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

KỸ THUẬT LẬP TRÌNH

TÀI LIỆU LƯU HÀNH NỘI BỘ

Tháng 4-2016

MỤC LỤC

GIẢI THUẬT

1.1. MỘT SỐ KHÁI NIỆM

1.1.1. *Thuật toán (Algorithm)*

Là một dãy các bước chặt chẽ và rõ ràng, xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho một số hữu hạn lần thực hiện ta thu được kết quả như mong đợi.

1.1.2. *Các đặc trưng của thuật toán*

- Tính dừng (tính kết thúc): Một thuật toán bao giờ cũng phải dừng sau một số hữu hạn các bước thực hiện.
- Tính phổ dụng (tính chính xác): thuật toán có thể giải bất kỳ bài toán nào trong một lớp các bài toán.
- Tính duy nhất: chạy chương trình nhiều lần trên cùng một tập dữ liệu đầu vào phải cho ra cùng một kết quả.
- Tính rõ ràng: giải thuật phải được thể hiện bằng các câu lệnh minh bạch; các câu lệnh được sắp xếp theo thứ tự nhất định.
- Tính khách quan: Một giải thuật dù được viết bởi nhiều người trên nhiều máy tính vẫn phải cho kết quả như nhau.

1.2. Giải thuật

- Trong thực tiễn, người ta chấp nhận các cách giải thường cho kết quả tốt nhưng ít phức tạp, hiệu quả và khả thi. Do đó, khái niệm thuật toán được mở rộng bằng một khái niệm mới là giải thuật. Giải thuật là những cách giải không hoàn toàn đáp ứng đầy đủ các tính chất của một thuật toán nhưng vẫn cho kết quả gần đúng.

  Để thuận tiện, trong tài liệu này sẽ sử dụng khái niệm giải thuật để chỉ chung cho thuật toán và giải thuật.
- Tóm lại, *Giải thuật* là một tập hợp hữu hạn của các chỉ thị hay phương cách được định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán.
- Mô tả thuật toán: có thể mô tả thuật toán bởi một trong ba cách:
  - Mã tự nhiên
  - Mã giả (*pseudocode*).
  - Dùng các biểu tượng được quy định để biểu diễn giải thuật (*flowchart*).

1.2.1. *Mã tự nhiên*

1.2.1.1. Khái niệm

Là viết nội dung chương trình bằng ngôn ngữ tự nhiên giản lược.

1.2.1.2. Quy ước

Phải dễ hiểu, dễ nhớ, nhất quán và không có sự hiểu lầm.

1.2.1.3. *Ví dụ 1.1*

Viết chương trình cho người dùng nhập một số nguyên dương (n). Tính tổng các số từ 1 đến n.

B1: Nhập số n
B2: Kiểm tra giá trị của n
- Nếu n<=0: quay lại B1
- Ngược lại (n>0) chuyển sang B3

B3: Gán giá trị cho biến Tổng=0 và SoDangXet=1
B4: Thực hiện khi SoDangXet <=n
- Cộng dồn SoDangXet vào biến Tổng
- Tăng SoDangXet lên 1 đơn vị.

Quay lại đầu B4
B5: Xuất giá trị của Tổng

1.2.2. *Mã giả (pseudocode)*

1.2.2.1. *Khái niệm*

Dùng ngôn ngữ đã được quy ước trước để diễn đạt thuật toán.

1.2.2.2. *Quy ước*

- Phải dễ hiểu, không cần mô tả chi tiết đến các kỹ thuật dùng trong lập trình.
- Có thể dùng các từ khóa của ngôn ngữ đang dùng, hoặc có thể sử dụng một số từ khóa đã được quy ước trước như:
  - **IF** <Điều kiện> **THEN** … **ENDIF**
  - **IF** <Điều kiện> **THEN. .. ELSE. .. ENDIF**
  - **WHILE** <Điều kiện> **DO** … **ENDWHILE**
  - **DO** … **UNTIL** <Điều kiện>
  - **WRITE** …
  - **READ** …
  - **RETURN** …

1.2.2.3. *Ví dụ 1.2*

Viết chương trình cho người dùng nhập 1 số nguyên dương (n). Tính tổng các số từ 1 đến n.

Input: ∅
Process:

```
DO
    READ n
UNTIL (n>0)
Tong=0
SoDangXet=1
WHILE (SoDangXet <=n) DO
    Tong = Tong + SoDangXet
    SoDangXet = SoDangXet +1.
ENDWHILE
```

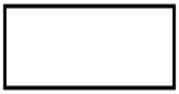
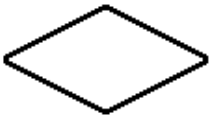
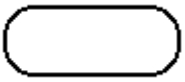
Output: *Tong*

1.2.3. *Lưu đồ*

1.2.3.1. Khái niệm

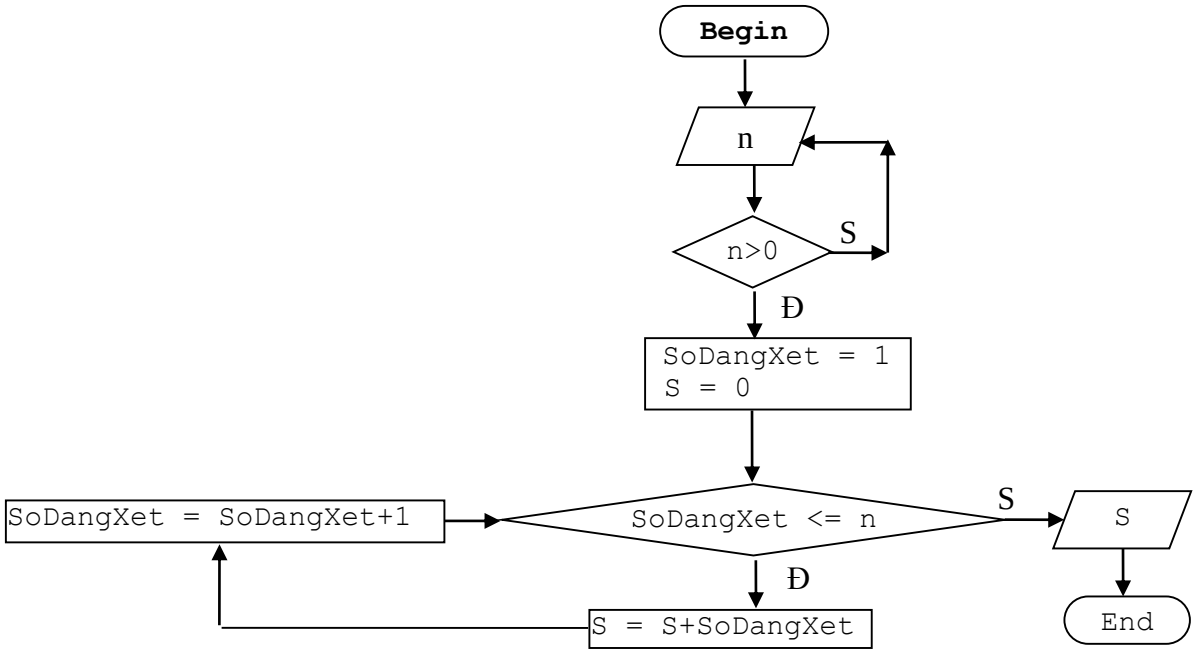
Còn được gọi là sơ đồ khối. Là sơ đồ thể hiện các bước của giải thuật liên quan đến một vấn đề nào đó được đưa vào giải quyết bằng máy tính.

1.2.3.2. Các ký hiệu dùng trong lưu đồ

KÝ HIỆU		Ý NGHĨA
	nhập	Nhập dữ liệu
	Xuất	Xuất dữ liệu
	xử lý	Nhóm lệnh để thực hiện một chức năng nào đó của chương trình (như gán giá trị cho biến, tính toán, …)
	Gọi hàm	Gọi hàm đã định nghĩa.
	Kiểm tra điều kiện	Quyết định rẽ nhánh căn cứ vào điều kiện chỉ định được ghi trong khối.
	Kiểm tra điều kiện	Quyết định rẽ nhánh căn cứ vào điều kiện chỉ định được ghi trong khối. Dùng cho trường hợp có nhiều đường ra của lệnh switch
		Hướng xử lý của lưu đồ.
	điểm đầu/cuối	Điểm đầu hay điểm cuối của lưu đồ.

1.2.3.3. Ví dụ 1.3

Viết chương trình cho người dùng nhập 1 số nguyên dương (n). Tính tổng các số từ 1 đến n.



1.2.4. *Một số ví dụ minh họa*

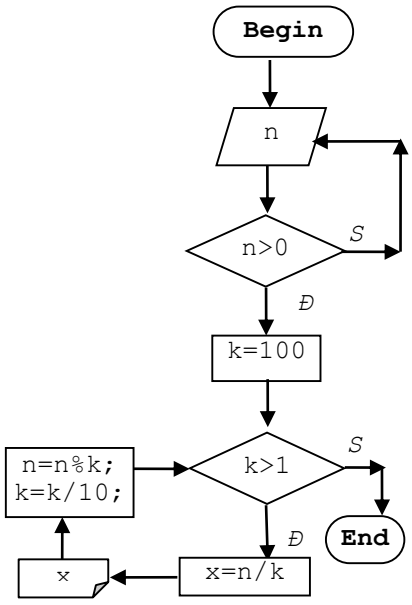
1.2.4.1. Ví dụ 1.4

Thực hiện cả 3 cách mô tả giải thuật cho ví dụ sau: Viết chương trình cho người dùng nhập 1 số nguyên dương (n) có giá trị trong khoảng từ 100 đến 999. In lần lượt từng số hạng (từ trái sang phải) có trong n ra màn hình. Vd, với n=123, sẽ lần lượt in ra từng số theo thứ tự từ trái sang phải là 1 2 3.

(i). Mã tự nhiên	*(ii). Mã giả*

B1: Nhập số *n*
B2: Kiểm tra giá trị của *n*
• Nếu *n*<=0: quay lại B1
• Ngược lại (*n*>0) chuyển sang B3
B3: Gán giá trị cho biến *i*=1; k=100; S=0
B4: Thực hiện khi *k* >1
• Gán x= n/k
• Xuất x
• Loại bỏ số bên trái của n vừa xét (n=n%k)
• Giảm k đi 10 lần (k=k/10)
• Quay lại đầu B4
	INPUT: ∅
	PROCESS:
	DO
	READ *n*
	UNTIL (n>0)
	i=1; k=100; S=0;
	WHILE (*k*>1) DO
	x= n/k
	WRITE x
	n=n%k
	k=k/10
	ENDWHILE
	OUTPUT: đã được lồng trong quá trình xử lý

(iii). Lưu đồ



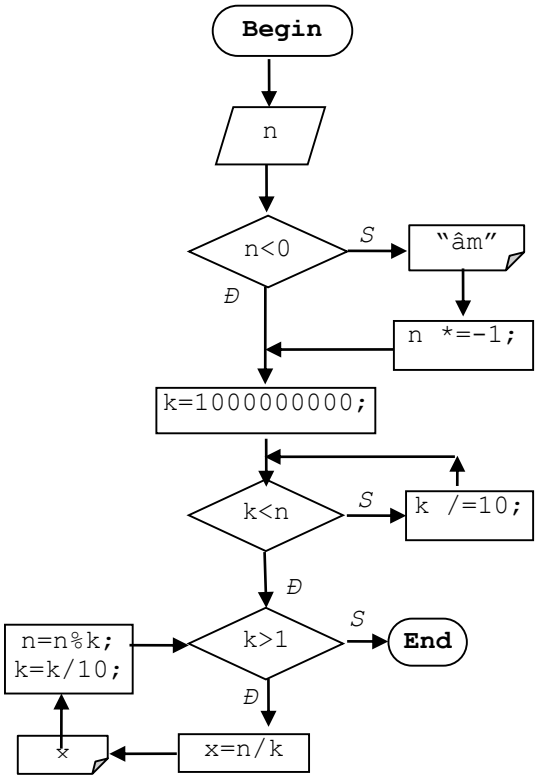
1.2.4.2. Ví dụ 1.5

Mở rộng ví dụ 1.4 để người dùng có thể nhập số nguyên dương (n) có giá trị trong khoảng ± 1 tỷ.

(i). Mã tự nhiên	*(ii). Mã giả*

B1: Nhập số *n*
B2: Kiểm tra giá trị của *n*
• Nếu *n*<0:
▫ In chuỗi "âm" ra màn hình
▫ Gán n=n*-1
• Sang B3
B3: Gán biến k=1000000000;
B4: (Giảm giá trị của k cho phù hợp với giá trị của n)
Trong khi (k>n)
k=k/10
B5: Thực hiện khi *k* >1
• Gán x= n/k
• Xuất x
• Loại bỏ số bên trái của n vừa xét (n=n%k)
• Giảm k đi 10 lần (k=k/10)
• Quay lại đầu B5
	INPUT: ∅
	PROCESS:
	DO
	READ *n*
	UNTIL (n>0)
	IF (n<0) THEN
	WRITE "âm";
	n*=-1;
	ENDIF
	k=1000000000;
	WHILE (*k*>n) DO
	k/=10;
	ENDWHILE
	WHILE (*k*>1) DO
	x= n/k
	WRITE x
	n=n%k
	k=k/10
	ENDWHILE
	OUTPUT: đã được lồng trong quá trình xử lý

(iii). Lưu đồ



1.3. Bài Tập

Thực hiện cả 3 cách mô tả giải thuật để giải các bài tập phần 3.8.2 (trang 26). Biến n (nếu có) sẽ do người dùng nhập khi chương trình thực hiện.

NGÔN NGỮ C

2.1. CÁC LỆNH CƠ BẢN TRONG C

2.1.1. *Từ khóa riêng của ngôn ngữ C*

- Gồm: *break, char, continue, case, do, double, default, else, float, for, goto, int, if, long, return, struct, switch, unsigned, while, typedef, union, void, ...*
- Trong chương trình C tên biến, tên hằng, tên hàm, … (do người dùng tự đặt) không được trùng với từ khóa riêng của ngôn ngữ C.

2.1.2. *Cấu trúc đơn giản của chương trình C*

```
// Các chỉ thị tiền xử lý
#include <tên tập tin.h>
// Các khai báo hằng số (nếu có)
#define MAX 100
// Khai báo biến toàn cục (nếu có, thường ít được dùng)
// Hàm chính của chương trình
void main()
{ //các lệnh của chương trình
}
```

2.1.2.1. *Các chỉ thị tiền xử lý*

2.1.2.1.1. *Chỉ thị #include <tên tập tin. h>*

- Dùng để chèn nội dung của một file khác vào chương trình nguồn tại đúng vị trí mà chỉ thị ***include*** xuất hiện. Khi lập trình trên môi trường. NET, một số tập tin loại này đã được đưa vào thư viện của. NET nên không có phần. ***h*** đi sau. Ví dụ: `#include <iostream>`
- Cách sử dụng:
  - *Khai báo dạng 1*: cho phép C ngầm định tìm tập tin. ***h*** tại thư mục định sẵn (khai báo thông qua menu *Options\Directories*) thường là các tệp nguyên mẫu của thư viện C. Ví dụ: `#include <stidio.h>`
  - *Khai báo dạng 2*: cho phép tìm tập tin. ***h*** theo đường dẫn được chỉ định. Các tập tin. h dạng này thường được tạo bởi lập trình viên và được đặt trong cùng thư mục chứa chương trình. Dạng này cho phép lập trình viên chia một chương trình thành nhiều module đặt trên một số tệp khác nhau để dễ quản lý. Ví dụ: `#include <C:\mylib.h>`

2.1.2.1.2. *Chỉ thị #define*

- Dùng để định nghĩa các tên hằng số (constant) và các macro có phạm vi sử dụng trong toàn chương trình hoặc cho đến khi được định nghĩa lại sau chỉ thị `#undef`.
- Trước khi dịch bộ tiền xử lý sẽ tìm trong chương trình và thay thế bất kỳ vị trí xuất hiện nào của tên_macro bằng giá trị đã được khai báo trong `#define`.
- ví dụ:

```
#define MAX 100     // thay MAX bằng 100
#define TRUE 1      // thay TRUE bằng 1
#define PI 3.1415   // thay thế PI bằng 3.1415
```

2.1.2.1.3. Chỉ thị #if <dãy lệnh ……> #endif

Các chỉ thị này giống như câu lệnh if, mục đích là báo cho chương trình dịch biết đoạn lệnh giữa #if (điều kiện) và #endif chỉ được dịch nếu điều kiện đúng.

2.1.2.1.4. Chỉ thị #ifdef và #ifndef

Chỉ thị này báo cho chương trình dịch biết đoạn lệnh có được dịch hay không khi một tên gọi đã được định nghĩa hay chưa. #ifdef được hiểu là nếu tên đã được định nghĩa thì dịch, còn #ifndef được hiểu là nếu tên chưa được định nghĩa thì dịch.

2.1.2.1.5. Hàm main()

- Các dạng của hàm main:

Dạng 1:
```
void main()
{  /* Các khai báo*/
   /* Các lệnh*/
}
```

Dạng 2:
```
Kiểu_trả_về main(void)
{  /* Các khai báo*/
   /* Các lệnh*/
   return
   Giá_trị_trả_về;
}
```

Dạng 3:
```
Kiểu_trả_về main(int i,
                 char *s[])
{  /* Các khai báo*/
   /* Các lệnh*/
   return
   Giá_trị_trả_về;
}
```

- Ví dụ:

Dạng 1:
```
void   main()
{ printf ("Hello");
}
```

Dạng 2:
```
int main()
{ printf ("Hello");
   return 0;
}
```

Dạng 3:
```
int main(int k, char *s[])
{ printf ("Hello");
   return 0;
}
```

2.1.3. *Biến và khai báo biến*

2.1.3.1. Khai báo biến

- Phải khai báo biến trước khi sử dụng.
- Tên biến gồm chữ cái, ký số, dấu gạch dưới (_) và phải bắt đầu bằng một ký tự.
- Không được trùng với các từ khoá của ngôn ngữ C/C++
- Tên biến có phân biệt ký tự hoa và thường
- Tên biến nên dễ hiểu, súc tích và gợi nhớ.
- Biến khai báo trong một khối được gọi là biến cục bộ, biến không thuộc khối nào gọi là biến toàn cục.
- Biến có tác dụng trong toàn khối kể từ lúc được khai báo.

2.1.3.2. Cú pháp

```
<data type>   variable_name1, variable_name2;
```

Ví dụ:
```
int    sum;
long int  datenum=5;
foat  a,b,c;
```

2.1.4. *Xuất/ nhập dữ liệu*

2.1.4.1. Cú pháp

```
printf ("chuỗi định dạng"[, đối mục 1, đối mục 2,…]);
```

2.1.4.2. Chuỗi định dạng

- Định dạng dữ liệu cần xuất

Ký tự	*Ý nghĩa*
%c	Ký tự đơn
%d	Số nguyên thập phân có dấu

%e	Số chấm động (ký hiệu có số mũ)
%f, %g	Số chấm động (ký hiệu thập phân)
%i	Số nguyên
%o	Số nguyên bát phân (Octal)
%p	Con trỏ (pointer)
%s	Chuỗi
%u	Số nguyên không dấu
%x	Số nguyên thập lục (Hexa)
%%	Xuất ra ký hiệu phần trăm (%)

- Các ký tự điều khiển và ký tự đặc biệt

Ký tự	*Ý nghĩa*	
\a	Alert (bell)	Tiếng kêu bip
\b	Backspace	Tương tự như phím Backspace
\f	Next page	
\n	Newline	Xuống dòng và chuyển con nháy về đầu dòng
\r	Carriage return	Nhảy về đầu dòng (không xuống dòng)
\t	Horizontal tab	Canh cột tab ngang
\'	Single quotation mark	In ra dấu '
\"	Double quotation mark	In ra dấu "
\\	Backslash	In ra dấu \
\0	Null character	
%%	percent	In ra dấu %

2.1.5. *Nhập dữ liệu*

- *Cú pháp:* ***scanf ("chuỗi định dạng"[, đối mục 1, đối mục 2,…]);***
- *Giải thích*: Sử dụng chuỗi định dạng tương tự khi xuất dữ liệu.

2.1.6. *Các kiểu dữ liệu*

2.1.6.1. *Số nguyên*

Tên kiểu	*K/Thước (byte)*	*Miền giá trị*	*Nhỏ nhất*	*Lớn nhất*
char	1	–128 to 127	CHAR_MIN	CHAR_MAX
unsigned char	1	0 đến 255	0	UCHAR_MAX
short	2	–32,768 đến 32,767	SHRT_MIN	SHRT_MAX
unsigned short	2	0 đến 65535	0	USHRT_MAX
int	2	–32,768 đến 32,767	INT_MIN	INT_MAX
unsigned int	2	0 đến 65535	0	UINT_MAX
long	4	–2,147,483,648 đến 2,147,483,647	LONG_MIN	LONG_MAX
unsigned long	4	0 đến 4,394,967,295	0	ULONG_MAX

2.1.6.2. *Số thực*

Tên kiểu	*Kích thước (byte)*	*Miền giá trị*	*Nhỏ nhất*	*Lớn nhất*
float	4	3.4E +/- 38 (7 digits)	FLT_MIN	FLT_MAX
double	8	1.7E +/- 308 (15 digits)	DBL_MIN	DBL_MAX
long double	10	1.2E +/- 4932 (19 digits)	LDBL_MIN	LDBL_MAX

2.1.6.3. Ký tự

- Kiểu ký tự bao gồm 256 ký tự trong đó bao gồm các chữ cái (chữ thường và chữ hoa), chữ số, các dấu, một số các ký hiệu.
- Phân loại kiểu dữ liệu ký tự trong C:

Tên kiểu	*Bytes*	*Tên gọi khác*	*Miền giá trị*
`char`	1	signed char	–128 to 127
`unsigned char`	1	Không có	0 to 255

- Để trình bày một hằng ký tự, ta đặt vào dấu nháy đơn: 'a', 'D' , …

2.1.6.4. Phép biến đổi kiểu (cast operator)

Cú pháp: `datatype(expression)`

Trong đó, `datatype` là kiểu dữ liệu mà ta muốn áp đặt cho cho biểu thức.

2.1.7. Toán tử trong C

2.1.7.1. Toán tử số học

- Mỗi phép toán số học sẽ kết hợp 2 toán hạng.

Phép toán	*Ý nghĩa*	*Ghi chú*
✶	Phép nhân	
/	Phép chia	Tùy thuộc vào kiểu dữ liệu
%	Phép chia lấy phần dư	Chỉ áp dụng cho hai số nguyên
+	Phép cộng	
-	Phép trừ	

- Xác định kiểu dữ liệu của kết quả trong phép toán số học:
  - Nếu cả hai toán hạng là số nguyên thì kết quả thuộc kiểu nguyên.
  - Nếu có một toán hạng nào là kiểu số thực thì kết quả thuộc kiểu thực.
- Sự kết hợp và độ ưu tiên của các toán tử:
  - Phần biểu thức được đặt trong ngoặc sẽ được ưu tiên tính toán trước.
  - Có thể có nhiều cặp dấu ngoặc được sử dụng lồng vào nhau, khi đó biểu thức đặt ở ngoặc trong cùng có ưu tiên cao nhất.
  - Độ ưu tiên của các phép toán theo thứ tự liệt kê ở bảng trên. Nếu có 2 phép toán giống nhau trong cùng biểu thức thì thứ tự xét độ ưu tiên là từ trái sang phải.

2.1.7.1.1. Các toán tử quan hệ (relational operators)

- Các biểu thức sử dụng toán tử quan hệ gọi là biểu thức quan hệ (relation expression).
- Các toán tử quan hệ trong C:

Toán tử	*Công dụng*	*Ví dụ*
<	So sánh nhỏ hơn	*tuoi<30*
>	So sánh lớn hơn	*chieucao >1.7*
<=	So sánh nhò hơn hoặc bằng	*trongluong<=80*
>=	So sánh nhò hơn hoặc bằng	*soluong>=100*
==	So sánh bằng	*loai=='a'*
!=	So sánh khác	*loai!='b'*

- Một biểu thức quan hệ (điều kiện) sẽ có một trong hai kết quả đúng hoặc sai. Nếu đúng thì biểu thức có giá trị 1, ngược lại có giá trị 0.

2.1.7.1.2. Toán tử logic

- Các toán tử logic: ° `AND (&&)`

- OR (||)
- NOT (!)

- Giá trị của biểu thức ghép được cho trong bảng sau. Trong đó `true`=1 và `false`=0:

A	B	A && B	A || B	! A
false	*false*	*false*	*false*	true
true	*false*	*false*	true	*false*
false	true	*false*	true	true
true	true	true	true	*false*

- Khi cần tạo ra các điều kiện phức hợp chúng ta sẽ sử dụng kết hợp giữa các toán tử quan hệ và các toán tử logic. Khi đó các biểu thức quan hệ đơn giản (như +, -, ✶, /,.. .) nên đặt trong cặp dấu ngoặc đơn ( ).

2.1.7.2. Toán tử một ngôi

Giúp tăng hoặc giảm giá trị của biến đếm đi 1 đơn vị.

i++(hay ++i) hoặc i-- hay(--i)

2.1.7.3. Toán tử sizeof

Dùng để xác định kích thước của một loại dữ liệu hoặc một biến.

Ví dụ:

```
printf( "kich thuoc kieu int la: %d", sizeof(int);
```

2.1.7.4. Biểu thức chọn theo điều kiện

- Cú pháp

(điều kiện) ? BT1: BT2

- Biểu thức nhận giá trị BT1 nếu điều kiện khác 0 (ĐÚNG), các trường hợp khác nhận giá trị BT2
- Ví dụ: Có thể định nghĩa sẵn một macro để tìm số lớn:

```
#define max(x, y) ((x>y) ? x: y)
```

2.1.7.5. Toán tử trên bit

- Toán tử trên bit chỉ có tác dụng trên các kiểu số nguyên.

Tên toán tử	*Ý nghĩa*	*Ghi chú*	
&	And	1 & 1 = 1	0 & 1 = 0
		1 & 0 = 0	0 & 0 = 0
|	Or	1 | 1 =1	0 | 1 =1
		1 | 0 = 1	0 | 0 = 0
^	XOr	1 ^ 1 = 0	0 ^ 1 = 1
		1 ^ 0 = 1	0 ^ 0 = 0
>>	Shift phải	A >> n = A/(2^n)	
<<	Shift trái	A << n = A*(2^n)	
~	Lấy phần bù	~1 = 0	~0 = 1

2.1.7.6. Độ ưu tiên của các phép toán

Toán tử	Độ ưu tiên	Trình tự kết hợp
() [] ->	1	Từ trái qua phải
! ~ ++ -- - + * & sizeof	2	Từ phải qua trái
* / %	3	Từ trái qua phải
+ -	4	Từ trái qua phải
<< >>	5	Từ trái qua phải

< <= >= >	6	Từ trái qua phải
== !=	7	Từ trái qua phải
&	8	Từ trái qua phải
|	9	Từ trái qua phải
^	10	Từ trái qua phải
&&	11	Từ trái qua phải
||	12	Từ trái qua phải
?:	13	Từ phải qua trái
= += -= *= /= %= ...	14	Từ phải qua trái

2.1.8. *Lệnh gán dữ liệu cho một biến*

2.1.8.1. *Cú pháp*

```
Tên_biến= biểu thức;
```

2.1.8.2. *Lưu ý*

- Biểu thức có thể là một hằng, một biến hoặc một biểu thức
- Trong một biểu thức gán có thể có nhiều toán tử gán, khi đó biểu thức sẽ được thực hiện từ phải qua trái. Ví dụ `a=b=c=5;`

2.1.9. *Hằng (const)*

- Sử dụng khi không cho phép dữ liệu thay đổi trong chương trình.
- Cú pháp: `const <data type> <name_const> = <value>;`
- Ví dụ: `const float PI=3.1416;`

2.2. CẤU TRÚC ĐIỀU KHIỂN

2.2.1. *Cấu trúc if – else*

2.2.1.1. *Công dụng*

Chỉ cho máy tính chọn thực hiện một trong hai dãy lệnh dựa vào kết quả của một Biểu thức Điều Kiện (btđk).

2.2.1.2. *Cú pháp*

```
if (btđk)
{    Nhóm Lệnh 1;
}
else
{    Nhóm Lệnh 2;
}
```

2.2.1.3. *Cấu trúc if không else*

- Đây là dạng biến đổi của cấu trúc trên, ở dạng này không có phần else.
- Cú pháp:

```
if (btđk)
{    Nhóm Lệnh
}
```

Giải thích: khi `btđk` có giá trị khác 0 thì lệnh sẽ được thực hiện. Ngược lại sẽ không làm gì và thực hiện các lệnh sau đó (nếu có).

2.2.1.4. *Cấu trúc if lồng nhau*

Cho phép trong mỗi phần của if hoặc else có thể chứa một cấu trúc if – else khác. Toàn bộ một cấu trúc if – else lồng nhau được xem như một câu lệnh đơn.

2.2.2. *Cấu trúc switch*

2.2.2.1. *Công dụng*

Khi sử dụng cấu trúc *if – else* để lần lượt so sánh giá trị của một biến/biểu thức với nhiều giá trị, ta nên sử dụng cấu trúc ***switch.***

2.2.2.2. *Cú pháp*

```
switch  (biểu thức)
{
     case  giá_trị_1:
           nhóm lệnh 1;
           break;
     case giá_trị _2:
           nhóm lệnh 2;
           break;
     ...
     case giá_trị_n:
           nhóm lệnh n;
           break;
     default:
           nhóm lệnh default;
}
```

Giải thích:

- *Biểu thức*: có thể là biến hoặc biểu thức.
- *case*: Nếu kết quả của biểu thức sau *switch* bằng với giá trị đi sau *case* nào thì thực hiện các lệnh thuộc về *case* đó (bắt đầu từ dòng lệnh ngay dưới *case* cho đến khi gặp lệnh *break*).
- *Default:* Nếu như giá trị của biểu thức không bằng một giá trị nào trong các giá trị có trong cấu trúc *switch* thì các lệnh thuộc *default* sẽ được thực hiện.:
- *Lệnh break*: có công dụng thoát khỏi *switch* sau khi đã thực hiện một nhóm lệnh thuộc về một nhãn nào đó. Nếu không có lệnh này thì sau khi thực hiện xong các lệnh cần thiết nó sẽ thực hiện tiếp các lệnh của các trường hợp bên dưới.

2.2.2.3. *Ví dụ*

Chương trình nhập vào một số nguyên (0<=so<=9), sau đó in lên màn hình tên gọi của số này:

```
#include <stdio.h>
void main()
{ int so;
  printf("Nhap so nguyen(0-9):");
  scanf("%d", &so);
  switch (so)
  {
    case 0: printf("Khong");
            break;
    case 1: printf("Mot");
            break;
    case 2: printf("Hai");
            break;
    case 3: printf("Ba");
            break;
    case 4: printf("Bon");
            break;
    case 5: printf("Nam");
            break;
    case 6: printf("Sau");
            break;
    case 7: printf("Bay");
            break;
    case 8: printf("Tam");
            break;
    case 9: printf("Chin");
            break;
    default:
        printf("Nhap sai");
  }//switch
}//main
```

2.2.3. *Các cấu trúc lặp*

2.2.3.1. Cấu trúc while

2.2.3.1.1. Công dụng

Cấu trúc while thường được dùng khi cần kiểm tra một số điều kiện trước khi cho nhóm lệnh thực hiện.

2.2.3.1.2. Cú pháp

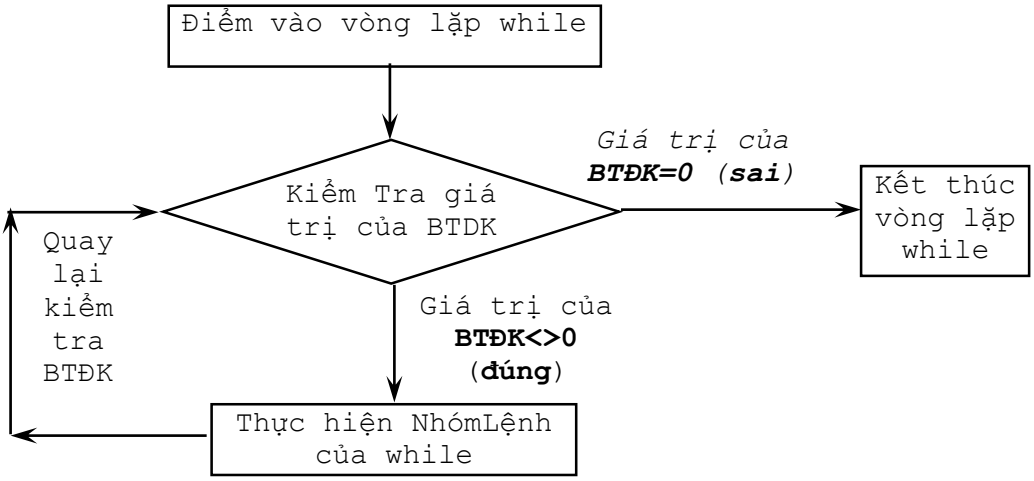
```
while (btđk)
{
    NhómLệnh;
}
```

2.2.3.1.3. Giải thích

- ***btđk*** phải được đặt trong cặp ngoặc đơn.
- *NhómLệnh* bên dưới ***btđk*** sẽ được thực hiện lặp đi lặp lại khi mà ***btđk*** vẫn có giá trị khác không (điều kiện còn đúng). Như vậy tất nhiên ở đâu đó trong đoạn lệnh của `while` phải có một lệnh làm thay đổi giá trị của ***btđk***.
- Câu lệnh bên dưới ***btđk*** phải là một lệnh duy nhất hoặc là một lệnh ghép (nhiều lệnh đặt vào trong {})

2.2.3.1.4. Hoạt động của cấu trúc while

- Kiểm tra giá trị của ***btđk***:
  - Nếu ***btđk*** có giá trị khác không.
    - Thực hiện lệnh bên dưới.
    - Quay lại kiểm tra ***btđk***.
  - Ngược lại: kết thúc cấu trúc while.
- Hình minh họa tiến trình hoạt động của ***while***.



2.2.3.2. Cấu trúc for

2.2.3.2.1. Công dụng

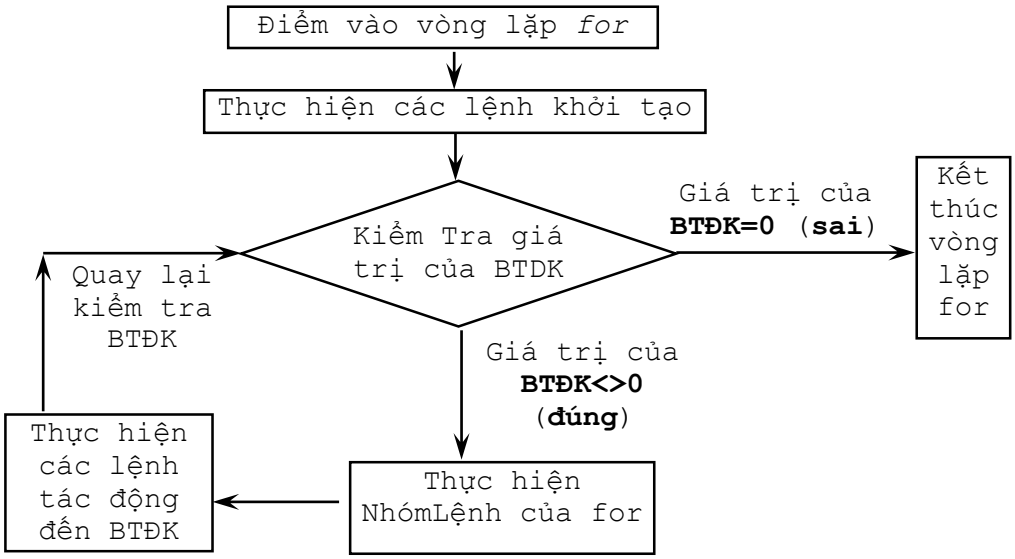
Cấu trúc for thường được dùng trong các trường hợp số lần lặp của vòng lặp được biết trước (giá trị cố định).

2.2.3.2.2. Cú pháp

```
for (lệnh khởi tạo; btđk; lệnh tác động đến điều kiện lặp)
{
    NhómLệnh;
}
```

Các thành phần trong ngoặc của for đều có thể vắng mặt tuy nhiên phải để đầy đủ các dấu chấm phẩy (;).

2.2.3.2.3. Hoạt động của cấu trúc for



2.2.3.3. Cấu trúc do-while

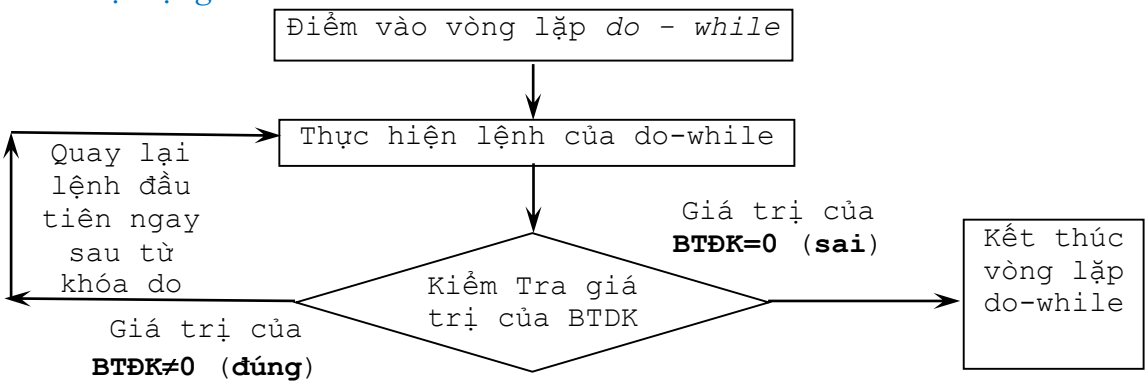
2.2.3.3.1. Công dụng

Cấu trúc do-while thường được dùng khi cho thực hiện trước một số lệnh, sau đó mới thực hiện kiểm tra một số điều kiện.

2.2.3.3.2. Cú pháp

```
do
{ NhómLệnh;
}while (btđk);
```

2.2.3.3.3. Hoạt động của do – while



2.2.3.4. Các cấu trúc lồng nhau

- Toàn bộ một cấu trúc while, for, do-while được xem như một câu lệnh vì vậy ta có thể lồng ghép các cấu trúc này vào nhau.
- Ví dụ: in các bảng cửu chương 2,3,4 lên màn hình

- Cách 1

```
void main()
{
    int i, bcc;
    for (i=1;i<=9;i++)
    {   for (bcc =2; bcc <=4; bcc ++)
            printf("%3d * %2d = %3d", bcc, i, i*bcc);
        printf("\n");
    }
}
```

- Cách 2:

```
void main()
{
    int i=1,j;
    while (i<=10)
    {   for (j=2;j<=4;j++)
            printf("%3d * %2d = %3d", bcc, i, i*bcc);
        printf("\n");
        i++;
    }
}
```

2.2.3.5. Lệnh break và continue

2.2.3.5.1. break

Công dụng: Thoát khỏi các cấu trúc switch, while, for, do-while tại thời điểm break được gọi thi hành mà không cần kiểm tra kết quả của BTĐK.

Khi có nhiều vòng lặp lồng nhau, câu lệnh break sẽ thoát khỏi vòng lặp chứa nó bên trong khối lệnh lặp.

2.2.3.5.2. continue

Công dụng: Khi lệnh continue được gọi thì chương trình sẽ quay trở về đầu vòng lặp để bắt đầu lần lặp mới. Nếu có các lệnh còn lại (cùng trong vòng lặp) đặt sau continue sẽ không được thực hiện.

2.3. MỘT SỐ THƯ VIỆN THƯỜNG DÙNG TRONG C

- Hàm thư viện là những hàm đã được định nghĩa sẵn trong một thư viện nào đó, muốn sử dụng các hàm thư viện thì phải khai báo thư viện trước khi sử dụng bằng lệnh `#include <tên thư viện.h>`
- Ý nghĩa của một số thư viện thường dùng:

(i) **`stdio.h`**: Thư viện chứa các hàm vào/ ra chuẩn (standard input/output). Gồm các hàm `printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw(),...`

(ii) **`conio.h`**: Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm `clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),...`

(iii) **`math.h`**: Thư viện chứa các hàm tính toán gồm các hàm `abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),...`

(iv) **`alloc.h`**: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhơ. Gồm các hàm `calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(),...`

(v) **`io.h`**: Thư viện chứa các hàm vào ra cấp thấp. Gồm các hàm `open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),...`

(vi) **`graphics.h`**: Thư viện chứa các hàm liên quan đến đồ họa. Gồm `initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(),...`

HÀM (Function)

3.1. TỔ CHỨC CHƯƠNG TRÌNH CÓ NHIỀU HÀM

3.1.1. *Khái niệm*

- Sử dụng hàm trong chương trình là sự chia nhỏ của chương trình cần thực hiện.
- Mỗi hàm là một đoạn chương trình độc lập ***thực hiện trọn vẹn một công việc nhất định*** sau khi thực hiện xong, hàm có thể sẽ trả về giá trị cho chương trình gọi nó.

3.1.2. *Mục đích khi sử dụng các hàm con*

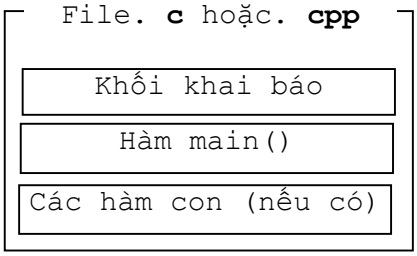
- Khi có một công việc giống nhau cần thực hiện ở nhiều vị trí.
- Chia chương trình có nhiều chức năng thành các chức năng con để chương trình được trong sáng, dễ hiểu, dễ quản lý. Trong mỗi chức năng con vừa có lại có thể phân chia thành những chức năng nhỏ/chi tiết hơn.

3.1.3. *Một số lưu ý khi xây dựng hàm trong C*

- Nội dung của hàm.
- Sự tương tác của nó với những hàm khác, bao gồm việc truyền dữ liệu chính xác vào trong hàm khi gọi hàm thực hiện và giá trị mà hàm sẽ trả về khi thực hiện xong.

3.1.4. *Cấu trúc một chương trình trong C*

3.1.4.1. *Cách 1: sử dụng 1 file cho toàn bộ chương trình*



- *Khối khai báo*: Bao gồm các khai báo về:
  - Khai báo tên các thư viện chuẩn của C được sử dụng trong chương trình.
  - Hằng số sẽ sử dụng trong chương trình.
  - Các kiểu dữ liệu tự định nghĩa.
  - Các biến toàn cục.
  - Hàm con (các nguyên mẫu hàm - prototype).
- *Hàm chính* (main()): Chứa các biến, các lệnh và các lời gọi hàm cần thiết trong chương trình.
- *Các hàm con*:
  - Được sắp xếp sao cho mỗi hàm nằm trên 1 đoạn riêng.
  - Không đặt nội dung của hàm này chứa trong hàm khác, hoặc nội dung của 2 hàm có phần giao nhau.
  - Không cần quan tâm thứ tự sắp xếp trước/sau của các hàm.

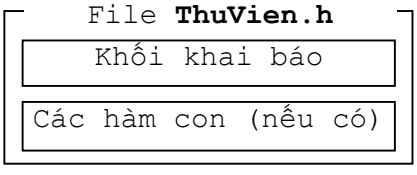
Ví dụ 3.1: toàn bộ chương trình sau được chứa trong file có tên *BaiTap.cpp*

```
//Khối khai báo
  // Khai báo thư viện cần dùng
  #include<conio.h>
  #include<stdio.h>
  #include<string.h>
  #include<dos.h>
  #include<process.h>
  //Khai báo nguyên mẫu hàm
  void ThayThe(char * S, char *St );
  void Doc1Sector(int vt);
  void Ghi1Sector(int vt);
```

```
//Hàm main
void main()
{    unsigned char buf[512];
     char S[20], St[20];
     printf("Nhap chuoi ban dau: ");
     gets(S);
     printf("Nhap chuoi can thay the:");
     gets(St) ;
     printf("\nXin cho…";
     TimVaThayThe(S,St,buf);
     printf("\n Thanh cong. Chuoi ket qua la: %s", S);
     getch();
}
//Khối chứa các hàm con
 void ThayThe(char * S, char *St )
 {  int l=strlen(St);
    for(int i=0;i<l;i++)
          S[i]=St[i];
 }
 void Doc1Sector(int vt, char buf[512])
 {    if(absread(0,1,vt,buf))
      {   printf(""\n loi doc dia,  nhan enter thoat");
          getch();
          exit(1);
      }
 }
 void Ghi1Sector(int vt, char buf[512])
 {    if(abswrite(0,1,vt,buf))
      {   printf("\n loi ghi dia, nhan enter thoat");
          getch();
          exit(1);
      }
 }
 void TimVaThayThe(char * S, char *St, unsigned char buf[])
 {    for(int i=33;i<=500;i++)
      {   Doc1Sector(i, buf);
          char * p=strstr(buf, S);
          if(p)
          {   ThayThe(p, St);
              Ghi1Sector(i, buf);
          }
      }
 }
```

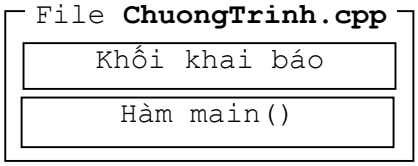
3.1.4.2. *Cách 2: sử dụng 2 file cho toàn bộ chương trình*

3.1.4.2.1. *File thư viện chứa các hàm do người dùng tự tạo*



- *Khối khai báo*: Bao gồm các khai báo về:
  - Khai báo tên thư viện chuẩn của ngôn ngữ C được sử dụng trong chương trình.
  - Các kiểu dữ liệu tự định nghĩa.
  - Hằng số sẽ sử dụng trong chương trình.
  - Các biến toàn cục.
  - Hàm con (các nguyên mẫu hàm - prototype).
- *Các hàm con*: (tương tự như cách 1)

3.1.4.2.2. File chương trình



- *Khối khai báo*: khai báo tên file thư viện chứa các hàm do người dùng tự tạo. Tên file (kể cả phần mở rộng) được đặt trong dấu nháy đôi.
- *Hàm chính* (main()): Chứa các biến, các lệnh và các lời gọi hàm cần thiết trong chương trình.

