.000	500.000	1.000.000

- Cách tính: Tính từ trái sang phải giá trị của các chữ số và nhóm chữ số giảm dần.

Ví dụ    **CVI** = 106  
           **CMIV** = 904

## 7 MẢNG HAI CHIỀU (Two array dimension)

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

- Biết cách khai báo biến kiểu mảng và các phép toán trên các phần tử của mảng;
- Xử lý các kỹ thuật cơ bản trên mảng hai chiều với các dữ liệu chuẩn của C/C++ như:
  - Nhập, xuất, liệt kê.
  - Tính tổng các giá trị trên mảng hai chiều kiểu dữ liệu số (nguyên, thực),
  - Tìm kiếm các phần tử nhỏ nhất, lớn nhất, trên mảng, trên dòng/cột.
  - Sắp xếp trên mảng hai chiều.
  - Xoá dòng, xoá cột,
  - Xoay mảng hai chiều theo dòng, theo cột, ...

### 7.1. 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 nhiều chiều là một mảng khác. Mảng nhiều chiều thường được sử dụng để lưu các ma trận, các tọa độ 2 chiều, 3 chiều, ...
- Trong chương này ta chỉ nghiên cứu các vấn đề liên quan đến mảng 2 chiều; việc thao tác trên các mảng 3, 4, ... chiều sẽ được thực hiện tương tự (chỉ cần tổng quát hóa lên).
- Mảng hai chiều được gọi là ma trận khi số dòng và số cột bằng nhau.

### 7.2. Khai báo mảng hai chiều

#### 7.2.1. Cú pháp

##### 7.2.1.1. Khai báo

**<kiểu cơ sở> <tên mảng> [<số dòng >] [<số cột >]**

- Ý nghĩa
  - *Kiểu cơ sở*: là kiểu dữ liệu của mỗi phần tử của mảng.
  - *Tên mảng*: là tên biến mảng, được đặt đúng theo quy tắc đặt tên trong C/C++.
  - *Số dòng*: là một hằng số nguyên, cho biết số lượng dòng tối đa trong.
  - *Số cột*: là một hằng số nguyên, cho biết số lượng cột tối đa trong mảng. Như vậy, số phần tử của mảng = *số dòng* X *số cột*.
- *Ví dụ 7.1*
  - Khai báo mảng hai chiều có tên là **Arr** gồm 8 hàng, 14 cột và kiểu cơ sở là int  
`int Arr [8] [14];`
  - Khai báo mảng hai chiều có tên là **Mang** gồm 10 hàng, 5 cột và kiểu cơ sở là float  
`float Mang [10] [5];`

- Khai báo mảng hai chiều có tên là **charArray** gồm 12 hàng, 30 cột và kiểu cơ sở là char.

```
char charArray[12][30];
```

### 7.2.1.2. Khai báo đồng thời gán giá trị

- Ví dụ 7.2 `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, mảng hai chiều 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

### 7.2.2. 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 [Chỉ số dòng] [Chỉ số cột].

Ví dụ với khai báo cho mảng a [4][6]. Khi đó, mảng gồm 4 x 6= 24 phần tử. Vị trí các phần tử của mảng theo dòng và cột như hình sau:

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

Phần tử a[1][3] là phần tử nằm trên dòng 1 cột 3 của mảng hai chiều ở trên.

- Chỉ số của phần tử mảng:
  - Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.
  - Chỉ số của mảng có thể là một hằng, một biến hay một biểu thức đại số.
- Với cách truy xuất theo kiểu này, Tên biến mảng [Chỉ số dòng] [Chỉ số cột] có thể coi như là một biến có kiểu dữ liệu là kiểu được chỉ ra trong khai báo biến mảng. Vì vậy, các phép toán (hoặc thao tác) trên kiểu dữ liệu trong C/C++ sẽ được áp dụng cho các phần tử biến có cùng kiểu.
- Ví dụ 7.3: với khai báo `int A[3][4] = { {2,3,9,4} , {5,6,7,6} , {2,9,4,7} };`

Gán giá trị của phần tử A[1][2] cho biến x: `int x=A[1][2]; // => x=7`

Gán giá trị 100 cho phần tử A[1][2]: `A[1][2]=100;`

Trước khi lệnh A[1][2]=100 thực hiện					Sau khi lệnh A[1][2]=100 thực hiện				
	0	1	2	3		0	1	2	3
0	2	3	9	4	0	2	3	9	4
1	5	6	7	6	1	5	6	100	6
2	2	9	4	7	2	2	9	4	7

## 7.3. Nhập dữ liệu cho mảng hai chiều

### 7.3.1. Khai báo hằng

```
#define m 20 // m là số dòng
#define n 30 // n là số cột
```

### 7.3.2. 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 gia tri cho phan tu [0][%d]: ",col);
    scanf ("%d" , &a[0][col]) ;
}
```

- Nhập dữ liệu cho dòng thứ hai

```
for (int col=0 ; col<n ; col++)
{
    printf ("Nhap gia tri cho phan tu [1][%d]: ",col);
    scanf ("%d" , &a[1][col]) ;
}
```

...

- Nhập dữ liệu cho dòng thứ m

```
for (int col=0 ; col<n ; col++)
{
    printf ("Nhap gia tri cho phan tu [%d][%d]: ",m-1,col);
    scanf ("%d" , &a[m][col]) ;
}
```

- Tổng quát: nhập dữ liệu cho mảng hai chiều gồm m dòng và n cột

```
void NhapMang (int A[][MAX], int m, int n)
{
    for (int row=0 ; row<m ; row++)
        for (int col=0 ; col<n ; col++)
        {
            printf ("Nhap gia tri cho phan tu [%d][%d]: ",row,col);
            scanf ("%d" , &a[row][col]) ;
        }
}
```

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

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

- 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 mảng hai chiều gồm m dòng n cột lưu trữ các số nguyên

```
void XuatMang(int a[][MAX], 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");
    }
}
```

## 7.5. Ma trận vuông

### 7.5.1. Khái niệm

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

### 7.5.2. Ma trận đơn vị

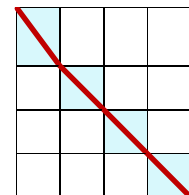
Ma trận đơn vị  $I_n$  là ma trận vuông kích thước  $(n \times n)$  có tất cả các phần tử nằm trên đường chéo chính bằng 1 và những phần tử còn lại bằng 0.

1	0	0
0	1	0
0	0	1

### 7.5.3. Tính chất của ma trận vuông

#### 7.5.3.1. 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:  
**chỉ số dòng = chỉ số cột**
- Ví dụ 7.4 Cho ma trận vuông  $A_{(n \times n)}$ . Xuất các giá trị có trên đường chéo chính



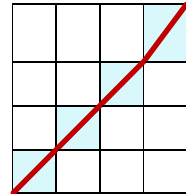
1			
	1		
		1	
			1

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

### 7.5.3.2. Đường chéo phụ

- Đường chéo phụ là đường chéo có:  
**chỉ số cột + chỉ số dòng = số dòng (hoặc số cột) - 1**
- Ví dụ 7.5 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]);
}
```

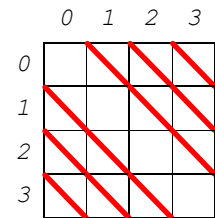


### 7.5.3.3. Đường chéo song song với đường chéo chính hoặc với đường chéo phụ

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

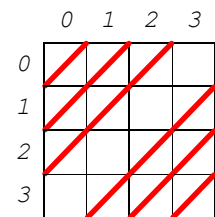
#### 7.5.3.3.1. Đường chéo song song với đường chéo chính

- Tính chất: Các phần tử cùng nằm trên 1 đường chéo sẽ có:  
**chỉ số dòng - chỉ số cột = hằng số**



#### 7.5.3.3.2. Đường chéo song song với đường chéo phụ

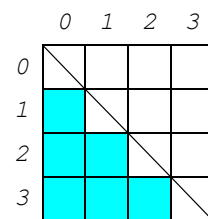
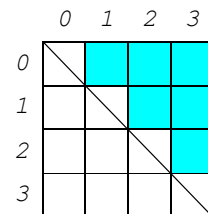
- Tính chất: Các phần tử cùng nằm trên 1 đường chéo sẽ có:  
**chỉ số dòng + chỉ số cột = hằng số**



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

#### 7.5.3.4.1. Đối với đường chéo chính

- Tam giác trên:
  - Tính chất: các phần tử thuộc tam giác trên của đường chéo chính sẽ có:  
**Chỉ số cột > chỉ số dòng**
- Tam giác dưới:
  - Tính chất: các phần tử thuộc tam giác trên của đường chéo chính sẽ có:  
**Chỉ số cột < chỉ số dòng**



#### 7.5.3.4.3. Đối với đường chéo phụ

- Tam giác trên:
  - Tính chất các phần tử thuộc tam giác trên của đường chéo phụ sẽ có:  
Chỉ số cột + chỉ số dòng < số dòng (hoặc số cột) - 1
- Tam giác dưới:
  - Tính chất các phần tử thuộc tam giác dưới của đường chéo phụ sẽ có:  
Chỉ số cột + chỉ số dòng > số dòng (hoặc số cột) - 1

0	0	1	2	3
0				
1				
2				
3				

0	0	1	2	3
0				
1				
2				
3				

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

##### 7.5.3.5.1. Đối với đường chéo chính

- Tính chất Truy xuất các phần tử đối xứng qua đường chéo chính dựa vào tính chất:  
 $A[\text{dòng}][\text{cột}]$  đối xứng với  $A[\text{cột}][\text{dòng}]$

0	0	1	2	3
0			X	
1				
2	X			
3				

##### 7.5.3.5.2. Đối với đường chéo phụ

- Tính chất Truy xuất các phần tử đối xứng qua đường chéo phụ dựa vào tính chất:  
 $A[\text{dòng}][\text{cột}]$  đối xứng với  $A[n-1-\text{cột}][n-1-\text{dòng}]$

0	0	1	2	3
0		X		
1				
2				
3			X	

## 7.6. Khai báo kiểu mảng hai chiều

- Để đơn giản trong việc khai báo mảng hai chiều, ta định nghĩa kiểu mảng hai chiều các phần tử với kiểu dữ liệu bất kỳ như sau:
  - Mảng hai chiều thường (số dòng và số cột khác nhau)
 

```
#define ROWS 100
#define COLS 200
typedef int MANG2CHIEU[ROWS][COLS];
```
  - Ma trận vuông (số dòng và số cột bằng nhau)
 

```
#define SIZE 100
typedef int MATRAN[SIZE][SIZE];
```
- Ví dụ 7.6 Khai báo ma trận các số nguyên A.
 

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



## 7.7. 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 7.7 chương trình sau tạo mảng hai chiều với các giá trị là ngẫu nhiên rồi in mảng hai chiều lên màn hình.

```
#define ROWS 200
#define COLS 200
typedef int MATRAN[ROWS][COLS];
void TaoMangNgauNhiem(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);
    TaoMangNgauNhiem (A, dong,cot);
    printf("\n Ma tran vua duoc tao:\n");
    XuatMaTran(A,dong,cot);
    return 0;
}
```

**//Cách 1: Khai báo kiểu MATRAN ⇒ sử dụng kiểu MATRAN cho tham số**

**void TaoMaTranNgauNhiem (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 TaoMaTranNgauNhiem2 (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");
    }
}
```

## 7.8. 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 (ma trận) hoặc không vuông. Để dùng cho ma trận vuông, ta thay biến d (dòng) và c (cột) bằng n (cạnh của ma trận vuông).

### 7.8.1. Nhập/xuất mảng hai chiều

- Nhập trực tiếp các phần tử của mảng hai chiều:

```
void NhapMang2Chieu (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 từ 0 đến 99 cho các phần tử của mảng hai chiều:

```
void TaoMang2ChieuNgauNhien (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 mảng hai chiều:

```
void XuatMang2Chieu(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");
    }
}
```

## 7.8.2. Liệt kê (xuất các phần tử của mảng thỏa điều kiện cho trước)

### 7.8.2.1. Liệt kê dựa trên giá trị của phần tử

- Liệt kê các số chẵn có trong mảng

```
void LietKeSoChan(int a[][COLS], int dong, int cot)
{
    printf("Cac so chan co trong mang: ");
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            if (A[i][j]%2==0)
                printf("%5d",A[i]);
    }
}
```

- Liệt kê các số trong mảng có giá trị lớn hơn X

```
void LietKeSoChan(int a[][COLS], int dong, int cot, int X)
{
    printf("Cac so trong mang co gia gia tri lon hon %d: ",X);
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            if (A[i][j]>X)
                printf("%5d",A[i][j]);
    }
}
```

- Liệt kê các số nguyên tố có trong mảng

```
void LietKeSoNguyenTo(int a[][COLS], int dong, int cot)
{
    printf("Cac so nguyen to co trong mang 2 chieu: ");
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            printf("%5d",A[i][j]);
    }
}

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

### 7.8.2.2. Liệt kê dựa trên chỉ số (vị trí) của phần tử

- Liệt kê các số tại vị trí có tổng của chỉ số dòng và cột là số chẵn

```
void LietKeSoTaiViTri (int A[][COLS], int dong, int cot)
{
    printf("Cac so nam tai vi tri co tong chi so la so chan:\n");
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            if ((i+j)%2==0)
                printf("A[%d][%d]=%d", i,j,A[i][j]);
    }
}
```

### 7.8.2.3. Liệt kê dựa trên kết hợp giữa chỉ số và giá trị của phần tử

- Liệt kê các số nguyên tố tại vị trí có tổng của chỉ số dòng và cột là số lẻ

```
void LietKeSoNguyenToTaiViTri (int A[][COLS], int dong, int cot)
{
    printf("Cac so nguyen to nam tai vi tri co tong chi so la\n");
    for (int i=0; i<dong; i++)
    {
        for(int j=0;j<cot;j++)
            if( (i+j)%2==1) && (LaSNT(A[i][j]) )
                printf("A[%d][%d]=%d", i,j,A[i][j]);
    }
}
```

### 7.8.3. Tính tổng các phần tử trong mảng hai chiều

#### 7.8.3.1. 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
- 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 mảng hai chiều đến cuối mảng hai chiều

```
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;
}
```

#### 7.8.3.2. Tính tổng theo điều kiện cho trước

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

- Đi từ đầu mảng hai chiều đến cuối mảng hai chiều

```
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;
}
```

- Ứng dụng: 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 ;
}
```

#### 7.8.4. Kỹ thuật đặt cờ hiệu

- Viết hàm kiểm tra xem trong mảng hai chiều 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 mảng hai chiều 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;
}
```

### 7.8.5. Kỹ thuật đặt lính canh

- Viết hàm tìm phần tử nhỏ nhất trong mảng hai chiều.

```
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 mảng hai chiều 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 ;
}
```

### 7.8.6. Đếm

- Viết hàm đếm các phần tử chẵn trong mảng hai chiều.

```
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 mảng hai chiều 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 (IsSNT(a[i][j])==1)
                dem ++ ;
    return dem;
}
```

### 7.8.7. Sắp xếp

Yêu cầu: Viết hàm sắp xếp mảng hai chiều tăng dần từ trên xuống dưới và từ trái sang phải

#### 7.8.7.1. 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%c]>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;
        }
    }
}

```

### 7.8.7.2. Có dùng mảng phụ

```

void SapXep(float a[MAX][], int dong, int cot)
{
    float b[1000];
    int k,i,j;
    // copy mảng hai chiều 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 mảng hai chiều ra mảng một chiều b
    k=0;
    for (i=0;i<dong;i++)
        for(j=0,j<cot;j++)
            a[i][j]= b[k++] ;
}

```

## 7.9. Phần thực hành có hướng dẫn

### 7.9.1. Yêu cầu

Định nghĩa 2 hằng số ROWS=100 và COLS=200. Viết chương trình gồm các hàm chức năng như sau:

- (i)- Viết hàm nhận 2 tham số là chuỗi thông báo và giá trị tối đa (value), hàm cho nhập vào một số nguyên dương X với giá trị X thỏa điều kiện:  $0 < X \leq \text{value}$ . Nếu nhập sai, chương trình báo lỗi và cho nhập lại cho đến khi đúng. Sau khi nhập hoàn tất, hàm trả về số X vừa nhập. Hàm này sẽ được dùng lần lượt để nhập giá trị cho số dòng và số cột của mảng 2 chiều. Nguyên mẫu hàm có dạng như sau:

```
int NhapSo(char str[], int value)
```

- (ii)- Viết hàm nhập vào mảng hai chiều các số nguyên gồm m dòng, n cột. Với tham số của hàm là tên mảng 2 chiều (A), số dòng (m), số cột(n). Nguyên mẫu hàm có dạng như sau:

```
void NhapMang2Chieu (int a[][COLS], int m, int n)
```

- (iii)- Viết hàm xuất mảng hai chiều các số nguyên vừa nhập ở trên.  
`void XuatMang2Chieu(int a[][SIZE], int m, int n)`
- (iv)- Viết hàm tính và trả về tổng các phần tử có trong mảng.
- (v)- Viết hàm tính và trả về tổng các phần tử nằm trên đường biên của mảng.
- (vi)- Viết hàm liệt kê các số may mắn có trong mảng.

X	X	X	X	X
X				X
X				X
X	X	X	X	X
Các phần tử trên đường biên				

### 7.9.2. Hướng dẫn

**Bước 1.** Định nghĩa các hằng số sẽ sử dụng trong chương trình.

```
#define ROWS 100
#define COLS 200
```

**Bước 2.** Viết hàm nhập X (0 < X <= value).

```
int NhapSo(char str[], int value)
{
    int n;
    do
    {
        printf("Nhap so %s (tu 1-100): ",str);
        scanf("%d",&n);
        if ((n<=0)|| (n> value))
            printf("Chi nhan gia tri tu 1 den 100");
    }while((n<=0)|| (n>value));
    return n;
}
```

**Bước 3.** Viết hàm nhập vào mảng gồm m dòng n cột

```
void NhapMang2Chieu (int a[][COLS], int m, int n)
{
    int i,j;
    for (i=0; i<m; i++)
        for (j=0;j<n;j++)
        {
            printf("Nhap gia tri cho phan tu a[%d][%d]: ",i,j);
            scanf("%d",&a[i][j]);
        }
}
```

**Bước 4.** Viết hàm xuất mảng số nguyên m x n phần tử vừa nhập ở trên.

```
void XuatMang2Chieu(int a[][COLS], int m, int n)
{
    printf("Ma tran vua tao:\n");
    for (int i=0; i<m; i++)
    {
        for(int j=0;j<n;j++)
            printf("%5d",a[i][j]);
        printf("\n");
    }
}
```

**Bước 5.** Viết hàm main () kết 3 hàm trên sao cho chạy ổn định rồi mới viết tiếp hàm khác.

```
int main()
{
    int m,n, a[ROWS][COLS];
    m=NhapSo("dong", ROWS);
    n=NhapSo("cot",COLS);
```



```
        NhapMang2Chieu (a, m, n);
        XuatMang2Chieu(a, m, n);
        return 0;
    }
```

**Bước 6.** Viết hàm tính tổng tất cả các phần tử của mảng. Gọi và nhận kết quả trả về từ hàm này trong hàm main để xuất kết quả.

```
int TongMang(int a[][COLS], int m, int n)
{
    int t=0;
    for (int i=0; i<m; i++)
        for(int j=0;j<n;j++)
            t+=a[i][j];
    return t;
}
```

**Bước 7.** Viết hàm Tính tổng các phần tử nằm trên đường biên của mảng. Gọi và nhận kết quả trả về từ hàm này trong hàm main để xuất kết quả.

```
int TongBien(int a[][COLS], int m, int n)
{
    int t=0;
    for (int i=0; i<m; i++)
    {
        for(int j=0;j<n;j++)
            if ( (i==0) || (i==m-1) || (j==0) || (j==n-1) )
                t+=a[i][j];
    }
    return t;
}
```

**Bước 8.** Viết hàm liệt kê các số may mắn có trong mảng. Gọi hàm này trong hàm main để xuất kết quả.

