

**CHƯƠNG I****ĐẠI CƯƠNG VỀ LẬP TRÌNH****I. KHÁI NIỆM THUẬT TOÁN:****I.1. Khái niệm:**

Thuật toán là tập hợp các quy tắc có logic nhằm giải một lớp bài toán nào đó để được một kết quả xác định.

**I.2. Các tính chất đặc trưng của thuật toán :****I.2.1. Tính tổng quát :**

Thuật toán được lập không phải chỉ để giải một bài toán cụ thể mà thôi mà còn phải giải được một lớp các bài toán có dạng tương tự.

**I.2.2. Tính giới hạn :**

Thuật toán giải một bài toán phải được thực hiện qua một số giới hạn các thao tác để đạt đến kết quả.

**I.2.3. Tính duy nhất :**

Toàn bộ quá trình biến đổi, cũng như trật tự thực hiện phải được xác định và là duy nhất. Như vậy khi dùng thuật toán cùng một dữ liệu ban đầu phải cho cùng một kết quả.

**I.3. Phân loại:**

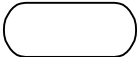
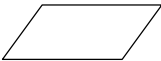
Theo cấu trúc, ta có thể phân thành ba loại thuật toán cơ bản sau :


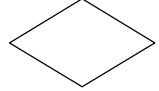
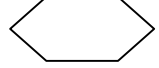
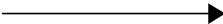
- Thuật toán không phân nhánh.
- Thuật toán có phân nhánh.
- Thuật toán theo chu trình có bước lặp xác định và có bước lặp không xác định.

**II. MÔ TẢ THUẬT TOÁN BẰNG LƯU ĐỒ :****II.1. Lưu đồ :**

Lưu đồ là một dạng đồ thị dùng để mô tả quá trình tính toán một cách có hệ thống. Người ta thường thể hiện thuật toán bằng lưu đồ.

**II.2. Các ký hiệu trên lưu đồ :**

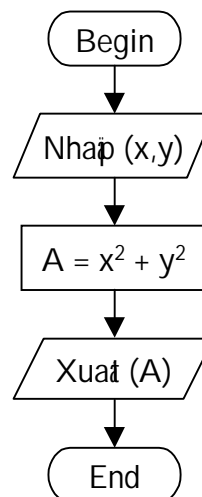
Tên khối	Ký hiệu	Ý nghĩa
Khối mở đầu hoặc kết thúc		Dùng mở đầu hoặc kết thúc chương trình
Khối vào ra		Đưa số liệu vào và ra hoặc in kết quả

Khối tính toán		Biểu diễn các công thức tính toán và thay đổi giá trị của các biến
Khối điều kiện		Dùng để phân nhánh chương trình
Chương trình con		Dùng để gọi chương trình con
Mũi tên		Chỉ hướng truyền thông tin, liên hệ các khối

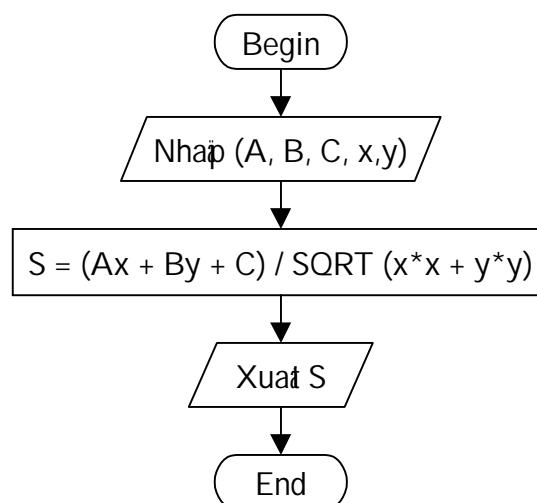
### II.3. Một số ví dụ biểu diễn thuật toán bằng lưu đồ

#### II.3.1. Thuật toán không phân nhánh:

Ví dụ 1: Tính  $A = x^2 + y^2$

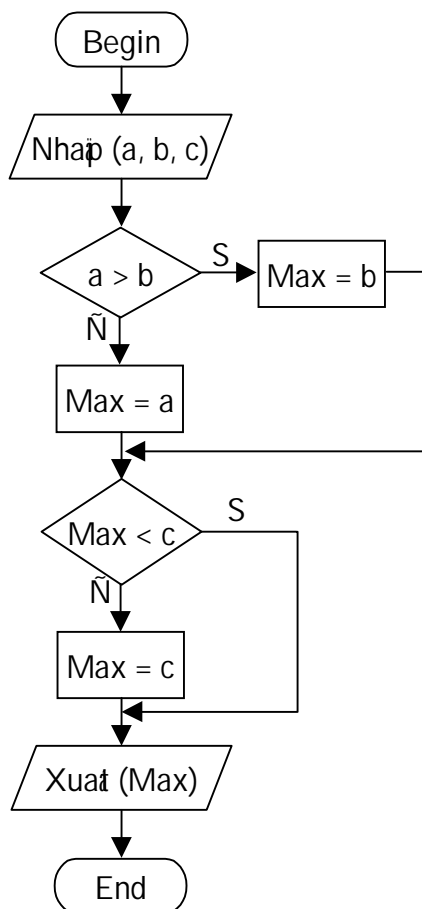


Ví dụ 2: Tính  $S = \frac{Ax + By + C}{\sqrt{x^2 + y^2}}$ ; biết A, B, C, x, y

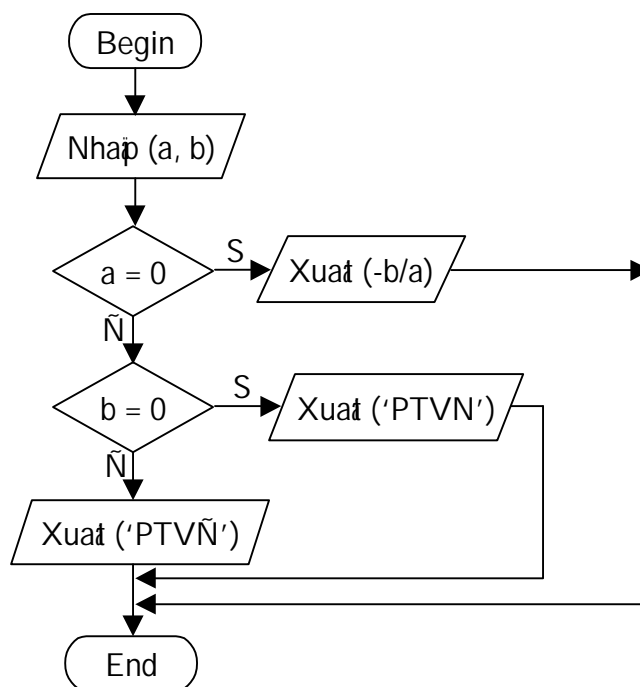


**II.3.2. Thuật toán có phân nhánh:**

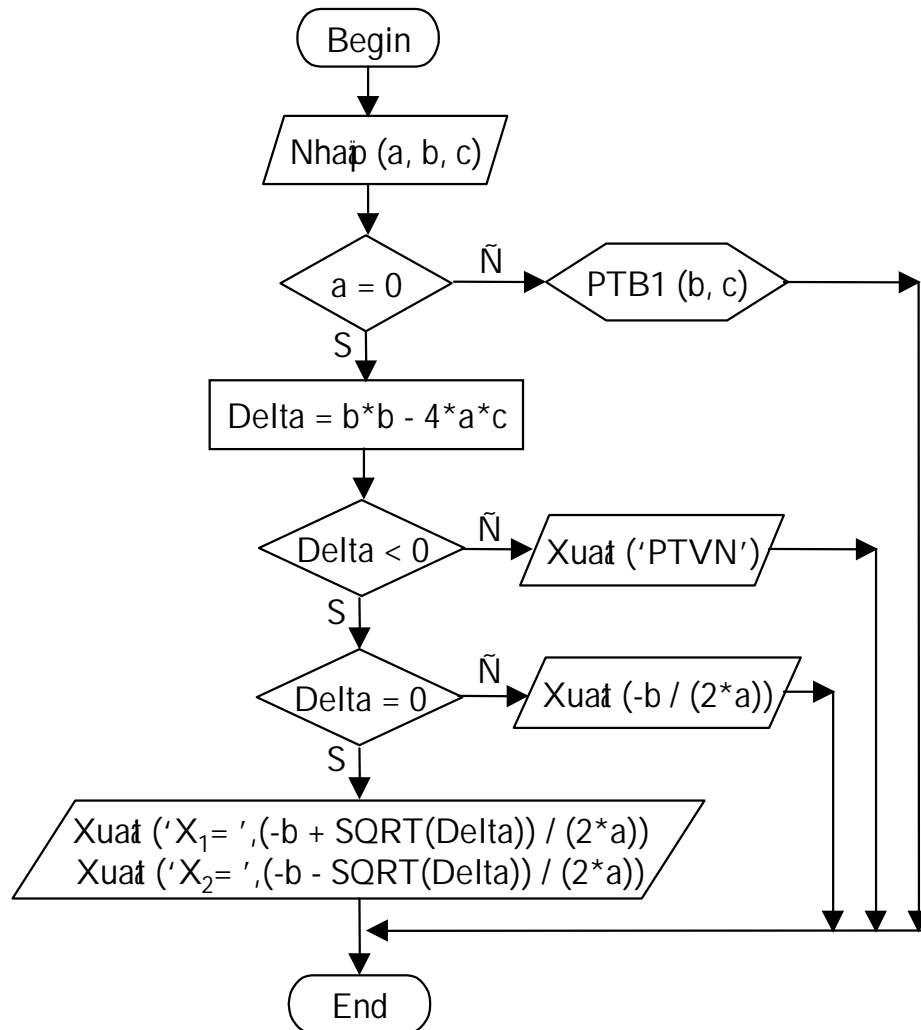
Ví dụ 1: Tìm giá trị max của ba số thực a, b, c



Ví dụ 2: Giải phương trình bậc nhất  $Ax+B=0$  với các nghiệm thực.

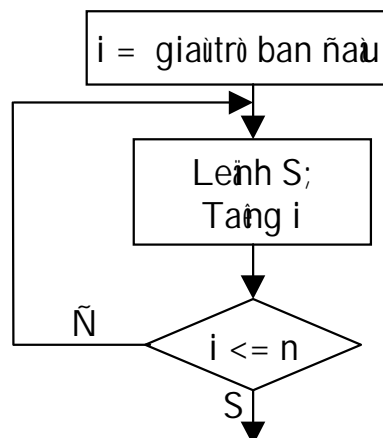


**Ví dụ 3 :** Giải phương trình bậc hai  $Ax^2+Bx+C=0$  với các nghiệm thực.



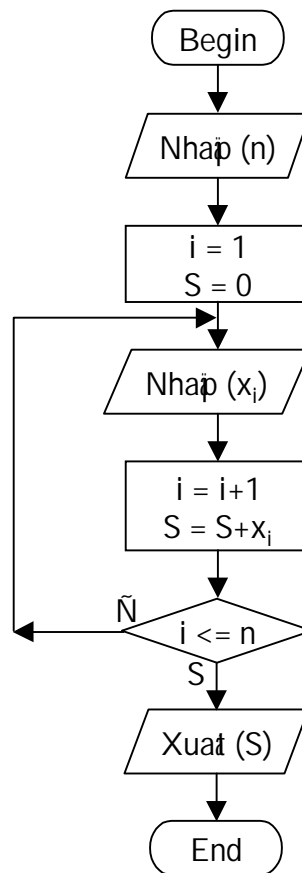
### II.3.3. Thuật toán có chu trình:

Thuật toán có chu trình với các bước lặp xác định thường được thể hiện bằng lưu đồ sau :



với  $n$  là giá trị kết thúc.

Ví dụ 4: Tính  $S = \sum_{i=1}^n x_i$ , với các  $x_i$  do ta nhập vào.



### III. CÁC NGÔN NGỮ LẬP TRÌNH & CHƯƠNG TRÌNH DỊCH:

#### III.1. Ngôn ngữ lập trình nh:

**III.1.1. Giới thiệu:** Con người muốn giao tiếp với máy tính phải thông qua ngôn ngữ. Con người muốn máy tính thực hiện công việc, phải viết các yêu cầu đưa cho máy bằng ngôn ngữ máy hiểu được. Việc viết các yêu cầu ta gọi là lập trình (programming). Ngôn ngữ dùng để lập trình được gọi là ngôn ngữ lập trình.

Nếu ngôn ngữ lập trình gần với vấn đề cần giải quyết, gần với ngôn ngữ tự nhiên thì việc lập trình sẽ đơn giản hơn nhiều. Những ngôn ngữ lập trình có tính chất như trên được gọi là ngôn ngữ cấp cao. Nhưng máy tính chỉ hiểu được ngôn ngữ riêng của mình, đó là các chuỗi số 0 với 1 và như vậy rõ ràng là khó khăn cho lập trình viên, vì nó không gần gũi với con người.

Hiện tại, ngôn ngữ lập trình được chia ra làm các loại sau:

#### **III.1.2. Phân loại ngôn ngữ lập trình nh:**

- Ngôn ngữ máy (machine language)

- Hợp ngữ (assembly language)
- Ngôn ngữ cấp cao (higher-level language)

Do máy tính chỉ hiểu được ngôn ngữ máy, cho nên một chương trình viết trong ngôn ngữ cấp cao phải được biên dịch sang ngôn ngữ máy. Công cụ thực hiện việc biên dịch đó được gọi là chương trình dịch.

### III.2. Chương trình dịch:

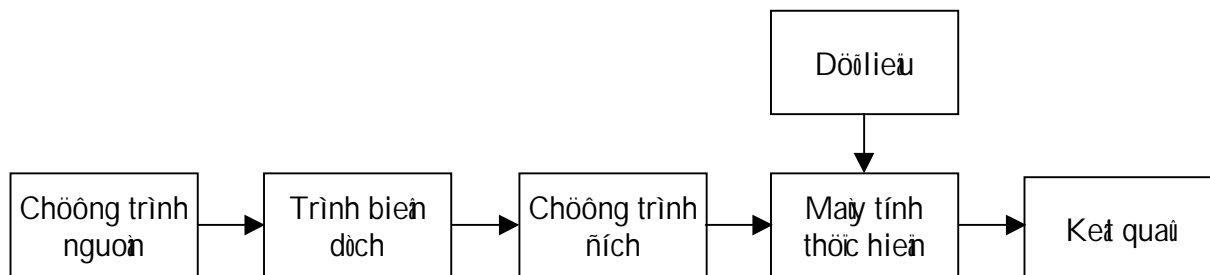
Chương trình dịch được chia ra làm 2 loại: trình biên dịch (compiler) và trình thông dịch (interpreter)

**III.2.1. Trình biên dịch:** là việc chuyển một chương trình trong ngôn ngữ cấp cao nào đó (chương trình nguồn) sang ngôn ngữ máy (chương trình đích).

- Thời gian chuyển một chương trình nguồn sang chương trình đích được gọi là thời gian dịch.

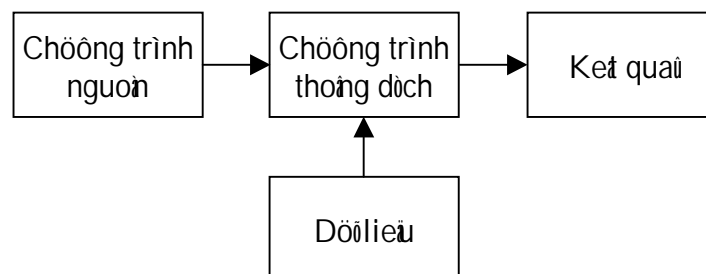
- Thời gian mà chương trình đích thực thi được gọi là thời gian thực thi.

Như vậy, chương trình nguồn và dữ liệu để chương trình thực thi được xử lý trong các thời điểm khác nhau, được gọi là thời gian dịch (compile time) và thời gian thực thi (run-time)



Hình I.1. Chương trình thực thi theo cơ chế dịch của trình biên dịch

**III.2.2. Trình thông dịch:** quá trình dịch và thực thi xảy ra cùng 1 thời gian, dịch đến đâu thì hành lệnh đến đó.



Hình I.2. Chương trình thực thi theo cơ chế dịch của trình thông dịch

## CHƯƠNG 2                      LÀM QUEN VỚI NGÔN NGỮ C

---

### **\* GIỚI THIỆU NGÔN NGỮ C**

Ngôn ngữ C do Dennis Ritchie là người đầu tiên đề xuất, đã thiết kế và cài đặt C trong môi trường UNIX. Nó có nguồn gốc từ ngôn ngữ BCPL do Martin Richards đưa ra vào năm 1967 và ngôn ngữ B do Ken Thompson phát triển từ ngôn ngữ BCPL năm 1970 khi viết hệ điều hành Unix.

C là ngôn ngữ lập trình đa dụng, cấp cao nhưng lại có khả năng thực hiện các thao tác như của ngôn ngữ Assembly. Vì thế ngôn ngữ C nhanh chóng được cài đặt, sử dụng trên máy vi tính và đã trở thành một công cụ lập trình khá mạnh, hiện nay đang có khuynh hướng trở thành một ngôn ngữ lập trình chính cho máy vi tính trên thế giới.

### **\* Đặc điểm ngôn ngữ C**

Ngôn ngữ C có những đặc điểm cơ bản sau :

- Tính cô đọng (compact) : Ngôn ngữ C chỉ có 32 từ khoá chuẩn, 40 toán tử chuẩn mà hầu hết được biểu diễn bởi các dãy ký tự ngắn gọn.

- Tính cấu trúc (structured) : Ngôn ngữ C có một tập hợp các phát biểu lập trình cấu trúc như phát biểu quyết định hoặc lặp. Do đó, nó cho phép chúng ta viết chương trình có tổ chức và dễ hiểu.

- Tính tương thích (compactable) : Ngôn ngữ C có bộ lệnh tiền xử lý và các thư viện chuẩn làm cho các chương trình viết bằng ngôn ngữ C có thể tương thích khi chuyển từ máy tính này sang máy tính kiểu hoàn toàn khác.

- Tính linh động (flexible) : Ngôn ngữ C là một ngôn ngữ rất linh động về ngữ pháp, nó có thể chấp nhận rất nhiều cách thể hiện mà không có ở ngôn ngữ khác như Pascal, nó giúp cho kí ch thước mã lệnh có thể thu gọn lại để chương trình thực thi nhanh chóng hơn.

- Biên dịch : Ngôn ngữ C được biên dịch bằng nhiều bước và cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và nối các đối tượng đó lại với nhau (link) thành một chương trình thực thi thống nhất.

## **I. CÁC KHÁI NIỆM CƠ BẢN**

### **I.1. Cấu trúc cơ bản của một chương trình nh C**

[tiền xử lý]

[Các hàm]

main()

```
{    [khai báo biến;]
    [nhập dữ liệu ;]
    [xử lý ;]
    [xuất ;]
}
```

Ví dụ : Chương trình hiển thị trên màn hình câu “Chao các bạn”

```
void main()
{ printf(“Chao cac ban\n”);
}
```

### Một vài nhận xét quan trọng :

- Chương trình C bao giờ cũng có một hay nhiều hàm, trong đó có một hàm chính bắt buộc phải có là hàm **main()**. Đây chính là hàm được thực hiện đầu tiên trong chương trình.

- Cặp dấu “{ }” để xác định một khối lệnh.

- Hàm printf(“Chao cac ban\n”) là hàm chuẩn của C dùng để xuất câu thông báo “Chao cac ban” ra màn hình. Ký tự “\n” là ký tự đặc biệt dùng để xuống dòng.

- Dấu “;” để chấm dứt một lệnh.

- Chương trình C có phân biệt chữ thường với chữ hoa. Đa số các từ khóa của C được viết bằng chữ thường, còn một số ít được viết bằng chữ hoa mà ta phải tuân thủ chặt chẽ, nếu không thì chương trình dịch sẽ không hiểu.

### \* Một vài ví dụ

Ví dụ 1: In bảng lũy thừa 2 của các số nguyên từ 10 đến 50

/\* Chương trình in bình phương các số từ 10 đến 50 \*/

#include <stdio.h>

void main()

```
{int n;                                /* Khai báo biến n kiểu nguyên */
    n=10;                              /* Gán n=10 */
    while (n<=50)                      /* Lặp từ 10 đến 50 bằng while */
    { printf(“%3d\t %5d\n”,n,n*n);      /* in dạng 5d là dành 5 vị trí để in n và n² */
        n++;                          /* Tăng n lên 1 */
    }                                  /* Hết while */
}
```



Ví dụ 2 : Tương tự như ví dụ 1 nhưng viết cách khác :

```
#include <stdio.h>
#define max 50 /*Tiền xử lý, định nghĩa a max =50*/
void main()
{ int n; /*Khai báo biến kiểu nguyên*/
  for (n=10; n<=max; n++) /*Lặp từ 10 đến 50 bằng for*/
    printf(“%3d\t %5d\n”,n,n*n); /*in n và n2 dạng 5d là năm chữ số*/
} /*Hết main*/
```

Ví dụ 3 : Chương trình in lũy thừa 2, 3, 4, 5; có dùng hàm để tính lũy thừa :

```
#include <stdio.h>
#define max 50 /*Tiền xử lý, định nghĩa a max =50*/
float luythua(int n, int m) /*Hàm luythua với 2 thông số*/
{ float s=1; /*Khai báo và khởi tạo biến s*/
  for ( ;m>0;m--) /*Lặp giảm dần từ m tới 1*/
    s=s*n;
  return s; /*Trả kết quả về*/
}
void main()
{ int n,n2,n3,n4,n5; /*Khai báo biến kiểu nguyên*/
  for (n=10;n<=50;n++) /*Lặp từ 10 đến 50 bằng for*/
  { n2= luythua(n,2); /*Gọi hàm luythua*/
    n3= luythua(n,3);
    n4= luythua(n,4);
    n5= luythua(n,5);
    printf(“%3d\t %5.2f\t %5.2f\t %5.2f\t %5.2f\n”,
n,n2,n3,n4,n5); /*in n và nm dạng 5 chữ số với 2 số lẻ */
  }
} /*Hết main*/
```

### **\* Hàm xuất chuẩn printf()**

#### **Cú pháp :**

**printf**(“chuỗi-định dạng”,thamso1,thamso2,...)

#### **Ý nghĩa :**

Hàm printf() sẽ xem xét chuỗi-định dạng, lấy giá trị các tham số (nếu cần) để đặt vào theo yêu cầu của chuỗi-định dạng và gọi ra thiết bị chuẩn.

Chuỗi-định dạng là một chuỗi ký tự, trong đó có những ký tự xuất ra nguyên vẹn hoặc xuất ở dạng đặc biệt, và có thể có những chuỗi điều khiển cần lấy giá trị của các tham số để thay vào đó khi in ra.

**- Những ký tự đặc biệt :**

Ký tự	Tác dụng	Mã ASCII
\n	Xuống hàng mới	10
\t	Tab	9
\b	Xóa ký tự bên trái	8
\r	Con trỏ trở về đầu hàng	13
\f	Sang trang	12
\a	Phát tiếng còi	7
\\	Xuất dấu chéo ngược	92
\'	Xuất dấu nháy đơn ‘	39
\''	Xuất dấu nháy kép “	34
\xdd	Xuất ký tự có mã ASCII dạng Hex là dd	
\ddd	Xuất ký tự có mã ASCII dạng Dec là ddd	
\0	Ký tự NULL	0

**- Chuỗi định dạng :**

% [ flag ][ width ][ .prec ][ F | N | h | l ] type

**Type :** định kiểu của tham số theo sau chuỗi-định dạng để lấy giá trị ra

Type	Ý nghĩa
d,i	Số nguyên cơ số 10
u	Số nguyên cơ số 10 không dấu
o	Số nguyên cơ số 8
x	Số nguyên cơ số 16, chữ thường(a,b,...,f)
X	Số nguyên cơ số 16, chữ in (A,B,...,F)
f	Số thực dạng [-]dddd.ddd...
e	Số thực dạng [-]d.ddd e[+/-]ddd
E	Số thực dạng [-]d.ddd E[+/-]ddd
g,G	Số thực dạng e(E) hay f tùy theo độ chính xác
c	Ký tự
s	Chuỗi ký tự tận cùng bằng '\0'
%	Dấu % cần in

**Flag** : Dạng điều khiển

Flag	Ý nghĩa
nếu không có	in dữ liệu ra với canh phải
-	in dữ liệu ra với canh trái
+	Luôn bắt đầu số bằng + hay -
#	in ra tùy theo type, nếu: 0 : Chèn thêm 0 đứng trước giá trị >0 x,X : Chèn thêm 0x hay 0X đứng trước số này e,E,f : Luôn luôn có dấu chấm thập phân G,g : Như trên nhưng không có số 0 đi sau

**Width** : định kích thước in ra

Width	Ý nghĩa
n	Dành ít nhất n ký tự, điền khoảng trắng các ký tự còn trống
0n	Dành ít nhất n ký tự, điền số 0 các ký tự còn trống
*	Số ký tự ít nhất cần in nằm ở tham số tương ứng

**Prec** : định kích thước phần lẻ in ra

Prec	Ý nghĩa
không có	độ chính xác như bình thường
0	d,i,o,u,x độ chính xác như cũ e,E,f Không có dấu chấm thập phân
n	nhiều nhất là n ký tự (số)
*	Số ký tự ít nhất cần in nằm ở tham số tương ứng

**Các chữ bổ sung** :

F	Tham số là con trỏ xa XXXX:YYYY
N	Tham số là con trỏ gần YYYY
h	Tham số là short int
l	Tham số là long int (d,i,o,u,x,X) double (e,E,f,g,G)

Ví dụ 1: char c='A';

char s[]="Blue moon!";



## I.2. Kiểu dữ liệu cơ bản

### I.2.1. Định nghĩa:

Kiểu dữ liệu cơ bản là kiểu dữ liệu có giá trị đơn, không phân chia được nữa như số, ký tự

### I.2.2. Phân loại:

Tên kiểu	Ý nghĩa	Kích thước	Phạm vi
char	Ký tự	1 byte	-128→127
unsigned char	Ký tự không dấu	1 byte	0→255
unsigned short	Số nguyên ngắn không dấu	2 bytes	0→65535
enum	Số nguyên có dấu	2 bytes	-32768→32767
short int	Số nguyên có dấu	2 bytes	-32768→32767
int	Số nguyên có dấu	2 bytes	-32768→32767
unsigned int	Số nguyên không dấu	2 bytes	0 → 65535
long	Số nguyên dài có dấu	4 bytes	-2147483648 → 2147483647
unsigned long	Số nguyên dài không dấu	4 bytes	0→4294967295
float	Số thực độ chính xác đơn	4 bytes	3.4 E-38→3.4 E+38
double	Số thực độ chính xác kép	8 bytes	1.7 E-308 → 1.7 E+308
long double	Số thực độ chính xác hơn double	10 bytes	3.4 E-4932 → 1.1 E+4932

### Chú ý :

- Ngôn ngữ C không có kiểu logic (boolean như Pascal) mà quan niệm 0 là false ; Khác 0 là true
- Ngôn ngữ C không có kiểu chuỗi như kiểu string trong Pascal
- Các kiểu đồng nhất:
  - int = short int = short = signed int = signed short int
  - long int = long
  - signed long int = long
  - unsigned int = unsigned = unsigned short = unsigned short int
  - unsigned long int = unsigned long

### I.3. **Biến**

**I.3.1. Tên biến :** Tên biến là một chuỗi ký tự bắt đầu bằng ký tự chữ, ký tự kế tiếp là ký tự chữ (dấu gạch dưới “\_” được xem là ký tự chữ) hoặc số và không được trùng với các từ khóa của C.

**Chú ý :** - Ngôn ngữ C phân biệt chữ thường với chữ hoa nên biến chữ thường với chữ hoa là khác nhau.

**Ví dụ :** Bien\_1      \_bien2      là hợp lệ  
bi&en      2a      a b      là không hợp lệ

- Ngôn ngữ C chỉ phân biệt hai tên hợp lệ với nhau bằng n ký tự đầu tiên của chúng. Thông thường n=8, nhưng hiện nay nhiều chương trình nh dịch cho phép n=32, như Turbo C cho phép thay đổi số ký tự phân biệt từ 8-32)

**Ví dụ :** Hai biến sau bị xem là cùng tên  
bien\_ten\_dai\_hon\_32\_ky\_tu\_dau\_tien\_1  
bien\_ten\_dai\_hon\_32\_ky\_tu\_dau\_tien\_2

#### I.3.2. ***Khai báo biến***

Các biến phải được khai báo trước khi sử dụng nhằm giúp cho chương trình nh dịch có thể xử lý chúng.

Khai báo biến có dạng :

Kiểu dữ liệu      tên biến n1 [,tenbiến2 ...] ;
---

**Ví dụ :** int      a,b,c;  
float   x,y,delta;  
char   c;

\* **Khai báo và khởi tạo biến:**

Kiểu dữ liệu      tên biến = giá trị ;
--

#### I.3.3. ***Hàm nhập dữ liệu chuẩn***

a) ***Hàm scanf()***

**Cú pháp:**    scanf(“chuỗi-định dạng“, địa chỉ tham số 1, địa chỉ tham số 2,...)

- Chuỗi-định dạng của scanf() gồm có ba loại ký tự :

- + Chuỗi điều khiển
- + Ký tự trắng
- + Ký tự khác trắng

↪ Chuỗi điều khiển có dạng :

%[width][h/l] type

Với **type**: xác định kiểu của biến địa chỉ tham số sẽ nhận giá trị nhập vào

Type	Ý nghĩa
d,i	Số nguyên cơ số 10 (int)
o	Số nguyên cơ số 8 (int)
u	Số nguyên cơ số 10 không dấu (unsigned)
x	Số nguyên cơ số 16 (int)
f,e	Số thực (float)
c	Ký tự (char)
s	Chuỗi ký tự
p	Con trỏ (pointer)
lf	Số thực (double)
Lf	Số thực (long double)

**Width** : xác định số ký tự tối đa sẽ nhận vào cho vùng đó.

Hàm scanf() chỉ nhận cho đủ width ký tự hoặc cho đến khi gặp ký tự trắng đầu tiên. Nếu chuỗi nhập vào nhiều hơn thì phần còn lại sẽ dành lại cho lần gọi scanf() kế tiếp.

Ví dụ 1: scanf(“%3s”,str);

Nếu nhập chuỗi ABCDEFG ↵

thì scanf() sẽ nhận tối đa 3 ký tự cất vào mảng str, còn DEFG sẽ được lấy nếu sau đó có lần gọi scanf(“%s”,str) khác.