Ví dụ 3.2:

- Nội dung file ***ThuVien.h***

```
//Khối khai báo
  // Khai báo thư viện cần dùng
  #include<conio.h>
  #include<stdio.h>
  #include<string.h>
  #include<dos.h>
  #include<process.h>
  //Khai báo nguyên mẫu hàm
  void ThayThe(char * S, char *St );
  void Doc1Sector(int vt);
  void Ghi1Sector(int vt);
//Khối chứa các hàm con
 void ThayThe(char * S, char *St )
 {  int l=strlen(St);
    for(int i=0;i<l;i++)
          S[i]=St[i];
 }
 void Doc1Sector(int vt, char buf[512])
 {    if(absread(0,1,vt,buf))
      {   printf(""\n loi doc dia,  nhan enter thoat");
          getch();
          exit(1);
      }
 }
 void Ghi1Sector(int vt, char buf[512])
 {    if(abswrite(0,1,vt,buf))
      {   printf("\n loi ghi dia, nhan enter thoat");
          getch();
          exit(1);
      }
 }
 void TimVaThayThe(char * S, char *St, unsigned char buf[])
 {    for(int i=33;i<=500;i++)
      {   Doc1Sector(i, buf);
          char * p=strstr(buf, S);
          if(p)
          {   ThayThe(p, St);
              Ghi1Sector(i, buf);
          }
      }
 }
```

- Nội dung file ***ChuongTrinh.cpp***

```
//Khối khai báo
  #include "ThuVien.h"
//Hàm main
void main()
{   unsigned char buf[512];
    char S[20], St[20];
```

```
        printf("Nhap chuoi ban dau: ");
        gets(S);
        printf("Nhap chuoi can thay the:");
        gets(St) ;
        printf("\nXin cho…";
        TimVaThayThe(S,St,buf);
        printf("\n Thanh cong. Chuoi ket qua la: %s", S);
        getch();
    }
```

3.2. PHÂN LOẠI THAM SỐ TRUYỀN CHO HÀM

- Các tham số đầu vào của một hàm được gọi là tham số hàm.
- Phân loại: có hai loại tham số là:
  - Tham trị : Giá trị của tham số *KHÔNG thay đổi* sau khi hàm thực hiện.
  - Tham biến: Giá trị của tham số *CÓ thay đổi* sau khi hàm thực hiện.

3.2.1. *Sử dụng tham số của hàm là tham trị*

- Tham trị là cách gọi tắt của "*tham số hình thức* ***trị***".
- Tham số dạng này chỉ mang ý nghĩa là *dữ liệu đầu vào*.
- Cơ chế truyền và xử lý dữ liệu của cách dùng tham trị như sau:
  - *Hàm gọi* (ví dụ như hàm main) có thể dùng hằng hoặc biến để truyền giá trị cho tham số của chương trình được gọi.
  - *Hàm được gọi* sẽ nhận giá trị truyền cho nó và lưu những giá trị này vào một vùng nhớ gọi là vùng tạm trong bộ nhớ. Khi đó mọi lệnh tác động lên tham số trong chương trình con này sẽ tác động lện vùng nhớ tạm. Vùng nhớ tạm sẽ bị xóa ngay sau khi hàm được gọi kết thúc.

3.2.2. *Sử dụng tham số của hàm là tham biến (hay tham chiếu)*

- Tham biến là cách gọi tắt của "*tham số hình thức* ***biến***".
- Theo cách này thì khi gọi thực hiện một hàm có tham số thì hàm gọi phải dùng biến để truyền dữ liệu cho các tham số của hàm được gọi. Lúc này tham số sẽ nhận địa chỉ trong bộ nhớ RAM của biến đã truyền cho nó, và mọi tác động lên tham số của hàm được gọi sẽ tác động trực tiếp lên biến tương ứng của chương trình gọi.
- Có thể sử dụng *tham biến* bằng 1 trong 2 cách sau:
  - *Tham biến* vừa là dữ liệu đầu vào vừa là dữ liệu đầu ra.
  - *Tham biến* chỉ đóng vai trò của dữ liệu đầu ra (kết quả thực hiện của hàm). Khi đó, dữ liệu đầu vào không ảnh hưởng đến kết quả thực hiện của hàm.
- Cú pháp để khai báo một tham số dạng tham biến:
  - data_type & //khai báo trên dòng prototype
  - data_type &reference_name //khai báo trên *function* header
- Ví dụ 3.3: chương trình sau sử dụng hàm `swap`, hàm `swap` có công dụng nhận giá trị của hai biến và đổi giá trị của hai biến cho nhau:

```
#include <stdio.h>
#include <conio.h>
void  swap(int&, int&);
void main()
{   int    i=5,j=10;
    printf("\n TRUOC khi goi ham swap,\ni=%d;j=%d\n",i,j);
    swap(i,j);
    printf("\n SAU khi goi ham swap,\n i=%d;j=%d\n",i,j);
    getch();
}
```

```
void  swap(int &x, int &y)
{   int temp=x;
        x=y;
        y=temp;
}
```

3.3. TRIỂN KHAI XÂY DỰNG HÀM

Gồm 3 bước chính:

- Khai báo tiêu đề của hàm
- Viết nội dung cho hàm
- Khai báo nguyên mẫu hàm

3.3.1. *Khai báo tiêu đề hàm (Function Header)*

3.3.1.1. *Cú pháp khai báo tiêu đề của một hàm*

```
<Kiểu dữ liệu trả về của hàm> Tên hàm  ([danh sách các tham số])
```

3.3.1.2. *Đặt tên cho hàm*

Mỗi hàm có một tên để phân biệt, tên hàm trong C được đặt theo quy tắc sau:

- Chỉ được dùng chữ cái, chữ số hoặc underscore (_) để đặt tên hàm.
- Ký tự đầu tiên phải là một chữ cái hoặc dấu underscore (_).
- Tên hàm không được trùng tên với từ khóa riêng của C.
- Có phân biệt chữ hoa, chữ thường.
- Số ký tự tối đa là 31.

Nên đặt tên hàm sao cho tên gọi *đúng với chức năng hay mục đích thực hiện của hàm và gợi nhớ.*

3.3.1.3. *Kiểu dữ liệu trả về của hàm*

- Xác định dựa vào kết quả của bài toán (Output). Gồm 2 loại:
  - **void:**
    - Hàm không trả về giá trị.
    - Thường dùng cho những chức năng*: Nhập/xuất dữ liệu, thống kê, sắp xếp, liệt kê.*

```
void Tên_hàm (danh sách các tham số)
{     Khai báo các biến cục bộ
     Các câu lệnh/khối lệnh hay lời gọi đến hàm khác
}
```

  - **Kiểu dữ liệu cơ bản (rời rạc/ liên tục) hay kiểu dữ liệu có cấu trúc:**
    - Kiểu dữ liệu tùy theo mục đích của hàm cần trả về giá trị gì thông qua việc phân tích bài toán.
    - Thường dùng cho những chức năng: *Đếm, kiểm tra, tìm kiếm, tính trung bình, tổng, tích,.. .*

```
<Kiểu dữ liệu> Tên_hàm ([danh sách các tham số])
{
       <Kiểu dữ liệu> kq;
       Khai báo các biến cục bộ
       Các câu lệnh/khối lệnh hay lời gọi đến hàm khác
       return kq;
}
```

3.3.1.4. Danh sách tham số

- Cần quan tâm:
  - Số lượng tham số.
  - Thứ tự của các tham số. Khi có nhiều tham số, các tham số phải cách nhau bởi dấu phẩy (,).
  - Kiểu dữ liệu của từng tham số.
- Ngoại trừ hàm *main* tất cả các hàm khác (không phải hàm có sẵn của C) có trong chương trình đều phải khai báo nguyên mẫu.

3.3.2. *Nội dung của hàm (hay phần thân hàm – Function Body)*

- Bao gồm các lệnh, các phép toán sẽ tác động lên các giá trị được truyền cho hàm thông qua các tham số, để tạo ra kết quả.
- Mỗi hàm sẽ được định nghĩa một lần trong chương trình.
- Mỗi hàm có thể được gọi thực hiện một (hoặc nhiều lần) bởi một hàm khác có trong chương trình.
- Trong nội dung của hàm có thể gọi bất kỳ hàm nào khác có trong chương trình.

3.3.3. *Khai báo nguyên mẫu hàm (prototypes)*

- Nguyên mẫu hàm thực chất là dòng đầu của hàm thêm dấu chấm phẩy (**;)** vào cuối, tuy nhiên tham số trong nguyên mẫu hàm có thể chỉ có kiểu dữ liệu mà bỏ qua phần tên của tham số.
- Các *prototype* thường được khai báo ở đầu chương trình sau các dòng *#include*.

3.3.4. *Ví dụ 3.4 (về khai báo nguyên mẫu hàm)*

Chương trình in lên màn hình giá trị lớn nhất của hai số nguyên được nhập từ bàn phím, trong chương trình có một hàm *findmax* dùng để tìm giá trị lớn nhất trong hai số:

```
#include <stdio.h>
int TimMax(int, int);     //prototype declare
void main()
{
    int  so1, so2;
    printf("Nhap  so thu nhat:");
    scanf("%d", &so1");
    printf("Nhap  so thu hai:");
    scanf("%d", &so2");
    printf("Gia tri lon nhat trong hai so vua nhap la:%d\n",
                                          TimMax(so1,so2);
}
int TimMax(int x, int y)     // Function Header
{
   if (x>=y) return x;          // Function body
   else return y;               // Function body
}
```

3.3.5. *Lời gọi hàm*

- Lời gọi hàm giống như một lệnh, nó xuất hiện trong chương trình khi có yêu cầu gọi thực hiện một hàm nào đó thực hiện. Lời gọi hàm bao gồm tên hàm các dữ liệu truyền cho hàm được gọi. Nếu nguyên mẫu của hàm có tham số thì khi gọi hàm phải truyền giá trị.
- Số lượng dữ liệu truyền cho hàm phải bằng số lượng tham số khi khai báo nguyên mẫu và đúng thứ tự đã khai báo (cùng kiểu dữ liệu của tham số).

- Minh họa lời gọi hàm CÓ tham số

Nguyên mẫu hàm

```
int addition (int a, int b)
```

Lời gọi hàm

```
z = addition ( 5 , 3 );
```

- Minh họa lời gọi hàm KHÔNG có tham số

Nguyên mẫu hàm

```
void duplicate (int& a,int& b,int& c)
```

Lời gọi hàm

```
duplicate ( x , y , z );
```

3.4. VÍ DỤ (về trình tự khi xây dựng hàm)

3.4.1. *Ví dụ 3.5*

Viết hàm nhận số nguyên dương n và in ra màn hình các ước số của n.

3.4.1.1. Phân tích bài toán

- **Input:** n (tham số)
  - Kiểu dữ liệu: số nguyên dương (*unsigned int*).
  - Giá trị n không bị thay đổi sau quá trình hàm thực hiện ⇨ Tham số của hàm là tham trị (không có ký hiệu & trước tên biến).
- **Output:** chỉ in ra các ước số của n mà không trả về giá trị nên kiểu dữ liệu của hàm là *void*.
- **Xác định tên hàm:** Hàm này dùng in ra các ước số của n nên có thể đặt là *LietKeUocSo.*

Ta có nguyên mẫu hàm:

```
void LietKeUocSo (unsigned int n);
```

3.4.1.2. Bổ sung nội dung hàm

```
#include<conio.h>
#include<stdio.h>
//Khai báo nguyên mẫu hàm
void LietKeUocSo (unsigned int n);
void main()
{   unsigned int n;
    printf("Nhap n: )";
    scanf ("%d",&n);
    printf("Cac uoc so cua n: )" ;
    LietKeUocSo(n); // hàm có kiểu dữ liệu hàm là void
                    //nên không cần gán giá trị vào biến
    getch();
}
void LietKeUocSo (unsigned int n)
{   for(int i=1; i<=n; i++)
        if (n%i==0)
          printf("%d \t",i);
}
```

3.4.2. *Ví dụ 3.6*

Viết chương trình nhập số nguyên dương n và tính tổng

$$S = 1 + 2 + 3 + \cdots + n \quad , \text{với } n>0$$

3.4.2.1. Phân tích bài toán

- **Input:** n (tham số)
  - Kiểu dữ liệu: số nguyên dương (*unsigned int*).
  - Giá trị n không bị thay đổi sau quá trình hàm thực hiện ⇨ Tham số của hàm là tham trị (không có ký hiệu & trước tên biến).
- **Output**: Trả về giá trị tổng của các số từ 1 đến n. Do n là số nguyên dương nên S cũng là số nguyên dương ⇨ Kiểu trả về của hàm là *unsigned int* (hoặc *unsigned long* cho trường hợp giá trị của tổng lớn hơn 2 bytes).
- **Xác định tên hàm**: Hàm này dùng tính tổng S nên có thể đặt là *TongS.*

Ta có nguyên mẫu hàm:

```
unsigned long TongS ( unsigned int n );
```

3.4.2.2. Bổ sung nội dung hàm

```
//Khai báo nguyên mẫu hàm
unsigned long TongS (unsigned int n);
void main()
{   unsigned int n;
    unsigned long kq;
    printf("Nhap n: )";
    scanf ("%d",&n);
    /* hàm có kiểu dữ liệu hàm không phải là void nên cần gán
    giá trị vào biến ở vế trái của biểu thức */
    kq  = TongS(n);
    printf("Tong can tinh la: %d", kq);
    getch( );
}
unsigned long TongS (unsigned int n)
{   unsigned long S=0;
    int i=1;
    while(i<=n)
    {   S+=i;
        i++;
    }
    return S;
}
```

3.5. PHẠM VI CỦA BIẾN

3.5.1. *Phân loại*

- **Biến cục bộ:** Là các biến được khai báo trong thân của một hàm nào đó. Các biến này chỉ có giá trị trong phạm vi hàm khai báo nó.
- **Biến toàn cục:** Là các biến được khai báo bên ngoài các hàm. Những biến này có giá trị với tất cả các hàm được khai báo sau nó.

3.5.2. *Ví dụ 3.7 (về phạm vi của biến)*

```
int a;
void  func(int );
void main()
{
    int   b=2;
    a=1;
    printf("\nTRUOC khi goi ham func, trong ham main() a=%d,
                                                    b=%d\n",a,b);
    func();
    printf("\nSAU khi goi ham func, trong ham main() a=%d,
                                                    b=%d\n",a,b);
}
void func()
{
    int b=3;
    printf("\nTRONG ham func, a=%d, b=%d\n",a,b);
    a=4;
}
```

📖 *Giải thích*:

- Trong ví dụ trên a là biến toàn cục, và b là biến cục bộ có cả trong `main` và `func`. Lệnh printf đầu của hàm `main` sẽ in lên kết quả:
  `TRUOC khi goi ham func, trong ham main(), a=1, b=2`
- Khi hàm `func` thực hiện, do b là biến cục bộ của hàm nên hàm sẽ in ra kết quả:
  `TRONG ham func a= 1; b=3`
- Lệnh gán cuối cùng trong hàm `func` (`a=4;`) đã làm thay đổi giá trị của biến toàn cục a, nên lệnh printf cuối của hàm `main` sẽ in lên kết quả:
  `SAU khi goi ham func, trong ham main(), a=4, b=2`
- Trong ví dụ trên, nếu ta khai báo biến a trong hàm `main` thì hàm `func` sẽ không hiểu được biến này, do đó khi biên dịch sẽ gây ra lỗi.

3.5.3. *Trùng tên giữa 2 biến cục bộ và toàn cục*

- Khi biến toàn cục và biến cục bộ trùng tên thì các biến cục bộ sẽ ưu tiên được hiểu trong hàm khai báo nó.
- Ví dụ 3.8:

```
int a=2;
void  func(int );
void main()
{
    a=2;
    printf("\nTRUOC khi goi ham func, a=%d\n",a);
    func();
    printf("\nSAU khi goi ham func, a=%d\n",a);
}
void func()
{
    int a=3;
    printf("\nTRONG ham func,a=%d\n",a);
}
```

📖 *Giải thích*:

- Trong ví dụ 3.8, a là biến toàn cục, trong hàm `main` a được gán giá trị là 2. Lệnh printf đầu của hàm `main` sẽ in lên kết quả:

```
TRUOC khi goi ham func, a=2
```

- Khi hàm `func` thực hiện, do biến a được khai báo là biến cục bộ của hàm nên hàm sẽ in ra kết quả:

```
TRONG ham func a=3
```

- Lệnh thứ tư trong hàm `main` gán giá trị cho biến toàn cục a tang lên 1 đơn vị (`a++;`), nên lệnh printf cuối của hàm `main` sẽ in lên kết quả:

```
SAU khi goi ham func, a=2
```

3.5.4. *Lưu ý khi sử dụng biến toàn cục*

- Sẽ dễ dẫn đến phá vỡ sự độc lập của các hàm.
- Làm mất đi sự cẩn thận cần thiết của người lập trình là:
  - Phải xác định rõ kiểu dữ liệu của tham số.
  - Quản lý biến cục bộ
  - Giá trị trả về của hàm.
- Do giá trị của biến toàn cục có thể thay đổi bởi bất cứ một hàm nào khai báo sau nó, vì vậy việc phát hiện ra những lỗi giải thuật có trong chương trình là rất khó khăn.

3.6. CÁC BƯỚC ĐỂ VIẾT MỘT CHƯƠNG TRÌNH CÓ SỬ DỤNG HÀM

Để viết các chương trình có sử dụng hàm ta theo các bước chung như sau:

- *Bước 1*: Xác định các bước cần làm để chia vấn đề muốn giải quyết thành các vấn đề nhỏ hơn.
- *Bước 2*: Với mỗi vấn đề nhỏ, chúng ta xác định các đối tượng có sẵn (còn gọi là "Các đối tượng nhập" hay " Dữ liệu vào"), các đối tượng cần phải tính (còn gọi là "Các đối tượng xuất" hay " Dữ liệu ra"), các đối tượng trung gian (các đối tượng cần sử dụng tạm trong quá trình tính toán).
- *Bước 3*: Chuyển mỗi vấn đề nhỏ thành một chương trình con (Hàm), các đối tượng nhập được chuyển thành tham trị, các đối tượng xuất được chuyển thành tham chiếu, các đối tượng trung gian được chuyển thành các biến riêng (còn được gọi là biến cục bộ) của hàm.
- *Bước 4*: Viết chương trình chính (hàm main): Khai báo các đối tượng được dùng trong chương trình chính, gọi các các chương trình con theo một trình tự thích hợp.

3.7. CÁC LỖI THƯỜNG GẶP KHI XÂY DỰNG VÀ SỬ DỤNG HÀM

- Lỗi thường gặp là truyền dữ liệu cho tham số khi gọi hàm thực hiện không chính xác chẳng hạn như truyền thiếu/thừa, dùng hằng để truyền cho tham chiếu.
- Cần lưu ý khi có hai biến có cùng tên cùng xuất hiện trong hàm gọi lẫn làm được gọi.
- Thiếu các nguyên mẫu của các hàm có trong chương trình.
- Thiếu dấu chấm phẩy ở cuối các dòng nguyên mẫu.
- Dư dấu chấm phẩy ở cuối dòng tiêu đề của mỗi hàm.
- Không ghi kiểu dữ liệu của tham số trong nguyên mẫu cũng như tiêu đề hàm.

3.8. BÀI TẬP

3.8.1. *Viết dòng mô tả của các hàm theo mô tả như sau*

3.8.1.1. Hàm sẽ nhận một giá trị kiểu double truyền cho nó khi được gọi thực hiện. Hàm trả về trị tuyệt đối của giá trị được truyền.

3.8.1.2. Hàm sẽ nhận 2 tham số kiểu float. Hàm trả về tích của 2 tham số.

3.8.1.3. Hàm nhận hai tham số kiểu số nguyên a và b. Sau khi thực hiện xong, hàm trả về giá trị của a lũy thừa b.

3.8.1.4. Hàm nhận một tham số kiểu số thực. Hàm trả về giá trị bình phương của tham số đầu vào.

3.8.1.5. Hàm nhận một tham số kiểu số nguyên n. In ra bảng cửu chương của tham số nhận vào.

Từ đây trở đi, sinh viên phải sử dụng kỹ thuật lập trình hàm để viết chương trình

3.8.2. *Kiểm tra số do người dùng nhập vào*

3.8.2.1. Cho người dùng nhập vào 1 số nguyên dương ***n*** thỏa lần lượt các yêu cầu sau đây. Nếu người dùng nhập vào số không đúng yêu cầu thì chương trình cho nhập lại

a. Điều kiện: ***n*>0**. Nếu nhập đúng, chương trình in ra các số chẵn nhỏ hơn ***n*.**

b. Điều kiện: **0 <= *n* <= 9**. Nếu nhập đúng, chương trình in ra cách đọc của số vừa nhập.

c. Điều kiện: ***n* <10 hoặc *n* > 50**. Nếu nhập đúng, chương trình in ra ***n*** số ngẫu nhiên.

d. Điều kiện: ***n* bội số của 5**. Nếu nhập đúng, chương trình in ra bảng cửu chương 5.

e. Điều kiện: ***n* nằm trong các số (2, 4, 6, 8, 10)**. Nếu nhập đúng, chương trình cho nhập tiếp 2 số nguyên i, j. Hãy tính và in ra cấp số cộng với số hạng đầu là i, công sai là j và số phần tử của dãy là ***n*** .

3.8.2.2. Viết chương trình cho nhập 2 số nguyên dương n và m sao cho số nhập sau (m) phải luôn lớn hơn số nhập trước (n).

3.8.2.3. Cho nhập 2 số a, b sao cho: số lớn nhất trong 2 số phải là một số dương và chia hết cho 7. Nếu nhập sai phải yêu cầu nhập lại cho đến khi đúng.

3.8.2.4. Cho nhập 2 số nguyên dương a và b sao cho số nhập sau (b) thỏa 2 điều kiện: là ước số của a và b bội số của 5. Nếu nhập sai, cho nhập lại cả 2 số a và b.

3.8.2.5. Nhập vào số nguyên dương là lũy thừa của 2^k hoặc 3^k.

3.8.3. *Tách 1 số nguyên gồm nhiều chữ số*

3.8.3.1. Viết chương trình nhập vào một số nguyên (n) 3 chữ số (từ 100-999), sau đó in ra các chữ số thuộc hàng trăm, hàng chục, hàng đơn vị.

```
Ví dụ: n nhập vào là 128. Chương trình in ra:
                Số hàng trăm:  1
                Số hàng chục: 2
                Số hàng đơn vị: 8
```

3.8.3.2. In ra cách đọc của một số dương

a. Nhập số nguyên dương n gồm 3 chữ số (n > 0), in ra cách đọc của số n.
Ví dụ: Nhập n = 105. In ra màn hình: Mot khong nam.

b. Tương tự như bài tập trên, nhưng cho phép giá trị của n nằm trong khoảng (0 <= n <= 4 tỷ). In ra cách đọc của n.

3.8.3.3. Cho nhập số nguyên dương n. Đếm số lượng chữ số chẵn, số chữ số lẻ có trong n.

3.8.3.4. Tìm chữ số lớn nhất của số nguyên dương n.

3.8.3.5. Đếm số lượng chữ số lớn nhất của số nguyên dương n.

3.8.3.6. Cho nhập số nguyên dương n. Liệt kê các số <n có dạng 2^k.

3.8.3.7. Viết chương trình nhập số nguyên dương n gồm k chữ số (2<k<=5), kiểm tra xem các chữ số của n có phải là số đối xứng hay không?
Ví dụ: Đối xứng: 12321. Không đối xứng: 12345

3.8.3.8. Viết chương trình nhập vào số nguyên n gồm ba chữ số. Xuất ra màn hình theo thứ tự đảo ngược của các chữ số.
Ví dụ: n=291. Xuất ra 192.

3.8.3.9. Hãy kiểm tra các chữ của số nguyên dương n có tăng dần từ trái sang phải hay không?

3.8.3.10. Cho ba số a, b, c được nhập vào từ bàn phím. Hãy in ra vị trí chứa số lớn nhất. Ví dụ : nhập 579, in ra : *"số lớn nhất ở hàng đơn vị"*

3.8.3.11. **(*)** Viết chương trình nhập số nguyên dương n gồm k chữ số $(0 < k \le 5)$, sắp xếp các chữ số của n theo thứ tự tăng dần.
Ví dụ: Nhập n=1536
Kết quả sau khi sắp xếp: 1356.

3.8.4. *Ước/bội số chung*

3.8.4.1. Cho nhập số nguyên dương n, liệt kê tất cả các ước số của n.

3.8.4.2. Cho nhập số nguyên dương n, tìm ước số lẻ lớn nhất của n (nhỏ hơn n).
Ví dụ: Ước số lẻ lớn nhất của 27 là 9.

3.8.4.3. Cho nhập 2 số nguyên dương n và m, liệt kê các ước số của n có giá trị nhỏ hơn m.
Ví dụ: Nhập n=16, m=7;
in ra các ước số của 16 nhỏ hơn 7 là 1, 2, 4.

3.8.4.4. Nhập vào số nguyên dương là lũy thừa của 2, nếu nhập sai cho nhập lại. In ra cách biểu diễn của n dưới dạng số mũ của 2.
Ví dụ : nhập n = 8 in ra ước số 8 = 2^3

3.8.4.5. Cho nhập số nguyên dương n (n>=2). Hãy phân tích n thành tích các thừa số nguyên tố.
Ví dụ : nhập n = 1350. Phân tích 1350 = 2*3*3*3*5*5

3.8.4.6. Cho nhập số nguyên dương n. In ra màn hình cách phân tích n thành thừa số nguyên tố.
Ví dụ : nhập n = 5000 in ra 5000 = 2^3 X 5^4.

3.8.4.7. Cho nhập 2 số nguyên dương a, b. Tìm USCLN của a và b.

3.8.4.8. Viết chương trình cho người dùng nhập vào tử và mẫu số, thực hiện đơn giản phân số.

3.8.4.9. Cho nhập 2 số nguyên dương n và m, liệt kê các bội số của n có giá trị nhỏ hơn m.

```
Ví dụ: Nhập n=6, m=27;
    in ra  các bội số của 6 nhỏ hơn 27 là 12, 24
```

3.8.4.10. Tìm ước chung lớn nhất và bội chung nhỏ nhất của 2 số nguyên dương n và m nhập từ bàn phím. (Sử dụng thuật toán Euclide : USCLN(A,B) = USCLN(B, A mod B) với A>B)

3.8.4.11. Có 3 tuyến xe bus cùng khởi hành tại bến vào lúc 5h và chạy theo những tuyến đường (hướng đi) khác nhau. Mỗi lượt chạy, xe thứ 1 chạy và trở lại bến sau 1h05', nghỉ 10 phút rồi tiếp tục chạy lượt kế tiếp; xe thứ 2 chạy và trở lại bến sau 55', nghỉ 5 phút rồi tiếp tục chạy lượt kế tiếp; xe thứ 3 chạy và trở lại bến sau 48', nghỉ 2 phút rồi tiếp tục chạy lượt kế tiếp. Hỏi sau bao lâu nữa 3 xe sẽ lại cùng xuất phát.

Tổng quát, cho người dùng nhập thời gian chạy 1 lượt, thời gian nghỉ của mỗi loại xe. Cho biết sau bao lâu 3 xe sẽ lại cùng xuất phát.

3.8.5. *Tổng/tích của 1 dãy số*

Viết chương trình cho người dùng nhập số nguyên dương n (và k nếu có), rồi in ra trị của S (hoặc P). Biết rằng:

3.8.5.1. $S= \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n+1}$

3.8.5.2. $S= \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \ldots + \frac{1}{2n}$

3.8.5.3. $S= 1 + \frac{1}{3} + \frac{1}{5} + \ldots + \frac{1}{2n-1}$

3.8.5.4. $S= \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \ldots + \frac{n}{n+1}$

3.8.5.5. $S= \frac{1}{2} + \frac{3}{4} + \frac{5}{6} + \ldots + \frac{2n+1}{2n+2}$

3.8.5.6. $P = 1 \times 3 \times 5 \times \ldots \times (2n-1)$

3.8.5.7. $S = (1) + (1+2)+(1+2+3)+(1+2+3+4)+ +(1+2+3+4+5)+ \ldots +(1+2+3+... +n)$

3.8.5.8. $S = (1) + (1x2) +(1x2x3)+ \ldots +(1x2x3x\ldots xn)$

3.8.5.9. $S= 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \ldots + \frac{1}{1+2+3+\ldots+n}$

3.8.5.10. $S = 1 - \frac{1}{1+2} + \frac{1}{1+2+3} - \ldots + (-1)^{n+1} \frac{1}{1+2+3+...+n}$

3.8.5.11. $S= 1 + \frac{1+2}{2} + \frac{1+2+3}{3} + \ldots + \frac{1+2+3+\ldots+n}{n}$

3.8.5.12. $S = \frac{1^2 + 2^2 + 3^2 + \ldots + n^2}{1*2 + 2*3 + 3*4 + \ldots + n(n+1)}$

3.8.5.13. $S = x^2 + x^4 + \ldots. + x^{2n}$

3.8.5.14. $S= x + \frac{x^2}{1+2} + \frac{x^3}{1+2+3} + \ldots + \frac{x^n}{1+2+3+\ldots+n}$

3.8.5.15. $S = 1 - 2 + 3 - 4 + (-1)^{n+1}n$

3.8.5.16. $S = x - x^2 + x^3 - \ldots. + (-1)^{n+1}x^n$

3.8.5.17. $S = -x^2 + x^4 - \ldots. + (-1)^n x^{2n}$

3.8.5.18. $S = x - x^3 + x^5 + \ldots + (-1)^n x 2^{n+1}$

3.8.5.19. $S = -x + \frac{x^2}{1+2} - \frac{x^3}{1+2+3} + \ldots. + (-1)^n \frac{x^n}{1+2+3+\ldots+n}$

3.8.5.20. Lập chương trình nhập và tính một đa thức dạng:

$$P(x) = a_0x^n + a_1x^{n-1} + \ldots\ldots + a_{n-1}x + a_n.$$

3.8.5.21. Lập chương trình cho nhập x và y. Tính một đa thức dạng:

$$P = ((-x/2+y^3/27+x^2/4)^{1/2})^{1/3} + (-x/2-(y^3/27+x^2/4)^{1/2})^{1/3}$$

3.8.5.22. $S = \sqrt{2+\sqrt{2+\sqrt{2+\ldots\ldots\sqrt{2+\sqrt{2}}}}}$ có n dấu căn.

3.8.5.23. $S = \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{\ldots 1}}}}$

3.8.6. *Giai thừa*

3.8.6.1. Cho nhập một số nguyên dương (n). In ra giai thừa của n. Với giai thừa của số n được định nghĩa như sau : 0!=1

n!=1*2*3*...*n

3.8.6.2. Cho nhập số nguyên dương n, rồi in ra trị của S (hoặc e^x). Biết rằng:

a. $S= 1 + \frac{1+2}{2!} + \frac{1+2+3}{3!} + \ldots + \frac{1+2+3+\ldots+n}{n!}$

b. $S= x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$

c. $S= 1 + \frac{x^2}{} + \frac{x^4}{} + \ldots + \frac{x^{2n}}{}$

$$2! \qquad 4! \qquad (2n)!$$

d. $S = 1 + x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n+1}}{(2n+1)!}$

e. $S = -x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^n}{n!}$

f. $S = -1 + \frac{x^2}{2!} - \frac{x^4}{4!} + \ldots + (-1)^{n+1} \frac{x^{2n}}{(2n)!}$

g. $S = 1 - x + \frac{x^3}{3!} - \frac{x^5}{5!} + \ldots + (-1)^{n+1} \frac{x^{2n+1}}{(2n+1)!}$

3.8.7. *Số nguyên tố*

3.8.7.1. Cho nhập số nguyên dương n. Kiểm tra xem n có phải là số nguyên tố hay không?

3.8.7.2. Cho nhập số nguyên dương n. Liệt kê các số nguyên tố < n.

3.8.7.3. Cho nhập số nguyên dương n Tìm số nguyên tố đầu tiên <n và có giá trị gần với n nhất.

3.8.7.4. Cho nhập số nguyên dương n. Tìm số nguyên tố đầu tiên >n và có giá trị gần với n nhất.

3.8.7.5. Cho nhập số nguyên dương n, liệt kê các ước số của n là số nguyên tố.

```
Ví dụ: Nhập n=36. Các ước số của 36 gồm 1, 2, 3, 4, 6, 9,
    12, 18
Nhưng chỉ in ra:  các số vừa là ước số của 36, vừa là số
    nguyên tố:    2∪ 3
```

3.8.7.6. Cho nhập vào hai số n và m, in ra n số nguyên tố mà giá trị của chúng phải lớn hơn m.

3.8.7.7. Viết chương trình nhập số nguyên dương n gồm k chữ số $(0 < k \le 5)$, đếm xem n có bao nhiêu chữ số là số nguyên tố.

3.8.7.8. Viết chương trình chứng minh bài toán Gobach (một số nguyên tố bất kỳ lớn hơn 5 đều có thể khai triển thành tổng của 3 số nguyên tố khác. Minh họa cho những số nguyên tố<100.

3.8.8. *CÁC BÀI TOÁN VỀ SỐ KHÁC*

📖 *Số hoàn thiện*: là số (***n***) có tổng các ước số không kể n bằng chính n.

Ví dụ: 6 là số hoàn thiện vì 1+2+3=6.

📖 *Số chính phương*: là số có căn bậc hai là một số nguyên.

Ví dụ 9 là số chính phương vì căn bậc hai của 9 là 3.

📖 *Số Armstrong*: là số có đặc điểm sau: số đó gồm n ký số, tổng các lũy thừa bậc n của các ký số bằng chính số đó.

Ví dụ 153 là một số có 3 ký số, và $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

3.8.8.1. Cho số nguyên dương x. Kiểm tra xem x có phải là số hoàn thiện hay không?

3.8.8.2. Tìm các số hoàn thiện nhỏ hơn 5000.

3.8.8.3. Cho số nguyên dương x. In ra các số hoàn thiện nhỏ hơn x?

3.8.8.4. Cho số nguyên dương x. In ra tổng các số hoàn thiện nhỏ hơn x?

3.8.8.5. Cho số nguyên dương x. Kiểm tra xem x có phải là số chính phương hay không?

3.8.8.6. Nhập số nguyên dương n (n>0). Liệt kê n số chính phương đầu tiên.

3.8.8.7. Liệt kê tất cả các hoán vị của một tập gồm n phần tử.

3.8.8.8. Viết chương trình nhập số nguyên dương N (N <= 2 tỷ), kiểm tra xem N có phải là số đối xứng hay không. (Số đối xứng là số có giá trị không đổi nếu đọc các chữ số từ phải qua trái, ví dụ: 34543).

3.8.8.9. Viết chương trình nhập vào một số nguyên dương không dấu, rồi in ra số tương ứng thuộc các cơ số Bibanry, Octal, Hexa.

3.8.8.10. Viết chương trình nhập một số thập phân n. Tạo menu lựa chọn các công việc in giá trị tương ứng với n theo các cơ số: 2, 8, 16.

3.8.8.11. Viết chương trình nhập vào một số nguyên thuộc hệ m rồi in giá trị tương ứng ở hệ n.Trong đó m và n là hai giá trị nhập từ bàn phím (n, m là một trong các giá trị 2, 8, 10, 16).

3.8.8.12. Viết chương trình nhập vào 1 số nguyên dương n (hệ thập phân). In ra màn hình giá trị nhị phân tương ứng của số vừa nhập.

3.8.8.13. Tìm các số Armstrong nhỏ hơn 1000.

3.8.8.14. Dãy Fibonaci {Fn} bậc 2 được định nghĩa:

$f_0 = f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$ (n>=2)

Viết chương trình nhập số nguyên dương n và in ra dãy Fibonaci n phần tử. Nhập số nguyên k < n, tính phần tử F_k

3.8.8.15. Tìm những giá trị nguyên x, y, z thỏa mãn công thức Pithagore $x^2+y^2=z^2$. Với 0<x, y, z <1000

3.8.8.16. Viết chương trình minh họa bài toán An Casi. Biết bài toán An Casi được mô tả như sau: với mọi số n nguyên dương thì ta luôn có đẳng thức sau:

$1 + 2^4 + \ldots + n^4 = 6n^5 + 15n^4 + 10n^3 - n$ (đẳng thức Ansi)

3.8.8.17. Tính tổ hợp chập k của n.

$$C_n^k = \frac{n!}{k!(n-k)!}$$

3.8.8.18. Viết chương trình in tam giác Pascal.

C_0^0	1
$C_1^0\ C_1^1$	1 1
$C_2^0\ C_2^1\ C_2^2$	1 2 1
$C_3^0\ C_3^1\ C_3^2\ C_3^3$	1 3 3 1
.....	.....
$C_n^0\ C_n^1\ C_n^2 \ldots C_n^n$	1 9 36 84 126 126 84 36 9 1

Với $C_n^k = \frac{n!}{k!(n-k)!}$

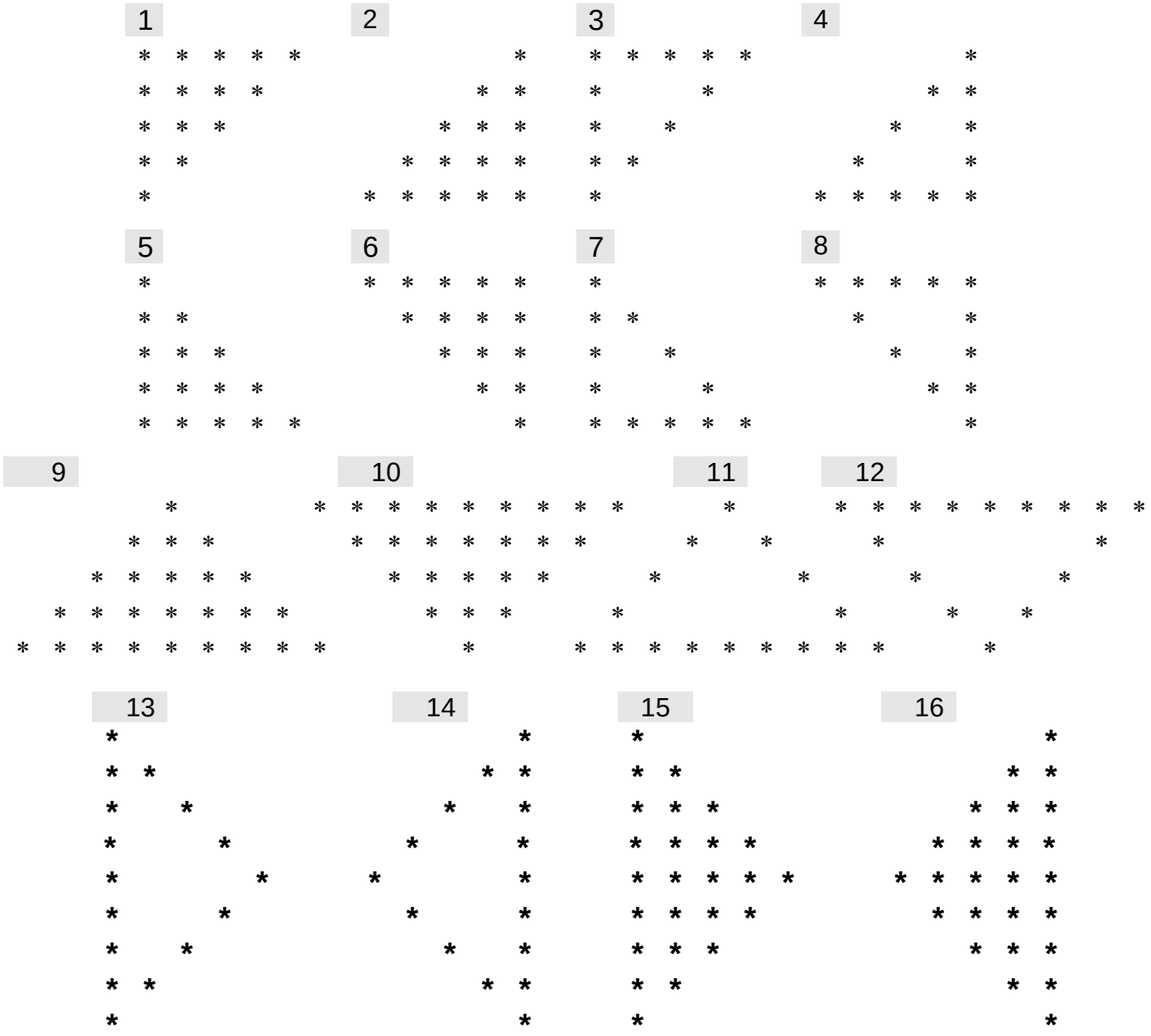
3.8.8.19. Viết chương trình hiển thị tháp PASCAL

```
                1
               121
              12321
             1234321
            123454321
           12345654321
          1234567654321
         123456787654321
        12345678987654321
```

° Tương tự, nhưng viết lại chương trình hiển thị tháp đảo ngược.

3.8.9. *Vẽ hình*

3.8.9.1. Viết chương trình cho người dùng nhập cạnh của tam giác vuông. Lần lượt in ra các hình sau:



17
```
1
22
333
4444
55555
```

18
```
55555
 4444
  333
   22
    1
```

19
```
55555
4444
333
22
1
```

20
```
    1
   22
  333
 4444
55555
```

21
```
1  1
22 12
333 123
4444 1234
55555 12345
```

22
```
12345
1234
123
12
1
```

23
```
12345
1234
123
12
1
```

24
```
    1
   12
  123
 1234
12345
```

25
```
    X
   X X
  X X X
 X X X X
X X X X X
```

26
```
X X X X X
 X X X X
  X X X
   X X
    X
```

27
```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

28
```
5 5 5 5 5
 4 4 4 4
  3 3 3
   2 2
    1
```

29
```
    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5
```

30
```
1 2 3 4 5
 1 2 3 4
  1 2 3
   1 2
    1
```

31
```
    X
   X X
  X X X
 X X X X
X X X X X
X X X X X
 X X X X
  X X X
   X X
    X
```

32
```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
5 5 5 5 5
 4 4 4 4
  3 3 3
   2 2
    1
```

33
```
    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5
1 2 3 4 5
 1 2 3 4
  1 2 3
   1 2
    1
```

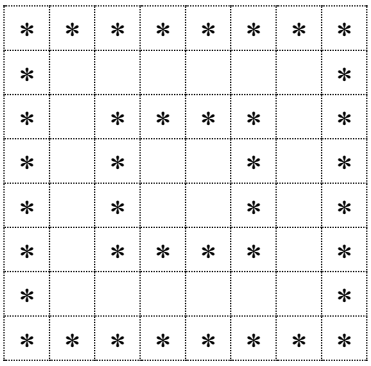
34
```
1
2 2
3   3
4     4
5       5
4     4
3   3
2 2
1
```

35
```
      1
    2 2
  3   3
4       4
5       5
4       4
  3   3
    2 2
      1
```

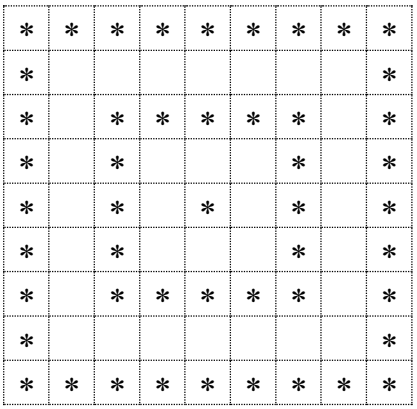
36
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

3.8.9.2. Viết chương trình cho người dùng nhập Cho nhập cạnh (n). Vẽ hình vuông có dạng

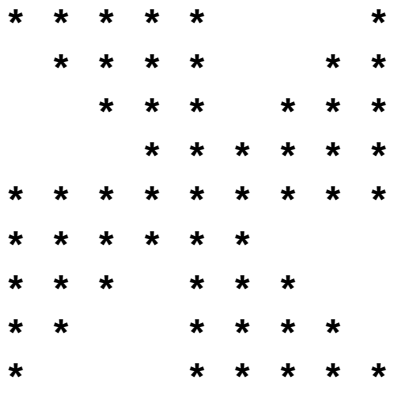
n=8



n=9



Hình cánh quạt



3.8.9.3. *Hình cánh quạt*: Sử dụng các hàm `cprintf()`, `textcolor()`, `delay()`, `kbhit()`, ... thay đổi màu để tạo cảm giác cho cánh quạt xoay cho đến khi nhấn một phím bất kỳ.

3.8.10. *Hình học giải tích*

3.8.10.1. Viết chương trình nhập tọa độ của 2 điểm A(x_A,y_A), B(x_B,y_B). Tính chiều dài đoạn AB.

3.8.10.2. Viết chương trình nhập tọa độ trực chuẩn của 3 điểm A(x_A,y_A), B(x_B,y_B), C(x_C,y_C) sau đó thực hiện kiểm tra: nếu ABC là tam giác thì in ra tọa độ của điểm D(x_D,y_D) tạo thành hình bình hành ABCD.

3.8.10.3. Viết chương trình nhập tọa độ trực chuẩn của 4 điểm A(xa,ya), B(xb,yb), C(xc,yc), M(xm,ym) sau đó thực hiện kiểm tra và in ra kết quả vị trí tương đối của M với ABC: nếu ABC thẳng hàng thì M nằm trên hay nằm ngoài ABC? Hoặc nếu ABC là tam giác thì M nằm trong, nằm trên cạnh hoặc nằm ngoài ABC? Biết Phương trình đường thẳng qua 2 điểm A và B là :

$$(x-x_A) / (x_A-x_B) = (y-y_A) / (y_A-y_B)$$

3.8.10.4. Viết chương trình nhập tọa độ trực chuẩn của 3 điểm A(xa,ya), B(xb,yb), C(xc,yc) sau đó thực hiện kiểm tra 3 điểm ABC có tạo thành tam giác không? Nếu 3 điểm ABC tạo thành tam giác thì in ra trị số của chu vi, diện tích và tọa độ trọng tâm của tam giác này.

3.8.10.5. Viết chương trình nhập tọa độ trực chuẩn của 3 điểm $A(x_A,y_A)$, $B(x_B,y_B)$, $C(x_C,y_C)$ sau đó thực hiện kiểm tra và in ra kết qủa ABC là hình gì? (đường thẳng, tam giác vuông, tam giác cân, tam giác vuông cân, tam giác đều, tam giác thường)

3.8.10.6. Viết chương trình nhập tọa độ trực chuẩn của 2 điểm khác nhau $A(x_A,y_A)$, $B(x_B,y_B)$, sau đó in ra tọa độ 2 điểm $C(x_C,y_C)$, $D(x_D,y_D)$ sao cho ABCD là một hình vuông.

3.8.10.7. Viết chương trình nhập tọa độ trực chuẩn của 2 điểm khác nhau $A(x_A,y_A)$, $B(x_B,y_B)$. Sau đó in ra vị trí tương đối của A và B so với đường tròn tâm O(0,0) bán kính OM (M là trung điểm của AB và M <>0).

3.8.10.8. Viết chương trình nhập tọa độ trực chuẩn của điểm $A(x_A,y_A)$ và một số dương R sau đó in ra tọa độ giao điểm cuả đường tròn tâm A bán kính R với 2 trục tọa độ.

3.8.10.9. Cho biết trong hệ tọa độ trực chuẩn xOy:
- Phương trình đường tròn tâm O(0,0) bán kính R là: $x^2 + y^2 = R^2$.
- Phương trình đường trẳng D // Oy đi qua điểm M(k,0) là: x=k.

3.8.10.10. Viết chương trình nhập vào bán kính R của đường tròn O và hoành độ $k \in [-R,+R]$ của điểm M sau đó in ra:
- Tọa độ giao điểm của đường tròn O với đường thẳng D.
- Diện tích tam giác tạo bởi gốc tọa độ O với 2 giao điểm trên.

3.8.10.11. Cho biết trong hệ tọa độ trực chuẩn xOy phương trình ellipse có tâm O và trục $AB = 2a \in xOx'$, trục $CD = 2b \in yOy'$ là: $x^2/a^2 + y^2/b^2 = 1$ với a, b, x, y là các số thực và a>0, b>0. Viết chương trình nhập vào các hệ số a, b của một ellipse dạng nêu trên sau đó in ra tọa độ giao điểm của ellipse với đường phân giác y=x.

3.8.10.12. Cho biết trong hệ tọa độ trực chuẩn xOy phương trình Parapol có trục song song với Oy là: $y = ax^2 + bx + c$, với a, b, c, x, y là các số thực và a<>0; Viết chương trình nhập các hệ số a, b, c của một Parapol có dạng trên sau đó in ra tọa độ: đỉnh của Parapol và giao điểm của Parapol với trục hoành nếu có.

3.8.11. *Đọc chương trình, xác định kết quả*

3.8.11.1. Đọc các chương trình dưới đây, cho biết kết quả xuất ra của chương trình

3.8.11.1.1.