```
bool LaSoMayMan(int k)
{
    while (k>0)
    {
        int so=k%10;
        if ( (so!=6) && (so!=8))
            return false;
        k=k/10;
    }
    return true;
}

void LietKeSoMayMan(int a[][COLS], int m, int n)
{
    printf("\nCac so may man co trong ma tran: ");
    for (int i=0; i<m; i++)
    {
        for(int j=0;j<n;j++)
            if (LaSoMayMan(a[i][j]))
                printf("%5d",a[i][j]);
    }
}
```

Sau khi hoàn tất, nội dung khai báo hằng số, khai báo nguyên mẫu hàm và hàm main sẽ có dạng như sau:

```
#define ROWS 100
#define COLS 200
int NhapSo(char *str);
void NhapMang2Chieu (int a[][COLS], int m, int n);
void XuatMang2Chieu(int a[][COLS], int m, int n);
int TongMang(int a[][SIZE], int m, int n);
int TongBien(int a[][SIZE], int m, int n);
bool LaSoMayMan(int k);
void LietKeSoMayMan(int a[][COLS], int m, int n);

int main()
{
    int m,n, a[ROWS][COLS];
    m=NhapSo("dong",ROWS);
    n=NhapSo("cot", COLS);
    NhapMang2Chieu (a, m, n);
    XuatMang2Chieu(a, m, n);
    printf("\nTong cac so co trong mang la: %d",TongMang(a,m,n));
    printf("\nTong cac so tren duong bien cua mang:
                                                %d",TongBien(a,m,n));

    LietKeSoMayMan(a, m, n);
    return 0;
}
```

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

**Yêu cầu chung:** mỗi yêu cầu được viết thành 1 hàm riêng.

### 7.10.1. Nhập (hoặc tạo) giá trị cho các phần tử trong mảng hai chiều

- (1)- Viết hàm thực hiện nhập giá trị cho các phần tử của mảng hai chiều các số nguyên dương gồm m dòng và n cột ( $5 \leq m, n < 200$ ). Khi nhập sai (m, n không nằm trong giá trị quy định hoặc giá trị của các phần tử trong mảng  $< 0$ , chương trình sẽ báo lỗi và sẽ yêu cầu nhập lại cho đến khi nhập đúng. Nguyên mẫu của hàm được yêu cầu có dạng như sau, trong đó SIZE được khai báo là hằng số với giá trị là 200.

**void NhapMang(int A[][SIZE], int &m, int &n)**

- (2)- Viết hàm khởi tạo giá trị các phần tử là ngẫu nhiên cho mảng hai chiều các số nguyên kích thước m dòng và n cột ( $2 \leq m, n < 50$ ) cho các trường hợp sau:
- Giá trị của các phần tử được phát sinh ngẫu nhiên và nằm trong khoảng từ 0 đến 99.
  - Giá trị của các phần tử được phát sinh ngẫu nhiên và nằm trong khoảng từ -100 đến 100.
  - Giá trị của các phần tử là  $a[i][j] = i * j$ .
  - Giá trị của các phần tử được phát sinh ngẫu nhiên và phải toàn là số lẻ.  
✎ Mở rộng cho các trường hợp số chẵn, số vừa âm vừa lẻ, ...
  - Giá trị của các phần tử được phát sinh ngẫu nhiên và phải toàn là số nguyên tố.  
✎ Mở rộng cho các trường hợp số hoàn thiện, ...).

f. Giá trị của các phần tử được phát sinh ngẫu nhiên sao cho phần tử  $a[i][j]$  chỉ nhận số chẵn khi  $i+j$  là số lẻ và chỉ nhận số lẻ khi  $i+j$  là số chẵn..

☞ *Mở rộng*: cho trường hợp ngược lại:  $i+j$  lẻ chỉ nhận số lẻ;  $i+j$  chẵn chỉ nhận số chẵn.

(3)- Cho mảng hai chiều các số thực  $A(m \times n)$ . Viết hàm xây dựng mảng hai chiều  $B (m \times n)$  từ mảng hai chiều  $A$  trong các trường hợp sau:

- $B[i][j] = \text{abs}(A[i][j])$ .
- $B[i][j]$  = giá trị lớn nhất của dòng  $i$  và cột  $j$  trong mảng hai chiều  $A$ .
- $B[i][j]$  = số lượng phần tử dương xung quanh ( $A[i][j]$ ) trong mảng hai chiều  $A$ .  
Như vậy,  $0 \leq B[i][j] \leq 8$ .
- $B[i][j]$  = tổng các số xung quanh ( $A[i][j]$ ) trong mảng hai chiều  $A$ .

### 7.10.2. Xuất

(4)- Viết hàm xuất mảng hai chiều các số nguyên  $A(m \times n)$  trong các trường hợp sau:

5	18	4	7		5	18	4	7		5	18	4	7		5	X	X	7
2	6	11	8		2			11		2	0	0	8		2	X	11	X
19	15	20	1			15			1	19	0	0	1		19	X	X	X
16	3	17	14		16			17		16	0	0	14		X	3	17	X
10	13	9	12			13			12	10	13	9	12		X	13	X	X
Mảng ban đầu					Kết quả thực hiện câu a					Kết quả thực hiện câu b					Kết quả thực hiện câu c			

- Thay giá trị của phần tử có chỉ số dòng + chỉ số cột là số chẵn bằng các khoảng trắng, các số khác giữ nguyên giá trị.
- Thay giá trị của phần tử không nằm trên đường biên bằng số 0 (zero), các giá trị trên đường biên được giữ nguyên giá trị.
- In ra các giá trị là số nguyên tố, các số khác được thay bằng ký tự X.

(5)- Giả sử định nghĩa dãy tăng trên mỗi dòng là dãy thỏa cả 2 điều kiện sau:

- Có ít nhất 3 số liên tiếp nhau.
- 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 (các số không có trong dãy tăng được in bằng ký tự X, ngược lại các có trong dãy tăng được in bằng chính giá trị của số đó):

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

Mảng hai chiều 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

### 7.10.3. Tính tổng/trung bình

#### 7.10.3.1. Thao tác trên toàn bộ các phần tử có trong mảng hai chiều

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

- Tính tổng tất cả các số
- 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.

- (8)- Tính giá trị trung bình cộng
- (9)- Tính giá trị trung bình cộng các số lẻ
- (10)- Tính tổng các số nguyên tố.
- (11)- 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  $1+2+3 = 6$**

### 7.10.3.2. Thao tác trên dòng có trong mảng hai chiều

- (12)- Tính tổng tất cả các số trên mỗi dòng. Sau đó in ra giá trị tổng dòng lớn nhất.
- (13)- Tính tổng các số chẵn trên mỗi dòng. Sau đó in ra giá trị tổng các số chẵn lớn nhất.
- (14)- Tính tổng các số âm ( $<0$ ) trên mỗi dòng. Sau đó in ra giá trị tổng âm lớn nhất.
- (15)- Tính giá trị trung bình cộng trên mỗi dòng. Sau đó in ra giá trị tổng trung bình lớn nhất.
- (16)- Tính giá trị trung bình nhân trên mỗi dòng. Sau đó in ra giá trị trung bình nhân lớn nhất.
- (17)- Tính tổng các số nguyên tố trên mỗi dòng. Nếu dòng không có số nguyên tố, xem như tổng của dòng đó là 0.
- (18)- (\*) In ra số nguyên tố lớn nhất trên mỗi dòng. Nếu dòng không có số nguyên tố, thì để trống cả dòng. Nếu có từ 2 số nguyên tố cùng lớn nhất trở lên thì chỉ in số đầu tiên (tính từ trái sang phải)

Từ bài 12 đến 18 thực hiện in ra màn hình như minh họa sau đây của bài 12:

Mảng hai chiều ban đầu

4	2	9	7
2	2	9	4
7	8	7	4
4	8	8	6

Kết quả in ra màn hình  $\Rightarrow$

4	2	9	7	Tổng dòng =	22
2	2	9	4	Tổng dòng =	17
7	8	7	4	Tổng dòng =	26
4	8	8	6	Tổng dòng =	26

Tổng dòng lớn nhất là 26, dòng lớn nhất là dòng 2 và 3

Minh họa kết quả in ra màn hình của bài 18 (ký hiệu  $\cup$  đại diện cho khoảng trắng)

Mảng hai chiều ban đầu

4	2	9	7
2	2	9	4
7	8	7	4
4	8	8	6

Kết quả in ra màn hình  $\Rightarrow$

$\cup$	$\cup$	$\cup$	7
2	$\cup$	$\cup$	$\cup$
7	$\cup$	$\cup$	$\cup$
$\cup$	$\cup$	$\cup$	$\cup$

### 7.10.3.3. Thao tác trên các cột có trong mảng hai chiều

Thực hiện tương tự như các yêu cầu có trong bài tập 7.10.3.2, nhưng thay dòng thành cột.

### 7.10.3.4. Thao tác trên các phần tử nằm trên đường biên trong mảng hai chiều

- (19)- Tính tổng:
  - a. Tất cả các số trên đường biên.
  - b. Tất cả các số KHÔNG thuộc đường biên.

### 7.10.3.5. Các phép toán trên mảng hai chiều

- (20)- Tính tổng hai ma trận A(m X n) và B(m x l). In kết quả ra màn hình.
- (21)- Tính hiệu hai ma trận A(m X n) và B(m x l). In kết quả ra màn hình.
- (22)- Tính tích hai ma trận A(m X n) và B(m x l). In kết quả ma trận tích C(n x l).

**7.10.5. Kỹ thuật đặt linh canh**

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

**7.10.5.1. Tìm kiếm trên toàn mảng hai chiều**

- (23)- Thay thế những phần tử có giá trị x bằng giá trị y, với x và y là 2 trong các tham số của hàm.
- (24)- Lớn nhất trong mảng hai chiều
- (25)- Tìm số chẵn xuất hiện đầu tiên trong mảng.
- (26)- Tìm giá trị dương lớn nhất trong mảng.
- (27)- Tìm giá trị âm lớn nhất trong mảng.
- (28)- Tìm giá trị lớn thứ nhì trong mảng.
- (29)- Tìm giá trị dương đầu tiên trong mảng.
- (30)- Tìm giá trị nguyên tố lớn nhất trong mảng
- (31)- Tìm số nguyên tố đầu tiên trong mảng.
- (32)- Tìm số chẵn lớn nhất trong mảng.
- (33)- Tìm giá trị dương nhỏ nhất trong mảng.
- (34)- Tìm số nguyên tố lớn nhất trong mảng.
- (35)- Tìm một chữ số xuất hiện nhiều nhất trong mảng.
- (36)- Tìm giá trị xuất hiện nhiều nhất trong mảng.
- (37)- Tìm số hoàn thiện nhỏ nhất trong mảng.
- (38)- Tìm số xuất hiện nhiều nhất trong mảng.

Ví dụ cho mảng hai chiều

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

- (39)- Tìm số chính phương lớn nhất trong mảng.

➤ **Lưu ý:** Các bài tập sau đây (39-45) có thể không có giá trị cần tìm xuất hiện trong mảng, do đó hàm nên có kiểu dữ liệu trả về là bool và hàm cần nhận thêm 1 tham biến để nhận giá trị khi việc tìm thấy giá trị có trong yêu cầu.

Ví dụ: với yêu cầu tìm giá trị âm lẻ lớn nhất, nguyên mẫu hàm có thể là:

TimAmLeLonNhat(int A[][SIZE], int d, int c, int &ketqua)

- (40)- Âm lẻ lớn nhất.
- (41)- Phần tử chẵn dương và nhỏ nhất trong mảng.
- (42)- Chẵn cuối cùng trong mảng.
- (43)- Tìm số đầu tiên nhỏ hơn x (với x là tham số được truyền cho hàm).
- (44)- Số nguyên tố xuất hiện đầu tiên (tính theo chiều từ trên xuống dưới và từ trái sang phải).

➤ **Mở rộng:** Số nguyên tố xuất hiện cuối cùng.

- (45)- Số hoàn thiện đầu tiên trong mảng.
- (46)- Số hoàn thiện lớn nhất trong mảng.

### 7.10.5.2. Tìm kiếm trên dòng/cột/đường chéo của mảng hai chiều

- (47)- Tìm tất cả các cột có chứa giá trị X.
- (48)- Tìm tất cả các dòng có chứa giá trị X.
- (49)- Tìm vị trí (chỉ số dòng + chỉ số cột) của phần tử mà giá trị của phần tử đó bằng tổng tất cả các phần tử xung quanh liền kề.
- (50)- Tìm cột có tổng nhỏ nhất trong mảng hai chiều.
- (51)- Tìm dòng có tổng lớn nhất trong mảng hai chiều.
- (52)- Tìm các dòng có chứa giá trị lớn nhất của mảng hai chiều trong mảng.
- (53)- Tìm giá trị lớn nhất trên một dòng i trong mảng.
- (54)- Tìm giá trị nhỏ nhất trên một cột j trong mảng.
- (55)- (\*) Tìm ma trận con (cạnh 2x2) có tổng lớn nhất trong mảng hai chiều mx n.
- (56)- (\*) Tìm hai giá trị gần nhau nhất trong mảng.

9	2	1	2	15	5	7
4	3	9	1	3	4	4
7	5	0	1	0	7	4
0	1	2	0	1	0	6
10	4	5	2	22	4	4
2	3	10	5	7	3	72
6	8	55	19	4	2	1

Bài 49: giá trị của phần tử bằng tổng tất cả các phần tử xung quanh liền kề

### 7.10.5.3. Tìm kiếm và liệt kê

- (57)- Liệt kê các dòng có từ 3 số liên tiếp nhau trở lên cùng chứa giá trị chẵn.
- (58)- Liệt kê các dòng có chứa giá trị âm.
- (59)- Liệt kê các dòng có chứa số nguyên tố.
- (60)- Liệt kê các dòng có chứa giá trị chẵn.
- (61)- Liệt kê các cột có chứa số chính phương.
- (62)- Liệt kê các dòng toàn âm.
- (63)- Liệt kê chỉ số của các dòng chứa toàn giá trị chẵn.
- (64)- Liệt kê các dòng có giá trị giảm dần.
- (65)- Liệt kê các dòng có giá trị tăng dần.
- (66)- Liệt kê các dòng chứa cả 3 loại giá trị : giá trị âm, giá trị dương và giá trị 0 (phần tử trung hòa).
- (67)- Liệt kê các dòng có tổng dòng lớn nhất.
- (68)- Liệt kê các cột có tổng cột nhỏ nhất.
- (69)- Liệt kê các dòng có nhiều số chẵn nhất.
- (70)- Liệt kê các dòng có nhiều số nguyên tố nhất.
- (71)- Liệt kê các dòng có nhiều số hoàn thiện nhất.
- (72)- (\*) Liệt kê các cột nhiều chữ số nhất. Ví dụ: 17 gồm 2 chữ số, 903 gồm 3 chữ số, ...

## 7.10.6. Đếm

### 7.10.6.1. Đếm giá trị trong toàn mảng hai chiều

- (73)- Số lượng số dương
- (74)- Số lượng số âm
- (75)- Số lượng số chẵn và dương.
- (76)- Số lượng số âm và lẻ
- (77)- Nhỏ nhất trong mảng.

- (78)- Số lượng số x xuất hiện trong mảng (với x là tham số được truyền cho hàm).  
 (79)- Số lượng số nhỏ hơn x (với x là tham số được truyền cho hàm).  
 (80)- Số lượng số nguyên tố.  
 (81)- Số lượng số nguyên tố nhỏ hơn x (với x là tham số được truyền cho hàm).  
 (82)- Đếm số lượng số dương trên các cạnh biên của mảng.

Ví dụ cho mảng hai chiều

-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

- (83)- (\*) Đếm số lượng phần tử cực đại trong mảng. Một phần tử được gọi là cực đại khi nó lớn hơn các phần tử xung quanh.  
 (84)- (\*) Đếm số lượng phần tử cực trị trong mảng. 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.  
 (85)- (\*) Đếm số lượng giá trị phân biệt có trong mảng. Lưu ý: Nếu có k phần tử ( $k \geq 1$ ) trong mảng hai chiều bằng nhau thì ta chỉ tính là 1.  
 (86)- (\*) Tính tổng các phần tử cực trị trong mảng. 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.  
 (87)- (\*) Đếm số lượng giá trị “Yên Ngựa” trên mảng hai chiều. Một phần tử được gọi là Yên Ngựa khi nó lớn nhất trên dòng và nhỏ nhất trên cột.  
 (88)- (\*) Đếm số lượng giá trị “Hoàng Hậu” trên mảng hai chiều. Một phần tử được gọi là Hoàng Hậu khi nó lớn nhất trên dòng, trên cột và hai đường chéo đi qua nó.

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

- (89)- Đếm giá trị âm có trên từng dòng.  
 (90)- Đếm giá trị là số nguyên tố có trên từng cột.  
 (91)- Đếm số lượng dòng giảm trong mảng.  
 (92)- Đếm số lượng phần tử cực đại trong mảng.  
 (93)- Đếm số lượng giá trị nhỏ nhất trong mảng.  
 (94)- Đếm số lượng giá trị chẵn nhỏ nhất trong mảng.

### 7.10.7. Sắp xếp

#### 7.10.7.1. Sắp xếp trên toàn mảng hai chiều

- (95)- Sắp xếp mảng hai chiều theo thứ tự tăng dần từ trên xuống dưới và từ trái qua phải theo phương pháp DỪNG mảng phụ.  
 (96)- Sắp xếp mảng hai chiều theo thứ tự tăng dần từ trên xuống dưới và từ trái qua phải theo phương pháp KHÔNG DỪNG mảng phụ.  
 (97)- Sắp xếp tất cả các cột trong mảng hai chiều sao cho các phần tử trên mỗi cột giảm dần từ trên xuống dưới (vị trí cột của các phần tử không đổi)  
 (98)- Sắp xếp các phần tử trong mảng hai chiều theo yêu cầu sau:  
 - Cột có chỉ số chẵn giảm dần từ trên xuống.  
 - Cột có chỉ số lẻ tăng dần từ trên xuống.

- (99)- Sắp xếp các phần tử trong mảng hai chiều theo yêu cầu sau:
- Dòng có chỉ số chẵn tăng dần.
  - Dòng có chỉ số lẻ giảm dần.
- (100)- Sắp xếp các giá trị âm của mảng hai chiều tăng dần, các giá trị dương giảm và các phần tử có giá trị bằng 0 giữ nguyên vị trí.
- (101)- Sắp xếp các giá trị chẵn của mảng hai chiều số nguyên tăng dần, các giá trị lẻ giảm dần.
- (102)- Sắp xếp các giá trị dương nằm trên biên mảng hai chiều tăng dần theo chiều kim đồng hồ.
- (103)- Sắp xếp các giá trị dương nằm trên biên mảng hai chiều các số thực tăng dần theo chiều kim đồng hồ.
- (104)- Cho mảng hai chiều vuông chứa các số thực A cạnh là n (với  $n \geq 3$ ). Sắp xếp các giá trị trong mảng hai chiều tam giác trên (không tính đường chéo) tăng dần từ trên xuống dưới và từ trái sang phải
- (105)- Sắp xếp các phần tử dương trong mảng hai chiều 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).

#### 7.10.7.2. Sắp xếp dòng/cột/đường chéo của mảng hai chiều

- (106)- Sắp xếp các dòng trên mảng hai chiều theo thứ tự tăng dần.
- (107)- Sắp xếp các cột trên mảng hai chiều theo thứ tự giảm dần.
- (108)- Sắp xếp các phần tử trên một dòng tăng dần từ trái sang phải.
- (109)- Sắp xếp các phần tử trên một dòng giảm dần từ trái sang phải.
- (110)- Sắp xếp các phần tử trên một cột tăng dần từ trên xuống dưới.
- (111)- Sắp xếp các dòng trong mảng hai chiều theo tiêu chuẩn sau: dòng có tổng dòng nhỏ hơn nằm ở trên và dòng có tổng dòng lớn hơn bằng nằm ở dưới.
- (112)- Sắp xếp các phần tử trên đường chéo chính tăng dần (từ trên xuống dưới)
- (113)- Sắp xếp các phần tử trên đường chéo phụ giảm dần (từ trên xuống dưới).
- (114)- Sắp xếp “các dòng” tăng dần theo tổng dòng

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

- (115)- Hoán vị hai dòng r1, r2 trong mảng hai chiều. Với r1 và r2 do người dùng nhập vào. Cần kiểm tra điều kiện  $0 \leq r1, r2 < n$
- (116)- Hoán vị hai cột c1, c2 trong mảng hai chiều. trong mảng hai chiều. Với c1 và c2 do người dùng nhập vào. Cần kiểm tra điều kiện  $0 \leq c1, c2 < n$
- (117)- Viết hàm hoán vị dòng có tổng lớn nhất với dòng có tổng nhỏ nhất.
- (118)- Dịch xuống xoay vòng các hàng trong mảng hai chiều.

Ví dụ

Mảng hai chiều ban đầu				Kết quả dịch xuống xoay vòng các hàng			
87	75	62	54	-20	67	53	23
46	40	33	28	87	75	62	54
20	18	15	10	46	40	33	28
-20	67	53	23	20	18	15	10

- (119)- Dịch lên xoay vòng các hàng trong mảng hai chiều.



(120)- Dịch trái xoay vòng các cột trong mảng hai chiều.

Ví dụ

Mảng hai chiều 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

(121)- Dịch phải xoay vòng các cột trong mảng hai chiều.

(122)- Dịch phải xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên mảng hai chiều.

(123)- Dịch trái xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên mảng hai chiều.

### 7.10.9. Thêm – Xoá dòng/cột

(124)- Viết hàm xoá một dòng r trên mảng hai chiều.

(125)- Viết hàm xoá một cột c trên mảng hai chiều.

(126)- Viết hàm xoá dòng có tổng lớn nhất trên mảng hai chiều.

(127)- Viết hàm tìm và thay thế các phần tử chẵn trong mảng hai chiều bằng ước số nhỏ nhất của nó.

(128)- Viết hàm chèn một dòng chứa toàn giá trị X vào vị trí r trên mảng hai chiều.

(129)- Viết hàm chèn một cột chứa toàn giá trị X vào vị trí c trên mảng hai chiều.

### 7.10.10. Ma trận vuông

#### 7.10.10.1. Tạo ma trận

(130)- Viết hàm tạo giá trị ngẫu nhiên cho các phần tử trong 1 ma trận vuông.

(131)- Viết hàm phát sinh ngẫu nhiên ma trận vuông (n X 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)

(132)- ✎ Mở rộng cho trường hợp số giảm dần.

(133)- 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 X 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.

(134)- Tính tổng các phần tử nằm trên đường chéo chính có trong mảng.

(135)- Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng.

(136)- Đưa các giá trị chẵn về đầu ma trận vuông các số nguyên.

(137)- Viết chương trình nhập/ xuất một ma trận vuông số thực n x n (n<=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).

(138)- 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)

(139)- 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).

### 7.10.10.2. Xuất

(140)- Cho ma trận vuông ( $n \times n$ ) các số nguyên. Viết hàm in ra:

- 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".
- Các phần tử nằm trên 2 đường chéo chính và phụ, các phần tử khác in khoảng trắng.
- 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).

(141)- Xuất theo hình cho trước:

- 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. Mỗi đường chéo in trên 1 dòng riêng. Ví dụ với ma trận đã cho trong hình, kết quả in ra:

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Ma trận ban đầu

10 7 1 5 4 2 9 13 16 11 15 8 6 3 12 14

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Yêu cầu của bài 141.a

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Yêu cầu của bài 141.b

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Yêu cầu của bài 141.c

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Yêu cầu của bài 141.d

- In các giá trị các phần tử của ma trận theo đường zigzag ngang. Ví dụ với ma trận đã cho trong hình, kết quả in ra:

10 7 5 9 15 13 4 1 2 16 8 3 14 12 6 11

- c. In các giá trị các phần tử của ma trận theo đường zigzắc chéo. Ví dụ với ma trận đã cho trong hình, kết quả in ra:

10 7 1 2 4 5 9 13 16 11 6 8 15 3 12 14

- d. 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 với ma trận đã cho trong hình, kết quả in ra:

10 7 5 9 15 3 14 12 6 11 2 1 4 13 8 16

### 7.10.10.3. Tính tổng – trung bình

#### 7.10.10.3.1. Đường chéo chính

(142)- Tính tổng tất cả các số trên đường chéo chính.

(143)- Tính tổng các số lẻ trên đường chéo chính.

⇒ Mở rộng:

- Tính tổng các số dương ( $\geq 0$ ) trên đường chéo chính.
- Tính giá trị trung bình cộng trên đường chéo chính.
- Tính tổng các số nguyên tố trên đường chéo chính.

#### 7.10.10.3.2. Đường chéo phụ

(144)- Tính tổng tất cả các số trên đường chéo phụ.

(145)- Tính tổng các số nguyên tố trên đường chéo phụ.

⇒ Mở rộng:

- Tính tổng các số dương ( $\geq 0$ ) trên đường chéo phụ.
- Tính giá trị trung bình cộng trên đường chéo phụ.
- Tính tổng các số lẻ trên đường chéo phụ.

#### 7.10.10.3.3. Tam giác trên và tam giác dưới

Lần lượt thực hiện các yêu cầu sau cho tam giác trên và tam giác dưới ở cả 2 dạng đường chéo chính và đường chéo phụ:

(146)- Tính tổng tất cả các số có trong tam giác.

(147)- Tính tổng các số lẻ có trong tam giác.

(148)- Tính tổng các số dương ( $\geq 0$ ) có trong tam giác.

(149)- Tính giá trị trung bình cộng các số trong tam giác.

(150)- Tính tổng các số hoàn thiện trong tam giác.

### 7.10.10.4. Tìm kiếm trên phạm vi có trong yêu cầu

(151)- Tìm giá trị X trên đường chéo chính.

(152)- Tìm giá trị X trên đường chéo phụ.

(153)- Tìm giá trị X trong tam giác trên của đường chéo chính.

(154)- Tìm giá trị X trong tam giác dưới của đường chéo chính.

(155)- Tìm giá trị X trong tam giác trên của đường chéo phụ.

(156)- Tìm giá trị X trong tam giác dưới của đường chéo phụ.

(157)- Tìm giá trị lớn nhất trong tam giác trên của đường chéo dạng 1

(158)- Tìm giá trị nhỏ nhất trong tam giác dưới của đường chéo dạng 2

(159)- Tìm giá trị lớn nhất trên đường chéo chính.

(160)- Tìm giá trị lớn nhất trên đường chéo phụ .

#### 7.10.10.5. Đếm

- (161)- Đếm giá trị là số chẵn trên đường chéo chính.  
 (162)- Đếm giá trị là số nguyên tố trên đường chéo phụ.  
 (163)- Đếm giá trị là số chính phương trong tam giác trên của đường chéo chính.  
 (164)- Đếm giá trị là số hoàn thiện trong tam giác dưới của đường chéo chính.  
 (165)- Đếm giá trị là số may mắn trong tam giác trên của đường chéo phụ.  
 (166)- Đếm giá trị là số lẻ trong tam giác dưới của đường chéo phụ.  
 (167)- Đếm số lượng cặp giá trị đối xứng nhau qua đường chéo chính.

#### 7.10.10.6. Xoay ma trận

- (168)- Xoay ma trận một góc 90 độ theo 2 cách thuận chiều kim đồng hồ và ngược chiều kim đồng hồ.  
 (169)- Xoay ma trận một góc 180 độ.  
 (170)- Xoay ma trận một góc 270 độ.

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

- (171)- Viết hàm sắp xếp ma trận theo các đường chéo song song với đường chéo phụ của ma trận  
 (172)- Viết hàm sắp xếp ma trận theo đường ziczắc ngang.  
 (173)- Viết hàm sắp xếp ma trận theo đường ziczắc chéo  
 (174)- Viết hàm sắp xếp ma trận theo đường xoắn ốc từ ngoài vào trong theo chiều kim đồng hồ.

10	7	5	9
1	4	13	15
2	16	8	3
11	6	12	14

Ma trận ban đầu

1	2	4	7
3	5	8	11
6	9	12	14
10	13	15	16

Yêu cầu của bài 171

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

Yêu cầu của bài 172

1	2	6	7
3	5	8	13
4	9	12	14
10	11	15	16

Yêu cầu của bài 173

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Yêu cầu của bài 174

#### 7.10.10.8. Thao tác khác trên ma trận

- (175)- 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)  
 (176)- (\*) Cho ma trận vuông A (n x n) các số thực. Hãy tìm ma trận vuông B (k\*k) sao cho tổng các giá trị trên ma trận vuông này là lớn nhất. Với  $2 \leq k \leq n$ .  
 (177)- (\*) 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/phụ đều bằng nhau.

#### 7.10.11. Kỹ thuật xử lý khác trên mảng hai chiều

- (178)- Hãy biến đổi mảng hai chiều bằng cách thay các phần tử chứa giá trị âm bằng giá trị tuyệt đối của chính phần tử đó.

- (179)- Hãy biến đổi mảng hai chiều các số thực bằng cách thay các giá trị bằng giá trị nguyên gần nó nhất (làm tròn số).
- (180)- Chiếu gương mảng hai chiều theo trục ngang theo 2 trường hợp:
- Chiếu nửa trên xuống nửa dưới (bỏ giá trị cũ của nửa dưới).
  - Chiếu nửa dưới lên nửa trên (bỏ giá trị cũ của nửa trên).
- (181)- Chiếu gương mảng hai chiều theo trục dọc.
- Chiếu nửa trái sang nửa phải (bỏ giá trị cũ của nửa phải).
  - Chiếu nửa phải sang nửa trái (bỏ giá trị cũ của nửa trái).
- (182)- Cho mảng 2 chiều chỉ chứa một trong 2 giá trị 0 hoặc -1. Một nhóm các ô chứa giá trị sẽ cùng thuộc một miền liên thông khi các ô trên, dưới, phải, trái của ô đang xét cũng chứa giá trị -1. Viết chương trình xác định số lượng miền liên thông có trong mảng 2 chiều đã cho.

1	0	-1	-1	-1
0	-1	-1	0	0
0	0	-1	-1	0
0	0	0	0	-1
0	-1	0	-1	0
0	0	-1	0	-1
-1	0	0	0	-1

Hình a: mảng hai chiều chỉ chứa giá trị 0 & -1

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

Hình b: đánh số thứ tự 8 miền liên thông của mảng a

- (183)- Nhập vào mảng hai chiều số thực cấp m X n. Tạo menu thực hiện các thao tác trên mảng hai chiều:
- 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±k]})
- (184)- Nhập vào mảng hai chiều số thực cấp m X n. Tạo menu thực hiện các thao tác trên mảng hai chiều:
- 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.
- (185)- 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, xóa 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.
  - Xóa cột có nhiều số nguyên tố nhất (nếu có 2 cột có cùng số lượng số nguyên tố thì xóa cột đầu tiên).

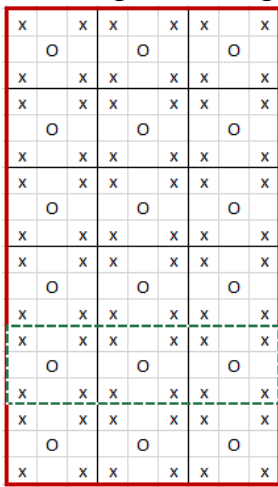
### 7.10.12. Thao tác trên nhiều mảng hai chiều

- (186)- Dùng mảng phụ, viết hàm xuất
- Các giá trị chẵn theo thứ tự giảm dần.
  - Số nguyên tố trong mảng hai chiều theo thứ tự tăng dần.
  - Các giá trị âm theo thứ tự giảm dần.
- (187)- Viết chương trình cho nhập chiều dài và chiều rộng của nền nhà, kích thước cạnh của viên gạch men lát nền (hình vuông) và kiểu của viên gạch men. Viết chương trình thực hiện lát nền nhà bằng các viên gạch men đã có.

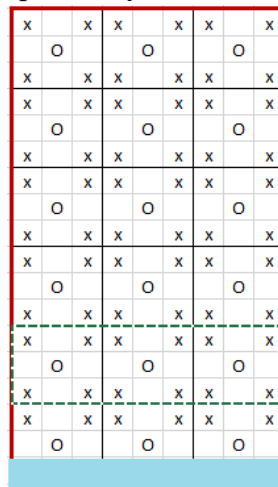


Ví dụ với hình dạng và kiểu bông của viên gạch vuông kích thước 3 x 3: `char g[3][3]= { {'X',' ','X'}, {' ','O',' '}, {'X',' ','X'} }`;

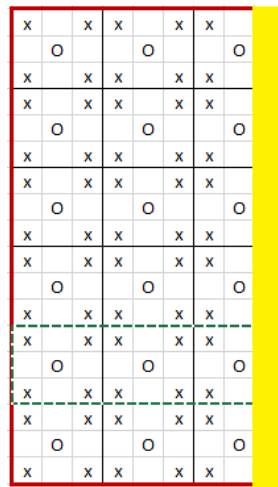
Lưu ý: Khi dùng viên gạch này lên các nền nhà có kích thước khác nhau sẽ có 1 trong các trường hợp sau xảy ra:



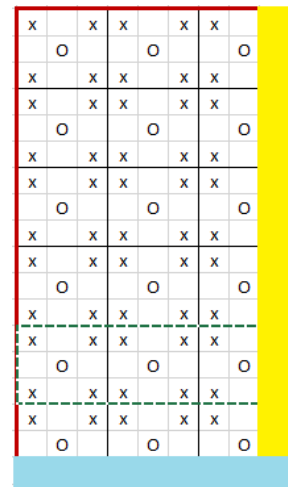
Trường hợp 1: Nền nhà rộng=9, dài=15 => không cần “cắt” gạch



Trường hợp 2: Nền nhà rộng=9, dài=14 => chiều dài không vừa khít nên cần “cắt” tất cả các viên gạch dưới cùng



Trường hợp 3: nền nhà: rộng=8, dài=15 => chiều rộng không vừa khít nên cần “cắt” tất cả các viên gạch bên trái.



Trường hợp 4: nền nhà: rộng=8, dài=14 => chiều dài và chiều rộng đều không vừa khít nên cần “cắt” tất cả các viên gạch dưới cùng và bên trái.

### 7.10.13. Game

- (188)- Viết chương trình bắn tàu với máy.
- (189)- Viết chương trình Puzzle 15 số.
- (190)- Viết chương trình minh họa việc tạo game Minesweeper.

Minh họa 1 game ở cấp độ Beginner



Game trong thực tế

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

Yêu cầu kết quả xuất ra màn hình

Đầu tiên, chương trình cho phép người dùng chọn cấp độ. Tùy vào cấp độ được chọn, chương trình sẽ tạo ra mảng 2 chiều với số dòng (row), số cột (col) và số “bom” tương ứng.

Nếu người dùng chọn 1 trong các cấp độ Beginner, Intermediate, Expert, chương trình sẽ có kích thước và số bom theo quy định cho trong bảng sau. Ngược lại, nếu người dùng chọn cấp độ *Custom*, chương trình sẽ cho người dùng nhập số dòng, số cột, số “bom” cần đặt (theo quy định cho trong bảng):

Cấp độ	row	col	Số bom
Beginner	9	9	10
Intermediate	16	16	40
Expert	16	30	99
Custom (do người dùng tự chọn)	$9 \leq row \leq 24$	$9 \leq col \leq 30$	$< row * col$

Sau khi tạo hoàn tất ma trận, chương trình hiển thị ra màn hình:

- Vị trí có bom (được phát sinh ngẫu nhiên và không trùng nhau để đảm bảo số lượng bom đúng quy định).
- Số nguyên trong mỗi ô: cho biết số bom có trong 8 ô xung quanh ô đó (nếu số lượng=0 thì hiện khoảng trắng).

Lưu ý: chương trình chỉ dừng ở việc tạo ra game, chứ chưa xử lý việc chơi game.

(191)- 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.



## 8 KIỂU DỮ LIỆU CÓ CẤU TRÚC (Struct data type)

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.

### 8.1. 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).

### 8.2. Khai báo kiểu cấu trúc

#### 8.2.1. Khai báo