Ví dụ 2: unsigned long money;

scanf(“%lu",&money);

Lưu ý: Nếu scanf(“%ul”, &money) thì giá trị nhập vào sẽ không được lưu trữ trong biến money, nhưng chương trình dịch không báo lỗi.

Ví dụ 3: Nhập vào tên và bị giới hạn trong khoảng [A-Z,a-z]

```
char name[20];
printf(“Name : ”);
scanf(“%[A-Za-z]",&name);
```

Trong trường hợp này, nếu ta gõ sai dạng thì name = ""

☞ Ký tự trắng: nếu có trong chuỗi-dạng sẽ yêu cầu scanf() bỏ qua một hay nhiều ký tự trắng trong chuỗi nhập vào. Ký tự trắng là ký tự khoảng trắng (' '), tab ('\t'), xuống hàng ('\n'). Một ký tự trắng trong chuỗi-định dạng sẽ được hiểu là chờ nhập đến ký tự khác trắng tiếp theo.

Ví dụ 4: scanf("%d",&num);

Hàm scanf() cho ta nhập một ký tự khác trống nữa thì mới thoát ra. Ký tự đó sẽ nằm trong vùng đệm và sẽ được lấy bởi hàm scanf() hoặc gets() tiếp theo.

↪ Ký tự khác trống: nếu có trong chuỗi-định dạng sẽ khiến cho scanf() nhận vào đúng ký tự như thế.

Ví dụ 5: scanf("%d/%d/%d",&d,&m,&y);

Hàm scanf() chờ nhận một số nguyên, cắt và bỏ d, kế đến là dấu '/', bỏ dấu này đi và chờ nhận số nguyên kế tiếp để cắt và bỏ m. Nếu không gặp dấu '/' kế tiếp số nguyên thì scanf() chấm dứt.

Chú ý: Hàm scanf() đòi hỏi các tham số phải là các địa chỉ của các biến hoặc là một con trỏ.

\* Toán tử địa chỉ &: Lấy địa chỉ của một biến

Ví dụ 6:     int n; → biến n  
                  &n; → địa chỉ của n  
                  printf("trị = %d, địa chỉ = %d",n,&n);

b) Hàm getch():

Hàm getch() dùng để nhận một ký tự do ta nhập trên bàn phím mà không cần gõ Enter với cú pháp:

ch = getch();	Không hiện ký tự nhập trên màn hình
ch = getche();	Hiện ký tự nhập trên màn hình

Với ch là biến kiểu char.

Ví dụ 7:

```
void main()
{
    char ch;
    printf("Go vao ky tu bat ky : ");
    ch = getche();
    printf("\n Ban vua go %c",ch);
    getch();
}
```

Ví dụ 8: Bạn nhập vào 1 chữ cái. Nếu chữ cái nhập vào là 'd' thì chương trình sẽ kết thúc, ngược lại chương trình sẽ báo lỗi và bắt nhập lại.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char ch;
```



```

printf("\nBan nhap vao 1 chu cai tu a den e: ");
while ((ch=getche()) != 'd')
{ printf("\nXin loi, %c la sai roi",ch);
  printf("\n Thu lai lan nua.\n");
}
}

```

Lưu ý: Hàm getch() còn cho phép ta nhập vào 1 ký tự mở rộng như các phím F1, F2,..., các phím di chuyển cursor. Các phím này luôn có 2 bytes: byte thứ nhất bằng 0, còn byte 2 là mã scancode của phím đó. Để nhận biết ta đã gõ phím ký tự hay phím mở rộng, ta có chương trình nh sau:

```

void main()
{
  int c;
  int extended = 0;
  c = getch();
  if (!c)
    extended = getch();
  if (extended)
    printf("The character is extended\n");
  else
    printf("The character isn't extended\n");
}

```

Phím	Mã scancode
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64
F7	65
F8	66
F9	67
F10	68
Home	71
↑	72
↓	80
←	75

→	77
PgUp	73
PgDn	81
End	79
Ins	82
Del	83

Bảng mã scancode của các phím mở rộng

c. Hàm `kbhit()`: Hàm `int kbhit()` sẽ kiểm tra xem có phím nào được gõ vào hay không. Nếu có, hàm `kbhit` sẽ trả về một số nguyên khác 0, và ngược lại.

Ký tự mà ta nhập vào qua hàm `kbhit()` có thể lấy được qua hàm `getch()` hoặc `getche()`.

Ví dụ:

```
void main()
{
    printf("Press any key to continue:");
    while (!kbhit()) /* do nothing */ ;
    char kytu=getch();
    printf("\nKy tu vua an : %c",kytu);
}
```

**I.4 Hằng:** Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình chương trình thực hiện.

#### I.4.1. Phân loại :

a. **Hằng số** : là các giá trị số đã xác định và không đổi.

	int	unsigned	long	hệ 8	hệ 16	float/double
Dạng	nnnn -nnnn	nnnnU/u	nnnnL/l -nnnnl/L	0nnnn	0xnnnn	nnnn.nnnn nnnn.nnnE/e±nnn
Ví dụ	4567 -12	123U 12uL	456789L -1234L	0345	0x1AB	123.654 123.234E-4

Chú ý :

- Các hằng số viết không dấu hoặc không số mũ được hiểu là số nguyên, ngược lại là double.
- Các hằng số nguyên lớn hơn int sẽ được lưu trữ theo kiểu long, còn lớn hơn long thì được lưu trữ theo kiểu double.
- Các hằng số nguyên dương lớn hơn long sẽ được lưu trữ theo kiểu double
- Một hằng số được lưu trữ theo dạng long nếu theo số đó có ký tự l (L),

dạng unsigned nếu sau đó có chữ u (U), dạng thập lục phân nếu trước số đó có 0x và dạng bát phân nếu trước số đó có 0

Ví dụ: 50000; 10 L; → Long  
 5U, 100u → unsigned  
 0x10 → hệ 16 = 16<sub>10</sub>  
 010 → hệ 8 = 8<sub>10</sub>

b. *Hằng ký tự*: là ký tự riêng biệt được viết trong hai dấu nháy đơn: 'A'

Giá trị của hằng ký tự là mã ASCII của nó.

Ví dụ: printf("%c có giá trị là %d", 'A', 'A');

→ 'A' có giá trị là 65

▪ Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác.

Ví dụ: '9'-'0'=57-48=9

▪ Hằng ký tự có thể là các ký tự đặc biệt dạng '\c<sub>1</sub>' mà ta đã xét ở hàm printf() như '\n', '\a', '\t' ...

c. *Hằng chuỗi*: Là một chuỗi ký tự nằm trong hai dấu nháy kép "".

Ví dụ: "Day la mot chuoi"  
 "Hang chuoi co ky tu đặc biệt như \n \248"  
 "" → chuỗi rỗng.

Chú ý:

- Phân biệt "A" ≠ 'A'

Hằng: Chuỗi Ký tự

Dạng lưu trữ:

A \0
------

A
---

- Nhận xét: ở dạng lưu trữ, ta thấy tận cùng của chuỗi có ký tự NULL '\0' mà không có ở dạng ký tự. Chính vì vậy mà không có ký tự rỗng ''.

- Một chuỗi có thể được viết trên nhiều hàng với điều kiện hàng trên phải có dấu '\n'.

Ví dụ: "Day la mot chuoi duoc viet tren \n  
 nhieu hang \n"

d. *Hằng biểu thức*: Là một biểu thức mà trong đó các toán hạng đều là các hằng. Khi đó chương trình dịch sẽ tính toán biểu thức trước, và kết quả được lưu trữ bằng một hằng số tương đương.

Ví dụ: 8\*20-13 → kết quả lưu trữ là 173

‘a -’A’ → “ là 97-65 = 32  
 1<8 → “ là 0 (sai)

#### I.4.2. Khai báo hằng:

**Cú pháp:** const tên\_hằng = biểu\_thức;

Ví dụ : const MAX = 50;

const PI = 3.141593;

**Chú ý:** - Ta có thể khai báo hằng bằng cách định nghĩa 1 macro như sau:

#define tên\_hằng giá\_trị

- Lệnh #define phải được khai báo ngoài hàm và sau nó không có dấu ;

### I.5. Phép toán

#### I.5.1. Phép gán:

**Cú pháp:** biến = biểu\_thức;

**Chú ý:** Phép gán trong ngôn ngữ C trả về một kết quả là trị của biểu thức

Ví dụ 1 : c = 10;

a = b = c;

printf(“a=%d , b=%d”,a,b); → a=10,b=10

Ví dụ 2 : x = b + 2\*c; ⇔ y = a + (x = b + 2\*c)

y = a + x;

Ví dụ 3 : (n+3) = 4+z; (không hợp lệ vì bên trái là biểu thức)

‘= c +’o’; (không hợp lệ vì bên trái là hằng)

#### I.5.2. Các phép toán số học :

a. Phép toán hai toán hạng : +, -, \*, /, %

Phép toán	Kiểu toán hạng	Kiểu kết quả
+, -, *	char, int, long, float, double	Kiểu của toán hạng có kiểu cao nhất
/	nguyên/nguyên	Kiểu nguyên và là phép chia nguyên
	thực(nguyên)/thực(nguyên)	Kiểu thực và là phép chia thực
%	nguyên/nguyên	Kiểu nguyên và là phép chia lấy phần dư

Ví dụ :

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

```
void main()
```

```

{ char cv;
  int iv = 121;
  float fv1, fv2;
  printf(" Chuyển kiểu : \n\n");
  cv = iv;
  printf("int được gán cho char : %d → %d (%c)\n\n", iv, cv, cv);
  fv1 = iv/50;
  printf(" int : %d / 50 = %f\n\n", iv, fv1);
  fv1 = iv/50.0;
  printf(" float : %d / 50.0 = %f\n\n", iv, fv1);
  fv1 = 1028.75;
  fv2 = fv1 + iv ;
  printf(" %f + %d = %f\n\n", fv1, iv, fv2);
  getch();
}

```

b. *Phép toán một toán hạng* : phép tăng ++, phép giảm --

a++ hoặc ++a       $\Leftrightarrow$       a = a+1

a-- hoặc --a       $\Leftrightarrow$       a = a-1

Chú ý : Tuy nhiên a++ sẽ khác ++a khi chúng đứng trong biểu thức (có phép gán).

a++ : Tăng a sau khi giá trị của a nó được sử dụng.

++a : Tăng a trước khi giá trị của a nó được sử dụng.

Ví dụ :

main()	a	b	n
{ int a=4 , b=6, n;	4	6	
n = a + b;	4	6	10
n = a++ + b;	5	6	10
n = ++a + b;	6	6	12
n = --a + b;	5	6	11
n = a-- + b;	4	6	11
n = a+ b;	4	6	10
}			

### I.5.3. *Phép gán phức hợp*:

Cú pháp: biến op= <biểu thức>  $\Leftrightarrow$  biến = biến op <biểu thức>

Với op là phép toán.

Các phép gán phức hợp : += , -= , \*= , /= , %= , <<= , >>=

Ví dụ :  $n = n * (10 + x) \Leftrightarrow n *= (10 + x)$

$n = n \% 10 \Leftrightarrow n \% = 10$

$I = I + 3 \Leftrightarrow I += 3$

<< : là phép dịch chuyển bit qua trái .

>> : là phép dịch chuyển bit qua phải .

#### **I.5.4. Phép toán quan hệ :**

< : nhỏ hơn

> : lớn hơn

>= : lớn hơn hoặc bằng

<= : nhỏ hơn hoặc bằng

!= : khác

== : bằng

#### **Chú ý:**

- Phân biệt toán tử so sánh == với phép gán =

- C không có kiểu dữ liệu boolean mà quy ước : Giá trị 0 là sai

Giá trị !=0 là đúng

#### **Ví dụ:**

a=10;

b= (a>6)\*(a-6) → b = 4

c= (a< 5)\*(a-5) → c = 0

Ví dụ: Tìm số lớn nhất trong 3 số nguyên a, b, c

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

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

```
void main ()
```

```
{ int a, b, c, max;
```

```
printf("Chương trình tìm số lớn nhất trong 3 số");
```

```
printf("Nhập a, b, c");
```

```
scanf("%d %d %d ", &a, &b, &c);
```

```
max = a;
```

```
if (max<b) max = b;
```

```
if (max<c) max = c;
```

```
printf("Số lớn nhất = %d", max);
```

```
getch();
```

```
}
```

**I.5.5.Toán tử logic:**

Toán tử		Ý nghĩa
NOT	!	Phủ định
AND	&&	Giao, và
OR		Hội

Thứ tự tính toán từ trên xuống.

Bảng chân trị:

x	! x	x	y	x && y
true	false	true	true	true
false	true	true	false	false
		false	true	false
		false	false	false

x	y	x    y
true	true	true
false	true	true
false	true	true
false	false	false

Ví dụ 1: Xét ký tự c có phải là ký số hay không?

```
char c;
if (c >= '0' && c <= '9')
    printf ("%c là kí tự số ", c);
```

Ví dụ 2: Xét ký tự ch là chữ cái hay không?

```
if ((ch>='a') and (ch<='z')) or ((ch>='A') and (ch<='Z'))
    printf ("%c là chu cai \n",ch);
```

Ví dụ 3:

```
int a=10, b=5, c=0;
a && b → 1
a && c → 0
a || c → 1
```

Ví dụ 4:

```
int a=10, b=5;
```

```
int i=2, j=0;
(a>b) && (i<j) → 0
(a<b) || (i>j) → 1
```

Ví dụ 5:

```
n=5;
while (n)
{ printf("\nSố n = %d",n);
  n--;
}
```

### **I.5.6. Toán tử phẩy:**

Cú pháp:

$T = (\text{exp1}, \text{exp2}, \text{exp3});$ // $T =$ kết quả của $\text{exp3}$
---

Ví dụ:  $m = (t=2, t*t+3) \rightarrow m=7; t=2$   
 $c = (a=10, b=5, a+b); \rightarrow a=10, b=5, c=15$

### **I.5.7. Toán tử điều kiện:**

Cú pháp :

$T = \langle \text{điều kiện} \rangle ? \langle \text{bt1} \rangle : \langle \text{bt2} \rangle;$
---

Nếu  $\langle \text{điều kiện} \rangle$  là đúng thì  $T = \langle \text{bt1} \rangle$ , ngược lại  $T = \langle \text{bt2} \rangle$

Ví dụ:  $A = i \geq \text{MAX} ? 1 : 0;$   
 $\text{printf} (" \text{max} (a,b) = \%d ", (a>b) ? a : b);$   
 $\text{lower} = (c \geq 'A' \&\& c \leq 'Z') ? c - 'A' + 'a' : c;$

### **I.5.8. Toán tử trên bit (bit wise) :**

Dạng	Ký hiệu u	Ý nghĩa
NOT bit	~	lấy bù 1
AND bit	&	giao
OR bit		hội
XOR bit	^	hội loại trừ
dịch trái	<<	nhân 2
dịch phải	>>	chia 2



Bảng chân trị:

Bit <i>A</i>	Bit <i>B</i>	Bit kết quả			
		$\sim A$	$A \& B$	$A   B$	$A \wedge B$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Ví dụ:

a= 4564	0001	0001	1101	0100
b= 13667	0011	0101	0110	0011
a & b	0001	0001	0100	0000
a   b	0011	0101	1111	0111
a ^ b	0010	0100	1011	0111

Ý nghĩa a:

1. Phép AND bit thường được dùng để kiểm tra một bit cụ thể nào đó trong thành phần dữ liệu x có trị 0 hay 1. Việc này thực hiện bằng cách sử dụng một mặt nạ (mask) với bit cần quan tâm bằng 1 còn các bit khác bằng 0. Ta lấy mask AND với giá trị x. Nếu kết quả thu được bằng mask thì là bit cần quan tâm là 1, ngược lại là 0.

Ví dụ 1:

```
void main()
{ unsigned x1; x2;
  printf ("\n cho 2 số hex(2 số) ");
  scanf ("%x %x ", &x1, &x2);
  printf ("% 02x & % 02x = % 02x\n", x1, x2, x1& x2);
}
```

Ví dụ 2: Ta muốn biết bit thứ 3 của số hexa ch là 1 hay 0 :

```
void main()
{ unsigned char ch, kq;
  printf ("\n cho 1 số hex 2 số :");
  scanf ( "%x", &ch);
  kq= ch & 0x08;
  if (kq== 0x08) printf ("bit 3 = 1");
  else printf ("bit 3 = 0");
}
```

2. Phép OR dùng để bật các bit cần thiết lên cũng nhờ vào một mặt nạ. Chẳng hạn như ta muốn bật bit thứ 7 của biến ch (unsigned char ch) lên 1:

```
ch = ch | 0x80;
```

Ví dụ 3:

```
void main()
{ unsigned char x1,x2;
  printf (“\n cho 2 số hex (ff hay nhỏ hơn):”);
  scanf (“%x %x”, &x1, &x2);
  printf (“ %02x | %02x %02x \n”, x1, x2, x1|x2);
}
```

3. Phép XOR dùng để “lật” bit nghĩa là hoán chuyển 0→1

Ví dụ 4: Để lật bit 3 ta có chương trình nh:

```
void main()
{ unsigned char ch;
  printf (“nhập 1 số hex <= ff:”);
  scanf (“%x”, &ch);
  printf (“%02x ^ 0x08 = %02x \n“, ch, ch ^ 0x08);
}
```

4. Toán tử <<, >>

<< dịch sang trái (nhân 2)

>> dịch sang phải (chia 2)

Ví dụ 5: num = 201 (0x00c9)

num :	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1
num << 2 :	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0

Kết quả = 0x0324 → 804 nghĩa là 201\* 4

Ví dụ 6:

```
void main()
{ unsigned char x1, x2 ;
  printf (“nhập 1 số hex <= ff và số bit : “);
  scanf ( "%x %d“, &x1, &x2);
  printf (“ %02x >> %d = %02x \n”, x1, x2, x1>> x2);
}
```

Chú ý: Trong phép dịch phải C làm theo 2 cách khác nhau tùy thuộc vào o

kiểu dữ liệu của toán hạng bên trái.

- Nếu toán hạng bên trái kiểu unsigned thì phép dịch sẽ điền 0 vào các bit bên trái.

- Nếu toán hạng bên trái kiểu signed thì phép dịch sẽ điền bit dấu vào các bit bên trái

Ví dụ 7: unsigned int num;

num = 39470; // 9A2E hexa

num =	1	0	0	1	1	0	1	0	0	0	1	0	1	1	1	0
num >> 2	9867	=	0x268B													
=	0	0	1	0	0	1	1	0	1	0	0	0	1	0	1	1
0 →																

Ví dụ 8: int num;

// 9A2E hexa

num = -26066

num =	1	0	0	1	1	0	1	0	0	0	1	0	1	1	1	0
num >> 2	-6517	=	0xE68B													
=	1	1	1	0	0	1	1	0	1	0	0	0	1	0	1	1
1 →																

Ví dụ 8: Chương trình đổi số hex ra số nhị phân :

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

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

```
void main()
```

```
{ int num;
```

```
unsigned int mask;
```

```
clrscr();
```

```
printf ("Chương trình đổi số hexa sang nhị phân\n");
```

```
printf ("Nhập vào số hexa :");
```

```
scanf ("%x",&num);
```

```
mask = 0x8000;
```

```
printf ("\n Dạng nhị phân của số %x là ",num);
```

```
for (int j=1; j<=16; j++)
```

```
{ printf ("%d",mask & num?1:0);
```

```
if (j==4 || j==8 || j==12) printf ("-");
```

```
mask >>=1;
```

```
}
```

```
getch();
```

```
}
```

Ví dụ 9: Chương trình nh máy tính bitwise

Đây là chương trình giả lập một máy tính thực hiện các toán tử bitwise.

```
#define TRUE 1
main()
{ char op[10];
  int x1, x2;
  while (TRUE)
  { printf ("\n\n Cho biểu thức ( vd ' ffoo & f11' ) : ");
    printf ("\n");
    switch ( op[0])
    { case '&':
      pbin (x1); printf ("& (and) \n ");
      pbin (x2);
      pline (); pbin (x1 & x2);
      break;
    case '|':
      pbin (x1); printf ("|\n ");
      pbin (x2);
      pline (); pbin (x1 | x2);
      break;
    case '^':
      pbin (x1); printf ("^\n");
      pbin (x2);
      pline (); pbin (x1 ^ x2);
      break;
    case '>':
      pbin (x1); printf (">>"); printf ("%d \n ",x2);
      pline (); pbin (x1 >> x2);
      break;
    case '<':
      pbin (x1); printf ("<<"); printf ("%d \n", x2);
      pline (); pbin (x1 << x2);
      break;
    case '~':
      pbin (x1); printf ("~ \n");
      pline (); pbin (~ x1);
      break;
    default : printf ("Toán tử không hợp lệ /n ");
  }
}
```

```

    }
    pbin (num)
    int num;
    { unsigned int mask;
      int j, bit;
      mask = 0x8000;
      printf ("%04x", num);
      for(j=0; j<16; j++)
      { bit = ( mask & num ) ? 1:0;
        printf ("%d", bit);
        if (j== 7) printf ("- ");
        mask >>= 1;
      }
      printf ("- ");
      mask >> 1;
    }
    pline ()
    { printf ("- - - - - \n");
    }

```

### \* Sự chuyển kiểu bắt buộc:

Trong C có 2 trường hợp chuyển kiểu: chuyển kiểu tự động và chuyển kiểu bắt buộc.

Chuyển kiểu bắt buộc: được áp dụng khi chuyển kiểu tự động không được.

**Cú pháp:** (Type) biểu thức

Ví dụ: d = (float) (f - 32)

int a= 100, b=6;

double f;

f=a/b // kết quả f=16

f= (double) a/ (double)b // kết quả f= 100.0 / 6.0= 16.666.

### \* Mức độ ưu tiên của các phép toán:

Độ ưu tiên	Phép toán	Thứ tự kết hợp
1	() [] →	→
2	! ~ ++ -- (type) * & size of	←
3	* / %	→
4	+ -	→
5	<< >>	→

6	< <= > >=	→
7	= = !=	→
8	&	→
9	^	→
10		→
11	&&	→
12		→
13	?	←
14	= + = - = ..	←

Ví dụ 1: 3/4 \* 6      #      3\*6 /4  
                  0 \* 6                      18 /4  
                  0                              4

Ví dụ 2: (float) 2 /4      #      (float) (2/4)  
                  2.0 /4                      (float) 0  
                  0.5                              0.0

## I.6. Chuỗi

**I.6.1. Định nghĩa a:** Chuỗi là một mảng mà các phần tử của nó có kiểu ký tự.

Khai báo một chuỗi ký tự chứa tối đa 49 ký tự

```
char chuỗi[50];
```

\* Lưu ý: Tất cả các chuỗi đều được kết thúc bằng ký tự NULL (\0). Do đó, nếu chuỗi dài 50 thì ta chỉ có thể chứa tối đa 49 ký tự.

### I.6.2. Khởi động trị:

```
char chuỗi[ ] = { 'A', 'N', 'H', '\0' };
```

```
char chuỗi[ ] = "ANH";
```

### I.6.3. Nhập / xuất chuỗi:

#### a. Nhập chuỗi:

```
gets (chuỗi)
```

#### b. Xuất chuỗi:

```
puts (chuỗi)
```

### Chú ý:

- Khi nhập chuỗi thì không được dùng hàm scanf vì hàm scanf không chấp nhận khoảng trắng.

Ví dụ: scanf("%s", chuỗi); // ta nhập vào Nguyễn Văn Ái thì

// chuỗi = “Nguyễn” vì hàm scanf cắt khoảng trắng

- Khi dùng hàm gets trong chương trình thì không nên dùng hàm scanf ở bất kỳ đâu dù rằng dùng hàm scanf để nhập số mà ta nên dùng hàm gets và hàm atoi, atof để nhập số.

Vì :

```
scanf("%d", &n);    // ta nhập số 5 ↴
gets (chuỗi);       // lúc này chuỗi = "" (chuỗi rỗng)
```

#### **I.6.4. Hàm chuyển đổi số sang chuỗi và ngược lại**

sốint = atoi (chuỗiisố) // chuyển chuỗi số sang số nguyên

sốf = atof (chuỗiisố) // chuyển chuỗi số sang số thực

\* Hai hàm này nằm trong <stdlib.h>

#### **I.6.5. Các hàm về chuỗi: (#include <string.h> )**

- **int strlen(S)** : trả về chiều dài chuỗi S.

- **int strcmp(S1, S2)**: so sánh chuỗi S1 với S2. Nếu chuỗi S1 giống S2 kết quả bằng 0. Nếu chuỗi S1 < S2 kết quả là âm, nếu chuỗi S1 > S2 kết quả > 0.

- **int stricmp(S1, S2)**: so sánh chuỗi S1, S2 không phân biệt chữ thường hay chữ hoa

- **int strncmp(S1, S2, n)**: chỉ so sánh n ký tự đầu của 2 chuỗi S1, S2 với nhau.

- **int strnicmp(S1, S2, n)**: chỉ so sánh n ký tự đầu của 2 chuỗi S1, S2 với nhau, không phân biệt chữ thường, chữ hoa

- **strcpy(dest, source)**: chép chuỗi từ nguồn source sang đích dest

Ví dụ: char string[10];

```
char *str1 = "abcdefghi";
```

```
strcpy(string, str1);
```

```
printf("%s\n", string); // "abcdefghi"
```

- **strncpy(dest, source, n)**: chép chuỗi từ nguồn sang đích với nhiều nhất là n ký tự.

Ví dụ:

```
char string[10];
```

```
char *str1 = "abcdefghi";
```

```
strncpy(string, str1, 3);    // string = "abcx1zwe12"
```

```
string[3] = '\0';           // Đặt ký tự kết thúc chuỗi và o cuối chuỗi.
```

```
printf("%s\n", string);     // "abc"
```

- **strcat(dest, src)**: nối chuỗi src và o sau chuỗi dest. Chiều dài của chuỗi kết quả bằng strlen(dest) + strlen(src)

Ví dụ:

```
char destination[25];
char *blank = " ", *c = "C++", *turbo = "Turbo";
strcpy(destination, turbo); // destination = "Turbo"
strcat(destination, blank); // destination = "Turbo "
strcat(destination, c);      // destination = "Turbo C++"
```

- **strncat(dest, src, n)**: nối nhiều nhất là n ký tự của src vào cuối chuỗi dest, sau đó thêm ký tự null vào cuối chuỗi kết quả.

Ví dụ:

```
char destination[25];
char *source = " States";
strcpy(destination, "United");
strncat(destination, source, 6);
printf("%s\n", destination); // destination = "United State"
```

- **char \* strchr(s, ch)**: trả về địa chỉ của ký tự ch đầu tiên có trong chuỗi S; nếu không có thì trả về NULL (thường dùng để lấy họ)

Ví dụ:

```
char string[15];
char *ptr, c = 'r';
strcpy(string, "This is a string");
ptr = strchr(string, c);
if (ptr)
    printf("Ký tự %c ở vị trí : %d\n", c, ptr-string);
else
    printf("Không tìm thấy ký tự %c\n", c);
```

- **char \* strstr(S1, S2)**: trả về vị trí của chuỗi S2 trong chuỗi S1; nếu S2 không có trong S1 thì hàm strstr trả về trị NULL.

**I.6.6. Mảng các chuỗi**

\***Khai báo**: Khai báo biến **ds** chứa tối đa 50 chuỗi ký tự, mỗi chuỗi ký tự có tối đa 30 ký tự.

```
char ds[50][30];
```

Chú ý:

- Không nên gán chuỗi với chuỗi (s1= s2) mà phải dùng hàm strcpy(S1,S2)
- Không được so sánh 2 chuỗi bằng các toán tử quan hệ (S1== S2, S1>S2, S1>= S2), mà phải dùng hàm strcmp(S1,S2).



Ví dụ: Viết chương trình nh tìm kiếm 1 từ trong 1 câu

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

void main ()
{
    char cau[80], từ[7], *ptr;
    printf("Nhập câu      :");
    gets(cau);
    printf("Nhập từ :");
    gets(từ);
    ptr = strstr(cau, từ);
    if (ptr == NULL) printf("Không có từ");
    else printf("có từ");
}
```

## **II. CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG C:**

Ngôn ngữ C là ngôn ngữ lập trình cấp cao có cấu trúc, gồm: cấu trúc tuần tự, chọn, và lặp.

### **II.1 Cấu trúc tuần tự (Sequence) :**

Các lệnh trong chương trình được thực hiện tuần tự từ lệnh này đến lệnh khác cho đến khi hết chương trình.

Ví dụ : Viết chương trình tính và in ra diện tích của hai đường tròn bán kính lần lượt là 3m và 4.5m cùng với hiệu số của 2 diện tích.

```
#define PI 3.14159
#include <stdio.h>
#include <conio.h>

void main()
{
    float r1, r2, hieuso;
    clrscr();
    printf("\nCHUONG TRINH TINH DIEN TICH 2 HINH TRON VA HIEU SO\n");
    printf("Ban kinh hinh tron thu nhat : ");
    scanf("%f",&r1);
    printf("Ban kinh hinh tron thu hai : ");
    scanf("%f",&r2);
    printf("Dien tich hinh tron 1 = %.2f\n",PI*r1*r1);
```

```
printf ("Dien tich hinh tron 2 = %.2f\n",PI*r2*r2);
hieuso = PI*r1*r1 - PI*r2*r2;
printf ("Hieu so dien tich 2 hinh tron = %.2f\n",hieuso);
getch();
}
```

## II.2. Cấu trúc chọn

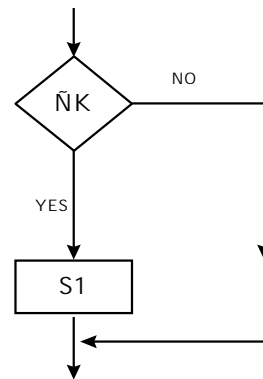
Ký hiệu: đk là biểu thức Logic

$S_1, S_2$  là các phát biểu hay 1 nhóm các phát biểu (lệnh)

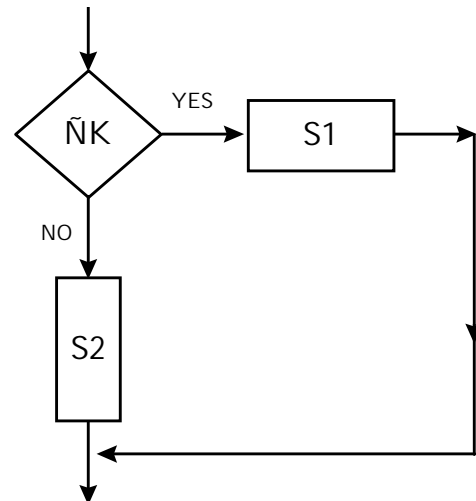
### II.2.1. Lệnh if else:

Cú pháp:

if (đk)  $S_1$ ;



if (đk)  $S_1$ ;  
else  $S_2$ ;



Chú ý: Các lệnh if else lồng nhau

```
if (đk1)   $S_1$ ;  
else if (đk2)   $S_2$ ;  
    else if (đk3)   $S_3$ ;  
        else   $S_4$ ;
```

Ví dụ 1: Tìm max(a,b,c)

```

if (a>b)
    if (a>c) max=a;
    else max=c;
else if (b>c) max =b;
    else max= c;

```

**Ví dụ 2:** Tính hàm  $f(x)$  :

```

f(x) = x2    , nếu -2 <= x < 2
      4      ,      x >= 2
if (x>=-2 && x<2)
    fx= x*x;
else if (x>=2)
    fx= 4;
    else { printf("\n Không xác định") ; exit(0) ;}

```

### II.2.2. *Lệnh chọn lựa: switch\_case*

#### Cú pháp:

```

switch (biểu thức)
{ case hằng 1: S1;
  case hằng 2: S2; break;
  .
  .
  .
  case hằng 3: Sn; break;
  default: S0;
}

```

#### Cách hoạt động:

- (biểu thức) có kết quả nguyên
- Hằng: ký tự, số nguyên, biểu thức có số nguyên
- Nếu kết quả bằng hằng nào đó thì nó sẽ làm lệnh S<sub>i</sub> và tuần tự thực hiện hết các lệnh ở dưới cho đến khi hết lệnh switch.
- Muốn ngắt sự tuần tự trên thì phải dùng lệnh break.