```
#include <stdio.h>
void main()
{    int  i=1, n=5, sum=0;
     while( i<=n)
     {    sum+=i*i;
          i++;
     }
     printf("i=%d, sum=%d",i,sum);
}
```

KQ: i=6, sum=55

3.8.11.1.2.

```
#include <stdio.h>
void main()
{    int f=1, n=5;
     do
     {    f*=n;
          n--;
     }while(n>1);
     printf("n=%d, f=%d",n,f);
}
```

KQ: n=1, f=120

3.8.11.1.3.

```
#include <stdio.h>
void main()
{    int i=0,n=5;
     while(i<n)
     {    if(i<3)
          {    i+=2;
               n+= i;
          }
          else
               i+=3;
     }
     printf("i=%d, n=%d",i,n);
}
```

KQ: i=13, n=11

3.8.11.1.4.

```
#include <stdio.h>
void main()
{    int a=1,b=3,c=0;
     while(a>0&&(b>0||c>0))
     {    a+=1;
          b-=2;
          c=a+b;
     }
     printf("a=%d, b=%d, c=%d",a,b,c);
}
```

KQ: a=5, b=-5, c=0

```
#include <stdio.h>
void main()
{    int i, a=0, b=0, c=0;
     for (i=0 ; i<=5 ; i++)
     {    if(i%2==0)
               a=i;
          else if (i%3= =0)
                    b=i;
                else
                    c=a+b;
     }
     printf("a=%d, b=%d, c=%d",a,b,c);
}
```

KQ: a=4,b=3,c=7

3.8.11.1.6.

```
#include <stdio.h>
void main()
{    int a, b, c;
     a=b=c=7;
     do
     {      a++;
          b--;
            c=2*b-a;
     }while(c>=0);
     printf("a=%d, b=%d, c=%d",a,b,c);
}
```

KQ: a=10, b=4, c=-2

3.8.11.1.7.

```
#include <stdio.h>
void main()
{    int i, a, b, c;
     a=b=c=1;
     for(i=8;i>b;i--)
     {    a=i;
          b=c+1;
          if(b%2==0)    c=2*b+1;
     }
     printf("a=%d, b=%d, c=%d",a,b,c);
}
```

KQ: a=7,b=6,c=13

3.8.11.1.8.

```
#include <stdio.h>
void main()
{    int a=10, b=a+3;
     for(i=5 ; i>b ; i++)
     {    if(a<b)
               i = (++a)/2;
          else
               i = b--;
     }
     printf("i=%d,a=%d, b=%d",i,a,b);
}
```

KQ: i=14, a=13, b=12

```
#include <stdio.h>
void main()
{    int a=10, b=a*3, i=3;
     do
     {    if(a<b)
               a = a+i;
          else
               b = a-b;
          i*=2;
     }while(2*i<a);
     printf("i=%d,a=%d, b=%d",i,a,b);
}
```

KQ: i=12, a=19, b=30

3.8.11.1.10.

```
#include <stdio.h>
void main()
{
     int a=10, b=a*2, c=0;
     while(b-a>0)
     {
          if(b<23)   c=b*2+1;
          else       b=c-a-2;
          a = c - (++b);
     }
     printf("a=%d, b=%d, c=%d",a,b,c);
}
```

KQ: a=23, b=22, c=45

3.8.11.2. Cho biết kết quả của hàm main và giá trị của các tham số sau khi thực hiện xong các hàm khác

3.8.11.2.1.

```
#include <stdio.h>
void f1(int m, int &n);
void main()
{
     int a=10, b=a*3;
     do
     {
          if(b<a*3)  a=b*2;
          else  b=a-b;
     }while(a<b);
     printf("\n1.a=%d, b=%d",a,b);
     f1(a,b);
     printf("\n2.a=%d, b=%d",a,b);
}
void f1(int m, int &n)
{
     if(m%3 ==0 && n%3==0)   m=n*2;
     else                    n=m*2;
}
```

KQ: 1.a=10, b=-20
2.a=10, b=20

```
#include <stdio.h>
void f2(int m, int &n) ;
void main()
{    int a=10, b=a%4;
     do
     {    if(b*4<a)  b=a*2;
          else   a=b+2;
     }while(a<b);
     printf("\n1.a=%d, b=%d",a,b);
     f2(a,b);
     printf("\n2.a=%d, b=%d",a,b);
}
void f2(int m, int &n)
{    if ((m == 20) || (n == 20))    n=m/2;
     else                           m=n*2;
}
```

KQ: 1.a=22, b=20
2.a=22, b=11

3.8.11.2.3.

```
#include <stdio.h>
void  f3(int &m, int n);
void main()
{    int i, a=10, b=2*(a++);
     for(i=0; i<a; i+=3)
          a-= i ;
     printf("\n1.a=%d, b=%d",a,b);
     f3(b, a);
     printf("\n2.a=%d, b=%d",a,b);
}
void f3(int &m, int n)
{
     if(m%2!=0 && n%2!=0)  n=m*2;
     else                  m=n*2;
}
```

KQ: 1.a=2, b=20
2.a=2,b=4

3.8.11.2.4.

```
#include <stdio.h>
void  f4(int m, int  &n);
void main()
{    int i, a=10, b=a/3;
     for(i=13; i>a; i--)
          if(b<a)
               b=i*2;
          else
               a=b-i;
     printf("\n1.a=%d, b=%d",a,b);
     f4(a, b);
     printf("\n2.a=%d, b=%d",a,b);
}
void f4(int m, int  &n)
{    if(m*n<100)
          m=n*3;
     n=m+40;
}
```

KQ: 1.a=14, b=26
2.a=14, b=54

```
#include <stdio.h>
void  f5(int &m, int n);
void main()
{    int i, a=10, b=a*2;
     for(i=5;a<b; i++)   a+=i;
     printf("\n1.a=%d, b=%d",a,b);
     f5(b, a);
     printf("\n2.a=%d, b=%d",a,b);
}
void f5(int &m, int n)
{    if(m*n>400)
          m=n*2;
     n=m+60;
}
```

KQ: 1.a=21, b=20
2.a=21, b=42

3.8.11.2.6.

```
#include <stdio.h>
void  f6(int &m, int n);
void main()
{    int  a=10, b=a/4;
     if(b+5>a)     a=b*2;
     else          b=b+a;
     printf("\n1.a=%d, b=%d",a,b);
     f6(a, b);
     printf("\n3.a=%d, b=%d",a,b);
}
void f6(int &m, int n)
{    while (2*n-m>0)
          n-=5;
     m=n+m;
     printf("\n2.m=%d, n=%d",m,n);
}
```

KQ: 1.a=10, b=12
2.m=12, n=2
3.a=12, b=12

3.8.11.2.7.

```
#include <stdio.h>
void  f7(int m, int  &n);
void main()
{    int  a=10, b=a*3;
     do
     {    if(b<3*a)       a=b*2;
          else            b=b-a;
     }while(a<b);
     printf("\n1.a=%d, b=%d",a,b);
     f7(a, b);
     printf("\n3.a=%d, b=%d",a,b);
}
void f7(int m, int  &n)
{    if ((m%3 == 0) &&  (n%3 ==0))    m=n*2;
     else                             n=m*2;
     printf("\n2.m=%d, n=%d",m,n);
}
```

KQ: 1.a=40, b=20
2.m=40, n=80
3.a=40, b=80

```
#include <stdio.h>
int f8(int m, int &n);
void main()
{    int i, a=10, b=a/2, c=a+b;
     for( i=10; a<b+16;i--)
          a+=i;
     printf("\n1.a=%d, c=%d",a,c);
     f8(a,c);
     printf("\n3.a=%d, c=%d",a,c);
}
int f8(int m, int &n)
{    if((m+n)%3 == 0)  m=n+2;
     else       n=m+2;
     printf("\n2.m=%d, n=%d",m,n);
     return m+n;
}
```

KQ: 1.a=29, c=15
2.m=29, n=31
3.a=29, c=31

3.8.11.2.9.

```
#include<stdio.h>
int f9(int &m, int n);
void main()
{    int i, a=10, b=a*2, c=a+b;
     for(i=5; a<b;i++)   a += i;
     printf("\n1.a=%d, c=%d",a,c);
     c=f9(a, b);
     printf("\n3.a=%d, c=%d",a,c);
}
int f9 (int  &m, int n)
{    if(m*n<=420)    m=n*2;
     else            n=m+60;
     printf("\n2.m=%d, n=%d",m,n);
     return m+n;
}
```

KQ: 1.a=21, c=30
2.m=40, n=20
3.a=40, c=60

3.8.11.2.10.

```
#include<stdio.h>
int f10(int &m, int  &n);
void main()
{    int a, b, c;
     for(a=5, b=10; a+b<20; a++, b++)
          c=a+b;
     printf("\n1.a=%d, c=%d",a,c);
     c=f10(a, b);
     printf("\n3.a=%d, c=%d",a,c);
}
int f10 (int  &m, int  &n)
{    if(m%2 == 0)    n=n*m;
     if(n%2 == 0)    m=n+m;
     printf("\n2.m=%d, n=%d",m,n);
     return m+n;
}
```

KQ: 1.a=8, c=19
2.m=112, n=104
3.a=112, c=216

Chương trình sau được viết trong ngôn ngữ C, giả sử bộ nhớ trong của máy tính đủ để thực thi chương trình. Hãy cho biết kết quả khi cho thực thi chương trình này.

```
#include <iostream.h>
#include <conio.h>
unsigned * pA, uN;
unsigned FunA(unsigned *, unsigned);

main(){
    clrscr();
    uN=10;
    pA=new unsigned[uN];
    cout<<": "<< FunA(pA,uN)<<endl;
    return(0);
}
unsigned FunA(unsigned * AI, unsigned N){
    if(!N) return (0);
    unsigned Temp=0;
    for(unsigned k=0; k<N; k++){
        if(k<3)
            AI[k]=k;
        else
            AI[k]=AI[k-2]+AI[k-1];
        Temp+=AI[k];
        cout<<AI[k]<<char(32);
    }
    return Temp;
}
```

KQ: 0 1 2 3 5 8 13 21 34 55: 142

3.8.11.3. Tìm lỗi trong chương trình, sửa lại cho đúng

3.8.11.3.1.

```
void A()
{   printf("Ben trong ham A");
    void B()
    {
          printf("Ben trong ham B");
    }
}
```

3.8.11.3.2.

```
int Sum(int x, int y)
{   int S;
    S = x + y;
}
```

3.8.11.3.3.

```
void A(int s);
{     printf("Gia tri cua S la: %d",s);
}
```

```
int Sum(int x, int y)
{
    int x, y;
      int S;
      S = x + y;
      return S;
}
```

3.8.11.3.5.

```
void sum(int a, int b)
{
    int S;
    S = a + b;
    return S;
}
```

3.8.11.3.6.

```
int TrungBinh(int n)
{
    float TB = 0;
    for(int i=1; i<n; i++)
        TB += i;
    return (float) TB/n;
}
```

3.8.11.3.7.

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

3.8.11.3.8.

```
int TinhTong(int a, int b)
{
    Tong = a + b;
    return Tong;
}
```

3.8.11.3.9.

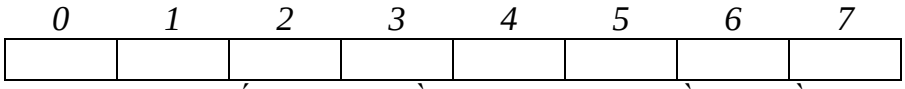
```
void ChanLe(int a)
{
    if( a%2 == 0);
        printf("a la so chan");
    else
        printf("a la so le");
}
```

MẢNG MỘT CHIỀU (One Dimensional Array)

4.1. GIỚI THIỆU

4.1.1. *Khái niệm*

- Mảng thực chất là một biến được cấp phát bộ nhớ liên tục và bao gồm nhiều biến thành phần.
- Chỉ số mảng (chỉ mục): là số thứ tự của các phần tử trong mảng (tính từ 0).



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

4.1.2. *Khai báo mảng*

```
<Kiểu dữ liệu> <Tên mảng> [<Số phần tử tối đa của mảng>];
```

- Ví dụ:

```
int A[100];//Khai bao mang so nguyen a gom 100 phan tu
float b[50];//Khai bao mang so thuc b gom 50 phan tu
```

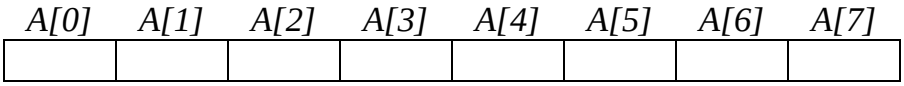
- Thông thường trong lập trình người ta thường định nghĩa trước một hằng chứa số phần tử của mảng. Điều này rất có lợi khi ta cần thay đổi số phần tử của mảng. Như vậy ta có thể khai báo một mảng như sau:

```
const int SIZE=500;
int diem[SIZE];
```

Hay

```
const int SIZE=8;
char A[SIZE];
```

- Khi khai báo như trên thì mỗi mảng sẽ được cung cấp đủ số ô nhớ cần thiết để lưu trữ các phần tử của mảng. Và các phần tử của một mảng được lưu trữ liên tục với nhau.



Hình ảnh của mảng A được lưu trong bộ nhớ

4.1.3. *Truy xuất các phần tử của mảng*

- Các thành phần của mảng là tập hợp các biến có cùng kiểu dữ liệu và cùng tên. Do đó để truy xuất các biến thành phần, ta dùng cơ chế chỉ mục.
- Cú pháp:

```
array_name[offset]
```

- Ví dụ: như A[2] (truy xuất đến phần từ thứ hai của mảng a và là phần tử thứ 3 tính từ đầu mảng).
- Mỗi một phần tử của mảng có thể sử dụng bất cứ ở vị trí nào mà một biến vô hướng hợp lệ xuất hiện. Ví dụ các vị trí xuất hiện của mảng A sau đây là hợp lệ:

```
A[0]= 'A';
A[i]=A[0]+32
```

- Chỉ số của một phần tử đặt trong [] là một hằng số nguyên hoặc bất kỳ một biểu thức nào mà giá trị của nó là một số nguyên.

```
Ví dụ:     A[i] , A[i+1], A[i*2]
```

- Có thuận lợi rất lớn khi ta sử dụng một biểu thức như là chỉ số của mảng, bởi khi đó nó cho phép ta có thể truy xuất từ đầu đến cuối mảng bằng một vòng lặp.

Ví dụ: có một mảng được khai báo như sau:

```
const int SIZE=5;
int A[SIZE];
```

giả sử như mảng đã có giá trị và ta cần tính tổng giá trị của mảng đưa vào biến sum, thay vì phải viết lệnh:

```
sum= A[0]+A[1]+A[2]+A[3]+A[4];
```

có thể viết lại như sau:

```
for (i=0;i<SIZE;i++)
     sum +=A[i];
```

Việc sử dụng vòng lặp để truy xuất một mảng rõ ràng rất thích hợp khi truy xuất một mảng lớn.

4.1.4. *Tham số của hàm là mảng*

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
//Khai bao hang so SIZE
const int SIZE=100;
//Khai bao prototype
void TaoMangNgauNhienDuong(int A[],int n);
void XuatToanBoMang (int A[], int n);
// Ham chinh cua chuong trinh
int main()
{   int n, A[SIZE];
    srand((unsigned int) time (NULL));
    printf("Nhap so phan tu cua mang: ");
    scanf("%d",&n);
    TaoMangNgauNhienDuong(A,n);
    printf("Mang da nhap:");
    XuatToanBoMang (A,n);
    return 0;
}
// Phan chua cac ham
void TaoMangNgauNhienDuong(int Arr[],int n)
{   int min=50;
    int max=100;
    for (int i=0; i<n; i++)
        Arr[i]=(rand()%(max-min+1))+min;
}
void XuatToanBoMang (int Arr[], int n)
{   for (int i = 0; i < n; i ++)
        printf("%5d",Arr[i]);
}
```

4.2. MỘT SỐ KỸ THUẬT CƠ BẢN XỬ LÝ TRÊN MẢNG

4.2.1. *Nhập*

4.2.1.1. Nhập (gán) dữ liệu cho 1 phần tử

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

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

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

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

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

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

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

VD: `int A[SIZE]={0};`

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

4.2.1.3.1. Người dùng tự nhập giá trị cho từng phần tử của mảng

Ví dụ:

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

4.2.1.3.2. Phát sinh giá trị ngẫu nhiên cho từng phần tử của mảng

Ví dụ:

```
void TaoMangNgauNhienDuong(int A[],int n)
{
    int min=50;
    int max=100;
    for (int i=0; i<n; i++)
        A[i]=(rand()%(max-min+1))+min;
}
void TaoMangNgauNhienAmDuong(int A[],int n)
{   int min=50;
    int max=100;
    for (int i=0; i<SIZE; i++)
    {   //B1: tao dau +/-
        int dau,x;
        x=rand();
        if (x%2==1)
```

```
                dau=-1;
            else
                dau=1;
            //B2: random so X dau
            A[i]=dau*((rand()%(max-min+1))+min );
        }
    }
```

4.2.2. *Xuất (liệt kê) mảng một chiều*

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

4.2.2.1. *Xuất (liệt kê) toàn bộ các phần tử của mảng*

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

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

4.2.2.2.1. *Điều kiện lựa chọn được gán trực tiếp trong biểu thức điều kiện của phát biểu if*

- *Điều kiện căn cứ trên **giá trị** của từng phần tử trong mảng:*
  - Xuất các số chẵn có trong mảng

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

  - Xuất các số âm có trong mảng

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

- *Điều kiện căn cứ vào **vị trí** của phần tử trong mảng:*
  - Xuất các số tại vị trí chẵn trong mảng

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

4.2.2.2.2. *Điều kiện lựa chọn là kết quả trả về của một hàm trong biểu thức điều kiện của phát biểu if*

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

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

4.2.3. *Tính tổng – Tính trung bình có điều kiện*

4.2.3.1. *Tính tổng*

4.2.3.1.1. *Tính tổng toàn bộ các phần tử trong mảng*

VD: tính tổng tất cả các số có trong mảng

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

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

VD: tính tổng các số âm có trong mảng

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

4.2.3.2. *Tính trung bình*

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

VD: tính tổng tất cả các số có trong mảng

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

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

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

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

VD: tính tổng các số âm có trong mảng

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

4.2.4. *Kỹ thuật đặt cờ hiệu*

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

4.2.4.1. *Dạng 1: (dạng ∀)*

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

4.2.4.1.1. *Kiểm tra giá trị từng phần tử riêng lẻ*

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

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

Lấy phủ định 2 vế của ① $\Leftrightarrow$ $\exists$ (A[i]<0) $\Rightarrow$ là sai ②

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

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

```
bool KiemTraMangToanDuong (int A[ ], int n)
{
    for (int i = 0; i < n; i ++ )
        if (A[i]<0) // ⇔ !(A[i]>=0)⇨Vi phạm điều kiện dương
            return  false;
    //Nếu đã tìm hết trên mảng nhưng vẫn không thấy số âm
    //⇒ mảng chứa toàn số dương
    return true;
}
```

4.2.4.1.2. Kiểm tra giá trị giữa 2 phần tử trong mảng

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

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

```
bool KiemTraMangTangDan (int A[ ], int n)
{
    for (int i = 0; i < n-1; i++)
        if (A[i]<A[i+1]) // ⇔ !(A[i]>=A[i+1])⇨Vi phạm
            return  false;
    //Nếu đã duyệt hết trên mảng nhưng vẫn không thấy vi phạm
    //⇒ mảng tăng
    return true;
}
```

4.2.4.2. Dạng 2: (dạng ∃)

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

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

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

4.2.5. *Kỹ thuật đặt lính canh*

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

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

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

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

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

```
    return -1;
}
```

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

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

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

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

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

4.2.6.2. *Đếm số lần xuất hiện của 1 gá trị nào đó*

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

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

4.2.6.3. *Đếm số lần xuất hiện của từng số có trong mảng*

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

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

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

```
            printf("So %d xuat hien %d lan",i,B[i]);
    }
```

4.2.7. *Sắp xếp*

Viết hàm sắp xếp mảng theo thứ tự tăng dần.

```
void HoanVi (int &a, int &b)
{
    int tam = a;
    a = b;
    b = tam;
}
void SapTang (int A[], int n)
{
    for (int i = 0; i < n-1 ; i++)
        for (int j = i+1; j < n; j++)
            if (A[i] > a [j])
                HoanVi (A[i], A[j]);
}
```

4.2.8. *Xoá 1 phần tử khỏi mảng*

Các bước thực hiện:

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

4.2.8.1. *Viết hàm xoá phần tử tại vị trí (vitri) cho trước trong mảng*

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

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

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

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

```
void XoaTatCaGiaTri (int A[], int &n, int SoCanXoa)
{
    int i = 0;
```

```
        while (i<n)
            if (A[i] == SoCanXoa)
              XoaTaiViTri (A, n, i);
            else
              i++;
    }
```

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

Các bước thực hiện:

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

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

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

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

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

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

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

4.2.10. *Tách mảng*

Cho mảng A kích thước n. Tách mảng A thành 2 mảng B và C sao cho: B có ½ phần tử đầu của mảng A, ½ phần tử còn lại đưa vào mảng C.

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

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

4.2.11. *Ghép mảng*

4.2.11.1. Ghép tuần tự

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

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

4.2.11.2. Ghép xen kẽ(đan xen)

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

Cách thực hiện:

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

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

*Đặt **ia** là chỉ số của mảng A; **ib**: chỉ số của mảng B và **ic** là chỉ số của mảng c.*

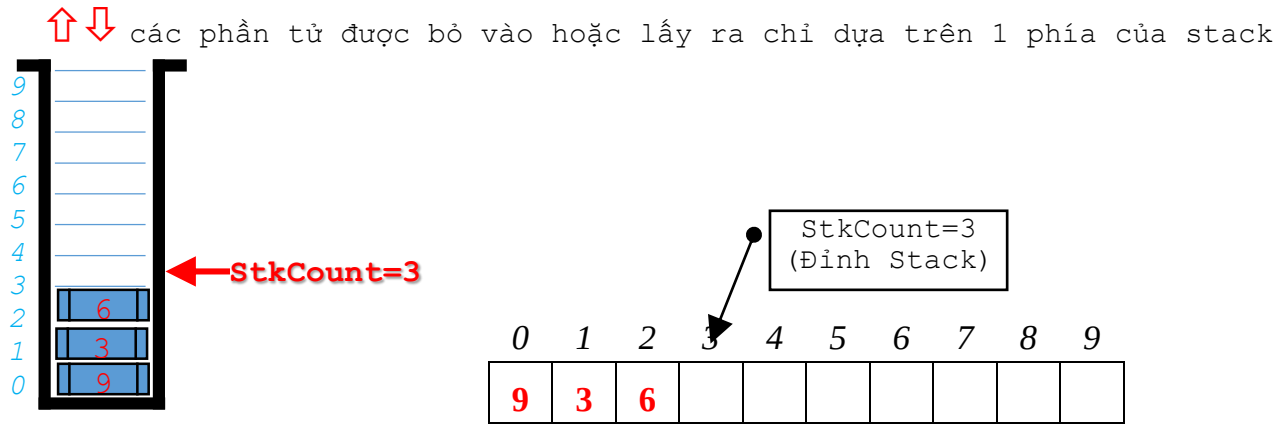
```
void NoiMang(int A[],int na,int B[],int nb,int C[],int &nc)
{
    int ia=0, ib=0, ic=0;
    while(ia<na && ib<nb)
    {
        if (A[ia]<B[ib])
            C[ic++]=A[ia++];
        else
          C[ic++]=B[ib++];
    }
    //nếu mảng A còn thì đưa hết vào C
    while(ia<na)
        C[nc++]=A[ia++];
    //nếu mảng B còn thì đưa hết vào C
    while(ib<nb)
        C[nc++]=B[ib++];
}
```

4.3. TỔ CHỨC STACK (NGĂN XẾP) TRÊN MẢNG MỘT CHIỀU

4.3.1. *Khái niệm*

Stack là một danh sách mà việc thêm vào và loại bỏ chỉ diễn ra cùng một đầu của danh sách, tức là theo cơ chế LIFO (Last In First Out). Stack gồm nhiều phần tử có cùng kiểu dữ liệu, phần tử trên cùng Stack luôn luôn có con trỏ chỉ tới ta gọi là Stack Pointer (ký hiệu: sp, trong các phần sau sẽ sử dụng với tên là **StkCount**).

Để tạo Stack ta có hai cách: danh sách tuyến tính (mảng) hoặc danh sách liên kết (con trỏ). Trong chương này, ta chỉ quan tâm đến việc tạo Stack bằng mảng một chiều.



Minh họa Stack có khả năng chứa tối đa 10 phần tử và hiện đang chứa 3 phần tử

4.3.2. *Ứng dụng của Stack*

- Stack thường được dùng trong các bài toán có cơ chế LIFO (vào sau ra trước).
- Stack cũng được dùng trong các bài toán khử đệ quy (chuyển một giải thuật đệ quy thành giải thuật không đệ quy)

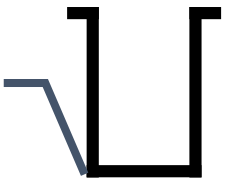
4.3.3. *Khai báo kiểu cấu trúc stack*

```
#define TRUE 1
#define FALSE 0
typedef struct stack
{
     int* StkArray;
     int StkMax;
     int StkCount;
} STACK;
```

4.3.4. *Các phép toán trên Stack*

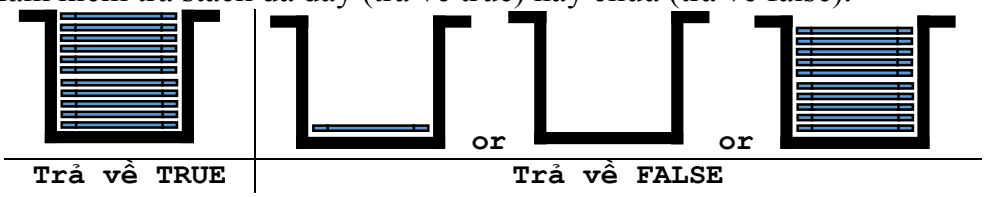
4.3.4.1. *InitStack: khởi tạo stack trước khi sử dụng*

```
int InitStack (STACK &s, int MaxItems)
{
     s.StkArray = new int[MaxItems];
     if (s.StkArray == NULL)
          return 0;          //Không cấp phát được bộ nhớ
     s.StkMax = MaxItems;    //sức chứa tối đa
     s.StkCount = 0;         //chưa có phần tử nào trong Stack
     return 1;               //khởi tạo thành công
}
```



4.3.4.2. IsFull

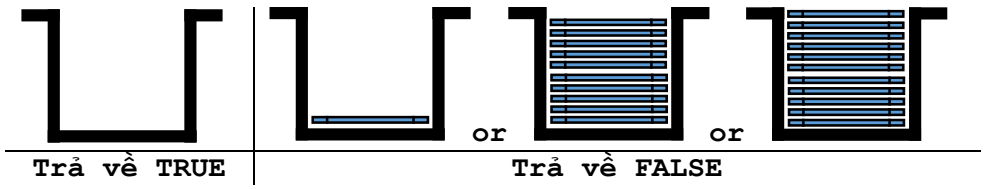
Hàm kiểm tra stack đã đầy (trả về true) hay chưa (trả về false).




```
int  IsFull(STACK s)
{ /*return true nếu Stack đầy, return false khi chưa đầy*/
    return (s.StkCount>s.StkMax);
}
```

4.3.4.3. IsEmpty

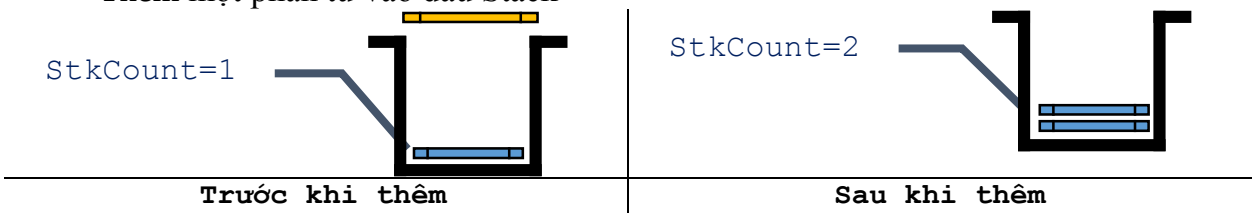
Hàm kiểm tra stack có rỗng (không chứa phần tử nào, trả về true) hay không (trả về false).




```
int IsEmpty(STACK s)
{ /*return true nếu Stack rỗng, return false nếu không rỗng*/
    return (s.StkCount==0);
}
```

4.3.4.4. Push

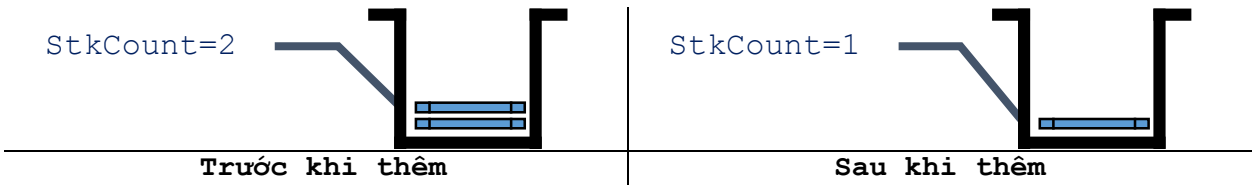
Thêm một phần tử vào đầu Stack




```
int Push(STACK &s, int NewItem)
{
    if (IsFull(s))
        return FALSE;  //stack đầy ⇨ không thể thêm
    s.StkArray[s.StkCount] = NewItem;
    s.StkCount++;
    return TRUE;      //thêm thành công
}
```

4.3.4.5. Pop

Xóa một phần tử khỏi Stack, trả cho chương trình gọi giá trị của phần tử vừa xóa. Trước khi xóa, ta phải kiểm tra Stack có khác rỗng hay không.



```
int Pop(STACK& s, int & OutItem)
{
     if (IsEmpty(s))
       return false;  //Stack rỗng ⇨ không lấy được
     s.StkCount--;
     OutItem = s.StkArray[s.StkCount];
     return true;      //lấy ra thành công
}
```

4.3.5. *Ví dụ*

4.3.5.1. Viết chương trình sử dụng stack để đổi số nguyên không âm ở hệ thập phân sang số ở hệ nhị phân

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define TRUE 1
#define FALSE 0
typedef struct stack
{
     int* StkArray;
     int StkMax;
     int StkCount;
} STACK;
int InitStack(STACK &s, int MaxItems)
{
     s.StkArray = new int[MaxItems];
     if (s.StkArray == NULL)
          return 0;
     s.StkMax = MaxItems;
     s.StkCount = 0;
     return 1;
}
int  IsFull(STACK s)
{ /*return true nếu Stack đầy, return false khi chưa đầy*/
     return (s.StkCount>s.StkMax);
}
int IsEmpty(STACK s)
{ /*return true nếu Stack rỗng, return false nếu không rỗng*/
     return (s.StkCount == 0);
}
int Push(STACK &s, int NewItem)
{
     if (IsFull(s))
          return FALSE;
     s.StkArray[s.StkCount] = NewItem;
     s.StkCount++;
     return TRUE;
}
int Pop(STACK &s, int &OutItem)
{
     if (IsEmpty(s))
          return FALSE;
     s.StkCount--;
     OutItem = s.StkArray[s.StkCount];
     return TRUE;
}
```

```
void main()
{
     STACK s;
     int  sodu, MaxItems,x;
     long  so, temp;
     char c;
     printf("\nNhap vao kich thuoc cua stack can dung: ");
     scanf("%d", &MaxItems);
     printf("\nNhap vao so thap phan can chuyen sang nhi phan: ");
     scanf("%ld", &so);
     temp = so;
     do
     {
          sodu = temp % 2;
          Push(s, sodu);
          temp = temp / 2;
     } while (temp != 0);
     printf("So %d trong he thap phan doi sang he nhi phan la:", so);
     while (!IsEmpty(s))
     {
          Pop(s, x);
          printf("%d", x);
     }
}
```

4.3.5.2. *Viết chương trình tính trị một biểu thức dạng hậu tố (PostFix)*

Biết rằng mỗi số hạng là 1 ký số và các toán tử trong biểu thức gồm có: cộng(+), trừ (-), nhân (*), chia (/), lũy thừa (^). Dạng hậu tố của biểu thức có dạng như sau (lưu ý: các toán hạng và toán tử được nhập liên tiếp – không có khoảng trắng):

```
82-        = 6
84-21+^    = 64
23+3^      = 125
```

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include <math.h>
#define MAXSIZE 100
#define TRUE 1
#define FALSE 0
// Khai bao stack chua cac toan hang
typedef struct stack
{
     int StkCount;
     double nodes[MAXSIZE];
}STACK;
int InitStack (STACK &s)
{
     s.StkCount = 0;
     return 1;
}
int IsEmpty(STACK s)
{
  if (s.StkCount == 0)
    return TRUE;
  else
    return FALSE;
}
```

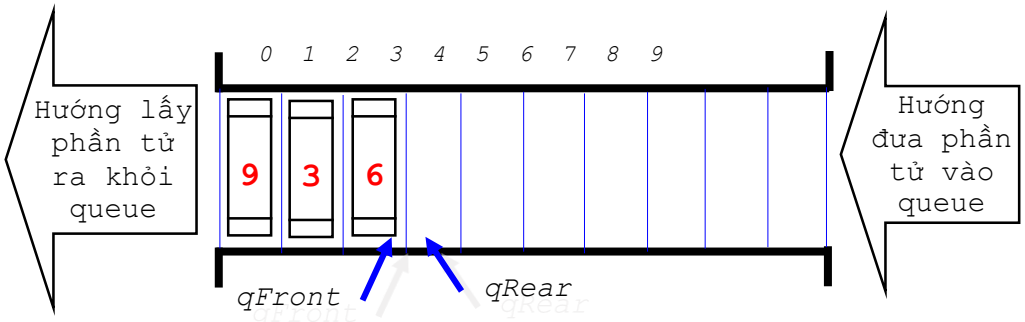
```
int push(STACK &s, double x)
{
  if (s.StkCount == MAXSIZE - 1) return 0;
  s.nodes[(s.StkCount)++] = x;
  return 1;
}
int pop(struct stack &s, double &x)
{
  if (IsEmpty(s))
    return 0;
  x = s.nodes[--(s.StkCount)];
  return 1;
}
/* Ham tinh: tính trị của hai toán hạng toanhang1 va toanhang2 qua
phép toán toantu */
double tinh(int toantu, double toanhang1, double toanhang2)
{
  switch (toantu)
  {
    case '+':
      return(toanhang1 + toanhang2);
      break;
    case '-':
      return(toanhang1 - toanhang2);
      break;
    case '*':
      return(toanhang1 * toanhang2);
      break;
    case '/':
      return(toanhang1 / toanhang2);
      break;
    case '^':
      return(pow(toanhang1, toanhang2));
      break;
    default:
      printf("%s", "toan tu khong hop le");
      exit(1);
  }
}
// Hàm XacDinhGiaTriBieuThuc: tính một biểu thức postfix
double XacDinhGiaTriBieuThuc (STACK s, char bieuthuc[])
{   int c, i;
    double toanhang1, toanhang2, tri;

    for (i = 0; (c = bieuthuc[i]) != '\0'; i++)
        if (c >= '0' && c <= '9')  // c la toan hang
            push(s, (double)(c - '0'));
        else            // c la toan tu
        {
            pop(s, toanhang2);
            pop(s, toanhang1);
            //tinh ket qua trung gian
            tri = tinh(c, toanhang1, toanhang2);
            push(s, tri);
        }
    pop(s, toanhang1);
    return(toanhang1);
}
```

```
void main()
{    char bieuthuc[MAXSIZE];
     STACK s;
     InitStack(s);  // khoi tao stack
     printf("\nNhap bieu thuc postfix can dinh tri: ");
     gets(bieuthuc);
     double ketqua = XacDinhGiaTriBieuThuc(s, bieuthuc);
     if (IsEmpty(s))
       printf("Bieu thuc %s co tri la %5.2f", bieuthuc, ketqua);
     else
       printf("Bieu thuc sai nen khong the tinh");
     getch();
}
```

4.4. TỔ CHỨC QUEUE (HÀNG ĐỢI) TRÊN MẢNG MỘT CHIỀU

4.4.1. *Khái niệm*

- Queue là một danh sách hạn chế mà việc thêm vào được thực hiện ở đầu danh sách, và việc loại bỏ được thực hiện ở đầu còn lại (FIFO - First In First Out).
- Queue chứa các phần tử có cùng kiểu dữ liệu.
- Queue cũng có thể được tổ chức theo danh sách tuyến tính hoặc danh sách liên kết. Ở đây, ta chỉ tổ chức queue theo mảng 1 chiều.



4.4.2. *Khai báo queue*

- Cấu trúc Queue gồm các thành phần chính:
  - ***qArray*** : mảng 1 chiều, mỗi phần tử của mảng là 1 phần tử trong queue.
  - **qMax** : sức chứa tối đa
  - **qCount** : số phần tử thực tế hiện có.
  - ***front*** : số nguyên chỉ đầu hàng đợi. Đây là vị trí của phần tử sẽ bị loại bỏ khi cần.
  - ***rear*** : số nguyên chỉ đầu và cuối hàng đợi. Đây là vị trí của phần tử sẽ được thêm mới vào queue khi cần.
- Khai báo

```
#define TRUE 1
#define FALSE 0
typedef struct QUEUE
{
     int* qArray;//chứa các phần tử của Queue
     int qMax;  //sức chứa tối đa
     int qCount;// số phần tử hiện có
     int qFront;// xác định vị trí đầu
     int qRear; // xác định vị trí cuối
};
```

4.4.3. *Ứng dụng của queue*

- Thường được dùng trong các bài toán có cơ chế FIFO (vào trước ra trước).

- Thường được dùng trong các công việc đang đợi phục vụ trong các hệ điều hành đa nhiệm, để quản lý các hàng đợi in trên máy in mạng (print server), ...

4.4.4. *Các phép toán trên queue*

4.4.4.1. *Khởi tạo Queue*

```
int InitQueue(QUEUE& q, int MaxItem)
{
      q.qArray = new int[MaxItem];
      if (q.qArray == NULL)
         return FALSE;    // không đủ bộ nhớ
     q.qMax = MaxItem;
     q.qCount = 0;
     q.qFront = q.qRear = -1;
     return TRUE;         // thành công
}
```

4.4.4.2. *Kiểm tra queue đã đầy hay chưa*

```
int IsFull(QUEUE q)
{
     if (q.qCount == q.qMax)
          return TRUE; // đã đầy
     return FALSE; // không (hoặc chưa) đầy
}
```

4.4.4.3. *Kiểm tra queue có rỗng hay không*

```
int IsEmpty(QUEUE q)
{
     if (q.qCount == 0)
          return TRUE; // rỗng
     return FALSE;     // không rỗng
}
```

4.4.4.4. *Thêm phần tử vào queue*

4.4.4.4.1. *Thêm một phần tử x vào queue*

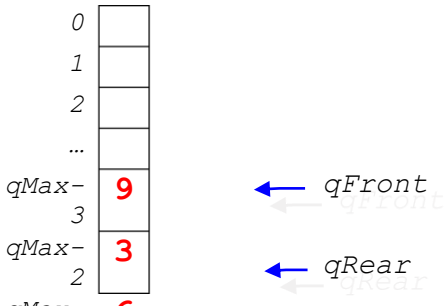
Khi thêm lưu ý xem hàng đợi bị tràn hay bị đầy để xử lý cho thích hợp.

```
int AddItemToQueue(QUEUE &q, int x)
{
     if (IsFull(q))
          return 0; // không thêm được vào queue
     if (q.qFront == -1) /*phần tử được thêm là phần tử đầu tiên có
                                                          trong queue*/
     {
          q.qFront = 0;
          q.qRear = -1;
     }
     if (q.qRear == q.qMax)
          q.qRear = -1;
     ++q.qRear;
     q.qArray[q.qRear] = x;
     q.qCount++;
     return 1;  // thêm vào queue thành công
}
```

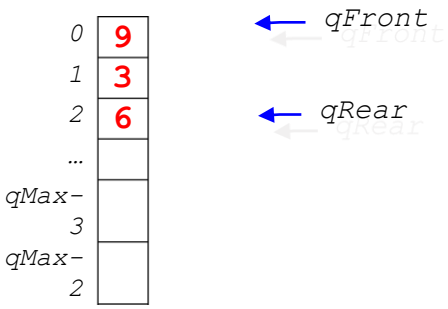
4.4.4.4.2. *Hàng đợi bị tràn*

Là trường hợp khi queue vẫn còn chỗ trống, nhưng `qRear = qMax-1` (=n) và `qFront>0`, tức là không thể thêm phần tử mới vào cuối của queue. Để khắc phục trường hợp này, có thể dùng một trong 2 cách:

- *Cách 1*: Di chuyển tịnh tiến từng phần tử lên để có `qFront=0` và `qRear<qMax-1`: trường hợp này Front luôn nhỏ hơn Rear).

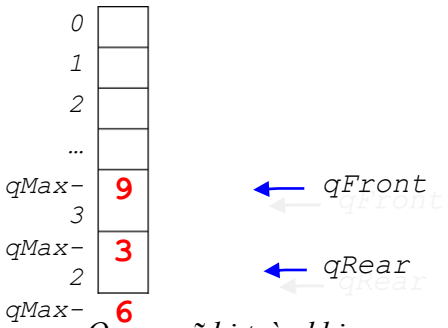


Queue sẽ bị tràn khi thêm phần tử mới

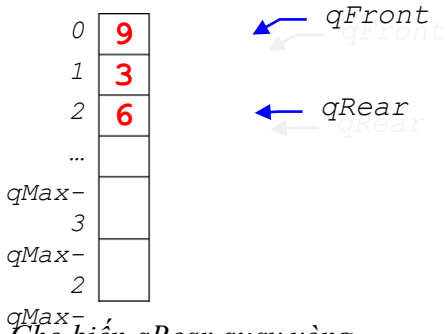


Tịnh tiến các phần tử về đầu của queue trước khi thêm phần tử mới ⇨ `qRear≥qFront`

- *Cách 2*: Di chuyển vòng: cho `qRear=0`, `qFront` giữ nguyên. Thông thường `qFront<qRear`, trong trường hợp này `qFront` lớn hơn `qRear`.



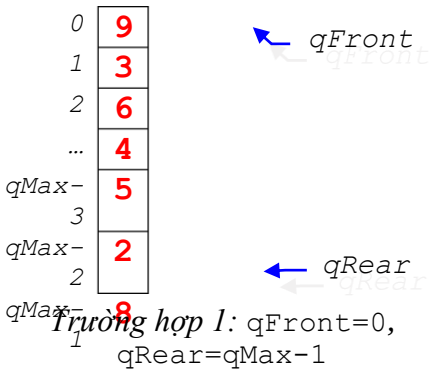
Queue sẽ bị tràn khi thêm phần tử mới



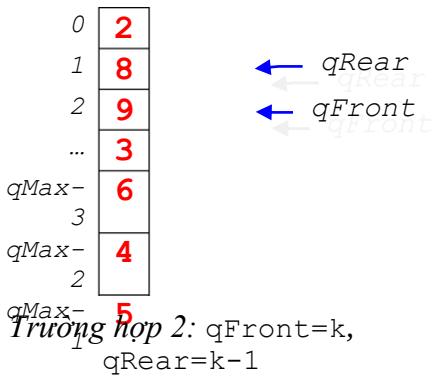
Cho biến qRear quay vòng `(qRear=0)` ⇨ `qRear<qFront`

4.4.4.4.3. *Hàng đợi bị đầy*

Hàng đợi không còn chỗ trống, do đó nếu tiếp tục thêm vào sẽ bị mất dữ liệu.