```
typedef struct <Tên cấu trúc>
{
    <kiểu dữ liệu> <trường 1> ;
    < kiểu dữ liệu > < trường 2> ;
    ...
    < kiểu dữ liệu > < trường n> ;
};
```

Trong đó:

- <Tên cấu trúc>: là tên của kiểu cấu trúc sẽ được dùng. Tên này phải được đặt theo quy tắc đặt tên cho các định danh trong C.
- < kiểu dữ liệu > < 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 cho các định danh trong C.
- Ví dụ 8.1: Khai báo một kiểu struct có tên **NgayThang** có các thành phần là: ngày, tháng, năm.

```
typedef struct DATE
{
    unsigned char    ngay;
    unsigned char    thang;
    int              nam;
};
```



- Ví dụ 8.2 : Khai báo một kiểu struct có tên KETQUA có các thành phần là: Mã số sinh viên, Mã số môn học, Lần thi, điểm thi.

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

- Ví dụ 8.3: Định nghĩa cấu trúc dữ liệu cho phân số

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

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

a.tu	a.mau
------	-------

### 8.2.2. Khai báo và 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 8.4:
 

```
DATE birth = {12, 10, 2000};
KETQUA kq = {"1234567", "Toan", 1, 4.5};
```

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

Ví dụ 8.5: 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
{
    unsigned char ngay, 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, cần lưu ý: Kiểu dữ liệu cấu trúc được sử dụng phải khai báo trước (đặt ở phía trên).

### 8.3. Sử dụng biến cấu trúc

#### 8.3.1. Khai báo biến kiểu cấu trúc

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 8.6     `struct DATE x ; //Khai bao bien x co kieu du lieu 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 8.7     `DATE x ; // Khai bao bien x co kieu DATE`

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

##### - Cú pháp

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


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

##### - Ví dụ 8.8: 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;
}
```

## 8.4. 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 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.

### 8.4.1. Phân số

Yêu cầu: Khai báo kiểu dữ liệu biểu diễn khái niệm phân số trong toán học và định nghĩa các hàm:

- (i). Nhập 1 phân số: cần thực hiện:
  - Rút gọn phân số, ví dụ  $4/8$  được rút gọn thành  $1/2$ .
  - Khi mẫu số là số âm, cần khử dấu trừ của mẫu số bằng cách nhân cả tử và mẫu cho -1.
  - Khi tử số = 0 thì mẫu số sẽ được gán = 1.
- (ii). Xuất 1 phân số dưới dạng tử/mẫu, ví dụ  $-2/3$ .
- (iii). Tính tổng, hiệu, tích, thương của 2 phân số. Sau khi tính toán cũng cần rút gọn phân số kết quả.
- (iv). So sánh 2 phân số,

```
struct PHANSO
{
    int tu;
    int mau;
};
```

```
// Khai báo nguyên mẫu hàm
void NhapPS(PHANSO &x);
void XuatPS(PHANSO x);
int TimUSCLN(int a, int b);
PHANSO Tong2PhanSo(PHANSO x, PHANSO y);
void Tong (PHANSO x, PHANSO y);
void RutGonPS (PHANSO &p);
int main()
{
    PHANSO A, B, C;
    printf("Nhap phan so thu 1");
    NhapPS(A);
    printf("Nhap phan so thu 2");
    NhapPS(B);
    Tong(A, B);
    return 0;
}
void NhapPS(PHANSO &p)
{
    printf("\nNhap tu so: ");
    scanf("%d", &p.tu);
    do
    {
        printf("Nhap mau so (!=0): ");
        scanf("%d", &p.mau);
        if (p.mau == 0)
            printf("Mau so phai khac zero (0)\n");
    } while (p.mau == 0);
    RutGonPS(p);
}
int TimUSCLN(int a, int b)
{
    a=abs(a);
    b=abs(b);
    while (a != b)
        if (a>b)
            a -= b;
        else
            b -= a;
    return a;
}
void XuatPS(PHANSO x)
{
    printf("%d/%d", x.tu, x.mau);
}
PHANSO Tong2PhanSo(PHANSO x, PHANSO y)
{
    PHANSO C;
    C.mau = x.mau*y.mau;
    C.tu = (x.tu*y.mau) + (y.tu*x.mau);
    RutGonPS(C);
    return C;
}
```

```

void RutGonPS (PHANSO &p)
{
    if (p.tu != 0)
    {
        int usc = TimUSCLN(abs(p.tu), abs(p.mau));
        p.tu /= usc;
        p.mau /= usc;
        // nếu mẫu là số âm thì khử dấu trừ của mẫu số
        if (p.mau < 0)
        {
            p.tu *= -1;
            p.mau *= -1;
        }
    }
    else
        p.mau = 1;
}

void Tong (PHANSO x, PHANSO y)
{
    PHANSO C;
    C = Cong2PhanSo(A, B);
    XuatPS(A);
    printf(" + ");
    XuatPS(B);
    printf(" = ");
    XuatPS(C);
}

```

#### 8.4.2. 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);
}

```

### 8.4.3. 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);
}
```

### 8.4.4. 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);
}
```

#### 8.4.5. 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 DONTTHUC;
```

- Định nghĩa hàm nhập

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

- Định nghĩa hàm xuất

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

#### 8.4.6. 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);
}
```

#### 8.4.7. 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;
```

#### 8.4.8. 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);
}
```

## 8.5. Mảng cấu trúc

### 8.5.1. Khai báo

Tương tự như khi khai báo mảng với kiểu dữ liệu chuẩn của C. Chỉ khác là thay kiểu dữ liệu chuẩn bằng kiểu dữ liệu có cấu trúc.

Ví dụ 8.9     `SINHVIEN A[SIZE];`  
                   `DIEM P[200];`

### 8.5.2. 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.

Ví dụ 8.10     `SINHVIEN A[SIZE];`  
                   `...`  
                   `A[i].DiemTB=7.25;`  
                   `A[j].DiemTB=A[i].DiemTB;`  
          Hay `DIEM P[200];`  
                   `...`  
                   `P[j].x=1357;`  
                   `P[j].y=P[i].x * 2;`

### 8.5.3. Viết chương trình sử dụng mảng cấu trúc

#### 8.5.3.1. 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.

Ví dụ 8.11     `void Nhap1SinhVien (SINHVIEN &sv) { ... }`  
                   `void Xuat1SinhVien(SINHVIEN sv) { ... }`

- 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 lệnh lặp.

Ví dụ 8.12     `void NhapMangSinhVien (SINHVIEN A[], int n)`  
                   `{`  
                              `for (int i=0;i<n;i++)`  
                                      `Nhap1SinhVien(A[i]);`  
                   `}`

```
hay void XuatMangSinhVien (SINHVIEN A[], int n)
{
    for (int i=0;i<n;i++)
        Xuat1SinhVien(A[i]);
}
```

### 8.5.3.2. Minh họa

- **Yêu cầu**: Tổ chức dữ liệu quản lý xe gắn máy với số lượng (n) dự kiến nằm trong khoảng  $0 < n \leq 100$ . Các thông tin liên quan đến mỗi xe gồm:

- Số xe (kiểu chuỗi, tối đa 12 ký tự).
- Số Km đã đi được (kiểu số thực).
- Hãng sản xuất (kiểu số nguyên). Quy ước: 0-Honda, 1-Yamaha, 2-Vespa, 3-Suzuki.
- Động cơ xăng (kiểu bool). Quy ước: true-động cơ dùng xăng; false- động cơ dùng xăng pha nhớt.

Viết chương trình gồm các hàm chức năng để thực hiện những công việc sau:

- Hàm nhập vào 1 xe.
- Hàm nhập danh sách n xe. Hàm này sẽ gọi hàm nhập 1 xe ở câu a thực hiện.
- Hàm xuất thông tin về 1 xe.
- Hàm xuất danh sách n xe. Hàm này sẽ gọi hàm xuất 1 xe ở câu c thực hiện
- Hàm xuất thông tin các xe của hãng X, có loại động cơ là Y. Với X và Y là 2 tham số đầu vào.

- **Thực hiện**:

- Khai báo đầu chương trình

```
#include <iostream>
#include <conio.h>
#define SIZE 100

typedef struct XE
{
    char SoXe[12];
    float SoKm;
    int HangSanXuat; //0-Honda, 1-Yamaha, 2-Vespa, 3-Suzuki
    bool DongCoXang; /*true-động cơ dùng xăng;
                     false- động cơ dùng xăng pha nhớt */
};
```

- Viết hàm thực hiện yêu cầu của câu a (Hàm nhập vào 1 xe)

```
void Nhap1Xe(XE &x)
{
    printf("\nNhap so xe: ");
    fflush(stdin);
    gets(x.SoXe);
    printf("Nhap so Km: ");
    scanf("%f", &x.SoKm);
    printf("Nhap hang san xuat (0-Honda, 1-Yamaha, 2-Vespa,
                                                3-Suzuki): ");
    scanf("%d", &x.HangSanXuat);
}
```

```
printf("Dong co xang (Y/N): ");
fflush(stdin);
char VietNam=getchar();
if ((VietNam=='Y') || (VietNam=='y'))
    x.DongCoXang=true;
else
    x.DongCoXang=false;
}
```

- Viết hàm thực hiện yêu cầu của câu b (Hàm nhập danh sách n xe)

```
int NhapN()
{
    int n;
    do
    {
        printf("Nhap n (0<n<=%d): ", SIZE);
        scanf("%d", &n);
        if ((n<=0) || (n>SIZE))
            printf("Chi nhan cac so tu 1 den %d.
                                Nhap lai.\n", SIZE);

    }while ((n<=0) || (n>SIZE));
    return n;
}

void NhapMang(XE A[],int n)
{
    for (int i=0;i<n;i++)
        Nhap1Xe(A[i]);
}
```

- Viết hàm thực hiện yêu cầu của câu c (Hàm xuất thông tin về 1 xe)

```
void Xuat1Xe(XE x)
{
    printf("\nSo xe: %s",x.SoXe);
    printf("\nSo Km da di: %f", x.SoKm);
    if (x.HangSanXuat==0)
        printf("\nHang san xuat: Honda");
    else
        if (x.HangSanXuat==1)
            printf("\nHang san xuat: Yamaha");
        else
            if (x.HangSanXuat==2)
                printf("\nHang san xuat: Vespa");
            else
                printf("\nHang san xuat: Suzuki");
    if (x.DongCoXang==true)
        printf("\nXe dong co xang");
    else
        printf("\nXe dong co xang pha nhot");
}
```

- Viết hàm thực hiện yêu cầu của câu d (Hàm xuất danh sách n xe)

```
void XuatMang(XE A[],int n)
{
    for (int i=0;i<n;i++)
        Xuat1Xe(A[i]);
}
```

- Viết hàm thực hiện yêu cầu của câu e (Hàm xuất thông tin các xe của hãng X, có loại động cơ là Y. Với X và Y là 2 tham số đầu vào)

```
void XuatXeTheoDieuKien(XE A[],int n, int hangXe, bool dongCo)
{
    printf("Cac xe cua hang ");
    if (hangXe==0)
        printf("Honda");
    else
        if (hangXe==1)
            printf("Yamaha");
        else
            if (hangXe==2)
                printf("Vespa");
            else
                printf("Suzuki");
    printf(" su dung dong co ");
    if (dongCo==true)
        printf("xang:\n");
    else
        printf("xang pha nhot:\n");
    for (int i=0;i<n;i++)
        if ((A[i].HangSanXuat==hangXe)
            && (A[i].DongCoXang==dongCo))
            Xuat1Xe(A[i]);
}
```

- Hàm main