**Ví dụ:** Nhập 1 ký tự số dạng hex đổi ra số thập phân

```

#include <stdio.h>
#include <conio.h>
void main()

```

```
{
    unsignedchar ch;
    int k;
    clrscr();
    printf("Nhap 1 ky tu so hex : ");
    ch=getche();
    switch (ch)
    {
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9': k=ch-'0'; break;
        case 'A':
        case 'B':
        case 'C':
        case 'D':
        case 'E':
        case 'F':k=ch-'A'+10; break;
        case 'a':
        case 'b':
        case 'c':
        case 'd':
        case 'e':
        case 'f': k= ch-'a'+10; break;
        default: k=0;
    }
    printf ("\nSo thap phan cua ky tu hexa %c la %d ",ch,k);
    getch();
}
```

Ví dụ: Viết chương trình nh tạo 1 máy tính nh có 4 phép toán + , - , \* , /

```
#include <stdio.h>
#include <conio.h>
void main()
{
```

```

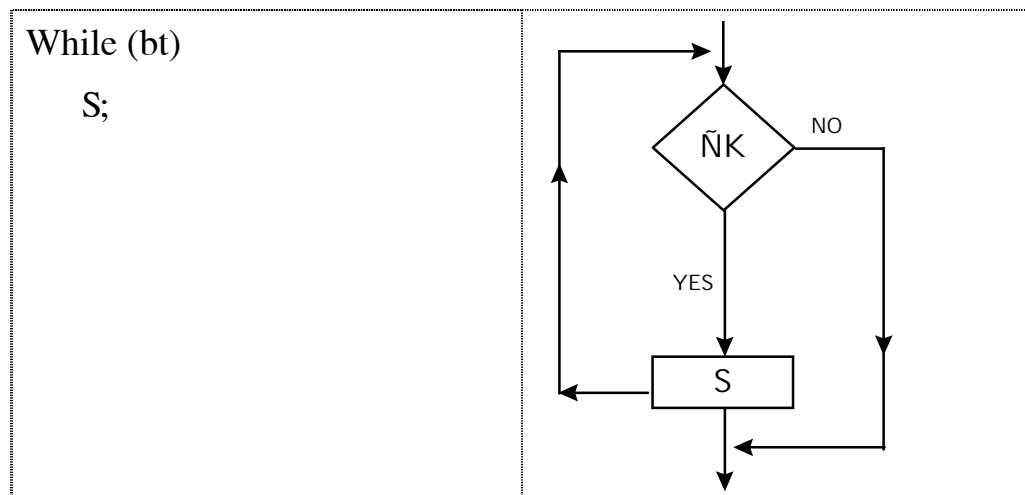
float num1, num2;
char op;
clrscr();
printf ("Go vao so, toan tu, so \n");
scanf("%f %c %f", &num1, &op, &num2);
switch (op)
{
    case '+': printf("= %f",num1+num2);
              break;
    case '-': printf("= %f",num1-num2);
              break;
    case '*': printf("= %f",num1*num2);
              break;
    case '/': printf("= %f",num1/num2);
              break;
    default : printf("Toá n tử lạ , không biết");
}
}

```

### II.3. Cấu trúc lặp

#### II.3.1. Lệnh while:

##### Cú pháp:



Chú ý: Trong phần thân lệnh phải có biến điều khiển vòng lặp.

Ví dụ 1: Viết chương trình nh in ra bảng mã ASCII

```

void main ()
{ int n=0;
  while (n <= 255)

```

```

    { printf("%c có mã ASCII là %d", n, n);
      n ++
    }
  }

```

**Ví dụ 2:** Nhập một chuỗi ký tự, và kết thúc nhập bằng ESC

```

#define ESC '\0x1b'
#include <stdio.h>
#include <conio.h>
void main()
{ char c;
  while (1)
    if ((c = getch()) == ESC) break;
}

```

**Ví dụ 3:** Viết chương trình nh in bảng cửu chương

```

void main ()
{ int a, b;
  b = 1;
  while (b <= 9)
  { a = 2;
    while (a <= 9)
    { printf("%d * %d = %d \t", a, b, a*b);
      a++;
    }
    b ++;
    printf("\n");
  }
}

```

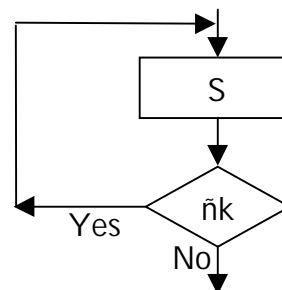
### II.3.2. Lệnh do while:

**Cú pháp:**

```

do
    S
while (bt);

```



**Ví dụ 1:** Viết chương trình nh in bảng mã ASCII

```

#include <stdio.h>

```

```

main ()
{ int n=0;
  do
    { printf("%c có mã ASCII %d\n", n, n);
      n ++;
    } while (n <= 255);
}

```

### II.3.3. Lệnh for:

#### Cú pháp:

```

for ([bt_khởi động]; [bt_đk]; [bt_lặp])
    S;

```

Ví dụ 1: Lặp lệnh S từ 1 đến 10

```

for (int I=1; I== 10; I++) → sai
    S;
for (int I=1; I<= 10; I++) → đúng
    S;.

```

Ví dụ 2: for (; ;)

```

{ c = getch()
  if (c == ESC) break;
}

```

#### So sánh vòng lặp while - for:

<pre> bt_khởi động; while (Biểu_Thức_đK) {     S;     BT_lặp; } </pre>	<pre> for ( bt_khởi động; bt_đk; bt_lặp)     S; </pre>
--	--

Ví dụ 3: Viết chương trình nh in ra bảng cửu chương bằng vòng for

```

void main ()
{ int a;
  for (a=2; a<= 9; a++)
    { for (b =1; b <= 9; b++)
      printf("%d* %d = %d \t", a, b, a*b);
      printf("\n");
    }
}

```

Ví dụ 4: Viết chương trình nh tính nh n giai thừa

```

void main ()
{ long gt = 1;
  for (int i = 1; i <= n; i++)
    gt = gt * i;
  printf("%d! = %u \n", n, gt);
}

```

Ví dụ 5: Viết chương trình nh tính nh biểu thức:

$$(1 + 1/1^2) * (1 + 1/2^2) * \dots (1 + 1/n^2)$$

```

void main ()
{ int i, n;
  float S;
  printf("Nhập số :");
  scanf("%d", &n);
  S = 1;
  for (i = 1; i <= n; i++)
    S = S * (1 + 1.0 / (i*i));
  printf("\nKet qua = %f", S);
}

```

### \* Phát biểu break, continue, goto:

#### 1. Break:

Lệnh break cho phép thoát ra sớm khỏi vòng lặp ( whiledo , for, dowhile), lệnh switch.

#### 2. Lệnh continue:

Lệnh continue chỉ dừng trong vòng lặp làm cho chương trình nh nhảy tắt về điều kiện kiểm tra của vòng lặp để bắt đầu một vòng lặp mới.

Ví dụ: Viết chương trình nh nhập một câu chữ thường kết thúc bằng dấu chấm, xuất ra bằng chữ hoa

```

void main ()
{ char ch;
  while (1)
  { ch = getch ();
    if ((ch >= 'a') && (ch <= 'z'))
      printf("%c", ch - 'a' + 'A');
    if (ch == ' ') continue;
    if (ch == '.') break;
  }
}

```



}

**3. Lệnh goto:** dùng để chuyển điều khiển chương trình về một vị trí nào đó.

**Cú pháp:** Goto nhãn;

Lệnh goto sẽ chuyển điều khiển chương trình ngay lập tức về vị trí đặt nhãn.

Ví dụ:

Again:

;

.

.

goto Again;

## **Bài tập:**

1. Viết chương trình tính diện tích

- Hình vuông
- Hình thang
- Tam giác thường
- Tam giác vuông
- Hình tròn

2. Tính khoảng cách một điểm  $(X_0, Y_0)$  tới một đường thẳng  $Ax + By + C = 0$

$$S = \frac{AX_0 + BY_0 + C}{A^2 + B^2}, \text{ nhập } A, B, C, X_0, Y_0$$

3. Cho 3 số thực x, y, z. Tìm

- a.  $\text{Max}(x+y+z, x*y*z)$
- b.  $\text{Min}^2((x+y+z) / 2, x*y*z) + 1$

4. Viết chương trình tìm số lớn nhất trong 3 số nguyên a, b, c

5. Giải phương trình bậc 2  $Ax^2 + Bx + C = 0$  trên trường số thực.

6. Viết chương trình tính diện tích hình tròn, biết bán kính r dương; Chương trình có kiểm tra số liệu nhập vào, nếu  $r \leq 0$  thì chương trình báo lỗi và bắt nhập lại.

7. Viết chương trình nhập số thập lục phân in ra số nhị phân, cứ 4 số thì cách một khoảng trắng.

8. Viết chương trình tạo 1 máy tính gồm các phép toán sau : + , - , \* , / , ^ ( $a^x$  với x nguyên dương), @ ( $e^x$ )

9. Viết chương trình kiểm tra 1 bit bất kỳ của số bằng 0 hay 1
10. Viết chương trình tính kết quả của một số sau khi dịch phải hoặc dịch trái n lần.
11. Đếm số lần xuất hiện của từ một trong 1 câu
12. Thay thế 1 từ trong 1 câu bằng 1 từ khác.
13. Nhập họ tên, tách họ tên ra làm 2 phần họ và tên riêng.
14. Viết chương trình đổi các ký tự đầu của các từ ra chữ in, các ký tự còn lại ra chữ thường.
15. Viết chương trình cho phép ta kiểm tra một password là đúng hay sai (nhập tối đa password 3 lần).
16. Cho chuỗi s, viết chương trình di chuyển chữ từ bên trái qua bên phải của màn hình tại dòng 5. Quá trình di chuyển dừng lại khi ta ấn phím ESC.
17. Bạn hãy viết chương trình tính giá trị tổng cộng của một sản phẩm có kể cả thuế, biết rằng tỷ suất thuế là 13.6% tính trên giá gốc. Giá gốc của sản phẩm được đọc vào, và cần in ra:
  - Tiền thuế
  - Giá đã có thuế
18. Trong một kỳ thi cuối khóa, các học viên phải thi 4 môn : môn I hệ số 2, môn II hệ số 4, môn III hệ số 1, môn IV hệ số 2, điểm các môn cho tối đa là 10 điểm. Viết chương trình nhập điểm của 4 môn và tính điểm trung bình.
19. Một người bán rượu bán N chai rượu có đơn giá là P. Nếu tổng số tiền vượt quá 5000000 thì việc chuyên chở là miễn phí, nếu không phí chuyên chở thường được tính bằng 1% tổng trị giá hàng. Viết chương trình nhập và o N, P. In ra các chi tiết Tổng trị giá hàng, tiền chuyên chở, và tổng số tiền phải trả.
20. Một sinh viên dự tuyển có các chi tiết sau : họ tên, điểm L1 của lần 1, điểm L2 của lần 2, điểm thi cuối kỳ CK. Viết chương trình xác định xem một sinh viên có được tuyển hay bị loại, biết rằng sẽ được tuyển nếu điểm được tính theo công thức sau lớn hơn hay bằng 7 :  $\text{Max}((L1+L2+CK)/3, CK)$ .
21. Viết chương trình cho biết tiền lương hàng ngày của một người giữ trẻ. Cách tính là 15000đ/giờ cho mỗi giờ trước 14 giờ và 25000 đ/giờ cho mỗi giờ sau 14 giờ. Giờ bắt đầu và giờ kết thúc công việc được nhập từ bàn phím.
22. Cho 3 số thực x, y z
  - a. Tồn tại hay không một tam giác có 3 cạnh có độ dài x, y, z ?
  - b. Nếu là tam giác thì nó vuông, cân, đều hay thường
23. Giải hệ phương trình bậc nhất:

$$\begin{cases} ax+by = c \\ a'x+b'y = c' \end{cases}$$

Hướng dẫn:

$$\begin{aligned} D &= \begin{vmatrix} a & b \\ a' & b' \end{vmatrix} = ab' - a'b \\ Dx &= \begin{vmatrix} c & b \\ c' & b' \end{vmatrix} = cb' - c'b \\ Dy &= \begin{vmatrix} a & c \\ a' & c' \end{vmatrix} = ac' - a'c \end{aligned}$$

if  $D \neq 0$       $x = Dx/D$ ;  $y = Dy/D$

else if  $((Dx \neq 0) \vee (Dy \neq 0))$      pt vô nghiệm

else     pt vô định

24. Giải hệ phương trình 3 ẩn số bậc nhất

$$\begin{cases} a_1x + b_1y + c_1z = d_1 \\ a_2x + b_2y + c_2z = d_2 \\ a_3x + b_3y + c_3z = d_3 \end{cases}$$

25. Viết chương trình giải phương trình trùng phương  $ax^4 + bx^2 + c = 0$

26. Người ta muốn lập hóa đơn cho khách hàng của Công ty điện lực. Chỉ số đầu và chỉ số cuối kỳ sẽ được cho biết. Biết rằng biểu giá được tính tùy theo điện năng tiêu thụ.

- Nếu điện năng tiêu thụ nhỏ hơn 100Kwh, giá mỗi Kwh là 500đ.

- Nếu điện năng tiêu thụ từ 100Kwh trở lên, thì mỗi Kwh dôi ra sẽ được tính giá là 800đ

- Phí khu vực là 5000đ cho mỗi khách hàng. Viết chương trình tính tiền phải trả tổng cộng gồm tiền điện, và phí khu vực

27. Biết 2 số X, Y biểu diễn thời gian tính theo giây. Người ta muốn viết chương trình :

- Đọc 2 số này và in ra tổng của chúng

- Chuyển các số này và tổng ra dạng giờ, phút, giây của chúng rồi in ra. Kiểm xem kết quả của 2 cách tính có như nhau không?

28. Viết chương trình in bảng cửu chương từ 2 → 9 theo hàng ngang

29. In tam giác \*, hình chữ nhật \*, rỗng - đặc

30. Vẽ lưu đồ và viết chương trình tính :

a.  $1+2+\dots+n$

- b.  $1+3+5+\dots+2n-1$   
c.  $2+4+6+\dots+2n$   
d.  $1^2+2^2+3^2+\dots+n^2$   
e.  $1/1+1/2+1/3+\dots+1/n$   
f.  $2+4+8+\dots+2^n$   
g.  $1+1/2^2+1/3^2+\dots+1/n^2$
31. Cho  $\epsilon = 0.000001$ , xác định các tổng sau đây sao cho số hạng cuối cùng của tổng là không đáng kể (số hạng cuối cùng  $< \epsilon$ )
- a.  $1/1+1/2+1/3+1/4+\dots$   
b.  $1+1/2^2+1/3^2+\dots$
32. Viết chương trình in ra bảng mã ASCII, 16 ký tự trên 1 dòng.
33. Vẽ lưu đồ và viết chương trình :
- a. Xét một số có phải là số nguyên tố hay không ?  
b. In trên màn hình 100 số nguyên tố đầu tiên
34. Viết chương trình cho phép một ký tự ngẫu nhiên rơi trên màn hình. Nếu người sử dụng không kịp ấn phím tương ứng và để chạm đáy màn hình thì thua cuộc.
35. Cho hàm Fibonacci:
- $$F_n = \begin{cases} 1 & ; n=0,1 \\ F_{n-1} + F_{n-2} & ; n \geq 2 \end{cases}$$
- a. Tìm  $F_n$ , với  $n$  do ta nhập  
b. In ra  $N$  phần tử đầu tiên của dãy Fibonacci
36. Viết chương trình đọc và o số nguyên  $N$  và in ra  $1*2*3*...*N$  cho đến khi  $N \leq 0$ .
37. Cho 1 dãy gồm một triệu từ 32 bit, hãy đếm số bit 1 trong mỗi từ.
38. Viết chương trình đoán số : người chơi sẽ đoán 1 số trong phạm vi từ 0 đến 100, tối đa 5 lần. Chương trình kiểm tra kết quả và xuất thông báo hướng dẫn:
- Số bạn đoán lớn hơn
  - Số bạn đoán nhỏ hơn
  - Bạn đoán đúng
  - Máy thắng cuộc

### III. HÀM - ĐỀ QUY:

#### III.1. Hàm:

**III.1.1. Mục đích:** Hàm là một chương trình con của chương trình chính, với các mục đích sau:

\* Tránh việc lặp đi lặp lại các đoạn chương trình giống nhau, nhờ đó, ta sẽ tiết kiệm lúc lập trình.

\* Tổ chức chương trình: Dùng hàm ta sẽ phân mảnh chương trình thành những khối nhỏ độc lập, mỗi khối là một hàm thực hiện một công việc nào đó. Từng hàm sẽ được lập trình, kiểm tra hoàn chỉnh; Sau đó, ta kết lại để tạo chương trình hoàn chỉnh. Nhờ vậy, chương trình về sau dễ hiểu và dễ sửa.

\* Tính độc lập: cho phép hàm độc lập với chương trình chính. Ví dụ hàm có những biến cục bộ mà chương trình chính và các hàm khác không thể đụng tới. Do đó, nếu ta có khai báo các biến trùng tên với các hàm khác thì cũng không sợ ảnh hưởng tới các biến trùng tên đó.

#### Chú ý:

- C không cho phép các hàm lồng nhau nghĩa là các hàm đều ngang cấp nhau (có thể gọi lẫn nhau).

- C không phân biệt thủ tục hay hàm, mà chỉ quan tâm đến kết quả trả về của hàm. Nếu hàm không trả về kết quả gì cả thì có thể xem là thủ tục.

Ví dụ: Vẽ hình chữ nhật đặc bằng dấu '\*':

```
#include <stdio.h>
#include <conio.h>
void ve_hcn(int d,int r)    // khai báo void cho biết hàm không trả về trị
nào cả
{ int i,j ;                // i, j là 2 biến cục bộ trong hàm ve_hcn
  for (i=1;i<=r;i++)
  {
    for (j=1;j<=d; ++j)
      printf("*");
    printf("\n");
  }
}
void main()
{ int d=20, r=5;
  clrscr();
  ve_hcn(d,r); // lời gọi hàm
  getch();
}
```

### III.1.2. Cú pháp định nghĩa hàm

#### Cú pháp:

```
Kiểu  tên_hàm (ds_đối_số)
    { Khai báo biến cục bộ;
      lệnh;
      [ return (expr); ]
    }
```

- **Kiểu:** Là kết quả trả về của hàm. Nếu không ghi kiểu, C sẽ tự hiểu là kiểu int. Nếu không muốn có kết quả trả về thì ghi kiểu **void**.

- **Danh sách đối số:** Liệt kê các đối số và kiểu của đối số gửi đến hàm, cách nhau bởi dấu ','. Nếu không có đối số ta chỉ cần ghi()

- **Lệnh return:** có các dạng sau:

```
return;
return (expr);
return expr;
```

Ví dụ: Hàm chuyển chữ thường sang chữ hoa

```
#include <stdio.h>
#include <conio.h>
Get_upper(char ch)
{ return (ch >='a' && ch <='z')? ch-'a'+'A':ch;
}
void main()
{ char ch;
  printf("\nNhap vao ky tu bat ky ");
  ch=getche();
  printf("\nKy tu %c qua ham Get_upper tro thanh %c",ch,Get_upper(ch));
  getch();
}
```

#### Lưu ý:

- Hạn chế của lệnh return là chỉ trả về một kết quả.
- Lệnh return không nhất thiết phải ở cuối hàm. Nó có thể xuất hiện ở bất kỳ nơi nào trong hàm. Khi gặp lệnh return, quyền điều khiển sẽ chuyển ngay về chương trình gọi.

### III.1.3. Các loại truyền đối số

#### a. Truyền theo trị

```
#include <stdio.h>
#include <conio.h>
int max (int a,int b)
{ int m= a>b?a:b;
  a=a*100;
  b=b*100;
  return m;
}
void main()
{ int a,b,c;
  clrscr();
  printf("\nChương trình tìm Max(a,b)\n");
  printf("Nhập a b : ");
  scanf("%d %d",&a,&b);
  c=max(a,b);
  printf("\nGiá trị lớn nhất =%d",c);
  printf("\nGiá trị a =%d",a);
  printf("\nGiá trị b =%d",b);
  getch();
}
```

Giả sử ta chạy chương trình nh trên:

```
Nhập a b : 12 24
Giá trị lớn nhất =24
Giá trị a =12
Giá trị b=24
```

#### Nhân xét:

- Ta nhận thấy rằng giá trị hai biến a, b trước và sau khi vào hàm max là không thay đổi (mặc dù trong hàm max, cả hai biến a và b đều thay đổi); đó là cơ chế của sự truyền đối số theo trị.

#### **Lời gọi hàm:    *tên hàm (ds đối số thực);***

- Nếu truyền đối số theo trị thì đối số thực có thể là biến, hoặc có thể là biểu thức.

Ví dụ: c = max(1000,b);

b. *Truyền theo địa chỉ* : đối số thực là địa chỉ của biến

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

```

max (int &a,int b)
{ int m= a>b? a : b;
  a=a *100;
  b=b*100;
  return m;
}
void main()
{ int a,b,c;
  clrscr();
  printf("\nChương trình tìm Max(a,b)\n");
  printf("Nhập a b : ");
  scanf("%d %d",&a,&b);
  c=max(a,b);           // a là tham số thực biến
  printf("\nGiá trị lớn nhất =%d",c);
  printf("\nGiá trị a =%d",a);
  printf("\nGiá trị b =%d",b);
  getch();
}

```

Giả sử ta chạy chương trình nh trên:

```

Nhập a b : 12 24
Giá trị lớn nhất =24
Giá trị a =1200
Giá trị b=24

```

Nhận xét:

- Ta nhận thấy rằng giá trị biến a trước và sau khi vào hàm max đã thay đổi; đó là cơ chế của sự truyền đối số theo địa chỉ .

**Lời gọi hàm:**    *tên hàm (tên biến);*

- Nếu truyền đối số theo địa chỉ thì tham số thực bắt buộc phải là một tên biến.

Ví dụ: c = max(1000,b); là sai

Ví dụ: Viết hàm giaohoán để hoán đổi giá trị của 2 biến nguyên a,b.

```

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

```



```

void main()
{ int a,b;
  printf ("a, b = ");
  scanf ("%d %d", &a, &b);
  giaohoa(a, b);
}

```

* <u>Truyền đối số là mảng</u>	gọi hàm (mang) hàm (kiểu mảng[]) hoặc hàm (kiểu *mang)
--------------------------------	---

Ví dụ: Cộng thêm một hàng số vào mảng tên là dayso.

```

#define SIZE 5 // dãy số có 5 số
#include <stdio.h>
#include <conio.h>
void add_const(int *a, int n, int con) // int *a ⇔ int a[]
{ for (int i=0; i<n; i++)
  *a = *(a++) + con;
}
void main()
{ int dayso [SIZE] = { 3,5,7,9,11 };
  int konst = 10;
  add_const(dayso, SIZE, konst);
  printf("\nDay so sau khi cong them hang so :");
  for (int i=0; i<SIZE; i++)
    printf("%d ", *(dayso+i)); // *(dayso+i) ⇔ dayso[i]
  getch();
}

```

### III.1.4. Khai báo nguyên mẫu của hàm

- Khai báo hàm theo nguyên mẫu đòi hỏi phải khai báo kiểu dữ liệu của đối số nằm trong định nghĩa hàm chứ không đặt chúng trên các dòng riêng.

- C không nhất thiết phải khai báo hàm theo nguyên mẫu. Tuy nhiên, nên có vì nó cho phép chương trình dịch phát hiện có lỗi do không đúng kiểu dữ liệu giữa trị truyền đến hàm và trị mà hàm mong muốn.

Ví dụ:

```

float dinhthuc (float a, float b, float c, float d)
{ return (a*d- b*c);
}
void main()

```

```

{ float a ,b, a1, b1;
  printf("Nhap a,b,a1,b1:");
  scanf("%f %f %f %f",&a,&b,&a1,&b1);
  printf("Dinh thuc = %f",dinhthuc(a,b,a1,b1);
  getch();
}

```

### III.1.5. Phạm vi tồn tại của biến:

a. *Biến toàn cục*: là biến được khai báo ngoài các hàm ( kể cả hàm main). Nó được phép truy nhập đến từ tất cả các hàm trong suốt thời gian chương trình hoạt động.

Ví dụ: Khai báo ngoài hàm main

```

Kiểu   tên biến;    // biến toàn cục
void main()
{
}

```

b. *Biến cục bộ*: là biến được khai báo trong các hàm, kể cả trong hàm main. Nó không cho phép các hàm khác truy nhập đến nó. Nó tồn tại trong thời gian sống của hàm chứa nó.

```

void main()
{ kiểu      tên biến;    → biến cục bộ trong hàm main()
}

```

c. *Biến ngoài*: là biến mà các hàm có thể truy xuất tới mà không phải phân phối bộ nhớ. Nó được dùng ở các hàm trên các tập tin khác nhau liên kết lại.

```

External   Kiểu   tên biến;
void main()
{
}

```

d. *Biến tĩnh*: là một biến cục bộ của một hàm nhưng vẫn giữ giá trị của lần gọi hàm đó cuối cùng

```

void main()
{ static      Kiểu   tên biến;
}

```

e. *Biến thanh ghi*: là một biến sử dụng các thanh ghi của CPU để tăng tốc độ truy xuất

```

register      Kiểu   tên biến;

```

### III.1.6. Các dẫn hướng tiên xử lý

#### III.1.6.1. #define

a. Định nghĩa hằng:

```
#define    tên hằng    giá trị
Ví dụ: #define    PI                3.14
        #define    MAX                100
        #define    THONGBAO    “Hết Giá ”
        #define    khoangtrang    ‘ ‘
```

b. Định nghĩa Macro:

```
#define    tên macro (đối số ) thao tác
Ví dụ: #define    sqr (x)    x*x
        #define    sum (x,y)    x+y
        a = b * sum (x,y); // ⇔ a = b * x+y;
        #define    sum (x,y)    (x+y)
        a = b * sum (x,y); // ⇔ a = b * (x+y);
```

\***Chú ý:** Trong các thao tác Macro nên sử dụng các dấu ngoặc để tránh dẫn ra một kết quả sai

```
Ví dụ: #define    max (a,b)    ((a) > (b) ? (a) : (b))
        #define    hoán vị (a,b) { int tạm =a; a= b; b= tạm; }
        #define    error (n)    printf (“ error %d”, n)
```

Dưới đây y một số Macro phân tích ký tự, tất cả đều trong <ctype.h>. Các macro này trả về trị khác 0 nếu thành công. Đối với mỗi macro, thành công được định nghĩa như sau:

Macro	Ký tự
isalpha (c)	c là ký tự $a \rightarrow z, A \rightarrow Z$
isupper (c)	c là ký tự $A \rightarrow Z$
islower (c)	c là ký tự $a \rightarrow z$
isdigit (c)	c là ký số $0 \rightarrow 9$
isxdigit (c)	$0 \rightarrow 9, A \rightarrow F, a \rightarrow z$
isctrl(c)	c là ký tự xóa hoặc ký tự điều khiển (0x7F hoặc 0x00 đến 0x1F)
isspace (c)	c là ký tự space, tab, carriage return, new line (0x09 đến 0x0D, 0x20)

Khai báo: char c;

**\* Phân biệt Macro với hàm:**

- Dùng Macro: truy xuất nhanh, tốn bộ nhớ.
- Dùng hàm: ngược lại

**III.1.6.2. #include**

Là tiền xử lý dùng để kết nối tập trung khai báo trong include với tập tin đang làm việc.

# include < tên tập tin.h>

# include “ tên tập tin.h”

Dạng < > : đi tìm tập tin.h trong thư mục đã được chỉ định trong Include Directories.

Dạng “ ”: tìm tập tin.h trong thư mục Source Directories, nếu không có, nó đi tìm trong thư mục đã được chỉ định trong Include Directories.

**III.2. Đệ qui (Recursion):**

**III.2.1. Khái niệm:** Đệ qui là 1 công cụ rất thường dùng trong khoa học máy tính và trong toán học để giải quyết các vấn đề. Trước hết, chúng ta hãy khảo sát thế nào là một vấn đề có đệ qui qua ví dụ sau:

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

Ta nhận thấy rằng, công thức trên có thể diễn đạt lại như sau:

$S(n) = S(n-1) + n$ , và

$S(n-1) = S(n-2) + (n-1)$

.....

$S(2) = S(1) + 2$

$S(1) = 1$

Như vậy, một vấn đề có đệ qui là vấn đề được định nghĩa lại bằng chính nó.