Trường hợp 1: `qFront=0, qRear=qMax-1`



Trường hợp 2: `qFront=k, qRear=k-1`

4.4.4.5. *Loại bỏ phần tử khỏi queue*

Loại bỏ phần tử khỏi Queue và trả về giá trị của phần tử vừa xóa. Trước khi xóa, phải kiểm tra Queue có khác rỗng hay không?

```
int RemoveOneItemOfQueue(QUEUE &q, int &x)
{
     if (IsEmpty(q))
          return FALSE;       // Queue rỗng
     x = q.qArray[q.qFront];
     if (q.qFront == q.qRear)  // Queue chi co 1 phan tu
     {
          q.qFront = -1;
          q.qRear = -1;
          q.qCount=0;
     }
     else
     {
       q.qFront++;
       if (q.qFront == q.qMax)
            q.qFront = 0;
       q.qCount--;
     }
   return TRUE;
}
```

4.4.5. *Ví dụ*

Viết chương trình đổi số thực không âm ở hệ thập phân sang số ở hệ nhị phân, tối đa ta chỉ lấy 8 số lẻ trong hệ nhị phân.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#define TRUE 1
#define FALSE 0
typedef struct QUEUE
{
     int *qArray; //chứa các phần tử của Queue
     int qMax;    //sức chứa tối đa
     int qCount;  // số phần tử hiện có
     int qFront;  // xác định vị trí đầu
     int qRear;   // xác định vị trí cuối
};
int InitQueue(QUEUE& q, int MaxItem)
{
     q.qArray = new int[MaxItem];
     if (q.qArray == NULL)
          return FALSE;  // không đủ bộ nhớ
     q.qMax = MaxItem;
     q.qCount = 0;
     q.qFront = q.qRear = -1;
     return TRUE;   // thành công
}
int IsFull(QUEUE q)
{
     if (q.qCount == q.qMax)
          return TRUE; // đã đầy
     return FALSE; // không (hoặc chưa) đầy
}
int IsEmpty(QUEUE q)
{
     if (q.qCount == 0)
```

```
            return TRUE; // rỗng
        return FALSE;     // không rỗng
    }
    int AddItemToQueue(QUEUE &q, int x)
    {
        if (IsFull(q))
            return 0; // không thêm được vào queue
        if (q.qFront == -1) /*phần tử được thêm là phần tử đầu tiên có
                                                              trong queue*/
        {
            q.qFront = 0;
            q.qRear = -1;
        }
        if (q.qRear == q.qMax)
            q.qRear = -1;
        ++q.qRear;
        q.qArray[q.qRear] = x;
        q.qCount++;
        return 1;  // thêm vào queue thành công
    }
    int RemoveOneItemOfQueue(QUEUE &q, int &x)
    {
        if (IsEmpty(q))
            return FALSE;       // Queue rỗng
        x = q.qArray[q.qFront];
        if (q.qFront == q.qRear)  // Queue chi co 1 phan tu
        {
            q.qFront = -1;
            q.qRear = -1;
            q.qCount=0;
        }
        else
        {
          q.qFront++;
          if (q.qFront == q.qMax)
              q.qFront = 0;
          q.qCount--;
        }
      return TRUE;
    }
    void DecToBin(QUEUE q, float num)
    {
        float PhanNguyen;
        float r, PhanLe;
        int k, so;
        /* Hàm modf tách số double n ra thành 2 phần: phần nguyên chứa
        trong bien PhanNguyen và phần thập phân chứa trong biến PhanLe*/
        PhanLe = modf(num, &PhanNguyen);
        printf("\n So %f he thap phan chuyen sang he nhi phan la:",num);
        //B1: Đổi phần nguyên của số thực num sang hệ nhị phân=> KHÔNG sử
      dụng STACK
        k = 1;
        int SoNguyen = (int)PhanNguyen;
        while (k * 2 <= SoNguyen)
            k *= 2;
        while (k > 0)
        {   printf("%d", SoNguyen / k);
            SoNguyen = SoNguyen%k;
```

```
            k = k / 2;
        }
        //B2: Đổi phần lẻ số thực num sang hệ nhị phân=> sử dụng QUEUE
        int dem = 0;
        do
        {
            r = PhanLe * 2;
            PhanLe = modf(r, &PhanNguyen);
            AddItemToQueue(q, PhanNguyen);
            dem++;
        } while (dem <8 && r != 1);
        printf(".");
        while (!IsEmpty(q))
        {
            RemoveOneItemOfQueue(q, k);
            printf("%d",k);
        }
    }
    // chuong trinh chinh
    void main()
    {
        float f;
        int n;
        QUEUE q;
        printf("\nSo Luong phan tu can dung cho queue: ");
        scanf("%d", &n);
        InitQueue(q, n);
        printf("\nNhap so thuc: ");
        scanf("%f", &f);
        DecToBin(q, f);
        getch();
    }
```

4.5. BÀI TẬP

4.5.1. *Thao tác trên một mảng*

4.5.1.1. Nhập mảng

4.5.1.1.1. Nhập giá trị các phần tử của mảng từ bàn phím

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

✍ *Mở rộng*:

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

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

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

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

VD1: 2 4 7 7 8 11 ⇨Hợp lệ

VD2: 2 4 ***3*** ⇨không hợp lệ⇨yêu cầu nhập lại

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

✍ *Mở rộng* cho các trường hợp số hoàn thiện, số chính phương, …

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

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

a. Với 5<n<200, và giá trị của các phần tử trong khoảng từ 0 đến 99.

b. Với 5<n<50, và cho giá trị của các phần tử trong khoảng từ -100 đến 100.

c. Với 5<n<50, và các số nguyên sao cho giá trị phát sinh ngẫu nhiên phải toàn là số lẻ.

✍ *Mở rộng* cho các trường hợp số chẵn, số vừa âm vừa lẻ, số nguyên tố, số hoàn thiện, số hoàn hảo, …

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

✍ *Mở rộng* cho trường hợp số giảm dần.

7/- Viết chương trình phát sinh ngẫu nhiên mảng 1 chiều các số nguyên gồm n phần tử, với yêu cầu những vị trí (hay chỉ số mảng) là số lẻ chỉ nhận giá trị nhập vào là số lẻ, và những vị trí (hay chỉ số mảng) là số chẵn chỉ nhận giá trị nhập vào là số chẵn.

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

4.5.1.2. Xuất mảng

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

4.5.1.2.1. Xuất theo giá trị phần tử có trong mảng

9/- Liệt kê các phần tử trong mảng có giá trị lẻ.

✍ *Mở rộng*: giá trị chẵn, giá trị âm, giá trị dương, …

10/- Liệt kê các phần tử trong mảng có giá trị là số chẵn và nhỏ hơn 20.

✍ *Mở rộng*: giá trị chẵn và là số âm, giá trị lẻ và là số dương, …

11/- Viết hàm (ngoài các tham số cần thiết) nhận tham số là số x, và in ra tất cả các số có giá trị lớn hơn x.

✍ *Mở rộng*: khác X, nhỏ hơn hay bằng X, bội số của X, ước số của X, …

12/- Liệt kê các giá trị là số nguyên tố có trong mảng.

✍ *Mở rộng* cho các trường hợp số hoàn thiện, số hoàn hảo, …

13/- Liệt kê các số trong mảng có giá trị thuộc [x,y] cho trước (x và y là tham số của hàm)

✍ *Mở rộng*: liệt kê các số chẵn trong mảng nguyên thuộc [x,y].

4.5.1.2.2. Xuất theo vị trí (hay chỉ số) của mảng

14/- Liệt kê các phần tử nằm tại vị trí lẻ.
✍ *Mở rộng* cho trường hợp: vị trí chẵn, vị trí là số hoàn thiện, số hoàn hảo, …

15/- Liệt kê các phần tử trong mảng có vị trí là số chẵn và nhỏ hơn 20.
✍ *Mở rộng* cho trường hợp: vị trí từ p đến q (thay vì từ 0 đến n-1 như thường dùng), vị trí từ 0 đến p hoặc vị trí từ q đến n-1, …

16/- Viết hàm (ngoài các tham số cần thiết) nhận tham số là số x, và in ra tất cả các số có vị trí lớn hơn x.
✍ *Mở rộng*: khác X, nhỏ hơn hay bằng X, bội số của X, ước số của X, …).

4.5.1.2.3. Xuất dựa trên kết hợp giữa vị trí và giá trị

17/- In các phần tử có giá trị là số nguyên tố nằm tại những vị trí chẵn trong mảng.

18/- * Liệt kê tất cả các cặp giá trị (a,b) trong mảng thỏa điều kiện a>=b và vị trí (chỉ số) chứa số a < vị trí (chỉ số) chứa số b.

19/- Viết hàm nhận tham số vào 1 số K (với -999<=K<=999). In ra cách đọc chữ số tương ứng (sử dụng mảng chứa các chuỗi "không", "một", "hai", …).
Ví dụ: nhập -456 in ra: Am bon nam sau.
✍ *Mở rộng*: cho trường hợp giá trị của K<= ± 2.000.000.000.

20/- Viết chương trình nhập vào năm. In ra tên của năm âm lịch tương ứng.
Ví dụ: nhập 1999 in ra Kỷ Mão.

Biết rằng:

CAN	*Giáp*	*At*	*Bính*	*Đinh*	*Mậu*	*Kỷ*	*Canh*	*Tân*	*Nhâm*	*Quý*		
CHI	*Tý*	*Sửu*	*Dần*	*Mão*	*Thìn*	*Tỵ*	*Ngọ*	*Mùi*	*Thân*	*Dậu*	*Tuất*	*Hợi*

4.5.1.2.4. Các trường hợp xuất khác

21/- Mỗi yêu cầu sau đây được viết thành một hàm riêng biệt:

a)- In mảng vừa tạo thành 2 dòng: ✧ Dòng 1: các số lẻ.
✧ Dòng 2: các số chẵn.

b)- In mảng vừa tạo thành 2 dòng: ✧ Dòng 1: các số âm
✧ Dòng 2: các số dương.

c)- Vẽ biểu đồ ngang theo giá trị của các số có trong array.
Ví dụ: array có 3 số 2, 7, 4 in ra:
```
x x
x x x x x x x
x x x x
```

d)- Vẽ biểu đồ đứng theo giá trị của các số có trong array.
Ví dụ: array có 3 số 2, 0, 4, 1 sẽ in ra:
```
      x
      x
x     x
x     x     x
```

4.5.1.3. Kỹ thuật đặt cờ hiệu (kết quả trả về là có/không thỏa điều kiện cần kiểm tra)

4.5.1.3.1. Kiểm tra mảng có tồn tại giá trị thỏa điều kiện cho trước

22/- Kiểm tra mảng có chứa giá trị 0 hay không? Có trả về 1, không có trả về 0.

✍ *Mở rộng*:
- Mảng có chứa số âm, số nguyên tố, số hoàn thiện, …
- Thay giá trị 0 bằng tham số X. Kiểm tra xem mảng có chứa giá trị nhỏ hơn hay bằng X, bội số của X, ước số của X, …).

23/- Liệt kê các số nguyên tố nhỏ hơn giá trị của tham số X, nếu mảng không tồn tại số nguyên tố nào nhỏ hơn X thì phải xuất ra một câu thông báo (VD: mảng không chứa số nguyên tố nhỏ hơn X).

✍ *Mở rộng* cho các trường hợp số hoàn thiện, số chính phương, …

4.5.1.3.2. Kiểm tra tất cả các phần tử của mảng thỏa điều kiện

24/- Kiểm tra mảng một chiều chứa toàn số âm hay không? Có trả về true, không có trả về false.

✍ *Mở rộng*: cho mảng toàn dương, mảng toàn chẵn, mảng toàn lẻ, mảng đồng nhất, mảng toàn số nguyên tố, …

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

✍ *Mở rộng*: trường hợp mảng giảm, mảng đối xứng, mảng cấp số cộng, …

4.5.1.4. Kỹ thuật đặt lính canh (áp dụng cho dạng bài toán "tìm kiếm một giá trị", "tìm kiếm một vị trí")

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

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

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

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

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

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

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

✍ *Mở rộng*:
- Tìm vị trí của phần tử có giá trị dương nhỏ nhất.

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

30/- Tìm giá trị của phần tử trong mảng "xa giá trị x nhất"

✍ *Mở rộng*:
- Tìm giá trị của phần tử trong mảng "gần giá trị x nhất".
- Tìm cả giá trị và vị trí của phần tử trong mảng "xa/gần giá trị x nhất".

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

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

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

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

✍ *Mở rộng*: Tìm giá trị đầu tiên trong mảng có dạng 3^k, 4^k,…

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

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

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

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

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

✍ *Mở rộng*:
- Giá trị tại vị trí đó bằng tổng 2 giá trị có trong vị trí kế cận (A[i]= A[i-1]+A[i+1]).
- Giá trị tại vị trí đó bằng tích 2 giá trị có trong vị trí kế cận (A[i]= A[i-1]*A[i+1]).
- Xét cả 2 vị trí đầu và cuối mảng (chỉ có 1 phần tử kế cận).

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

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

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

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

44/- Tìm chữ số xuất hiện nhiều nhất trong mảng

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

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

45/- Tìm 2 giá trị gần nhau nhất trong mảng

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

Ví dụ: cho mảng A

24	45	23	13	43	-12

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

9	30	8	2	28	27

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

✍ *Mở rộng*: Tìm phần tử đầu tiên trong mảng "gần giá trị x nhất".

Ví dụ: cho mảng A

24	45	23	13	43	-12

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

9	30	8	2	28	27

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

4.5.1.5. Kỹ thuật đếm / tính tổng / tính trung bình

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

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

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

✍ *Mở rộng*:

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

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

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

51/- Tổng các phần tử lớn hơn phần tử liền kề

52/- Tổng các phần tử đối xứng

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

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

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

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

55/- Viết hàm tính giá trị trung bình của các số hoàn thiện trong mảng.

✍ *Mở rộng*:

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

4.5.1.6. Đếm

56/- Đếm số lần xuất hiện của giá trị x trong mảng.

✍ *Mở rộng*:

- Đếm số lần xuất hiện của giá trị dương, giá trị âm, số chẵn, số lẻ, bội số của 5, bội số của X, ước số của 7, ước số của X, …

- Đếm số lần xuất hiện của giá trị là số nguyên tố, là số hoàn thiện, là số chính phương, số Armstrong, …

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

58/- Đếm các phần tử có chữ số đầu là chữ số lẻ

✍ *Mở rộng*:
- Đếm các phần tử có chữ số đầu là chẵn, …
- Đếm các phần tử có chữ số hàng chục là 5
- Đếm các phần tử có chữ số hàng đơn vị là 1 trong các số 3, 6, 9

59/- Đếm số đối xứng trong mảng

60/- Đếm số lượng phần tử lớn hơn các phần tử liền kề, thực hiện cho 2 trường hợp:
- KHÔNG xét 2 phần tử đầu và cuối mảng vì không đủ 2 phần tử liền kề).
- CÓ xét 2 phần tử đầu và cuối mảng (2 phần tử này chỉ có 1 phần tử liền kề trước hoặc sau).

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

4.5.1.7. Trung bình

61/- Trung bình cộng các số dương

✍ *Mở rộng*:
- Trung bình cộng các số âm, số chẵn, số lẻ.
- Trung bình cộng các số lớn hơn x, nhỏ hơn x
- Trung bình cộng các số nguyên tố, các số hoàn thiện, các số chính phương, …
- Trung bình nhân các số dương, số âm, số chẵn, số lẻ

62/- (*) Khoảng cách trung bình giữa các giá trị trong mảng

4.5.1.8. Kỹ thuật thêm

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

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

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

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

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

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

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

4.5.1.9. Kỹ thuật xóa

70/- Xóa phần tử có chỉ số k trong mảng một chiều nguyên.
- ✍ *Mở rộng*: chỉ số là số chẵn, số lẻ, số nguyên tố, bội số của m, …

71/- Xóa tất cả phần tử có giá trị bằng X.
- ✍ *Mở rộng*: giá trị nhỏ hơn / lớn hơn X, số chẵn, số lẻ, số âm, giá trị là số nguyên tố …...

72/- Xóa tất cả các số (dương) lớn nhất trong mảng một chiều nguyên.
- ✍ *Mở rộng*: cho các giá trị âm lớn nhất, chẵn lớn nhất, lẻ lớn nhất, nguyên tố lớn nhất, bội số lớn nhất (hay ước số lớn nhất) của số k, …

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

74/- Xoá phần tử tại vị trí lẻ trong mảng.

75/- Nhập vào giá trị X. Viết hàm xoá phần tử có giá trị gần X nhất.

4.5.1.10. Kỹ thuật xử lý mảng

76/- Đảo ngược thứ tự mảng một chiều các số nguyên.

77/- Đảo ngược thứ tự các giá trị chẵn trong mảng một chiều nguyên.

78/- Chuyển các phần tử có giá trị chẵn về đầu mảng.

79/- Chuyển các phần tử có giá trị âm về cuối mảng.

80/- Dịch trái xoay vòng 1 lần trên mảng một chiều.
- ✍ *Mở rộng*:
  - Dịch phải xoay vòng 1 lần trên mảng một chiều.
  - Dịch phải/trái k lần; …

4.5.1.11. Sắp xếp mảng

81/- Sắp xếp bằng phương pháp đổi chỗ trực tiếp.

82/- Sắp xếp sao cho số âm giảm dần, số dương tăng dần.

83/- Sắp xếp các phần tử chẵn nằm bên trái theo thứ tự tăng dần còn các phần tử lẻ bên phải theo thứ tự giảm dần.

84/- Sắp xếp các phần tử âm giảm dần từ trái sang phải, phần tử dương tăng dần từ phải sang trái.

85/- Sắp xếp sao cho số lẻ tăng dần, số dương giữ nguyên vị trí.
- ✍ *Mở rộng*: Sắp xếp mảng theo thứ tự tăng dần của các phần tử là số nguyên tố, số chính phương (các phần tử khác giữ nguyên vị trí)

4.5.1.12. Kỹ thuật mảng con

86/- Liệt kê các mảng con (> 2 phần tử) tăng trong mảng một chiều.
- ✍ *Mở rộng*:
  - Liệt kê các mảng con toàn dương.
  - Liệt kê mảng con tăng có tổng giá trị các phần tử là lớn nhất.
  - Liệt kê mảng con tăng có số lượng phần tử nhiều nhất (dài nhất). Nếu có 2 dãy con dài bằng nhau thì xuất mảng con đầu tiên.

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

Mảng con dài nhất là: 1 2 6 8

📖 *Hướng dẫn:*

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

Nếu còn tăng và chưa hết mảng thì (CuoiMoi = vị trí đang xét và DemMoi tăng 1)
Ngược lại thì so sánh:
Nếu DemMoi > DemMax
thì (DauMax = DauMoi và CuoiMax = CuoiMoi).

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

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

Ví dụ: *1 2 3 4 2 3 4 5 6 4 5 6 => TB = 15.*

4.5.1.13. Tổng hợp

89/- Nhập vào một mảng số thực kết thúc bằng việc nhập số 0 (số 0 không lưu vào mảng) hoặc khi đã nhập đủ 20 số. Kiểm tra có hay không các tính chất sau đây của mảng:

a. Mảng đơn nhất? (Không có phần tử trùng nhau trong mảng)

b. Mảng đan dấu? (2 phần tử kề nhau phải khác đấu. Mảng 1 phần tử xem như đan dấu)

c. Mảng tuần hoàn? (Mảng tuần hoàn: Nếu A_i, A_{i+1}, A_{i+2} là 3 phần tử liên tiếp trong mảng thì: $A_{i+1} \geq A_i$ và $A_{i+1} \geq A_{i+2}$ hoặc $A_{i+1} \leq A_i$ và $A_{i+1} \leq A_{i+2}$. Mảng có 2 phần tử xem như tuần hoàn).

d. Mảng tăng dần? Mảng giảm dần?

90/- Nhập vào một mảng số nguyên gồm n phần tử. Tạo menu và thực hiện các thao tác trên mảng:

a. Xuất mảng ra màn hình.
b. Đếm số phần tử là bội số của 3 của mảng (đếm số phần tử có điều kiện).
c. Tìm phần tử lớn nhất, nhỏ nhất và tính tổng, tích các phần tử của mảng.
d. Thêm vào mảng phần tử x tại vị trí k (k<n).
e. Xóa phần tử tại vị trí k (k<n).
f. Tìm phần tử x trong mảng, chỉ ra vị trí xuất hiện của x.
g. Tìm cặp phần tử có tổng bình phương đúng bằng k (k nhập từ bàn phím).
h. Sắp xếp mảng tăng dần.
i. Sắp xếp các phần tử ở vị trí chẵn tăng dần, vị trí lẻ giảm dần.

91/- Nhập mảng số nguyên a có n phần tử (0< n <=20). Tạo menu để thực hiện các công việc:

a. Sắp các phần tử vị trí lẻ tăng dần, các phần tử vị trí chẵn giảm dần
b. Sắp các phần tử dương về đầu mảng có thứ tự giảm dần, các phần tử âm cuối mảng có thứ tự tăng dần.
c. Sắp các số nguyên tố về đầu mảng có thứ tự tăng dần, các phần tử còn lại có thứ tự giảm dần.

92/- Viết hàm liệt kê các bộ 4 số a, b, c, d trong mảng các số nguyên (có ít nhất 4 phần tử và đôi một khác nhau) sao cho a + b = c + d.

93/- (*) Cho mảng các số nguyên a gồm n phần tử (n<=30000) và số dương k (k<=n). Hãy chỉ ra số hạng lớn thứ k của mảng.

Ví dụ: Mảng a: 6 3 1 10 11 18
k = 3
Kết quả: 10

94/- Viết chương trình tính tổng tất cả các phần tử xung quanh trên mảng các số nguyên. Biết rằng *Phần tử xung quanh là hai phần tử bên cạnh cộng lai bằng chính nó; v*í dụ: 1 3 2 ➔ 1,2 là hai phần tử xung quanh của 3).

Ví dụ: *1 **3** 2 **5** 3 **9** 6 ➔ tổng 17*

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

96/- Viết hàm tìm và xóa tất cả các phần tử trùng với x trong mảng một chiều các số nguyên, nếu không tồn tại phần tử x trong mảng thì trả về -1.

97/- (**) Viết hàm xoá những phần tử sao cho mảng kết quả có thứ tự tăng dần và số lần xoá là ít nhất.

98/- Cho mảng a gồm n số nguyên có thứ tự tăng dần. Nhập vào một phần tử nguyên X, viết hàm chèn X vào mảng sao cho mảng vẫn có thứ tự tăng dần (không sắp xếp).

99/- Viết chương trình tìm số lẻ nhỏ nhất lớn hơn mọi số chẵn có trong mảng.

100/- Viết hàm tìm giá trị chẵn nhỏ nhất nhỏ hơn mọi giá trị lẻ trong mảng các số nguyên.

101/- Viết hàm tìm phần tử xuất hiện nhiều nhất trong mảng các số nguyên.

102/- Viết chương trình đếm và liệt kê các mảng con tăng dần trong mảng một chiều các số nguyên.

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

103/- Viết chương trình tìm mảng con tăng dần có tổng lớn nhất trong mảng một chiều.

104/- (*) Viết chương trình nhập vào một mảng số a gồm n số nguyên (n <= 100). Tìm và in ra mảng con tăng dài nhất

Ví dụ: Nhập mảng a: **1 2 3 6** 4 7 8 **3 4 5 6 7 8 9** 4 5
Mảng con tăng dài nhất: 3 4 5 6 7 8 9

105/- Viết chương trình nhập vào mảng số a gồm n số nguyên (n <= 100).

a. Hãy đảo ngược mảng đó.

Ví dụ: *Nhập a: 3 4 5 2 0 4 1*
Mảng sau khi đảo: 1 4 0 2 5 4 3

b. (*) Hãy kiểm tra xem mảng đã cho có thứ tự chưa (mảng được gọi là thứ tự khi là mảng tăng hoặc mảng giảm).

106/- Cho mảng A có n phần tử hãy cho biết mảng này có đối xứng hay không.

107/- Cho mảng A có n phần tử. Nhập vào số nguyên k (k >= 0), dịch phải xoay vòng mảng A k lần.

Ví dụ: *Mảng A: 5 7 2 3 1 9*
Nhập k = 2
Dịch phải xoay vòng mảng A:1 9 5 7 2 3

108/-(**) Viết chương trình in ra tam giác Pascal (dùng mảng một chiều).

109/- Tìm giá trị trong mảng các số thực " xa giá trị x nhất"

Ví dụ:

24	45	23	13	43	-12

Giá trị x: 15

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

9	30	8	2	28	27

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

110/- Tìm một vị trí trong mảng một chiều các số thực mà tại vị trí đó là giá trị "gần giá trị x nhất ".

Ví dụ:

24	45	23	13	43	-12

Giá trị x: 15

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

9	30	8	2	28	27

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

4.5.2. *Thao tác trên nhiều mảng*

4.5.2.1. Nhập xuất mảng

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

- *Mở rộng*: cho số nguyên tố, số hoàn thiện, số chính phương, …

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

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

B[i] = (2*A[i] -1)nếu A_i>0
B[i] = (-2*A[i] +1) nếu A_i<0
B[i] = 1 nếu A[i]=0

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

4.5.2.2. Nhập / tách mảng

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

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

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

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

Ví dụ: Mảng a : 3 2 7 5 9
Mảng b : 1 8 10 4 12 6
Mảng c : **2 8 10 4 12 6** 3 7 5 9 1

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

a. Sắp xếp hai mảng A, B theo thứ tự tăng dần.
b. Trộn xen kẻ 2 mảng trên thành mảng c sao cho mảng c cũng có thứ tự tăng dần.
c. Sắp xếp lại để có mảng B giảm dần (mảng A vẫn tăng dần). Trộn 2 mảng A và B thành mảng C tăng dần.

4.5.2.3. Đếm – liệt kê

119/- Cho 2 mảng A, B. Đếm và liệt kê các phần tử chỉ xuất hiện 1 trong 2 mảng.

120/- Cho 2 mảng A, B. Đếm và liệt kê các phần tử chỉ xuất hiện trong mảng A nhưng không xuất hiện trong mảng B.

121/- Cho 2 mảng A, B. Đếm phần tử xuất hiện trong cả 2 mảng.

4.5.2.4. Khác

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

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

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

Ví dụ: với n=12712851. Sẽ xuất ra màn hình: *số 1 xuất hiện 3 lần*
số 2 xuất hiện 2 lần
số 5 xuất hiện 1 lần
số 7 xuất hiện 1 lần
số 8 xuất hiện 1 lần

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

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

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

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

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

Ví dụ1: **123**A45**1**8B**23** ➔ có các mảng con là123A451, 23A4518B2, 3A4518B23
Ví dụ2: **123**A45**3**8B**21** ➔ có các mảng con là 123A4538B21, 23A4518B2, 3A453
Ví dụ3: 123456789 ➔ in ra "Khong co day con".

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

Ví dụ: A = 52568, B = 462384 -> C = 54625682384, D = 45256236884.

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

Ví dụ: Mảng: 5 6 11 4 4 5 4
So lon nhat la 11.
So 11 xuat hien 1 lan

✍ *Mở rộng*:
- Đếm số lần xuất hiện của giá trị nhỏ nhất, dương nhỏ nhất, âm lớn nhất.
- Đếm số lần xuất hiện của số nguyên tố nhỏ nhất, …).

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

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

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

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

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

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

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

4.5.3. *Stack & Queue*

4.5.3.1. stack

1. Viết chương trình sử dụng stack để đổi giá trị của 1 số thực không âm từ cơ số 10 sang cơ số 2, 8, 16 theo minh họa sau:

```
Nhập giá trị hệ thập phân: X
Kết quả xuất ra màn hình:
Giá trị tương ứng của X ở hệ 2 là: Y
Giá trị tương ứng của X ở hệ 8 là: Z
Giá trị tương ứng của X ở hệ 16 là: W
```

Trong đó
- **X** là giá trị do người dùng nhập vào.
- **Y**, **Z**, **W** là kết quả thực hiện của chương trình.

2. Viết chương trình sử dụng stack để đổi giá trị của 1 số thực không âm từ cơ số 2 (hoặc 8, 16) sang cơ số 10 theo minh họa sau:

```
Nhập giá trị: X
Cơ số của số vừa nhập: Y
```

Kết quả xuất ra màn hình:

```
Giá trị tương ứng của X ở hệ 10 là: Z
```

Trong đó

- **X, Y** là giá trị do người dùng nhập vào dưới dạng số nguyên có kiểu là **unsigned long**. Y chỉ nhận một trong 3 giá trị 2 | 8 | 16.
- **Z**, **W** là kết quả thực hiện của chương trình.

4.5.3.2. Queue

3. Tương tự như bài tập 1, nhưng yêu cầu chỉ sử dụng cấu trúc dữ liệu là queue để đổi giá trị của 1 số thực không âm từ cơ số 10 sang cơ số 2, 8, 16 theo minh họa sau:

```
Nhập giá trị hệ thập phân: X
Kết quả xuất ra màn hình:
Giá trị tương ứng của X ở hệ 2 là: Y
Giá trị tương ứng của X ở hệ 8 là: Z
Giá trị tương ứng của X ở hệ 16 là: W
```

Trong đó

- **X** là giá trị do người dùng nhập vào.
- **Y**, **Z**, **W** là kết quả thực hiện của chương trình.

4. Tương tự như bài tập 2, nhưng yêu cầu chỉ sử dụng cấu trúc dữ liệu là queue để đổi giá trị của 1 số thực không âm từ cơ số 2 (hoặc 8, 16) sang cơ số 10 theo minh họa sau:

```
Nhập giá trị: X
Cơ số của số vừa nhập: Y
```

Kết quả xuất ra màn hình:

```
Giá trị tương ứng của X ở hệ 10 là: Z
```

Trong đó

- **X, Y** là giá trị do người dùng nhập vào dưới dạng số nguyên có kiểu là **unsigned long**. Y chỉ nhận một trong 3 giá trị 2 | 8 | 16.
- **Z**, **W** là kết quả thực hiện của chương trình.

MẢNG HAI CHIỀU

(Two Dimensional Array)

5.1. KHÁI NIỆM

- Ma trận là một tập hợp các biến có cùng kiểu và cùng tên. Khi đó việc truy xuất đến một phần tử trong ma trận được thực hiện thông qua hai biến chỉ số dòng và chỉ số cột.
- Mảng hai chiều thực chất là mảng một chiều trong đó mỗi phần tử của mảng là một mảng một chiều, và được truy xuất bởi hai chỉ số dòng và cột.
  Ví dụ: với số cột m=4 và số dòng n=3, chỉ số được đánh số như sau

Dòng \ Cột	***0***	***1***	***2***	***3***
0	8	16	9	92
1	1	15	4	3
2	12	17	6	2

- Mảng nhiều chiều: là mảng có từ hai chiều trở lên.

5.2. KHAI BÁO

Từ khái niệm trên ta có cú pháp khai báo mảng hai chiều như sau:

5.2.1. *Cách 1: Con trỏ hằng*

- Cú pháp:

```
<Kiểu dữ liệu> <Tên mảng> [<Số dòng tối đa>][<Số cột tối đa> ];
```

Trong đó: **[<Số dòng tối đa>]** và **[<Số cột tối đa>]**: là một số nguyên dương chỉ số lượng phần tử tối đa có thể có trên một chiều trong mảng.

- Ví dụ:

```
int A[10][10];//Khai báo mảng 2 chiều kiểu int gồm 10 dòng, 10 cột
float b[5][20];//Khai báo mảng 2 chiều kiểu float gồm 5 dòng, 20 cột
```

- Cũng như mảng một chiều mảng hai chiều cũng có thể khởi tạo khi khai báo.

Ví dụ:
```
int  A[2][3]= {  1, 3, 5,
                 4, 7, 6};
```

Việc xuống dòng trong ví dụ chỉ nhằm mục đích dễ hình dung ma trận ban đầu. Trong thực tế ta có thể nhập các giá trị trên cùng 1 dòng như sau:

```
int  A[2][3]= { 1, 3, 5, 4, 7, 6};
```

Giá trị khởi tạo cho mảng hai chiều được thực hiện theo từng dòng từ trên xuống dưới và từ trái sang phải. Tất cả các các giá trị khởi tạo được đặt trong {}. Với cách khởi tạo mảng A như trên thì giá trị các phần tử là:

```
A[0][0]=1,   A[0][1]=3,    A[0][2]=5
A[1][0]=4,   A[1][1]=7,    A[1][2]=6
```

5.2.2. *Cách 2: Con trỏ*

- Cú pháp:

```
<Kiểu dữ liệu> **<Tên mảng>;
```

Ví dụ:
```
int **A;//Khai báo mảng động 2 chiều kiểu int
```

```
float **B ;//Khai báo mảng động 2 chiều kiểu float
```

- Tương tự như mảng một chiều, để sử dụng ta phải cấp phát vùng nhớ cho mảng 2 chiều bằng malloc hoặc calloc và sau khi dùng cần hủy vùng nhớ đã cấp bằng lệnh free.

  Ví dụ: Khai báo mảng các số nguyên A có kích thước 5x6

```
int **A;
A = (int **) malloc(5);
for (int  i=0 ; i<5 ; i++)
    A[i]=(int *) malloc(6);
```

5.3. TRUY XUẤT PHẦN TỬ CỦA MẢNG HAI CHIỀU

- Truy xuất các thành phần của mảng hai chiều phải dựa vào chỉ số dòng và chỉ số cột.

tên mảng[tọa độ dòng][tọa độ cột]

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

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

	0	*1*	*2*	*3*
0	**2**	**3**	**9**	**4**
1	**5**	**6**	**7**	**6**
2	**2**	**9**	**4**	**7**

> *Lưu ý: Khi nhập liệu cho mảng hai chiều, nếu kiểu dữ liệu của mảng là:*
> - *Số nguyên : nhập liệu theo cách thông thường.*
> - *Số thực : phải thông qua biến trung gian.*

- Hàm nhập/xuất ma trận

  - Ví dụ 1: Hàm nhập ma trận các số nguyên

```
void TaoMangSoNguyen (int A[][MAX], int &d, int &c)
{
    printf("Nhap so dong: ");
    scanf("%d",&d);
    printf("Nhap so cot: ");
    scanf("%d",&c);
    for(int row=0; row<d; row++)
        for(int col=0; col<c; col++)
        {   printf("Nhap gia tri cho phan tu A[%d][%d]",row,col);
            scanf("%d",&A[row][col]);
        }
}
```

  - Ví dụ 2: Hàm nhập ma trận các số thực

```
void TaoMangSoThuc (float A[][MAX],int &d,int &c)
{   float temp;
    printf("Nhap so dong: ");
    scanf("%d",&d);
    printf("Nhap so cot: ");
    scanf("%d",&c);
    for(int row=0; row<d; row++)
        for(int col=0; col<c; col++)
        {   printf("Nhap gia tri cho phan tu A[%d][%d]",row,col);
            scanf("%d",&temp);
            A[row][col]=temp;
        }
}
```

- Ví dụ 3: Hàm xuất ma trận

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

5.4. MA TRẬN VUÔNG

5.4.1. *Khái niệm*

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

5.4.2. *Ma trận đơn vị*

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

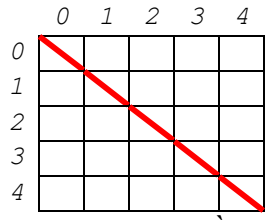
1	0	0
0	1	0
0	0	1

5.4.3. *Tính chất của ma trận vuông*

5.4.3.1. *Đường chéo*

5.4.3.1.1. *Đường chéo chính*

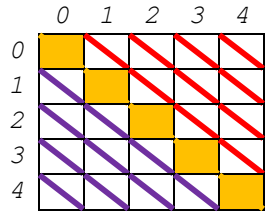
- Đường chéo chính là đường chéo nối các phần tử ở 2 góc đối đỉnh từ góc trái-trên đến góc phải-dưới.



- Đường chéo chính gồm tập hợp các phần tử có ***chỉ số dòng = chỉ số cột***
- Ví dụ: Cho ma trận vuông A(n x n). Xuất các giá trị có trên đường chéo chính:

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

- Đường chéo loại 1:

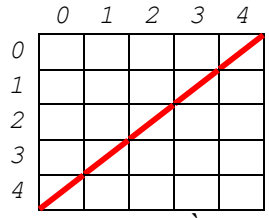


- Là tập hợp các đường chéo song song với đường chéo chính. Có 2(n-1) đường chéo loại 1.
- Truy xuất các phần tử trên cùng đường chéo loại 1: các phần tử nằm trên cùng đường chéo loại 1 sẽ có chung đặc tính là

chỉ số cột – chỉ số dòng = |hằng số|

5.4.3.1.2. Đường chéo phụ

- Đường chéo phụ là đường chéo nối các phần tử ở 2 góc đối đỉnh từ góc phải-trên đến góc trái-dưới.



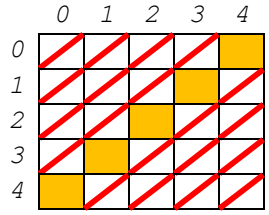
- Đường chéo phụ gồm tập hợp các phần tử có:

chỉ số cột + chỉ số dòng = số dòng (hoặc số cột) -1

- Ví dụ: Cho ma trận vuông A(n x n). Xuất các giá trị có trên đường chéo chính:

```
void XuatDuongCheoPhu(int A[][MAX], int n)
{
     for (int i=0 ; i<n ; i++)
          printf("%5d", A[i][n-i-1]);
}
```

- Đường chéo loại 2:



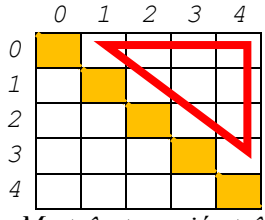
- Là tập hợp các đường chéo song song với đường chéo phụ. Có 2(n-1) đường chéo loại 2.
- Truy xuất các phần tử trên cùng đường chéo loại 2: các phần tử nằm trên cùng đường chéo loại 1 sẽ có chung đặc tính là

chỉ số cột + chỉ số dòng = |hằng số|

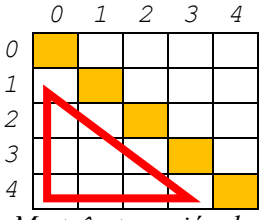
5.4.3.2. Tam giác tạo thành từ các đường chéo

5.4.3.2.1. Đối với đường chéo loại 1

- *Tam giác trên*: Truy xuất các phần tử thuộc tam giác trên của đường chéo chính: dựa vào tính chất: **cột > dòng**
- *Tam giác dưới*: Truy xuất các phần tử thuộc tam giác dưới của đường chéo chính dựa vào tính chất: **cột < dòng**

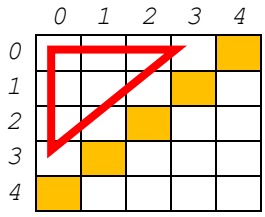


Ma trận tam giác trên đường chéo chính

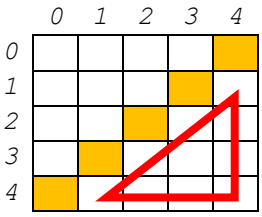


Ma trận tam giác dưới đường chéo chính

5.4.3.2.2. *Đối với đường chéo loại 2*



Ma trận tam giác trên đường chéo phụ



Ma trận tam giác dưới đường chéo phụ

- *Tam giác trên*: Truy xuất các phần tử thuộc tam giác trên của đường chéo phụ dựa vào tính chất:

```
chỉ số cột + chỉ số dòng < số dòng (hoặc số cột) -1
```

- *Tam giác dưới*: Truy xuất các phần tử thuộc tam giác dưới của đường chéo phụ dựa vào tính chất:

```
chỉ số cột + chỉ số dòng > số dòng (hoặc số cột) -1
```

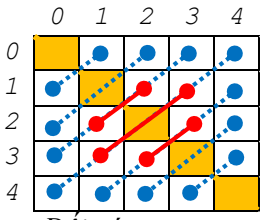
5.4.3.3. *Phần tử đối xứng qua đường cheó chính/phụ*

- *Đối với đường chéo chính*: các cặp phần tử đối xứng với nhau dựa vào tính chất:

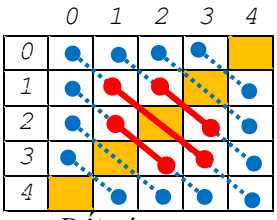
```
A[dòng][cột]   đối xứng với   A[cột][dòng]
```

- *Đối với đường chéo phụ*: các phần tử đối xứng với nhau dựa vào tính chất:

```
A[dòng][cột]   đối xứng với   A[n-1-cột][n-1-dòng]
```



Đối xứng qua đường chéo chính



Đối xứng qua đường chéo phụ

5.4.4. *Khai báo kiểu ma trận*

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

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

```
#define ROWS 100
#define COLS 200
typedef int MATRAN[ROWS][COLS];
```

- Ma trận vuông (số dòng và số cột bằng nhau)

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

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

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

5.5. THAM SỐ CỦA HÀM LÀ MẢNG HAI CHIỀU

Nguyên tắc truyền và nhận dữ liệu giữa hàm gọi và hàm được gọi cho mảng hai chiều cũng giống như hàm một chiều nghĩa là hàm được gọi sẽ tác động trực tiếp lên mảng truyền cho nó. Tuy nhiên việc khai báo có khác nhau.

Ví dụ: chương trình sau tạo mảng hai chiều với các giá trị là ngẫu nhiên rồi in ma trận lên màn hình.

//Cách 1: không sử dụng kiểu MATRAN

```
#define ROWS 200
#define COLS 200
void TaoMangNgauNhien(MATRAN a, int dong, int cot);
void XuatMaTran(MATRAN a, int dong, int cot);
int main()
{
    srand((unsigned int) time(NULL));
    int A[ROWS] [COLS];
    int dong, cot;
    printf("Nhap so dong cua ma tran: ");
    scanf("%d",&dong);
    printf("Nhap so cot cua ma tran: ");
    scanf("%d",&cot);
    TaoMangNgauNhien (A, dong, cot);
    printf("\n Ma tran vua duoc tao:\n");
    XuatMaTran(A, dong, cot);
    return 0;
}
void TaoMangNgauNhien (int A[][COLS], int dong, int cot)
{
    int i,j;
    for (i=0; i<dong; i++)
        for (j=0;j<cot;j++)
            A[i][j]=rand()%100;
}
void XuatMang(int A[][COLS], int dong, int cot)
{   int i,j;
    for (i=0; i<dong; i++)
    {
        for(j=0;j<cot;j++)
            printf("%5d",A[i][j]);
        printf("\n");
    }
}
```

//Cách 2: khai báo và sử dụng kiểu MATRAN⇒ sử dụng kiểu MATRAN cho tham số

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

```
                A[i][j]=rand()%100;
    }
    void XuatMaTran(MATRAN A, int dong, int cot)
    {   int i,j;
        for (i=0; i<dong; i++)
        {   for(j=0;j<cot;j++)
                printf("%5d",A[i][j]);
            printf("\n");
        }
    }
```

5.6. MỘT SỐ KỸ THUẬT CƠ BẢN TRÊN MẢNG HAI CHIỀU

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

5.6.1. *Nhập / xuất ma trận*

- Nhập trực tiếp các phần tử của ma trận:

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

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

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

- Xuất các phần tử của ma trận:

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

5.6.2. *Tính tổng*

- Viết hàm tính tổng các phần tử trong ma trận.

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

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

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

5.6.3. *Kỹ thuật đặt cờ hiệu*

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

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

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

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

5.6.4. *Kỹ thuật đặt lính canh*

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

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

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

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

5.6.5. *Sắp xếp*

- Viết hàm sắp xếp ma trận tăng dần từ trên xuống dưới và từ trái sang phải
  - *Không dùng mảng phụ*:

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

  - *Có dùng mảng phụ:*

```
void SapXep(float A[MAX][], int dong, int cot)
{   float b[1000];
    int k,i,j;
    // copy ma trận ra mảng một chiều b
    k=0;
    for (i=0;i<dong;i++)
        for(j=o,j<cot;j++)
            b[k++] = A[i][j];
    // Sắp xếp mảng một chiều b
    for (i=0; i<= k-2; i++)
        for(j=0;j<=k-1;j++)
            if (b[i] > b[j])
            {
                float temp = b[i];
                b[i] = b[j];
                b[j] = temp;
            }
    // copy ma trận ra mảng một chiều b
    k=0;
    for (i=0;i<dong;i++)
        for(j=o,j<cot;j++)
            A[i][j]= b[k++] ;
            k = k + 1;
}
```

5.6.6. *Đếm*

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

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

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

```
int DemNguyenTo(int A[][100], int dong, int cot)
{   int dem = 0;
    for (int i=0; i<dong; i++)
        for(int j=0; j<cot; j++)
```