```
int main()
{
    XE A[SIZE];
    int n, hangXe;
    bool dongCo;
    n=NhapN();
    NhapMang(A,n);
    printf("\nDanh sach xe vua nhap: \n") ;
    XuatMang(A,n);
    printf("\nNhap hang san xuất cần xem (0-Honda, 1-Yamaha,
        2-Vespa, 3-Suzuki): ");

    scanf("%d",&hangXe);
    printf("Xe sử dụng động cơ xăng (Y/N): ");
    fflush(stdin);
    char VietNam=getchar();
    if ((VietNam=='Y') || (VietNam=='y'))
        dongCo=true;
    else
        dongCo=false;
    XuatXeTheoDieuKien(A, n, hangXe, dongCo);
    return 0;
}
```

- Khai báo nguyên mẫu hàm

```
int NhapN();
void Nhap1Xe(XE &x);
void NhapMang(XE A[],int n);
```

```
void Xuat1Xe(XE x);  
void XuatMang(XE A[], int n);  
void XuatXeTheoDieuKien(XE A[], int n, int hangXe, bool dongCo);
```

## 8.6. Bài tập sinh viên tự thực hiện

### 8.6.1. Kiểu cấu trúc đơn

- (1)- Từ bài phân số trong phần 8.4.1, yêu cầu bổ sung như sau:
- Tạo một mảng các phân số: void NhapMangPS(PHANSO A[], int n). Hàm này sẽ gọi hàm nhập 1 phân số (Nhap1PS) đã có.
  - Xuất mảng phân số: void XuatMangPS(PHANSO A[], int n). Hàm này sẽ gọi hàm xuất 1 phân số đã có.
  - Tìm phân số lớn nhất, phân số nhỏ nhất. PHANSO TimPSMax(PHANSO A[], int n)
  - Sắp xếp mảng phân số tăng dần: void SortMangPS(PHANSO A[], int n)
  - Cho biết tổng, hiệu, tích, thương của 2 phân số lớn nhất và phân số nhỏ nhất đang có trong mảng.

- (2)- Tổ chức dữ liệu quản lý danh mục các bộ phim. Các thông tin liên quan đến mỗi bộ phim gồm:

- Tên phim (kiểu chuỗi, tối đa 50 ký tự).
- Doanh thu (kiểu số thực).
- Thể loại (kiểu số nguyên). Quy ước: 0-hình sự, 1-tình cảm, 2-hài.
- Năm sản xuất (kiểu số nguyên).
- Phim Việt Nam sản xuất (kiểu bool). Quy ước: true-Việt Nam; false-nước ngoài.

Viết chương trình gồm các hàm chức năng để thực hiện những công việc sau:

- Hàm nhập vào 1 bộ phim mới cùng với các thông tin liên quan đến bộ phim này.
  - Hàm nhập danh sách n phim, với n do người dùng nhập trực tiếp trong hàm.
  - Hàm xuất thông tin về 1 bộ phim.
  - Hàm xuất danh sách n phim đang có trong danh sách.
  - Hàm cho nhập một thể loại: In ra danh sách các bộ phim thuộc thể loại này.
  - Hàm tính tổng doanh thu của các phim do Việt Nam sản xuất.
  - Hàm nhận 2 tham số nguyên là *fromYear* (từ năm) và *toYear* (đến năm). Hàm thực hiện tính doanh thu trung bình của tất cả các phim năm sản xuất nằm trong khoảng từ *fromYear* đến *toYear*.
  - Hàm nhận 2 tham số nguyên là *LowerBound* (cận dưới của doanh thu) và *UpperBound* (cận trên của doanh thu). Hàm thực hiện đếm số lượng phim có doanh thu thấp hơn *LowerBound* hoặc lớn hơn *UpperBound*.
- (3)- Tổ chức dữ liệu quản lý danh mục các máy tính của một cửa hàng. Các thông tin liên quan đến mỗi máy tính gồm:
- Model (kiểu chuỗi, tối đa 10 ký tự).
  - Hãng sản xuất (kiểu chuỗi, tối đa 30 ký tự).
  - Thời gian bảo hành (kiểu số nguyên). Quy ước: **1: 3 tháng; 2: 6 tháng; 3: 12 tháng; 4: 24 tháng; 5: 36 tháng.**
  - Giá tiền (kiểu float)

- Sản xuất tại Việt Nam (kiểu bool). Quy ước: true-Việt Nam; false-nước ngoài.

Viết chương trình gồm các hàm chức năng để thực hiện những công việc sau:

- a. Hàm nhập vào 1 máy tính mới cùng với các thông tin liên quan đến máy tính này. Cần kiểm tra giá trị nhập, nếu nhập sai, chương trình phải yêu cầu nhập lại cho đến khi nhập đúng. Các thông tin cần kiểm tra gồm:
    - Thời gian bảo hành phải nằm trong khoảng từ 1 đến 5.
    - Sản xuất tại Việt Nam: cho người dùng nhập Y hoặc y là sản xuất tại Việt Nam; nhập N hoặc n là sản xuất tại nước ngoài. Các ký tự được nhập phải là 1 trong các ký tự Y, y, N, n.
  - b. Hàm nhập danh sách n máy tính, với n do người dùng nhập trực tiếp trong hàm.
  - c. Hàm xuất thông tin về 1 máy tính. Yêu cầu:
    - Thời gian bảo hành phải thể hiện rõ số tháng bảo hành.
    - Sản xuất tại Việt Nam: phải ghi rõ “*Sản xuất tại Việt Nam*” hoặc “*Sản xuất tại nước ngoài*”.
  - d. Hàm xuất các máy tính đang có trong danh sách.
  - e. Hàm cho nhập hãng sản xuất: In ra danh sách các máy tính do hãng này sản xuất.
  - f. Hàm nhận 2 tham số nguyên là *fromPrice* (từ giá) và *toPrice* (đến giá). Hàm thực hiện đếm số lượng model máy tính có giá tiền nằm trong khoảng từ *fromYear* đến *toYear*.
  - g. Hàm nhận 2 tham số nguyên là *lowerPrice* (giá thấp nhất) và *upperPrice* (giá cao nhất). Hàm thực hiện in ra danh sách các máy tính do Việt Nam sản xuất và có giá tiền thấp hơn *lowerPrice* hoặc cao hơn *upperPrice*.
- (4)- 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?
- (5)- 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.
- (6)- 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.
- (7)- 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.
- (8)- Sử dụng cấu trúc để viết chương trình tính khoảng cách giữa 2 ngày.  
Ví dụ    ngay1= 15/11/2007  
             ngay2= 11/9/2009  
in ra ngay 15/11/2007 và ngay 11/8/2009 cách nhau 26 ngày 9 tháng và 1 năm
- (9)- 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.

### 8.6.2. Cấu trúc lồng nhau

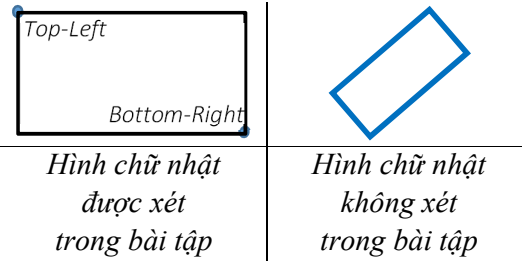
- (10)- Viết chương trình về hỗn số dưới dạng các hàm:
- a. Dựa vào cấu trúc phân số đã có ở các bài tập trước, khai báo kiểu dữ liệu để biểu diễn một hỗn số (HONSO), với hỗn số gồm 2 trường (field) là phần nguyên và phần phân số.
  - b. Viết hàm rút gọn 1 hỗn số. void RutGonPS(HONSO &hs)  
Trong đó cần thực hiện các xử lý sau:
    - Khi tử số lớn hơn mẫu số, cần chuyển phần thương của phép chia giữa tử và mẫu sang cho phần nguyên.
    - Khi tử số = 0 thì mẫu số sẽ được gán = 1.
    - Khử dấu âm khi tử và mẫu số cùng mang dấu âm.
    - Khi tử số là số dương và mẫu số là số âm thì chuyển dấu âm cho tử số (mẫu số luôn mang dấu dương).
    - Tối giản hỗn số. Ví dụ: 1(-9/2) sẽ tối giản thành -5(1/2).
  - c. Nhập 1 hỗn số. Chương trình sẽ báo lỗi và yêu cầu người dùng nhập lại cho đến khi mẫu số khác 0 là hoàn tất: void Nhap1PS(HONSO &hs)
  - d. Tạo một mảng các hỗn số: void NhapMangPS(HONSO A[], int n). Hàm này sẽ gọi hàm nhập 1 hỗn số ở câu c.
  - e. Xuất 1 hỗn số dưới dạng phân nguyên(tử số/mẫu số), ví dụ -3(1/2):  
void Xuat1PS(HONSO hs)

- f. Xuất mảng hỗn số: void XuatMangPS(HONSO A[], int n). Hàm này sẽ gọi hàm xuất 1 hỗn số ở câu e.
- g. Tính tổng, hiệu, tích, thương hai hỗn số, kết quả trả về 1 hỗn số. Ví dụ:  
HONSO Tong2PS(HONSO ps1, HONSO ps2)
- h. So sánh hai hỗn số: HONSO Tong2PS(HONSO ps1, HONSO ps2)
- i. Tìm hỗn số lớn nhất, hỗn số nhỏ nhất. Ví dụ:  
HONSO TimHonSoMax(HONSO A[], int n)
- j. Sắp xếp mảng hỗn số tăng dần: void SortMangHS(HONSO A[], int n)
- k. Cho biết tổng, hiệu, tích, thương của 2 hỗn số lớn nhất và hỗn số nhỏ nhất đang có trong mảng.
- (11)- Viết chương trình dưới dạng các hàm để quản lý mảng thông tin về sinh viên. Trong đó ngoài những thành phần có kiểu dữ liệu chuẩn của C/C++, cấu trúc sinh viên còn có 2 thành phần có kiểu struct khác là ngày sinh và điểm số.
- a.- Khai báo cấu trúc sinh viên, biết thông tin về một sinh viên gồm có:
- Mã số sinh viên (chuỗi 10 ký tự).
  - Họ Tên (là chuỗi tối đa 30 ký tự);
  - Ngày sinh (gồm 3 thành phần có cùng kiểu số nguyên là ngày tháng năm).
  - Giới tính (Nam hoặc Nữ).
  - Số lượng môn đã học (số nguyên n, có giá trị  $0 \leq n \leq 50$ ).
  - Danh sách các môn đã học (là mảng 1 chiều, trong đó mỗi phần tử của mảng gồm 2 thông tin: tên môn học (tối đa 30 ký tự), điểm số (kiểu số thực).
- b.- Viết hàm nhập một ngày.
- c.- Viết hàm xuất một ngày dưới dạng day/month/year.
- d.- Viết hàm nhập một môn học để người dùng nhập tên môn học và điểm số của môn học đó.
- e.- Viết hàm xuất một môn học (gồm 2 thông tin tên môn học và điểm số của môn học đó).
- f.- Viết hàm nhập dữ liệu cho một sinh viên (void Nhap1SV(SINHVIEN &sv)).
- g.- Viết hàm nhập danh sách sinh viên, lưu trên mảng một chiều. Hàm này sẽ gọi hàm nhập 1 sinh viên. (NhapMangSV(SINHVIEN A[], int n)).
- h.- Viết hàm xuất dữ liệu một sinh viên với thông tin vừa nhập ở trên (void Xuat1SV(SINHVIEN &sv))
- i.- Viết hàm xuất danh sách sinh viên. (XuatMangSV(SINHVIEN A[], int n))
- j.- Xuất thông tin của sinh viên có mã sinh viên là “X”.
- k.- Đếm số lượng sinh viên có năm sinh là 2000.
- l.- Sắp xếp danh sách sinh viên theo MSSV
- m.- Xóa sinh viên có mã số “SV123”.
- (12)- Có 5 đối tượng trong mặt phẳng sẽ có trong cùng một chương trình cần viết như sau:
- Điểm (pixel): bao gồm tọa độ x, tọa độ y.
  - Đoạn thẳng (line segment): được xác định bởi 2 điểm (x1, y1) là điểm bắt đầu – (x2, y2) là điểm kết thúc.
- Cho 2 điểm A(x<sub>A</sub>, y<sub>A</sub>) và B(x<sub>B</sub>, y<sub>B</sub>). Phương trình đường thẳng đi qua 2 điểm này được xác định qua công thức:



$$\frac{x - x_A}{x_B - x_A} = \frac{y - y_A}{y_B - y_A}$$

- Đường thẳng (line): là đoạn thẳng nhưng được mở rộng theo cả 2 hướng, tức là không còn điểm đầu và điểm cuối.
- Hình chữ nhật (rectangle): được xác định bởi 2 điểm Top-Left( $x_1, y_1$ ) và Bottom-Right( $x_2, y_2$ ). Trong bài tập này chỉ xét các hình chữ nhật nằm ngang, các hình chữ nhật nằm nghiêng sẽ không được xét trong bài tập này
- Hình tròn (circle): được xác định bởi một điểm  $O(x, y)$  là tâm; và  $R$  là bán kính. Phương trình dạng tổng quát của đường tròn (Hệ tọa độ Descartes) là:  $(x - a)^2 + (y - b)^2 = R^2$ .



Yêu cầu thực hiện:

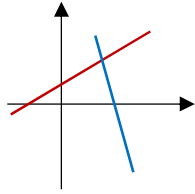
- Hãy định nghĩa cấu trúc của các kiểu dữ liệu cho 4 đối tượng trên mà sinh viên cho là hợp lý nhất.
- Cho nhập 2 phương trình đường thẳng. Tìm giao điểm của 2 đường thẳng.

🔗 Nhắc lại:

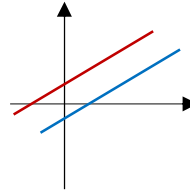
- Khi hai đường thẳng giao nhau: chúng chỉ gặp nhau tại một điểm được thể hiện bằng cặp tọa độ  $x$  và  $y$ . Vì cả hai đường thẳng đều đi qua điểm đó nên cặp tọa độ  $x, y$  phải thỏa mãn cả hai phương trình. Để tìm giao điểm này, ta cho vế phải của 2 phương trình bằng nhau để tìm  $x$ . Từ  $x$  có được, thay vào phương trình để tìm  $y$ . Ví dụ:

Phương trình đường thẳng 1:  $y = x + 3$   
 Phương trình đường thẳng 2:  $y = -2x + 15$   
 Để tìm  $x$ : cho 2 vế phải bằng nhau:  $x + 3 = -2x + 15$   
 Giải phương trình bậc 1 này để tìm  $x$ .

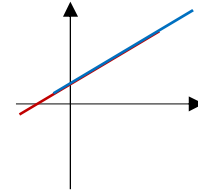
- Khi hai đường thẳng không giao nhau:
  - Trường hợp hai đường thẳng song song: khi đó, các số hạng  $x$  sẽ bị triệt tiêu và phương trình được đơn giản thành một mệnh đề sai (chẳng hạn  $0=1$ ). Đáp án lúc này là "hai đường thẳng không giao nhau" hoặc "không có nghiệm thực".
  - Trường hợp hai phương trình biểu diễn cùng một đường thẳng: khi đó chúng "giao nhau" ở mọi điểm. Các số hạng  $x$  sẽ bị triệt tiêu và phương trình được đơn giản thành một mệnh đề đúng (chẳng hạn  $2=2$ ). Đáp án của trường hợp này là "hai đường thẳng trùng nhau".



2 đường thẳng giao nhau

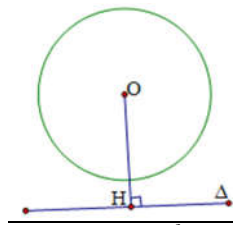


2 đường thẳng song song

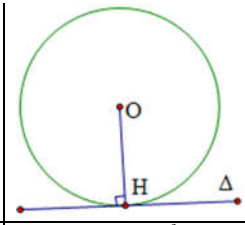


2 đường thẳng trùng nhau

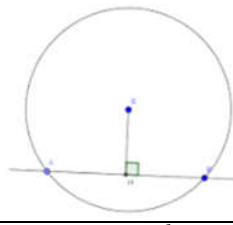
c. Cho nhập 1 phương trình đường thẳng, 1 phương trình đường tròn. Tìm giao điểm của chúng (nếu có).



SL giao điểm = 0



SL giao điểm = 1



SL giao điểm = 2

↳ Nhắc lại: Giả sử cho phương trình đường thẳng:  $y = x + 7$

Và phương trình đường tròn:  $y = x^2 + 2x + 1$

Thực hiện qua các bước

- Cho vế phải của 2 phương trình bằng nhau ( $x + 7 = x^2 + 2x + 1$ ).
- Dựa kết quả vừa có ( $x^2 + x - 6$ ), giải phương trình bậc 2 để tìm nghiệm. Nghiệm có được chính là giao điểm cần tìm.

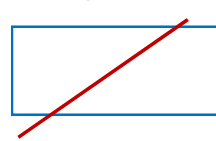
d. Cho nhập 1 phương trình đường thẳng và tọa độ 2 điểm Top-Left và Bottom-Right của 1 hình chữ nhật. Tìm giao điểm của chúng (nếu có).



SL giao điểm = 0



SL giao điểm = 1



SL giao điểm = 2



SL giao điểm = n

### 8.6.3. Con trỏ và kiểu dữ liệu có cấu trúc

(13)- Tổ chức dữ liệu quản lý danh mục các xe máy của một cửa hàng. Các thông tin liên quan đến mỗi xe gồm:

- Mã model (kiểu số nguyên).
- Model (kiểu chuỗi, tối đa 10 ký tự).
- Hãng sản xuất (kiểu chuỗi, tối đa 30 ký tự).
- Lắp ráp tại Việt Nam (kiểu bool). Quy ước: true-Việt Nam; false-nước ngoài.
- Giá bán (kiểu float). Quy ước giá bán >0
- Thời gian bảo hành (kiểu số nguyên). Quy ước: **1: 1 năm; 3: 3 năm; 5: 5 năm.**
- Số máy (kiểu chuỗi, tối đa 8 ký tự).
- Số khung (kiểu chuỗi, tối đa 12 ký tự).
- Màu sơn (kiểu chuỗi, tối đa 20 ký tự).

Viết chương trình sử dụng con trỏ để viết các hàm chức năng thực hiện những công việc sau:

- a. Khai báo cấu trúc Model gồm mã model, tên model, tên hãng sản xuất, lắp ráp tại Việt Nam, giá bán, thời gian bảo hành. Cần kiểm tra giá trị nhập cho các trường (field, thuộc tính), nếu nhập sai, chương trình phải yêu cầu nhập lại cho đến khi nhập đúng:
    - Lắp ráp tại Việt Nam: cho người dùng nhập Y hoặc y là Lắp ráp tại Việt Nam; nhập N hoặc n là Lắp ráp tại nước ngoài. Các ký tự được nhập phải là 1 trong các ký tự Y, y, N, n.
    - Thời gian bảo hành chỉ nhận 1 trong 3 giá trị 1 hoặc 3 hoặc 5.
  - b. Khai báo cấu trúc Xe gồm các thuộc tính còn lại và thuộc tính Mã model.
  - c. Hàm nhập vào 1 Model mới cùng với các thông tin liên quan đến Model này.
  - d. Hàm nhập vào 1 xe mới cùng với các thông tin liên quan đến xe này.
  - e. Hàm nhập danh sách n xe, với n do người dùng nhập trực tiếp trong hàm.
  - f. Hàm xuất thông tin về 1 xe. Yêu cầu thuộc tính Lắp ráp xuất tại Việt Nam: phải ghi rõ “*Lắp ráp tại Việt Nam*” hoặc “*Lắp ráp tại nước ngoài*”.
  - g. Hàm xuất các xe đang có trong danh sách.
  - h. Hàm cho nhập hãng sản xuất: In ra danh sách các xe do hãng này sản xuất gồm tên model, số máy, số khung, màu sơn.
  - i. Hàm nhận 1 tham số là số khung, cho biết tên hãng sản xuất và tên model của xe.
  - j. Hàm nhận 2 tham số nguyên là giá X và tên hãng Y. Hàm thực hiện in ra danh sách các xe lắp ráp tại Việt Nam sản xuất, của hãng X và có giá thấp hơn Y.
-

## 9 KIỂU DỮ LIỆU CON TRỎ (Pointer data type)

Sau khi học xong bài này, sinh viên có thể

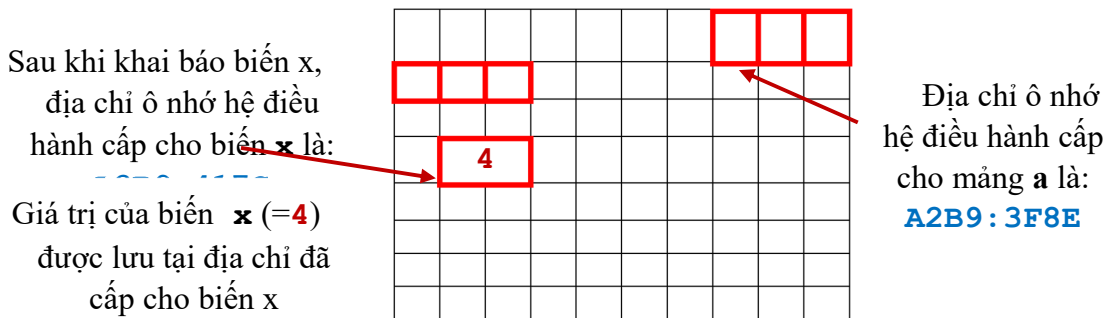
- Hiểu khái niệm về con trỏ;
- Biết cách khai báo và sử dụng biến kiểu con trỏ;
- Biết xử lý các phép toán trên mảng một chiều theo kiểu con trỏ;
- Biết xử lý các phép toán trên mảng hai chiều theo kiểu con trỏ;
- Đi sâu vào các giải thuật trên mảng 1 chiều, 2 chiều như tìm kiếm, sắp xếp, thêm phần tử,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

### 9.1. Khái niệm

#### 9.1.1. 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:  
Giá trị lưu trong biến num là: 22  
Số byte trong bộ nhớ dành cho biến này là: 2  
Địa chỉ của ô nhớ dành cho biến num là: 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.

### 9.1.2. Biến động

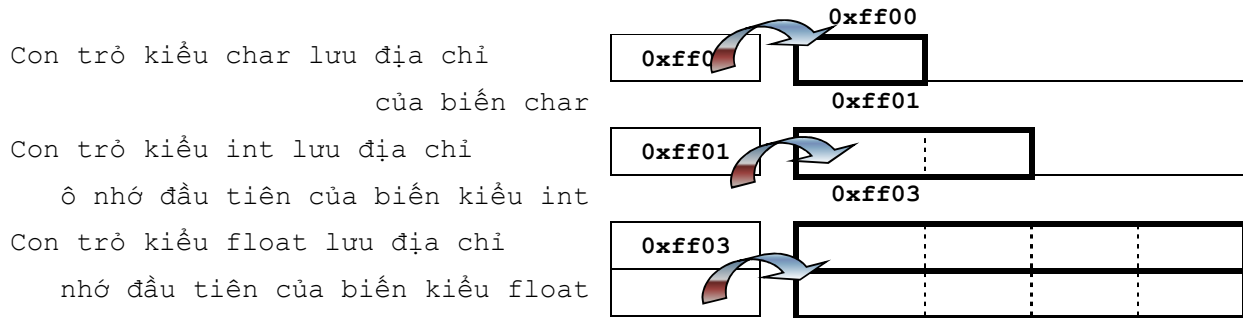
- Để tránh những hạn chế trên, ngôn ngữ C cung cấp cho ta một loại biến đặc biệt gọi là biến động với các đặc điểm sau:
  - Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
  - Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được 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.

## 9.2. Khai báo và sử dụng biến con trỏ

### 9.2.1. 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 9.1:** Khai báo 2 biến *a, b* có kiểu *int* và 2 biến *px, py* là 2 biến con trỏ  
`int a, b, *px, *py;`
- **Ví dụ 9.2:** 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:



### 9.2.2. 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 9.3

```
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.