Để tính  $S(n)$ : ta có kết quả của  $S(1)$ , thay nó vào  $S(2)$ , có  $S(2)$  ta thay nó vào  $S(3)$  ...., cứ như vậy có  $S(n-1)$  ta sẽ tính được  $S(n)$

**\*Một số ví dụ**

**1. Hàm giai thừa:**

$$\begin{aligned}
 n! &= \begin{cases} 1*2*3*\dots*(n-1)*n, & n>0 \\ 1, & n=0 \end{cases} \\
 &= \begin{cases} n*(n-1)! & , n>0 \\ 1 & , n=0 \end{cases}
 \end{aligned}$$

**Nhân xét:**

- Theo công thức trên, ta nhận thấy trong định nghĩa của giai thừa ( $n!$ ) có định nghĩa lại chính nó nên hàm giai thừa có đệ quy.
- Với  $n \geq 0$ , điều kiện dừng tính hàm giai thừa là  $n=1$

**2. Hàm FIBONACCI:**

$$F_n = \begin{cases} 1 & ; n=0,1 \\ F_{n-1} + F_{n-2} & ; n>1 \end{cases}$$

- Theo định nghĩa trên, hàm Fibonacci có lời gọi đệ quy.
- Quá trình tính dừng lại khi  $n=1$

**III.2.2. Hàm đệ quy trong ngôn ngữ C:**

Ngôn ngữ C có trang bị cơ chế gọi hàm đệ quy. Hàm đệ quy là hàm gọi đến chính hàm đó một cách trực tiếp hay gián tiếp.

**Ví dụ 1:** Viết hàm đệ quy tính  $S(n) = 1 + 2 + 3 + \dots + n$

```
#include <stdio.h>
#include <conio.h>
int S(int n)
{ if (n==1)                // điều kiện dừng
    return 1;
  else                    // bước đệ quy
    return (S(n-1) + n);
}
void main()
{ int n;
  printf("\n Nhập n = ");
  scanf("%d",&n);
  printf("Tổng S = 1 + 2 + ... + %d = %d",n, S(n));
  getch();
}
```

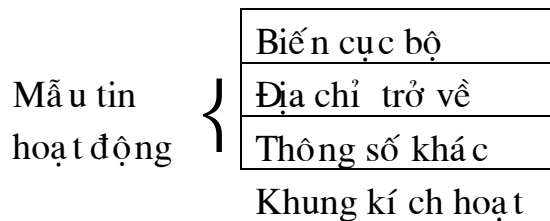
**Ví dụ 2 :** Viết hàm đệ quy tính hàm giai thừa  $n$ .

```
long giai thua(int n)
{ return ((n==0) ? 1 : n*giai thua(n-1));
}
```

**III.2.3. Hàm đệ quy và Stack:**

Một chương trình C thường gồm có hàm main() và các hàm khác. Khi chạy chương trình C thì hàm main() sẽ được gọi chạy trước, sau đó hàm main() gọi các hàm khác, các hàm này trong khi chạy có thể gọi các hàm khác nữa. Khi

một hàm được gọi, thì một khung kí ch hoạt của nó được tạo ra trong bộ nhớ stack. Khung kí ch hoạt này chứa các biến cục bộ của hàm và mẫu tin hoạt động của hàm. Mẫu tin hoạt động chứa địa chỉ trở về của hàm gọi nó và các tham số khác.

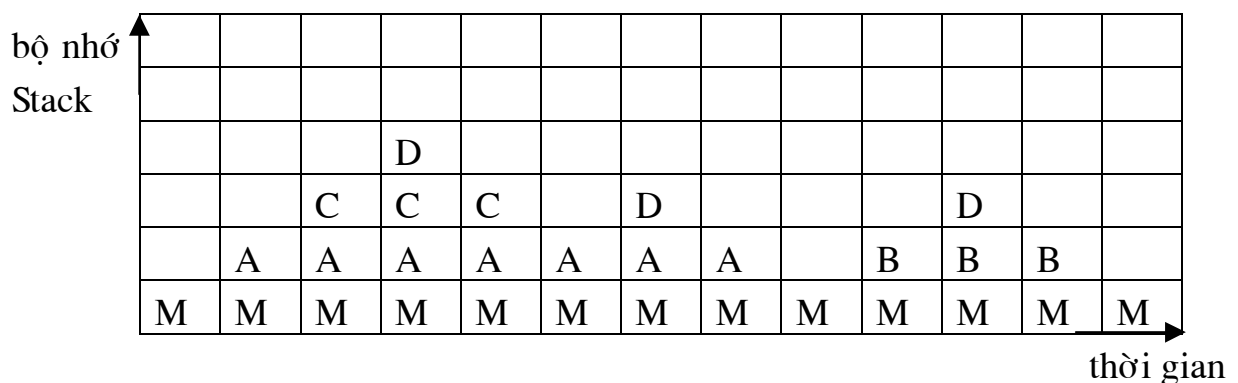


Sau khi hàm được gọi đã thì hành xong thì chương trình sẽ thực hiện tiếp dòng lệnh ở địa chỉ trở về của hàm gọi nó, đồng thời xóa khung kí ch hoạt của hàm đó khỏi bộ nhớ.

Giả sử ta có cơ chế gọi hàm trong một chương trình C như sau:

main()	A()	B()	C()	D()
{ .....	{.....;	{.....;	{.....;	{.....;
A();	C();	D();	D();	.....;
.....;	.....;	}	.....;	}
B();	D();		}	
.....;	}			
}				

Hình sau đây cho ta thấy sự chiếm dụng bộ nhớ stack khi chạy chương trình C như mô tả ở trên.



Tương tự với trường hợp hàm đệ qui, khi gọi đệ qui lẫn nhau thì một loạt các khung kí ch hoạt sẽ được tạo ra và nạp vào bộ nhớ Stack. Cấp đệ qui càng cao thì số khung kí ch hoạt trong Stack càng nhiều, do đó, có khả năng dẫn đến tràn Stack (Stack overflow). Trong nhiều trường hợp khi lập trình, nếu có thể được ta nên gỡ đệ qui cho các bài toán.

#### IV. STRUCTURE:

Các kiểu đơn giản tại một thời điểm chỉ lưu giữ được một giá trị duy nhất. Còn một biến kiểu mảng dùng để lưu trữ các giá trị cùng kiểu dữ liệu với nhau, chẳng hạn như một dãy số, một dãy các ký tự,... Nhưng trong thực tế, điều này vẫn chưa đủ vì các thành phần mà ta lưu giữ thường là khác kiểu dữ liệu với nhau.

Ví dụ : Ta muốn lưu giữ các thông tin về một sinh viên như sau : MASO, HO, TEN, NGAYSINH, NOISINH, PHAI, DIACHI, LOP . Với các thành phần như vậy, thì rõ ràng các thành phần của 1 sinh viên không thể cùng kiểu mà phải thuộc các kiểu khác nhau, cụ thể là :

- MASO, HO, TEN : mảng chữ
- NGAYSINH : int ngày , tháng , năm ;
- NOISINH : mảng chữ
- PHAI : unsigned int;
- LOP : mảng chữ;

Do đó, để lưu trữ được các thành phần khác nhau của một đối tượng ta phải sử dụng một kiểu dữ liệu trong C là **Structure**. (tương tự như record trong Pascal)

##### IV.1. Định nghĩa:

Một biến có kiểu structure được dùng để lưu trữ một đối tượng có nhiều thành phần. Các thành phần có thể thuộc các kiểu dữ liệu khác nhau.

**IV.2. Khai báo**: Muốn khai báo kiểu **hocvien** dùng để lưu trữ họ, tên, điểm môn TOAN, LY, HOA, ĐTB, Xếp loại của một học viên, ta có :

```
struct hocvien
{
    char ho[30];
    char ten[7];
    float toan, ly, hoa , dtb;
    char xeploai[10];
};
```

- Để khai báo biến **hv** có kiểu **hocvien** :

```
struct hocvien hv;
```

- Để truy xuất tới một thành phần, ta dùng dấu chấm, ví dụ như: **hv.ho** để truy xuất tới họ của học viên.

- \* Khai báo kết hợp: vừa khai báo kiểu structure vừa khai báo biến có kiểu đó.

```
struct hocvien
```

```

{   char ho[30];
    char ten[7];
    float toan, ly, hoa , dtb;
    char xeploai[10];
} hv1, hv2;           // khai báo 2 biến hv1, hv2 cùng kiểu học viên
- Khai báo structure lồng nhau:

```

Ví dụ:

```

void main()
{ struct ngaysinh
{ unsigned int ngay, thang, nam;
};
struct hocvien
{ char ho[30];
  char ten[7];
  struct ngaysinh ngsinh;
  float toan, ly, hoa, dtb;
  char xeploai[10];
} ;
  struct hocvien hv;
}

```

Trong trường hợp này, để truy xuất tới tháng sinh của học viên hv, ta viết như sau: **hv.ngsinh.thang**.

## **V. FILE:**

### **V.1. File văn bản:**

- File văn bản là file được lưu trữ dưới dạng kiểu ký tự
- Có 2 cách truy xuất theo kiểu ký tự.
- Truy xuất theo từng ký tự
- Truy xuất theo từng dòng

#### **V.1.1. Khai báo tập tin:**

Khai báo biến kiểu file:

```
FILE *fptr
```

#### **V.1.2. Mở tập tin:**

```
fptr = fopen ("tên file", "kiểu");
```

- Trong "tên file" , ta có thể chỉ định một đường dẫn đầy đủ như sau



"C:\THUNKTL\VIDU.TXT". Hàm fopen nếu mở file thành công sẽ trả về một con trỏ file cho tập tin "tên file", và con trỏ này được cất giữ trong biến fptr (biến kiểu FILE).

Nếu không có file "tên file" trên đĩa thì hàm fopen sẽ trả về trị NULL (nếu fptr == NULL nghĩa là không có file đó)

- Kiểu: gồm có:

“r” : đọc (file phải có sẵn, nếu không có file, hàm fopen trả về trị NULL)

“w” : ghi (nếu có file sẽ xóa file cũ)

“a” : nối vào cuối tập tin

“r+” : đọc / ghi, tập tin phải có sẵn trên đĩa

“a+” : đọc, ghi và o cuối tập tin, nếu trên đĩa chưa có tập tin thì nó sẽ được tạo ra.

Ví dụ: Đếm số ký tự trong file VB.TXT.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{ FILE *fptr;
  int dem=0;
  char ch;
  if ((fptr = fopen("VB.txt", "r")) == NULL) // mở file để đọc
  {
    printf("File nay khong the mo\n");
    exit(0);      // kết thúc chương trình nh, hàm exit thuộc về stdlib.h
  }
  while (!feof(fptr))
  { ch=fgetc(fptr);          // đọc 1 ký tự trong file fptr ra
    dem++;
  }
  fclose(fptr);
  printf("\nSo ky tu trong file VB.TXT =%d",dem);
  getch();
}
```

### V.1.3. Đóng file:

fclose (fptr)

**V.1.4. Đọc / ghi ký tự:** Cho biến ký tự `charch`;

- Đọc ký tự từ tập tin  
`ch = getc (fptr)`
- Ghi ký tự lên tập tin  
`putc (ch, fptr)`

Ví dụ 1: Tạo 1 file trực tiếp từ bàn phím. Quá trình tạo sẽ dừng lại khi ta ấn phím Enter.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{ FILE *fptr;
  char tenfile[67];
  char ch;
  clrscr();
  printf("Cho biet ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"w"))==NULL)    // mở file mới để ghi
  { printf("Viec tao file co loi\n");
    exit(0);
  }
  while ((ch=getche()) !='\r')
    putc(ch,fptr);
  fclose(fptr);
}
```

Ví dụ 2: In nội dung tập tin ra màn hình

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{ FILE *fptr;
  char tenfile[67];
  char ch;
  clrscr();
  printf("Cho biet ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"r"))==NULL)
```

```
{ printf("Viec mo file co loi\n");  
  exit(0);  
}  
while ((ch=getc(fp)) !=EOF)  
  printf("%c",ch);  
fclose(fp);  
}
```

Ví dụ 3: Chương trình nh đếm số từ trong file

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{ FILE *fptr;
  char tenfile[67];
  int ch;
  int dem=0, tu=0;
  clrscr();
  printf("Cho biet ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"r"))==NULL) // mở file để đọc
  { printf("Viec mo file co loi\n");
    exit(0);
  }
  while ((ch=getc(fptr)) !=EOF)
  { if ((ch>='a' && ch <='z') || (ch>='A' && ch<='Z'))
    tu=1;
    if ((ch==' ' || ch=='\n' || ch=='\t') && tu)
    { dem++;
      tu=0;
    }
  }
  printf("So tu trong file =%d",dem);
  fclose(fptr);
}

```

**V.1.5. Đọc / ghi chuỗi ký tự:**

\* Hàm **fgets** (chuỗi, chiều dài, fptr);

Hàm fgets đọc 1 chuỗi ký tự từ trong file fptr và o biến <chuỗi> với chiều dài tối đa là <chiều dài>. Hàm này trả về NULL khi đã đọc hết file

\* Hàm **fputs** (chuỗi, fptr): ghi 1 chuỗi ký tự trong <chuỗi> và o file fptr. Hàm này không tự động thêm và o mã kết thúc để chuyển dòng mới, do đó ta phải ghi thêm mã này và o tệp tin bằng lệnh **fputs ("\n", fptr);**

Ví dụ 1: Chương trình nh ghi chuỗi lên file, cho đến khi chuỗi nhập và o là rỗng thì kết thúc.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
void main()
{ FILE *fptr;
  char tenfile[67];
  char chuoi[80];
  clrscr();
  printf("Cho biet ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"w"))==NULL)    // tạo file mới
  { printf("Viec tao file co loi\n");
    exit(0);
  }
  while (strlen(gets(chuoi)) > 0)    // hàm strlen() trong <string.h>
  { fputs(chuoi,fptr);
    fputs("\n",fptr);
  }
  fclose(fptr);
}
```

Ví dụ 2: Đọc các chuỗi ký tự từ tập tin, và in nó trên màn hình nh.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{ FILE *fptr;
  char tenfile[67];
  char chuoi[81];
  clrscr();
  printf("Cho biet ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"r"))==NULL)
  { printf("Viec tao file co loi\n");
    exit(0);
  }
  while (fgets(chuoi,80,fptr)!= NULL)
    printf("%s",chuoi);
  fclose(fptr); getch();
}
```

**V.1.6. Xóa file:** Lệnh remove xóa file được chỉ định qua <tenfile>

**Cú pháp:** remove (ten file)

Hàm remove trả về 0 : xóa thành công

trả về -1 : có lỗi khi xóa file, và lúc này biến **errno** có 1 trong

2 giá trị sau:

ENOENT : không tìm thấy file muốn xóa

EACCES : không cho phép xóa file mà bạn chỉ định

**Lưu ý:** File nên đóng trước khi xóa.

**Ví dụ:** Xóa file do ta chỉ định.

```
#include <stdio.h>
void main()
{ char filename[80];
  /* prompt for file name to delete */
  printf("File muốn xóa: ");
  gets(filename);
  /* delete the file */
  if (remove(filename) == 0)
    printf("Removed %s.\n", filename);
  else
    perror("remove");    // in thông báo lỗi mà hàm remove gây ra
}
```

## **V.2. File nhị phân (file có cấu trúc)**

File nhị phân là file dùng để lưu trữ các cấu trúc dưới dạng struct hoặc union

### **V.2.1. Khai báo:**

FILE \* fptr;

### **V.2.2. Mở file:**

fptr = fopen (tenfile, "kiểu");

. rb ( b: binary): mở chỉ để đọc

. wb : để ghi. Nếu file đã có thì xóa trước khi mở.

. ab : nối thêm; mở để ghi thêm vào cuối file, nếu file chưa có thì tạo mới

. rb+ : mở file đã có để cập nhật (đọc/ghi)

. wb+ : tạo file mới cho phép đọc/ghi

. ab+ : mở để nối thêm vào cuối file, cho phép đọc/ghi

**V.2.3. Đóng file:**

fclose (fptr)

**V.2.4. Đọc/ghi file:** Hàm **fread** : đọc số mẫu tin(cấu trúc) trong file fptr và o <biến cấu trúc>.

**fread (&biến cấu trúc, sizeof (biến cấu trúc) , số cấu trúc, fptr);**

Hàm fread trả về số cấu trúc đọc được

Hàm **fwrite** ghi dữ liệu trong <biến cấu trúc> vào file fptr.

**fwrite (&biến cấu trúc, sizeof (biến cấu trúc) , số cấu trúc, fptr);**

Hàm fwrite trả về số cấu trúc ghi được lên file

**Chú ý:**

- Để kiểm tra việc đọc file ta kiểm tra số cấu trúc được đọc. Nếu số cấu trúc trả về bằng 0 mà ta cần đọc là 1 cấu trúc thì điều đó chứng tỏ đã hết file.

**\* Ghi một mảng cấu trúc lên file**

fwrite(tên mảng, sizeof (tên mảng), 1, fptr);

⇔ for (i= 0; i< n; i++)

fwrite (&tên mảng[i], sizeof (tên mảng[i] , 1, fptr);

Ví dụ 1: Chương trình ghi lên file nhị phân

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

```
#include <stdlib.h>
```

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

```
void main()
```

```
{ struct hocvien
```

```
{ char hoten[30];
```

```
int tuoi;
```

```
} hv;
```

```
FILE *fptr;
```

```
char tenfile[67];
```

```
char tuoi[3];
```

```
printf("Nhap ten file :");
```

```
gets(tenfile);
```

```
if ((fptr=fopen(tenfile,"wb")) == NULL) // mở file nhị phân để ghi
```

```
{ printf ("Khong the tao file\n"); exit(0);
```

```
}
```

```
do
```

```
{ printf("Nhap ho ten hoc vien :");
```

```

    gets(hv.hoten);
    if (strlen(hv.hoten) !=0)
    { printf("Nhap tuoi :");
      gets(tuoi);
      hv.tuoi = atoi(tuoi);           // macro doi chuoi qua so nguyen
      fwrite(&hv, sizeof(hv), 1, fptr) ; // ghi noi dung 1 mau tin trong bien hv
                                          // vao file fptr
    }
  }
  while (strlen(hv.hoten)!=0);
  fclose (fptr);
}

```

**Ví dụ 2:** Ghi dữ liệu mảng vào file nhị phân

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main()
{ struct hocvien
  { char hoten[30];
    int  tuoi;
  } hv;
  struct hocvien table[3];
  FILE *fptr;
  char tenfile[67];
  char tuoi[3];
  int i=0;
  printf("Nhap ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"wb")) == NULL)
  { printf ("Khong the tao file\n"); exit(0);
  }
  do
  { printf("Nhap ho ten hoc vien :");
    gets(hv.hoten);
    printf("Nhap tuoi :");
    gets(tuoi);
    hv.tuoi = atoi(tuoi);           // macro doi chuoi qua so nguyen
    table[i++]=hv;
  }
}

```



```

    }
    while (i<3);
    fwrite(table, sizeof(table), 1, fptr) ; // ghi noi dung toan bo hoc vien trong
                                         // table vao file fptr

// hoặc for (i=0; i<3; i++)
//    fwrite(&table[i], sizeof(table[i]), 1, fptr)
    fclose (fptr);
}

```

**Ví dụ 3:** Chương trình nh đọc file nhị phân, và in danh sách học viên ra màn hình nh.

```

// In danh sách học viên ra màn hình nh
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main()
{ struct hocvien
  { char hoten[30];
    int tuoi;
  } hv;
  FILE *fptr;
  char tenfile[67];
  char tuoi[3];
  printf("Nhap ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"rb")) == NULL) // Mở file để đọc
  { printf ("Khong the mo file\n"); exit(0);
  }
  clrscr();
  printf("    Ho va ten    Tuoi");
  while (fread(&hv,sizeof(hv),1,fptr) ==1)
  {
    printf("\n%-20s",hv.hoten);
    printf("%3d",hv.tuoi);
  }
  fclose (fptr);
}

```

**V.2.5. Truy xuất tập tin ngẫu nhiên:** (điều khiển con trỏ tập tin trên file nhị phân)

\* Con trỏ file: Mỗi tập tin đều có con trỏ file sau khi được mở. Con trỏ file là con trỏ chỉ đến từng byte trên file. Khi đọc hay ghi dữ liệu trên tập tin, ta đã làm dịch chuyển con trỏ file một số byte, đây chính là số byte mà kiểu dữ liệu đã chiếm. Khi đóng rồi mở tập tin, con trỏ file luôn ở đầu tập tin; ngoại trừ trường hợp ta mở bằng tùy chọn 'a' thì con trỏ file sẽ ở cuối tập tin để ghi thêm dữ liệu vào cuối tập tin. Hàm fseek cho phép ta di chuyển con trỏ file đến vị trí mong muốn.

### Cú pháp:

```
int fseek (FILE * fptr, long nbytes, kiểu)
```

+ nbytes : số bytes tính từ vị trí kiểu cho đến vị trí cần tới

+ kiểu là số nguyên :

kiểu = 0 (tính từ đầu tập tin)

kiểu = 1 (tính từ vị trí hiện tại)

kiểu = 2 (tính từ cuối tập tin)

Nếu fseek trả về 0 nghĩa là nó đã di chuyển tới vị trí đó.

Lưu ý: số thứ tự trên tập tin tính từ 0.

Ví dụ: Viết chương trình truy xuất ngẫu nhiên một mẫu tin theo số thứ tự của nó trong file nhị phân

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main()
{ struct hocvien
  { char hoten[30];
    int tuoi;
  } hv;
  FILE *fptr;
  char tenfile[67];
  int stt, sobytes;

  printf("Nhap ten file :");
  gets(tenfile);
  if ((fptr=fopen(tenfile,"rb")) == NULL)
  { printf ("Khong the mo file\n"); exit(0);
  }
  clrscr();
```

```

printf("Cho biet so thu tu mau tin can di chuyen den :");
scanf("%d",&stt);
sobytes = stt * sizeof(hv);
if (fseek(fptr,sobytes,0)!=0)
{ printf ("Khong the di chuyen con tro file toi vi tri ban chi dinh duoc");
  exit(0);
}
fread(&hv,sizeof(hv),1,fptr) ;
printf("\n%-20s",hv.hoten);
printf("%3d",hv.tuoi);
fclose (fptr);
getch();
}

```

### **V.3. Phát hiện lỗi khi truy xuất tập tin**

Phần lớn những hàm xuất, nhập tập tin chuẩn không thông báo rõ nội dung của lỗi. Chẳng hạn như:

- putc () sẽ trả về EOF khi có lỗi hoặc cuối tập tin
- fgets () sẽ trả về là NULL khi đọc hết file hoặc khi có lỗi

Do đó, để phát hiện lỗi khi truy xuất tập tin, ta dùng macro ferror (FILE \*fptr)

```
int    ferror (file * ptr)
```

Macro ferror trả về một trị khác 0 nếu phát hiện ra lỗi trên file fptr.

\* Để xuất câu thông báo lỗi ta dùng hàm perror ()

```
void perror (const char * str)
```

với str : chuỗi ký tự chứa câu thông báo

\* Hai hàm này thường được sử dụng ngay sau khi sử dụng các hàm đọc / ghi file

Ví dụ: fwrite (&table, sizeof (table), 1, fptr);

```

if (ferror (fptr) != 0)
{ perror ("Loi ghi du lieu");
  exit (0);
}

```

Ví dụ :

```

#include <stdio.h>
void main()
{
  FILE *fp;

```

```

fp = fopen("perror.dat", "r");
if (!fp)
    perror("Không thể mở file để đọc");
}

```

Khi chạy chương trình này, nếu trên đĩa chưa có tập tin perror.dat thì sẽ hiện thông báo lỗi: Không thể mở file để đọc: No such file or directory.

## **Bài tập**

### 1. Tạo tập tin điện tích.h

```

#define Pi 3.14
#define dthv(x) x*x
#define dtht(x) (Pi*x*x)

```

2. Viết chương trình tính diện tích dựa vào file dientich.h ở trên
3. Viết hàm đệ qui tính tích  $P(n) = 1 * 2 * 3 * \dots * n$ ,  $n > 0$
4. Viết hàm đệ qui tính hàm mũ  $x^n$ , với  $n$  nguyên.
5. Viết hàm đệ qui tính phần tử thứ  $n$  của hàm Fibonacci.
6. Viết hàm đệ qui giải quyết bài toán Tháp Hà Nội. Có 3 cột A, B, C. Cột A hiện đang có  $n$  đĩa kích thước khác nhau, đĩa nhỏ ở trên đĩa lớn ở dưới. Hãy dời  $n$  đĩa từ cột A sang cột C (xem cột B là cột trung gian) với điều kiện mỗi lần chỉ được dời 1 đĩa và đĩa đặt trên bao giờ cũng nhỏ hơn đĩa đặt dưới.
7. Viết chương trình mã hóa và giải mã một file văn bản sao cho nếu ta đã mã hóa rồi thì chương trình không mã hóa nữa, và nếu file đó chưa mã hóa thì không được giải mã.
8. Cho biết trong một file văn bản do ta nhập vào có bao nhiêu ký tự, bao nhiêu từ, và bao nhiêu dòng; biết rằng các từ cách nhau khoảng trắng, dấu tab, dấu chấm.
9. Viết chương trình tạo một menu thực hiện các chức năng sau trên file văn bản:
  - Tạo file mới
  - Đọc file
  - Xóa file
  - Ghi nối đuôi file
  - Copy file
  - Di chuyển file từ thư mục này sang thư mục khác
  - Tìm một từ xuất hiện bao nhiêu lần trong file (không phân biệt chữ in, chữ

thường)

- Thay thế từ này bằng từ khác

10. Tạo menu thực hiện các công việc sau:

- Nhập danh sách có kiểu học viên và vào một file tên 'HOSO.TXT', mỗi học viên có các thông tin sau: maso (int), hoten (chuỗi tối đa 30 ký tự), phái (NAM/NU), tuổi (int).

- Liệt kê danh sách học viên ra màn hình theo dạng sau:

Mã số	Họ và tên	Phái	Tuổi
-------	-----------	------	------

- Truy xuất ngẫu nhiên theo thứ tự mẫu tin
- Tìm kiếm một người trong file theo mã số
- Cập nhật sửa đổi các mẫu tin theo mã số (Nhập mã số, sau đó hiệu chỉnh lại hoten, phái, và tuổi).
- Xóa một người trong file theo mã số.

## CHƯƠNG 3 CÁC THUẬT TOÁN TRÊN CẤU TRÚC DỮ LIỆU MẢNG

### I. MẢNG KHÔNG SẮP XẾP VÀ THUẬT TOÁN TÌM KIẾM TRÊN MẢNG CHƯA CÓ THỨ TỰ

#### I.1. Một số khái niệm về mảng:

##### I.1.1. Định nghĩa a:

Mảng là 1 dãy các phần tử có cùng kiểu dữ liệu được sắp xếp liên tiếp nhau trong bộ nhớ

0100		int
0102	1	
0104	2	Mảng n phần tử
	n-1	

Bộ nhớ

#### ↪ Khai báo:

**Cú pháp:** Khai báo mảng 1 chiều

Kiểu\_DL Tên\_mảng [kích\_thước];

♦ Kiểu\_DL : là 1 trong các kiểu dữ liệu cơ bản, đó là kiểu của phần tử của mảng

♦ Tên\_mảng: là tên của mảng được đặt 1 cách hợp lệ

♦ Kích\_thước: là 1 hằng nguyên cho biết số phần tử tối đa của mảng

Ví dụ 1: Khai báo 1 mảng số nguyên

- int n ;  
int M[n] ; SAI
- int M[10] ; đúng vì kích thước mảng phải là hằng không phải là biến
- #define max 100  
int M[max] ;

Ví dụ 2: Khai báo 1 danh sách họ tên học viên của 1 lớp học

```
char dshv[50][30]; // dshv có thể chứa tối đa họ tên 50 học viên,
// chiều dài họ tên mỗi học viên tối đa là 30 ký tự
```

**Cú pháp:** Khai báo mảng 2 chiều

Kiểu\_DL Tên\_mảng [kích thước 1][kích thước 2]

**Chú ý:** Một mảng trong C, các phần tử được đánh số từ 0 tới n-1

**Ví dụ:** Với M[10]

thì thành phần thứ 1 là M[0]

thành phần cuối cùng M[9]

\* C không bắt buộc, không kiểm tra xem biến đếm có vượt ra khỏi giới hạn cho phép của mảng chưa. Do đó, chúng ta phải kiểm tra biến đếm trong chương trình (phải nhỏ hơn n)

### I.1.2. Khởi động trị cho mảng:

Ta khởi động được trị cho mảng trong 2 trường hợp sau:

- Mảng được khai báo là biến ngoài (main) nghĩa là biến toàn cục
- Mảng được khai báo cục bộ

**Ví dụ 1:**     int     M[3] = {10,11,12}

main()

{  
}

**Ví dụ 2:**

main()

{ static     int     M[ ]={10,22,30};

.....

}

- Ta có thể gán 1 hằng cho cả mảng như sau:  
memset (M,0,sizeof(int) \*3) ; // gán 0 cho mảng M với M có 3 phần tử
- Từ khóa static dùng để khai báo 1 biến cục bộ thường trực cho phép duy trì giá trị riêng của nó ở những lần gọi hàm sau này.

- Khởi tạo mảng 2 chiều:

int     M[2][3]= {{1,2,3},  
                  {0,1,0}};

### I.1.3.Truy xuất thành phần của mảng: M[chỉ số]

- Truy xuất thành phần thứ 2 của mảng 1 chiều: M[1]
- Truy xuất thành phần thứ i của mảng 1 chiều: M[i-1]
- Truy xuất thành phần dòng 2, cột 3 của mảng 2 chiều M[1][2]

### I.1.4. Đọc (nhập) dữ liệu cho mảng:

- Để nhập dữ liệu cho mảng ta phải nhập dữ liệu cho từng thành phần của mảng.

**Ví dụ 1:**

```

int    n,i;
float  M[10];
printf("\nCho biet so phan tu cua mang:")
scanf ("%d",&n);
for ( i=0; i< n; i++)
{ printf("a[%d]= ",i+1);
  scanf ("%f",&M[i]);
}

```

**Ví dụ 2:** Nhập và o mảng 2 chiều.

```

int m, n, i, j;
float M[10] [10];
printf("So dong ="); scanf("%d",&n);
printf("So cot ="); scanf("%d",&m);
for(i= 0; i< n; i++)
  for(j= 0; j<m; j++)
    { printf("M[%d] [%d] = ",i,j);
      scanf("%f", &M[i][j]);
    }

```

**I.1.5. Xuất dữ liệu u kiểu u mảng:** Để xuất dữ liệu u mảng ta cũng phải xuất dữ liệu u của từng thành phần n mảng

**Ví dụ:**

```

int i, n;
float M[10];
for(i = 0; i< n; i++)
  printf("a[%d] = %f",i+1, M[i]);

```

## **I.2. Thuật toán tìm kiếm trên mảng chưa có thứ tự:**

Do mảng chưa có thứ tự nên ta áp dụng phương pháp tìm kiếm tuyến tính tìm từ đầu u mảng cho đến cuối u mảng. Trong chương trình sau đây, hàm Timkiem sẽ trả về trị -1 nếu không có mã sinh viên n trong danh sách ds, ngược lại hàm sẽ trả về vị trí của mã số đó trong danh sách ds.