```
                    if (LaSNT(A[i][j]==1)
                        dem ++ ;
            return dem;
        }
```

5.7. BÀI TẬP

5.7.1. *Bài tập nhập xuất*

1.- Viết chương trình khởi tạo giá trị các phần tử là ngẫu nhiên cho ma trận các số nguyên kích thước n X m.

2.- Viết hàm nhập ma trận các số nguyên dương (nhập sai báo lỗi và cho nhập lại).

3.- Viết hàm tạo ma trận a các số nguyên gồm n dòng m cột. Trong đó phần tử a[i] [j] = i * j.

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

5.- Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột (5 < n,m < 200), sao cho giá trị của các phần tử trong khoảng từ 0 đến 99.

6.- Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột (5 <= n,m < 100), sao cho giá trị của các phần tử phải toàn là số lẻ.

- *Mở rộng* cho các trường hợp số chẵn, số vừa âm vừa lẻ, …

7.- Viết chương trình phát sinh ngẫu nhiên ma trận vuông A cạnh n (với 5<n<50) các số nguyên sao cho giá trị phát sinh ngẫu nhiên phải toàn là số nguyên tố.

- *Mở rộng* cho các trường hợp số hoàn thiện, số hoàn hảo, …).

8.- Viết hàm phát sinh ngẫu nhiên ma trận các số nguyên dương gồm n dòng và m cột (5 <= n,m < 100), sao cho giá trị của các phần tử trong khoảng từ -100 đến 100.

9.- Viết hàm phát sinh ngẫu nhiên ma trận vuông (n X n) các số nguyên dương, sao cho số phát sinh sau phải lớn hơn hay bằng số phát sinh liền trước đó (các phần tử của ma trận được sắp xếp tăng dần từ trái sang phải và từ trên xuống dưới

- *Mở rộng* cho trường hợp số giảm dần.

10.- Gỉa sử gọi vị trí của 1 phần tử trong ma trận = chỉ số dòng + chỉ số cột của phần tử đó. Viết hàm phát sinh ngẫu nhiên ma trận vuông (n X n) các số nguyên dương, sao cho những vị trí là số lẻ chỉ nhận giá trị nhập vào là số lẻ, và những vị trí là số chẵn chỉ nhận giá trị nhập vào là số chẵn.

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

11.- Cho ma trận các số thực A(m x n). Hãy xây dựng ma trận B (m x n) từ ma trận A sao cho B[i][j] = abs (A[i][j]).

12.- Cho ma trận các số thực A(m x n). Hãy xây dựng ma trận B (m x n) từ ma trận A sao cho B[i][j] = lớn nhất của dòng i và cột j trong ma trận A.

13.- Cho ma trận các số thực A(mxn). Hãy xây dựng ma trận B (mxn) từ ma trận A sao cho B[i][j] = số lượng phần tử dương xung quanh (A[i][j]) trong ma trận A (B[i][j] tốI đa là 8 và nhỏ nhất là 0).

14.- (*) Xây dựng ma phương bậc n. Một ma trận được gọi là ma phương khi tổng các phần tử trên các dòng, các cột và hai đường chéo chính + phụ đều bằng nhau.

5.7.2. *Xuất*

15.- Cho ma trận vuông (n X n) các số nguyên. Viết hàm in ra:

15.1.- Các phần tử có giá trị là số lẻ sẽ được in giá trị, ngược lại các phần tử có giá trị là số chẵn sẽ được in thay thế bằng ký tự "X".

15.2.- Các phần tử nằm trên 2 đường chéo chính (các phần tử khác in khoảng trắng).

15.3.- Các phần tử nằm trên đường chéo phụ (các phần tử trên 2 đường chéo chính in khoảng trắng).

16.- Xuất theo hình cho trước:

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

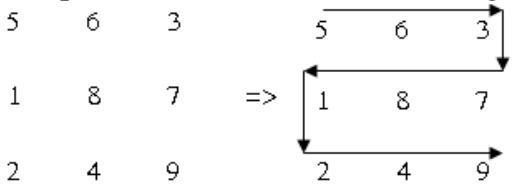
16.1.- In các giá trị các phần tử của ma trận theo thứ tự của đường chéo song song với đường chéo phụ của ma trận.

Ví dụ: Ma trận nhập vào:

Kết quả in ra: 7.0; 8.5;12; 9.5; 5.5; 8.0; 10; 6.2; 3.3; 9.5; 15; 8.0; 1.0; 12; 9; 13;

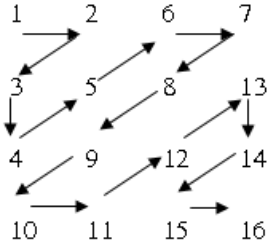
16.2.- In các giá trị các phần tử của ma trận theo đường ziczắc ngang.

Ví dụ:



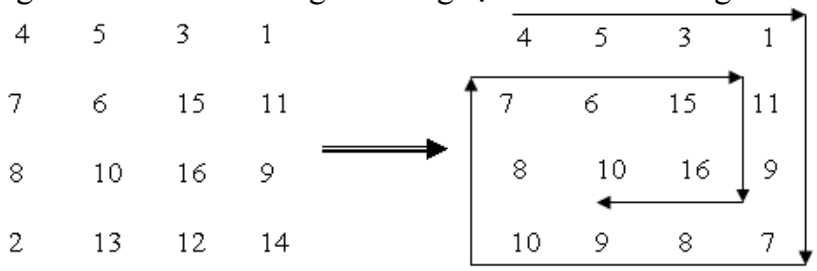
16.3.- In các giá trị các phần tử của ma trận theo đường ziczắc chéo

Ví dụ:



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

Ví dụ:



5.7.3. *Kỹ thuật tính toán*

5.7.3.1. Tính tổng/tích trên toàn bộ các phần tử có trong ma trận

17.- Tính tổng các số dương trong ma trận các số thực.

18.- Tính tích các giá trị lẻ trong ma trận các số nguyên.

19.- Tính tổng các giá trị trên một dòng trong ma trận các số thực.

20.- Tính tích các giá trị dương trên một cột trong ma trận các số thực.

21.- Tính tổng các giá trị dương trên một dòng trong ma trận các số thực.

22.- Tính tích các số chẵn trên một cột trong ma trận các số nguyên.

23.- Tính trung bình cộng các số dương trong ma trận các số thực.

24.- Tính tổng các giá trị nằm trên biên của ma trận.

25.- Tính trung bình nhân các số dương trong ma trận các số thực.

26.- Hãy biến đổi ma trận bằng cách thay các giá trị âm bằng giá trị tuyệt đối của nó.

27.- Hãy biến đổi ma trận bằng cách thay các giá trị bằng giá trị nguyên gần nó nhất.

28.- Tính tổng các số hoàn thiện trong ma trận các số nguyên.

5.7.3.2. Thao tác trên dòng có trong ma trận

29.- Tính tổng tất cả các số trên mỗi dòng. Sau đó in ra giá trị tổng dòng lớn nhất.

30.- Tính tổng các số chẵn trên mỗi dòng. Sau đó in ra giá trị tổng các số chẵn lớn nhất.

31.- Tính tổng các số âm (<0) trên mỗi dòng. Sau đó in ra giá trị tổng âm lớn nhất.

32.- Tính giá trị trung bình cộng trên mỗi dòng. Sau đó in ra giá trị tổng trung bình lớn nhất.

33.- Tính giá trị trung bình nhân trên mỗi dòng. Sau đó in ra giá trị trung bình nhân lớn nhất.

34.- Tính tổng các số nguyên tố trên mỗi dòng. Nếu dòng không có số nguyên tố, xem như tổng của dòng đó là 0.

35.- (*)In ra số nguyên tố lớn nhất trên mỗi dòng. Nếu dòng không có số nguyên tố, xem như số nguyên tố lớn nhất của dòng đó là 0.

Từ bài 29 đến 35 thực hiện in ra màn hình theo dạng:

Ma trận ban đầu										
4	2	9	7	Kết quả in ra màn hình	4	2	9	7	Tổng dòng =	22
2	2	9	4		2	2	9	4	Tổng dòng =	17
7	8	3	4		7	8	3	4	Tổng dòng =	22
4	3	5	6		4	3	5	6	Tổng dòng =	18
					Tổng dòng lớn nhất là 22, dong co tong lon nhat la dong 0 va dong 2					

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

- Dãy có ít nhất 3 số.
- Giá trị của số đi trước phải nhỏ hơn hay bằng số đi sau liền kề.
- In ra các dãy tăng có trên mỗi dòng theo dạng của ví dụ sau:

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

Ma trận ban đầu

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

Kết quả in ra màn hình

5.7.3.3. Thao tác trên cột có trong ma trận

Thực hiện tương tự như bài tập trong phần 5.7.3.2 nhưng thay dòng thành cột.

5.7.3.4. Thao tác trên tam giác/đường chéo của ma trận vuông

37.- Tính tổng các phần tử thuộc ma trận tam giác trên (không tính đường chéo) trong ma trận vuông các số thực

38.- Tính tổng các phần tử thuộc ma trận tam giác dưới (không tính đường chéo) trong ma trận vuông các số thực

39.- Tính tổng các phần tử trên đường chéo chính.

40.- Tính tổng các phần tử trên đường chéo phụ.

41.- Tính tổng các phần tử chẵn nằm trên biên của ma trận vuông các số nguyên

5.7.4. *Kỹ thuật tìm kiếm*

5.7.4.1. Tìm giá trị trong toàn ma trận

42.- Lớn nhất trong ma trận

43.- Phần tử chẵn dương và nhỏ nhất trong ma trận.

44.- Am lẻ lớn nhất

45.- Nhỏ nhất trong ma trận.

46.- Tìm số đầu tiên nhỏ hơn x (với x là tham số được truyền cho hàm).

47.- Số nguyên tố xuất hiện đầu tiên (tính theo chiều từ trên xuống dưới và từ trái sang phải).

48.- Số nguyên tố xuất hiện cuối cùng.

49.- Chẵn cuối cùng trong ma trận.

50.- Số hoàn thiện đầu tiên trong ma trận.

51.- Số hoàn thiện lớn nhất trong ma trận.

5.7.4.2. Tìm vị trí

Thực hiện các yêu cầu tương tự như bài tập trong phần 5.7.4.1. nhưng thay thế yêu cầu tìm giá trị bằng tìm vị trí (chỉ số).

5.7.4.3. Tìm trên phạm vi có trong yêu cầu

52.- Tìm giá trị X trên đường chéo chính.

53.- Tìm giá trị X trên đường chéo phụ.

54.- Tìm giá trị X trong tam giác trên của đường chéo chính.

55.- Tìm giá trị X trong tam giác dưới của đường chéo chính.

56.- Tìm giá trị X trong tam giác trên của đường chéo phụ.

57.- Tìm giá trị X trong tam giác dưới của đường chéo phụ.

5.7.4.4. Tìm kiếm khác

58.- Viết hàm tìm cột có tổng nhỏ nhất trong ma trận.

59.- Viết hàm tìm dòng có tổng lớn nhất trong ma trận.

5.7.5. *Kỹ thuật đếm*

60.- Viết hàm đếm số lượng số dương trong ma trận các số thực.

61.- Đếm số lượng số nguyên tố trong ma trận các số nguyên.

62.- Đếm tần suất xuất hiện của một giá trị x trong ma trận các số thực.

63.- Đếm số chữ số trong ma trận các số nguyên dương.

64.- Đếm số lượng số dương trên một hàng trong ma trận các số thực.

65.- Đếm số lượng số hoàn thiện trên một hàng trong ma trận các số nguyên.

66.- Đếm số lượng số âm trên một cột trong ma trận các số thực.

67.- Đếm số lượng số dương trên biên ma trận trong ma trận các số thực.

Ví dụ: cho ma trận

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

Số lượng giá trị dương nằm trên biên: 7

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

69.- (*) Đếm số lượng phần tử cực trị trong ma trận các số thực. Một phần tử được gọi là cực trị khi nó lớn hơn các phần tử xung quanh hoặc nhỏ hơn các phần tử xung quanh.

70.- (*) Đếm số lượng giá trị có trong ma trận các số thực. Lưu ý: Nếu có k phần tử (k>=1) trong ma trận bằng nhau thì ta chỉ tính là 1.

71.- (*) Tính tổng các phần tử cực trị trong ma trận các số thực. Một phần tử được gọi là cực trị khi nó lớn hơn các phần tử xung quanh hoặc nhỏ hơn các phần tử xung quanh.

72.- (*) Đếm số lượng giá trị "Hoàng Hậu" trên ma trận. Một phần tử được gọi là Hoàng Hậu khi nó lớn nhất trên dòng, trên cột và hai đường chéo đi qua nó.

73.- (*) Đếm số lượng giá trị "Yên Ngựa" trên ma trận. Một phần tử được gọi là Yên Ngựa khi nó lớn nhất trên dòng và nhỏ nhất trên cột.

74.- Đếm số lượng cặp giá trị đối xứng nhau qua đường chéo chính

75.- Đếm số lượng dòng giảm trong ma trận

76.- Đếm số lượng phần tử cực đại trong ma trận

77.- Đếm số lượng giá trị dương trên đường chéo chính trong ma trận vuông các số thực.

78.- Đếm số lượng giá trị âm trên đường chéo phụ trong ma trận vuông các số thực

79.- Đếm số lượng giá trị chẵn trong ma trận tam giác trên của ma trận vuông các số nguyên

5.7.6. *Kỹ thuật đặt cờ hiệu*

80.- Kiểm tra ma trận có tồn tại số dương hay không?

81.- Kiểm tra ma trận có tồn tại số hoàn thiện hay không?

82.- Kiểm tra ma trận có tồn tại số lẻ hay không?

83.- Kiểm tra ma trận có toàn dương hay không?

84.- Kiểm tra một hàng ma trận có tăng dần hay không?

85.- Kiểm tra một cột trong ma trận có giảm dần hay không?

86.- Kiểm tra các giá trị trong ma trận có giảm dần theo dòng và cột hay không?

Ví dụ: Ma trận sau đây giảm dần theo hàng và cột.

87	75	62	54

46	40	33	28
20	18	15	10

87.- Liệt kê các dòng toàn âm trong ma trận các số thực.

Ví dụ: Cho ma trận

0	*1*	*2*	*3*	*Chỉ số*
-87	-75	-62	-54	*0*
46	40	33	28	*1*
-20	-18	-15	-10	*2*

Các dòng toàn âm trong ma trận: 0,2.

88.- Liệt kê chỉ số các dòng chứa toàn giá trị chẵn trong ma trận các số nguyên.

89.- Liệt kê các dòng có chứa số nguyên tố trong ma trận các số nguyên.

90.- Liệt kê các dòng có chứa giá trị chẵn trong ma trận các số nguyên.

91.- Liệt kê các dòng có chứa giá trị âm trong ma trận các số thực.

92.- Liệt kê các cột trong ma trận các số nguyên có chứa số chính phương.

93.- Liệt kê các dòng trong ma trận các số thực thỏa mãn đồng thời các điều kiện sau: dòng có chứa giá trị âm, giá trị dương và giá trị 0 (phần tử chung hòa).

94.- Liệt kê các dòng giảm dần trong ma trận số thực.

95.- Liệt kê các dòng tăng thực trong ma trận số thực.

96.- Cho hai ma trận A và B. Kiểm tra ma trận A có là ma trận con của ma trận B hay không?

97.- Cho hai ma trận A và B. Đếm số lần xuất hiện của ma trận A trong ma trận B.

98.- Kiểm tra đường chéo chính có tăng dần từ trên xuống dưới hay không

99.- Kiểm tra ma trận có đối xứng qua đường chéo chính hay không

100.- Kiểm tra ma trận có đối xứng qua đường chéo phụ hay không

101.- Kiểm tra ma trận có phải là ma phương bậc n không ?

5.7.7. *Kỹ thuật đặt lính canh*

102.- Liệt kê các dòng có chứa giá trị lớn nhất của ma trận trong ma trận các số thực.

103.- Đếm số lượng giá trị nhỏ nhất trong ma trận các số thực.

104.- Đếm số lượng giá trị chẵn nhỏ nhất trong ma trận các số nguyên.

105.- Liệt kê các dòng có tổng dòng lớn nhất trong ma trận.

106.- Liệt kê các cột có tổng cột nhỏ nhất trong ma trận.

107.- Liệt kê các dòng có nhiều số chẵn nhất trong ma trận số nguyên.

108.- Liệt kê các dòng có nhiều số nguyên tố nhất trong ma trận số nguyên.

109.- Liệt kê các dòng có nhiều số hoàn thiện nhất trong ma trận số nguyên.

110.- (*) Liệt kê các cột nhiều chữ số nhất trong ma trận các số nguyên (lietkecot).

111.- (*)Cho ma trận vuông A (n x n) các số thực. Hãy tìm ma trận vuông B (k x k) sao cho tổng các giá trị trên ma trận vuông này là lớn nhất.

5.7.8. *Kỹ thuật hoán vị/thêm/xóa dòng/cột*

112.- Hoán vị hai dòng trên ma trận.

113.- Hoán vị hai cột trên ma trận.

114.- Dịch xuống xoay vòng các hàng trong ma trận.

Ví dụ:

Ma trận ban đầu

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

Kết quả dịch xuống xoay vòng các hàng lần 1

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

115.- Dịch lên xoay vòng các hàng trong ma trận.

116.- Dịch trái xoay vòng các cột trong ma trận.

Ví dụ:

Ma trận ban đầu

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

Kết quả dịch tráixoay vòng các cột lần 1

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

117.- Dịch phải xoay vòng các cột trong ma trận.

118.- Dịch phải xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên ma trận.

119.- Dịch trái xoay vòng theo chiều kim đồng hồ các giá trị nằm trên biên ma trận.

120.- Xóa một dòng trong ma trận.

Ví dụ:

Ma trận ban đầu

0	*1*	*2*	*3*
-87	-75	-62	-54
46	40	33	28
-20	-18	-15	-10
123	32	54	23

Chỉ số

0
1
2
3

Kết quả sau khi xóa dòng 1

0	*1*	*2*	*3*
-87	-75	-62	-54
-20	-18	-15	-10
123	32	54	23

121.- Xóa một cột trong ma trận.

122.- Xoay ma trận một góc 90 độ.

123.- Xoay ma trận một góc 180 độ.

124.- Xoay ma trận một góc 270 độ.

125.- Chiếu gương ma trận theo trục dọc.

126.- Chiếu gương ma trận theo trục ngang.

127.- Chiếu gương ma trận theo trục ngang theo 2 trường hợp:

- Chiếu nửa trên xuống nữa dưới (bỏ giá trị cũ của nửa dưới).
- Chiếu nửa dưới lên nửa trên (bỏ giá trị cũ của nửa trên).

128.- Chiếu gương ma trận theo trục dọc.

- Chiếu nửa trái sang nửa phải (bỏ giá trị cũ của nửa trái).
- Chiếu nửa phải sang nửa trái (bỏ giá trị cũ của nửa phải).

5.7.9. *Sắp xếp*

129.- Viết hàm sắp xếp các phần tử trên một dòng tăng dần từ trái sang phải.

130.- Viết hàm sắp xếp các phần tử trên một dòng giảm dần từ trái sang phải.

131.- Viết hàm sắp xếp các phần tử trên một cột tăng dần từ trên xuống dưới.

132.- Viết hàm sắp xếp các phần tử trên một cột giảm dần từ trên xuống dưới.

133.- Viết hàm xuất các giá trị chẵn trong ma trận các số nguyên theo thứ tự giảm dần.

134.- Viết hàm xuất các số nguyên tố trong ma trận các số nguyên ra màn hình theo thứ tự tăng dần.

135.- Viết hàm sắp xếp các phần tử trong ma trận theo yêu cầu sau:

- Dòng có chỉ số chẵn tăng dần.
- Dòng có chỉ số lẻ giảm dần.

136.- Viết hàm sắp xếp các phần tử trong ma trận theo yêu cầu sau:

- Cột có chỉ số chẵn giảm dần từ trên xuống.
- Cột có chỉ số lẻ tăng dần từ trên xuống.

137.- Sắp xếp các phần tử trong ma trận các số thực tăng dần theo hàng và cột bằng hai phương pháp sử dụng mảng phụ và không dùng mảng phụ.

138.- Sắp xếp các phần tử dương trong ma trận các số thực tăng dần theo hàng và cột bằng phương pháp sử dụng mảng phụ và không dùng mảng phụ (các số âm giữ nguyên giá trị và vị trí).

139.- Sắp xếp các phần tử chẵn trong ma trận các số nguyên giảm dần theo hàng và cột bằng hai phương pháp sử dụng mảng phụ và không dùng mảng phụ.

140.- Sắp xếp các giá trị âm của ma trận tăng dần, các giá trị dương giảm và các phần tử có giá trị bằng 0 giữ nguyên vị trí.

141.- Sắp xếp các giá trị chẵn của ma trận số nguyên tăng dần, các giá trị lẻ giảm dần.

142.- Sắp xếp các giá trị dương nằm trên biên ma trận tăng dần theo chiều kim đồng hồ.

143.- Sắp xếp các giá trị dương nằm trên biên ma trận các số thực tăng dần theo chiều kim đồng hồ.

144.- Hãy sắp xếp các dòng trong ma trận theo tiêu chuẩn sau: dòng có tổng dỏng nhỏ hơn nằm ở trên và dòng có tổng dòng lớn hơn bằng nằm ở dưới.

145.- Hãy sắp xếp các giá trị các phần tử trong ma trận tăng dần theo hình xoắn trôn ốc.

146.- Hãy sắp xếp các giá trị các phần tử trong ma trận tăng dần theo hình xoắn ziczac.

147.- Xuất các giá trị âm trong ma trận các số thực giảm dần (ma trận không thay đổi sau khi xuất)

148.- Sắp xếp các phần tử trên đường chéo chính tăng dần

149.- Sắp xếp các phần tử trên đường chéo phụ giảm dần

150.- Hoán vị hai dòng trong ma trận

151.- Hoán vị hai cột trong ma trận

152.- Sắp xếp "các dòng" tăng dần theo tổng dòng

153.- Đưa các giá trị chẵn về đầu ma trận vuông các số nguyên

154.- Cho ma trận vuông A(n x n) các số thực (n>=3). Sắp xếp các giá trị trong ma trận tam giác trên (không tính đường chéo) tăng dần từ trên xuống dưới và từ trái sang phải

155.- Cho ma trận vuông A(n x n) các số thực(n>=3).Sắp xếp các giá trị trong ma trận tam giác dưới (không tính đường chéo) giảm dần từ trên xuống dưới và từ trái sang phải

156.- (*) Xây dựng ma phương bậc A(nxn).Một ma trận được gọi là ma phương khi tổng các phần tử trên các dòng,các cột và hai đường chéo chính phụ đều bằng nhau.

5.7.10. *Xây dựng ma trận*

157.- Cho ma trận các số thực A(m x n). Hãy xây dựng ma trận B (m x n) từ ma trận A sao cho B[i][j] = abs (A[i][j]).

158.- Cho ma trận các số thực A(m x n). Hãy xây dựng ma trận B (m x n) từ ma trận A sao cho B[i][j] = lớn nhất của dòng i và cột j trong ma trận A.

159.- Cho ma trận các số thực A(m x n). Hãy xây dựng ma trận B (m x n) từ ma trận A sao cho B[i][j] = số lượng phần tử dương xung quanh (A[i][j]) trong ma trận A (B[i][j] tốI đa là 8 và nhỏ nhất là 0).

5.7.11. *Các phép toán trong ma trận*

160.- Tính tổng hai ma trận

161.- Tính hiệu hai ma trận

162.- Tính tích hai ma trận

163.- (*).Tìm ma trận nghịch đảo

164.- (*) Tính định thức của ma trận

165.- (**) Tạo ma phương bậc (n x n).

CẤU TRÚC (STRUCTURES)

6.1. KHÁI NIỆM

- Cấu trúc (struct) thực chất là một kiểu dữ liệu do người dùng định nghĩa bằng cách gom nhóm các kiểu dữ liệu cơ bản có sẵn trong C thành một kiểu dữ liệu phức hợp nhiều thành phần. Nhờ vậy ta có thể lưu trữ dữ liệu thuộc nhiều kiểu khác nhau vào trong cùng 1 biến.
- Một biến cấu trúc sẽ có các thành viên của cấu trúc còn gọi là trường (fields). Mỗi trường thuộc một kiểu dữ liệu nào đó.

6.2. ĐỊNH NGHĨA KIỂU DỮ LIỆU

6.2.1. *Cú pháp*

Cách 1

```
struct
{
    khai báo các trường
} danh sách tên biến cấu trúc;
```

Cách 2

```
struct <tên kiểu>
{
    khai báo các trường
};
```

6.2.2. *Nhận xét*

- *Cách 1*: nếu như trong các hàm sử dụng những biến cục bộ có cấu trúc giống nhau thì việc khai báo sẽ tốn nhiều thời gian.
- *Cách 2*: định nghĩa một kiểu cấu trúc nằm ngoài các hàm (kiểu dữ liệu toàn cục), trong các hàm chỉ việc khai báo biến có kiểu cấu trúc đã định nghĩa.

6.2.3. *Ví dụ 1*

Cần quản lý kiểu dữ liệu DATE gồm các thành phần

- *Thứ (thu)* : chuỗi có tối đa 4 ký tự.
- *Ngày (ngay)* : số nguyên 1 byte.
- *Tháng (thang)*: số nguyên 1 byte.
- *Năm (nam)* : số nguyên 2 bytes.

Ta định nghĩa kiểu dữ liệu DATE như sau:

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

6.2.4. *Định nghĩa một tên mới cho kiểu dữ liệu đã có bằng từ khoá typedef*

- **Cú pháp**

```
typedef struct < tên cấu trúc > < tên mới >;
```

- **Ví dụ**

```
typedef struct DATE d;
```

6.2.5. *Kiểu dữ liệu có cấu trúc có thể lồng vào nhau.*

Ví dụ 2: Định nghĩa kiểu dữ liệu của học sinh HOCSINH gồm:

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

Ta định nghĩa kiểu HOCSINH như sau:

```
struct DATE
{
     char           thu[5];
     unsigned char  ngay;
     unsigned char thang;
     int            nam;
};
typedef struct HOCSINH
{
     char             MSHS[6];
     char             hoten[31];
     struct DATE      ngaysinh;
     char             diachi[51];
     unsigned char    phai[4];
     float            diemtb;
};
```

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

6.3. KHAI BÁO

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

struct < tên cấu trúc > < tên biến > ;

Ví dụ:

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

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

Ví dụ: `DATE x ;  // Khai bao bien x co kieu DATE`

Biến con trỏ kiểu cấu trúc: Ngoài cách khai báo như trên ta có thể khai báo theo kiểu con trỏ như sau:

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

Để sử dụng ta cũng phải cấp phát vùng nhớ giống như kiểu dữ liệu chuẩn.

Ví dụ:

```
DATE   *y; //Khai bao con tro y kieu cau truc DATE
y = (DATE *) malloc ( sizeof (DATE));
```

6.4. TRUY XUẤT

- Truy xuất biến cấu trúc nghĩa là truy xuất các thành phần của biến này.
- Tùy vào kiểu của biến x mà ta có 2 trường hợp truy xuất như sau:
  - Biến x là một biến cấu trúc thông thường, ta dùng toán tử dấu chấm "**.**"

- **Cú pháp**

```
< Tên cấu trúc >.< Tên field >
```

- **Ví dụ**

```
DATE x ;          // khai bao bien x kieu DATE
x.ngay = 5 ;      // gan ngay bang 5
```

- Biến x là một biến con trỏ, ta dùng toán tử mũi tên "->" (Gồm dấu trừ '–' và dấu lớn hơn '>').
  - **Cú pháp**

```
< Tên cấu trúc > -> < Tên field >
```

  - **Ví dụ**

```
DATE *x ;         //khai bao bien x kieu con tro DATE
x -> ngay = 5 ;  // gan ngay bang 5
```

✍ *Đối với kiểu dữ liệu có struct lồng nhau phải truy cập đến thành phần cuối cùng có kiểu dữ liệu cơ bản.*

- **Ví dụ**: Giả sử, có kiểu HOCSINH như trên

```
HOCSINH hs;      // khai bao bien hs kieu HOCSINH
```

Muốn in học sinh A sinh vào tháng mấy ta phải truy cập như sau:

```
printf ("Thang sinh cua hoc sinh A la: %d", hs.ngaysinh.thang);
```

6.5. KHỞI TẠO GIÁ TRỊ CHO BIẾN CẤU TRÚC:

- Cách khởi tạo giá trị cho biến cấu trúc giống như khởi tạo cho mảng.
- Ví dụ 1: `DATE birth =(12,10,2000);`
- Ví dụ 2: Viết chương trình nhập vào toạ độ hai điểm trong mặt phẳng và tính

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

```
double KhoangCach2Diem (DIEM d1,DIEM d2)
{
   return sqrt(pow(d1.x - d2.x,2)+pow(d1.y - d2.y,2));
}
int main ()
{
   DIEM A , B;     //khai bao 2 diem A, B
   Nhap (A,'A');
   Xuat (A,'A');
   Nhap (B,'B');
   Xuat (B,'B');
   printf("\n Chieu dai canh AB: %5.2f",KhoangCach2Diem(A, B));
   return 0;
}
```

6.6. SỬ DỤNG CẤU TRÚC ĐỂ TRỪU TƯỢNG HÓA DỮ LIỆU

Phương pháp luận giải quyết bài toán bằng phương pháp trừu tượng hóa dữ liệu:

(sử dụng phương pháp lập trình hướng thủ tục hàm)

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

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

6.6.1. *Phân số*

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

```
struct PhanSo
{    int tu ;
     int mau ;
};
typedef struct PhanSo PHANSO;
void Nhap(PHANSO &x);
void Xuat(PHANSO x);
int TimUSCLN(int a,int b);
PHANSO Tong2PhanSo(PHANSO x, PHANSO y);
void Nhap(PHANSO &x)
{    printf("\nNhap tu: ");
     scanf("%d",&x.tu);
     printf("Nhap mau: ");
     scanf("%d",&x.mau);
}
void Xuat(PHANSO x)
{    printf("%d/%d",x.tu,x.mau);
}
```

```
int TimUSCLN(int a,int b)
{
     while (a!=b)
          if (a>b)
               a-=b;
          else
               b-=a;
     return a;
}
PHANSO Tong2PhanSo(PHANSO x, PHANSO y)
{
     PHANSO C;
     C.mau=x.mau*y.mau;
     C.tu= (x.tu*y.mau) + (y.tu*x.mau);
     int USC=TimUSCLN(abs(C.tu),abs(C.mau));
     C.tu=C.tu/USC;
     C.mau=C.mau/USC;
     return C;
}
int main ()
{
     PHANSO A,B,C;
     printf("Nhap phan so thu 1");
     Nhap(A);
     printf("Nhap phan so thu 2");
     Nhap(B);
     C=Tong2PhanSo(A,B);
     printf("Tong 2 phan so vua nhap la: ");
     Xuat(C)  ;
     return 0;
}
```

6.6.2. *Hỗn số*

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

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

- Định nghĩa hàm nhập

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

- Định nghĩa hàm xuất

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

6.6.3. *Điểm trong mặt phẳng Oxy*

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

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

- Định nghĩa hàm nhập

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

- Định nghĩa hàm xuất

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

6.6.4. *Điểm trong không gian Oxyz*

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

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

- Định nghĩa hàm nhập

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

- Định nghĩa hàm xuất

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

6.6.5. *Đơn thức*

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

```
struct DonThuc
{
    float a;
    int n; // số mũ của x
};
typedef struct DonThuc DONTHUC;
```

- Định nghĩa hàm nhập

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

- Định nghĩa hàm xuất

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

6.6.6. *Ngày*

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

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

- Định nghĩa hàm nhập ngày

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

- Định nghĩa hàm xuất ngày

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

6.6.7. *Đường thẳng trong mặt phẳng Oxy*

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

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

6.6.8. *Đường tròn trong mặt phẳng Oxy*

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

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

- Định nghĩa hàm nhập điểm

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

- Định nghĩa hàm nhập đường tròn

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

- Định nghĩa hàm xuất điểm

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

- Định nghĩa hàm xuất đường tròn

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

6.7. MẢNG CẤU TRÚC

6.7.1. *Khai báo*

Tương tự như mảng một chiều hay ma trận (Kiểu dữ liệu bây giờ là kiểu dữ liệu có cấu trúc

6.7.2. *Truy xuất phần tử trong mảng*

Tương tự như truy cập trên mảng một chiều hay ma trận. Nhưng do từng phần tử có kiểu cấu trúc nên phải chỉ định rõ cần lấy thành phần nào, tức là phải truy cập đến thành phần cuối cùng có kiểu là dữ liệu cơ bản.

6.7.3. *Nguyên tắc viết chương trình có mảng cấu trúc:*

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

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

6.8. THAM SỐ KIỂU CẤU TRÚC

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

```
#define MAX 100
struct ThiSinh
{
    char MaTS[7];
    char HoTen[31];
    float DiemLT;
    float DiemTH;
    float DiemTB;
};
typedef struct ThiSinh TS;
void NhapMotTS (TS &ts,int stt)
{
    printf("Nhap thong tin thi sinh thu %d:\n",stt+1);
    printf("Ma so (toi da 6 ky tu): ");
    gets(ts.MaTS);
    fflush(stdin);
    printf("Ho ten (toi da 30 ky tu): ");
    gets(ts.HoTen);
    fflush(stdin);
    printf("Nhap diem ly thuyet: ");
    scanf("%f",&ts.DiemLT);
    printf("Nhap diem thuc hanh: ");
    scanf("%f",&ts.DiemTH);
    ts.DiemTB=(ts.DiemLT+ts.DiemTH)/2;
    fflush(stdin);
}
```

```
void NhapToanBoTS (TS A[MAX],int &n)
{
     printf("Nhap so luong thi sinh: ");
     scanf("%d",&n);
     fflush(stdin);
     for(int i=0;i<n;i++)
          NhapMotTS(A[i],i);
}
void  XuatMotTS (TS A[MAX],int n, int vt)
{
     printf("%6s",A[vt].MaTS) ;
     printf("%30s",A[vt].HoTen);
     printf("%5.2f",A[vt].DiemLT);
     printf("%5.2f",A[vt].DiemTH);
     printf("%5.2f",A[vt].DiemTB);
     printf("\n");
}
void  XuatToanBoTS (TS A[MAX],int n)
{
     for(int i=0;i<n;i++)
          XuatMotThiSinh (A, n, i);
}
void SortDiemTBTang(TS A[MAX],int n)
{
     for(int i = 0 ; i < n - 1 ;  i++)
          for(int j = i + 1; j<n; j++)
               if(A[i].DiemTB<A[j].DiemTB)
               {
                    TS tam= A[i];
                    A[i]=A[j];
                    A[j]=tam;
               }
}
void TimTheoMa(TS A[MAX],int n)
{
     char Ma[7];
     printf("Nhap ma so can tim: ");
     gets(Ma);
     int i=0;
     while( (i<n) && (strcmp(A[i].MaTS,Ma)!=0))
          i++;
     if (i<n)
     {
          printf("Thong tin ve thi sinh co ma so %s:\n",Ma);
          XuatMotThiSinh (A, n, i);
     }
     else
          printf("KHONG TIM THAY");
}
```

6.9. BÀI TẬP

6.9.1. *Bài tập không có phần gợi ý*

1.- Viết chương trình nhập mảng 1 chiều các ngày. Tìm hai ngày gần nhau nhất và hai ngày xa nhau nhất.

2.- Viết chường trình nhập mảng 1 chiều các thời gian.tính ngày thơi gian gần nhau nhất và hai thời gian xa nhau nhất.

3.- Viết chương trình nhập mảng 1 chiều các tỉnh. Biết rằng 1 tỉnh gồm những thành phần sau:

- Mã tỉnh: kiểu số nguyên 2 byte.
- Tên tỉnh: chuỗi tối đa 30 ký tự.
- Dân số: kiểu số nguyên 4 byte.
- Diện tích: kiểu số thực.

Yêu cầu:

(i) Xuất các tỉnh có dân số lớn hơn 1.000.000.
(ii) Tìm 1 tỉnh có diện tích lớn nhất.
(iii) Sắp xếp các tỉnh giảm dần theo diện tích.

4.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các hộp sữa (HOPSUA). Biết rằng một hộp sữa gồm những thành phần sau:
  - Nhãn hiệu: chuỗi tối đa 20 ký tự.
  - Trọng lượng: kiểu số thực.
  - Hạn sử dụng: kiểu dữ liệu ngày (NGAY).

(ii) Xuất mảng.
(iii) Nhập vào một ngày x. Hãy đếm số lượng hộp sữa trong mảng quá hạn sử dụng so với ngày x.
(iv) Tìm trong một hộp sữa có trọng lượng lớn nhất trong mảng.

5.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các vé xem phim (VE). Biết rằng một vé xem phim gồm những thành phần sau:
  - Tên phim:chuỗi tối đa 20 ký tự.
  - Giá tiền: kiểu số nguyên 4 byte.
  - Xuất chiếu: kiểu thời gian (THOIGIAN).
  - Ngày xem: kiểu dữ liệu ngày (NGAY).

(ii) Xuất mảng.
(iii) Tính tổng giá tiền của tất cả các vé trong mảng.
(iv) Sắp xếp các phần tử trong mảng tăng dần theo ngày xem và xuất chiếu.

6.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các mặt hàng (MATHANG). Biết rằng một mặt hàng gồm những thành phần sau:
  - Tên mặt hàng: chuỗi tối da 20 ký tự.
  - Đơn giá: kiểu số nguyên 4 byte.
  - Số lượng tồn: kiểu số nguyên 4 byte.

(ii) Xuất mảng.
(iii) Tìm mặt hàng có tổng giá trị tồn là lớn nhất
(iv) Đếm số lượng mặt hàng có số lượng tồn lớn hơn 1.000.

7.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các chuyến bay. Biết rằng một chuyến bay gồm những thành phần sau:
- Mã chuyến bay: chuỗi tối đa 5 ký tự.
- Ngày bay: kiểu dữ liệu ngày.
- Giờ bay: kiểu thời gian.
- Nơi đi: chuỗi tối đa 20 ký tự.
- Nơi đến: chuỗi tối đa 20 ký tự.

(ii) Xuất mảng.

(iii) Tìm một ngày có nhiều chuyến bay nhất.

(iv) Tìm một chuyến bay trong mảng theo mã chuyến bay.

8.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các cầu thủ. Biết rằng một cầu thủ gồm những thành phần sau:
- Mã cầu thủ: chuỗi tối đa10 ký tự.
- Tên cầu thủ: chuỗi tối đa30 ký tự.
- Ngày sinh: kiểu dữ liệu ngày (NGAY).

(ii) Xuất mảng.

(iii) Liệt kê danh sách các cầu thủ nhỏ tuổi nhất trong mảng.

(iv) Sắp xếp các cầu thủ giãm dần theo ngày sinh

9.- Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các nhân viên (NHANVIEN). Biết rằng một nhân viên gồm những thành phần sau:
- Mã nhân viên: chuỗi tối đa 5 ký tự.
- Tên cầu thủ: chuỗi tối đa 30 ký tự.
- Lương nhân viên: kiểu số thực.

(ii) Xuất mảng.

(iii) Tìm nhân viên có lương cao nhất trong mảng.

(iv) Tính tổng lương của các hân viên.

(v) Sắp xếp mảng tăng dần theo lương nhân viên.

10.-Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các thí sinh (THISINH). Biết rằng một thí sinh gồm những thành phần sau:
- Mã thí sinh: chuỗi tối đa 5 ký tự.
- Tên thí sinh: chuỗi tối đa 30 ký tự.
- Điểm toán: kiểu số thực.
- Điểm lý: kiểu số thực.
- Điểm hóa: kiểu số thực.
- Điểm tổng cộng: kiểu số thực.

(ii) Xuất mảng.

(iii) Liệt kê các thí sinh đậu trong mảng. một thí sinh được gọi là đậu khi có tổng điểm 3 môn lớn hơn 15 và không có mô nào bị điểm không.

(iv) Sắp xếp theo thứ tự giãm dần theo điểm tổng cộng.

11.-Viết chương trình thực hiện những yêu cầu sau:

(i) Nhập mảng một chiều các thí sinh (THISINH). Biết rằng một thí sinh gồm những thành phần sau:
- Mã luận văn: chuỗi tối đa 10 ký tự.

- Tên luận văn: chuỗi tối đa 100 ký tự.
- Họ tên sinh viên thực hiện : chuỗi tối đa 30 ký tự
- Họ tên viên hướng dẫn : chuỗi tối đa 30 ký tự
- Năm thực hiện:kiểu số nguyên 2 byte.

(ii) Xuất mảng.
(iii) Tìm năm có nhiều luận văn nhất.
(iv) Liệt kê các luận văn thực hiện gần nhất.

12.-Viết chương trình thực hiện những yêu cầu sau:
(i) Nhập mảng một chiều các thí sinh (THISINH). Biết rằng một thí sinh gồm những thành phần sau:
- Tên thí sinh: chuỗi tối đa 30 ký tự.
- Điểm toán: kiểu số nguyên 2 byte.
- Điểm văn: kiểu số nguyên 2 byte.
- Điểm tổng cộng: kiểu số thực.

(ii) Xuất mảng.
(iii) Đếm số lượng sinh viên giỏi trong mảng (toán vá văn cùng lớn hơn 8).

13.-Viết chương trình thực hiện những yêu cầu sau:
(i) Nhập mảng một chiều các lớp học (LOPHOC). Biết rằng một lớp học gồm những thành phần sau:
- Tên lớp: chuỗi tối đa 30 ký tự.
- Sĩ số: kiểu số nguyên 2 byte.
- Danh sách các học sinh trong lớp: tối đa 50 học sinh.

(ii) Xuất mảng.
(iii) Tìm một lớp có sĩ số đông nhất.
(iv) Tìm một học sinh có điểm trung bình lớn nhất.

14.-Viết chương trình thực hiện những yêu cầu sau:
(i) Nhập mảng một chiều các sổ tiết kiệm (SOTIETKIEM). Biết rằng một sổ tiết kiệm gồm những thành phần sau:
- Mã sổ: chuỗi tối đa 5 ký tự.
- Loại tiết kiệm: chuỗi tối đa 10 ký tự.
- Họ tên khách hàng: chuỗi tối đa 30 ký tự
- Chứng minh nhân dân : kiểu số nguyên 4 byte.
- Ngày mở: kiểu dữ liệu ngày.
- Số tiền gởi: kiểu số thực.

(ii) Xuất mảng.

15.-Viết chương trình thực hiện những yêu cầu sau:
(i) Nhập mảng một chiều các đại lý (DAILY). Biết rằng một đại lý gồm những thành phần sau:
- Mã đại lý: chuỗi tối đa 5 ký tự.
- Tên đại lý: chuỗi tối đa 30 ký tự.
- Điện thoại:kiểu số nguyên 4 byte.
- Ngày tiếp nhận: kiểu dữ liệu ngày.
- Địa chỉ: chuỗi tối đa 50 ký tự
- E-mail: chuỗi tối đa 50 ký tự

(ii) Xuất mảng.
(iii) Tìm đại lý theo tên đại lý.

6.9.2. *Bài tập có gợi ý*

16.- Đơn thức

🕮 Các yêu cầu

- Tính tích hai đơn thức
- Tính đạo hàm cấp 1 đơn thức
- Tính thương hai đơn thức
- Tính đạo hàm cấp k đơn thức
- Tính giá trị đơn thức tạI vị trí $x=x_0$
- Định nghĩa toán tử tích(operation*) cho hai đơn thức.
- Định nghĩa toán tử thương(operation/) cho hai đơn thức

🕮 Các khai báo hàm

```
void nhap(DONTHUC &);
void xuat(DONTHUC );
DONTHUC tich(DONTHUC,DONTHUC );
DONTHUC thuong(DONTHUC,DONTHUC );
DONTHUC daoham(DONTHUC );
DONTHUC daoham(DONTHUC,int);
float tinhf(DONTHUC,float);
DONTHUC operation*(DONTHUC,DONTHUC );
DONTHUC operation/(DONTHUC,DONTHUC );
```

17.- Đa thức

🕮 Các yêu cầu

- Tính hiệu hai đa thức
- Tính tổng hai đa thức
- Tính tích hai đa thức
- Tính thương hai đa thức
- Tính đa thức dư của phép chia đa thức thứ nhất cho đa thức thứ hai
- Tính đạo hàm cấp 1 của đa thức
- Tính đạo hàm cấp k của đa thức
- Tính giá trị của đa thức tạI vị trí x = x0
- Định nghĩa toán tử cộng (operation+) cho hai đa thức
- Định nghĩa toán tử hiệu (operation-) cho hai đa thức
- Định nghĩa toán tử tích (operation*) cho hai đa thức
- Định nghĩa toán tử thương (operation/) cho hai đa thức
- Tìm nghiệm của đa thức trong đoạn [a,b] cho trước.

🕮 Các khai báo hàm

```
void nhap(DATHUC &);
void xuat(DATHUC );
DATHUC tong(DATHUC,DATHUC );
DATHUC hieu(DATHUC,DATHUC );
DATHUC tich(DATHUC,DATHUC );
DATHUC thuong(DATHUC,DATHUC );
DATHUC daoham(DATHUC );
DATHUC daoham(DATHUC,int);
float tinhf(DATHUC,float);
```

18.- Phân số

🕮 Các yêu cầu

- Rút gọn phân số
- Tính tổng hai phân số
- Tính hiệu hai phân số
- Tính tích hai phân số
- Tính thương hai phân số
- Kiểm tra phân số tối giản
- Quy đồng hai phân số
- Kiểm tra phân số dương
- Kiểm tra phân số âm
- So sánh hai phân số (Hàm số trả về một trong 3 giá trị: 0,-1,1)
- Định nghĩa toán tử cộng (operation +) cho hai phân số
- Định nghĩa toán tử hiệu (operation -) cho hai phân số
- Định nghĩa toán tử tích (operation *) cho hai phân số
- Định nghĩa toán tử thương (operation /) cho hai phân số
- Định nghĩa toán tử tăng một (operation ++)
- Định nghĩa toán tử giảm một (operation --)

🕮 Các khai báo hàm

```
void nhap(PHANSO &);
void xuat(PHANSO );
void rutgon(PHANSO &);
PHANSO tong(PHANSO,PHANSO);
PHANSO hieu(PHANSO,PHANSO);
PHANSO tich(PHANSO,PHANSO);
PHANSO thuong(PHANSO,PHANSO);
int ktthoigian(PHANSO);
void quidong(PHANSO&,PHANSO&);
int ktduong(PHANSO);
int ktam(PHANSO);
int sosanh(PHANSO,PHANSO);
```

19.- Hỗn số

🕮 Các yêu cầu: tương tự như đối với PhanSo

🕮 Các khai báo hàm

- Khai báo kiểu dữ liệu để biểu diễn thông tin của một hỗn số
- Nhập hỗn số
- Xuất hỗn số
- Rút gọn hỗn số
- Tính tổng hai hỗn số
- Tính hiệu hai hỗn số
- Tính tích hai hỗn số
- Tính thương hai hỗn số
- Kiểm tra hỗn số tối giản
- Quy đồng hai hỗn số

20.- Số phức

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:
- Khai báo kiểu dữ liệu để biểu diễn thông tin của một số phức
- Nhập số phức
- Xuất số phức
- Tính tổng hai số phức
- Tính hiệu hai số phức
- Tính tích hai số phức
- Tính thương hai số phức
- Tính lũy thừa bậc n của số phức

🕮 Khai báo các hàm

```
void nhap(SOPHUC &);
void xuat(SOPHUC );
SOPHUC tong(SOPHUC,SOPHUC);
SOPHUC hieu(SOPHUC,SOPHUC);
SOPHUC tich(SOPHUC,SOPHUC);
SOPHUC thuong(SOPHUC,SOPHUC);
SOPHUC luythua(SOPHUC,int);
```

🕮 Một số gợi ý:
- Khai báo kiểu dữ liệu để biểu diễn thông tin của một số phức

```
struct sophuc
{
    float thuc;
    float ao;
};
typedef struct sophuc SOPHUC;
```

- Định nghĩa hàm nhập

```
void nhap(SOPHUC &x)
{
    float temp;
    printf("Nhap thuc:");
    scanf("%f",&temp);
    x.thuc = temp;
    printf("Nhap ao:");
    scanf("%f",&temp);
    x.ao = temp;
}
```

- Định nghĩa hàm xuất

```
void xuat(SOPHUC x)
{
    printf("%8.3f + i*%8.3f",x.thuc,x.ao);
}
```

21.- Điểm trong mặt phẳng Oxy

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:
- Khai báo dữ liệu
- Nhập tọa độ điểm trong mặt phẳng
- Xuất toa độ điểm theo định dạng
- Tính khỏa cách giửa 2 điểm
- Tính khỏa cách giửa 2 điểm theo phương Ox

- Tính khỏa cách giữa 2 điểm theo phương Oy
- Tìm toa độ điểm đối xứng qua gốc toa độ
- Tìm toa độ điểm đối xứng qua trục hoành
- Tìm toa độ điểm đối xứng qua trục tung
- Tìm toa độ điểm đối xứng qua đường phân giác thứ nhất(y=x)
- Tìm toa độ điểm đối xứng qua đường phân giác thứ hai(y=-x)
- Kiểm tra có thuộc phần tư thứ nhất không?
- Kiểm tra có thuộc phần tư thứ hai không?
- Kiểm tra có thuộc phần tư thứ ba không?
- Kiểm tra có thuộc phần tư thứ tư không?

🕮 Khai báo các hàm

```
void nhap (DIEM&);
void xuat(DIEM);
float khoangcach(DIEM,DIEM);
float khoangcachX(DIEM,DIEM);
float khoangcachY(DIEM,DIEM);
DIEM dxgoc(DIEM);
DIEM dxox (DIEM);
DIEM dxoy (DIEM);
DIEM dxpg1 (DIEM);
DIEM dxpg2 (DIEM);
int ktphantu 1(DIEM);
int ktphantu 2(DIEM);
int ktphantu 3(DIEM);
int ktphantu 4(DIEM);
```

22.- Điểm trong không gian Oxyz

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:

- Khai bai kiểu dữ liệu biểu diễn tọa độ 1 điểm trong không gian Oxyz.
- Nhập toa độ điểm trong khong gian Oxyz
- Xuất toa độ điểm theo định dạng(x,y,z)
- Tính khoảng cách giữa hai điễm trong không gian
- Tính khoảng cách giữa hai điễm trong không gian theo phươngX
- Tính khoảng cách giữa hai điễm trong không gian theo phươngY
- Tính khoảng cách giữa hai điễm trong không gian theo phươngZ
- Tìm toa độ điểm đối xứng qua gốc toa độ
- Tìm toa độ điểm đối xứng qua mặt phẳng Oxy
- Tìm toa độ điểm đối xứng qua mặt phẳng Oxz
- Tìm toa độ điểm đối xứng qua mặt phẳng Oyz

🕮 Khai báo hàm

```
void nhap(DIEMKG&);
void xuat(DIEMKG&);
float khoangcach(DIEMKG,DIEMKG);
float khoangcachx(DIEMKG,DIEMKG);
float khoangcachy(DIEMKG,DIEMKG);
float khoangcachz(DIEMKG,DIEMKG);
DIEMKG dxgoc(DIEMKG);
```

```
DIEMKG dxoxy(DIEMKG);
DIEMKG dxoxz(DIEMKG);
DIEMKG dxoyz(DIEMKG);
```

23.- Đường tròn trong mặt phẳng Oxy

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:

- Khai báo kiểu dữ liệu để biểu diễn đường tròn.
- Nhập đường tròn
- Xuất đường tròn theo định dạng((x,y),r)
- Tính chu vi đường tròn
- Tinh diện tích đường tròn
- Xét vị trí tương đối giữa hai đường tròng(khong cắt nha, tiếp xúa, cắt nhau).
- Kiểm tra một điểm có nằm trong đường tròng hay không
- Cho hai đường tròn. Tính diện tích phần mặt phẳng bị phủ bởi hai đường tròn đó.

🕮 Khai báo hàm

```
void nhap(DIEM&);
void nhap(DUONGTRON&);
void xuat(DIEM);
void xuat(DUONGTRON);
float chuvi(DUONGTRON);
float dientich(DUONGTRONG);
float khoangcach(DIEM.DIEM);
int tuongdoi(DUONGTRON,DUONGTRON);
int tkthuoc(DUONGTRON,DIEM);
```

24.- Hình cầu trong không gian Oxyz

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:

- Khai báo kiểu dữ liệu để biểu diễn hình cầu trong không gian Oxyz
- Nhập hình cầu
- Xuất đường cầu theo định dạng((x,y,z),r).
- Tính diện tích xung quanh hình cầu
- Tính thể tích hình cầu.
- Xét vị trí tương đối giữa hai hình cầu (không cắt nhau, tiếp xúc, cắt nhau).
- Kiểm tra 1 toa độ điểm có nắm bên trong hình càu hay không

🕮 Khai báo hàm

```
void nhap(DIEM&);
void nhap(HINHCAU&);
void xuat (DIEMKG);
void xuat(HINHCAU);
float dientichxungquanh(HINHCAU);
float thetich(HINHCAU);
float khoangcach(DIEMKG,DIEMKG);
int tuongdoi(HINHCAU,HINHCAU);
int tkthuoc(HINHCAU, DIEMKG);
```

🕮 Một số gợi ý

- Khai báo kiểu dữ liệu để biểu diễn thông tin của hình cầu

```
struct diemkg
{
    float x;
    float y;
    float z;
};
typedef struct diemkg DIEMKG;
struct hinhcau
{
    DIEM I;
    float R;
};
typedef struct hinhcau HINHCAU;
```

- Định nghĩa hàm nhập

```
void NhapDiem(DIEMKG&P)
{
    float temp;
    printf("nhap x: ");
    scanf ("%F",&tem);
    P.x=temp;
    printf("Nhap y: ");
    scanf("%f",&temp);
    P.y=temp;
    printf("Nhap z: ");
    scanf("%f",&temp);
    P.z=temp;
}
void NhapHinhCau(HINHCAU&c)
{
    float temp;
    nhap (c.I);
    print("Nhap ban kinh:");
    scanf("%&",&temp);
    c.R=temp;
}
```

- Định nghĩa hàm xuất

```
void XuatDiem(DIEMKG P)
{
    printf("(%8.3f,%8.3f,%8.3f)",P.x,P.y,P.z);
}
void XuatHinhCau(HINHCAU c)
{
    printf("((%8.3f,%8.3f,%8.3f),%8.3f)",c.I.x, c.I.y, c.I.z,c.R);
}
```

25.- Tam giác trong mặt phẳng Oxy

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:

- Khai báo kiểu dữ liệu để biểu diễn tam giác trong mặt phẳng Oxy.
- nhập tam giác.
- Xuất tam giác theo định dạng ((x1,y1);(x2,y2);(x3,y3)).
- Kiểm tra tao độ 3 đỉnh có thật sự lập thành 3 đỉnh của 1 tam giác hay không?
- Tính chu vi tam giác
- Tính điện tích tam giác.

- Tìm tọa độ trọng tâm của tam giác.
- Tìm một điểm trong tam giác có hoành độ lớn nhất.
- Tìm một điểm trong tam giác có tung độ nhỏ nhất.
- Tính tổng khoảng cách từ điểm P(x,y) tới 3 đỉnh của tam giác.
- Kiểm tra một tọa độ điểm có nằm trong tam giác hay không.
- Cho biết dạng của tam giác (đều, vuông, vuông cân, cân, thường).

🕮 Khai báo hàm

```
void nhap (DIEM&);
void nhap (TAMGIAC&);
void xuat (DIEM);
void xuat (TAMGIAC);
float khoangcach (DIEM,DIEM);
int kiemtra (TAMGIAC);
float chuvi (TAMGIAC);
float dien tich (TAMGIAC);
DIEM trongtam (TAMGIAC);
DIEM hoanhlonnhat (TAMGIAC);
DIEM tungthapnhat (TAMGIAC);
int ktthuoc (TAMGIAC,DIEM);
int dangtamgiac(TAMGIAC);
```

🕮 Một số gợi ý

- Khai báo dữ liệu để biểu diễn tam giác trong mặt phẳng Oxy.