- **Ví dụ 9.4** viết hàm hoán vị 2 biến a và b bằng 2 cách sử dụng tham chiếu và sử dụng kiểu dữ liệu con trỏ

sử dụng tham chiếu	sử dụng kiểu con trỏ
<pre>#include &lt;stdio.h&gt; #include &lt;conio.h&gt; void HoanVi(int &amp;a, int &amp;b) {     int tam;     tam = a;     a = b;     b = tam; } void main() {     int x = 3, y = 7;     printf("TRUOC khi hoan vi,                                 x= %d, y=%d", x, y);      HoanVi(x, y);     printf("\nSAU khi hoan vi,                                 x= %d, y=%d", x, y); }</pre>	<pre>#include &lt;stdio.h&gt; #include &lt;conio.h&gt; void HoanVi(int *a, int *b) {     int tam;     tam = *a;     *a = *b;     *b = tam; } void main() {     int x = 3, y = 7;     printf("TRUOC khi hoan vi,                                 x= %d, y=%d", x, y);      HoanVi(&amp;x, &amp;y);     printf("\nSAU khi hoan vi,                                 x= %d, y=%d", x, y); }</pre>

### 9.2.3. Các thao tác trên con trỏ

#### 9.2.3.1. 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 9.5**

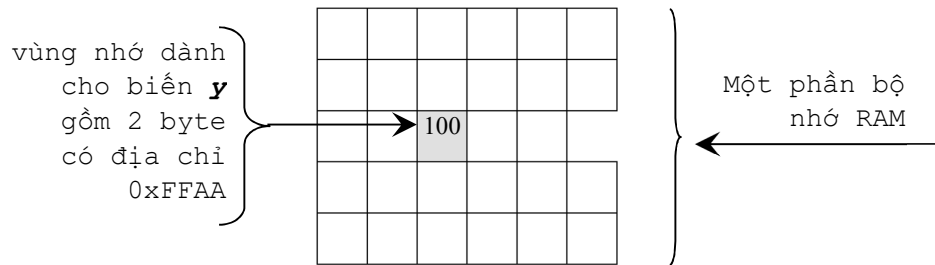
```
void main()
{
    int x=100;
    int *px;
    printf("Bien x=%d, duoc luu tru tai dia chi: %x\n", x, &x);
    px=&x;
    printf("Bien px duoc luu tru tai dia chi: %x\n", &px);
    printf("Bien px dang tham chieu den dia chi: %x\n", *px);
    printf("Gia tri dang luu tai dia chi %x: %d", px, *px);
}
```

- **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.

#### 9.2.3.2. 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 9.6:** giả sử có một biến con trỏ *y*. giá trị của biến hiện là 0xFFAA (địa chỉ của ô nhớ thứ FF AA 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  $\star y$  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  $\star$ 
  - 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:
  - (i). Truy nhập trực tiếp tới biến.
  - (ii). 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 9.7:** Đoạn chương trình sau thấy rõ sự thay đổi này:

```
#include <stdio.h>
#include <conio.h>
int main()
{ int a, b, *pa, *pb;
  a= 2 ;
  b= 3 ;
  printf("\nGia tri cua bien a=%d \nGia tri cua bien b=%d ",a,b);
  pa=&a;
  pb=&b;
  printf("\nNoi dung cua o nho con tro pa tro toi=%d",*pa);
  printf("\nNoi dung cua o nho con tro pb tro toi=%d ",*pb);
  *pa=20; /*Thay doi gia tri cua bien a (hien con tro pa dang chi den)*/
  *pb=20; /*Thay doi gia tri cua bien b (hien con tro pb dang chi den)*/
  printf("\nGia tri moi cua bien a=%d \nGia tri moi cua bien b=%d ",a,b);
  /* gia tri cua 2 bien a,b da bi thay doi*/
  return 0;
}
```

Kết quả thực hiện chương trình:



- Ví dụ 9.8

```
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 9.9

```
void main()
{
    int x=7, y=5;
    int *px, *py;
    px=&x;
    printf("y= %d\n", y); //y=5
    printf("px chưa giá trị %d\n", *px); //7
    printf("px= %x\n", px); //px=9ffbe8
    y=*px; //y sẽ chứa giá trị đang
    // lưu giữ tại địa chỉ mà biến con trỏ px chỉ đến (=7)*/
    printf("Địa chỉ y= %d\n", y); //aafb8
    printf("y= %d\n", y); //7
}
```

### 9.2.3.3. 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.

## 9.3. 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

### 9.3.1. 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.

#### 9.3.1.1. 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 => bo lỗi khi biên dịch
```

#### 9.3.1.2. Xuất giá trị

```
int *px, x=5;
px=&x;
printf("%5d", *px);
```

### 9.3.1.3. 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;          // ⇔ y= 4+1=5
d=sqrt((double) *px);

/* ⇔ d =  $\sqrt{x} = \sqrt{4} = 2$ . Do hàm sqrt chỉ nhận tham số kiểu double nên
    phải thực hiện ép kiểu */
```

### 9.3.1.4. 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.

### 9.3.1.5. 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 */
```

### 9.3.1.6. Ép kiểu

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

## 9.3.2. 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 9.10**

```
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 9.11:** (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 9.12:** (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]).

### 9.3.3. 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 9.13**

```
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

### 9.3.4. Phép so sánh

Dùng cho phép so sánh 2 con trỏ cùng kiểu

- (i).  $p1 < p2$  nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới
- (ii).  $p1 = p2$  nếu địa chỉ p1 trỏ tới bằng địa chỉ p2 trỏ tới
- (iii).  $p1 > p2$  nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

### 9.3.5. Một số minh họa

#### 9.3.5.1. Minh họa 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("Địa chỉ biến addr đang trỏ tới: %u\n",addr);
printf("Giá trị tại vùng nhớ addr đang trỏ tới: %d\n", *addr);
return 0;
}
```

### 9.3.5.2. Minh họa 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("Tổng các phần tử có trong mảng= %d",s);
    return 0;
}
```

### 9.3.5.3. Minh họa 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;
}
```

## 9.4. Sử dụng con trỏ để cấp phát và thu hồi bộ nhớ động

### 9.4.1. Giới thiệu

- Trong bộ nhớ ảo của máy được chia làm 4 phần.
  - *Code segment*: là nơi chứa mã máy dạng nhị phân
  - *Data segment*: là nơi chứa các biến tĩnh (static) và biến toàn cục
  - *Heap*: là vùng nhớ không do CPU quản lý, lập trình viên phải tự quản lý vùng nhớ này. Hay nói khác, đây là vùng nhớ dành cho con trỏ.
  - *Stack*: Đây là vùng nhớ do CPU quản lý, lập trình viên không được phép "can thiệp" vào vùng nhớ này. Vùng nhớ này chứa các biến cục bộ. Khi bạn tạo biến x bằng cách `int x;` thì nó sẽ được tạo trong vùng nhớ Stack. Mà khi được tạo trong Stack thì bạn không được phép "can thiệp" như giải phóng (delete) , ....
- Để cấp phát bộ nhớ động, ta sử dụng các hàm/toán tử trong thư viện `stdlib.h` hoặc `alloc.h`.
  - (i). Hàm `malloc`
  - (ii). Hàm `calloc`
  - (iii). Hàm `realloc`
  - (iv). Toán tử `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 hay toán tử delete []

### 9.4.2. Các hàm/toán tử cấp phát vùng nhớ

#### 9.4.2.1. 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 9.14** 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;
}
```

#### 9.4.2.2. 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 9.15** tương tự ví dụ khi dùng hàm malloc, chỉ khác là ở đây 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));
    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;
}

```

#### - 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.

#### 9.4.2.3. 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 9.16**

```

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*/

```

#### 9.4.2.4. Toán tử new

- **Công dụng:** cấp phát bộ nhớ khi chương trình đang thực thi (run time) bên trong vùng nhớ Heap.
- **Cú pháp:** `Kiểu_dữ_liệu* new Kiểu_dữ_liệu`
  - Trong đó `Kiểu_dữ_liệu` có thể là bất kỳ kiểu dữ liệu có sẵn nào (char, int, float, mảng hoặc các kiểu dữ liệu tự định nghĩa như lớp hoặc cấu trúc, ...)
  - Khi cấp phát không thành công, kết quả trả về NULL.
- **Ví dụ 9.17**

```

//cấp phát bộ nhớ cho con trỏ p thuộc kiểu int
int *p = new int;
/*cấp phát bộ nhớ cho mảng một chiều arr gồm 5 phần tử thuộc kiểu double*/
double *arr = new double[5];
/*cấp phát bộ nhớ cho mảng hai chiều Arr kích thước 3x4 thuộc kiểu float*/
float *Arr = new float[5];
// cấp phát bộ nhớ cho kiểu dữ liệu cấu trúc

```

```
typedef struct SINHVIEN
{
    char HoTen[30];
    int Tuoi;
    float Diem;
    SINHVIEN * pNext;
}
...
SINHVIEN *sv= new SINHVIEN;
...
```

### 9.4.3. Hàm và 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:
  - Sử dụng hàm free: khi sử dụng 1 trong các hàm malloc, calloc, realloc để cấp phát bộ nhớ cho 1 biến con trỏ.
  - Sử dụng toán tử delete: khi sử dụng toán tử new để xin cấp phát bộ nhớ.
  - Sử dụng toán tử delete []: khi thu hồi bộ nhớ của mảng các con trỏ.

### 9.4.4. Toán tử sizeof

- Toán tử **sizeof** cho biết kích thước (tính theo byte) của một kiểu dữ liệu hay một đối tượng dữ liệu.
- Kiểu dữ liệu có thể là kiểu chuẩn (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 9.18

```
typedef float KieuThuc;
struct DiemThi
{
    char    masv[8];
    char    mamh[5];
    int     lanthi; // 2 bytes
    float    diem;  // 4 bytes
} x;
sizeof (int)           //cho trị 2
sizeof (KieuThuc)      //cho trị 4
sizeof (x)             //cho trị 19
```

## 9.5. Con trỏ với mảng một chiều và chuỗi ký tự

### 9.5.1. Truy cập các phần tử mảng theo dạng con trỏ

- 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.

- Cho mảng A gồm n phần tử và con trỏ p đã được gán = &A:

```
int *p, A[SIZE], n;
...
p = &A[0];
```

Khi đó, hình ảnh mảng số nguyên A gồm n phần tử lưu trong bộ nhớ do con trỏ **p** quản lý sẽ có dạng (giả sử địa chỉ của **A[0]** là **b9f2**):

	<b>p</b>	<b>b9f2</b>									
Chỉ số	0	1	2	3	4	5	6	...	n-2	n-1	
Sử dụng mảng	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	...	A[n-2]	A[n-1]	
Sử dụng con trỏ	p	p+1	p+2	p+3	p+4	p+5	p+6	...	p+n-2	p+n-1	

- Ta có các quy tắc truy xuất các phần tử trong mảng như sau:

Ý nghĩa	Truy cập bình thường	Truy cập dạng con trỏ
Địa chỉ phần tử đầu tiên	&<Tên mảng>[0]	<Tên mảng>
<i>Ví dụ</i>	&A[0]	A
Địa chỉ phần tử có chỉ số là <b>i</b>	&<Tên mảng> [<i>]	<Tên mảng> + <i>
<i>Ví dụ</i>	&A[i]	A + i
Giá trị phần tử có chỉ số là <b>i</b>	<Tên mảng> [<i>]	*(<Tên mảng> + <i>)
<i>Ví dụ</i>	A[i]	*(A + i)

- **Ví dụ 9.19:** thực hiện truy xuất các phần tử mảng 1 chiều theo nhiều cách

```
#include <stdio.h>
const int SIZE = 5;
int main()
{
    int i, *p, A[SIZE];
    //Gán lần lượt các số chính phương vào mảng
    for (i = 0; i < SIZE; i++)
        *(A + i) = (i+1) * (i+1);
    printf("\t\tDUYET MANG THEO NHIEU CACH KHAC NHAU");
    printf("\n(a).-Dua tren con tro va ten mang:\n");
    for (i = 0; i < SIZE; i++)
        printf("%5d", *(A + i));
    printf("\n(b).-Dua tren dia chi co dinh ma bien con tro p
                                                dang giu:\n");
    p = &A[0];
    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\n");
    for (i = 0; i < SIZE; i++)
        printf("%5d", *p++);
    return 0;
}
```



**9.5.3. Minh họa thao tác trên mảng một chiều bằng kiểu dữ liệu con trỏ****- Yêu cầu**

- (i). Khai báo hằng số SIZE cho biết số lượng phần tử tối đa sẽ có trong mảng. Giả sử cho SIZE = 100.
- (ii). Viết hàm nhận 2 tham số là Min và Max. Hàm cho nhập số nguyên n ( $\text{Min} < n \leq \text{Max}$ ). Nếu nhập sai, chương trình báo lỗi và yêu cầu nhập lại; ngược lại, khi nhập đúng, chương trình trả về số nguyên n vừa nhập.
- (iii). Cấp phát bộ nhớ cho con trỏ kiểu số nguyên (\*A), gồm n phần tử.
- (iv). Viết hàm nhập mảng một chiều kiểu số nguyên (\*A) gồm n phần tử (với \*A và n là tham số của hàm).
- (v). Viết hàm xuất mảng số nguyên A gồm n phần tử vừa nhập ở trên (với \*A và n là tham số của hàm)
- (vi). Viết hàm sắp xếp mảng số nguyên A gồm n phần tử tăng dần theo giá trị (với \*A và n là tham số của hàm).
- (vii). Viết hàm main kết nối các hàm trên thực hiện.

**- Thực hiện**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
// khai báo nguyên mẫu hàm
const int SIZE = 100;
int NhapSo(int Min, int Max);
bool CapPhat(int *p, int n);
void NhapMang(int *a, int n);
void XuatMang(int *a, int n);
int Tong(int *a, int n);
void HoanVi(int *x, int *y) ;
void SortTang(int *a, int n);

int main()
{
    int n, *A;
    n=NhapSo(0,SIZE);
    // cấp phát vùng nhớ cho mảng A
    if (CapPhat(a,n))
    {
        NhapMang(A, n);
        printf("Mang vua nhap: ");
        XuatMang(A, n);
        printf("\nTong cac phan tu trong mang: %d", Tong(A, n));
        SortTang(A,n);
        printf("\nMang sau sap tang: ");
        XuatMang(A, n);
        // giải phóng vùng nhớ cho mảng A
        free (A);
        /* có thể thay thế bằng 1 trong 2 cách sau
        Cách 1: delete A; A=NULL;
        Cách 2: delete[] A; A=NULL;*/
    }
}
```

```
        else
            printf("Khong cap duoc vung nho cho mang");return 0;
        return 0;
    }
    bool CapPhat(int *&p, int n)
    {
        p=(int*)calloc (n,sizeof(int));
        if (p==NULL)
        {
            printf("Khong cap duoc vung nho cho mang");
            return false;
        }
        return true;
    }
    int NhapSo(int Min, int Max)
    {
        int n;
        do
        {
            printf("Nhap n (%d-%d): ", Min,Max);
            scanf("%d", &n);
            if (n <= Min || n >Max)
                printf("Nhap sai, nhap lai.");
        } while (n <= Min || n >Max);
        return n;
    }
    void NhapMang(int *a, int n)
    {
        for (int i = 0; i < n; i++)
        {
            printf("a[%d]= ", i);
            scanf("%d", (a + i));
        }
    }
    void XuatMang(int *a, int n)
    {
        int i;
        for (int i = 0; i < n; i++)
            printf("%5d", *(a + i));
    }
    int Tong(int *a, int n)
    {
        int s=0;
        for (int i = 0; i<n; i++)
            s = s + *(a + i) ;
        return s;
    }
    void HoanVi(int *x, int *y)
    {
        int tam = *x;
        *x = *y;
        *y = tam;
    }
```

```
void SortTang(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));
}
```

#### 9.5.4. Minh họa thao tác trên chuỗi ký tự bằng kiểu dữ liệu con trỏ

- **Nhắc lại**

- Trong ngôn ngữ C, chuỗi được thể hiện bằng mảng các ký tự (char).
- Chuỗi được kết thúc bằng ký tự '\0' (gọi là ký tự NULL trong bảng mã ASCII).
- 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.

- **Yêu cầu**: Cho nhập chuỗi ký tự. Thực hiện đổi các ký tự thường thành ký tự in hoa.

- **Thực hiện**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define SIZE 100
void DoiThanhChuHoa(char *str);
int main()
{
    char *str=new char[SIZE],*p;
    printf("Nhap chuoi: ");
    gets(str);
    p=str;
    DoiThanhChuHoa(p);
    printf("Chuoi sau khi chuyen thanh chu hoa: %s",p);
    delete(str);
    return 0;
}

void DoiThanhChuHoa(char *str)
{
    int i=0;
    while (*(str + i)!='\0')
    {
        if ( (*(str + i)>='a') && (*(str + i)<='z') )
            *(str + i)-=32;
        i++;
    }
}
```

## 9.7. Con trỏ với mảng hai chiều

### 9.7.1. Thao tác trên mảng 2 chiều

- Giả sử, cho mảng 2 chiều với  $m=3$  và  $n=4$ , ta có số thứ tự các phần tử sẽ có dạng như sau:

		<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>
<b>*a</b>	<b>0</b>	0	1	2	3
	<b>1</b>	4	5	6	7
	<b>2</b>	8	9	10	11

#### 9.7.1.1. Cách 1: nhìn mảng 2 chiều thông qua mảng 1 chiều các con trỏ mà mỗi con trỏ này sẽ quản lý 1 mảng 1 chiều

- Mỗi dòng của mảng 2 chiều do 1 con trỏ quản lý, như :
  - Con trỏ  $*a$  (hay  $*(a+0)$ ) quản lý dòng 0.
  - Con trỏ  $*(a+1)$  quản lý dòng 1.
  - Con trỏ  $*(a+2)$  quản lý dòng 2.
  - ...
  - Con trỏ  $*(a+m-1)$  quản lý dòng  $m-1$ .
- Các con trỏ này lại hợp với nhau thành 1 mảng các con trỏ.

	<b>**a</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>...</b>	<b>n-1</b>
<b>0</b>	$*(a)$	$a[0][0]$	$a[0][1]$	$a[0][2]$	...	$a[0][n-1]$
<b>1</b>	$*(a+1)$	$a[1][0]$	$a[1][1]$	$a[1][2]$	...	$a[1][n-1]$
<b>2</b>	$*(a+2)$	$a[2][0]$	$a[2][1]$	$a[2][2]$	...	$a[2][n-1]$
<b>...</b>	...	...	...	...	...	...
<b>m-1</b>	$*(a+m-1)$	$a[m-1][0]$	$a[m-1][1]$	$a[m-1][2]$	...	$a[m-1][n-1]$

- Quy tắc truy xuất phần tử  $A[\text{row}][\text{col}]$  của mảng hai chiều A thông qua con trỏ như sau:

Ý nghĩa	Truy cập bình thường	Truy cập dạng con trỏ
Địa chỉ phần tử tại dòng <b>row</b> cột <b>col</b>	$\&\text{Tên mảng}[\text{row}][\text{col}]$	$*(\text{Tên mảng} + \text{row}) + \text{col}$
<i>Ví dụ</i>	$\&A[\text{row}][\text{col}]$	$*(a + \text{row}) + \text{col}$
Giá trị phần tử tại dòng <b>row</b> cột <b>col</b>	$\text{Tên mảng}[\text{row}][\text{col}]$	$*(\text{Tên mảng} + \text{row}) + \text{col}$
<i>Ví dụ</i>	$A[\text{row}][\text{col}]$	$*(a + \text{row}) + \text{col}$

#### 9.7.1.2. Cách 2: nhìn mảng 2 chiều thông qua mảng 1 chiều

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 mảng hai chiều 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	$*a$	$*(a+1)$	$*(a+2)$	$*(a+3)$	$*(a+4)$	$*(a+5)$	$*(a+6)$	$*(a+7)$	$*(a+8)$	$*(a+9)$	$*(a+10)$	$*(a+11)$
	Dòng 0				Dòng 1				Dòng 2			

- Sau khi khai báo và cấp phát vùng nhớ cho mảng 1 chiều a, để truy cập đến phần tử a[i][j] của mảng a, ta sử dụng công thức như đã biết trong phần 9.5.1. Cụ thể là:

Ý nghĩa	Truy cập dạng con trỏ
Địa chỉ phần tử đầu tiên	<Tên mảng>
Ví dụ	A
Địa chỉ phần tử có chỉ số là i	<Tên mảng> + <i>
Ví dụ	A + i
Giá trị phần tử có chỉ số là i	*(<Tên mảng> + <i>)
Ví dụ	*(A + i)

- **Ví dụ 9.20:** nhập và xuất mảng 2 chiều với số dòng là 3, số cột là 4 và khai báo biến a là con trỏ để quản lý mảng 1 chiều.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int m=3, n=4;
    int *a;
    int sl = m*n;
    a = (int*)calloc (sl,sizeof(int));
    if (a == NULL)
    {
        printf("Khong cap duoc vung nho cho mang");
        return false;
    }
    else
    {
        for (int i=0 ; i<sl ; i++)
        {
            printf("nhap a[%d][%d]:", i/n , i%n);
            scanf ("%5d", a+i);
        }
        printf("Ma tran ban dau:");
        for (int i=0 ; i<sl ; i++)
        {
            if (i%n==0)
                printf("\n");
            printf ("%4d", *(a+i));
        }
        delete a;
    }
    return 0;
}
```

## 9.7.2. Minh họa thao tác trên mảng hai chiều bằng kiểu dữ liệu con trỏ

### 9.7.2.1. Thao tác trên mảng 2 chiều (cách 1)

- **Yêu cầu**
  - Viết chương trình nhập vào một mảng hai chiều số nguyên gồm m dòng, n cột có m x n phần tử (m, n<=100).