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

```

```

#define MAX_SOSV 100 // số sinh viên tối đa trong danh sách
typedef struct sinhvien // định nghĩa u kiểu sinhvien

```



```
{ char maso[6];
  char hoten[30];
};
```

```
typedef struct danhsach_sv    // định nghĩa a kiểu u danhsach_sv
{ int tssv;
  sinhvien sv[MAX_SOSV];
} ;
```

```
void Nhap_ds (struct danhsach_sv *psv)
```

```
{ char sosv[4];
  printf("Số sinh viên muốn nhập :");
  gets(sosv);
  psv->tssv=atoi(sosv);
  for (int i=0; i<psv->tssv; i++)
  { printf("Mã số :");
    gets(psv->sv[i].maso);
    printf("Họ tên :");
    gets(psv->sv[i].hoten);
  }
}
```

```
void Lietke_ds (struct danhsach_sv *psv)
```

```
{ int i=0;
  clrscr();
  printf ("   Mã số   Họ & tên \n");
  while (i < psv->tssv)
  { printf ("%8s   %-s\n", psv->sv[i].maso,psv->sv[i].hoten);
    i++;
  }
  getch();
}
```

```
/* Hàm Timkiem tìm maso trong danhsach *psv */
```

```
int Timkiem(danhsach_sv *psv, char maso[])
```

```
{ int i=0;
  while ((i<psv->tssv) && (strcmp(psv->sv[i].maso, maso)!=0))
    i++;
  return (i==psv->tssv ? -1 : i) ;
```

```

}
void main()
{ struct danh sach_sv ds;
  char maso[6];
  int vitri;
  Nhap_ds(&ds);           // Gọi hàm Nhap_ds với tham số là ds
  Lietke_ds(&ds);
  printf("Ma so sinh vien ban can tim :");
  gets(maso);
  vitri = Timkiem(&ds, maso);
  if (vitri != -1)
    printf("Ho ten cua sinh vien la %s", ds.sv[vitri].hoten);
  else printf(" Khong co sinh vien voi ma ban nhap vao");
  getch();
}

```

## II. CÁC THUẬT TOÁN SẮP XẾP:

Trong thực tế cuộc sống cũng như trong lĩnh vực lập trình, việc quản lý dữ liệu thường đòi hỏi sự tìm kiếm các dữ liệu cần thiết; Để thuận tiện cho việc tìm kiếm, dữ liệu thường được sắp xếp theo một thứ tự nào đó.

Có rất nhiều phương pháp sắp thứ tự, trong bài giảng này ta chỉ khảo sát hai phương pháp sắp xếp là Bubble\_Sort và Quick\_Sort.

Để thuận tiện ta giả sử mảng là dãy số có tối đa 100 số, và các thuật toán dưới đây dùng để sắp xếp dãy số theo thứ tự tăng dần.

### II.1. Sắp xếp theo phương pháp Bubble Sort (phương pháp nổi bọt)

- Nội dung: Ta cho i duyệt dãy  $a[0], \dots, a[n-1]$ ; nếu  $a[i-1]$  lớn hơn  $a[i]$  thì ta hoán đổi ( $a[i-1], a[i]$ ). Lặp lại quá trình duyệt dãy này cho đến khi không có xảy ra việc đổi chỗ của hai phần tử.

Ví dụ: Ta sắp thứ tự dãy số sau : 26 33 35 29 19 12 32

Bước 0	1	2	3	4	5	6
26	12	12	12	12	12	12
33	26	19	19	19	19	19
35	33	26	26	26	26	26
29	35	33	29	29	29	29
19	29	35	33	32	32	32

12	19	29	35	33	33	33
32	32	32	32	35	35	35

- Chương trình nh:

```
#include <stdio.h>
#include <conio.h>
int mang[100];           // biến toàn cục
int size ;
void Bubble_Sort(int A[100], int n)
{ int i,j,temp;
  for (i=1; i<n; i++)
    for (j=n-1; j>=i; j--)
      if (A[j-1] > A[j])
      {   temp = A[j-1];
          A[j-1] = A[j];
          A[j] = temp;
      }
}

int Nhap_day_so (int A[])
{ int i,n;
  printf("\nSo phan tu cua mang :"); scanf("%d",&n);
  for (i=0; i<n; i++)
  { printf ("A[%d] = ",i+1);
    scanf("%d",&A[i]);
  }
  return n;
}

void Liet_ke (int A[], int n)
{ int i;
  printf("\n Gia tri mang da duoc sap : \n");
  for (i=0; i<n; i++)
    printf ("%5d",A[i]);
  getch();
}

void main()
{
  size= Nhap_day_so(mang);
```

```

    Bubble_Sort(mang,size);
    Liet_ke(mang,size);
}

```

Ta nhận thấy phương pháp này có thể được cải tiến dễ dàng. Nếu ở lần duyệt dãy nào đó mà không có sự đổi chỗ giữa hai phần tử thì dãy đã có thứ tự và giải thuật kết thúc. Trong trường hợp này, ta dùng một cờ hiệu flag để ghi nhận điều này, và giải thuật Bubble Sort được cải tiến như sau:

```

#define FALSE 0
#define TRUE 1

void Bubble_Sort_Ad(int A[], int n)
{ int i,temp;
  unsigned char flag=TRUE;
  while (flag)
  { flag = FALSE ;
    for (i=0; i<n-1; i++)
      if (A[i] > A[i+1])
      { temp = A[i];
        A[i] = A[i+1];
        A[i+1] = temp;
        flag=TRUE;
      }
  }
}

```

## II.2. Sắp xếp theo phương pháp Quick Sort

**II.2.1. Nội dung:** Chọn một phần tử bất kỳ trong danh sách làm điểm chốt x, so sánh và đổi chỗ những phần tử trong danh sách này để tạo ra 3 phần: phần có giá trị nhỏ hơn x, phần có giá trị bằng x, và phần có giá trị lớn hơn x. Lại tiếp tục chia 2 phần có giá trị nhỏ hơn và lớn hơn x theo nguyên tắc như trên; quá trình chia phần sẽ kết thúc khi mỗi phần chỉ còn lại một phần tử, lúc này ta đã có một danh sách có thứ tự.

Ví dụ: Xét dãy 26 33 35 29 19 12 32

• Lần chia phần thứ nhất : Chọn phần tử chốt có khóa là 29, đặt là x

26	33	35	<u>29</u>	19	12	32
$i \rightarrow$					$\leftarrow j$	

Dùng hai biến chỉ số i và j để duyệt từ hai đầu danh sách đến x. Nếu i gặp

phần tử lớn hơn hay bằng x sẽ dừng lại, j gặp phần tử nhỏ hơn hay bằng x sẽ dừng lại, rồi đổi chỗ hai phần tử này; sau đó tiếp tục duyệt cho đến khi  $i > j$  thì ngừng lại.

Lúc này dãy sẽ có 3 phần khác nhau như hình vẽ sau :

26	33	35	<u>29</u>	19	12	32
	i				j	
26	12	35	<u>29</u>	19	33	32
		i		j		
26	12	19	<u>29</u>	35	33	32
			ij			
26	12	19	<u>29</u>	35	33	32
		j		i		

• Lần chia phần thứ hai cho dãy con 26 12 19, chọn chốt  $x=12$

26	<u>12</u>	19	→	<u>12</u>	26	19
i	j			j	i	

Kết thúc ta sẽ có hai phần : 12 ; 26 19

• Lần chia phần thứ 3 cho dãy con 26 19, chọn chốt  $x=26$

<u>26</u>	19	→	19	<u>26</u>	Kết thúc quá trình chia nhỏ dãy con 26 12 19
i	j		j	i	

- Lần chia phần thứ 4 cho dãy con 35 33 32, chọn chốt  $x=33$

35	<u>33</u>	32	→	32	<u>33</u>	35	→	32	33	35
i		j			ij			j		i

Kết thúc ta sẽ có ba phần : 32 ; 33 ; 35

Đến đây quá trình chia phần kết thúc vì tất cả các phần chỉ có một phần tử, lúc này ta sẽ có một danh sách có thứ tự là :

12 19 26 29 32 33 35

## II.2.2. Giải thuật:

### a. Giải thuật không đệ quy:

- Ta tạo một Stack, mỗi phần tử của Stack có 2 thành phần là q, r chứa chỉ số đầu và chỉ số cuối của dãy cần sắp. Ban đầu,  $\text{Stack}[0].q = 0$  và  $\text{Stack}[0].r = n-1$

- Tiến hành phân hoạch dãy số gồm các số bắt đầu từ chỉ số q đến chỉ số r

- Sau mỗi lần chia phần, ta kiểm tra xem phần có giá trị nhỏ hơn chốt và phần có giá trị lớn hơn chốt nếu có từ 2 phần tử trở lên thì đưa vào Stack. Sau mỗi lần phân hoạch, ta lại lấy dãy số mới từ Stack ra phân hoạch tiếp.

- Quá trình nh cứ như thế cho tới khi Stack rỗng thì kết thúc.

\* Chương trình nh:

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
int mang[100];
int size ;
void Quick_Sort(int A[100], int n)
{ struct Element_Stack      // kiểu phần tử trong Stack
{
    int q, r;
};
Element_Stack Stack[50];    // Stack có tối đa 50 phần tử
int sp=0;                   // con trỏ Stack, khởi tạo sp=0
int i,j,x,q,r,temp;
Stack[0].q =0 ;             // chỉ số đầu của mảng cần sắp
Stack[0].r =n-1;            // chỉ số cuối của mảng cần sắp

do
{ // Lấy một phần hoạch ra từ Stack
  q = Stack[sp].q ; r =Stack[sp].r ;
  sp--;                      // Xóa 1 phần tử khỏi Stack
do
{ // Phân đoạn dãy con a[q] ,..., a[r]
  i = q; j =r;
  x = A[(q+r) / 2] ; // Lấy phần tử giữa của dãy cần sắp thứ tự là m chốt
do
{ while (A[i] < x) i++; //Tìm phần tử đầu tiên có trị lớn hơn hay bằng x
  while (A[j] > x) j--; //Tìm phần tử đầu tiên có trị nhỏ hơn hay bằng x
  if (i<=j)                // Đổi chỗ A[i] với A[j]
  { temp = A[i];
    A[i] =A[j];
    A[j] = temp;
    i++ ; j--;
  }
} while (i<=j);
```

```

    if (i<r)                // phần thứ ba có từ 2 phần tử trở lên
    { // Đưa vào o Stack chỉ số đầu và chỉ số cuối của phần thứ ba
        sp++;
        Stack[sp].q=i;
        Stack[sp].r=r;
    }
    r = j;    // Chuẩn bị vị trí để phân hoạch phần có giá trị nhỏ hơn chốt
} while (q< r);
} while (sp!=-1); // Kết thúc khi Stack rỗng
}

```

```

int Nhap_day_so (int A[])

```

```

/* Tạo dãy n số ngẫu nhiên từ 0 đến 9999 đưa vào mảng A */

```

```

{ int i,n;
  printf("\nSố phần tử của mảng :"); scanf("%d",&n);
  randomize();                // dùng <time.h> và <stdlib.h>
  for (i=0; i<n; i++)
    A[i]= rand() % 10000; // Phát sinh các số ngẫu nhiên từ 0 đến 9999
  return n;
}

```

```

void Liet_ke (char str[],int A[], int n)

```

```

{ int i;
  printf("\n%s\n",str);
  for (i=0; i<n; i++)
    printf ("%5d",A[i]);
  getch();
}

```

```

void main()

```

```

{
  size= Nhap_day_so(mang);
  Liet_ke("Dãy số ngẫu nhiên :",mang,size);
  Quick_Sort(mang,size);
  Liet_ke("Giá trị mảng đã được sắp :",mang,size);
}

```

b. Giải thuật Quick Sort đệ quy: về cơ chế thực hiện thì cũng giống như

giải thuật không đệ qui, nhưng ta không kiểm soát Stack mà để cho quá trình gọi đệ qui tự tạo ra Stack.

\* Chương trình nh:

```
void Sort(int A[], int q,int r)
{ int temp;
  int i=q;
  int j=r;
  int x = A[(q+r) / 2] ; // Lấy phần tử giữa của dãy cần sắp thứ tự là chốt
do
{ // Phân đoạn dãy con a[q] ,..., a[r]
  while (A[i] < x) i++; //Tìm phần tử đầu tiên có trị lớn hơn hay bằng x
  while (A[j] > x) j--; //Tìm phần tử đầu tiên có trị nhỏ hơn hay bằng x
  if (i<=j) // Đổi cho A[i] với A[j]
  { temp = A[i];
    A[i]=A[j];
    A[j] = temp;
    i++ ; j--;
  }
} while (i<=j);
if (q<j) // phần thứ nhất có từ 2 phần tử trở lên
  Sort(A,q,j);
if (i<r) // phần thứ ba có từ 2 phần tử trở lên
  Sort (A,i,r);
}
void Quick_Sort(int A[], int n)
{ Sort( A,0,n-1); // Gọi hàm Sort với phần tử đầu có chỉ số 0 đến
// phần tử cuối cùng có chỉ số n-1
}
```

### III. TÌM KIẾM TRÊN MẢNG ĐÃ CÓ THỨ TỰ:

Giả sử dãy số của ta là dãy số đã có thứ tự tăng dần, và x là giá trị cần tìm. Các hàm tìm kiếm sẽ trả về trị -1 nếu không có x trong dãy, ngược lại các hàm tìm kiếm sẽ trả về chỉ số của x trong dãy.

#### III.1. Tìm kiếm nhanh bằng phương pháp lặp:

- Nội dung: Do dãy số đã có thứ tự tăng dần, nên nếu ta tìm được phần tử đầu tiên có trị vừa lớn hơn x thì ta có thể kết luận dãy số không chứa trị x. Vì vậy, ta sẽ rút ngắn thời gian tìm kiếm.



**- Giải thuật:**

```

int Search(int A[], int n, int x)
{ int i=0;
  while (i<n && A[i] < x)
    i++;
  return (i<n && A[i]==x ? i : -1) ;
}

void main()
{ int so,vitri;
  size= Nhap_day_so(mang);
  Quick_Sort(mang,size);
  Liet_ke("Gia tri mang da duoc sap :",mang,size);
  printf("\nGia tri muon tim :");
  scanf("%d",&so);
  vitri = Search(mang,size, so);
  if (vitri !=-1)
    printf("Vi tri cua so %d trong day = %d",so,vitri);
  else printf(" Khong co so %d trong day",so);
  getch();
}

```

**III.2. Phép tìm kiếm nhị phân:****- Nội dung:**

➤ Bước 1: Phạm vi tìm kiếm ban đầu là toàn bộ danh sách.

➤ Bước 2: Lấy phần tử chỉ nh giữa của phạm vi tìm kiếm (gọi là y) so sánh với x.

Nếu  $x=y$  thì ta đã tìm thấy, trả về chỉ số. Giải thuật kết thúc

Nếu  $x < y$  thì phạm vi tìm kiếm mới là các phần tử nằm phía trước của y.

Nếu  $x > y$  thì phạm vi tìm kiếm mới là các phần tử nằm phía sau của y.

➤ Bước 3: Nếu còn tồn tại phạm vi tìm kiếm thì lặp lại bước 2, ngược lại giải thuật kết thúc với kết quả là không có x trong dãy số.

**- Giải thuật:**

```

int Binary_Search(int A[], int n, int x)
{ unsigned char found=FALSE; // Giả sử ban đầu ta chưa tìm thấy x trong dãy
  // Phạm vi ban đầu tìm kiếm là từ k=0 → m = n-1

```

```

int k=0;
int m=n-1;
int j;
while (k<=m && !found)
{ j=(k+m) /2;           //chỉ số phần tử giữa
  if (A[j]==x)
    found=TRUE;
  else if (x>A[j]) k=j+1;      // Phạm vi tìm mới là (j+1, m)
    else m=j-1;           // Phạm vi tìm mới là (k, j-1)
}
return (found ? j : -1);
}

```

### III.3. Phép tìm kiếm nhị phân đệ quy:

- Nội dung: tương tự như trên

↳ Bước 1: Phạm vi tìm kiếm ban đầu là toàn bộ danh sách ( $k=0 \rightarrow m=n-1$ ).

↳ Bước 2: Lấy phần tử chính giữa của phạm vi tìm kiếm (gọi là y) so sánh với x.

Nếu  $x=y$  thì ta đã tìm thấy, trả về chỉ số. Giải thuật kết thúc

Nếu  $x < y$  thì phạm vi tìm kiếm mới là các phần tử nằm phía trước của y, nên ta gọi đệ quy với phạm vi mới là  $(k, j-1)$

Nếu  $x > y$  thì phạm vi tìm kiếm mới là các phần tử nằm phía sau của y, nên ta gọi đệ quy với phạm vi mới là  $(j+1, m)$

↳ Điều kiện dừng:  $x=y$  hoặc  $k > m$ .

- Giải thuật:

```

int Binary_Search2(int A[], int k, int m, int x)
{ int j=(k+m) /2;
  if (k>m) return -1;
  else if (A[j]==x) return j;
    else Binary_Search2(A, (A[j]<x ? j+1:k), (A[j] > x ? j-1:m), x);
}

```

**Bài tập:**

1. Cho một dãy n số thực A :
  - a) Tìm phần tử nhỏ nhất của dãy số A
  - b) Tìm phần tử lớn nhất của dãy số A
  - c) Tính giá trị trung bình của dãy số A.
2. Hiện đang lưu hành các tờ giấy bạc 50000đ, 20000đ, 10000đ, 5000đ, 2000đ, 1000đ, 500đ, 200đ, 100đ. Nếu có x đồng, hỏi rằng nên chọn các tờ giấy bạc nào để số lượng các tờ giấy bạc là ít nhất.
3. Viết chương trình nhập một ma trận số nguyên có kích thước M x N. In ra:
  - Tổng các phần tử của ma trận
  - Số các phần tử dương, phần tử âm, phần tử 0 của ma trận
  - Phần tử lớn nhất, nhỏ nhất của ma trận
  - Các phần tử trên đường chéo chính của ma trận (với M = N)
  - Tổng các phần tử trên đường chéo chính của ma trận (với M = N)
4. Cho 2 ma trận vuông A(n,n) và B (n,n) :
  - Tính ma trận tổng  $C = A + B$ ,  
 biết  $C[i,j] = A[i,j] + B[i,j]$ ,  $\forall i,j \in [1,n]$
  - Tính ma trận tích  $D = A * B$ ,  
 biết  $D[i,j] = \sum_{k=1}^n A[i,k] * B[k,j]$  ;  $i, j = 1..n$
5. Tạo một menu thực hiện các công việc sau:
  - a. Nhập danh sách có kiểu học viên và vào một mảng, mỗi học viên có các thông tin sau: maso (int), hoten (chuỗi tối đa 30 ký tự), phái(NAM/NU), điểm (float), hạng (unsigned char). Quá trình nhập sẽ dừng lại khi mã số nhập vào là 0.
  - b. Liệt kê danh sách học viên ra màn hình theo dạng sau:  
 Mã số              Họ và tên              Phái              Điểm              Hạng
  - c. Tìm kiếm một học viên trong danh sách theo mã số, và in ra các thông tin còn lại của học viên đó.
  - d. Sắp xếp danh sách học viên theo điểm tăng dần.
  - e. Xếp hạng danh sách học viên theo qui tắc cùng điểm thì cùng hạng, hạng của học viên sau bằng hạng của nhóm học viên trước cộng số người của nhóm học viên trước cùng điểm.
  - f. Giả sử danh sách học viên đã có thứ tự tăng dần theo điểm; Nhập thêm 1 học viên sao cho sau khi nhập thì danh sách vẫn còn có thứ tự.
  - g. Cập nhật sửa đổi các dữ liệu theo mã số (Nhập mã số, sau đó hiệu chỉnh)

lại hoten, phai, điểm).

h. Loại bỏ 1 học viên ra khỏi danh sách dựa vào mã số.

6. a) Viết chương trình tạo ngẫu nhiên một dãy số 20000 số nguyên có giá trị từ 0 đến 9999.

b) Đếm số lần so sánh của 2 giải thuật tìm kiếm tuần tự (trên dãy số chưa có thứ tự) và tìm kiếm nhị phân (trên dãy số đã được sắp thứ tự).

c) Trong trường hợp dãy số trên là 200000 số nguyên phân biệt khác nhau trong miền giá trị [1..300000] thì ta tối ưu giải thuật sắp xếp về mặt không gian và thời gian như thế nào? Cho biết thời gian thực thi của giải thuật Quick Sort và giải thuật bạn cài đặt.

7. Cho biết thời gian thực thi của 2 giải thuật Bubble Sort và Quick Sort trên dãy số có số phần tử khá lớn.

8. Giả sử ta đã có 1 dãy số thực A tăng dần. Viết giải thuật thêm 1 số thực x vào dãy A sao cho sau khi thêm x thì dãy vẫn tăng dần?

9. Viết hàm xóa tất cả các phần tử có trị bằng x trong dãy số A.

10. Viết chương trình tính điểm của một lớp:

- Nhập các thông tin sau cho mỗi học viên : hoten, namsinh, trung bình HK1, trung bình HK2

- In ra danh sách các học viên của lớp theo thứ tự giảm dần của ĐTB toàn năm

$$TB\ toàn\ năm = (TB\ HK1 + TB\ HK2)/2$$
  
theo mẫu sau:

### DANH SÁCH ĐIỂM LỚP .....

STT	Họ & Tên	TB HK1	TB HK2	TB toàn năm	Hạng
1					
2					
3					

**Lưu ý:**- Các học viên cùng ĐTB thì cùng hạng

- In 17 học viên trên một trang màn hình

11. Cho 2 dãy số A có n phần tử và B có m phần tử với thứ tự tăng dần. Hãy trộn 2 dãy số trên thành 1 dãy mới C sao cho sau khi trộn thì C cũng tăng dần.

## CHƯƠNG 4

CON TRỎ (POINTER)**I. ĐỊNH NGHĨA**

Con trỏ là một kiểu dữ liệu dùng để chứa địa chỉ. Biến con trỏ là một biến chứa địa chỉ của một thực thể nào đó, thực thể đó là biến hoặc là hàm.

Con trỏ thường được dùng để:

- Trả về nhiều trị từ hàm qua cơ chế truyền theo tham số theo địa chỉ trong hàm (tham số hình thức biến).
- Tạo các cấu trúc dữ liệu phức tạp như danh sách liên kết và cây nhị phân.
- Truyền mảng và chuỗi giữa các hàm khá thuận lợi.

**I.1. Khai báo:** Khai báo biến `pi` là con trỏ trỏ đến một số nguyên.

`int *pi;`

Lúc này, `pi` chiếm 2 bytes chứa địa chỉ của số nguyên mà nó đang chỉ đến, đồng thời trình biên dịch của C cũng biết `pi` đang chỉ đến một số nguyên (do khai báo). Để đưa một giá trị nguyên vào vùng nhớ mà `pi` đang trỏ đến, ta dùng lệnh: `*pi = 1;`

Ví dụ:

`void main()`

`{ int x=4, y=10;`

`int *px, *py; // px, py là các biến con trỏ`

`px = &x; // đưa địa chỉ của x, y vào px và py`

`py = &y;`

`*px = *px + *py; // tăng giá trị của vùng nhớ mà px đang trỏ tới`  
`// thêm y, tương đương với x = x+y`

`}`

Minh họa chương trình trên trong bộ nhớ:

Biến		<code>int x=4, y=10;</code> <code>int *px, *py;</code>	<code>px=&amp;x;</code> <code>py=&amp;y;</code>	<code>*px = *px + *py;</code>
x	950	4	4	14
	951			
y	952	10	10	10
	953			
px			950	950
py			952	952

Hình 7.1. Cơ chế truy xuất giá trị qua biến con trỏ.

**Tổng quát:** Kiểu \*biến;

**I.2. Truyền địa chỉ cho hàm:** Trong 1 số trường hợp ta muốn gọi địa chỉ của 1 biến x cho hàm. Nhờ vào cơ chế truyền theo địa chỉ này mà hàm có thể trả về nhiều giá trị cho chương trình gọi.

**Ví dụ:** Hàm hoán đổi giá trị của 2 biến x, y

```
void hoandoi (int *a, int *b)
{
    int tam;
    tam = *a;
    *a = *b;
    *b = tam;
}

void main()
{
    int x,y;
    printf ("x, y = ");
    scanf ("%d %d", &x, &y);
    giaohoa(&x, &y); // Truyền địa chỉ của 2 biến x,y cho hàm hoandoi
}
```

## II CÁC PHÉP TOÁN TRÊN BIẾN CON TRỎ:

### II.1. Toán tử địa chỉ &:

Nếu x là biến thông thường, &x sẽ là địa chỉ của biến x

**Ví dụ:** float x, \*pf;

```
x = 50;
pf = x; // sai vì pf là biến con trỏ nên ta viết pf = &x;
x = pf; // sai ; ta viết x = *pf; { lấy nội dung của pf }
```

### II.2. Toán tử nội dung \*:

Nếu p là pointer thì \*p là nội dung của nó.

**Ví dụ:** int x,y, \*p;

```
x = 50;
p = &x; // p chứa địa chỉ của vùng nhớ x
y = *p; // y = *p = 50 vì p chứa địa chỉ của vùng nhớ x
```

**Ví dụ:** a=2;

```
p = &a;
b = (*p) + + + 3; // b =5, *p = 3, a = 3.
```

(vì p trỏ tới địa chỉ a nên \*p tăng thì a tăng)

**Tóm lại:** \*x là biến mà x giữ địa chỉ  
&x là địa chỉ của x nếu x là biến thông thường

### **II.3. Phép cộng trừ biến con trỏ với một số nguyên:**

Nếu p là biến pointer thì p+n là địa chỉ của một biến mới cách nó n biến theo chiều tăng, còn p-n thì ngược lại.

#### **Chú ý:**

- Phép cộng con trỏ với một số nguyên chỉ được áp dụng trên một dãy biến cùng kiểu

- Không được cộng 2 pointer với nhau

- Không được nhân, chia, lấy dư biến con trỏ với bất kỳ số nào

Ví dụ: Giả sử ta có mảng nums[] = {10,20,30,40,50}. Việc tham khảo tới nums[i] thực chất là dùng dạng ký hiệu con trỏ, vì khi biên dịch, trình biên dịch sẽ chuyển đổi ký hiệu mảng thành ký hiệu con trỏ.

```
void main()
{ static int nums [] = {10,20,30,40,50};
  for (int i =0; i<5; i++)
    printf ("%d\n", *(nums + i));
}
```

Lưu ý: \*(nums+i) tương đương với nums[i]

### **II.4. Phép gán và phép so sánh:**

- **Phép gán:** các biến con trỏ có thể gán cho nhau với điều kiện phải cùng kiểu

Ví dụ: int \*p1, \*p2;  
           \*p1 = 10;  
           p2 = p1;       // lúc này \*p2 = 10;

- **Phép so sánh:** ta có thể so sánh 2 biến con trỏ xem có cùng địa chỉ hay không, đương nhiên 2 biến con trỏ đó phải cùng kiểu với nhau.

### **II.5. Sự chuyển kiểu:**

Cú pháp:    ( Kiểu)       \*tên biến

Ví dụ: int \*p1, num ;  
           float \*p2;  
           num =5;  
           p1 = &num;  
           \*p2 = (float ) \*p1;   //   \* p2 = 5.0

Ví dụ: int num, \*p, n;  
           char c;  
           p = &num;

```
*p = 97;    // num = 97
n = *p;     // n = 97
c = (char) *p; // c = 'a'
```

**Chú ý:** Địa chỉ của một biến được xem như một con trỏ hằng, do đó nó không được phép gán, tăng hoặc giảm.

**Ví dụ:** int num, \*p, n;  
 p = & num;  
 p ++; // đúng  
 ( & num) ++; // sai  
 con trỏ hằng

## **II.6. Khai báo một con trỏ hằng và con trỏ chỉ đến đối tượng hằng:**

### **a. Con trỏ hằng:**

Kiểu \* const p = giá trị;

### **b. Con trỏ chỉ đến đối tượng hằng:**

Kiểu const \*p = giá trị hằng;

hoặc Const kiểu \*p = giá trị hằng;

**Ví dụ:** char \*const p2 = "ABCD"  
 const char \*p1 = "ABCD"  
 p2 ++; // sai  
 p1 ++; // đúng; \*p1 = 'B' ; p1 = "BCD"

## **III. SỰ TƯƠNG QUAN GIỮA CON TRỎ VÀ MẢNG**

**Ví dụ:** Ta có mảng A như sau:

```
int A[10], *p;  
thì A = &A[0]
```

Nếu p = A thì để truy xuất tới phần tử thứ i của mảng A, ta có các cách sau:

$$A[i] \Leftrightarrow *(A + i) \Leftrightarrow *(p + i)$$

$$\& A[i] \Leftrightarrow (A + i) \Leftrightarrow (p + i)$$

**Ví dụ:** Nhập mảng A:

```
int A[10], *p, i;  
p = A;  
for (i = 0; i < 9; i++)  
    scanf ("%d", p+i);
```

**Xuất mảng A:**

```
for (i = 0; i < 9; i++)  
    printf ("%d", *(p+i));
```



Ví dụ: Sắp xếp mảng bằng cách tham khảo con trỏ.

```
const n =10 ;
int a[n], *p;
void main()
{ int j,temp;
  clrscr();
  p=a;
  printf("\Nhập dãy số A :\n");
  for (int i=0; i<n; i++)
  { printf("A[%d] = ",i+1);
    scanf("%d",p+i);
  }
  // Sắp xếp mảng A theo giải thuật Bubble Sort
  for ( i=1; i<n; i++)
    for (j=n-1; j>=i; j--)
      if (*(p+j-1) > *(p+j))
      { temp = *(p+j);
        *(p+j) = *(p+j-1);
        *(p+j-1) = temp;
      }
  printf("\n Mảng A sau khi sắp xếp :\n");
  for ( i=0; i<n; i++)
    printf("%8d",*(p+i));
}
```

\* Đối với mảng 2 chiều:

Nếu ta khai báo : Kiểu a [10] [20] , \*p; thì mảng a có dạng:

a	0	1	2	3	.....	18	19
0							
1							
2							
3							
a[i]							
.							
9							

Nhân xét:

a = &a[0][0] = &a[0]

a[i] = &a[i][0]

$a[i][j]$  nội dung của ô  $i,j$

Với  $p = a$  thì để truy xuất tới ô  $a[i][j]$  :

$$a[i][j] = (*(p+i) + j)$$

$$\&a[i][j] = (*(p+i) + j)$$

Ví dụ: Nếu ta có  $\text{int } a[4][5] =$

{ {1,2,3,4,5}, {2,3,4,5,6}, {3,4,5,6,7}, {4,5,6,7,8} } ;

thì :

- $a$  là địa chỉ của toàn bộ mảng (giả sử là 1000)
- Do  $a$  là mảng nguyên nên mỗi phần tử chiếm 2 bytes, và mỗi dòng của mảng sẽ chiếm 10 bytes.
- Trình biên dịch của C biết số cột (do khai báo) nên nó sẽ hiểu  $a+1$  là đem 1000 + 10 bytes, kết quả là 1010 là địa chỉ của dòng thứ 2 trong  $a$ . Tương tự, 1020 là địa chỉ của dòng thứ 3 trong  $a$ .