```
struct diem
{
    float x;
    float y;
};
typedef struct diem DIEM;
struct tamgiac
{
    DIEM A;
    DIEM B;
    DIEM C;
};
typedef struct tamgiac TAMGIAC;
```

- Định nghĩa hàm nhập điểm

```
void nhap (DIEM&P)
{
    float temp;
    printf("Nhap x: ");
    scanf ("%f",&temp);
    P.x=temp;
    printf("Nhap y: ");
    scanf("%f",&temp);
    P.y=temp;
}
```

- Định nghĩa hàm nhập tam giác

```
void nhap (TAMGIAC &t)
{   Nhap (t.A);
    Nhap (t.A);
    Nhap (t.A);
}
```

- Định nghĩa hàm xuất điểm

```
void xuat (DIEM P)
{
    printf("(%8.3f,%8.3f)",P.x,P.y);
}
```

- Định nghĩa hàm xuất tam giác

```
void xuat (TAMGIAC t)
{
    Xuat (t.A);
    Xuat (t.B);
    Xuat (t.C);
}
```

26.- Ngày

🕮 Hãy viết các hàm thực hiện các yêu cầu sau:

- Khai báo dữ liệu để biểu diễn ngày
- Nhập ngày.
- Xuất ngày theo định dạng (ng/th/nm).
- Kiểm tra năm nhuận.
- Tính số thứ tự ngày trong năm.
- Tính số thứ tự ngày kể từ ngày 1/1/1.
- Tìm ngày khi biết năm và số thứ tự của ngày trong năm.
- Tìm ngày khi biết số thứ tự ngày kể từ ngày 1/1/1.
- Tìm ngày kế tiếp.
- Tìm ngày hôm wa.
- Tìm ngày kế đó k ngày.
- Tìm ngày trước đó k ngày.
- Khoảng cách giữa 2 ngày.
- So sánh 2 ngày.

6.9.3. *Các bài tập khác*

6.9.3.1. Mảng 1 chiều kiểu cấu trúc

27.- Hãy khai báo dữ liệu để biểu diễn thong tin của một tỉnh (TINH). Biết rằng một tỉnh gồm những thành phần như sau:

- Mã tỉnh: kiểu số nguyên 2 byte.
- Tên tỉnh: chuỗi tối đa 30 ký tự.
- Dân số: kiểu số nguyên 4 byte.
- Diện tích: kiểu số thực.

Sau đó viết hàm nhập và xuất cho kiểu dữ liệu này.

28.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một hộp sữa (HOPSUA). Biết rằng một hộp sữa gồm những thành phần sau:

- Nhãn hiệu: chuỗi tối đa 20 ký tự.
- Trọng lượng: kiểu số thực.
- Hạn sử dụng: kiểu dữ liệu ngày (NGAY).

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

29.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một vé xem phim (VE). Biết rằng một vé xem phim gồm những thành phần như sau:

- Tên phim: chuỗi tối đa 20 ký tự.
- Giá tiền: kiểu số nguyên 4 byte.

- Xuất chiếu: kiểu thời gian (THOIGIAN).
- Ngày xem: kiểu dữ liệu ngày (NGAY).

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

30.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một mặt hàng (MATHANG). Biết rằng một mặt hàng gồm những thành phần như sau:
- Tên mặt hàng: chuỗi tối đa 20 ký tự.
- Đơn giá: Kiểu số nguyên 4 byte.
- Số lượng tồn: kiểu số nguyên 4 byte.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

31.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một chuyến bay. Biết rằng một chuyến bay gồm những thành phần như sau:
- Mã chuyến bay: chuỗi tối đa 5 ký tự.
- Ngày bay: kiểu dữ liệu ngày.
- Giờ bay: kiểu thời gian.
- Nơi đi: chuỗi tối đa 20 ký tự.
- Nơi đến: chuỗi tối đa 20 ký tự.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

32.- Hãy khai báo biến dữ liệu để biểu diễn thông tin của một cầu thủ. Biết rằng một cầu thủ gồm những thành phần như sau:
- Mã cầu thủ: chuỗi tối đa 10 ký tự.
- Tên cầu thủ: chuỗi tối đa 30 ký tự.
- Ngày sinh: kiểu dữ liệu ngày.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

33.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một đội bóng (DOIBONG). Biết rằng một đội bóng gồm những thành phần như sau:
- Mã đội bóng: chuỗi tối đa 5 ký tự.
- Tên đội bóng: chuỗi tối đa 30 ký tự.
- Danh sách các cầu thủ: mảng một chiều các cầu thủ (tối đa 30 phần tử)

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

34.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một nhân viên (NHANVIEN). Biết rằng một nhân viên gồm những thành phần như sau:
- Mã nhân viên: chuỗi tối đa 5 ký tự.
- Tên nhân viên: Chuỗi tối đa 30 ký tự.
- Lương nhân viên: kiểu số thực.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

35.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một thí sinh (THISINH). Biết rằng một thí sinh gồm những thành phần như sau:
- Mã thí sinh: chuỗi tối đa 5 ký tự.
- Họ tên thí sinh: chuỗi tối đa 30 ký tự.
- Điểm toán: kiểu số thực.
- Điểm lý: kiểu số thực.
- Điểm hóa: kiểu số thực.
- Điểm tổng cộng: kiểu số thực.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

36.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một luận văn (LUANVAN). Biết rằng một luận văn gồm những thành phần như sau:

- Mã luận văn: chuỗi tối đa 10 ký tự.
- Tên luận văn: chuỗi tối đa 100 ký tự.
- Họ tên sinh viên thực hiện: chuỗi tối đa 30 ký tự.
- Họ tên giáo viên hướng dẫn: chuỗi tối đa 30 ký tự.
- Năm thực hiện: kiểu số nguyên 2 byte.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

37.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một học sinh (HOCSINH). Biết rằng một học sinh gồm những thành phần như sau:

- Tên học sinh: chuỗi tối đa 30 ký tự.
- Điểm toán: kiểu số nguyên 2 byte.
- Điểm văn: kiểu số nguyên 2 byte.
- Điểm trung bình: kiểu số thực.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

38.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của một lớp học (LOPHOC). Biết rằng một lớp học gồm những thành phần sau:

- Tên lớp: chuỗi tối đa 30 ký tự.
- Sĩ số: kiểu số nguyên 2 byte.

Danh sách các học sinh trong lớp (tối đa 50 hoc sinh) sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

39.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của 1 sổ tiết kiệm(SOTIETKIEM). Biết rằng 1 sổ tiết kiệm gồm những thành phần như sau:

- Mã sổ: chuỗi tối đa 5 ký tự.
- Loại tiết kiệm: chuỗi tối đa 10 ký tự.
- Họ tên khách hàng: chuỗi tối đa 30 ký tự.
- Chứng minh nhân dân: kiểu số nguyên 4 byte.
- Ngày mở sổ: kiểu dữ liệu ngày.
- Số tiền gửi: kiểu số thực.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

40.- Hãy khai báo kiểu dữ liệu để biểu diễn thông tin của 1 đại lý(DAILY). Biết rằng mỗi đại lý gồm những thành phần sau:

- Mã đại lý: chuỗi tối đa 5 ký tự.
- Tên đại lý: chuỗi tối đa 30 ký tự.
- Điện thoại: kiểu số nguyên 4 byte.
- Ngày tiếp nhận: kiểu dữ liệu ngày.
- Địa chỉ: chuỗi tối đa 50 ký tự.
- E-mail: chuỗi tối đa 50 ký tự.

Sau đó viết hàm nhập và hàm xuất cho kiểu dữ liệu này.

41.- Xây dựng mảng 1 chiều tọa độ điểm

- Viết hàm nhập mảng.
- Viết hàm xuất mảng.
- Đếm số lượng điểm trong bảng có hoành độ dương.
- Đếm số lượng phần tử không trùng với phần tử khác trong mảng.
- Tìm 1 điểm trong mảng có tung độ lớn nhất trong mảng.
- Tìm 1 điểm trong mảng gần gốc tọa độ nhất.
- Tìm 1 điểm trong mảng xa góc tọa độ nhất.
- Giả sử mang nhiều hơn 2 phần tử và các phần tử trong mảng đôi 1 không trùng nhau. Hãy viết hàm tìm tọa độ hai điểm" gần nhau nhất "trong mảng.

- Giả sử mang nhiều hơn 2 phần tử và các phần tử trong mảng đôi 1 không trùng nhau. Hãy viết hàm tìm tọa độ hai điểm" xa nhau nhất "trong mảng.
- Cho n điểm p1, p2, p3,…, pn trong mặt phẳng OXY trong đó không có hai điểm nào trùng nhau. Một tam giác với các đỉnh thuộc các điểm p1, p2, p3,…, pn được gọi là độc lập nếu nó không chứa (n-3) điểm còn lại. Hãy viết hàm tìm 1 tam giác độc lập trong n điểm này.
- Cho n điểm p1, p2, p3, …, pn trong mặt phẳng OXY trong đó không có hai điểm nào trùng nhau.hãy tìm 1 đa giác lồi với các đỉnh là các điểm trong số n điểm cho trước, sao cho đa giác lồi chứa tất cả các đỉnh còn lại.

42.-Mảng 1 chiều phân số

- Viết hàm nhập mảng.
- Viết hàm xuất mảng.
- Viết hàm đếm số lượng phân số dương trong mảng.
- Viết hàm đếm số lượng phân số âm trong mảng.
- Tìm phân số lớn nhất
- Tìm phân số nhỏ nhất.
- Tìm giá trị lớn nhất trong mảng.
- Tìm giá trị dương đầu tiên trong mảng. Nếu mảng không có giá trị dương thì trả về giá trị phân số 0/1.
- Tìm 1 vị trí mà giá trị tại vị trí đó là nhỏ nhất trong mảng.
- Hãy tìm giá trị dương nhỏ nhất trong mảng. Nếu mảng không có giá trị dương thì trả về phân số không dương là -1/1.
- Hãy tìm vị trí giá trị dương nhỏ nhất trong mảng. Nếu mảng không có giá trị dương thì trả về 1 giá trị ngoài đoạn[0,n-1] là -1 nhằm mô tả không có vị trí nào thỏa ĐK.
- Sắp xếp mảng tăng dần.
- Sắp xếp mảng giảm dần.

43.-Xây dựng mảng 1 chiều số phức

- Viết hàm nhập mảng.
- Viết hàm xuất mảng.
- Tính tổng tất cả các số phức có trong mảng.
- Tính tích tất cả các số phức có trong mảng.
- Tìm số phức đầu tiện trong mảng có phần thực dương và phần ảo dương.
- Sắp các số phức trong mảng tăng dần theo phần thực.

44.-Xây dựng mảng 1 chiều các đường tròn

- Cho n đường tròn. Tìm 1 đường tròn gần gốc tọa độ nhất.
- Cho n đường tròn.hãy kiểm tra xem có 1 đường tròn nào trong n đường tròn đi qua gốc tọa độ hay không.
- Cho n đường tròn,tìm 1 đường tròn trong n đường tròn đó gần trục OX nhất.
- Cho n đường tròn.hãy kiểm tra xem có đường tròn nào tiếp xúc trục tung hay không.
- Cho n đường tròn. Hãy kiểm tra xem các đường tròn này có đôi 1 cắt nhau hay không.
- Cho n đường tròn đôi 1 không trùng nhau.hãy liệt kê tất cả các cặp đường tròn tiếp xúc nhau.
- Cho n đường tròn. Hỏi có tồn tại điểm trong mặt phẳng OXY thuộc tất cả các đường tròn này không. Nếu có chỉ ra tọa độ 1 điểm.

45.-Xây dựng mảng 1 chiều các đường thẳng

- Cho n đường thẳng.hãy kiểm tra các đường thẳng này có cùng song song với nhau không.
- Cho n đường thẳng. hãy kiểm tra các đường thẳng này có cặp đường thẳng cùng song song với nhau không.
- Cho n đường thẳng. hãy kiểm tra các đường thẳng này có cặp đường thẳng trùng nhau không.
- Cho n đường thẳng. hãy kiểm tra xem có 3 đường thẳng nào đồng quy hay không.
- Cho n đường thẳng. Và tọa độ 1 điểm P. Hãy kiểm tra xem có đường thẳng nào đi qua điểm P hay không.
- Cho n đường thẳng và tọa độ điểm P không thuộc bất cứ đường thẳng nào. Hãy tìm 1 đường thẳng gần với điểm P nhất.

6.9.3.2. Mảng 2 chiều kiểu cấu trúc

46.-Ma trận tọa độ điểm

- Viết hàm nhập ma trận.
- Viết hàm xuất mảng.
- Đếm số lượng điểm trong ma trận thuộc góc phần tư thứ 3 trong mặt phẳng tọa độ Oxy.
- Đếm tần suất xuất hiện của tọa độ một điểm trong ma trận.
- Đếm số lượng điểm không trùng với điểm khác trong ma trận.
- Tìm tọa dộ gần nhất với tọa độ điểm P(x,y) nhất trong ma trận.
- Đếm số lượng điểm trong ma trận thuộc đường thẳng ax + by + c = 0.
- Tìm một điểm trong ma trận gần đường thẳng ax + by + c = 0 (DUONGTHANG) nhất và không năm trên đường thẳng này.
- Đếm số lượng điểm trong ma trận nằm trong đường tròn r(I,R) (DUONGTRON).

47.-Ma trận phân số

- Viết hàm nhập mảng.
- Viết hàm xuất mảng.
- Tìm phân số lớn nhất trong ma trận.
- Đếm số lượng phân số nhỏ nhất trong ma trận.
- Tìm phân số âm lớn nhất trong ma trận.
- Liệt kê các phân số tối giản trong ma trận theo thứ tự tăng dần.
- Sắp xếp ma trận tăng dần từ trái sang phải và từ trên xuống dưới.

48.-Ma trận số phức

- Viết hàm nhập ma trận số phức.
- Viết hàm xuất ma trận số phức.
- Tìm số phức đầu tiên trong ma trận có phần thực dương và phần ảo dương
- Tìm số phức cuối cùng trong ma trận có phần thực âm và phần ảo âm.
- Đếm số lượng dòng trong ma trận có chứa toàn số phức thỏa điều kiện: “phần thực và phần ảo trái dấu nhau”.
- Tìm số phức có phần thực lớn nhất trong ma trận.

ĐỆ QUY *(Recursion)*

7.1. KHÁI NIỆM

Một hàm được gọi có tính đệ quy nếu trong thân của hàm đó có lệnh gọi lại chính nó một cách tường minh hay tiềm ẩn.

7.2. PHÂN LOẠI ĐỆ QUY

- Đệ quy tuyến tính.
- Đệ quy nhị phân.
- Đệ quy phi tuyến.
- Đệ quy hỗ tương.

7.2.1. *Đệ quy tuyến tính*

7.2.1.1. *Khái niệm*

Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó một cách tường minh.

7.2.1.2. *Cấu trúc hàm đệ quy tuyến tính:*

```
<Kiểu dữ liệu hàm> TenHam (<danh sách tham số>)
{   if (điều kiện dừng)
    {   . . .
        //Trả về giá trị hay kết thúc công việc
    }
    //Thực hiện một số công việc (nếu có)
    . . . TenHam (<danh sách tham số>);
    //Thực hiện một số công việc (nếu có)
}
```

7.2.1.3. *Một số ví dụ*

7.2.1.3.1. *Ví dụ 1*

Tính $S(n) = 1 + 2 + 3 + ... + n$

- Đây là một ví dụ có thể cài đặt dễ dàng bằng phương pháp thông thường, tuy nhiên nếu dựa vào định nghĩa của S(n) chúng ta có thể cài đặt một cách tự nhiên bằng phương pháp đệ quy như sau:
- Trước khi cài đặt hàm đệ quy ta xác định:
  - *Quy tắc (công thức) tính: S(n) = S(n-1) + n.*
    Ta có : S(n) = 1 + 2 + 3 + ... + (n-1) + n
    Do đó : S(n-1) = 1 + 2 + 3 + ... + (n-1).
    Suy ra quy tắc (công thức) tính: S(n) = S(n-1)+n;
  - *Điều kiện dừng: S(0) = 0.*
  - *Cài đặt hàm đệ quy:*

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

7.2.1.3.2. *Ví dụ 2*

Tính P(n) = n!

- Trước khi cài đặt hàm đệ quy ta xác định:
  - *Điều kiện dừng: P(0) = 0! = 1.*
  - *Quy tắc (công thức) tính: P(n) = P(n-1) * n.*
- Cài đặt hàm đệ quy:

```
long GiaiThua (int n)
{
     if(n==0)
          return 1;
     return ( GiaiThua(n-1) *  n );
}
```

7.2.1.3.3. *Ví dụ 3*

Cho mảng một chiều các số nguyên. Viết hàm tính tổng cho các số chẵn trong mảng bằng phương pháp đệ quy.

- Ta có hình ảnh mảng a có n phần tử như sau :

0	1		n-2	n-1
12	43	…	232	44

0	1		n-2
12	43	…	232

- Cài đặt hàm đệ quy:

```
long Tong(int a[], int n)
{
     if (n==0)
          return 0;
     if(a[n-1]%2==0)
          return Tong(a,n-1)+a[n-1];
     return Tong(a,n-1);
}
```

7.2.1.3.4. *Ví dụ 4*

Cho mảng một chiều các số thực. Viết hàm đếm số lượng giá trị dương trong mảng bằng phương pháp đệ quy.

- Cài đặt hàm đệ quy:

```
int DemDuong(float a[],int n)
{
     if(n==0)
          return 0;
     if(a[n-1]>0)
          return 1+ DemDuong (a,n-1);
     return DemDuong (a,n-1);
}
```

7.2.1.3.5. *Ví dụ 5*

Hàm tính căn bậc 3 của một số thực có thể cài đặt đệ quy theo hai hàm exp và log nhờ vào nhận xét sau đây:

$$\sqrt[3]{x} = \begin{cases} \sqrt[3]{x} & khi \quad x \geq 0 \\ -\sqrt[3]{-x} & khi \quad x < 0 \end{cases}$$

- Cài đặt hàm đệ quy:

```
float sqrt3(float x)
{
    if(x==0)
        return 0;
    if(x<0)
        return (-sqrt3(-x));
    return (exp((log(x)/3)));
}
```

7.2.2. *Đệ quy nhị phân*

7.2.2.1. *Cấu trúc hàm đệ quy nhị phân*

Trong thân của hàm có 2 lời gọi hàm gọi lại chính nó 1 cách tường minh.

```
<Kiểu dữ liệu hàm> TenHam (<danh sách tham số>)
{   if (điều kiện dừng)
    {   . . .
        //Trả về giá trị hay kết thúc công việc
    }
    //Thực hiện một số công việc (nếu có)
    ... TenHam (<danh sách tham số>); //Giải quyết vấn đề nhỏ hơn
    //Thực hiện một số công việc (nếu có)
    ... TenHam (<danh sách tham số>); //Giải quyết vấn đề còn lại
    //Thực hiện một số công việc (nếu có)
}
```

7.2.2.2. *Ví dụ*

7.2.2.2.1. *Ví dụ 1*

- Tính số hạng thứ n của dãy Fibonaci được định nghĩa như sau:

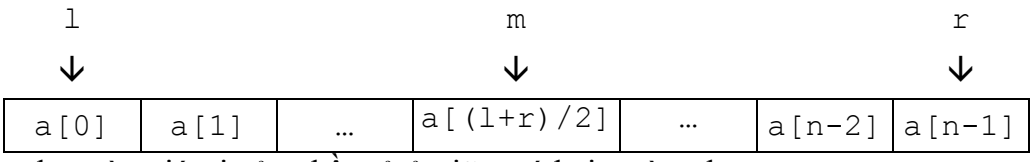
$$f_1 = f_0 = 1 ;$$
$$f_n = f_{n-1} + f_{n-2} ; \quad (n>1)$$

- Trước khi cài đặt hàm đệ quy ta xác định:
  - Điều kiện dừng: f(0) = f(1) = 1.
- Cài đặt hàm đệ quy:

```
long Fibonaci (int n)
{
    if(n==0 || n==1)
        return 1;
    return  Fibonaci(n-1) + Fibonaci(n-2);
}
```

7.2.2.2.2. *Ví dụ 2*

- Cho dãy số nguyên a gồm n phần tử có thứ tự tăng dần. Tìm phần tử có giá trị x có xuất hiện trong mảng không?
- Trước khi cài đặt hàm đệ quy ta xác định:
  - Điều kiện dừng: Tìm thấy x hoặc xét hết các phần tử.
- *Giải thuật*: Do dãy số đã có thứ tự tăng nên ta có thể áp dụng cách tìm kiếm theo phương pháp nhị phân. Ý tưởng của phương pháp này là tại mỗi bước ta tiến hành so sánh x với phần tử nằm ở vị trí giữa của dãy để thu hẹp phạm vi tìm.
- Gọi: l: biên trái của dãy (ban đầu l=0).
  r: biên phải của dãy (ban đầu r = n-1).
  m: vị trí ở giữa (m = (l+r)/2).



- Thu hẹp dựa vào giá trị của phần tử ở giữa, có hai trường hợp:
  - Nếu x lớn hơn phần tử ở giữa thì x chỉ có thể xuất hiện ở bên phải vị trí này. (từ m+1 đến r).
  - Ngược lại nếu x nhỏ hơn phần tử ở giữa thì x chỉ có thể xuất hiện ở bên trái vị trí này. (từ l đến m-1).
  - Quá trình này thực hiện cho đến khi gặp phần tử có giá trị x, hoặc đã xét hết các phần tử.
- Cài đặt hàm đệ quy:

```
int TimNhiPhan(int a[], int l, int r, int x)
{
     int m = (l+r)/2;
     if(l>r)
          return -1;// Không có phần tử x
     if(a[m]>x)
          return TimNhiPhan(a, l, m-1, x);
     if(a[m]<x)
          return TimNhiPhan(a, m+1, r, x);
     return m;  //Trả về vị trí tìm thấy
}
```

7.2.2.2.3. *Ví dụ 3*

- Bài toán tháp Hà Nội:
  - Bước 1: Di chuyển n -1 đĩa nhỏ hơn từ cọc A sang cọc B.
  - Bước 2: Di chuyển đĩa còn lại từ cọc A sang cọc C.
  - Bước 3: Di chuyển n -1 đĩa nhỏ hơn từ cọc B sang cọc C.
- Cài đặt hàm đệ quy:

```
void ThapHaNoi (int n, char A, char B, char C)
{
     if (n == 1)
          printf("Di chuyen dia tren cung tu A den C");
     else
     {
          ThapHaNoi(n-1, A, C, B);
          ThapHaNoi(1, A, B, C);
          ThapHaNoi(n-1, B, A, C);
     }
}
```

7.2.2.2.4. *Ví dụ 4*

- Viết hàm đệ quy tính số hạng thứ n của dãy Fibonacy. Dãy Fibo được định nghĩa như sau:

$$f(0) = f(1) = 1$$

$$f(n) = f(n-1) + f(n-2) \quad n > 1$$

- Điều kiện dừng: $f(0) = 1$ và $f(1) = 1$
- Cài đặt hàm đệ quy:

```
long Fibo(int n)
{
     if(n==0)
          return 1;
```

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

7.2.2.2.5. Ví dụ 5

- Viết hàm đệ quy tính tổng của biểu thức sau:

$$S(x,n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Ta có:

$$S(x,n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^i}{i!} + \cdots + \frac{x^{n-3}}{(n-3)!} + \frac{x^{n-2}}{(n-2)!} + \frac{x^{n-1}}{(n-1)!} + \frac{x^n}{n!}$$

Suy ra:

$$S(x,n-1) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^{n-2}}{(n-2)!} + \frac{x^{n-1}}{(n-1)!}$$

$$S(x,n-1) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^{n-3}}{(n-3)!} + \frac{x^{n-2}}{(n-2)!}$$

Lấy $S(x,n) - S(x,n-1)$ ta được : $S(x,n) - S(x,n-1) + \frac{x^n}{n!}$ (*)

Tương tự lấy $S(x,n-1) - S(x,n-2)$ ta được:

$$S(x,n-1) - S(x,n-2) + \frac{x^{n-1}}{(n-1)!}$$

Nhân hai vế của biểu thức trên cho $\frac{x}{n}$ ta được :

$$\Rightarrow \frac{x}{n} S(x, n-1) = \frac{x}{n} S(x, n-2) + \frac{x^{n-1}}{(n-1)!}\frac{x}{n}$$

$$\Rightarrow \frac{x}{n} S(x, n-1) = \frac{x}{n} S(x, n-2) + \frac{x^n}{n!}$$

$$\Rightarrow \frac{x^n}{n!} = \frac{x}{n} S(x, n-1) - \frac{x}{n} S(x, n-2)$$

$$\Rightarrow \frac{x^n}{n!} = \frac{x}{n} (S(x, n-1) - S(x, n-2)) \quad (**)$$

Thay (**) vào (*) ta được:

$$S(x, n) = S(x, n-1) + \frac{x}{n} (S(x, n-1) - S(x, n-2))$$

$$S(x, n) = (1 + \frac{x}{n}) S(x, n-1) - \frac{x}{n} S(x, n-2)$$

Công thức trên được gọi là công thức đệ quy nhị phân.

Điều kiện dừng: S(x,0) = 0 và S(x,1) = x

- Cài đặt hàm đệ quy:

```
float Tinh (float x, int n)
{
    if (n==0)    return 0;
    if (n==1)    return x;
    return (1+x/n)* Tinh (x, n-1) - (x/n) * Tinh (x, n-2);
}
```

7.2.3. *Đệ quy phi tuyến*

7.2.3.1. *Cấu trúc hàm đệ quy phi tuyến*

Trong thân của hàm có lời gọi hàm gọi lại chính nó được đặt bên trong vòng lặp.

```
<Kiểu dữ liệu hàm> TenHam (<danh sách tham số>)
{   for (int i = 1; i<=n; i++)
    {     //Thực hiện một số công việc (nếu có)
          if (điều kiện dừng)
          {
                ...
               //Trả về giá trị hay kết thúc công việc
          }
          else
          {     //Thực hiện một số công việc (nếu có)
               TenHam (<danh sách tham số>);
          }
    }
}
```

7.2.3.2. *Ví dụ*

7.2.3.2.1. *Ví dụ1*

- Tính số hạng thứ n của dãy {Xn} được định nghĩa như sau:

  $X_0 = 1$;

  $X_n = n^2X_0 + (n-1)^2X_1 + \ldots + 1^2X_{n-1}$; (n#1)

- Trước khi cài đặt hàm đệ quy ta xác định:

  - Điều kiện dừng:X(0) = 1.

- Cài đặt hàm đệ quy:

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

7.2.3.2.2. *Ví dụ 2*

- Tính số hạng thứ n của dãy được định nghĩa như sau:

$x(0) = 1$

$x(n) = n^2*(0) + (n-1)^2*(1) + \ldots + (n-i)^2*(i) + \ldots + 2^2*(n-2) + 1^2*(n-1)$

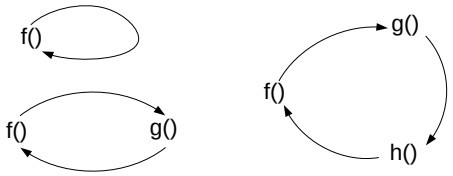
- Cài đặt hàm đệ quy:

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

7.2.4. *Đệ quy hỗ tương*

7.2.4.1. *Cấu trúc hàm đệ quy hỗ tương*

- Hai hàm được gọi là đệ quy hồ tương nếu trong thân của hàm này có lời gọi hàm tới hàm kia và bên trong hàm kia có lời gọi tới hàm này.
- Hình vẽ minh họa




```
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>);
<Kiểu dữ liệu hàm> TenHam1 (<danh sách tham số>)
{     //Thực hiện một số công việc (nếu có)
     …TenHam2 (<danh sách tham số>);
     //Thực hiện một số công việc (nếu có)
}
<Kiểu dữ liệu hàm> TenHam2 (<danh sách tham số>)
{     //Thực hiện một số công việc (nếu có)
     …TenHam1 (<danh sách tham số>);
     //Thực hiện một số công việc (nếu có)
}
```

7.2.4.2. *Ví dụ*

Tính số hạng thứ n của hai dãy {Xn}, {Yn} được định nghĩa như sau:

$X_0 = Y_0 = 1$;

$X_n = X_{n-1} + Y_{n-1}$; (n>0)

$Y_n = n^2 X_{n-1} + Y_{n-1}$; (n>0)

- Trước khi cài đặt hàm đệ quy ta xác định:

  + Điều kiện dừng:X(0) = Y(0) = 1.

- Cài đặt hàm đệ quy:

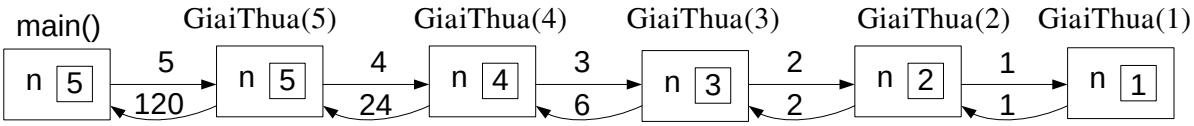
```
long TinhYn(int n);
long TinhXn (int n)
{
     if(n==0)
          return 1;
     return TinhXn(n-1) + TinhYn(n-1);
}
long TinhYn (int n)
{
     if(n==0)
          return 1;
     return n*n*TinhXn(n-1) + TinhYn(n-1);
}
```

7.3. TÌM HIỂU CÁCH HOẠT ĐỘNG CỦA HÀM ĐỆ QUY

Phục vụ cho công việc kiểm chứng kết quả thực thi của chương trình bằng tay.

7.3.1. *Ví dụ 1*

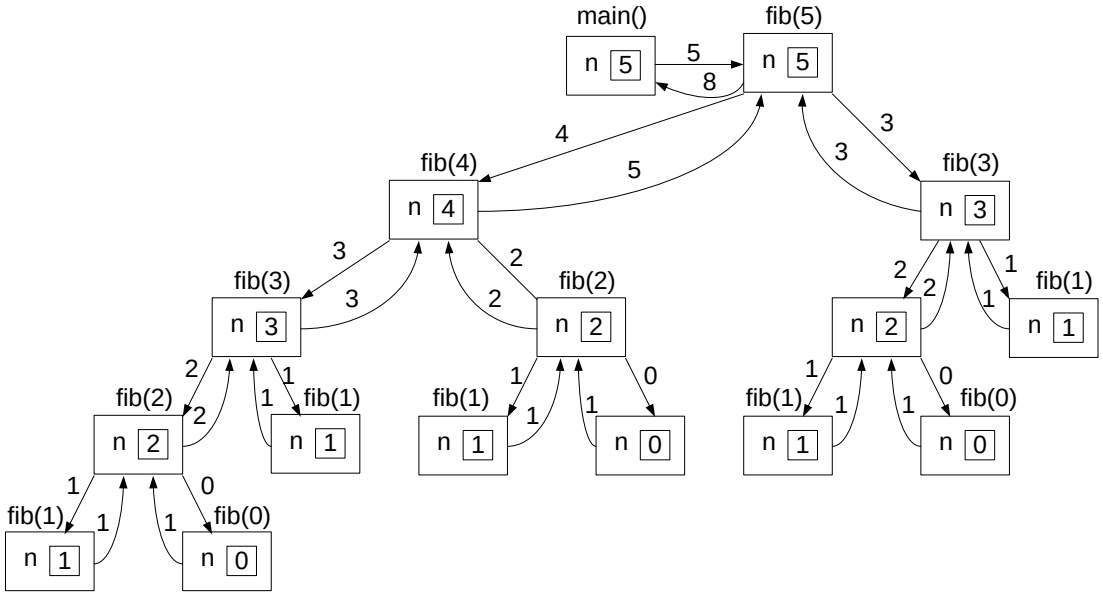
Lấy lại Ví dụ tính P(n) = n! bằng phương pháp đệ quy như đã mô tả cài đặt ở trên với n = 5



Lệnh gọi khởi đầu trong hàm main(), truyền đến hàm GiaiThua(). Ở đó, giá trị của tham số n là 5, do đó nói gọi GiaiThua(4), truyền 4 đến hàm GiaiThua(). Ở đó giá trị của tham số n là 4, do đó nó gọi GiaiThua(3), truyền 3 đến hàm GiaiThua(). Tiến trình này tiếp tục (đệ quy) đến khi gọi GiaiThua(1) được thực hiện từ bên trong lệnh gọi GiaiThua(2). Ở đó, giá trị của tham số n là 1, do đó nó trả về giá trị 1, mà không thục hiện thêm bất kì lệnh gọi nào. Sau đó lần ngược về lệnh gọi GiaiThua(2) trả 2*1=2 trở về lệnh gọi GiaiThua(3). Sau đó lệnh gọi GiaiThua(3) trả 3*2=6 trở về lệnh gọi GiaiThua(4). Sau đó lệnh gọi GiaiThua(4) trả 4*6=24 trở về lệnh gọi GiaiThua(5). Sau cùng, lệnh gọi GiaiThua(5) trả về giá trị 120 cho hàm main().

7.3.2. *Ví dụ 2*

Lấy lại Ví dụ tính số hạng thứ n của dãy Fibonaci như đã mô tả cài đặt ở trên với n = 5, quá trình thực hiện tương tự như trong ví dụ trước, ta có sơ đồ:



7.3.3. *Ví dụ 3*

Viết chương trình nhập vào mảng một chiều số nguyên a, xuất ra màn hình và tính tổng các phần tử có giá trị chẵn bằng phương pháp đệ quy.

```
#include <stdio.h>
#define MAX 100
void Nhap(int a[], int n)
{
     if(n==0)
          return;
     Nhap(a, n-1);
     printf("\nNhap phan tu thu %d: ",n);
     scanf("%d",&a[n-1]);
}
```

```
long TongChan(int a[], int n)
{
     if(n==0)
          return 0;
     long s = TongChan(a, n-1);
     if(a[n-1]%2==0)
          s+=a[n-1];
     return s;
}
void Xuat(int a[], int n)
{
     if(n==0)
          return;
     Xuat(a, n-1);
     printf("%5d",a[n-1]);
}
int main()
{
     int a[MAX], n;
     long s;
     printf("Nhap so phan tu cua mang: ");
     scanf("%d",&n);
     Nhap(a, n);
     Xuat(a, n);
     s=TongChan(a, n);
     printf("\nTong cac so chan trong mang la: %ld",s);
     return 0;
}
```

7.4. MỘT SỐ BÀI TOÁN ĐỆ QUY THÔNG DỤNG

7.4.1. *Bài toán tháp Hà Nội*

Có 3 chồng đĩa đánh số 1, 2 và 3. đầu tiên chồng 1 có n đĩa được xếp sao cho đĩa lớn hơn nằm bên dưới và 2 chồng còn lại không có đĩa nào. Yêu cầu: chuyển tất cả các đĩa từ chồng 1 sang chồng 3, mỗi lần chuyển 1 đĩa và được phép sữ dụng chồng 2 làm trung gian. Hơn nữa trong quá trình chuyển đĩa phải bảo đảm quy tắc đĩa lớn nằm bên dưới.

7.4.2. *Bài toán phát sinh hoán vị*

Cho tập hợp A có n phần tử được đánh số 1,2,…,n. Một hoán vị của A là một dãy a_1, a_2, …, a_n. Trong đó $a_i \in A$ và chúng đôi một khác nhau. Hãy viết hàm phát sinh tất cả các hoán vị của tập hợp A.

7.4.3. *Bài toán Tám Hậu*

Cho bàn cờ vua kích thước (8x8). Hãy sắp 8 quân hậu vào bàn cờ sao cho không có bất ỳ 2 quân hậu nào có thể ăn nhau.

7.4.4. *Bài toán Mã Đi Tuần*

Cho bàn cờ vua kích thước (8x8). Hãy di chuyển quân mã trên khắp bàn cờ sao cho mỗi ô đi đúng 1 lần.

7.5. ĐỆ QUY VÀ MẢNG MỘT CHIỀU

7.5.1. *Xuất mảng*

- Xuất xuôi (từ trái sang phải)
  - Ý tưởng: xuất mảng a có n-1 phần tử trước, sau đó xuất tiếp phần tử cuối cùng trong mảng.
  - Hàm cài đặt:

```
void Xuat(int A[], int n)
{
    if(n==0)
      return;
    Xuat(A, n-1);
    printf("%5d",A[n-1]);
}
```

- Xuất ngược (từ phải sang trái)
  - Ý tưởng: sắp xếp hàm đệ quy của bài toán 1. hay nói một cách khác ta xuất phần tử cuối cùng trong mảng, sau đó xuất mảng a có n-1 phần tử.
  - Hàm cài đặt:

```
void Xuat(int A[], int n)
{
    if(n==0)
      return;
    printf("%5d",A[n-1]);
    Xuat(A, n-1);
}
```

7.5.2. *Kỹ thuật Đếm*

7.5.2.1. *Ví dụ 1: cho mảng một chiều các số thực. Viết hàm đệ quy đếm số lượng giá trị dương trong mảng.*

- Ý tưởng: đếm số lượng dương trong mảng a có n-1 phần tử. sau đó xét tới phần tử cuối cùng trong mảng.
- Hàm cài đặt:

```
int DemDuong (float A[], int n)
{
    if (n==0)
        return;
    int Dem = DemDuong (A, n-1);
    if (A[n-1] >0)
        Dem++;
    return Dem;
}
```

- Cải tiến hàm cài đặt: hạn chế biến đếm bằng cách xem xét giá trị cuối cùng trong mảng trước.

```
int DemDuong (float a[], int n)
{   if (n==0)
        return;
    if (A[n-1] > 0)
        return 1 + DemDuong (A[n-1]);
    return DemDuong (A, n-1);
}
```

7.5.2.2. *Ví dụ 2: Cho mảng một chiều các số nguyên. Viết hàm đệ quy đếm số lượng giá trị phân biệt có trong mảng.*

- *Ý tưởng*: đếm số lượng giá trị phân biệt trong mảng a có n-1 phần tử. sau đó xét tới phần tử cuối cùng trong mảng có trùng với phần tử đứng đằng trước nó không. Nếu không trùng ta tăng giá trị phân biệt lên 1.
- Hàm cài đặt

```
int DemPhanBiet (int A[], int n)
{
     if (n<1)
          return 0;
     if (n==1)
          return 1;
     int Dem = DemPhanBiet (A, n-1);
     int flag = 1;
     for (int i=0; i<=n-2; i++)
          if (A[i] == A[n-1])
               flag = 0;
     if (flag == 1)
          Dem++;
     return Dem;
}
```

- Cải tiến thứ nhất của hàm cài đặt: không thực hiện việc kiểm tra cờ hiệu có còn bật hay không.

```
int DemPhanBiet (int A[], int n)
{    if (n<1)
          return 0;
     if (n==1)
          return 1;
     int Dem = DemPhanBiet (A, n-1);
     int Flag = 1;
     for (int i=0; i<=n-2; i++)
          if (A[i] == A[n-1])
               Flag = 0;
     Dem = Dem + Flag
     return dem;
}
```

- Cải tiến thứ 2 của hàm cài đặt: thêm điều kiện lặp để hạn chế số lần lặp kiểm tra.