- Xuất ra mảng hai chiều số nguyên vừa nhập.
- Tính tổng các phần tử có trong mảng hai chiều.
- Sắp xếp mảng 2 chiều tăng dần từ trái sang phải và từ trên xuống dưới bằng cách:
  - Copy từ mảng 2 chiều sang mảng 1 chiều.
  - Sắp xếp tăng dần trên mảng 1 chiều.
  - Copy từ mảng 1 chiều về lại mảng 2 chiều.

- **Thực hiện**

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define ROWS 100
#define COLS 200
int NhapCanh(char str[], int Min, int Max);
void NhapMaTran(int **a, int m, int n);
void XuatMaTran(int **a, int m, int n);
long tinhTong(int **a, int m, int n);
void HoanVi(int *x, int *y);
void Sort1Array(int *B, int sl);
void sort2Array(int **a, int m, int n);
int main()
{
    int **a, m, n, i;
    m=NhapCanh("dong", 0, ROWS);
    n=NhapCanh("cot", 0, COLS);

    //Cap phat vung nho cho ma tran
    a = new int*[m];
    for (int i = 0; i < m; i++)
        a[i] = new int[n];
    NhapMaTran(a, m, n);
    XuatMaTran(a, m, n);
    printf("Tong cac phan tu trong m tran: %ld", tinhTong(a,m,n));
    sort2Array(a, m, n);
    printf("\nMang hai chieu sau sap xep tang dan tu trai sang
                                                phai va tu tren xuong duoi:\n");
    XuatMaTran(a, m, n);
    //giai phong bo nho cho tung dong cua ma tran
    for (i = 0; i < m; i++)
        delete a[i];
    // giai phong con tro quan ly cac dong
    delete [] a;
    return 0;
}
int NhapCanh(char str[], int Min, int Max)
{
    int n;
    do
    {
        printf("Nhap %s (%d-%d): ", str, Min, Max);
        scanf("%d", &n);
        if ((n<Min) || (n>Max))
```

```
        printf("Chi nhan so tu %d-%d. Nhap lai.\n", Min,Max);
    } while ((n<Min) || (n>Max));
    return n;
}

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

void XuatMaTran(int **a, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
            printf("%5d", (*(a + i) + j));
        printf("\n");
    }
}

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

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

void sort2Array(int **a, int m, int n)
{
    int *B = new int[m*n];
    int i, j, sl = m*n ;
    //B1: copy tu mang 2 chieu sang mang 1 chieu
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            *(B + (i*n + j)) = (*(a + i) + j);
    //B2: sort mang b
    Sort1Array(B, sl);
    //B3: copy tu mang 1 chieu sang mang 2 chieu
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            (*(a + i) + j) = *(B + (i*n + j));
}
```

```
void Sort1Array(int *B, int sl)
{
    for (int i = 0; i < sl-1; i++)
        for (int j = i+1; j < sl; j++)
            if (* (B+i) > * (B+j) )
                HoanVi( B+i, B+j );
}
```

#### 9.7.2.2. Sử dụng kiểu dữ liệu con trỏ trên mảng 1 chiều để thao tác trên mảng 2 chiều (cách 2)

- **Yêu cầu**

- Viết chương trình nhập vào một mảng hai chiều số nguyên gồm m dòng, n cột có m x n phần tử (m, n ≤ 100).
- Xuất ra mảng hai chiều số nguyên vừa nhập.
- Sắp xếp mảng 2 chiều tăng dần từ trái sang phải và từ trên xuống dưới..

- **Thực hiện**

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define ROWS 100
#define COLS 200
bool CapPhat(int *&p, int n);
int NhapCanh(char str[], int Min, int Max);
void KhoiTao (int *a , int m ,int n);
void NhapMang (int *a , int m , int n);
void XuatMang (int *a, int m , int n);
void HoanVi(int *x, int *y);
void SortArray(int *B, int sl);
int main()
{
    int i,j,tam, m,n;
    int *a;
    m=NhapCanh("dong", 0, ROWS);
    n=NhapCanh("cot", 0, COLS);
    if (CapPhat(a, n))
    {
        a = (int *) calloc (m*n, sizeof (int));
        NhapMang(a,m,n);
        printf("Ma tran ban dau:\n");
        XuatMang(a,m,n);
        SortArray(a, m*n);
        printf("\nMa tran sau sort tang dan:\n");
        XuatMang(a,m,n);
        delete a;
    }
    return 0;
}
```



```
int NhapCanh(char str[], int Min, int Max)
{
    int n;
    do
    {
        printf("Nhap %s (%d-%d): ", str, Min, Max);
        scanf("%d", &n);
        if ((n<Min) || (n>Max))
            printf("Chi nhan so tu %d-%d. Nhap lai.\n", Min,Max);
    } while ((n<Min) || (n>Max));
    return n;
}

bool CapPhat(int *&p, int n)
{
    p=(int*)calloc (n,sizeof(int));
    if (p==NULL)
    {
        printf("Khong cap duoc vung nho cho mang");
        return false;
    }
    return true;
}

void NhapMang (int *a , int m , int n)
{
    int sl=m*n;
    for (int i=0 ; i<sl ; i++)
    {
        printf("nhap a[%d][%d]:", i/n , i%n);
        scanf ("%5d", a+i);
    }
}

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

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

void SortArray(int *B, int sl)
{
    for (int i = 0; i < sl-1; i++)
        for (int j = i+1; j < sl; j++)
            if (*(B+i)>*(B+j) )
                HoanVi(B+i, B+j);
}
```

## 9.8. Con trỏ với kiểu dữ liệu có cấu trúc (struct)

### 9.8.1. Truy cập đến các thành phần của struct

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

#### 9.8.1.1. Cách 1: cách thông thường

- **Cú pháp** <tên con trỏ>-><tên thành phần>
- **Ví dụ 9.21** `printf("%f ", p->diem);`

#### 9.8.1.2. Cách 2: dùng con trỏ

- **Cú pháp** (\*<tên con trỏ>).<tên thành phần>
- **Ví dụ 9.22** `printf("%f ", (*p).diem);`

### 9.8.2. Minh họa thao tác trên struct bằng kiểu dữ liệu con trỏ

- **Yêu cầu:** Cần quản lý kết quả học tập của sinh viên. Thông tin cần lưu trữ gồm: mã số SV, mã môn học, lần thi (kiểu số nguyên), điểm (kiểu số thực). Viết chương trình để quản lý danh sách các kết quả học tập này theo các yêu cầu:
  - Khai báo cấu trúc KETQUA gồm các thông tin cần lưu trữ.
  - Viết hàm cho nhập số nguyên n sao cho n>0. Nếu nhập sai, chương trình báo lỗi và yêu cầu nhập lại cho đến khi nhập đúng. Hàm trả về số nguyên n vừa nhập.
  - Trong hàm main(), khai báo mảng A có kiểu là cấu trúc KETQUA và thực hiện cấp phát vùng nhớ cho mảng A gồm n phần tử.
  - Viết hàm nhập 1 kết quả với tham biến là 1 biến thuộc kiểu cấu trúc KETQUA.
  - Viết hàm nhập danh sách n các cấu trúc KETQUA. Với n là tham số kiểu số nguyên. Hàm này sẽ gọi hàm nhập 1 kết quả.
  - Viết hàm xuất 1 kết quả với tham số là 1 biến thuộc kiểu cấu trúc KETQUA.
  - Viết hàm xuất mảng A thuộc kiểu cấu trúc KETQUA, gồm n phần tử. Hàm này sẽ gọi hàm xuất 1 kết quả.
  - Bổ sung các lệnh trong hàm main để gọi các hàm trên thực hiện.

- **Thực hiện:**

```
#include "stdio.h"
#include "conio.h"
typedef struct KETQUA
{
    char MaSV[8];
    char MaMH[5];
    int LanThi;
    float Diem;
};
int NhapN();
void Nhap1KetQua(KETQUA *px);
void NhapMang(KETQUA *A, int n);
void Xuat1KetQua(KETQUA *px);
void XuatMang(KETQUA *A, int n);
```

```
int main()
{
    KETQUA *A;
    int n=NhapN();
    A=new KETQUA[n];
    printf("\nNhap danh sach ket qua:\n");
    NhapMang(A,n);
    printf("\nXuat danh sach ket qua:\n");
    XuatMang(A,n);
    return 0;
}

int NhapN()
{
    int n;
    do
    {   printf("Nhap n (>0): ");
        scanf("%d", &n);
        if (n<=0)
            printf("Chi nhan so >0. Nhap lai.\n");
    } while (n<=0);
    return n;
}

void NhapMang(KETQUA *A,int n)
{
    for(int i=0;i<n;i++)
    {
        printf("Nhap ket qua cua SV thu %d:\n",i+1);
        Nhap1KetQua(A+i);
    }
}

void Nhap1KetQua(KETQUA *px)
{
    float tam;
    printf("\nMa sinh vien :");
    fflush(stdin);
    gets( (*px).MaSV);           //hoặc dùng (px->MaSV)
    printf("Ma mon hoc:");
    gets( (*px).MaMH);           //hoặc dùng (px->MaMH)
    printf("Lan thi:");
    scanf("%d", &(*px).LanThi); //hoặc dùng (&px->LanThi)
    printf("Diem thi:");
    scanf("%f", &(*px).Diem);   //hoặc dùng (&px->Diem)
}

void XuatMang(KETQUA *A,int n)
{
    for(int i=0;i<n;i++)
    {
        printf("\nKet qua cua SV thu %d:",i+1);
        Xuat1KetQua(A+i);
    }
}
```

```
void Xuat1KetQua(KETQUA *px)
{
    printf("\nMa sinh vien :%s", (*px).MaSV); // hoặc dùng px->masv
    printf("\nMa mon hoc:%s", (*px).MaMH);    // hoặc dùng px->mamh
    printf("\nLan thi:%d", (*px).LanThi);     // hoặc dùng px->lanthi
    printf("\nDiem thi:%.2f", (*px).Diem);    // hoặc dùng px->diem
}
```

### 9.8.3. Minh họa tham số của hàm là con trỏ của kiểu struct

- **Yêu cầu:** Cho biết 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.

- **Thực hiện**

```
#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 *px);
int main()
{
    DIEMTHI x, *px;
    px=new DIEMTHI;
    printf("\nNhap diem thi");
    Nhap(px);
    printf("\nXuat diem thi");
    Xuat(px);
    return 0;
}
void Xuat(DIEMTHI *px)
{
    printf("\nMa sinh vien :%s", (*px).MaSV); // hoặc dùng px->masv
    printf("\nMa mon hoc:%s", (*px).MaMH);    // hoặc dùng px->mamh
    printf("\nLan thi:%d", (*px).LanThi);     // hoặc dùng px->lanthi
    printf("\nDiem thi:%.2f", (*px).Diem);    // hoặc dùng px->diem
}
void Nhap(DIEMTHI *px)
{
    float tam;
    printf("\nMa sinh vien :");
    gets((*px).MaSV); //hoặc dùng (px->MaSV)
    printf("Ma mon hoc:");
    gets((*px).MaMH); //hoặc dùng (px->MaMH)
    printf("Lan thi:");
    scanf("%d", &(*px).LanThi); //hoặc dùng (&px->LanThi)
    printf("Diem thi:");
    scanf("%f", &(*px).Diem); //hoặc dùng (&px->Diem)
}
```

## 9.9. Bài thực hành có hướng dẫn

### 9.9.1. Mảng 1 chiều: 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

#### Hướng dẫn

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

```
void NhapSoPT (int &n)
{
    do
    {
        printf (" nhập số phần tử của mảng : ");
        scanf(" %d ", &n);
        if (n<=0 || n>100)
            printf (" nhập sai, nhập lại ");
    } while(n<=0 || n>100);
}
```

- Viết hàm xin cấp phát bộ nhớ cho con trỏ p

```
int CapPhatVungNho (int *p, int n)
{
    p = (int*) calloc (n, sizeof (int));
    if (p==NULL)
    {
```

- ```
        printf("khong du bo nho de cap phat");
        return 0;
    }
    return 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()**  
{  
 int \*a, n;  
 NhapSoPT (n);  
 if ((CapPhatVungNho(a,n)==1)  
 {  
 NhapMang(a, n);  
 XuatMang(a, n);  
 free(a);  
 }  
 getch();  
}
  - Viết hàm tính tổng các phần tử có trong mảng.
  - Gọi hàm tính tổng trong hàm main ().
  - Thực hiện tương tự cho các hàm khác trong yêu cầu.

**9.9.2. Mảng 2 chiều: 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

### 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 capphat (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);  
}
```



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

### 9.10.1. Trắc nghiệm

- (1)- Cho các khai báo sau `int a=3, b = 2, *pint, *pfloat;` Chọn phát biểu đúng?  
 a. `pint = *pfloat;`      b. `pint=&a;`      c. `pfloat=&pint;`      d. `pfloat = a;`
- (2)- Cho biết kết quả xuất ra màn hình khi thực hiện đoạn chương trình sau
- ```
void main()
{
    int a, b, c, *px;
    a=10; b=20; c=30;
    px=&a;
    cout<<*px<<endl; px=px-1;
    cout<<*px<<endl; px=px-1;
    cout<<*px<<endl;
}
```
- a. 10 20 30      b. 10 30 20      c. 20 30 60      d. a,b,c đều sai.
- (3)- Cho biết kết quả xuất ra màn hình khi thực hiện đoạn chương trình sau
- ```
void main()
{
    int *px, A[3] = {10,20,30};
    px=&A[0];
    cout<<*px<<endl; px=px-1;
    cout<<*px<<endl; px=px-1;
    cout<<*px<<endl;
}
```
- a. 10 20 30      b. 30 20 10      c. In địa chỉ các ô nhớ trong mảng      d. Báo lỗi.
- (4)- Cho biết ý nghĩa của đoạn chương trình viết trong C sau
- ```
int * pInt, a = 100;
pInt = new int; pInt = &a;
```
- a. Cho con trỏ `pInt` trỏ tới địa chỉ của biến `a`  
 b. Cấp phát bộ nhớ cho con trỏ `pInt` `sizeof(int)` bytes  
 c. Gán giá trị tại địa chỉ con trỏ `pInt` là `a`  
 d. Phương án b) không chính xác
- (5)- Cho khai báo: `int *pAIn[10];` Cho biết ý nghĩa của lệnh viết trong C sau:
- ```
for (int j = 0; j < 10; j++) pAIn[j] = new int;
```
- a. Cấp phát bộ nhớ cho các con trỏ trong mảng con trỏ `pAIn` để quản lý các địa chỉ lưu trữ các số `int`  
 b. Cấp phát bộ nhớ cho mảng con trỏ `pAIn` để lưu trữ các số `int`  
 c. Cấp phát bộ nhớ cho mảng con trỏ `pAIn`  
 d. Khởi tạo địa chỉ ban đầu cho mảng con trỏ `pAIn`
- (6)- Cho khai báo `int *p, a[100];`  
 Lệnh nào dưới đây gây lỗi (sai cú pháp)?  
 a. `p=a;`      b. `a++;`      c. `p=&a[0];`      d. a,b,c đều sai
- (7)- Chọn câu đúng nhất về kiểu dữ liệu con trỏ  
 a. Các ngôn ngữ lập trình thường cung cấp cho chúng ta một kiểu dữ liệu đặc biệt để lưu trữ các địa chỉ của bộ nhớ, đó là con trỏ.  
 b. Con trỏ là kiểu dữ liệu cơ sở.  
 c. Con trỏ là kiểu dữ liệu do người dùng định nghĩa.  
 d. Cả a, b, c đều sai.

(8)- Chọn câu đúng nhất về kiểu dữ liệu con trỏ

- a. Các ngôn ngữ lập trình thường cung cấp cho chúng ta một dữ liệu đặc biệt để lưu trữ các địa chỉ của bộ nhớ, đó là con trỏ.
- b. Con trỏ là kiểu dữ liệu cơ sở
- c. Con trỏ là kiểu dữ liệu do người dùng định nghĩa
- d. Cả a, b, c đều sai

(9)- Xét đoạn mã sau

```
void main()
{
    int *x, y=2;
    x=&y;
    *x= *x+1;
    cout<<y;
}
```

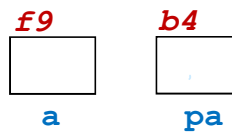
Cho biết kết quả sau khi thi hành chương trình trên ?

- a. 1
- b. 2
- c. 3
- d. 4

### 9.10.2. Đọc chương trình, cho biết kết quả

(10)- Giả sử biết địa chỉ của biến a là f9, địa chỉ của biến con trỏ pa là b4. Cho biết kết quả thực hiện của chương trình sau:

```
int main()
{
    int a=4, *pa;
    pa = &a;
    printf("%d \n", a) ;
    printf("%d \n", *pa);
    printf("%d \n", *&pa);
    printf("%d \n", &*pa);
    printf("%x \n", &a);
    printf("%x \n", pa);
    printf("%x \n", &*pa);
    printf("%x \n", *(&pa));
    printf("%x \n", *(&pa));
    a = *pa*2;
    printf("%d \n", a);
    printf("%d \n", *pa);
    printf("%x \n", &pa);
    return 0;
}
```



(11)- Cho biết kết quả thực hiện của chương trình sau:

```
int main()
{
    int a, b, i, *pa,*pb,*pi;
    pa=&a;
    pb=&b;
    pi=&i;
    printf("Nhap a: ");
    scanf("%d",pa);
    printf("Nhap b: ");
    scanf("%d",pb);
}
```

```

printf("%d + %d = %d\n\n", *pb, *pa, *pa+*pb);
for (i=1; i<11; i++)
    printf("%d x %d = %d\n", *a, *i, *pa*( *pi));
return 0;
}

```

(12)- Cho biết kết quả thực hiện của chương trình sau:

```

int main()
{
    int i = 1, n = 12, d = 0, *pa, *pb, *pc;
    pa = &n;
    pb = &i;
    pc = &d;
    for (; *pb <= *pa; (*pb)++)
        if ((*pa) % (*pb) == 0)
        {
            printf("%5d", *pb);
            (*pc)++;
        }
    printf("\nSLUS của %d la %d", *pa, *pc);
    return 0;
}

```

(13)- Cho biết kết quả thực hiện của chương trình sau:

```

int main()
{
    int *pa, *pb, *pc;
    pa = new int (100);
    pb = new int (1);
    pc=pb;
    for (; (*pb) <= (*pa); (*pb)++)
        if ((int) sqrt ((*pb)) * (int) sqrt ((*pb)) == (*pb))
            printf("%5d", (*pc));
    return 0;
}

```

(14)-

```

int main()
{
    int x=20, i;
    printf("x= %d\n", x);
    int *p=&x;
    printf("*p= %d\n", *p);
    x=x*x;
    printf("*p= %d\n", *p);
    p=(int*) calloc(6, sizeof(int));
    for(i=0; i<6; i++)
        *(p+i)=i*10;
    for(i=0; i<6; i++)
        printf("%d ", *(p+i));
    return 0;
}

```

(15)-

```

int main()
{
    int *A, *pa, *pb;
    pa = new int(100);
    A = new int[*pa/5];
    pb = new int(0);
    while (*pa > 0)
    {
        A[(*pb)++] = *pa;
        *pa -=5;
    }
    printf("\n");
    while (*pb > 0)
        printf("%5d", A[--(*pb)]);
    return 0;
}

```

(16)-

```
int main()
{
    int *p= new int(5);
    int *a = (int*)calloc(*p, sizeof(int));
    int *b = (int*)calloc(*p, sizeof(int));
    for (int i = *p; i >=0; i--)
        *(a + i) = 2 * i +1;
    for (int i = *p; i >=0; i--)
        *(b + i) = (i+*(a + i)) + *(a + i);
    for (int i = 0; i < *p; i++)
        printf("%5d",*(b + i));
    printf("\n");
    for (int i = 0; i < *p; i++)
        printf("%5d",*(b + i)-*(a + i));
    free(a);
    free(b);
}
```

(17)- Cho biết kết quả thực hiện của chương trình sau:

```
int main()
{
    int *p= new int(5);
    int *a = (int*)calloc(*p, sizeof(int));
    int *b = (int*)calloc(*p, sizeof(int));
    for (int i = 0; i < *p; i++)
        *(a + i) = i * i;
    for (int i = 0; i < *p; i++)
        *(b + i) = (i+*(a + i)) + *(a + i);
    for (int i = 0; i < *p; i++)
        printf("%5d",*(b + i));
    printf("\n");
    for (int i = 0; i < *p; i++)
        printf("%5d",*(b + i)-*(a + i));
    free(a);
    free(b);
}
```

**Dùng phương pháp con trỏ để thực hiện tất cả các bài tập sau**

### **9.10.3. Mảng một chiều**

- (18)- Viết hàm nhập vào một mảng một chiều các số nguyên gồm n phần tử ( $0 < n < 100$ )
- (19)- Viết hàm nhập vào một mảng một chiều các số thực gồm n phần tử ( $0 < n < 100$ )
- (20)- Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên
- (21)- Viết hàm xuất mảng số thực n phần tử vừa nhập ở trên
- (22)- Tính tổng các phần tử có trong mảng
- (23)- Tính tổng các phần tử chẵn có trong mảng
- (24)- Tính tổng các phần tử lẻ có trong mảng
- (25)- Tính tổng các phần tử nguyên tố có trong mảng