$a$	0	1	2	3	4
1000	1	2	3	4	5
1010	2	3	4	<b>5</b>	6
1020	3	4	5	6	7
1030	4	5	6	7	8

Lúc này:  $a[1]$  hay  $a+1$  là địa chỉ của dòng thứ 2 trong mảng 2 chiều  $a$ .

Ta có :  $*(a+1) == 1010$  // địa chỉ của phần tử đầu tiên trên dòng 1

$*(a+1)+3 == 1016$  // địa chỉ của phần tử có chỉ số 3 trên dòng 1

$*(*(a+1)+3) == 5$  // nội dung của phần tử  $a[1][3]$

Tóm lại:  $a[i][j] = (*(a+i) + j)$

Ta còn có một cách khác để truy xuất tới  $a[i][j]$  :

**Nếu** : Kiểu  $a[n_0] [n_1] \dots [n_m]$  ,  $*p$ ;

$p = a$ ;

**thì**  $a[i_0] [i_1] \dots [i_m] = (*(*(p+i_0) + i_1) + \dots i_m)$

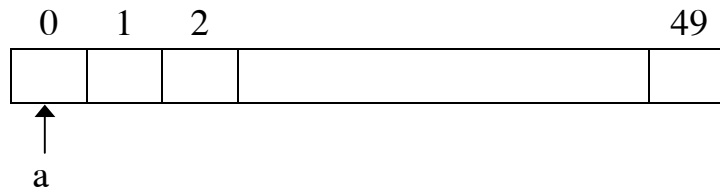
**Chú ý:**

### 1. Sự khác nhau giữa con trỏ và mảng:

- Biến con trỏ thì có thể tăng, giảm hoặc gán còn biến mảng là một con trỏ hằng do đó không thể tăng, giảm hoặc gán.
- Ta có thể lấy địa chỉ của con trỏ nhưng không thể lấy địa chỉ của mảng vì bản thân mảng đã là địa chỉ.
- Khi ta khai báo một mảng thì chương trình biên dịch sẽ cấp phát một vùng nhớ cho nó.

Ví dụ 1: Kiểu  $a[50]$

Trong bộ nhớ :



- Biến con trỏ khi được khai báo thì chỉ được cấp một ô nhớ mà nội dung của nó chẳng biết chỉ đến đâu

Ví dụ 2: `a[1]` xác định thành phần thứ 2

`p+1` : nội dung không xác định

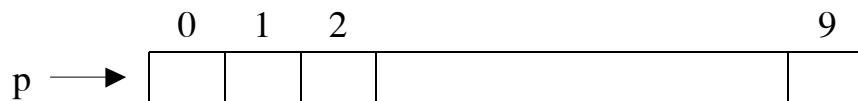
⇒ phải có `p = a` để `p` chỉ tới `a`

- Nếu ta muốn tạo một mảng bằng con trỏ thì ta phải xin cấp phát một vùng nhớ bằng hàm `malloc()`

Ví dụ: `int *p;`

`p = (int) malloc ( 10* sizeof(int));`

Trong bộ nhớ:



Ví dụ: `int *p;`

`p = malloc (10* sizeof (int));`

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

`scanf ("%d", p+i)`

- Để loại bỏ vùng nhớ được cấp cho con trỏ ta dùng hàm `free (p)`

## 2. Sự khác nhau giữa tham số của hàm là mảng và đối số là pointer:

Hàm (kiểu `a[]`)

Hàm (kiểu `*p`)

Chú ý:

Nếu: Kiểu `a[50][30], *p;`

`p = a;`

thì Hàm (Kiểu `a[][30]`)

Hàm (Kiểu `*p`)

## 3. Hàm trả về kiểu con trỏ:

Kiểu `*hàm (đối số)`

Ví dụ:

`char *strcat (char s1[], char s2[])`

{ `int i=0, j=0;`

`while ( s1[i]!='\0' ) i++;`

`while ( (s1[i++] = s2[j++]) !='\0' ) ;`

```
    return s1 ;
}
```

#### IV. CON TRỎ VÀ CHUỖI

**IV.1. Khai báo:** Để khai báo s là 1 chuỗi ký tự và p là con trỏ trỏ đến 1 chuỗi ký tự, ta viết như sau:

```
char s[50];
char *p;
```

Để khởi tạo 1 chuỗi trong cả 2 trường hợp:

```
static char s[] = "ABCD";
char *p = "ABCD";
```

Lúc này, nếu ta: puts (p) ; // → ABCD

```
puts (++p) ; // → BCD
```

#### IV.2. Xét một số ví dụ về các hàm xử lý chuỗi

**a. Hàm strcpy:** Sao chép chuỗi ký tự từ source qua dest.

	0	1	2	3	4	5	
source	A	B	C	D	E	F	\0
dest							

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

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

```
void strcpy (char dest[], char source[])
```

```
{ int i=0;
```

```
    while ((dest[i++] = source[i]) != '\0');
```

```
}
```

```
void main()
```

```
{ char src_str[]="Hoang Van";
```

```
  char dst_str[20];
```

```
  strcpy(dst_str,src_str);
```

```
  printf("\n%s", dst_str);
```

```
}
```

Viết lại hàm strcpy bằng con trỏ:

```
void strcpy (char *dest, const char *source)
```

```
{ while ((*dest = *source) != '\0')
```

```
  { dest ++;
```

```

        source ++;
    }
}

```

**b. Hàm Strcmp () :** dùng để so sánh 2 chuỗi s1, s2 với nhau.

```

int    strcmp (char s1[] , char s2 []);
{ int  i= 0;
    while (s1[i] == s2[i])
        { if (s1[i] == '\0')
            return 0;
          i++;
        }
    return (s1[i]- s2[i]);
}

```

Cài đặt lại bằng con trỏ:

```

int strcmp (char *s1, const char *s2);
{ while (*s1 == *s2)
    { if (*s1 == '\0')    return 0;
      *s1++;
      *s2++;
    }
    return (*s1 - *s2)
}

```

**c. Hàm strcat:** nối chuỗi s2 sau chuỗi s1

```

char *strcat (char s1[],char s2[])
{ int i=0,j=0;
  while ( s1[i]!='\0' ) i++;
  while ( (s1[i++] = s2[j++]) !='\0' ) ;
  return s1 ;
}

```

Cài đặt lại bằng con trỏ:

```

char *strcat (char *s1, const char *s2)
{ char *p;
  p=s1;
  while ( *s1!='\0' )
      s1 ++;
  while ( (*s1=*s2) !='\0' )
      { s1 ++; s2 ++;
      }
}

```

```

        return p ;
    }

```

**d. Hàm strchr:** trả về địa chỉ của ký tự c trong chuỗi s.

```

#include <stdio.h>
#include <conio.h>
char *strchr (char s[], char c)
{
    int i=0;
    while ( s[i]!=c && s[i]!='\0' )
        i++;
    if ( s[i] != c) return (char *)0;
    else return &s[i] ;
}

```

Cài đặt lại bằng con trỏ:

```

char *strchr (char *s, char c)
{
    while ( *s !=c && *s!='\0' )
        s++;
    if ( *s != c) return (char *)0;
    else return s ;
}
char str[]="Ky ";
void main()
{
    char *vt;
    vt=strchr(str,'y');
    if (vt==NULL )
        printf("Khong co ky tu trong chuoi" );
    else printf("\nVi tri =%d", vt-str+1);
    getch();
}

```

#### IV.3. Mảng con trỏ chỉ đến chuỗi

- **Khai báo:** Để khai báo 1 mảng con trỏ chỉ đến chuỗi, ví dụ như danh sách họ tên, ta viết như sau:

```

char * ds[5]=                                // mảng chuỗi ds[5][7]

{ "Hoang", "Van", "Chi", "Ngoc", "Nguyet" }

```

Nếu ta khởi tạo mảng mà không dùng con trỏ thì lượng ô nhớ cấp phát cho mỗi phần tử của mảng đều bằng với số ký tự của chuỗi dài nhất; Trong khi đó,

nếu khởi tạo bằng mảng con trỏ như trên thì lượng ô nhớ sẽ cấp phát vừa đủ cho từng phần tử của mảng.

ds[0]	→	H	o	a	n	g	\0
ds[1]	→	V	a	n	\0		
ds[2]	→	C	h	i	\0		
ds[3]	→	N	g	o	c	\0	
ds[4]	→	N	g	u	y	e	t \0

Hình 3.2. Mảng con trỏ trỏ đến chuỗi

ds[0]		H	o	a	n	g	\0	
ds[1]		V	a	n	\0			
ds[2]		C	h	i	\0			
ds[3]		N	g	o	c	\0		
ds[4]		N	g	u	y	e	t \0	

Hình 3.3. Mảng các chuỗi

\* Xử lý con trỏ đến chuỗi: xét ví dụ sắp xếp danh sách họ tên theo chỉ mục:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main()
{ const SISO_MAX = 50;
  char ds[SISO_MAX][30]; // mảng chuỗi
  char *p[SISO_MAX];    // mảng con trỏ đến chuỗi
  char *tam;
  char siso[2];
  int i,j,n;
  clrscr();
  gotoxy(10,2); printf("Nhap si so lop:");
  gets(siso);
  n= atoi(siso);
  for (i=0; i<n; i++)
  { printf("Ho ten hoc vien thu %d :",i+1);
    gets(ds[i]);
    p[i] = ds[i]; // lấy địa chỉ của chuỗi họ tên trong ds đưa
                  // vào mảng con trỏ p
  }
```

```
for (i=0; i<n-1;i++)
    for (j=i+1; j<n; j++)
        if ( strcmp(p[i],p[j]) >0)
        { tam = p[i];
          p[i] = p[j];
          p[j] = tam;
        }
printf("\n Danh sach ho ten sau khi sap xep\n");
for (i=0; i<n;i++)
    printf("%s\n", p[i]);
}
```

Lưu ý: Chương trình trên thực chất ta chỉ hoán đổi giá trị của mảng con trỏ p chứ mảng chuỗi ds vẫn như cũ.

### Bài tập:

1. Sắp xếp lại danh sách học viên theo thứ tự họ tăng dần bằng con trỏ.
2. Sắp xếp lại danh sách học viên theo thứ tự tên tăng dần, nếu trùng tên thì sắp theo họ bằng con trỏ.
3. Sắp xếp lại danh sách học viên theo thứ tự tên tăng dần, nếu trùng tên thì sắp theo họ bằng mảng chuỗi.

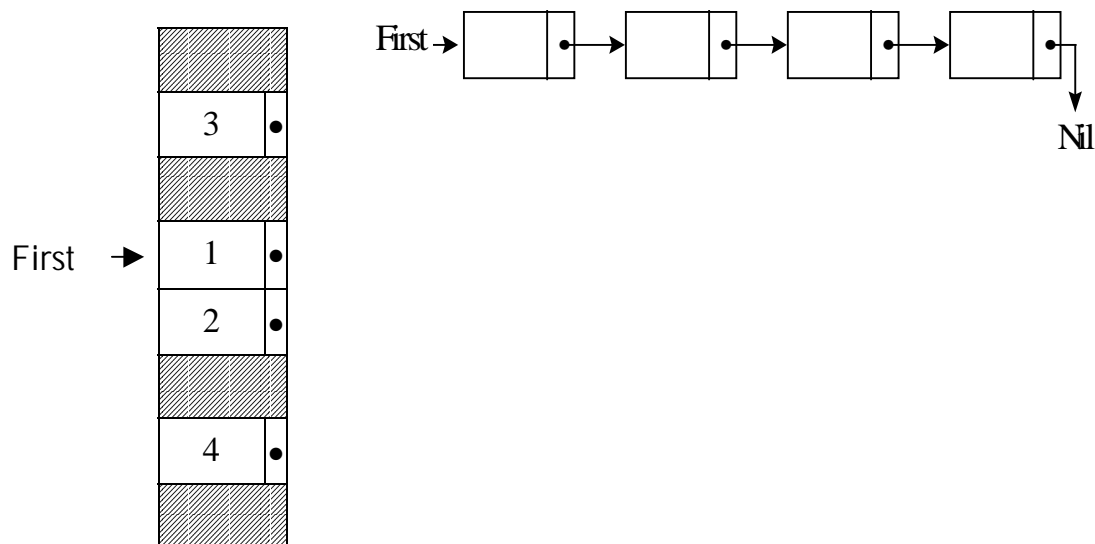


## CHƯƠNG 5 CÁCH THUẬT TOÁN TRÊN CẤU TRÚC DANH SÁCH LIÊN KẾT (LINKED LIST)

### I. KHÁI NIỆM:

Cấu trúc danh sách liên kết là cấu trúc động, việc cấp phát nút và giải phóng nút trên danh sách xảy ra khi chương trình đang chạy. Ta thường cấp phát nút cho danh sách liên kết bằng biến động.

Các phần tử sẽ được cấp phát vùng nhớ trong quá trình thực thi chương trình, do đó chúng có thể nằm rải rác ở nhiều nơi khác nhau trong bộ nhớ (không liên tục).



Các phần tử trong danh sách được kết nối với nhau theo chuỗi liên kết như hình trên:

- First là con trỏ chỉ đến phần tử đầu của danh sách liên kết
- Phần tử cuối của danh sách liên kết với vùng liên kết có giá trị NULL
- Mỗi nút của danh sách có trường info chứa nội dung của nút và trường next là con trỏ chỉ đến nút kế tiếp trong danh sách.

#### \* Lưu ý:

- Cấu trúc danh sách liên kết là cấu trúc động, các nút được cấp phát hoặc bị giải phóng khi chương trình đang chạy.

- Danh sách liên kết rất thích hợp khi thực hiện các phép toán trên danh sách thường bị biến động. Trong trường hợp xóa hay thêm phần tử trong danh sách liên kết thì ta không dời các phần tử đi như trong mảng mà chỉ việc hiệu chỉnh lại trường next tại các nút đang thao tác. Thời gian thực hiện các phép toán thêm vào và loại bỏ không phụ thuộc vào số phần tử của danh sách liên kết.

- Tuy nhiên, danh sách liên kết cũng có các điểm hạn chế sau:
  - + Vì mỗi nút của danh sách liên kết phải chứa thêm trường next nên danh sách liên kết phải tốn thêm bộ nhớ.
  - + Tìm kiếm trên danh sách liên kết không nhanh vì ta chỉ được truy xuất tuần tự từ đầu danh sách.
- **Khai báo** : Một phần tử của danh sách liên kết ít nhất phải có hai thành phần : nội dung của phần tử (info) và thành phần next liên kết phần tử này với phần tử khác.

Giả sử ta khai báo kiểu NODEPTR là kiểu con trỏ chỉ đến nút trong 1 danh sách liên kết, mỗi phần tử có 2 thành phần : info (int) và next .

```
struct node
{
    int info ;
    struct node *next ;
};
typedef struct node *NODEPTR;
```

- Để khai báo biến First quản lý danh sách liên kết ta viết như sau:

```
NODEPTR First;
```

- Khởi tạo danh sách liên kết : First = NULL;

- **Ghi chú** :

- Thành phần chứa nội dung có thể gồm nhiều vùng với các kiểu dữ liệu khác nhau.
- Thành phần liên kết cũng có thể nhiều hơn một nếu là danh sách đa liên kết hoặc danh sách liên kết kép.
- First là con trỏ trỏ đến phần tử đầu tiên của danh sách liên kết, nó có thể là kiểu con trỏ (như khai báo trên), và cũng có thể là một struct có hai thành phần: First trỏ đến phần tử đầu tiên của danh sách liên kết, và Last trỏ đến phần tử cuối của danh sách liên kết.

```
struct Linked_List;
{
    First NODEPTR;
    Last NODEPTR;
};
```

## **II. CÁC PHÉP TOÁN TRÊN DANH SÁCH LIÊN KẾT:**

### **II.1. Tạo danh sách:**

- a. **Khởi tạo danh sách** (Initialize): dùng để khởi động một danh sách liên kết, cho chương trình hiểu là hiện tại danh sách liên kết chưa có phần tử.

```
void Initialize(NODEPTR &First)
{
```

```
First = NULL;
}
```

b. Cấp phát vùng nhớ (New\_Node): cấp phát một nút cho danh sách liên kết. Hàm New\_Node này trả về địa chỉ của nút vừa cấp phát.

Trong chương trình có sử dụng hàm malloc (trong <alloc.h>), hàm này cấp phát một khối nhớ tĩnh theo byte từ bộ nhớ heap. Nếu cấp phát thành công, hàm malloc trả về địa chỉ của vùng nhớ vừa cấp phát, ngược lại nó sẽ trả về NULL.

```
NODEPTR New_Node()
{
    NODEPTR p;
    p = (NODEPTR)malloc(sizeof(struct node));
    return (p);
}
```

c. Thêm vào đầu danh sách (Insert\_First): thêm một nút có nội dung x vào đầu danh sách liên kết.

```
void Insert_First (NODEPTR &First, int x)
{
    NODEPTR p;
    p = New_Node();
    p->info = x;
    p->next = First;
    First = p;
}
```

d. Thêm nút mới vào sau nút có địa chỉ p (Insert\_After): thêm một nút có nội dung x vào sau nút có địa chỉ p trong danh sách liên kết First.

```
void Insert_After(NODEPTR p, int x)
{
    NODEPTR q;
    if(p == NULL)
        printf("khong them nut moi vao danh sach duoc");
    else
    {
        q = New_Node();
        q->info = x;
        q->next = p->next;
        p->next = q;
    }
}
```

## II.2. Cập nhật danh sách:

a. Giải phóng vùng nhớ(Free\_Node): Hàm này dùng để hủy nút đã cấp phát, và trả vùng nhớ về lại cho memory heap.

```
void Free_Node(NODEPTR p)
{
    free(p);
}
```

b. Kiểm tra danh sách liên kết rỗng hay không (Empty): hàm Empty trả về TRUE nếu danh sách liên kết rỗng, và ngược lại.

```
int Empty(NODEPTR First)
{
    return(First == NULL ? TRUE : FALSE);
}
```

c. Xóa phần tử đầu của danh sách (Delete\_First): muốn xóa 1 phần tử khỏi danh sách liên kết thì ta phải kiểm tra xem danh sách có rỗng hay không. Nếu danh sách có phần tử thì mới xóa được.

```
void Delete_First (NODEPTR First)
{ NODEPTR p;
  if (Empty(First))
      printf("Danh sach rong nen khong the xoa");
  else
  {
      p = First; // nut can xoa la nut dau
      First = p->next;
      Free_Node(p);
  }
}
```

d. Xóa phần tử đứng sau nút có địa chỉ p (Delete\_After):

```
void Delete_After(NODEPTR p)
{ NODEPTR q;
  // nếu p là NULL hoặc sau p không có nút
  if((p == NULL) || (p->next == NULL))
      printf("khong xoa duoc");
  else
  {
      q = p->next; // q chỉ nút cần xóa
      p->next = q->next;
```

```

    Free_Node(q);
}
}

```

e. Xóa toàn bộ danh sách (Delete\_All): ta có thể sử dụng lệnh \*First = NULL để xóa toàn bộ danh sách, nhưng trong bộ nhớ, các vùng nhớ đã cấp phát cho các nút không giải phóng về lại cho memory heap, nên sẽ lãng phí vùng nhớ. Do đó, ta sử dụng giải thuật sau:

```

void Delete_All (NODEPTR &First)
{
    NODEPTR p;
    while (First != NULL)
    {
        p=First;
        First = First->next;    // hoặc First = p->next
        Free_Node(p);
    }
}

```

**II.3. Duyệt danh sách:** Thông thường ta hay duyệt danh sách liên kết để thực hiện một công việc gì đó, như liệt kê dữ liệu trong danh sách hay đếm số nút trong danh sách...

```

void Traverse(NODEPTR First)
{
    NODEPTR p;
    int stt = 0;
    p = First;
    if(p == NULL)
        printf("\n (Không có sinh viên trong danh sách)");
    while(p != NULL)
    {
        printf("\n %5d%8d", stt++, p->info);
        p = p->next;
    }
}

```

**II.4. Tìm kiếm (Search):** Tìm nút đầu tiên trong danh sách có info bằng với x. Do đây là danh sách liên kết nên ta phải tìm từ đầu danh sách.

Hàm Search nếu tìm thấy x trong danh sách thì trả về địa chỉ của nút có trị bằng x trong danh sách, nếu không có thì trả về trị NULL.

```

NODEPTR Search(NODEPTR First, int x)
{
    NODEPTR p;

```

```

p = First;
while(p != NULL && p->info != x )
    p = p->next;
return (p);
}

```

**II.5. Sắp xếp** (Selection\_Sort): sắp xếp danh sách liên kết theo thứ tự info tăng dần.

- Nội dung: Ta so sánh tất cả các phần tử của danh sách để chọn ra một phần tử nhỏ nhất đưa về đầu danh sách; sau đó, tiếp tục chọn phần tử nhỏ nhất trong các phần tử còn lại để đưa về phần tử thứ hai trong danh sách. Quá trình này lặp lại cho đến khi chọn ra được phần tử nhỏ thứ (n-1).

- Giải thuật:

```

void Selection_Sort(NODEPTR First)
{ NODEPTR p, q, pmin;
  int min;
  for(p = First; p->next != NULL; p = p->next)
  {   min = p->info;
      pmin = p;
      for(q = p->next; q != NULL; q = q->next)
          if(min > q->info)
          {
              min = q->info;
              pmin = q;
          }
      // hoan doi truong info cua hai nut p va pmin
      pmin->info = p->info;
      p->info = min;
  }
}

```

**Bài tập:**

1. Viết chương trình tạo một menu thực hiện các công việc sau:
  - a. Nhập danh sách liên kết theo giả thuật thêm về đầu danh sách, mỗi phần tử gồm có các thông tin sau: mssv (int), và hoten ( char hoten[30] ).
  - b. Liệt kê danh sách ra màn hình
  - c. Cho biết tổng số nút trong danh sách liên kết, đặt tên hàm là Reccount ( int Reccount(NODEPTR First) )
  - d. Thêm 1 phần tử có nội dung info (mssv, hoten) vào sau phần tử có thứ tự thứ i trong danh sách.

**Ghi chú:** - Thứ tự theo qui ước bắt đầu là 1

  - Nếu (i = 0) thêm vào đầu danh sách
  - Nếu i > Reccount(&First) thì thêm vào cuối danh sách.
  - e. In ra họ tên của sinh viên có mã do ta nhập vào.
  - f. Loại bỏ nút có mã do ta nhập vào, trước khi xóa hỏi lại "Bạn thật sự muốn xóa (Y/N) ? "
  - g. Sắp xếp lại danh sách theo thứ tự mã số giảm dần.
  - h. Ghi toàn bộ danh sách vào file tên 'DSSV.DAT'
  - i. Nhập danh sách từ file 'DSSV.DAT' vào danh sách liên kết. Nếu trong danh sách liên kết đã có nút thì xóa tất cả dữ liệu hiện có trong danh sách liên kết trước khi đưa dữ liệu từ file vào.
2. Viết chương trình tạo một danh sách liên kết theo giả thuật thêm vào cuối danh sách, mỗi nút chứa một số nguyên.
3. -Viết hàm tên Delete\_Node để xóa nút có địa chỉ p.  
- Viết một hàm loại bỏ tất cả các nút có nội dung x trong danh sách liên kết First.
4. Viết hàm Copy\_List trên danh sách liên kết để tạo ra một danh sách liên kết mới giống danh sách liên kết cũ.
5. Ghép một danh sách liên kết có địa chỉ đầu là First2 vào một danh sách liên kết có địa chỉ đầu là First1 ngay sau phần tử thứ i trong danh sách liên kết First1.
6. Viết hàm lọc danh sách liên kết để tránh trường hợp các nút trong danh sách liên kết bị trùng info.
7. Đảo ngược vùng liên kết của một danh sách liên kết sao cho:
  - First sẽ chỉ đến phần tử cuối
  - Phần tử đầu có liên kết là NULL.

8. Viết hàm Left\_Traverse (NODEPTR &First) để duyệt ngược danh sách liên kết.
9. Viết giải thuật tách một danh sách liên kết thành hai danh sách liên kết, trong đó một danh sách liên kết chứa các phần tử có số thứ tự lẻ và một danh sách liên kết chứa các phần tử có số thứ tự chẵn trong danh sách liên kết cũ.
- 10.- Tạo một danh sách liên kết chứa tên học viên, điểm trung bình, hạng của học viên (với điều kiện chỉ nhập tên và điểm trung bình). Quá trình nhập sẽ dừng lại khi tên nhập vào là rỗng.  
- Xếp hạng cho các học viên. In ra danh sách học viên thứ tự hạng tăng dần (Ghi chú : Cùng điểm trung bình thì cùng hạng).
11. Nhập hai đa thức theo danh sách liên kết. In ra tích của hai đa thức này.  
Ví dụ: Đa thức First1 :  $2x^5+4x^2-1$   
Đa thức First2 :  $10x^7-3x^4+x^2$   
 $\Rightarrow$  Kết quả in ra :  $20x^{12} + 34x^9 - 8x^7 - 12x^6 + 7x^4 - x^2$   
(Ghi chú : Không nhập và in ra các số hạng có hệ số bằng 0)
12. Viết giải thuật thêm phần tử có nội dung x vào danh sách liên kết có thứ tự tăng dần sao cho sau khi thêm danh sách liên kết vẫn có thứ tự tăng.
13. Loại bỏ phần tử có nội dung là x trong danh sách liên kết có thứ tự tăng dần.
14. Cho 2 danh sách liên kết First1, First2 có thứ tự tăng dần theo info. Viết giải thuật Merge để trộn 2 danh sách liên kết này lại sao cho danh sách liên kết sau khi trộn cũng có thứ tự tăng dần.



## CHƯƠNG 6 CÁC THUẬT TOÁN TRÊN CẤU TRÚC CÂY (Tree)

Cây là một cấu trúc dữ liệu rất thông dụng và quan trọng trong nhiều phạm vi khác nhau của kỹ thuật máy tính.

**Ví dụ :** Tổ chức các quan hệ họ hàng trong một gia đình, mục lục của một cuốn sách, xây dựng cấu trúc về cú pháp trong các trình biên dịch.

Trong chương trình này, chúng ta khảo sát các khái niệm cơ bản về cây, các phép toán trên cây nhị phân, cũng như các phép toán trên cây nhị phân cân bằng (AVL tree) và ứng dụng của hai loại cây nhị phân tìm kiếm (BST), cây nhị phân cân bằng (AVL tree).

### I. PHÂN LOẠI CÂY:

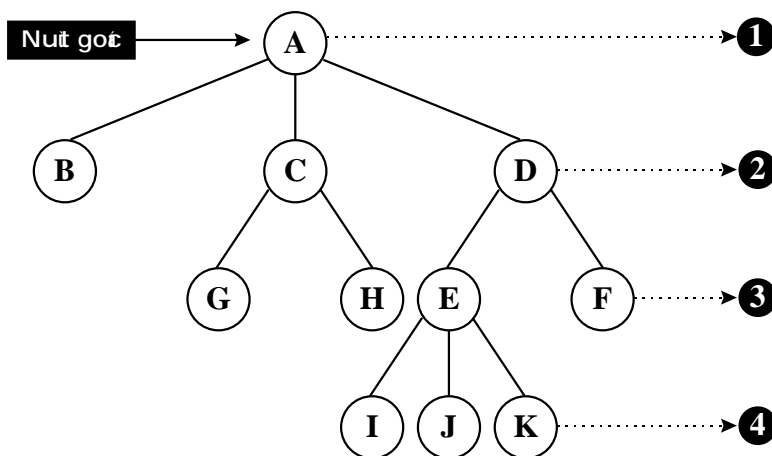
#### I.1. Một số khái niệm cơ bản:

1. **Cây:** Cây là tập hợp các phần tử gọi là nút, một nút (tương tự như một phần tử của dãy) có thể có kiểu bất kỳ. Các nút được biểu diễn bởi 1 ký tự chữ, một chuỗi, một số ghi trong một vòng tròn.

*Một số định nghĩa a theo đệ quy*

- ♦ Một nút đơn cũng chính là một cây.
- ♦ Các nút được gọi là ở cùng một cây khi có đường đi giữa các nút này.
- ♦ Một cây sẽ bao gồm một nút gốc (Root) và **m** cây con, trong mỗi cây con lại có một nút gốc và **m1** cây con nhỏ hơn v.v.
- ♦ Một cây không có một nút nào cả gọi là cây rỗng.

**Ví dụ 1 :**



- A là nút gốc với 3 cây con lần lượt có 3 nút gốc riêng là B, C, D

- Nút cha (*ancestor*)

Nút con (*descendent*)

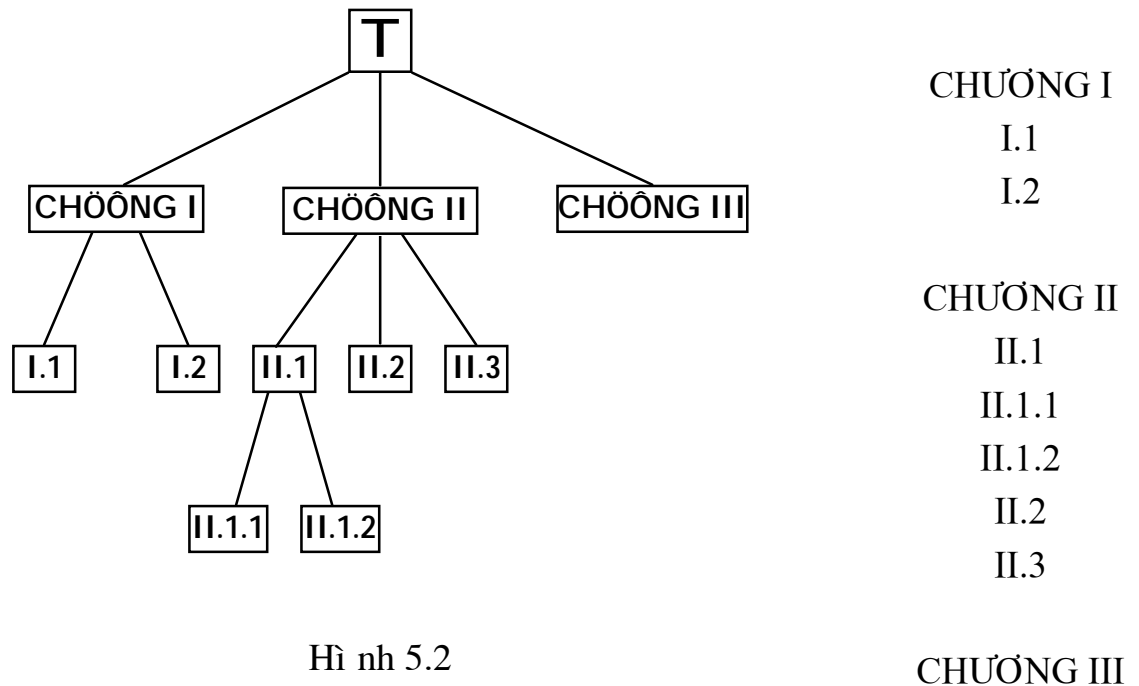
A là nút cha của B, C, D

G, H là nút con của C

G, H không quan hệ cha con với A

Hình 5.1. Cây với nút gốc là A

Ví dụ 2 : Với đề cương một môn học T, ta có thể biểu diễn dạng cây như sau :



Hình 5.2

2. **Nút cha** (*Ancestor*) : Nút đứng trên của một nút được gọi là nút cha  
C là nút cha của G, H

**Nút con** (*descendent*) : Nút đứng sau một nút khác được gọi là nút con của nút đó.

Nút I, J, K là nút con của nút E

3. **Bậc** (*degree*) :

- Bậc của nút là số cây con của nút đó.  
C có bậc là 2, E có bậc là 3 (Hình 5.1)
- Bậc của cây là bậc lớn nhất của các nút trong cây.  
Cây trong hình 5.1 có bậc là 3.

Cây bậc n được gọi là cây n phân như cây nhị phân, cây tam phân.

4. **Nút lá và nút trung gian**:

- Nút lá là nút có bậc bằng 0 (tức là không có cây con nào) :
- Nút trung gian: là một nút có bậc khác 0 và không phải là nút gốc.

Ví dụ: Trong hình 5.1, B, G, H, I, J, K, F là nút lá

C, D, E là nút trung gian.

5. **Mức của nút** (*level*) : Nút gốc có mức là 1

Mức của nút con = mức của nút cha + 1

Ví dụ: trong hình 5.1,

A có mức là 1

B, C, D có mức là 2

G, H, E, F có mức là 3

I, J, K có mức là 4

6. **Chiều cao của cây** (*height*): là mức lớn nhất của các nút lá trong cây.

Ví dụ: Cây trong hình 5.1 có chiều cao là 4

7. **Thứ tự của các nút** (*order of nodes*): Nếu cây được gọi là có thứ tự thì phải đảm bảo vị trí của các nút con từ trái qua phải, tức là nếu thay đổi vị trí của một nút con bất kỳ thì ta đã có một cây mới.

Ví dụ:



Hình 5.3: Sau khi đổi vị trí của 2 nút B, C ta đã có cây mới.

8. **Chiều dài đường đi** (*Path length*):

- Chiều dài đường đi của nút x: là số các cạnh đi từ nút gốc đến nút x.

Ví dụ: Trong hình 5.1:

Nút gốc A có chiều dài đường đi là 1

Nút B, C, D có chiều dài đường đi là 2

**Tổng quát**: một nút tại mức i có chiều dài đường đi là i

- Chiều dài đường đi của cây: là tổng của các chiều dài đường đi của tất cả các nút trong cây.

Ví dụ: Chiều dài đường đi của cây trong hình 5.1 là 31.

Chiều dài đường đi trung bình của cây:

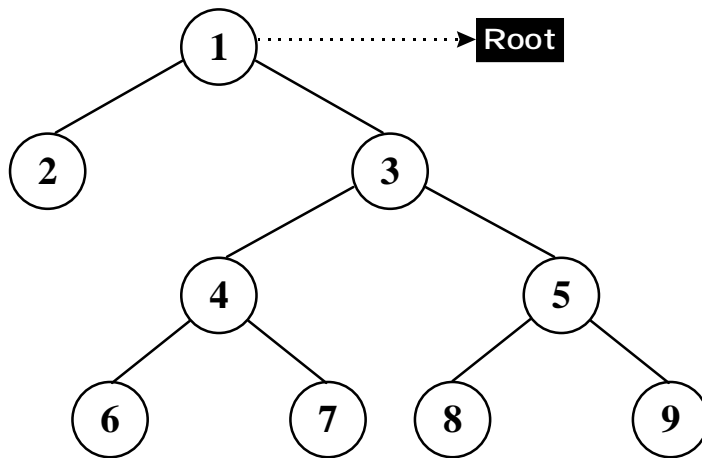
$$P_i = (\sum_i n_i \cdot i) / n$$

trong đó  $n_i$  là số các nút ở mức i và n là tổng số các nút trong cây.

**I.2. Cách biểu diễn cây**: để biểu diễn 1 cây, ta có nhiều cách như biểu diễn bằng đồ thị, bằng giản đồ, bằng chỉ số.. Nhưng thông thường, ta hay dùng dạng đồ thị để biểu diễn 1 cây như hình 5.1

**I.3. Biểu diễn thứ tự các nút trong cây**:

Một cây thường tổ chức các nút theo một thứ tự nhất định căn cứ vào một nội dung gọi là khóa của các nút. Có thể tổ chức cây có khóa tăng dần theo mức từ trái qua phải như ví dụ sau :



ROOT  $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$

Như vậy khi duyệt lại cây theo mức tăng dần và từ trái qua phải ta sẽ lại có được thứ tự các nút như trên.

Hình 5.4. Cây có thứ tự tăng dần theo mức từ trái qua phải

## II. CÂY NHỊ PHÂN (Binary tree)

### II.1. Định nghĩa :

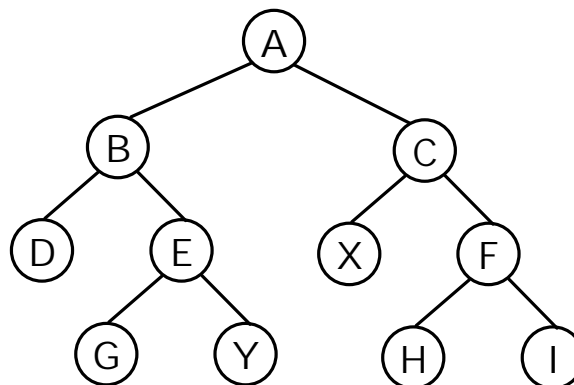
1. **Cây nhị phân** là cây có bậc bằng 2, tức là số nút con tối đa của một nút bất kỳ trong cây là 2.

Cây nhị phân có thể là một cây rỗng (không có nút nào) hoặc cây chỉ có một nút, hoặc cây chỉ có các nút con bên trái (Left Child) hoặc nút con bên phải (Right Child) hoặc cả hai.

Ví dụ: Hình 5.4 là cây nhị phân.

### 2. Các cây nhị phân đặc biệt:

- Cây nhị phân đúng: Một cây nhị phân được gọi là cây nhị phân đúng nếu nút gốc và tất cả các nút trung gian đều có 2 nút con.



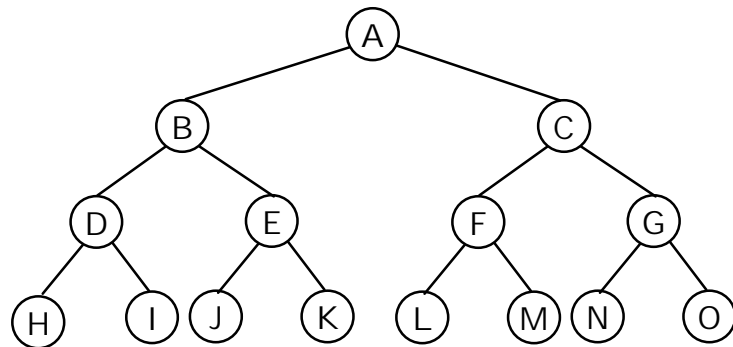
Hình 5.5. Cây nhị phân đúng

Ghi chú: nếu cây nhị phân đúng có  $n$  nút lá thì cây này sẽ có tất cả  $2n-1$  nút.

- Cây nhị phân đầy: Một cây nhị phân gọi là cây nhị phân đầy với chiều cao  $d$  thì :

- . Nó phải là cây nhị phân đúng và
- . Tất cả các nút lá đều có mức là  $d$ .

Hình 5.5 không phải là cây nhị phân đầy

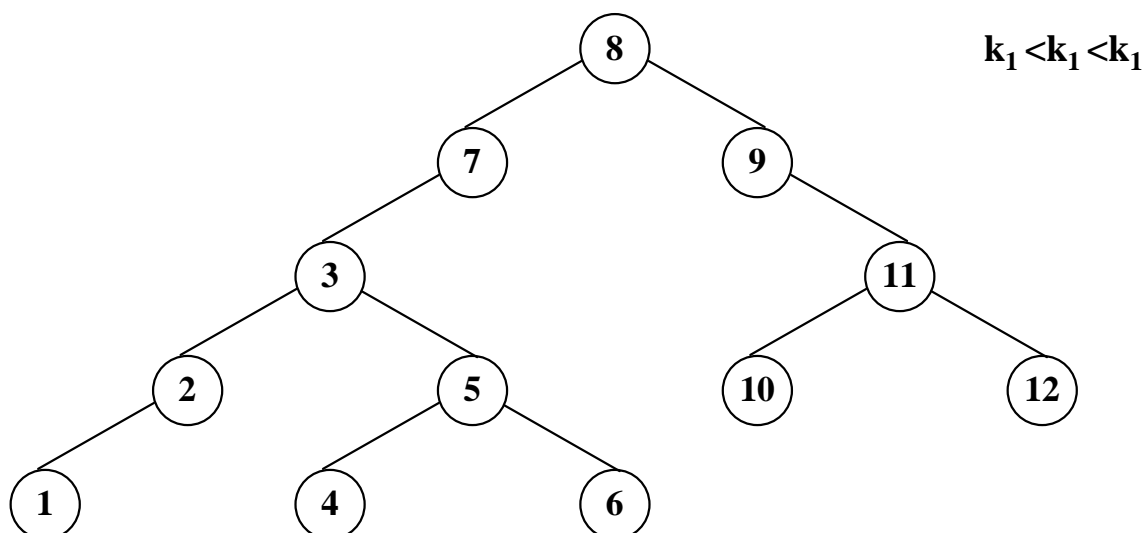


Hình 5.6. Cây nhị phân đầy.

Ghi chú: Cây nhị phân đầy là cây nhị phân có số nút tối đa ở mỗi mức.

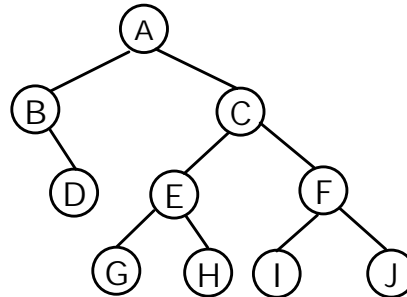
- Cây nhị phân tìm kiếm (Binary Search Tree): Một cây nhị phân gọi là cây nhị phân tìm kiếm nếu và chỉ nếu đối với mọi nút của cây thì khóa của một nút bất kỳ phải lớn hơn khóa của tất cả các nút trong cây con bên trái của nó và phải nhỏ hơn khóa của tất cả các nút trong cây con bên phải của nó.

Ví dụ:



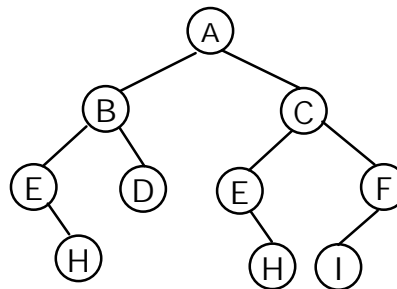
Hình 5.7. Cây nhị phân tìm kiếm (BST)

- Cây nhị phân cân bằng (AVL): Một cây nhị phân được gọi là cây nhị phân cân bằng nếu và chỉ nếu đối với mọi nút của cây thì chiều cao của cây con bên trái và chiều cao của cây con bên phải hơn kém nhau nhiều nhất là 1. (Theo Adelson-Velski và Landis).



Hình 5.8. Cây nhị phân cân bằng

- Cây nhị phân cân bằng hoàn toàn: Một cây nhị phân được gọi là cây nhị phân cân bằng hoàn toàn nếu và chỉ nếu đối với mọi nút của cây thì số nút của cây con bên trái và số nút của cây con bên phải hơn kém nhau nhiều nhất là 1.



Hình 5.9. Cây nhị phân cân bằng hoàn toàn

3. **Các phép duyệt cây nhị phân (Traverse)**: là quá trình đi qua các nút đúng một lần. Khi duyệt cây, ta thường dùng 3 cách duyệt cơ bản sau:

• **Preorder** - Tiền tự (NLR) duyệt qua nút gốc trước, sau đó đi qua cây con bên trái lại áp dụng Preorder cho cây con bên trái. Cuối cùng qua cây con bên phải, áp dụng Preorder cho cây con bên phải.

Ví dụ: Theo cây nhị phân 5.4, ta có:

**ROOT** → 1 → 2 → 3 → 4 → 6 → 7 → 5 → 8 → 9

• **Inorder** - Trung tự (LNR) : qua cây con bên trái duyệt trước (theo thứ tự LNR), sau đó thăm nút gốc. Cuối cùng qua cây con bên phải (theo thứ tự LNR)

Ví dụ: Theo cây nhị phân 5.4, ta có:

**ROOT** → 2 → 1 → 6 → 4 → 7 → 3 → 8 → 5 → 9

• **Postorder** - Hậu tự (LRN) : qua cây con bên trái duyệt trước (theo thứ tự LRN), sau đó qua cây con bên phải (theo thứ tự LRN). Cuối cùng thăm nút gốc.

Ví dụ: Theo cây nhị phân 5.4, ta có:

**ROOT** → 2 → 6 → 7 → 4 → 8 → 9 → 5 → 3 → 1

**Ghi chú :** Đối với cây ta có thể tổ chức thứ tự theo khóa là một nội dung của nút hoặc ta đặt thêm 1 field gọi là khóa của nút .

## **II.2. Các phép toán trên cây nhị phân:**

- **Khai báo:** Để tổ chức dữ liệu theo cây nhị phân, ta có thể dùng một nội dung của dữ liệu để làm khóa sắp xếp và tổ chức cây theo nhiều cách khác nhau. Nhưng thông thường để thuận tiện cho việc tìm kiếm và thực hiện các phép toán khác trên cây, người ta tạo thêm một khóa riêng trong các phần tử và tạo ra cây nhị phân tìm kiếm.

Để khai báo biến **tree** quản lý một cây nhị phân, với nội dung info chứa số nguyên, ta khai báo như sau:

```
struct nodetype
{
    int key;
    int info;
    struct nodetype *left;
    struct nodetype *right;
};
typedef struct nodetype *NODEPTR;
NODEPTR tree;
```

### **1. Tạo cây:**

a. **Khởi tạo cây**(Initialize): dùng để khởi động cây nhị phân, cho chương trình hiểu là hiện tại cây nhị phân rỗng.

```
void Initialize(NODEPTR &root)
{
    root = NULL;
}
```

**Lời gọi hàm:** Initialize(tree);

b. **Cấp phát vùng nhớ** (New\_Node): cấp phát một nút cho cây nhị phân. Hàm New\_Node này trả về địa chỉ của nút vừa cấp phát.

```
NODEPTR New_Node(void)
{
    NODEPTR p;
    p = (NODEPTR)malloc(sizeof(struct nodetype));
    return(p);
}
```

**Lời gọi hàm:** p= New\_Node();

c. Tạo cây BST (Create\_Tree): Trong giả i thuật tạo cây BST, ta có dùng hàm Insert.

Hàm Insert: dùng phương pháp đệ qui thêm nút có khóa x, nội dung a vào cây có nút gốc root. Cây nhị phân tạo được qua giả i thuật Create\_Tree là cây nhị phân tìm kiếm (BST).

```
void Insert(NODEPTR root, int x, int a)
{
    NODEPTR p;
    if(x == root->key)                // khóa bị trùng, dừng chương trình
    {
        printf("bi trung khoa, khong them nut nay duoc");
        return;
    }
    if(x < root->info && root->left == NULL) // điều kiện dừng giả i thuật đệ qui
    {
        p = New_Node();           // cấp phát vùng nhớ
        p->key = x;
        p->info = a;
        p->left = NULL;
        p->right = NULL;
        root->left = p;
        return;
    }
    if(x > root->info && root->right == NULL) // điều kiện dừng giả i thuật đệ qui
    {
        p = New_Node();
        p->key = x;
        p->info = a;
        p->left = NULL;
        p->right = NULL;
        root->right = p;
        return;
    }
    if(x < root->info)                // bước đệ qui
        Insert(root->left, x, a);    // gọi đệ qui qua nhánh trái
    else
        Insert(root->right, x, a);  // gọi đệ qui qua nhánh phải
}
```



```

void Create_Tree(NODEPTR &root)
{ int khoa, noidung;
  char so[10];
  NODEPTR p;
  do
  { printf("Nhap khoa :");
    gets(so) ;
    khoa = atoi(so);
    if (khoa !=0)
    { printf("Nhap noi dung :");
      gets(so) ;
      noidung = atoi(so);
      if (root==NULL)
      { p = New_Node();
        p->key = khoa;
        p->info = noidung;
        p->left = NULL;
        p->right = NULL;
        root =p;
      }
      else Insert(root,khoa,noidung);
    }
  } while (khoa!=0);          // khóa =0 thì dừng nhập
}

```

Ghi chú : Để tạo cây nhị phân do biến tree quản lý, ta gọi:  
Create\_Tree(tree);

## 2. Cập nhật cây:

a. Giải phóng vùng nhớ(Free\_Node): giải phóng vùng nhớ mà p đang trỏ đến.

```

void Free_Node(NODEPTR p)
{
  free(p);
}

```

Lời gọi hàm: Free\_Node (p);

b. Kiểm tra cây nhị phân rỗng hay không (Empty): hàm Empty trả về TRUE nếu cây nhị phân rỗng, và ngược lại.

```

int Empty(NODEPTR root)
  return(root == NULL ? TRUE : FALSE);
}

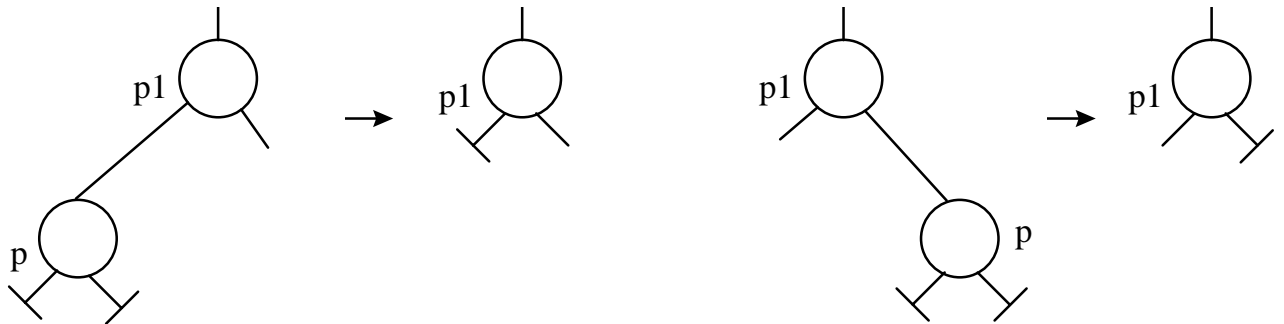
```

Lời gọi hàm: Empty(tree)

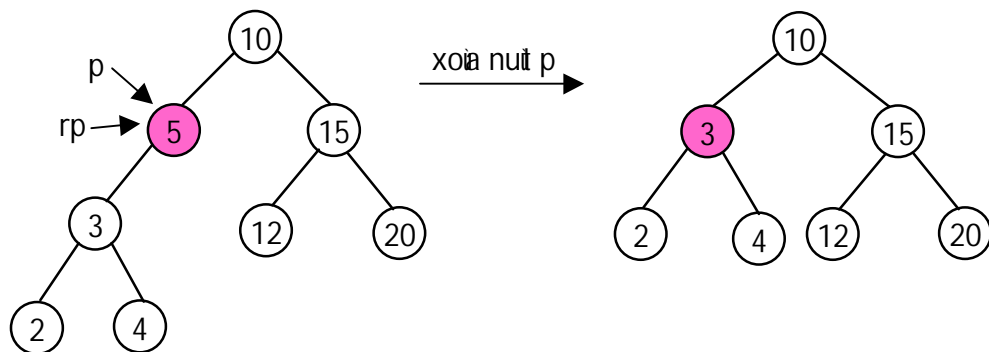
c. Hủy bỏ một nút trong cây nhị phân BST (Remove):

Xóa nút có khóa là  $x$  trong cây nhị phân tìm kiếm sao cho sau khi xóa thì cây nhị phân vẫn là cây nhị phân tìm kiếm. Ta có 3 trường hợp :

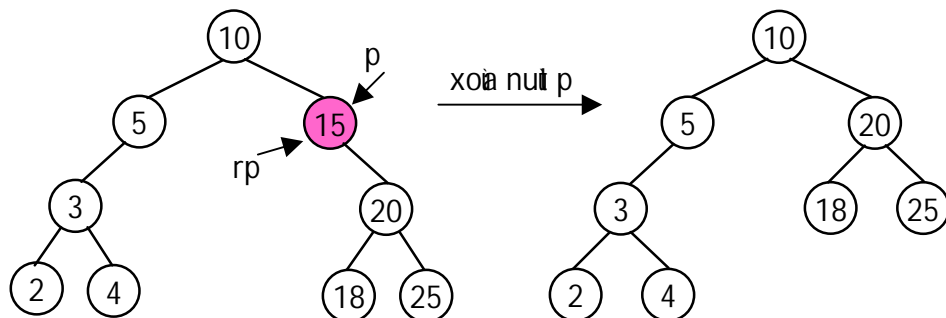
- Trường hợp 1: nút  $p$  cần xóa là nút lá. Việc xóa nút  $p$  chỉ đơn giản là hủy nút  $p$



- Trường hợp 2: Nút  $p$  cần xóa có 1 cây con, thì ta cho  $rp$  chỉ tới nút  $p$ . Sau đó, ta tạo liên kết từ nút cha của  $p$  tới nút con của  $p$ , cuối cùng hủy nút  $p$ .



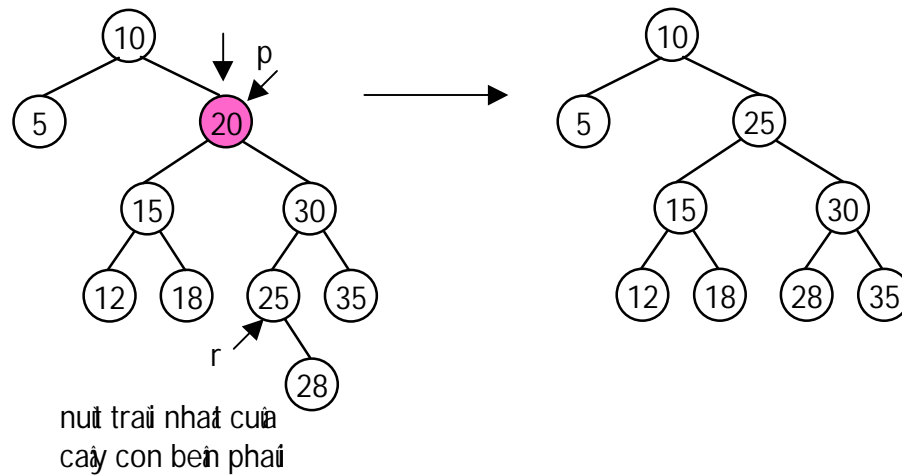
Hình 5.10. Xóa nút  $p$  trong trường hợp nút này có 1 cây con bên trái.



Hình 5.11. Xóa nút  $p$  trong trường hợp nút này có 1 cây con bên phải.

- Trường hợp 3: Nút  $p$  cần xóa có 2 cây con. Ta cho  $rp$  chỉ tới nút  $p$ . Do tính chất nút cực trái của cây con bên phải của  $p$  có khóa vừa lớn hơn khóa của  $p$ , nên để loại  $p$  thì ta sẽ cho  $r$  chỉ tới nút cực trái đó. Sau đó, ta sao chép nội dung

và khóa của nút  $r$  và o nút mà  $rp$  đang chỉ tới. Ta tạo liên kết thích hợp để bứt nút  $rp$  ra khỏi cây nhị phân và cuối cùng xóa nút  $rp$ .



Hình 5.12. Xóa nút  $p$  trong trường hợp nút này có 2 cây con.

Hàm **Remove** xóa nút có khóa là  $x$ :

NODEPTR  $rp$ ;

void remove\_case\_3 ( NODEPTR & $r$  )

{

if ( $r->left \neq \text{NULL}$ )

remove\_case\_3 ( $r->left$ );

//den đây  $r$  là nút cực trái của cây con bên phải có nút gốc là  $rp$

else

{

$rp->key = r->key;$  //Chép nội dung của  $r$  sang  $rp$ ;

$rp->info = r->info;$  // để lát nữa free( $rp$ )

$rp = r;$

$r = r->right;$

}

}

void remove (int  $x$  , NODEPTR & $p$  )

{

if ( $p == \text{NULL}$ ) printf ("Không tìm thấy");

else

if ( $x < p->key$ ) remove ( $x$ ,  $p->left$ );

else if ( $x > p->key$ )

```

        remove(x, p->right);
    else // p^.key = x
    {
        rp = p;
        if (rp->right == NULL) p = rp->left;
        // p là nút lá hoặc là nút chỉ có cây con bên trái
    else if (rp->left == NULL)
        p = rp->right; // p là nút có cây con bên phải
    else remove_case_3(rp->right);
    free(rp);
    }
}

```

Lời gọi hàm: Remove(x, tree); // x là khóa của nút muốn xóa

d. Tìm kiếm (Search): Tìm nút có khóa bằng x trên cây nhị phân BST có gốc là root. Nếu tìm thấy x trong cây thì trả về địa chỉ của nút có trị bằng x trong cây, nếu không có thì trả về trị NULL.

Do cây nhị phân là BST nên ta có thể tìm kiếm nhanh bằng phương pháp tìm kiếm nhị phân.

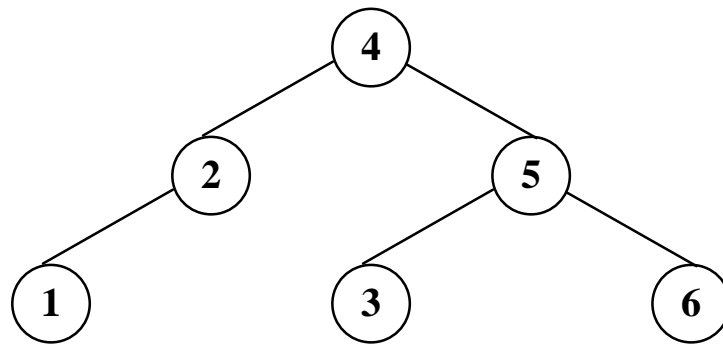
```

NODEPTR Search(NODEPTR root, int x)
{
    NODEPTR p;
    p = root;
    while(p != NULL && x != p->key)
        if(x < p->key)
            p = p->left;
        else
            p = p->right;
    return(p);
}

```

Lời gọi hàm: p=Search(tree, x);

**3. Các phép duyệt cây:** Có 3 cách duyệt cơ bản là NLR, LNR, LRN và một cách đặc biệt là duyệt cây theo mức. Ở đây, ta chỉ xét 3 phương pháp duyệt NLR, LNR và LRN. Xét cây sau :



a. Duyệt cây theo thứ tự NLR (Preorder):

```

void Preorder (NODEPTR root)
{ if(root != NULL)
  { printf("%d ", root->info);
    Preorder(root->left);
    Preorder (root->right);
  }
}
  
```

b. Duyệt cây theo thứ tự LNR (Inorder):

```

void Inorder(NODEPTR root)
{ if(root != NULL)
  { Inorder(root->left);
    printf("%d ", root->info);
    Inorder(root->right);
  }
}
  
```

c. Duyệt cây theo thứ tự LRN (Posorder):

```

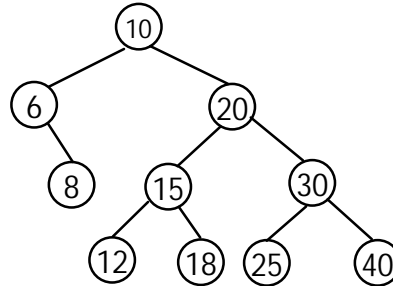
void Posorder(NODEPTR root)
{ if(root != NULL)
  { Posorder(root->left);
    Posorder(root->right);
    printf("%d ", root->info);
  }
}
  
```

### III. CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG (AVL):

Chúng ta tạo cây nhị phân tìm kiếm mục đích là để tìm khóa cho nhanh, tuy nhiên để tăng tốc độ tìm kiếm thì cây cần phải cân đối về 2 nhánh theo từng nút trong cây. Do vậy, ta sẽ tìm cách tổ chức lại cây BST sao cho nó cân bằng.

### III.1. Định nghĩa:

- Cây nhị phân tìm kiếm cân bằng (AVL) là cây nhị phân tìm kiếm mà tại tất cả các nút của nó chiều cao của cây con bên trái của nó và chiều cao của cây con bên phải chênh lệch nhau không quá một.



Hình 5.14. Cây nhị phân tìm kiếm cân bằng

**Lưu ý:** Với cây AVL, việc thêm vào hay loại bỏ 1 nút trên cây có thể làm cây mất cân bằng, khi đó ta phải cân bằng lại cây. Tuy nhiên việc cân bằng lại trên cây AVL chỉ xảy ra ở phạm vi cục bộ bằng cách xoay trái hoặc xoay phải ở một vài nhánh cây con nên giảm thiểu chi phí cân bằng.

- **Chỉ số cân bằng** (balance factor) của một nút  $p$  trên cây AVL =  $lh(p) - rh(p)$

Trong đó:  $lh(p)$  là chiều cao của cây con bên trái của  $p$

$rh(p)$  là chiều cao của cây con bên phải của  $p$

Ta có các trường hợp sau:

$bf(p) = 0$  nếu  $lh(p) = rh(p)$

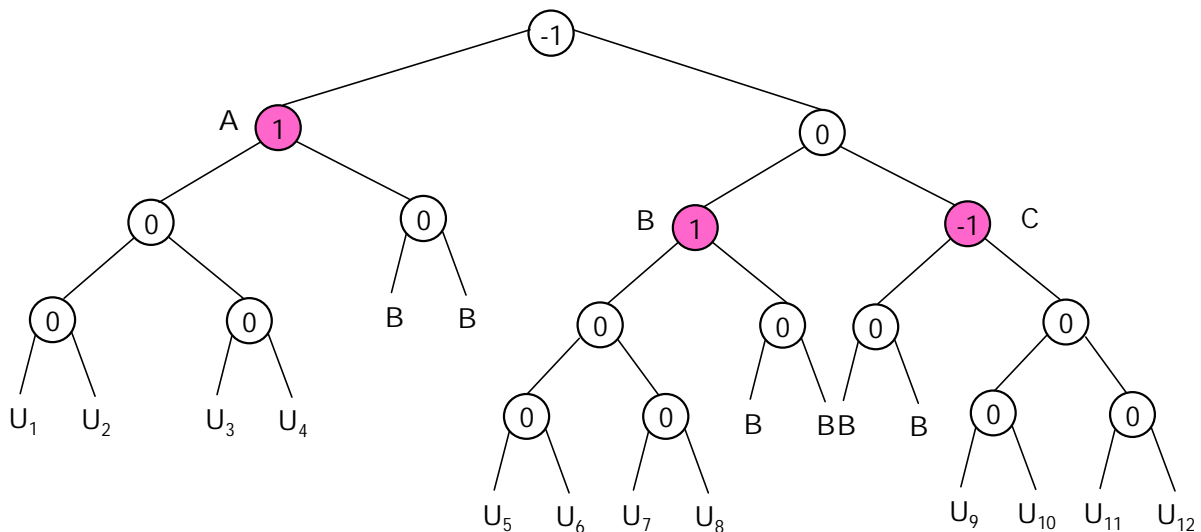
nút  $p$  cân bằng

$bf(p) = 1$  nếu  $lh(p) = rh(p) + 1$

nút  $p$  bị lệch về trái

$bf(p) = -1$  nếu  $lh(p) = rh(p) - 1$

nút  $p$  bị lệch về phải



Hình 5.15. Minh họa các vị trí có thể thêm nút lá vào cây AVL, khi thêm nút lá vào 1 trong các vị trí B thì cây vẫn cân bằng, khi thêm nút lá vào 1 trong các vị

trí U thì cây sẽ mất cân bằng. Các số trên cây là chỉ số cân bằng của các nút trước khi thêm nút

### III.2. Các phép toán trên cây AVL:

\* **Khai báo:** Ta khai báo cây AVL với mỗi nút có thêm trường bf cho biết chỉ số cân bằng của nút đó.