```
int DemPhanBiet (int A[], int n)
{    if (n<1)
          return 0;
     if (n==1)
          return 1;
     int Dem = DemPhanBiet (A, n-1);
     int Flag = 1;
     for (int i=0; i<=n-2 && flag==1; i++)
          if (A[i] == A[n-1])
               Flag = 0;
     return Flag + DemPhanBiet (A, n-1);
}
```

7.5.3. *Kỹ thuật tính toán*

7.5.3.1. *Ví dụ 1: Cho mảng một chiều các số thực. Viết hàm đệ quy tính tổng các giá trị có trong mảng.*

- Ý tưởng: Tính tổng các giá trị có trong mảng a có n-1 phần tử. Sau đó cộng thêm phần tử cuối cùng trong mảng.
- Hàm cài đặt

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

7.5.3.2. *Ví dụ 2: Cho mảng một chiều các số thực. Hãy viết hàm đệ quy tính tổng các giá trị dương có trong mảng.*

- Ý tưởng: tính tổng các giá trị dương có trong mảng a có n-1 phần tử. sau đó cộng thêm phần tử cuối cùng trong mảng nếu phần tử này dương.
- Hàm cài đặt

```
float TongDuong (float A[], int n)
{
    if (n==0)
        return 0;
    float s = TongDuong (A, n-1)
    if (A[n-1] > 0)
        s = s + A[n-1];
    return s;
}
```

- Cải tiến của hàm cài đặt: tinh giản biến s bằng cách thực hiện việc kiểm tra phần tử cuối cùng trước.

```
float TongDuong (float A[], int n)
{   if (n==0)
      return 0;
    float s = TongDuong (A, n-1)
    if (A[n-1] > 0)
        return TongDuong (A, n-1) + A[n-1];
    return TongDuong (A, n-1);
}
```

7.5.3.3. *Ví dụ 3: cho mảng một chiều các số thực. Hãy viết hàm đệ quy tính tích các giá trị lớn hơn các giá trị đứng trước nó trong mảng.*

- Điều kiện dùng trong bài toán này là n==1 vì phần tử đầu tiên trong mảng không có phần tử đứng đằng trước nó.
- Hàm cài đặt

```
float Tich (float A[], int n)
{   if (n<1)
        return 1;
    if (n==1)
        return 1;
    if (A[n-1] >A[n-2])
        return Tich (A, n-1) * A[n-1];
    return tich (A, n-1);
}
```

7.5.4. *Kỹ Thuật Đặt Cờ Hiệu*

- *Yêu cầu*: Cho mảng một chiều các số thực. hãy viết hàm đệ quy kiểm tra mảng có thỏa mảng tính chất "toàn giá trị âm".
- *Ý tưởng*: kiểm tra tính chất "toàn giá trị âm" trên mảng a có n-1 phần tử. sau đó kiểm tra phần tử cuối cùng trong mảng.
- Hàm cài đặt

```
int ToanAm (float A[], int n)
{
    if (n==0)
        return 0;
    if (n==1)
    {
        if (A[0] < 0)
          return 1;
        return 0;
    }
    if ((ToanAm (A, n-1)==1) && (A[n-1] <0))
        return 1;
    return 0;
}
```

- Cải tiến hàm cài đặt: xét phần tử cuối trước với điều kiện phủ định tính chất âm.

```
int ToanAm (float A[], int n)
{
    if (n==0)
        return 0;
    if (n==1)
    {
        if (A[0] < 0)
            return 1;
        return 0;
    }
    if (A[n-1] >= 0)
        return 1;
    return toanam (A, n-1);
}
```

7.5.5. *Kỹ Thuật Tìm Kiếm*

7.5.5.1. *Ví dụ 1: Cho mảng một chiều các số thực. Hãy viết hàm đệ quy tìm giá trị lớn nhất có trong mảng.*

- Ý tưởng: Tìm giá trị lớn nhất trên mảng a có n-1 phần tử. sau đó so sánh với phần tử cuối cùng trong mảng.
- Hàm cài đặt

```
float LonNhat (float A[], int n)
{
    if (n==1)
        return A[0];
    float Max = LonNhat (A, n-1);
    if (A[n-1] > Max)
        Max = A[n-1];
    return Max;
}
```

7.5.5.2. *Ví dụ 2: Cho mảng một chiều các số thực. Hãy viết hàm đệ quy tìm vị trí mà giá trị tại đó là giá trị lớn nhất có trong mảng.*

- Ý tưởng: tìm vị trí giá trị lớn nhất trên mảng a có n-1 phần tử. sau đó so sánh với phần tử cuối cùng trong mảng.
- Hàm cài đặt

```
int ViTriLonNhat (float A[], int n)
{
    if (n==1)
        return 0;
    int MaxPos = ViTriLonNhat (A, n-1);
    if (A[n-1]>A[MaxPos])
        MaxPos = n-1;
    return MaxPos;
}
```

7.5.6. *Kỹ thuật Sắp Xếp*

7.5.6.1. *Ví dụ 1: Cho mảng một chiều các số thực. Hãy viết hàm đệ quy sắp xếp các giá trị trong mảng tăng dần.*

- Ý tưởng: đưa giá trị lớn nhất về cuối mảng. sau đó sắp xếp mảng a có n-1 phần tử tăng dần.
- Hàm cài đặt 1: So sánh và hoán vị phần tử cuối nếu xảy ra "nghịch thế".

```
void SapXep (float A[], int n)
{
    if (n==1)
        return ;
    for (int i=0; i<=n-2; i++)
        if (A[i] >A[n-1])
        {
            float temp=a [i];
            a [i] = a [n-1];
            a [n-1] = temp;
        }
    SapXep(A, n-1);
}
```

- Hàm cài đặt 2: tìm vị trí lớn nhất và hoán vị giá trị tại vị trí đó với phần tử cuối cùng.

```
void SapXep (float A[], int n)
{   if (n==1)
        return ;
    int vt=0;
    for (int i=0; i<=n-1; i++)
        if (A[i] >A[vt])
        {   vt = i;
            float temp=A[vt];
            A[vt] = A[n-1];
            A[n-1] = temp;
            SapXep(A, n-1);
        }
}
```

7.5.6.2. *Ví dụ 2: Cho mảng một chiều các số nguyên. Hãy viết hàm đệ quy sắp xếp các giá trị chẵn trong mảng tăng dần, các giá trị lẻ vẫn giữ nguyên giá trị và vị trí trong mảng.*

- Ý tưởng: Cải tiến từ thuật toán sắp xếp đệ quy trên.
- Hàm cài đặt:

```
void SapXep (float A[], int n)
{    if (n==1)
         return ;
     for (int i=0; i<=n-2; i++)
         if (A[i]>A[n-1] && A[i]%2==0 && A[n-1]%2==0)
         {
              int temp=A[i];
              A[i] = A[n-1];
              A[n-1] = temp;
         }
     SapXep(A, n-1);
}
```

- Cải tiến hàm cài đặt: Kiểm phần tử cuối có phải là giá trị chẵn để hạn chế số lần lặp.

```
void SapXep (float A[], int n)
{    if (n==1)
         return ;
     if (A[n-1]%2==0)
     {
         for (int i=0; i<=n-2; i++)
              if (A[i]>A[n-1] && A[i]%2==0)
              {    int temp=A[i];
                   A[i] = A[n-1];
                   A[n-1] = temp;
              }
     }
     SapXep(A, n-1);
}
```

7.6. BÀI TẬP

Viết các hàm đệ Quy thực hiện các yêu cầu sau:

1.- Tính S(n) = 1+2+3+…+n.

2.- Tính S(n) = $1^2+2^2+3^2+\ldots+n^2$.

3.- Tính S(n) = $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$

4.- Tính S(n) = $1+\frac{1}{2}+\frac{1}{4}+\ldots+\frac{1}{2n}$

5.- Tính S(n) = $1+\frac{1}{3}+\frac{1}{5}+\ldots+\frac{1}{2n+1}$

6.- Tính S(n) = $\frac{1}{1\times 2}+\frac{1}{2\times 3}+\ldots+\frac{1}{n\times(n+1)}$

7.- Tính S(n) = $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\ldots+\frac{n}{n+1}$

8.- Tính S(n) = $\frac{1}{2}+\frac{3}{4}+\frac{5}{6}+\ldots+\frac{2n+1}{2n+2}$

9.- Tính T(n) = $1\times 2\times\ldots\times n$

10.-Tính T(x, n) = x^n

11.-Tính S(n) = 1 + 1.2 + 1.2.3 + 1.2.3…n

12.-Tính S(n) = $x + x^2 + x^3 +\ldots x^n$

13.-Tính S(n) = $x^2 + x^4 +\ldots x^{2n}$

14.-Tính S(n) = $x + x^3 + x^5 +\ldots x^{2n+1}$

15.-Tính S(n) = $1 + \frac{1}{1+2} + \frac{1}{1+2+3} +\ldots+ \frac{1}{1+2+3+...+n}$.

16.-Tính S(n) = $x + \frac{x^2}{2!} + \frac{x^3}{3!} +\ldots+ \frac{x^n}{n!}$.

17.-Tính S(n) = $1 + \frac{x^2}{2!} + \frac{x^4}{4!} +\ldots+ \frac{x^{2n}}{(2n)!}$.

18.-Tìm ước số lẻ lớn nhất của số nguyên dương n. Ví dụ n=100 ước lẻ lớn nhất của 100 là 25.

19.-Tính S(n) = $\sqrt{2+\sqrt{2+\sqrt{2+...\sqrt{2+\sqrt{2}}}}}$ có n dấu căn.

20.-Tính S(n) = $\sqrt{n+\sqrt{n-1+\sqrt{n-2+...\sqrt{2+\sqrt{1}}}}}$ có n dấu căn.

21.-Tính S(n) = $\sqrt{1+\sqrt{2+\sqrt{3+...\sqrt{n-1+\sqrt{n}}}}}$ có n dấu căn.

22.-

(có n dấu phân số)

$$S(n)= \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{\ldots\cfrac{1}{1+\cfrac{1}{1+1}}}}}}$$

23.-Hãy đếm số lượng chữ số của số nguyên dương n.

24.-Hãy tính tổng các chữ số của số nguyên dương n.

25.-Hãy tính tích các chữ số của số nguyên dương n.

26.-Hãy đếm số lượng chữ số lẻ của số nguyên dương n.

27.-Hãy tính tổng các chữ số chẵn của số nguyên dương n.

28.-Hãy tính tích các chữ số lẻ của số nguyên dương n.

29.-Cho số nguyên dương n. Hãy tìm chữ số đầu tiên của n.

30.-Hãy tìm chữ số đảo ngược của số nguyên dương n.

31.-Tìm chữ số lớn nhất của số nguyên dương n.

32.-Tìm chữ số nhỏ nhất của số nguyên dương n.

33.-Hãy kiểm tra số nguyên dương n có toàn chữ số lẻ hay không?

34.-Hãy kiểm tra số nguyên dương n có toàn chữ số chẵn hay không?

MẢNG MỘT CHIỀU

35.-Viết hàm đệ quy thực hiện các bài tập trong chương Mảng Một Chiều.

36.-Viết hàm đệ quy thực hiện các bài tập trong chương Ma Trận.

37.-Viết hàm đệ quy thực hiện các bài tập trong chương Ma Trận Vuông.

38.-Viết hàm đệ quy thực hiện các bài tập trong chương Mảng Các Cấu Trúc bằng phương pháp đệ quy.

39.-Tính tổng tất cả các phần tử trong mảng

40.-Tính tổng các số lẻ có trong mảng.

✍ *Mở rộng:* cho trường hợp: số nguyên tố, số chính phương, …

41.-Tìm giá trị là số lẻ cuối cùng có trong mảng. Nếu không có số lẻ trả về -1.

✍ Mở rộng: cho trường hợp giá trị cuối cùng là số nguyên tố, giá trị cuối cùng là số chính phương, …

42.-Tìm vị trí chứa số lẻ cuối cùng có trong mảng. Nếu không có số lẻ trả về -1.

✍ Mở rộng: cho trường hợp: vị trí cuối cùng chứa số nguyên tố, vị trí cuối cùng số chính phương, …

43.-Kiểm tra xem mảng có được sắp xếp tăng dần hay không?

44.-Kiểm tra xem mảng có chứa số nguyên tố hay không?

MẢNG HAI CHIỀU

45.-Tính tổng tất cả các phần tử trong mảng 2 chiều.

46.-Tính tổng các số lẻ có trong mảng 2 chiều.

✍ Mở rộng: cho trường hợp: số nguyên tố, số chính phương, …

47.-Tìm giá trị là số lẻ cuối cùng có trong mảng 2 chiều. Nếu không có số lẻ trả về -1.

✍ Mở rộng: cho trường hợp giá trị cuối cùng là số nguyên tố, giá trị cuối cùng là số chính phương, …

48.-Tìm vị trí chứa số lẻ cuối cùng có trong mảng 2 chiều. Nếu không có số lẻ trả về -1.

✍ Mở rộng: cho trường hợp: vị trí cuối cùng chứa số nguyên tố, vị trí cuối cùng số chính phương, …

49.-Cho ma trận A (nXm) chỉ chứa các giá trị 0 và -1. Giả sử có định nghĩa về "thành phần liên thông" như sau: một nhóm các ô liên tục (chỉ tính 4 phía trên, dưới, trái, phải) cùng chứa giá trị -1 được xem là 1 "thành phần liên thông". Viết hàm đếm số "thành phần liên thông" có trong ma trận.

Ví dụ:

0	***-1***	***-1***	0	0	0	***-1***
0	***-1***	***-1***	0	0	0	0
0	***-1***	0	0	0	0	0
0	***-1***	***-1***	0	0	0	***-1***
0	0	0	0	0	0	***-1***
0	***-1***	0	***-1***	***-1***	***-1***	***-1***
0	***-1***	0	0	0	0	***-1***

Có 4 thành phần liên thông

0	0	***-1***	0
0	***-1***	***-1***	***-1***
0	0	***-1***	0
0	0	0	0
-1	0	0	0
0	***-1***	0	0
0	0	0	0

Có 3 TPLT

CON TRỎ (Pointer)

8.1. BIẾN VÀ ĐỊA CHỈ

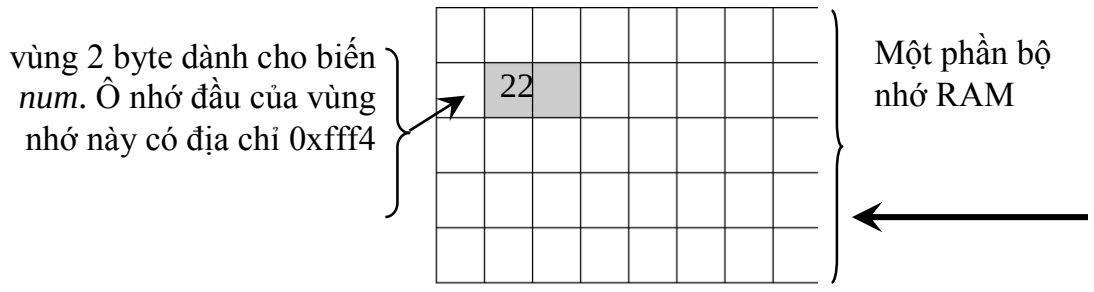
- Mỗi biến sử dụng trong chương trình sẽ có 3 chi tiết liên quan:
  - Kiểu dữ liệu của biến.
  - Giá trị lưu trong biến
  - Địa chỉ lưu trữ của biến trong bộ nhớ.
- Việc đặt tên biến giúp cho chương trình dễ hiểu nhờ tên của biến nói lên ý nghĩa sử dụng của biến trong chương trình. Mỗi tên biến như vậy sẽ tương ứng với một vị trí nào đó trong ô nhớ, và việc xác đính sự tương ứng này sẽ do trình biên dịch và máy tính hoàn tất mỗi khi chương trình được thực hiện.
- Khảo sát chương trình sau:

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int num = 22;
  printf("Gia tri luu trong bien num la: %d\n",num);
  printf("So byte bo nho danh cho bien nay la: %d\n", sizeof(num));
  printf("Dia chi cua o nho danh cho bien num la: %d", &num);
}
```

  - Phép & trong chương trình trên là để lấy địa chỉ của một biến hay còn gọi là tham chiếu tới biến. Con trỏ được sử dụng rất nhiều trong C do:
    - Giúp chương trình ngắn gọn và có hiệu quả hơn các cách khác.
    - Đôi khi là cách duy nhất để biểu diễn tính toán.
  - Chương trình trên khi thực hiện sẽ in lên màn hình:

```
Gia tri luu trong bien num la: 22
So byte trong bo nho danh cho bien nay la: 2
Dia chi cua o nho danh cho bien num la: 2ffdbc
```

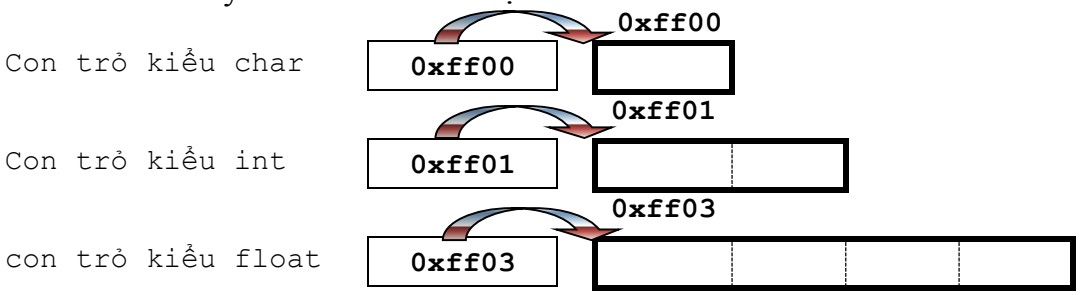
  - Để hiểu rõ hơn về các nội dung đã xuất trong chương trình trên ta hãy xem hình minh họa sau:



  - Trong ví dụ trên:
    - Địa chỉ của biến *num* có thể thay đổi khi thực hiện trên máy tính khác hoặc ở những lần thực hiện sau đó.
    - Số byte có thể là 4 nếu sử dụng Visual Studio .NET

8.2. KHAI BÁO CON TRỎ

- Cũng như tất cả các biến khác, biến con trỏ phải được khai báo trước khi chúng được sử dụng để lưu các địa chỉ. Khi khai báo biến con trỏ chúng ta phải cho biết con trỏ này sẽ lưu địa chỉ của vùng nhớ lưu loại dữ liệu nào.
- Cú pháp khai báo biến con trỏ: `datatype *pointer name;`
- Ví dụ:
  - Khai báo biến con trỏ px kiểu int: `int *px;`
  - Khai báo int *px chỉ ra ba điều:
    - px là một biến con trỏ.
    - vùng nhớ được lưu trong px phải là vùng nhớ lưu trữ một số nguyên kiểu int (hay biến mà px trỏ đến phải là một biến kiểu int).
    - nếu px xuất hiện trong ngữ cảnh *px thì nó cũng được coi là tương đương với việc dùng biến có kiểu int.
- Một biến con trỏ có kích thước hai byte và phụ thuộc vào khai báo ban đầu nó sẽ chứa địa chỉ của vùng nhớ 1byte, 2 byte, 4byte… Trong thực tế dù là con trỏ kiểu char, int , float, hay double thi con trỏ cũng chỉ chứa ô nhớ đầu tiên của vùng nhớ mà nó chỉ tới, nhưng nhờ vào sự khai báo này máy tính sẽ biết cần phải lấy bao nhiêu ô nhớ khi truy xuất tới con trỏ này. Xem hình minh họa sau:



8.3. BIẾN THAM CHIẾU (&)VÀ BIẾN CON TRỎ (*)

- Biến tham chiếu là một biến khi khai báo có thêm dấu `&` phía trước. Biến này cũng là một loại con trỏ nhưng có nhiều hạn chế so với con trỏ.
  - Biến tham chiếu dùng để tham chiếu tới địa chỉ của một biến và chỉ một mà thôi (Địa chỉ lưu trong biến tham chiếu không thể thay đổi được).
  - Địa chỉ mà biến này tham chiếu tới phải được khởi tạo ngay tại thời điểm khai báo ngay từ đầu.
  - Sau khi đã tham khảo tới một biến thì việc sử dụng biến tham chiếu giống như một biến thông thường và những lệnh tác động lên biến tham chiếu này sẽ ảnh hưởng tới biến mà nó tham chiếu tới.
  - Ví dụ:

```
int main()
{
  int n=123, &x=n;
  printf("Gia tri trong vung nho ma x tham chieu toi: %d",x); //123
  x=100;
  printf("\nGia tri cua n hien tai la: %d",n);              //100
  return 0;
}
```

- Biến con trỏ (*): được dùng để lấy giá trị tại địa chỉ mà biến con trỏ đang tới trỏ.

8.4. CON TRỎ VÀ ĐỊA CHỈ

8.4.1. *Phép toán một ngôi &*

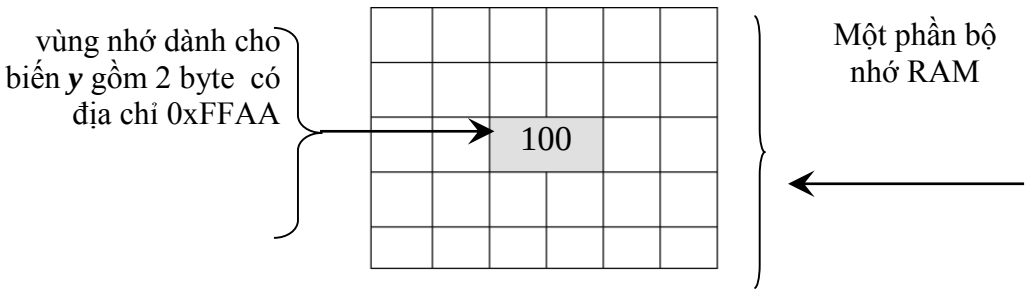
- Vì con trỏ chứa địa chỉ của đối tượng nên ta có thể xâm nhập vào đối tượng gián tiếp qua con trỏ.
- Ví dụ:

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

- Các kiểu dùng sai: Phép toán & chỉ áp dụng được cho các biến và phần tử mảng, do đó các trường hợp dùng sau đây là không hợp lệ:
  - Kết cấu kiểu &(x+1) và &3.
  - Lấy địa chỉ của biến register cũng là sai.

8.4.2. *Phép toán một ngôi **

- Khi toán tử này đi trước một tên biến con trỏ (ví dụ như ✶numaddr, ✶chrpoint) có nghĩa là nói đến giá trị đang lưu tại vùng nhớ có địa chỉ đang lưu trong biến con trỏ.
- Ví dụ giả sử có một biến con trỏ y. giá trị của biến hiện là 0xFFAA (địa chỉ của ô nhớ thứ FFAA trong RAM) giá trị đang lưu trữ tại ô nhớ này là 100. Hình minh họa bộ nhớ lúc này như sau:



như vậy lúc này, giá trị của y là **0xFFAA.**
giá trị của ✶**y** là 100 (giá trị lưu tại vùng nhớ **0xFFAA**).

Đây là một cách lấy giá trị gián tiếp, theo cách này máy tính phải dò đến hai địa chỉ thì mới nhận được giá trị 100.(đầu tiên dò đến địa chỉ chứa trong y, sau đó từ địa chỉ này mới dò đến ô nhớ có địa chỉ tương ứng để nhận địa chỉ.

- Công dụng của phép toán ✶:
  - Làm việc trên các phần tử của mảng.
  - Tạo và xóa các biến động khi chương trình đang thực hiện.
- Ví dụ:

```
void main()
{
    int x=7, y=5;
    int *px, *py;
    px=&x;
    printf("y= %d\n",y);                    //y=5
```

```
        printf("px chua gia tri %d\n", *px);    //7
        printf("px= %x\n", px);                 //px=9ffbe8
        y=*px;                                  //7
        printf("Dia chi y= %d\n",y);            //aafbd8
        printf("y= %d\n",y);                    //7
}
```

8.4.3. *Độ ưu tiên của các phép toán một ngôi & và **

Các phép toán một ngôi & và * có mức ưu tiên cao hơn các phép toán số học.

8.5. CÁC PHÉP TOÁN TRÊN CON TRỎ

8.5.1. *Cấp phát và thu hồi bộ nhớ*

Để xin cấp phát bộ nhớ cho một biến con trỏ ta có thể sử dụng các hàm:

malloc, calloc, realloc

Hoặc toán tử

new

8.5.2. *Thu hồi bộ nhớ*

Để thu hồi bộ nhớ đã cấp phát cho một biến con trỏta có thể sử dụng hàm

free

hoặc toán tử

delete

8.5.3. *Con trỏ cấu trúc*

Để truy xuất đến một thành phần của biến con trỏ cấu trúc ta dùng tóan tử mũi tên.

->

8.5.4. *Địa chỉ ô nhớ*

Để xuất một địa chỉ ô nhớ ta sử dụng mã định dạng là:

%p

8.5.5. *Phép gán*

Con trỏ dùng để lưu địa chỉ. Mỗi kiểu địa chỉ cần có kiểu con trỏ tương ứng. Phép gán địa chỉ cho con trỏ chỉ có thể thực hiện được khi kiểu địa chỉ phù hợp với kiểu con trỏ.

8.5.5.1. *Lưu trữ địa chỉ của một biến vào một biến con trỏ đã được khai báo phù hợp*

```
int m;
int *d= &m;       // lưu trữ địa chỉ biến m vào con trỏ d
float a[30], *pa, (*pm)[30];
pa=a;             // hợp lệ
pm=&a;            // hợp lệ
pm=a;             // KHƠNG hợp lệ ⇒ bo lỗi khi bin dịch
```

8.5.5.2. *Xuất giá trị*

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

8.5.5.3. *Gán giá trị cho biến từ biến con trỏ (biến con trỏ xuất hiện bên về <u>phải</u> của biểu thức)*

```
int x=4;
int y=*px+1;      // ⇔ y= 4+1=5

d=sqrt((double) *px); // ⇔ d = √x =√4 =2. Do hàm sqrt chỉ nhận
                      //tham số kiểu double nên phải thực hiện ép kiểu.
```

8.5.5.4. *Gán giá trị cho biến con trỏ: (biến con trỏ xuất hiện bên về <u>trái</u> của biểu thức)*

```
    *px=0;   // x = 0
```

```
*px+=1;  // tăng giá trị của biến x lên thêm 1 đơn vị (x=1)
(*px)++; // tăng giá trị của biến x lên thêm 1 đơn vị (x=2)
```

Lưu ý: Các dấu ngoặc đơn ở câu lệnh cuối là cần thiết, nếu không thì biểu thức sẽ tăng px thay cho tăng ở biến mà nó trỏ tới vì phép toán một ngôi như * và ++ được tính từ phải sang trái.

8.5.5.5. *Gán giá trị giữa 2 biến con trỏ của cùng kiểu dữ liệu: (biến con trỏ xuất hiện ở cả 2 vế của biểu thức)*

```
int *px, *py;
py=px;          // lệnh này sẽ sao nội dung của px vào py,
                // nghĩa là làm cho py trỏ tới nơi mà px trỏ.
```

8.5.5.6. *Ép kiểu*

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

8.5.6. *Phép tăng giảm địa chỉ*

8.5.6.1. *Ví dụ 1: (dùng trên mảng 1 chiều)*

```
float x[30],*px;
px=&x[10];        // px trỏ tới phần tử x[10]
```

Cho con trỏ px là con trỏ float trỏ tới phần tử x[10]. Kiểu địa chỉ float là kiểu địa chỉ 4 byte, nên các phép tăng giảm địa chỉ được thực hiện trên 4 byte. Vì thế:

```
px+i            // px trỏ tới phần tử x[10+i]
px-i            // pxtrỏ tới phần tử x[10-i]
```

8.5.6.2. *Ví dụ 2: (dùng trên mảng 2 chiều) Giả sử ta khai báo*

```
float b[40][50];
```

Khai báo trên cho ta một mảng b gồm các dòng 50 phần tử kiểu số thực. Kiểu địa chỉ của b là 50*4=200 byte.

Do vậy:

```
b trỏ tới đầu dòng thứ nhất ( phần tử b[0][0]).
b+1 trỏ tới đầu dòng thứ hai ( phần tử b[1][0]).
..........
b+i trỏ tới đầu dòng thứ i ( phần tử b[i][0]).
```

8.5.7. *Phép truy cập bộ nhớ*

Con trỏ float truy nhập tới 4 byte, con trỏ int truy nhập 2 byte, con trỏ char truy nhập 1 byte. Giả sử ta có các khai báo:

Ví dụ 1:

```
float *pf;
int *pi;
char *pc;
```

Khi đó:

- Nếu con trỏ pf trỏ đến byte thứ 100 thì *pf biểu thị vùng nhớ 4 byte liên tiếp từ byte 100 đến 103.
- Nếu con trỏ pi trỏ đến byte thứ 100 thì *pi biểu thị vùng nhớ 2 byte liên tiếp từ byte 100 đến 101.
- Nếu con trỏ pc trỏ đến byte thứ 100 thì *pc biểu thị vùng nhớ 1 byte chính là byte 100.

8.5.8. *Phép so sánh*

Cho phép so sánh các con trỏ cùng kiểu, ví dụ nếu p1 và p2 là các con trỏ cùng kiểu thì nếu :

p1<p2 nếu địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới.

p1=p2 nếu địa chỉ p1 trỏ tới cũng là địa chỉ p2 trỏ tới.
p1>p2 nếu địa chỉ p1 trỏ tới cao hơn địa chỉ p2 trỏ tới.

8.5.9. *Một số ví dụ*

- Ví dụ 1: Phân biệt giữa địa chỉ và giá trị của biến:

```
int main()
{
    int  *addr, n=158, m=22;
    addr=&n;
    printf("Dia chi bien addr dang tro toi: %u\n",addr);
    printf("Gia tri tai vung nho addr dang tro toi: %d\n", *addr);
    addr=&m;
    printf("Dia chi bien addr dang tro toi: %u\n",addr);
    printf("Gia tri tai vung nho addr dang tro toi: %d\n", *addr);
    return 0;
}
```

- Ví dụ 2: Đoạn chương trình tính tổng các số thực dùng phép so sánh con trỏ:

```
int main()
{
  float a[5]={2,4,6,8,10},*p,*pcuoi,tong=0.0;
  int s=0,n=5;

  pcuoi=a+n-1;    //pcuoi chi den phan tu cuoi cua mang
  for (p=a ; p<=pcuoi ; p++)
    s+=*p;
  printf("Tong cac phan tu co trong mang= %d",s);
  return 0;
}
```

- Ví dụ 3: Dùng con trỏ char để tách các byte của một biến nguyên:

```
int main()
{ unsigned int n=16689; /* 01000001 00110001*/
  char *pc;                    //       A1
  pc=(char*)(&n);
  printf("%c",*pc);            //       1
  pc++;
  printf("%c",*pc);            //       A
  return 0;
}
```

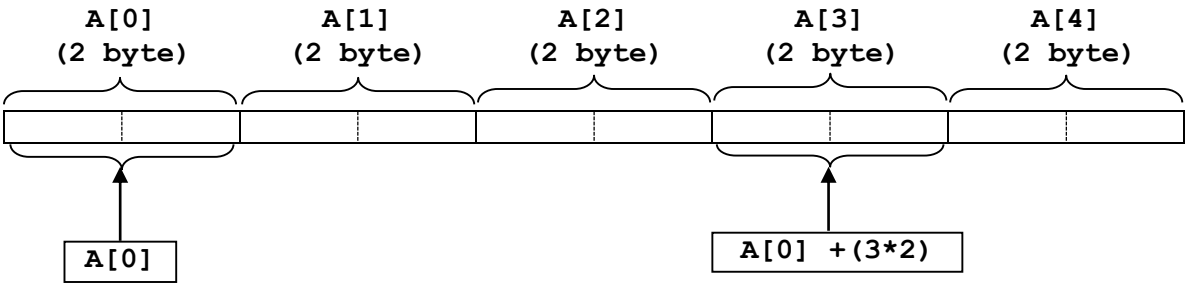
8.6. CON TRỎ VÀ MẢNG MỘT CHIỀU

8.6.1. *Mối liên hệ giữa con trỏ và mảng 1 chiều*

8.6.1.1. *Xác định địa chỉ của các phần tử trong mảng 1 chiều*

Cho mảng 1 chiều A gồm 5 phần tử, để truy xuất phần tử A[3] (giả sử A là một mảng số nguyên) thì máy tính sẽ xác định địa chỉ của phần tử này như sau:

*&A[3]= &A[0] + (3*2) giả sử kiểu int chiếm 2 byte trong bộ nhớ.*



8.6.1.2. Mối liên hệ

- Trong ngôn ngữ C, khi khai báo một mảng thì tên của mảng sẽ chứa địa chỉ khởi đầu của mảng đó và ta không thể thay đổi địa chỉ chứa trong tên mảng này vì vậy tên mảng được gọi là một *hằng con trỏ*.
- Khi thao tác trên mảng người ta thường truy xuất các phần tử thông qua chỉ số mảng để người lập trình cảm thấy nhẹ nhàng bởi cách truy xuất này giúp che đậy việc sử dụng địa chỉ bên trong máy tính.
- Bảng sau mô tả ý nghĩa giữa hình thức truy xuất phần tử mảng thông qua chỉ số và thông qua con trỏ:

Phần tử thứ i của mảng Arr	*Địa chỉ của phần tử thứ i*		*Giá trị của phần tử thứ i*	
	Sử dụng toán tử tham chiếu &	*Sử dụng con trỏ addr*	*Sử dụng chỉ số*	*Sử dụng con trỏ addr*
0	&A[0]	addr	A[0]	✶(addr)
1	&A[1]	addr+1	A[1]	✶(addr+1)
2	&A[2]	addr+2	A[2]	✶(addr+2)
3	&A[3]	addr+3	A[3]	✶(addr+3)

- Lưu ý: `✶(addr+3)` hoàn toàn có ý nghĩa khác với `✶addr+3`.

  `✶(addr+3)` = *giá trị phần tử* `Arr[3]`

  `✶addr+3` = *giá trị của phần tử* `Arr[0]+3`

8.6.1.3. Ví dụ: ứng dụng cách làm của máy tính để truy xuất đến các phần tử của mảng thông qua con trỏ.

```
int main()
{
    const int SIZE=5;
    int i, addr, Arr[SIZE]={98,87,76,65,54};
    addr=Arr[0];
    printf("Duyet mang dua tren chi so mang: ");
    for (i=0;i<SIZE;i++)
        printf("%5d",Arr[i]);
    printf("\nDuyet mang dua tren con tro: ");
    for (i=0;i<SIZE;i++)
        printf("%5d",*(Arr+i));
    return 0;
}
```

8.6.2. Sử dụng con trỏ trên mảng 1 chiều

8.6.2.1. Phép toán lấy địa chỉ

Phép toán này chỉ áp dụng cho các phần tử của mảng một chiều. Giả sử ta có khai báo:

```
double b[20];
```

Khi đó phép toán:

```
&b[9]
```

sẽ cho địa chỉ của phần tử b[9].

8.6.2.2. Tên mảng là một hằng địa chỉ

Khi khai báo: `float a[10];`

Máy sẽ bố trí bố trí cho mảng a mười khoảng nhớ liên tiếp, mỗi khoảng nhớ là 4 byte. Như vậy, nếu biết địa chỉ của một phần tử nào đó của mảng a, thì ta có thể dễ dàng suy ra địa chỉ của các phần tử khác của mảng.

Trong C ta có: `a` tương đương với `&a[0]`

`a+i` tương đương với `&a[i]`
`*(a+i)` tương đương với `a[i]`

8.6.2.3. Phép toán số học trên con trỏ:

- Giá trị lưu trong biến con trỏ là một địa chỉ, vì vậy khi ta thực hiện phép cộng hoặc trừ trên một biến con trỏ ta sẽ nhận được một địa chỉ khác trong biến con trỏ này.

Ví dụ: có khai báo sau

```
int nums[100];
int *point;
point=&nums[0]; //giá trị của point là địa chỉ của phần tử nums[0]
```

Hoặc

```
point=num;
```

Nếu có lệnh gán

```
point=point+2;
```

lúc này giá trị của point là địa chỉ của phần tử nums[2] (giá trị trong con trỏ point đã thay đổi).

- Khi thực hiện cộng hoặc trừ một giá trị trên biến con trỏ thì giá trị này phải là hằng, biến hoặc biểu thức có giá trị kiểu số nguyên, và phải chắc rằng sau khi cộng hoặc trừ thì giá trị của biến con trỏ vẫn trỏ tới một địa chỉ có nghĩa.

Ví dụ: tiếp theo ví dụ trước, nếu ta sau đó có lệnh **point**=**point**+200, lệnh này làm thay đổi giá trị trong **point** và địa chỉ mới là một địa chỉ không có nghĩa đối với mảng **nums.**

- Cũng có thể áp dụng các phép toán tăng/giảm (++/--) một ngôi trên biến con trỏ.

Ví dụ:

```
int main()
{
     const int SIZE=5;
     int i, *p, Arr[SIZE]={98,87,76,65,54};
     p=&Arr[0]; // point=Arr
     printf("DUYET MANG THEO NHIEU CACH KHAC NHAU");
     printf("\n(a).-Dua tren ten mang:");
     for (i=0;i<SIZE;i++)
         printf ("%5d",*(Arr+i));
     printf("\n(b).-Dua tren dia chi co dinh ma bien con tro
                                                  p dang giu:");
     for (i=0;i<SIZE;i++)
           printf ("%5d",*(p+i));
     printf("\n(c).-Dua tren viec thay doi dia chi trong
                              bien con tro p qua moi lan lap");
     for (i=0;i<SIZE;i++)
           printf ("%5d",*p++);
     return 0;
}
```

Lưu ý: Nếu đặt đoạn lệnh ***(c)*** lên trên cùng thì kết quả của đoạn chương trình ***(b)*** sẽ không còn đúng nữa.

8.6.2.4. Con trỏ trỏ tới các phần tử của mảng một chiều:

8.6.2.4.1. Khi con trỏ p trỏ tới phần tử a[k] thì:

- `p+i` trỏ tới phần tử thứ `i` sau `a[k]`, có nghĩa là nó trỏ tới `a[k+i]`.
- `p-i` trỏ tới phần tử thứ `i` trước `a[k]`, có nghĩa là nó trỏ tới `a[k-i]`.
- `*(p+i)` tương đương với `pa[i]`.

Như vậy, sau hai câu lệnh :

```
float a[20],*p;
```

```
int i=2;
p=a;
```

thì bốn cách viết sau đều lấy giá trị của phần tử thứ i trong mảng a:

```
a[i]      *(a+i)      p[i]      *(p+i)
```

8.6.2.4.2. Ví dụ: tính tổng các giá trị có trong mảng:

- Cách 1:

```
int main()
{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    for (i=0;i<SIZE;++i)
        tong+=A[i];
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}
```

- Cách 2:

```
int main()
{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    p=A;
    for (i=0;i<SIZE;++i)
        tong+=p[i];
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}
```

- Cách 3:

```
int main()
{
    const int SIZE=5;
    int tong=0, i, *p, A[SIZE]={8,7,6,5,4};
    p=A;
    for (i=0;i<SIZE;++i)
        tong+=*(p+i);
    printf("Tong cac phan tu mang la: %d",tong);
    return 0;
}
```

- *Chú ý*: Mảng một chiều và con trỏ tương ứng phải cùng kiểu.

8.7. BÀI TẬP

1. Hãy viết đoạn chương trình để khai báo biến số nguyên a và xuất ra địa chỉ ô nhớ được cấp phát cho biến này khi chương trình chạy.

2. Hãy khai báo biến con trỏ cấu trúc phân số có tên là p.

3. Hãy cho biết đoạn chương trình dưới đây câu lệnh nào đúng, câu lệnh nào sai:

```
int a;
int *p;
a = 5;
p =7;
```

4. Hãy cho biết trong đoạn chương trình dưới đây câu lệnh nào đúng, câu lệnh nào sai:

```
int a;
int *p;
a = 5;
p = a;
```

5. Hãy cho biết trong đoạn chương trình dưới đây câu lệnh nào đúng, câu lệnh nào sai:

```
int a;
int *p;
a = 5;
p = &a;
```

6. Hãy cho biết kết quả thực hiện của đoạn chương trình dưới đây:

```
int a;
int *p;
a = 5;
p =&a;
a++;
printf("\nGiá trị của biến a: %d", a);
printf("\nGiá trị tại địa chỉ của biến con trỏ p đang trỏ tới: %d", *p);
```

7. Hãy cho biết kết quả thực hiện của đoạn chương trình dưới đây:

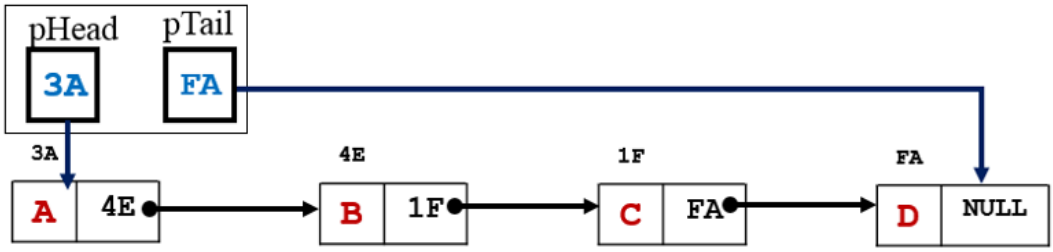
```
int a;
int *p;
a = 5;
p =&a;
a++;
*p+ +;
printf("\nGiá trị của biến a: %d", a);
printf("\nGiá trị tại địa chỉ của biến con trỏ p đang trỏ tới:%d", *p);
```

8. Hãy khai báo biến con trỏ p các số nguyên và viết lệnh cấp phát không gian bộ nhớ để biến con trỏ p có thể chứa 100 số nguyên. Sau đó viết lệnh thu hồi không gian bộ nhớ đã cấp phát cho con trỏ.

9. Viết hàm hoán vị 2 số nguyên bằng cách sử dụng kỹ thuật con trỏ.

DANH SÁCH LIÊN KẾT

9.1. DANH SÁCH LIÊN KẾT (DSLK) ĐƠN (*Linked List*)



9.1.1. *Khai báo cấu trúc của một node*

- Gồm hai phần:
  - *Phần Data*: lưu trữ thông tin của một NODE
  - *Phần con trỏ pNext*: giữ địa chỉ (trỏ đến) phần tử kế tiếp.
- Định nghĩa cấu trúc quản lý dữ liệu trên mỗi node:

```
typedef  struct  tagNode
{
      Data  Info; //Data là kiểu đã định nghĩa trước
      struct tagNode* pNext; //con trỏ chỉ đến cấu trúc node
}NODE;
```

- Định nghĩa thêm một cấu trúc để quản lý DSLK đơn như sau:

```
typedef  struct tagList
{
    NODE* pHead;
    NODE* pTail;
}LIST;
```

- Ví dụ
  - ***Khai báo một node của DSLK đơn là các số nguyên***

```
typedef  struct  tagNode
{
    int  Info;   //Data là kiểu số nguyên
    struct tagNode* pNext;  //con trỏ chỉ đến cấu trúc node
}NODE;
```

  - ***Khai báo một node của DSLK đơn lưu trữ hồ sơ sinh viên***
    - Bước 1: định nghĩa 1 kiểu dữ liệu trước là sinh viên, để lưu trữ 1 sinh viên

```
typedef  struct  SinhVien
{
    char Ten[30];
    int MaSV;
}SV;
```

    - Bước 2: định nghĩa 1 node để lưu trữ 1 sinh viên trong DSLK đơn

```
typedef  struct  tagNode
{
    SV Info;   //Data là kiểu SV
    struct tagNode* pNext;  //con trỏ chỉ đến cấu trúc node
}NODE;
```

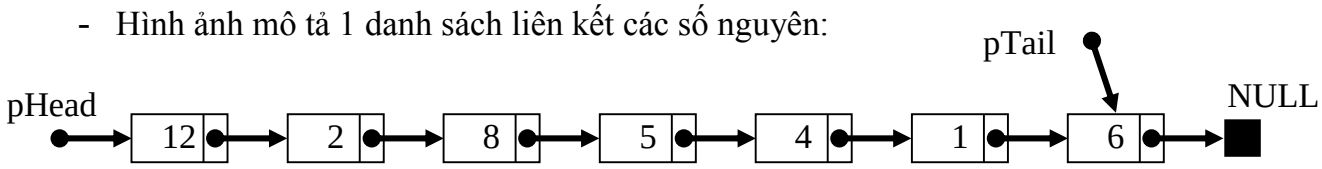
- ***Khai báo cấu trúc dữ liệu cho dslk đơn các phân số***

```
typedef  struct phanso
{
     int  tu;
     int  mau;
} PHANSO;
typedef  struct node
{
     PHANSO Info;
     struct nodẽ* pNext;
} NODE;
```

- ***Khai báo cấu trúc dữ liệu cho DSLK đơn các số phức***

```
struct sophuc
{
  float thuc;
  float ao;
} SOPHUC;
typedef struct node
{
  SOPHUC Info;
  struct node* pNext;
} NODE;
```

- Hình ảnh mô tả 1 danh sách liên kết các số nguyên:



- Khi sử dụng, nên khai báo thêm hai biến như sau:

```
NODE* new_ele; //giữ địa chỉ của một phần tử mới được tạo
int x;         //lưu giữ thông tin khi Data là số nguyên
SV s;          //lưu giữ thông tin khi Data là sinh viên
```

9.1.2. *Các thao tác cơ bản (dùng DSLK với kiểu dữ liệu của Data là int để minh họa)*

9.1.2.1. *Khởi tạo DSLK*

```
void init (LIST &L)
{
     L.pHead = NULL;
     L.pTail  = NULL;
}
```

9.1.2.2. *Tạo ra một NODE mới cho DSLK đơn*

- Với Data chứa số nguyên x

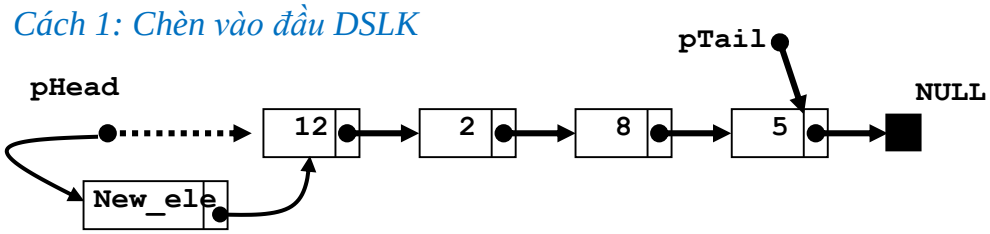
```
NODE* GetNode(int  x)
{    NODE*  p;
     p= new NODE;      //cấp phát vùng nhớ cho phần tử
     if(p= =NULL)
     {    printf("\n Không đủ bộ nhớ, nhấn ENTER để thoát");
          getch();
          exit(1);
     }
     p->Info=x;        //gán thông tin cho phần tử p
     p->pNext=NULL;
     return p;
}
```

- Với Data chứa thông tin về một sinh viên:

```
NODE* GetNode(SV  x)
{
     NODE*  p;
     p= new NODE;      //cấp phát vùng nhớ cho phần tử
     if(p == NULL)
     {
          printf("\n Không đủ bộ nhớ, nhấn ENTER để thoát");
          getch();
          exit(1);
     }
     p->Info=x;        //gán thông tin cho phần tử p
     p->pNext=NULL;
     return p;
}
```

9.1.2.3. Chèn một phần tử vào DSLK

9.1.2.3.1. Cách 1: Chèn vào đầu DSLK

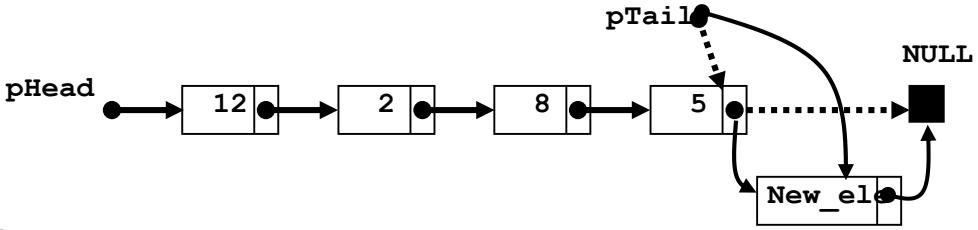


- Thuật toán:
  - Bước 1: nếu danh sách rỗng thì gán `pHead = pTail = new_ele`
  - Bước 2: ngược lại
    - B21: `new_ele->next=head`
    - B22: `head=new_ele`

- Cài đặt:

```
void AddFirst(LIST &L,NODE* new_ele)
{
     if(L.pHead= =NULL) //DSLK rỗng
          L.pHead=L.pTail=new_ele;
     else
     {
          new_ele->pNext=L.pHead;
          L.pHead=new_ele;
     }
}
```

9.1.2.3.2. Cách 2: chèn vào cuối danh sách

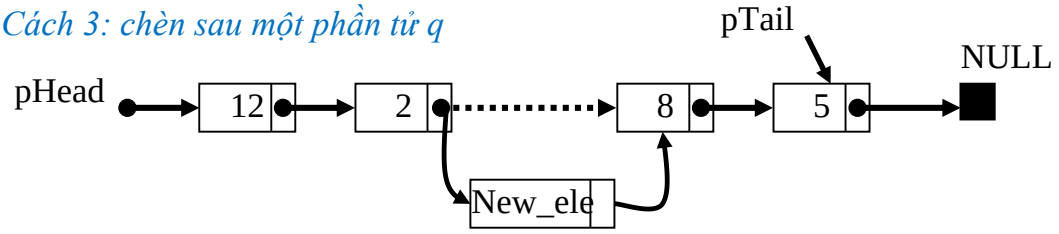


- Thuật toán:
  - Bước 1: nếu danh sách rỗng thì gán `pHead = pTail = new_ele`
  - Bước 2: ngược lại
    - b21: `pTail->next = new_ele`
    - b22: `pTail = new_ele`

- Cài đặt:

```
void AddTail(LIST &L,NODE* new_ele)
{
    if(L.pHead == NULL) //DSLK rỗng
        L.pHead = L.pTail = new_ele;
    else
    {
        L.pTail->pNext = new_ele;
        L.pTail = new_ele;
    }
}
```

9.1.2.3.3. *Cách 3: chèn sau một phần tử q*



- Thuật toán:

  nếu (q!= NULL) thì
  - **new_ele->next=q->next**
  - **q->next=new_ele**

- Cài đặt:

```
void AddAfter(LIST &L, NODE* q, NODE* new_ele)
{
    if(q != NULL)
    {
        new_ele->pNext=q->pNext;
        q->pNext=new_ele;
        if(q == L.pTail)
            L.pTail=new_ele;
    }
    else
        AddFirst(L,new_ele);
}
```

9.1.2.4. ***Viết hàm nhập DSLK*** *(các node vào sau sẽ nằm ở đầu DSLK)*

```
void input (LIST &l)
{
    int n;
    printf ("Nhap so luong node:");
    scanf ("%d",&n);
    init (1);
    for (int i=1; i<=n; i++)
    {
        KDL x;
        nhap (x)
        NODE *p = getnode (x);
        addhead (l,p);
    }
}
```

9.1.2.5. *Duyệt DSLK*

- Thuật toán:
  - Bước 1: nếu danh sách rỗng thì thông báo danh sách rỗng
  - Bước 2: ngược lại
    - B21: **p=head**
    - B22: trong khi chưa hết danh sách thì
      - xử lý theo yêu cầu phần tử p đang xét.
      - **p=p->pnext** xử lý phần tử kế tiếp
- Cài đặt: ví dụ duyệt danh sách để xuất ra màn hình

```
void XuatDS(LIST L)  //hàm xuất danh sách các số nguyên
{
     if(L.pHead= =NULL) //DSLK rỗng
          printf("Danh sách rỗng");
     else
     {
          NODE* p=L.pHead;
          while(p)
          {
               printf(%5d",p->Info);
               p=p->pNext;
          }
     }
}
```

9.1.2.6. *Tìm một phần tử trong DSLK*

- Thuật toán:
  - Bước 1: **p=head;** // int i=0;
  - Bước 2: trong khi chưa hết danh sách và chưa tìm thấy x thì **p=p->next** để tìm phần tử kế tiếp
  - Bước 3: trả về con trỏ p
- Cài đặt: ví dụ tìm một phần tử x có trong danh sách không, trả về phần tử x.