- (26)- Tìm phần tử chẵn đầu tiên có trong mảng
- (27)- Tìm phần tử lẻ đầu tiên có trong mảng
- (28)- Tìm phần tử nguyên tố đầu tiên có trong mảng
- (29)- Tìm phần tử chẵn cuối cùng có trong mảng
- (30)- Tìm phần tử chính phương cuối cùng có trong mảng
- (31)- Tìm phần tử lớn nhất có trong mảng
- (32)- Đếm số phần tử chẵn có trong mảng
- (33)- Đếm số phần tử lớn nhất có trong mảng
- (34)- In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (35)- Thêm một phần tử vào đầu mảng.
- (36)- Thêm một phần tử vào cuối mảng.
- (37)- Thêm một phần tử vào vị trí x trong mảng.
- (38)- Xóa phần tử chẵn đầu tiên.
- (39)- Xóa tất cả các phần tử lớn nhất trong mảng
- (40)- Sắp xếp mảng tăng dần
- (41)- Viết hàm nhập vào một mảng hai chiều các số nguyên gồm m dòng, n cột ( $0 < m, n < 100$ )
- (42)- Viết hàm nhập vào một mảng hai chiều các số thực gồm m dòng, n cột ( $0 < m, n < 100$ )
- (43)- Viết hàm xuất mảng hai chiều các số nguyên mxn phần tử vừa nhập ở trên
- (44)- Viết hàm xuất mảng hai chiều các số thực mxn phần tử vừa nhập ở trên
- (45)- Tính tổng các phần tử có trong mảng
- (46)- Tính tổng các phần tử chẵn có trong mảng
- (47)- Tính tổng các phần tử lẻ có trong mảng
- (48)- Tính tổng các phần tử nguyên tố có trong mảng
- (49)- Tính tổng các phần tử nằm trên một dòng
- (50)- Tính tổng các phần tử nằm trên đường chéo chính có trong mảng
- (51)- Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng
- (52)- Tính tổng các phần tử nằm trên đường biên có trong mảng
- (53)- Tìm phần tử lớn nhất có trong mảng
- (54)- Tìm và in ra vị trí của phần tử lớn nhất đầu tiên có trong mảng
- (55)- Tìm phần tử chẵn đầu tiên có trong mảng
- (56)- Tìm phần tử lẻ đầu tiên có trong mảng
- (57)- Tìm phần tử nguyên tố đầu tiên có trong mảng
- (58)- Tìm phần tử chẵn cuối cùng có trong mảng

- (59)- Tìm phần tử chính phương cuối cùng có trong mảng
- (60)- Đếm số phần tử chẵn có trong mảng
- (61)- Đếm số phần tử lớn nhất có trong mảng
- (62)- Tìm dòng có tổng lớn nhất
- (63)- Sắp xếp mảng tăng dần
- (64)- Thêm một phần tử vào đầu mảng.
- (65)- Thêm một phần tử vào cuối mảng.
- (66)- Thêm một phần tử vào vị trí x trong mảng.
- (67)- Xóa phần tử chẵn đầu tiên.
- (68)- Xóa tất cả các phần tử lớn nhất trong mảng

#### **9.10.4. Chuỗi ký tự**

- (69)- Nhập chuỗi
- (70)- Xuất chuỗi
- (71)- Đếm số ký tự ' a ' có trong chuỗi
- (72)- Cắt khoảng trắng có trong chuỗi
- (73)- Đếm khoảng trắng trong chuỗi
- (74)- Đếm số từ có trong chuỗi
- (75)- Sắp xếp chuỗi tăng dần
- (76)- Nhập vào 2 chuỗi, xuất chuỗi theo thứ tự từ điển
- (77)- Nhập 3 chuỗi, xuất chuỗi theo thứ tự từ điển

#### **9.10.5. Mảng hai chiều**

- (78)- Viết hàm nhập vào số dòng, số cột ( $0 < m, n < 100$ ) .
- (79)- Viết hàm nhập vào mảng hai chiều các số nguyên gồm m dòng, n cột ( $0 < m, n < 100$ ).
- (80)- Viết hàm xuất mảng hai chiều các số nguyên vừa nhập ở trên.
- (81)- Tính tổng các phần tử có trong mảng .
- (82)- Tính tổng các phần tử chẵn có trong mảng .
- (83)- Tính tổng các phần tử nguyên tố có trong mảng .
- (84)- Tính tổng các phần tử nằm trên đường chéo chính có trong mảng .
- (85)- Tính tổng các phần tử nằm trên đường chéo phụ có trong mảng .
- (86)- Tính tổng các phần tử nằm trên đường biên.
- (87)- Tìm phần tử lớn nhất có trong mảng
- (88)- Đếm số phần tử lẻ có trong mảng
- (89)- Đếm số phần tử lớn nhất có trong mảng
- (90)- In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng

- (91)- Tính tổng các phần tử nằm trên một dòng .
- (92)- Tìm dòng có tổng lớn nhất
- (93)- Xoá dòng
- (94)- Xoá cột
- (95)- Sắp xếp mảng tăng dần
- (96)- Xoay mảng về trái .
- (97)- Xoay mảng về phải

#### 9.10.6.Struct

##### (98)- **Phân số**

- (i). Viết hàm nhập vào một phân số
- (ii). Viết hàm nhập vào một dãy phân số
- (iii). Viết hàm xuất một phân số
- (iv). Viết hàm xuất một dãy phân số
- (v). Viết hàm tìm phân số lớn nhất trong dãy phân số
- (vi). Viết hàm tính tổng các phân số có trong dãy

##### (99)- **Sinh viên**

- (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: Quản trị , 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 Nữ thuộc ngành công nghệ thông tin.
  - (vii). Sắp xếp danh sách sinh viên theo tên
  - (viii). Sắp xếp danh sách sinh viên theo điểm toán .
-

## 10 ĐỆ QUY (Recursion)

Sau khi học xong chương 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.

### 10.1. 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 10.1: giai thừa của một số nguyên dương  $n$  được định nghĩa như sau:

$$\begin{aligned} n! &= 1 * 2 * 3 * \dots * (n-1) * n \\ &= (n-1)! * n \quad (\text{với } 0! = 1) \end{aligned}$$

Như vậy, để tính  $n!$  ta thấy nếu  $n=0$  thì  $n!=1$  ngược lại thì  $n! = (n-1)! * n$

Với định nghĩa trên thì hàm đệ quy tính  $n!$  được viết:

```
#include <stdio.h>
#include <conio.h>
int GiaiThua (int n)
{
    if (n==0)
        return 1;
    else
        return GiaiThua (n-1) * n;
}
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(n));
    return 0;
}
```

Ví dụ 10.2 : 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 tinhhtong (int n)
{
    if (n == 0)
        return 0;
    long s= tinhhtong (n - 1);
    return s + n ;
}
```

## 10.2. 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 .

### 10.2.1.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

```
<kiểu dữ liệu trả về của hàm> < tên hàm >
{
    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 hàm > */
    }
}
```

- Ví dụ 10.3 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);
}
```

### 10.2.2.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:

```
<kiểu dữ liệu trả về của hàm> <tên hàm>
{
    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 hàm> để giải quyết vấn đề nhỏ hơn*/
        /*gọi đệ quy đến hàm <tên hàm> để giải quyết vấn đề còn lại*/
    }
}
```

- Ví dụ 10.4: 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);
}
```

### 10.2.3. 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:

```
<kiểu dữ liệu trả về của hàm> <tên hàm>
{
    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 hàm>
        }
    }
}
```

- Ví dụ 10.5 Cho dãy  $X_n$  được xác định theo công thức truy hồi như sau:

$$\begin{cases} X_0 = 1 \\ X_n = 1^2x_0 + 2^2x_1 + \dots + n^2x_{n-1} \end{cases}$$

```
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;
}
```

#### 10.2.4.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.
- Các hàm đệ quy hỗ tương có dạng như sau:

```
<Kiểu dữ liệu hàm> TenHam1 (<danh sách tham số>)
{
    if<điều kiện dừng>
    {
        // làm một số công việc ...
    }
    ... TenHam2 (<danh sách tham số>);
    //Thực hiện một số công việc (nếu có)
}
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>)
{
    if<điều kiện dừng>
    {
        // làm một số công việc ...
    }
    ... TenHam1 (<danh sách tham số>);
    //Thực hiện một số công việc (nếu có)
}
```

- Ví dụ 10.6 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) + TinhY(n-1) ;
}
```

```
long TinhY (int n)
{
    if (n == 0)
        return 1;
    return 2 * TinhX(n-1) + 3 * TinhY(n-1);
}
```

### 10.3. Xây dựng hàm đệ quy

#### 10.3.1. Các bước đề nghị khi xây dựng hàm đệ quy

- Bước 1. Tham số cần truyền cho hàm là gì? Số lượng tham số là bao nhiêu?  
Bước 2. Hàm có trả về giá trị hay không? (*void, int, float, ...*)  
Bước 3. Điều kiện dừng là gì?  $\Rightarrow$  Khi đó giá trị trả về là bao nhiêu?  
Bước 4. Phần không đệ quy là gì? (*n-1, n/10, n%10, A[n-1], ...*)  
Bước 5. Phần đệ quy là gì?  $\Rightarrow$  Kích thước nhỏ hơn của *n* là gì (*n-1, n/10, n%10, ...*)?  
Bước 6. Tinh chỉnh lại nội dung hàm (nếu có thể).

#### 10.3.2. Ví dụ 10.7

Yêu cầu viết hàm đệ quy tính giai thừa cho số nguyên dương *n*.

**B1:** Tham số cần truyền cho hàm là số nguyên *n*.

**B2:** Hàm trả về số nguyên là giai thừa của *n*.

**B3:** Điều kiện dừng: khi *n=1*  $\Rightarrow 1!=1$ .

Do:

$$(n)! = 1 * 2 * 3 * \dots * (n-2) * (n-1) * n$$

$$(n-1)! = 1 * 2 * 3 * \dots * (n-2) * (n-1)$$

$$(n-2)! = 1 * 2 * 3 * \dots * (n-2)$$

**B4:**  $\Rightarrow$  Phần không đệ quy: ***n***

**B5:**  $\Rightarrow$  Phần đệ quy: ***n-1***

Nội dung hàm cần xây dựng:

```

B2 → long GiaiThua (int n) ← B1
{
    if (n==1) ← B3
        return 1;
    else //gọi đệ quy ← B5
        return GiaiThua(n-1) * n; ← B4
}
    
```

**B6:** Tinh chỉnh lại hàm ở B5:

```
long GiaiThua (int n)
{
    if (n==1)
        return 1;
    return GiaiThua(n-1) * n ;
}
```

**10.3.3. Ví dụ 10.8**

Viết hàm đệ quy tính tổng các số có trong mảng 1 chiều các số nguyên A gồm n phần tử.

**B1:** Tham số cần truyền cho hàm là: mảng số nguyên A và số nguyên n (số lượng phần tử đang có trong mảng).

**B2:** Hàm trả về số nguyên là tổng các số có trong mảng A.

Do chỉ số của các phần tử trong mảng được tính từ 0 đến n-1, nên:

Tổng các số từ A[0] đến A[n-1]: Tổng các giá trị từ A[0] đến A[n-2] + **A[n-1]**

Tổng các số từ A[0] đến A[n-2]: Tổng các giá trị từ A[0] đến A[n-3] + **A[n-2]**

Tổng các số từ A[0] đến A[n-3]: Tổng các giá trị từ A[0] đến A[n-4] + **A[n-3]**

**B3:** Điều kiện dừng: khi n=1  $\Rightarrow$  return A[0] hoặc khi n=0 return 0.

**B4:**  $\Rightarrow$  Phần không đệ quy: **A[n-1]**

**B5:**  $\Rightarrow$  Phần gọi đệ quy: mảng **A** và số lượng phần tử là **n-1**

Nội dung hàm cần xây dựng:

```
int Tong(int A[], int n)
{
    if (n == 0)
        return 0;
    return Tong(A, n - 1) + A[n - 1];
}
```

**10.3.4. Ví dụ 10.9**

Viết hàm đệ quy tính tổng các số lẻ có trong mảng 1 chiều các số nguyên A gồm n phần tử.

Tương tự như ví dụ trước, hàm này có tham số truyền cho hàm, kiểu dữ liệu trả về và điều kiện dừng là không thay đổi, chỉ khác ở bước 4 và bước 5

**B1:** Tham số cần truyền cho hàm là: mảng số nguyên A và số nguyên n (số lượng phần tử đang có trong mảng).

**B2:** Hàm trả về số nguyên là tổng các số có trong mảng A.

Do chỉ số của các phần tử trong mảng được tính từ 0 đến n-1, nên:

Tổng các số từ A[0] đến A[n-1]: Tổng các giá trị từ A[0] đến A[n-2] + **A[n-1]**

Tổng các số từ A[0] đến A[n-2]: Tổng các giá trị từ A[0] đến A[n-3] + **A[n-2]**

Tổng các số từ A[0] đến A[n-3]: Tổng các giá trị từ A[0] đến A[n-4] + **A[n-3]**

**B3:** Điều kiện dừng: khi n=1  $\Rightarrow$  return A[0] hoặc khi n=0 return 0.

**B4:** Phần không đệ quy:

□ Nếu A[n-1] là số lẻ: **+ A[n-1]**

□ Nếu A[n-1] là số chẵn: **+ 0.**

**B5:** Phần gọi đệ quy: luôn mảng **A** và số lượng phần tử là **n-1**

Nội dung hàm cần xây dựng:

```
int TongLe(int A[], int n)
{
    if (n == 0)
        return 0;
    if (A[n - 1] % 2 == 1)
        return TongLe(A, n-1) + A[n-1];
    else
        return TongLe(A, n-1);
}
```

**B6:** Tinh chỉnh lại nội dung của hàm ở bước 5:

```
int TongLe(int A[], int n)
{
    if (n == 0)
        return 0;
    return TongLe(A, n-1) + (A[n-1] % 2 == 1 ? A[n-1] : 0);
}
```

## 10.4. Một số bài toán kinh điển dùng phương pháp đệ quy

### 10.4.1. Bài toán Tháp Hà Nội

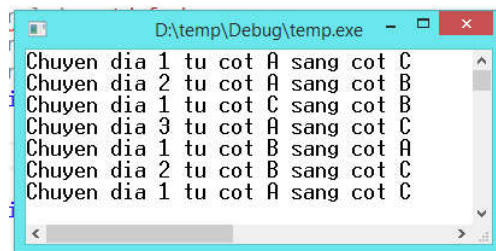
Truyền thuyết kể rằng: Một nhà toán học Pháp sang Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa quý gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo cách:

- 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  $n=64$   $T=(2^{64} - 1) * t$ . Giả sử  $t=1/100s$  thì  $T = 5.8$  tỷ năm.

```
#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()
{
    ThapHaNoi(3, 'A', 'B', 'C');
}
```



Mã lệnh

Kết quả

### 10.4.2. 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 phương pháp chia để trị cho việc 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);
    }
}
```

### 10.5. 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).

### 10.6. 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.

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

Dùng phương pháp đệ quy giải các bài tập sau

### 10.7.1. Thao tác trên 1 số nguyên

- (1)- Đếm số lượng chữ số của số nguyên dương n. VD:  $n=29730 \Rightarrow 29730$  gồm 5 chữ số
- (2)- Tính tổng các chữ số của số nguyên dương n.
- (3)- Tính tích các chữ số của số nguyên dương n.
- (4)- Đếm số lượng chữ số lẻ của số nguyên dương n.
- (5)- Tính tổng các chữ số chẵn của số nguyên dương n.
- (6)- Tính tích các chữ số lẻ của số nguyên dương n.
- (7)- Cho số nguyên dương n. Tìm chữ số đầu tiên của n.
- (8)- Tìm chữ số đảo ngược của số nguyên dương n.
- (9)- Tìm chữ số lớn nhất của số nguyên dương n.
- (10)- Tìm chữ số nhỏ nhất của số nguyên dương n.
- (11)- Kiểm tra số nguyên dương n có toàn chữ số lẻ hay không?
- (12)- Tìm ước số lẻ lớn nhất của số nguyên dương n. Ví dụ  $n=100$  ước số lẻ lớn nhất của 100 là 25.
- (13)- Kiểm tra số nguyên dương n có toàn chữ số chẵn hay không?

### 10.7.2. Ước số chung lớn nhất

- (14)- Tìm ước số chung lớn nhất của hai số nguyên dương a và b.

### 10.7.3. Tổng/tích một dãy số

- (15)- Tính  $P(n) = 1 \times 3 \times 5 \times \dots \times (2n + 1)$ , với  $n > 0$
- (16)- Tính  $S(n) = 1 - 2 + 3 - 4 + \dots + (-1)^{n+1} n$ , với  $n > 0$
- (17)- Tính  $S(n) = 1 + 1.2 + 1.2.3 + \dots + 1.2.3 \dots n$ , với  $n > 0$
- (18)- Tính  $S(n) = 1^2 + 2^2 + 3^2 + \dots + n^2$ , với  $n > 0$
- (19)- Tính  $S(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$ , với  $n > 0$
- (20)- Tính  $S(n) = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \dots + \frac{1}{1+2+3+\dots+n}$ , với  $n > 0$
- (21)- Tính  $P(x,y) = x^y$
- (22)- Tính  $S(n) = 1 + (1+2) + (1+2+3) + \dots + (1+2+3+\dots+n)$ , với  $n > 0$



- (23)- Tính  $S(n) = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \dots + \frac{1}{2n}$
- (24)- Tính  $S(n) = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \dots + \frac{1}{2n+1}$
- (25)- Tính  $S(n) = \frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \frac{1}{3 \times 4} + \dots + \frac{1}{n \times (n+1)}$
- (26)- Tính  $S(n) = \frac{1}{2} + \frac{3}{4} + \frac{5}{6} + \dots + \frac{2n+1}{2n+2}$
- (27)- Tính  $S(n) = x^2 + x^4 + \dots + x^{2n}$ .
- (28)- Tính  $S(n) = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \frac{1}{1+2+3+4} + \dots + \frac{1}{1+2+3+\dots+n}$
- (29)- Tính  $S(n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$

#### 10.7.4. Vẽ hình

- (30)- Viết chương trình cho người dùng nhập cạnh (n) của tam giác. Giả sử với n=5, chương trình sẽ lần lượt in ra các hình sau:

| a         | b | c         | d |
|-----------|---|-----------|---|
| * * * * * |   | * * * * * |   |
| * * * *   |   | * * * *   |   |
| * * *     |   | * * *     |   |
| * *       |   | * *       |   |
| *         |   | *         |   |

| e         | f | g         | h |
|-----------|---|-----------|---|
| * * * * * |   | * * * * * |   |
| * * * *   |   | * * * *   |   |
| * * *     |   | * * *     |   |
| * * *     |   | * * *     |   |
| * * *     |   | * * *     |   |

#### 10.7.5. Mảng một chiều

- (31)- Viết hàm nhập mảng một chiều các số nguyên gồm n phần tử ( $0 < n \leq 100$ ).
- (32)- Viết hàm xuất mảng số nguyên n phần tử vừa nhập ở trên.
- (33)- Tính tổng tất cả các phần tử trong mảng.
- (34)- Tính tổng các số lẻ có trong mảng.  
 ✎ Mở rộng: cho trường hợp: số chẵn, số nguyên tố, số chính phương, ...
- (35)- Tìm phần tử chẵn đầu tiên có trong mảng.  
 ✎ Mở rộng: cho trường hợp: số lẻ, số nguyên tố, số chính phương, ...
- (36)- Tìm phần tử chẵn cuối cùng có trong mảng.  
 ✎ Mở rộng: cho trường hợp giá trị cuối cùng là số lẻ, số nguyên tố, giá trị cuối cùng là số chính phương, ...
- (37)- Tìm phần tử lớn nhất có trong mảng.

- (38)- Đếm số phần tử chẵn có trong mảng.
- (39)- Đếm số phần tử lớn nhất có trong mảng.
- (40)- In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng.
- (41)- Sắp xếp mảng tăng dần.
- (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?

### 10.7.6. 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 mảng hai chiều A ( $m * n$ ) 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 mảng hai chiều. 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 |

Có 4 thành phần liên thông(TPLT)

|    |    |    |    |
|----|----|----|----|
| 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  |

Có 3 TPLT

### 10.7.7. Một số bài toán đệ quy thông dụng

- (49)- Bài toán phát sinh hoán vị: Cho tập hợp A có n phần tử được đánh số 1,2,...,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.
- (50)- Bài toán Tám Hậu: Cho bàn cờ vua kích thước (8x8). Hãy sắp 8 quân hậu vào bàn cờ sao cho không có bất kỳ 2 quân hậu nào có thể ăn nhau.
- (51)- Bài toán Mã Đi Tuần: Cho bàn cờ vua kích thước (8x8). Hãy di chuyển quân mã trên khắp bàn cờ sao cho mỗi ô đi đúng 1 lần.

## 11 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.

### 11.1. 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).

#### 11.1.1. Phân loại kiểu tập tin

Có 3 loại dữ liệu kiểu tập tin:

##### 11.1.1.1. 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.

##### 11.1.1.2. 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.

##### 11.1.1.3. 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.

#### 11.1.2. 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.

### 11.1.3. 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*).

## 11.2. Các bước 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*.

### 11.2.1. 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 11.1** `FILE *f1, *f2;`

### 11.2.2. 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 11.2** 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 của các chế đ 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.

### 11.2.3. Các thao tác đọc/ghi dữ liệu trên file

Tùy thuộc chế độ (mode) được dùng khi mở tập tin (văn bản hay nhị phân) mà ta có thể sử dụng các hàm đọc ghi tương ứng. Các hàm đọc/ghi trên file văn bản hoặc nhị phân sẽ được giới thiệu trong các mục 11.3. Truy cập tập tin văn bản và 11.4. Truy cập tập tin nhị phân.

### 11.2.4. Đóng tập tin

#### 11.2.4.1. 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.
  - Là *EOF* nếu có xuất hiện lỗi.

#### 11.2.4.2. 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*.

### 11.3. Truy cập tập tin văn bản

#### 11.3.1. Ghi dữ liệu lên tập tin văn bản

##### 11.3.1.1. 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.

##### 11.3.1.2. Hàm `fputs()`

- Hàm này dùng để ghi một chuỗi ký tự chứa trong vùng đệm lên tập tin văn bản.
- Cú pháp: `int fputs (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.

##### 11.3.1.3. 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                                                        |
|--------------------------------------------------------------------------------|----------------------------------------------------------------|
| <code>%d</code>                                                                | Ghi số nguyên                                                  |
| <code>%.số chữ số thập phân] f</code>                                          | Ghi số thực có <số chữ số thập phân> theo quy tắc làm tròn số. |
| <code>%o</code>                                                                | Ghi số nguyên hệ bát phân                                      |
| <code>%x</code>                                                                | Ghi số nguyên hệ thập lục phân                                 |
| <code>%c</code>                                                                | Ghi một ký tự                                                  |
| <code>%s</code>                                                                | Ghi chuỗi ký tự                                                |
| <code>%e</code> hoặc <code>%E</code> hoặc <code>%g</code> hoặc <code>%G</code> | Ghi số thực dạng khoa học (nhân 10 mũ x)                       |

##### 11.3.1.4. Ví dụ 11.3

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

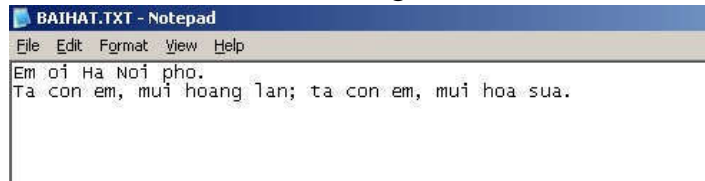
```
#include<stdio.h>
#include<conio.h>
int main ()
{
    FILE *f;
    clrscr ();
    f = fopen ("D:\\Baihat.txt", "r+");
    if (f!=NULL)
```

```

    {
        fputs("Em oi Ha Noi pho.\n",f);
        fputs("Ta con em, mui hoang lan; ta con em, mui hoa sua.",f);
        fclose(f);
    }
    return 0;
}

```

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



### 11.3.2. Đọc dữ liệu từ tập tin văn bản

#### 11.3.2.1. 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*.

#### 11.3.2.2. Hàm `fgetc()`

- Cú pháp: `char *fgetc (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*.

#### 11.3.2.3. 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 11.4:** 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();
    }
    return 0;
}
```

## 11.4.Truy cập tập tin nhị phân

### 11.4.1.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.

### 11.4.2.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.



### 11.4.3. 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*. *fseek* trả về:
  - = 0 nếu thành công.
  - <>0 nếu di chuyển có lỗi.
- *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

### 11.4.4. 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.

### 11.4.5. 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)`

## 11.5. Bài tập có hướng dẫn

### 11.5.1. Bài tập 1

- **Yêu cầu**: Viết chương trình tạo tập tin chứa 10000 số nguyên ghi vào file *SONGUYEN.INP* mỗi dòng 10 số, với giá trị của các số nguyên được phát sinh ngẫu nhiên và nằm trong khoảng từ 0-999. Sau đó viết chương trình đọc file *SONGUYEN.INP*, sắp xếp theo thứ tự tăng dần và lưu kết quả vào file *SONGUYEN.OUT*. Thực hiện bằng 2 cách:
  - Thao tác trên tập tin văn bản.
  - Thao tác trên tập tin nhị phân.