```
struct nodetype
{
    int key;
    int info;
    int bf;
    struct nodetype *left, *right;
};
typedef struct nodetype *NODEPTR;
```

#### III.2.1. Thêm nút:

- **Nội dung:** Thêm 1 nút có khóa x, nội dung a vào cây nhị phân tìm kiếm cân bằng sao cho sau khi thêm thì cây nhị phân vẫn là cây nhị phân tìm kiếm cân bằng.

- **Giải thuật:**

- Thêm nút vào cây như bình thường, nghĩ a là nút vừa thêm sẽ là nút lá.
- Tính lại chỉ số cân bằng của các nút có bị ảnh hưởng
- Kiểm tra xem cây có bị mất cân bằng hay không? Nếu cây bị mất cân bằng thì ta cân bằng lại cây.

\* **Trước hết, ta hãy xét xem các trường hợp nào khi thêm nút làm cây bị mất cân bằng.**

Xem cây hình 5.15, ta nhận thấy:

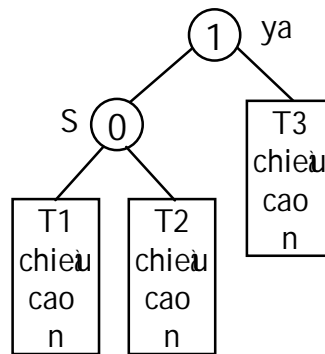
- Nếu thêm nút vào 1 trong 6 vị trí B trên cây thì cây vẫn cân bằng.
- Nếu thêm nút vào 1 trong các vị trí U1 → U12 trên cây thì cây sẽ mất cân bằng.

+ Thêm các nút vào sau bên trái của nút A ( $cf_A = 1$ ) thì cây sẽ bị mất cân bằng vì nút A đang bị lệch trái. Đó là các vị trí U1, U2, U3, U4.

+ Thêm các nút vào sau bên phải của nút C ( $cf_C = -1$ ) thì cây sẽ bị mất cân bằng vì nút C đang bị lệch phải. Đó là các vị trí U9, U10, U11, U12.

**Tóm lại:** Ta có 2 trường hợp khi thêm nút x vào cây AVL làm cây mất cân bằng, đó là thêm các nút vào sau bên trái của nút có  $cf = 1$ , và thêm các nút vào sau bên phải của nút có  $cf = -1$ .

\* **Cân bằng lại cây:** Gọi ya là nút trước gần nhất bị mất cân bằng khi thêm nút x vào cây AVL. Do cả 2 trường hợp bị mất cân bằng khi thêm nút x là tương tự nhau nên ta chỉ xét trường hợp  $bf_{ya}=1$  và nút lá thêm vào là nút sau bên trái của nút ya.



Hình 5.16. Nhánh cây con nút gốc ya trước khi thêm nút.

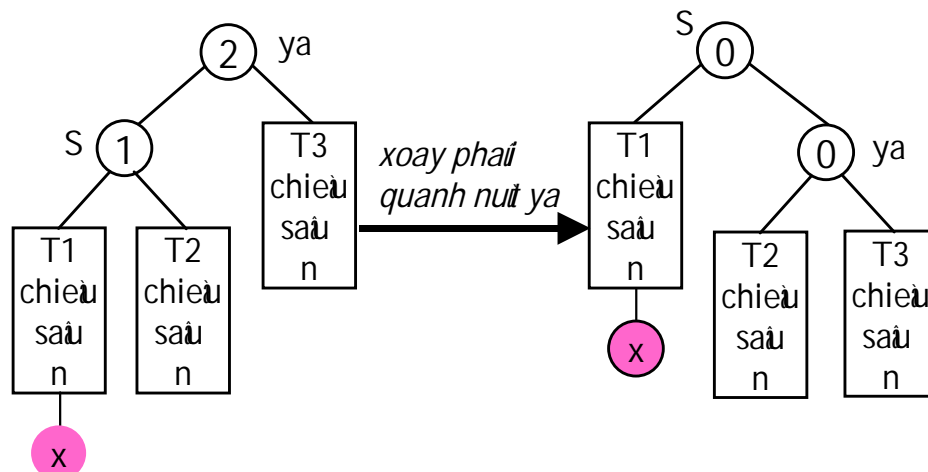
Nhận xét:

- Vì nút ya có  $bf_{ya} = 1$  nên nút ya chắc chắn có nút con bên trái s với  $bf_s = 0$
- Vì ya là nút gần nhất có bf là 1 nên nút s và các nút trước khác của nút x (sẽ thêm vào) có bf là 0.

-Độ cao (T1) = Độ cao(T2) = Độ cao(T3)

**Trường hợp 1a:** Nếu thêm nút mới x vào vị trí nút sau bên trái của s (thuộc nhánh T1) ta xoay phải quanh nút ya

- Nút s sẽ là nút gốc mới của nhánh cây này với  $bf_s = 0$
- Nút ya là nút con bên phải của s với  $bf_{ya} = 0$ .

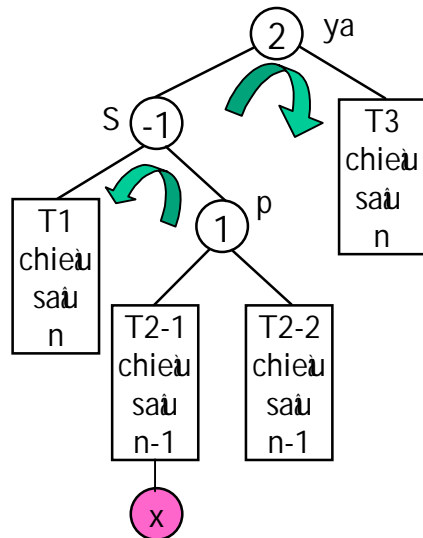


Hình 5.17. Xoay phải quanh nút ya để cân bằng lại cây.

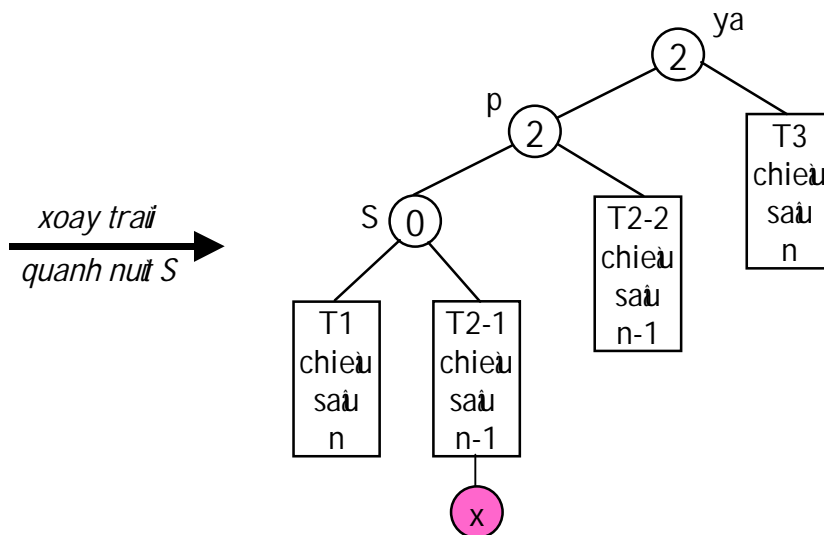


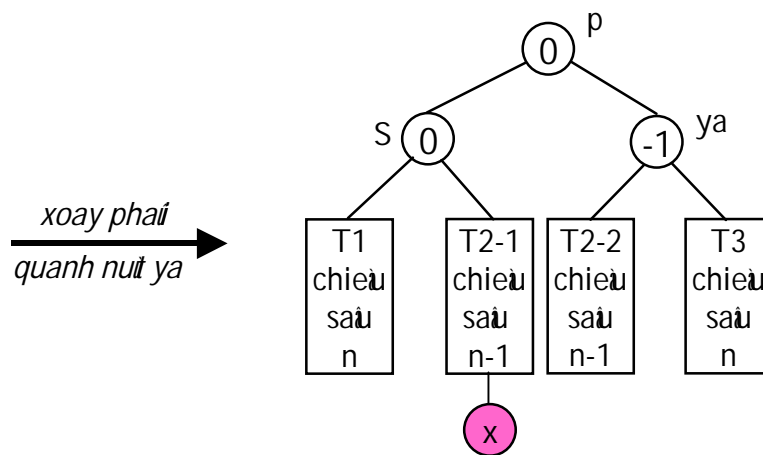
**Trường hợp 1b:** Nếu thêm nút mới  $x$  vào vị trí nút sau bên phải của  $s$  (thuộc nhánh T2) ta xoay 2 lần (xoay kép): xoay trái quanh nút  $s$  và xoay phải quanh nút  $ya$

- Nút  $p$  sẽ là nút gốc mới của nhánh cây này với  $bf_p = 0$
- Nút  $ya$  là nút con bên phải của  $p$  với  $bf_{ya} = -1$
- Nút  $s$  là nút con bên trái của  $p$  với  $bf_s = 0$



Cây AVL sau khi thêm nút  $x$





Hình 5.18. Xoay kép (xoay trái quanh nút s, xoay phải quanh nút ya) để cân bằng lại cây.

Bảng sau đây phân biệt các trường hợp cây bị mất cân bằng khi thêm nút và các phép xoay cây tương ứng để cân bằng lại cây:

Trường hợp	Trước khi thêm nút x	Sau khi thêm nút x	Các phép xoay cây và chỉ số cân bằng mới
1.a	$bf_{ya} = 1$ $bf_s = 0$	$bf_{ya} = 2$ $bf_s = 1$	<i>Xoay phải quanh nút ya</i> $bf_s=0, bf_{ya} = 0$
1.b	$bf_{ya} = 1$ $bf_s = 0$	$bf_{ya} = 2$ $bf_s = -1$	<i>Xoay kép</i> 1. Xoay trái quanh nút s 2. Xoay phải quanh nút ya $bf_s=0, bf_{ya} = -1$
2.a	$bf_{ya} = -1$ $bf_s = 0$	$bf_{ya} = -2$ $bf_s = -1$	<i>Xoay trái quanh nút ya</i> $bf_s=0, bf_{ya} = 0$
2.b	$bf_{ya} = -1$ $bf_s = 0$	$bf_{ya} = -2$ $bf_s = 1$	<i>Xoay kép</i> 1. Xoay phải quanh nút s 2. Xoay trái quanh nút ya $bf_s=0, bf_{ya} = 1$

- Giải thuật:

⇒ Phép xoay trái (Rotate\_Left): xoay trái cây nhị phân tìm kiếm có nút gốc là root, yêu cầu root phải có nút con bên phải (gọi là nút p). Sau khi xoay trái thì nút p trở thành nút gốc, nút gốc cũ trở thành nút con bên trái của nút gốc mới.

Phép xoay trái trả về con trở chỉ nút gốc mới.

```

NODEPTR Rotate_Left(NODEPTR root)
{
    NODEPTR p;
    if(root == NULL)
        printf("Khong the xoay trai vi cay bi rong.");
    else
        if(root->right == NULL)
            printf("Khong the xoay trai vi khong co nut con ben phai.");
        else
        {
            p = root->right;
            root->right = p->left;
            p->left = root;
        }
    return p;
}

```

↪ Phép xoay phải (Rotate\_Right): xoay phải cây nhị phân tìm kiếm có nút gốc là root, yêu cầu root phải có nút con bên trái (gọi là nút p). Sau khi xoay phải thì nút p trở thành nút gốc, nút gốc cũ trở thành nút con bên phải của nút gốc mới.

Phép xoay phải trả về con trỏ chỉ nút gốc mới.

```

NODEPTR Rotate_Right(NODEPTR root)
{
    NODEPTR p;
    if(root == NULL)
        printf("Khong the xoay phai vi cay bi rong.");
    else
        if(root->left == NULL)
            printf("Khong the xoay phai vi khong co nut con ben trai.");
        else
        {
            p = root->left;
            root->left = p->right;
            p->right = root;
        }
    return p;
}

```

⇒ Thêm nút (Insert): thêm nút có khóa x, nội dung a vào cây AVL:

- Thêm nút theo giả i thuật thêm nút vào cây nhị phân tìm kiếm .

- Cân bằng lại cây bằng cách xoay đơn hay xoay kép

```
void Insert(NODEPTR &pavltree, int x, int a)
```

```
{
    NODEPTR fp, p, q, // fp là nút cha của p, q là con của p
        fya, ya,      /* ya là nút trước gần nhất có thể mất cân bằng
                        fya là nút cha của ya */
        s;            // s là nút con của ya theo hướng mất cân bằng
    int imbal;        /* imbal = 1 nếu bị lệch về nhánh trái
                        = -1 nếu bị lệch về nhánh phải */

    // Khởi động các giá trị
    fp = NULL;
    p = pavltree;
    fya = NULL;
    ya = p;
    // tìm nút fp, ya và fya, nút mới thêm vào là nút là con của nút fp
    while(p != NULL)
    {
        if(x == p->info) // bị trùng nội dung
            return;
        if (x < p->info)
            q = p->left;
        else
            q = p->right;
        if(q != NULL)
            if(q->bf != 0) // trường hợp chỉ số cân bằng của q là 1 hay -1
            {
                fya = p;
                ya = q;
            }
        fp = p;
        p = q;
    }
    // Thêm nút mới (nút la) là con của nút fp
    q = New_Node(); // cấp phát vùng nhớ
    q->key = x;
    q->info = a;
    q->bf = 0;
    q->left = NULL;
```

```
    q->right = NULL;
    if(x < fp->info)
        fp->left = q;
    else
        fp->right = q;
/* Hieu chỉnh chỉ số cân bằng của tất cả các nút giữa ya và q, nếu bị lệch
   về phía trái thì chỉ số cân bằng của tất cả các nút giữa ya và q đều là
   1, nếu bị lệch về phía phải thì chỉ số cân bằng của tất cả các nút giữa
   ya và q đều là -1 */
    if(x < ya->info)
        p = ya->left;
    else
        p = ya->right;
    s = p;    // s là con nút ya
    while(p != q)
    {
        if(x < p->info)
        {
            p->bf = 1;
            p = p->left;
        }
        else
        {
            p->bf = -1;
            p = p->right;
        }
    }
// xác định hướng lệch
    if(x < ya->info)
        imbal = 1;
    else
        imbal = -1;

    if(ya->bf == 0)
    {
        ya->bf = imbal;
        return;
    }
    if(ya->bf != imbal)
    {
```

```
    ya->bf = 0;
    return;
}
if(s->bf == imbal) // Truong hop xoay don
{
    if(imbal == 1) // xoay phai
        p = Rotate_Right(ya);
    else // xoay trai
        p = Rotate_Left(ya);
    ya->bf = 0;
    s->bf = 0;
}
else // Truong hop xoay kep
{
    if(imbal == 1) // xoay kep trai-phai
    {
        ya->left = Rotate_Left(s);
        p = Rotate_Right(ya);
    }
    else // xoay kep phai-trai -
    {
        ya->right = Rotate_Right(s);
        p = Rotate_Left(ya);
    }
    if(p->bf == 0) // truong hop p la nut moi them vao
    {
        ya->bf = 0;
        s->bf = 0;
    }
    else
        if(p->bf == imbal)
        {
            ya->bf = -imbal;
            s->bf = 0;
        }
        else
        {
            ya->bf = 0;
            s->bf = imbal;
        }
}
```

```

        p->bf = 0;
    }
    if(fya == NULL)
        pavltree = p;
    else
        if(ya == fya->right)
            fya->right = p;
        else
            fya->left = p;
    }

```

\* Để tạo cây nhị phân tìm kiếm cân bằng, ta sử dụng giải thuật sau:

```

void Create_AVLTree(NODEPTR &root)
{ int khoa, noidung;
  char so[10];
  NODEPTR p;
  do
  { printf("Nhap khoa :");
    gets(so) ;
    khoa = atoi(so);
    if (khoa !=0)
    { printf("Nhap noi dung :");
      gets(so) ;
      noidung = atoi(so);
      if (root==NULL)
      { p = New_Node();
        p->key = khoa;
        p->info = noidung;
        p->bf = 0 ;
        p->left = NULL;
        p->right = NULL;
        root =p;
      }
      else Insert(root,khoa,noidung);
    }
  } while (khoa!=0);          // khóa =0 thì dừng nhập
}

```

Ghi chú : Để tạo cây nhị phân do biến tree quản lý, ta gọi:

```
Create_AVLTree(tree);
```

### III.2.2. Cập nhật cây:

1. **Tìm kiếm (Search):** Tìm nút có khóa bằng x trên cây nhị phân AVL có gốc là root. Nếu tìm thấy x trong cây thì trả về địa chỉ của nút có trị bằng x trong cây, nếu không có thì trả về trị NULL.

Do AVL là cây nhị phân BST nên ta có thể tìm kiếm nhanh bằng phương pháp tìm kiếm nhị phân, và do tính chất lúc này cây cân bằng nên thời gian tìm kiếm sẽ nhanh hơn rất nhiều.

```

NODEPTR search(NODEPTR root, int x)
{
    NODEPTR p;
    p = root;
    while(p != NULL && x!=p->key)
        if(x < p->key)
            p = p->left;
        else
            p = p->right;
    return(p);
}

```

#### 2. **Xóa nút: Remove(root, x):**

- Nội dung: xóa nút có khóa x trên cây AVL với địa chỉ sao đầu root sao cho sau khi xóa thì cây vẫn là AVL.

- Giải thuật:

Nếu root == NULL thì Thông báo ("Không thể xóa được nút x trên cây")

Nếu root != NULL thì

Nếu x < root->info thì :

+ Gọi đệ qui để xóa nút x ở nhánh bên trái của root :

Remove(root->left, x)

+ Gọi balance\_left để cân bằng lại cây nút gốc root nếu nhánh cây con bên trái bị giảm độ cao.

Nếu x > root->info thì :

+ Gọi đệ qui để xóa nút x ở nhánh bên phải của root :

Remove(root->right, x)

+ Gọi balance\_right để cân bằng lại cây nút gốc root nếu nhánh cây con bên phải bị giảm độ cao.

Nếu x == root->info thì :

Xóa nút root như phép toán xóa trên cây nhị phân BST.

- Chương trình nh : tự cài đặt.



### III.2.3. Các phép duyệt cây:

Do cây AVL cũng là cây nhị phân nên ta sẽ áp dụng lại các phương pháp duyệt Preorder, Inorder và Postorder vào cây AVL.

a. Duyệt cây theo thứ tự NLR (Preorder):

```
void Preorder (NODEPTR root)
{
    if(root != NULL)
    {
        printf("%d ", root->info);
        Preorder(root->left);
        Preorder (root->right);
    }
}
```

b. Duyệt cây theo thứ tự LNR (Inorder):

```
void Inorder(NODEPTR root)
{
    if(root != NULL)
    {
        Inorder(root->left);
        printf("%d ", root->info);
        Inorder(root->right);
    }
}
```

c. Duyệt cây theo thứ tự LRN (Posorder):

```
void Posorder(NODEPTR root)
{
    if(root != NULL)
    {
        Posorder(root->left);
        Posorder(root->right);
        printf("%d ", root->info);
    }
}
```

**Bài tập:**

1. Viết lại các chương trình duyệt cây nhị phân theo phương pháp không đệ qui.
2. Viết chương trình tạo một menu thực hiện các mục sau:
  - a. Tạo cây nhị phân tìm kiếm với nội dung là số nguyên (không trùng nhau).
  - b. Liệt kê cây nhị phân ra màn hình theo thứ tự NLR
  - c. Đếm tổng số nút, số nút lá, và số nút trung gian của cây.
  - d. Tính độ cao của cây.
  - e. Loại bỏ nút có nội dung là x trong cây nhị phân BST.
  - f. Thêm nút có nội dung x vào cây nhị phân BST sao cho sau khi thêm thì cây vẫn là BST.
3. Cho một cây nhị phân tree, hãy viết chương trình để sao chép nó thành một cây mới tree2, với khóa, nội dung, và liên kết giống như cây tree.
4. Viết các hàm kiểm tra xem cây nhị phân:
  - a. Có phải là cây nhị phân đúng không.
  - b. Có phải là cây nhị phân đầy không.
5. Viết hàm kiểm tra nút x và y có trên cây hay không, nếu có cả x lẫn y trên cây thì xác định nút gốc của cây con nhỏ nhất có chứa x và y.
6. Cho một cây biểu thức, hãy viết hàm Calculate (NODEPTR root) để tính giá trị của cây biểu thức đó, biết rằng các toán tử được dùng trong biểu thức là:
 
$$+ \quad - \quad * \quad / \quad ^ \quad \% \quad !$$
7. Vẽ lại hình ảnh cây nhị phân tìm kiếm, cây nhị phân tìm kiếm cân bằng nếu các nút thêm vào cây theo thứ tự như sau:
 
$$8 \quad 3 \quad 5 \quad 2 \quad 20 \quad 11 \quad 30 \quad 9 \quad 18 \quad 4$$
8. Nhập vào 1 biểu thức số học, chuyển biểu thức đó thành cây nhị phân, nút gốc là toán tử, nút lá là các toán hạng, biết rằng các toán tử được dùng trong biểu thức là:
 
$$+ \quad - \quad * \quad / \quad ^ \quad \% \quad !$$

## MỤC LỤC

CHƯƠNG I	ĐẠI CƯƠNG VỀ LẬP TRÌNH	1
I.	KHÁI NIỆM THUẬT TOÁN	1
I.1.	Khái niệm	1
I.2.	Các tính chất đặc trưng của thuật toán	1
I.3.	Phân loại	1
II.	MÔ TẢ THUẬT TOÁN BẰNG LƯU ĐỒ	1
II.1.	Lưu đồ	1
II.2.	Các ký hiệu trên lưu đồ	1
II.3.	Một số ví dụ biểu diễn thuật toán bằng lưu đồ	2
III.	CÁC NGÔN NGỮ LẬP TRÌNH & CHƯƠNG TRÌNH DỊCH	5
III.1.	Ngôn ngữ lập trình	5
III.2.	Chương trình dịch	6
CHƯƠNG 2	LÀM QUEN VỚI NGÔN NGỮ C	7
*	GIỚI THIỆU NGÔN NGỮ C	7
I.	CÁC KHÁI NIỆM CƠ BẢN	7
I.1.	Cấu trúc cơ bản của một chương trình C	7
I.2.	Kiểu dữ liệu cơ bản	13
I.3.	Biến	14
I.4.	Hằng	18
I.5.	Phép toán	20
*	Sự chuyển kiểu	29
*	Mức độ ưu tiên của các phép toán	29
I.6.	Chuỗi	30
II.	CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG C	33
II.1.	Cấu trúc tuần tự (Sequence)	33
II.2.	Cấu trúc chọn	34
II.2.1.	Lệnh if else	34
II.2.2.	Lệnh switch_case	35
II.3.	Cấu trúc lặp	37
II.3.1.	Lệnh while	37
II.3.2.	Lệnh do while	38
II.3.3.	Lệnh for	39
*	Phát biểu break, continue, goto	40
Bài tập		41

III. HÀM - ĐỆ QUY	45
III.1. Hàm	45
III.2. Đệ qui (Recursion)	52
IV. STRUCTURE	54
IV.1. Định nghĩa	55
IV.2. Khai báo	55
V. FILE	56
V.1. File văn bản	56
V.2. File nhị phân (file có cấu trúc)	61
V.3. Phát hiện lỗi khi truy xuất tập tin	66
Bài tập	67
CHƯƠNG 3. CÁC THUẬT TOÁN TRÊN CẤU TRÚC DỮ LIỆU MẢNG	69
I. MẢNG KHÔNG SẮP XẾP VÀ THUẬT TOÁN TÌM KIẾM	69
TRÊN MẢNG CHƯA CÓ THỨ TỰ	
I.1. Một số khái niệm về mảng	69
I.2. Thuật toán tìm kiếm trên mảng chưa có thứ tự	71
II. CÁC THUẬT TOÁN SẮP XẾP	73
II.1. Sắp xếp theo phương pháp Bubble_Sort	73
II.2. Sắp xếp theo phương pháp Quick_Sort	75
III. TÌM KIẾM TRÊN MẢNG ĐÃ CÓ THỨ TỰ	79
III.1. Tìm kiếm nhanh bằng phương pháp lặp	79
III.2. Phép tìm kiếm nhị phân	80
III.3. Phép tìm kiếm nhị phân đệ qui	81
Bài tập	82
CHƯƠNG 4 CON TRỎ (POINTER)	84
I. ĐỊNH NGHĨA	84
I.1. Khai báo	84
I.2. Truyền địa chỉ cho hàm	85
II CÁC PHÉP TOÁN TRÊN BIẾN CON TRỎ	85
II.1. Toán tử địa chỉ &	85
II.2. Toán tử nội dung *	85
II.3. Phép cộng trừ biến con trỏ với một số nguyên	86
II.4. Phép gán và phép so sánh	86
II.5. Sự chuyển kiểu	86
II.6. Khai báo một con trỏ hằng và con trỏ chỉ đến đối tượng hằng	87
III. SỰ TƯƠNG QUAN GIỮA CON TRỎ VÀ MẢNG	87

IV. CON TRỎ VÀ CHUỖI-----	91
IV.1. Khai báo-----	91
IV.2. Xét một số ví dụ về các hàm xử lý chuỗi-----	91
IV.3. Mảng con trỏ chỉ đến chuỗi-----	93
Bài tập-----	95
CHƯƠNG 5    CÁC THUẬT TOÁN TRÊN CẤU TRÚC -----	96
DANH SÁCH LIÊN KẾT (LINKED LIST).	
I. KHÁI NIỆM-----	96
II. CÁC PHÉP TOÁN TRÊN DANH SÁCH LIÊN KẾT-----	97
II.1. Tạo danh sách-----	97
II.2. Cập nhật danh sách-----	99
II.3. Duyệt danh sách-----	100
II.4. Tìm kiếm-----	100
II.5. Sắp xếp-----	101
Bài tập-----	102
CHƯƠNG 6    CÁC THUẬT TOÁN TRÊN CẤU TRÚC CÂY-----	104
I. PHÂN LOẠI CÂY-----	104
I.1. Một số khái niệm cơ bản-----	104
I.2. Cách biểu diễn cây-----	106
I.3. Biểu diễn thứ tự các nút trong cây-----	106
II. CÂY NHỊ PHÂN (BINARY TREE)-----	107
II.1. Định nghĩa-----	107
II.2. Các phép toán trên cây nhị phân-----	110
1. Tạo cây-----	110
2. Cập nhật cây-----	112
3. Các phép duyệt cây-----	116
III. CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG (AVL)-----	117
III.1. Định nghĩa-----	117
III.2. Các phép toán trên cây AVL-----	118
III.2.1. Thêm nút-----	118
III.2.2. Cập nhật cây-----	126
III.2.3. Các phép duyệt cây-----	127
Bài tập-----	129

## TÀI LIỆU THAM KHẢO

- |   |   |      |
|---|---|------|
| 1. Kỹ thuật lập trình Turbo C                       | Đỗ Phúc, Nguyễn Phi Khử,<br>Tạ Minh Châu,<br>Nguyễn Đình Tê | 1992 |
| 2. Cấu trúc dữ liệu – Ứng dụng<br>và cài đặt bằng C | Nguyễn Hồng Chương  | 1999 |
| 3. Những viên ngọc Kỹ thuật<br>lập trình            | John Bentley  |      |