- **Thực hiện**

(i)- Thực hiện trên tập tin văn bản

```
//Khai báo thư viện cần dùng
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<conio.h>
//Định nghĩa các biến
#define MAX 10000
#define VALUE 1000
#define InFile "D:\\SoNguyen1.inp"
#define OutFile "D:\\SoNguyen1.out"
//Khai báo các nguyên mẫu hàm
void TaoFile();
void DocFile(int a[]);
void swap(int &a, int &b);
void Sort(int a[]);
void TaoFileKQ(int a[]);
void main()
{
    srand(time(NULL));
    int a[MAX];
    TaoFile();
    DocFile(a);
    XuatMang(a);
    TaoFileKQ(a);
}
void TaoFile()
{
    FILE *f;
    f = fopen(InFile, "wt");
    if (f == NULL)
        printf("Khong tao duoc file");
    else
    {
        for (int i = 0; i < MAX / 10; i++)
        {
            for (int j = 0; j < 10; j++)
            {
                int so = rand() % VALUE;
                fprintf(f, "%5d", so);
            }
        }
        fclose(f);
    }
}
void DocFile(int a[])
{
    FILE *f;
    f = fopen(InFile, "rt");
    if (f == NULL)
        printf("Khong tao duoc file");
    else
    {
        for (int i = 0; i < MAX; i++)
```

```

        fscanf(f, "%d", &a[i]);
        fclose(f);
    }
}

void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void Sort(int a[])
{
    for (int i = 0; i < MAX - 1; i++)
        for (int j = i + 1; j < MAX; j++)
            if (a[i]>a[j])
                swap(a[i], a[j]);
}

void TaoFileKQ(int a[])
{
    //B1: Sắp xếp mảng a tăng dần
    Sort(a);
    //B2: Ghi mảng a đã được sắp tăng dần vào file
    FILE *f;
    f = fopen(OutFile, "wt");
    if (f == NULL)
        printf("Khong tao duoc file");
    else
    {
        for (int i = 0; i < MAX ; i++)
            fprintf(f, "%d", a[i]);
        fclose(f);
        printf("Da tao thanh cong file %s", OutFile);
    }
}

```

(ii)- Thực hiện trên tập tin nhị phân

```

//Khai báo thư viện cần dùng
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<conio.h>
//Định nghĩa các biến
#define MAX 10000
#define InFile "D:\\SoNguyen2.inp"
#define OutFile "D:\\SoNguyen2.out"
//Khai báo các nguyên mẫu hàm
void TaoFile();
void DocFile(int a[]);
void swap(int &a, int &b);
void Sort(int a[]);
void TaoFileKQ(int a[]);

void main()
{
    srand(time(NULL));

```

```

    int a[MAX];
    TaoFile();
    DocFile(a);
    TaoFileKQ(a);
}

void TaoFile()
{
    FILE *f;
    f = fopen(OutFile, "wb");
    if (f == NULL)
        printf("Khong tao duoc file");
    else
    {
        for (int i = 0; i < MAX/10; i++)
        {
            for (int j = 0; j < 10; j++)
            {
                int so = rand()%1000;
                fwrite(&so, sizeof(int), 1, f);
            }
        }
        fclose(f);
    }
}

void DocFile(int a[])
{
    FILE *f;
    f = fopen(OutFile, "rb");
    if (f == NULL)
        printf("Khong tao duoc file");
    else
    {
        for (int i = 0; i < MAX; i++)
            fread(&a[i], sizeof(int), 1, f);
        fclose(f);
    }
}

void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void Sort(int a[])
{
    for (int i = 0; i < MAX - 1; i++)
        for (int j = i + 1; j < MAX; j++)
            if (a[i]>a[j])
                swap(a[i], a[j]);
}

void TaoFileKQ(int a[])
{
    //B1: Sắp xếp mảng a tăng dần
    Sort(a);
    //B2: Ghi mảng a đã được sắp tăng dần vào file
    FILE *f;

```

```

f = fopen(OutFile, "wb");
if (f == NULL)
    printf("Khong tao duoc file");
else
{
    for (int i = 0; i < MAX; i++)
        fwrite(&a[i], sizeof(int), 1, f);
    fclose(f);
    printf("Da tao thanh cong file %s", OutFile);
}
}

```

### 11.5.2. Bài tập 2

#### - Yêu cầu

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.

#### - Thực hiện

```

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

```

```

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

```

## 11.6. Bài tập sinh viên tự thực hiện

### 11.6.1. Tập tin văn bản (Text file)

#### 11.6.1.1. Thao tác trên dữ liệu đơn

(1)- Viết chương trình gồm các hàm sau:

- Hàm tạo file có chức năng tạo một file chứa 10000 số nguyên ngẫu nhiên có giá trị từ 1 đến 32767 với tên file là "SoNguyen.inp".
- Hàm đọc file " SoNguyen.inp" vừa tạo ở trên và xuất ra màn hình trên nhiều dòng, mỗi dòng 10 số.
- Hàm đọc file "SoNguyen.inp" và ghi các số chẵn vào file "SoChan.out" và những số lẻ vào file "SoLe.out".

#### 11.6.1.2. Mảng 1 chiều

(2)- Cho file "Array.inp" chứa dữ liệu của 1 mảng các số nguyên A gồm n phần tử, với cấu trúc của file như sau:

- Dòng 1 chứa số nguyên n ( $0 < n \leq 100$ ).
- Dòng 2 chứa n số nguyên cách nhau bởi khoảng trắng.

- Viết hàm đọc file "Array.inp" và in n số có trong file này ra màn hình.
- Viết hàm đọc file "Array.inp" và lưu vào mảng 1 chiều A.
- Viết hàm các số nguyên tố có trong mảng A. Yêu cầu: các số nguyên tố được ghi tiếp theo nội dung đã có (không xóa nội dung cũ).
- Thực hiện tìm phần tử lớn nhất có trong mảng A, sau đó ghi kết quả sắp xếp vào file "Array1.out", với cấu trúc của file "Array1.out" như sau:
  - Dòng 1 ghi giá trị lớn nhất có trong mảng.
  - Dòng 2 ghi chỉ số (vị trí) của chứa các số lớn nhất trong mảng (do trong mảng có thể chứa nhiều số cùng có giá trị lớn nhất).
- Thực hiện sắp xếp các số nguyên trong mảng A tăng dần sau đó ghi kết quả sắp xếp vào file "Array2.out", với cấu trúc của file "Array2.out" như sau:
  - Dòng 1 chứa n phần tử của mảng các số nguyên.
  - Dòng 2 chứa n số nguyên đã được xếp tăng dần.
- Thực hiện đếm số lần xuất hiện của từng giá trị có trong mảng A, sau đó ghi kết quả thực hiện vào file "Array3.out", với cấu trúc của file "Array3.out" gồm nhiều dòng, mỗi dòng chứa 2 số: giá trị và số lần xuất hiện của giá trị đó.

Ví dụ nội dung tập tin "Array.inp" như sau:

```
8
5 4 2 -2 4 5 9 5
```

Nội dung file "Array3.out" sau khi hoàn tất

```
-2 1
2 1
4 2
5 3
9 1
```

(3)- Viết chương trình lần lượt thực hiện các chức năng như sau:

a.- Cho người dùng nhập các số nguyên, việc nhập kết thúc khi người dùng nhập vào giá trị 0. Ghi vào file MANG.TXT có nội dung như sau:

- Dòng đầu chứa số n.
- Dòng kế tiếp chứa các số nguyên mà người dùng đã nhập (không ghi số 0 cuối cùng mà người dùng đã nhập).

Ví dụ nội dung tập tin MANG.TXT như sau:

```
6
5 36 2 -2 4 9
```

b.- Viết hàm đọc file Mang.txt và in giá trị của n số có trong file ra màn hình.

c.- Viết hàm đọc file Mang.txt và thực hiện:

- Đếm xem trong file có bao nhiêu số chính phương?
- In các số chính phương này ra màn hình.
- Tìm số chính phương nhỏ nhất.

d.- Viết hàm đọc file Mang.txt và thực hiện:

- Đếm xem trong file có bao nhiêu số nguyên tố palindrome?
- In các số nguyên tố palindrome này ra màn hình.
- Tìm số nguyên tố palindrome lớn nhất.

☞ Số nguyên tố Palindrome: Là số nguyên tố vẫn giữ nguyên giá trị khi các chữ số của nó được đảo ngược. Ví dụ: 2, 3, 5, 7, 11, 101, 131, 151, 181, 191, 10301, 11311, 12421.

e.- Viết hàm đọc file Mang.txt và thực hiện:

- Đếm xem trong file có bao nhiêu số nguyên tố strobogrammatic?
- In các số nguyên tố strobogrammatic này ra màn hình.
- Tìm số nguyên tố strobogrammatic lớn nhất.

☞ Số strobogrammatic: Là một số có giá trị không đổi khi xoay số đó 180 độ (180°). Nhận xét: các số *strobogrammatic* chỉ chứa các số sau đây: 0, 1, 6, 8, 9. Ví dụ: 0, 1, 8, 11, 69, 88, 96, 101, 111, ...

(4)- Viết chương trình tạo một tập tin chứa các số nguyên có giá trị ngẫu nhiên không trùng nhau. Sắp xếp chúng theo thứ tự tăng dần và lưu trữ sang tập tin khác.

(5)- Viết chương trình thực hiện các yêu cầu sau:

a. Cho nhập n ( $1 < n \leq 500$ ), nếu sai yêu cầu nhập lại



- b. Tạo file input.txt chứa n số ngẫu nhiên, với giá trị các số ngẫu nhiên nằm trong khoảng từ -1000 đến +1000. Cấu trúc file input.txt gồm 2 dòng: dòng đầu chứa số n, dòng kế tiếp chứa n số ngẫu nhiên đã tạo được.
  - c. Viết hàm đọc file input.txt. Tìm xem các số may mắn có trong file này và ghi kết quả vào file LuckyNumber.txt. Cấu trúc file LuckyNumber.txt gồm 2 dòng: dòng đầu chứa số lượng số may mắn, dòng kế tiếp chứa các số may mắn có trong file input.txt.
- ☞ Số may mắn có thể là số âm hoặc dương và các số này chỉ chứa các ký số 6 hoặc ký số 8. Ví dụ số may mắn như 6, - 86, 68, -888, ...; số không may mắn như 7, - 96, 65, -848, ....

### 11.6.1.3. Mảng 2 chiều

- (6)- Cho file *Mang2Chieu.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 mảng hai chiều m dòng và n cột là các số nguyên.
  - a. Viết chương trình đọc file mảng hai chiều trên.
  - b. Xuất mảng hai chiều đó ra màn hình.
  - c. Tính tổng các số chẵn có trong mảng hai chiều.
  - d. Tìm phần tử lớn nhất có trong mảng hai chiều.
  - e. Đếm số phần tử lớn nhất có trong mảng hai chiều.
  - f. Sắp xếp mảng tăng dần.
  - g. Nếu mảng 2 chiều là ma trận vuông thì in các giá trị trên đường chéo chính và đường chéo phụ nối tiếp vào nội dung file *Mang2Chieu.txt* (ghi tiếp theo, không xóa nội dung cũ).
- (7)- 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.
- (8)- Viết chương trình gồm 2 hàm sau:
  - Hàm TaoFile:
    - Nhận tham số là số dòng, số cột cần có của 1 mảng hai chiều.
    - File cần tạo có tên MaTran.txt. Cấu trúc của file có dạng như sau:
      - Dòng 1: chứa số dòng của mảng hai chiều
      - Dòng 2: chứa số cột của mảng hai chiều
      - Dòng 3 đến dòng cuối: mỗi dòng chứa giá trị các phần tử có trên từng dòng của mảng hai chiều. 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.
- (9)- 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”.

(10)- Nhập một ma trận vuông nxn từ một tập tin văn bản MATRAN.TXT có nội dung như sau:

- Dòng đầu tiên chứa cấp n
- Mỗi dòng kế tiếp chứa n số tương ứng với mỗi dòng của ma trận.

Ví dụ nội dung tập tin MATRAN.TXT như sau:

```
3
5 8 2
4 7 6
8 0 0
```

**Yêu cầu:**

- Đọc dữ liệu từ tập tin vào một mảng trong bộ nhớ.
  - Tính tổng đường chéo chính.
  - Tính tổng đường chéo phụ.
  - Sắp xếp các phần tử trên đường chéo chính tăng dần từ trên xuống dưới.
  - Sắp xếp các phần tử trên đường chéo phụ giảm dần từ trên xuống dưới.
- (11)- Viết chương trình thực hiện các yêu cầu sau:

- Cho nhập n ( $1 < n \leq 20$ ), nếu sai yêu cầu nhập lại.

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

**input.txt**

|   |   |   |
|---|---|---|
| 3 |   |   |
|   |   | 9 |
|   | 4 |   |
| 8 |   |   |

**CheoPhu.out**

|    |    |    |    |
|----|----|----|----|
| 3  | 5  | 9  | 17 |
| 2  | 4  | 7  | 13 |
| 8  | 1  | 6  | 15 |
| 13 | 10 | 21 |    |

**TongDongCot.out**

- Tạo file *input.txt* chứa  $n \times n$  số ngẫu nhiên, với giá trị các số ngẫu nhiên đều là số dương (nằm trong khoảng từ 0 đến 400) và không trùng nhau. Cấu trúc file *input.txt* gồm  $n+1$  dòng: dòng đầu chứa số  $n$ ,  $n$  dòng kế tiếp mỗi dòng chứa  $n$  số ngẫu nhiên đã tạo được.
- Viết hàm đọc file *input.txt*. Tạo file *CheoPhu.out* như hình minh họa. Cấu trúc file *CheoPhu.out* gồm  $n+1$  dòng: dòng đầu chứa số  $n$ ,  $n$  dòng kế tiếp mỗi dòng chứa số nằm trên đường chéo phụ của ma trận  $n \times n$ , những số không nằm trên đường chéo phụ sẽ được in bằng khoảng trắng.
- Viết hàm đọc file *input.txt*. Thực hiện tính tổng các dòng, tổng các cột, sau đó lưu kết quả vào file *TongDongCot.out* như hình minh họa.

### 11.6.2. Tập tin nhị phân (Binary file)

- (12)- Tạo sẵn file chứa nội dung tùy ý. Viết hàm tìm và thay thế một chuỗi con bằng một chuỗi khác.
- (13)- 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.
- (14)- Viết chương trình nhập tên một tập tin văn bản và thống kê tần số xuất hiện của một chữ cái từ A đến Z trong văn bản. Khi đếm tần số xuất hiện, không phân biệt

chữ thường với chữ hoa (chẳng hạn a và A là như nhau). Kết quả thống kê ghi vào một tập tin văn bản trên đĩa, tần số xuất hiện của mỗi chữ cái được lưu trên một dòng của tập tin văn bản.

- (15)- Viết chương trình đếm số từ và số dòng trong một tập tin văn bản.
- (16)- Viết chương trình đọc một chuỗi tối đa 100 kí tự từ bàn phím. Lưu các ký tự là nguyên âm vào tập tin "NguyenAm.txt". Đọc các kí tự từ tập tin này và hiển thị lên màn hình console.
- (17)- Viết chương trình đọc và hiện một tập tin văn bản. Sau đó trình bày những thông kê sau: số ký tự, số từ, số dòng của nó.
- (18)- 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.
- (19)- Viết chương trình tính số lần xuất hiện ký tự chữ cái trong một tập tin văn bản.
- (20)- Viết chương trình tính số từ có trong một tập tin văn bản.
- (21)- Viết chương trình nối hai tập tin văn bản lại với nhau thành một tập tin mới.
- (22)- Viết chương trình lưu lại nội dung của màn hình vào tập tin SCREEN.DAT (tương tự lệnh COPY CON của DOS).
- (23)- Viết chương trình hiện chương trình được lưu trong SCREEN.DAT (tương tự lệnh TYPE của DOS).
- (24)- Tương tự bài trên nhưng những dòng trống sẽ được bỏ qua.
- (25)- Viết chương trình cho phép nhập từ bàn phím và ghi vào 1 tập tin tên HocSinh.txt với mỗi phần tử của tập tin là một cấu trúc bao gồm các trường:
  - Mã số học sinh (Ma, char [3]).
  - Họ tên học sinh (HoTen, char[30]).
  - Điểm lý thuyết (lt, float).
  - Điểm thực hành (th, float).
  - Điểm trung bình (dtb float). Biết  $dtb = (lt * 60\%) + (th * 40\%)$

**Yêu cầu cài đặt:**

- Viết hàm cho nhập số nguyên dương n.
  - Viết hàm cho nhập thông tin của n học sinh và ghi vào file HocSinh.txt.
  - Viết hàm in ra danh sách các học sinh có điểm trung bình dưới 5.
  - Viết hàm xuất các học sinh có trong file HocSinh.txt ra màn hình theo thứ tự giảm dần của điểm trung bình.
- (26)- Để quản lý được hàng hóa, người ta cần biết được các thông tin sau về hàng hóa:
- Mã hàng (mh, char[5]).
  - Số lượng (sl, int).
  - Đơn giá (dg, float).
  - Số tiền (st, float). Biết số tiền = số lượng \* đơn giá

**Yêu cầu cài đặt:**

- Viết hàm cho nhập các hàng hóa cần ghi vào file DSHH.TXT. Kết thúc việc nhập bằng cách đánh ENTER vào mã hàng. Sau khi nhập xong yêu cầu in toàn bộ danh sách hàng hóa ra màn hình.
- Viết hàm xuất các hàng hóa có trong file DSHH.TXT ra màn hình.
- Viết hàm nhận tham số là mã hàng hóa (mh) cần tìm. Nếu tìm thấy mã mh sẽ in thông tin của hàng hóa có mã mh ra màn hình; ngược lại in ra “Không tìm thấy”.

(27)- Lập trình nhập vào từ bàn phím các thông tin liên quan đến việc tiêu thụ điện của khách hàng như sau: mã số điện kế, họ tên chủ hộ, địa chỉ, tháng, năm, chỉ số điện kế đầu tháng, chỉ số điện kế cuối tháng.

a. Thực hiện việc nhập các thông tin trên cho đến khi mã số điện kế bằng rỗng thì dừng quá trình nhập. Ghi dữ liệu vừa nhập vào tập tin có tên TIENDIEN.DAT (tạo tập tin mới).

b. Nhập từ bàn phím các thông tin: mã số điện kế, tháng, năm. Tìm trong tập tin đã lưu và in ra màn hình một phiếu tiêu thụ điện của khách hàng như sau:

PHIẾU THU TIỀN ĐIỆN

Mã số điện kế: .....  
Họ tên chủ hộ: .....  
Địa chỉ: .....  
Tháng năm: .....  
Chỉ số điện kế đầu tháng: .....  
Chỉ số điện kế cuối tháng: .....

---

## **TÀI LIỆU THAM KHẢO**

- [1]. Hoàng Kiêm – Giải một bài toán trên máy tính như thế nào – Tập 1 & Tập 2- Nhà Xuất Bản Giáo Dục -2001
- [2]. Trần Đan Thư – Giáo trình lập trình C – Tập 1& Tập 2 – Nhà Xuất Bản Đại Học Quốc Gia – 2001
- [3]. 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
- [4]. Nguyễn Thanh Hùng – Các bài tập tin học chọn lọc – Nhà Xuất Bản Giáo Dục – 1996.
- [5]. Nguyễn Tiến Huy – Bài giảng Kỹ thuật lập trình – 2003.
- [6]. Phạm Văn Ất, Kỹ thuật Lập trình C- cơ bản và nâng cao, Nhà Xuất Bản Khoa Học Và Kỹ Thuật - 2003.
- [7]. Dương Anh Đức – Trần Hạnh Nhi – Giáo trình cấu trúc dữ liệu - Trường Đại Học Khoa Học Tự Nhiên Tp.Hồ Chí Minh – 2003
- [8]. Brain W. Kernighan & Dennis Ritchie, The C Programming Language, Prentice Hall Publisher, 1988.