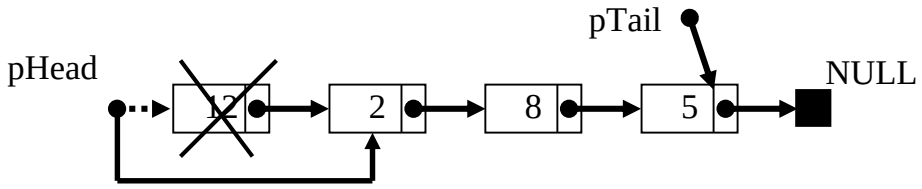
```
NODE*  Search(LIST L, int x)  //tìm các số nguyên trên DSLK
{
     NODE* p=L.pHead;
     while(p  && p->Info!=x)
          p=p->pNext;
     return p;
}
```

- Cài đặt: ví dụ tìm một phần tử x có trong danh sách không, trả về phần tử đứng trước x

```
NODE*SearchBefore(LIST L,int x)//tìm các số nguyên trên DSLK
{
     NODE* p=L.pHead;
     while(p->pNext  && p->pNext->Info!=x)
          p=p->pNext;
     return p;
}
```

9.1.2.7. Hủy một phần tử khỏi DSLK

9.1.2.7.1. Hủy đầu DSLK: trả về giá trị của phần tử bị hủy khỏi DSLK



- Thuật toán:

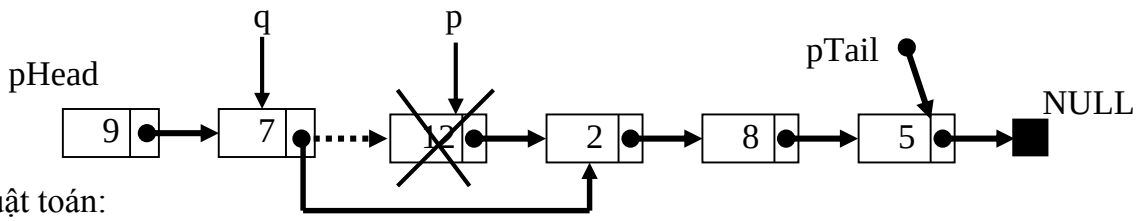
  Nếu DSLK không rỗng thì:

  - Bước 1: `p=pHead;` // p là phần tử cần hủy
  - Bước 2: `pHead` = `pHead` ->next; // tách p ra khỏi DSLK
  - Bước 3: `delete p;` //hủy bộ nhớ đã cấp phát cho NODE mà p đang trỏ đến.
  - Bước 4: nếu `pHead=NULL` thì `pTail=NULL`;//sau khi hủy, DSLK rỗng (hủy DSLK chỉ có 1 phần tử duy nhất).

- Cài đặt:

```
int  RemoveHead(LIST &L)  //hàm hủy đầu danh sách
{
     if(L.pHead)//DSLK không rỗng
     {
          NODE* p=L.pHead;
          int x=p->Info;
          L.pHead=L.pHead->pNext;
          delete p;
          if(L.pHead= = NULL) L.pTail=NULL;
          return x;
     }
     return  NULLDATA;//giá trị báo cho biết không hủy được
}
```

9.1.2.7.2. Hủy phần tử đứng sau phần tử q:



- Thuật toán:

  Nếu `q!=NULL` thì:

  - Bước 1: `p=q->next;` // p là phần tử cần hủy
  - Bước 2: nếu `p!=NULL` thì
    - b21: `q->next=p->next;` //tách p ra khỏi DSLK
    - b22: `delete p`; //hủy bộ nhớ đã cấp phát cho NODE mà p đang trỏ đến.

- Cài đặt:

```
int RemoveAfter(LIST &l,NODE* q) //hàm hủy phần tử sau q
{
     if(q  && q->pNext)
     {
          NODE* p=q->pNext;
          int x=p->Info;
          if(p= = L.pTail) L.pTail=q;
```

```
            q->pNext=p->pNext;
            delete p;
            return x;
        }
        return  NULL; //giá trị báo cho biết không hủy được
    }
```

9.1.2.7.3. *Hủy 1 phần tử có giá trị k:*

- Thuật toán:
  - Bước 1: nếu phần tử đầu có giá trị k thì gọi hàm hủy đầu
  - Bước 2: ngược lại gọi hàm SearchBefore tìm phần tử q đứng trước phần tử có giá trị k.
    - nếu `q!=Tail` thì gọi hàm `RemoveAfter`.

- Cài đặt:

```
int  RemoveNode(LIST &l,int k)//hàm hủy phần tử có giá trị k
{
    if(L.pHead->Info= = k)
    return RemoveHead(l);
    NODE* q=SearchBefore(l,k);
    if(q!=L.pTail)
        return RemoveAfter(l,q);
    return  NULLDATA; //giá trị báo cho biết không hủy được
}
```

9.1.2.7.4. *Hủy toàn bộ danh sách*

- Thuật toán:

  Trong khi chưa hết danh sách thì
  - Bước 1: `p=head;`
  - Bước 2: `head=head->next;`
  - Bước 3: `delete p;`
  - Bước 4: `Tail=NULL;`

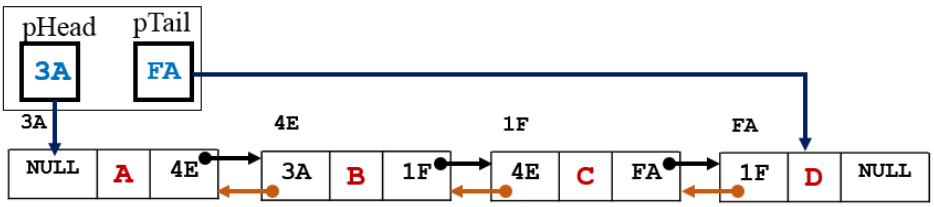
- Cài đặt:

```
void  RemoveLIST(LIST &l)
{
    while(L.pHead)
    {
        NODE* p=L.pHead;
        L.pHead=L.pHead->pNext;
        delete p;
    }
    L.pTail=NULL;
}
```

9.2. CÁC LOẠI DSLK KHÁC

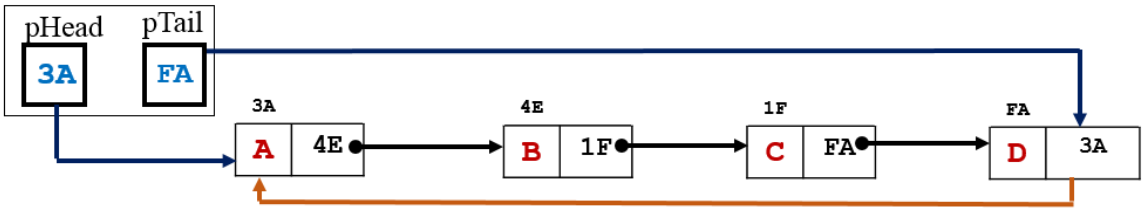
9.2.1. *Danh sách liên kết đôi (Double-Linked List)*

Mỗi phần tử liên kết với phần tử đứng trước và sau nó trong danh sách



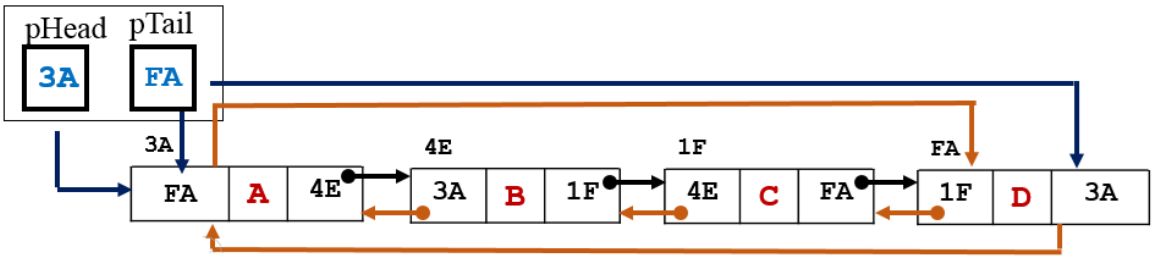
9.2.2. *Danh sách liên kết đơn vòng (Circular-Linked List):*

Con trỏ PNext của phần tử cuối sẽ liên kết với phần tử đầu danh sách.



9.2.3. *Danh sách liên kết đôi vòng (Circular-Double Linked List)*

- Con trỏ Next của phần tử cuối sẽ liên kết với phần tử đầu danh sách.
- Con trỏ Prev của phần tử đầu sẽ liên kết với phần tử cuối danh sách.



9.3. BÀI TẬP

9.3.1. *Xây dựng các hàm từ prototype đã cho*

1. Kiểm tra danh sách có rỗng hay không?

```
int IsEmpty (LIST L);
```

2. Thêm một node p vào trước node q trong danh sách.

```
void AddBefore (LIST &L, NODE *q, NODE *p);
```

3. Nối hai DSLK đơn L1 vào cuối DSLK L2.

```
LIST cong (LIST |1, LIST |2);
```

9.3.2. *Khai báo DSLK*

4. Khai báo cấu trúc dữ liệu cho DSLK đơn các số nguyên.

5. Khai báo cấu trúc dữ liệu cho DSLK đơn các số thực.

6. Khai báo cấu trúc dữ liệu cho DSLK đơn các tọa độ điểm trong mặt phẳng Oxy.

7. Khai báo cấu trúc dữ liệu cho DSLK đơn các học sinh. Biết rằng học sinh gồm những thành phần thông tin như sau:

- Họ tên: tối đa 30 ký tự.
- Điểm toán: kiểu số nguyên.
- Điểm văn: kiểu số nguyên.
- Điểm trung bình: kiểu số thực.

8. Khai báo cấu trúc dữ liệu cho DSLK đơn các thí sinh. Giả sử thí sinh gồm những thành phần thông tin như sau:

- Mã thí sinh: tối đa 5 ký tự.
- Họ tên: tối đa 30 ký tự.
- Điểm toán: kiểu số thực.
- Điểm lý: kiểu số thực.
- Điểm hóa: kiểu số thực.
- Điểm tổng cộng: kiểu số thực.

9. Khai báo cấu trúc dữ liệu cho DSLK đơn các phòng trong khách sạn (PHONG). Giả sử phòng khách sạn gồm những thành phần thông tin như sau:
- Mã phòng: tối đa 5 ký tự.
- Tên phòng: tối đa 30 ký tự.
- Đơn giá thuê: kiểu số thực.
- Số lượng giường: kiểu số nguyên.
- Tình trạng phòng: 0 rảnh, 1 bận.

9.3.3. *Tạo node cho DSLK đơn*

10. Viết hàm tạo node trong DSLK đơn các số phức.

11. Viết hàm tạo node trong DSLK đơn các số nguyên.

12. Viết hàm tạo node trong DSLK đơn các số thực.

13. Viết hàm tạo node trong DSLK đơn các tọa độ điểm trong mặt phẳng Oxy.

14. Viết hàm tạo node trong DSLK đơn các học sinh.

15. Viết hàm tạo node trong DSLK đơn các thí sinh.

16. Viết hàm tạo node trong DSLK đơn các phòng trong khách sạn (PHONG).

17. Viết hàm tạo node trong DSLK đơn các đường tròn.

9.3.4. *Duyệt DSLK*

18. Viết hàm đếm số lượng node trong một DSLK đơn.

19. Viết hàm xuất DSLK đơn các phân số.

20. Viết hàm xuất DSLK đơn các số phức.

21. Viết hàm xuất DSLK đơn các số nguyên.

22. Viết hàm xuất DSLK đơn các số thực.

23. Viết hàm xuất DSLK đơn tọa độ điểm trong mặt phẳng Oxy.

24. Viết hàm đếm số lượng phân số dương trong DSLK đơn các phân số.

25. Viết hàm đếm số lượng phân số âm trong DSLK đơn các phân số.

26. Viết hàm tìm phân số dương đầu tiên trong DSLK đơn các phân số.

27. Viết hàm tìm phân số dương cuối cùng trong DSLK đơn các phân số.

28. Viết hàm kiểm tra DSLK các phân số có toàn phân số dương hay không?

29. Viết hàm kiểm tra DSLK đơn các phân số có tồn tại phân số âm hay không?

30. Viết hàm tìm phân số lớn nhất trong DSLK đơn các phân số.

31. Viết hàm tìm phân số nhỏ nhất trong DSLK đơn các phân số.

32. Viết hàm sắp xếp các phân số tăng dần trong DSLK đơn các phân số.

33. Viết hàm sắp xếp các phân số dương tăng dần trong DSLK đơn các phân số.

9.3.5. *Thao tác trên những cấu trúc dữ liệu có phần Data khác nhau*

34. Viết chương trình thực hiện các yêu cầu sau trên DSLK đơn các nhân viên:

a. Khai báo cấu trúc dữ liệu của một DSLK đơn các nhân viên. Biết rằng thông tin của một nhân viên bao gồm: họ tên (30 ký tự), ngày sinh (kiểu ngày), lương (số thực), giới tính (0. Nữ, 1. Nam).
b. Cài đặt tất cả các thao tác cơ bản cho DSLK các nhân viên.
c. Liệt kê các nhân viên trên 40 tuổi trong DSLK.
d. Đếm số lượng nhân viên có lương lớn hơn 1.000.000 đồng trong DSLK.
e. Viết hàm sắp xếp các nhân viên giảm dần theo năm sinh (không quan tâm ngày và tháng sinh).

35. Viết chương trình thực hiện các yêu cầu sau trên DSLK đơn các học sinh:
a. Khai báo cấu trúc dữ liệu của một DSLK đơn các học sinh. Biết rằng thông tin của một học sinh bao gồm: họ tên (30 ký tự), điểm toán, điểm văn và điểm trung bình (tất cả là số thực).
b. Cài đặt tất cả các thao tác cơ bản cho DSLK các học sinh.
c. Liệt kê các học sinh có điểm toán nhỏ hơn 5 trong DSLK.
d. Đếm số lượng học sinh có điểm tóan và điểm văn lớn hơn 8 điểm trong DSLK.
e. Viết hàm sắp xếp các học sinh giảm dần theo điểm trung bình.

36. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu của một DSLK đơn tọa độ các điểm trong mặt phẳng Oxy.
b. Cài đặt tất cả các thao tác cơ bản cho DSLK tọa độ các điểm trong mặt phẳng Oxy.
c. Liệt kê các tọa độ các điểm trong phần tử thứ I của mặt phẳng Oxy.
d. Tìm điểm có tung độ lớn nhất trong DSLK.
e. Viết hàm sắp xếp tọa độ các điểm giảm dần theo khoảng cách từ nó đến góc tọa độ.

37. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu của một DSLK đơn các hộp sữa (HOPSUA). Biết rằng một hộp sữa gồm những thành phần sau: Nhãn hiệu (chuỗi tối đa 20 ký tự), Trọng lượng (kiểu số thực), Hạn sử dụng (kiểu dữ liệu ngày).
b. Cài đặt tất cả các thao tác cơ bản cho DSLK các hộp sữa.
c. Đếm số lượng hộp sữa sản xuất trước năm 2003 trong DSLK.
d. Tìm hộp sữa mới nhất trong DSLK.
e. Sắp xếp các hộp sữa tăng dần theo hạn sử dụng.

38. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu cho DSLK đơn các phòng trong khách sạn (Phong). Giả sử phòng khách sạn gồm những thành phần thông tin như sau:
   - Mã phòng: tối đa 5 ký tự.
   - Tên phòng: tối đa 30 ký tự.
   - Đơn giá thuê: kiểu số thực.
   - Số lượng giường: kiểu số nguyên.
   - Tình trạng phòng: 0 rảnh, 1 bận.
b. Cài đặt tất cả các thao tác cơ bản cho DSLK các phòng.
c. Liệt kê các phòng trống trong DSLK.
d. Tính tổng số lượng giường có trong DSLK.
e. Sắp xếp danh sách liên kết tăng dần theo đơn giá thuê.

39. Viết chương trình thực hiện các yêu cầu sau:

a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các quyển sách. Biết rằng thông tin của một quyển sách bao gồm: tên sách (50 ký tự), tên tác giả (30 ký tự) và năm xuất bản.
b. Cài đặt tất cả các thao tác cơ bản cho DSLK các quyển sách.
c. Tìm quyển sách cũ nhất trong DSLK t.
d. Tìm một năm có nhiều sách xuất bản nhất và liệt kê tất cả các quyển sách xuất bản trong năm đó.

40. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các tỉnh. Biết rằng thông tin của một tỉnh bao gồm: tên tỉnh (tối đa 30 ký tự), diện tích (kiểu số thực), dân số (số nguyên dài).
b. Cài đặt tất cả các thao tác cơ bản cho danh sách liên kết các tỉnh.
c. Tính tổng diện tích của tất cả các tỉnh trong dslk.
d. Tìm địa chỉ của một node chứa tỉnh có diện tích lớn nhất trong danh sách.
e. Tìm một tỉnh có dân số lớn nhất trong dslk đơn.
f. Sắp xếp danh sách tăng dần theo diện tích.

41. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các vé xem phim (Ve). Biết rằng một vé xem phim gồm những thành phần như sau:
   - Tên phim: chuỗi tối đa 20 ký tự.
   - Giá tiền: kiểu số nguyên 4 byte.
   - Xuất chiếu: kiểu thời gian (ThoiGian).
   - Ngày xem: kiểu dữ liệu ngày (Ngay).
b. Nhập danh sách.
c. Xuất danh sách.
d. Tính tổng giá tiền của tất cả các vé trong DSLK.
e. Sắp xếp các phần tử trong mảng tăng dần theo ngày xem và xuất chiếu.

42. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các mặt hàng (MatHang). Biết rằng một mặt hàng gồm những thành phần sau:
   - Tên mặt hàng: chuỗi tối đa 20 ký tự.
   - Đơn giá: kiểu số nguyên 4 byte.
   - Số lượng tồn: kiểu số nguyên 4 byte.
b. Nhập danh sách.
c. Xuất danh sách.
d. Tìm mặt hàng có tổng giá trị tồn là lớn nhất.
e. Đếm số lượng mặt hàng có số lượng tồn lớn hơn 1.000.

43. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các chuyến bay. Biết rằng một chuyến bay gồm những thành phần như sau:
   - Mã chuyến bay: chuỗi tối đa 5 ký tự.
   - Ngày bay: kiểu dữ liệu ngày.
   - Giờ bay: kiểu thời gian.
   - Nơi đi: chuỗi tối đa 20 ký tự.
   - Nơi đến: chuỗi tối đa 20 ký tự.
b. Nhập danh sách.
c. Xuất danh sách.

d. Tìm một ngày có nhiều chuyến bay nhất.
e. Tìm một chuyến bay trong danh sách theo mã chuyến bay.

44. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các cầu thủ. Biết rằng một cầu thủ gồm những thành phần như sau:
   - Mã cầu thủ: chuỗi tối đa 10 ký tự.
   - Tên cầu thủ: chuỗi tối đa 30 ký tự.
   - Ngày sinh: kiểu dữ liệu ngày.
b. Nhập danh sách.
c. Xuất danh sách.
d. Liệt kê danh sách các cầu thủ nhỏ tuổi nhất trong danh sách.
e. Sắp xếp các cầu thủ giảm dần theo ngày sinh.

45. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các nhân viên (NhanVien). Biết rằng một nhân viên gồm những thành phần như sau:
   - Mã nhân viên: chuỗi tối đa 5 ký tự.
   - Tên nhân viên: chuỗi tối đa 30 ký tự.
   - Lương nhân viên: kiểu số thực.
b. Nhập danh sách.
c. Xuất danh sách.
d. Tìm một nhân viên có lương cao nhất trong danh sách.
e. Tính tổng lương của các nhân viên.
f. Sắp xếp danh sách tăng dần theo lương nhân viên.

46. Viết chương trình thực hiện các yêu cầu sau:
a. Hãy khai báo cấu trúc dữ liệu cho DSLK đơn các thí sinh. Biết rằng một thí sinh gồm những thành phần như sau:
- Mã thí sinh: chuỗi tối đa 5 ký tự.
- Họ tên thí sinh: chuỗi tối đa 30 ký tự.
- Điểm toán: kiểu số thực.
- Điểm lý: kiểu số thực.
- Điểm hóa: kiểu số thực.
- Điểm tổng cộng: kiểu số thực.

b. Nhập danh sách.
c. Xuất danh sách.
d. Liệt kê các thí sinh đậu trong danh sách. Một thí sinh được gọi là đậu khi có tổng điểm 3 môn lớn hơn 15 và không có môn nào bị điểm không.
e. Sắp xếp danh sách theo thứ tự giảm dần theo điểm tổng cộng.

47. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu cho danh sách liên kết đơn các luận văn (LuanVan). Biết rằng một luận văn gồm những thành phần như sau:
   - Mã luận văn: chuỗi tối đa 10 ký tự.
   - Tên luận văn: chuỗi tối đa 100 ký tự.
   - Họ tên sinh viên thực hiện: chuỗi tối đa 30 ký tự.
   - Họ tên giáo viên hướng dẫn: chuỗi tối đa 30 ký tự.
   - Năm thực hiện: kiểu số nguyên 2 byte.
b. Nhập danh sách.
c. Xuất danh sách.

d. Tìm năm có nhiều luận văn nhất.
e. Liệt kê các luận văn thực hiện gần nhất.

48. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu cho danh sách liên kết đơn các lớp học (LopHoc). Biết rằng một lớp học gồm những thành phần như sau:
   - Tên lớp: chuỗi tối đa 30 ký tự.
   - Sĩ số: kiểu số nguyên 2 byte.
   - Danh sách các học sinh trong lớp (tối đa 50 học sinh).
b. Nhập danh sách.
c. Xuất danh sách.
d. Tìm một lớp có sĩ số đông nhất.
e. Tìm một học sinh có điểm trung bình lớn nhất.

49. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu cho danh sách liên kết đơn các sổ tiết kiệm (SoTietKiem). Biết rằng một sổ tiết kiệm gồm những thành phần như sau:
   - Mã số: chuỗi tối đa 5 ký tự.
   - Loại tiết kiệm: chuỗi tối đa 10 ký tự.
   - Họ tên khách hàng: chuỗi tối đa 30 ký tự.
   - Chứng minh nhân dân: kiểu số nguyên 4 byte.
   - Ngày mở số: kiểu dữ liệu ngày.
   - Số tiền gởi: kiểu số thực.
b. Nhập danh sách.
c. Xuất danh sách.
d. Tính tổng số các tiền gởi có trong danh sách.
e. Tìm đia chỉ một nút theo Chứng Minh Nhân Dân.

50. Viết chương trình thực hiện các yêu cầu sau:
a. Khai báo cấu trúc dữ liệu của một danh sách liên kết đơn các đại lý (DaiLy). Biết rằng một đại lý gồm những thành phần như sau:
   - Mã đại lý: chuỗi tối đa 5 ký tự.
   - Tên đại lý: chuỗi tối đa 30 ký tự.
   - Điện thoại: kiểu số nguyên 4 byte.
   - Ngày tiếp nhận: kiểu dữ liệu ngày.
   - Địa chỉ: chuỗi tối đa 50 ký tự.
   - E-Mail: chuỗi tối đa 50 ký tự.
b. Nhập danh sách.
c. Xuất danh sách.
d. Tìm một đại lý theo tên đại lý.
e. Tìm một đại lý được tiếp nhận gần đây nhất.

51. Viết chương trình thực hiện những yêu cầu sau:
- Khai báo cấu trúc dữ liệu của một danh sách liên kết đơn để lưu tọa độ các đỉnh của một đa giác lồi trong mặt phẳng Oxy.
- Tính chu vi của đa giác.
- Diện tích của đa giác.

9.3.6. *Các dạng DSLK khác*

52. Có nhu cầu xây dựng DSLK đôi cho các trường hợp sau:

- Phân số
- Hỗn số
- Sổ tiết kiệm (bài tập 49)
- Đại lý (bài tập 50)

Đối với mỗi dạng cấu trúc dữ liệu ở trên, lần lượt thực hiện:

a. Khai báo cấu trúc dữ liệu cần dùng

b. Viết hàm khởi tạo danh sách liên kết

c. Viết hàm tạo node trong danh sách liên kết kép với thông tin do người dùng nhập vào.

d. Viết hàm thêm node trong các trường hợp:

- vào đầu danh sách
- vào cuối danh sách
- vào sau node q đang có trong danh sách.

e. Viết hàm sắp xếp DSLK tăng dần theo giá trị khóa có trong Data.

f. Viết hàm hủy node trong các trường hợp:

- Node ở đầu danh sách
- Node ở cuối danh sách
- Node nằm ngay sau node q đang có trong danh sách.

53. Thực hiện lại bài tập 52 với DSLK là DSLK đơn vòng

54. Thực hiện lại bài tập 52 với DSLK là DSLK đôi vòng

CÂY NHỊ PHÂN TÌM KIẾM

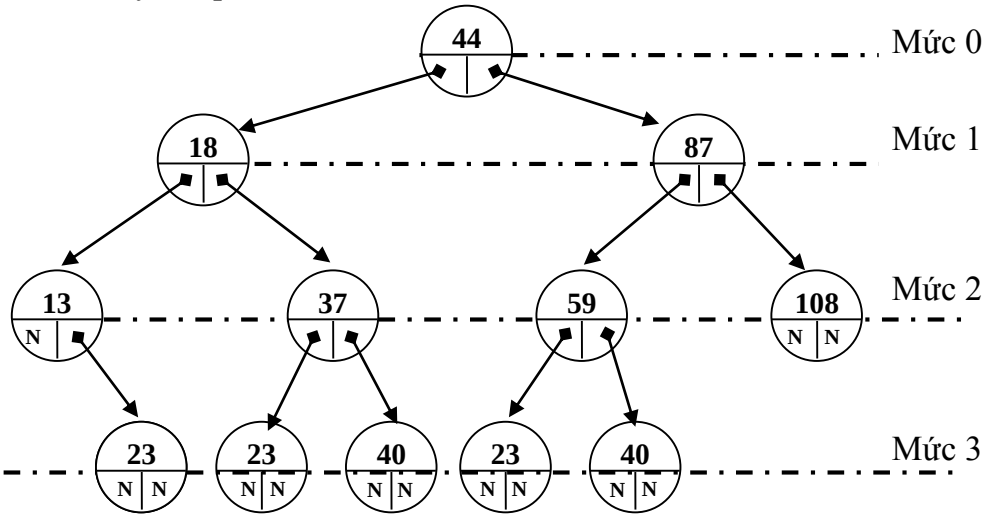
10.1. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ CẤU TRÚC CÂY (*TREE*)

- *Nút (Node)*: là một phần tử trong cây. Mỗi nút có thể chứa một dữ liệu bất kỳ.
- *Bậc của một nút:* là số cây con của nút đó.
- *Bậc của một cây:* là bậc lớn nhất của các nút. Cây có bậc n thì gọi là cây n-phân.
- *Nút gốc (Root node): là nút không có nút cha*
- *Nút lá (Leaf node hay nút ngoài – External node): là nút có bậc = 0 (nút không có nút con)*
- *Nút nội (Internal node hay Nút nhánh): là nút có nút cha và có nút con*
- *Nút cha (Parent node)*
- *Nút con (Child node)*
- *Nút anh em (Sibling node):* nút có cùng nút cha
- *Nhánh (Branch)*: là nhánh nối giữa hai nút
- *Mức của một gốc:*
  - Mức (gốc(T))=0;
  - Nếu T_1 là con của T_0 thì: Mức(T_1)=Mức(T_0)+1
- *Độ dài đường đi từ gốc đến nút x:* là số nhánh cần đi qua kể từ gốc đến x.
- *Độ cao của cây:* Độ dài đường đi từ gốc đến nút lá ở mức thấp nhất.

10.2. CÂY NHỊ PHÂN

10.2.1. ***Định nghĩa:*** *cây nhị phân là cây mà mỗi gốc có tối đa 2 cây con*

Hình ảnh cây nhị phân



10.2.2. *Biểu diễn cây nhị phân*

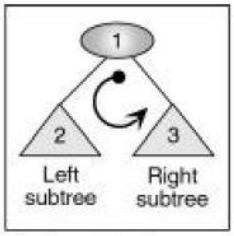
10.2.2.1. Khai báo cấu trúc của cây

```
typedef struct tagNODE
{
    int key;
    tagNODE* pLeft;
    tagNODE* pRight;
}NODE;
typedef NODE* TREE;
```

10.2.2.2. Thao tác cơ bản - Duyệt Cây

*10.2.2.2.1. Duyệt gốc trước (**pre-order traversal** hay Node-Left-Right)*

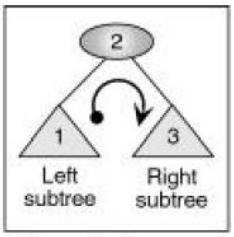
```
void  NLR(TREE Root)
{
    if(Root)
    {
        <Xử lý Root>;//xử lý theo yêu cầu
        NLR(Root->pLeft);
        NLR(Root->pRight);
    }
}
```



pre-order traversal

*10.2.2.2.2. Duyệt gốc giữa (**in-order traversal** hay Left-Node-Right)*

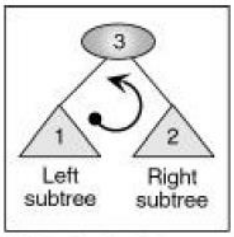
```
void  LNR(TREE Root)
{
    if(Root)
    {
        LNR(Root->pLeft);
        <Xử lý Root>;//xử lý theo yêu cầu
            LNR(Root->pRight);
    }
}
```



in-order traversal

*10.2.2.2.3. Duyệt gốc sau (**post-order traversal** hay Left-Right-Node)*

```
void  LRN(TREE Root)
{   if(Root)
    {   LRN(Root->pLeft);
        LRN(Root->pRight);
        <Xử lý Root>;//xử lý theo yêu cầu
    }
}
```



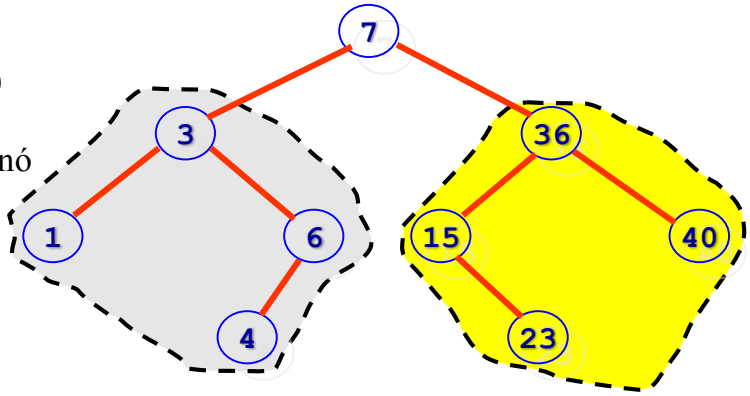
post-order traversal

10.3. CÂY NHỊ PHÂN TÌM KIẾM (BST- *Binary Search Tree*)

10.3.1. *Định nghĩa*

Cây nhị phân tìm kiếm (CNPTK)

- Là cây nhị phân.
- Tại mỗi node đang xét, giá trị của nó
  - Lớn hơn tất cả bên trái.
  - Nhỏ hơn tất cả bên phải.
- Giá trị các node không được trùng nhau.



10.3.2. *Thao tác cơ bản*

10.3.2.1. Thêm node vào cây

10.3.2.1.1. Tạo một phần tử có Key=X

```
NODE* GetNode(int X)
{
     NODE* p=new NODE;
     if(p!=NULL)
     {
          p->key = X;
          p->pLeft = p->pRight = NULL;
     }
     return p;
}
```

10.3.2.1.2. Thêm một phần tử X vào cây

```
bool AddNode (TREE &t, int X)
{
    if(t!=NULL)
    { if(X==t->Key)  // Đã có giá trị X trên cây
          return false;
      if(X<t->Key)
          return AddNode (t->pLeft, X);//Tìm vị trí thích hợp bên trái
      return AddNode (t->pRight, X);// Tìm vị trí thích hợp bên phải
    }
    else// Tìm thấy vị trí thích hợp. Thực hiện thêm node vào cây
    {
          NODE* p;
          p=GetNode(X);
          if(p!=NULL)
               t=p;
          return true;
    }
}
```

10.3.2.1.3. Tạo cây với giá trị các node được phát sinh ngẫu nhiên

```
void CreateTree_Random(TREE &T,int sl)
{
     int X,i=0;
     while (i<sl)
     {
       do
       {
            X= rand()%100;
       }while (AddNode(T,X)==false);
       i++;
     }
}
```

10.3.2.1.4. Tạo cây với giá trị lấy từ giá trị có trong mảng đã có

```
void CreateTreeFromArray(TREE &T,int A[], int sl)
{ /* Cho mảng A có kích thước tối đa là SIZE. sl: l số
     lượng node cần tạo trên cây. VD này xét trong trường hợp giá
     trị các phần tử trong mảng A có thể trùng nhau */
     int dem=0,i=0;
     while ((dem<sl) && (i<size))
     {
          while ((AddNode(T,A[i])==false) && (i<size))
```

```
                i++;
            dem++;
        }
    }
```

10.3.2.2. Duyệt cây

- Có 3 công việc luôn phải có trong hàm duyệt cây là
  - *(i)* In giá trị của node đang xét (gọi tắt là N)
    ```
    printf("%5d",T->key);
    ```
  - *(ii)* Duyệt cây trên nhánh trái của node đang xét. (gọi tắt là L).
    ```
    if (T->pLeft!=NULL)
        PrintTreeLNR(T->pLeft);
    ```
  - *(iii)* Duyệt cây trên nhánh phải của node đang xét(gọi tắt là R).
    ```
    if (T->pRight!=NULL)
        PrintTreeLNR(T->pRight);
    ```
- Tuỳ vào việc đảo thứ tự của N, L, R ta sẽ có 6 cách duyệt cây lần lượt là: NLR, NRL, LNR, LRN, RLN, RNL. Khi đó ta chỉ việc sắp xếp thứ tự các lệnh có trong hàm sao cho đáp ứng yêu cầu. Ví dụ, với cách duyệt theo thứ tự LNR, ta có hàm sau:

```
void PrintTreeLNR(TREE T)
{
    if (T)
    {
        if (T->pLeft!=NULL)
            PrintTreeLNR(T->pLeft);
        printf("%5d",T->key);
        if (T->pRight!=NULL)
            PrintTreeLNR(T->pRight);
    }
}
```

10.3.2.3. Tìm một phần tử x trong cây

10.3.2.3.1. Đệ quy

```
NODE* SearchNode(TREE T, int X)
{   if(T)
    {   if(T->key==X)
            return T;
        if(T->key>X)
            return SearchNode(T->pLeft,X);
        else
            return SearchNode(T->pRight,X);
    }
    return NULL;
}
```

10.3.2.3.2. Khử đệ quy

```
NODE* SearchNode(TREE T, int X)
{   NODE* p=T;
    while(p)
    {
        if(p->key==X)  return p;
        if(p->key>X) p=p->pLeft;
        else     p=p->pRight;
    }
    return NULL;
}
```

10.3.2.4. Xoá phần tử khỏi cây

10.3.2.4.1. Xoá phần tử trên cây có khoá là X

Gọi p là nút chứa khoá x. Có 3 trường hợp:

(i) TH 1: p là nút lá ⇒ huỷ p

(ii) TH 2: p chỉ có 1 con bên trái hoặc phải:

- Tạo liên kết từ cha của p đến con của p.
- Huỷ p.

*(iii)*TH 3: p có đủ 2 con:

- Tìm phần tử thế mạng cho p. Có thể chọn 1 trong 2 nguyên tắc:
  - Phần tử phải nhất của cây con trái (phần tử lớn nhất trong các phần tử nhỏ hơn p).
  - Phần tử trái nhất của cây con phải (phần tử nhỏ nhất trong các phần tử lớn hơn p).
- Chép thông tin của phần tử thế mạng vào p
- Huỷ phần tử thế mạng

```
// hàm tìm phần tử thế mạng trái nhất cây con bên phải
void SearchStandFor(TREE &T, TREE &p)
{
     if(T->pLeft)
          SearchStandFor(T->pLeft,p);
     else
     {
          p->Key=T->Key;
          p=T;
          T=T->pRight;
     }
}
//hàm xóa phần tử x
bool  DelNode(TREE &T, int x)
{
     if(T==NULL)
          return false;
     if(T->Key>x)
          return DelNode(T->pLeft,x);
     else
          if(T->Key<x)
               return DelNode(T->pRight,x);
          else//Tìm thấy T->Key = X
          {
               NODE*p = T;
               if(T->pLeft==NULL)  //T không có cây con bên trái
                    T=T->pRight;
               else
                 if(T->pRight==NULL) //T không có cây con bên phải
                      T=T->pLeft;
                 else//có cả hai nhánh con ⇒ tìm phần tử thế mạng q:
                 { // là trái nhất của cây con bên phải
                      SearchStandFor(T->pRight,p);
                 }
               delete p;
               return true;
          }
}
```

10.3.2.4.2. Xoá tất cả các phần tử trên cây (xoá toàn bộ cây)

```
void  RemoveTree(TREE &T)
{
      if(T)
      {
           RemoveTree(T->pLeft);
           RemoveTree(T->pRight);
           delete T;
      }
}
```

10.3.2.5. Kỹ thuật đặ lính canh

Viết hàm tìm giá trị lớn nhất trong cây nhị phân các số.

```
int  TimMax (TREE  t)
{
     if  ( (t->pLeft == NULL)  &&  (t->pRight==NULL) )
          return  t->info;
     int  left = lonnhat (t->pLeft);
     int  right = lonnhat (t->pLeft);
     int  ln = t->info;
     if (left > ln)
          ln = left;
     if  (right > ln)
          ln = right;
     return ln;
}
```

10.3.2.6. Kỹ thuật đếm

Viết hàm đếm các giá trị âm có trong cây?

```
int  demam (TREE  t)
{
     if  (t = = NULL)
          return  0;
     int  left = demam (t->Left);
     int  right = demam (t->right);
     return  (1 + left + right);
}
```

10.3.2.7. Kỹ thuật tính toán

Cho cây nhị phân T các số thực. Viết hàm tính tổng các giá trị dương nhỏ hơn 200 có trong cây

```
float  TongDuongNhoHon200 (TREE  t)
{
     if  (t ==NULL)
          return  0;
     float  left = TongDuongNhoHon200 (t->pLeft);
     float right = TongDuongNhoHon200 (t->pRight);
     if  ( (t->info > 0) && (t->info<200) )
          return  (t →info + left + right);
     return  (left + right);
}
```

10.3.2.8. Kỹ thuật đặt cờ hiệu

Cho cây nhị phân T các số thực. Viết hàm kiểm tra trong cây có tồn tại giá trị 0 hay không?

```
int  KiemTraTonTai (TREE  t)
{
     if  (t == NULL)
```

```
        return  0;
    if  (KiemTraTonTai (t->pLeft) == 1)
        return  1;
    if  (KiemTraTonTai (t->pRight) == 1)
        return  1;
    if  (t->info == 0)
        return  1;
    return  0;
}
```

10.4. KÝ PHÁP BA LAN (*Polish notation*)

10.4.1. *Giới thiệu*

- Do nhà logic toán học người Ba Lan Jan Łukasiewicz đề xuất khoảng năm 1920.
- *Ký pháp Ba lan* còn gọi là ký pháp tiền tố (*prefix notation*), là một cách viết một biểu thức đại số rất thuận lợi cho việc thực hiện các phép toán.
- Đặc điểm cơ bản của cách viết này là:
  - Không cần dùng đến các dấu ngoặc;
  - Luôn thực hiện từ trái sang phải.

10.4.2. *Các phương pháp biểu diễn phép toán hai ngôi*

Một phép toán hai ngôi trên tập hợp X có thể được diễn tả dưới các dạng sau, ví dụ để biểu diễn biểu thức `A-[B/(C+D)]`:

Dạng trung tố (Infix)	Dạng hậu tố (Postfix)	Dạng tiền tố (Prefix)
A-B/(C+D)	ABCD+/-	-A/B+CD

Trong đó:

- A, B, C, D được gọi là toán hạng.
- Các phép tính trừ (-), chia(/), cộng(+) được gọi là toán tử.
- Dạng ký pháp trung tố (*Infix notation*): toán tử được đặt ở *giữa* hai toán hạng tham gia vào phép toán.

  VD1: Xem xét một biểu thức đại số dạng Infix sau: `1 + 2 * 3 = ?`

1 + 2 * 3	= ?
1 + 6	= 7

Phép nhân (*) có độ ưu tiên cao hơn phép cộng (+)

  VD2: `(1+2) * 3 = ?`

(1 + 2) * (3 – 4)	= ?
3 * -1	= -3

Các phép toán trong ngoặc đơn có độ ưu tiên cao nhất

  Rõ ràng, với dạng ký pháp này, để người dùng không bị nhầm lẫn, thường phải bổ sung thêm các dấu ngoặc đơn để xác định độ ưu tiên của các phép tính.
- Dạng ký pháp tiền tố (*Prefix notation*): toán tử được đặt ở *trước* hai toán hạng tham gia vào phép toán.
  - VD3: Xem xét một biểu thức đại số dạng Prefix sau:

+ 1 * 2 3	= ?
+ 1 6	= 7

  - VD4:

* + 1 2 - 3 4	= ?
* 3 -1	= -3

- Dạng ký pháp hậu tố (*Postfix notation*): toán tử được đặt ở *sau* hai toán hạng tham gia vào phép toán.

VD5:

1 2 3 * +	= ?
1 6 +	= 7

VD6:

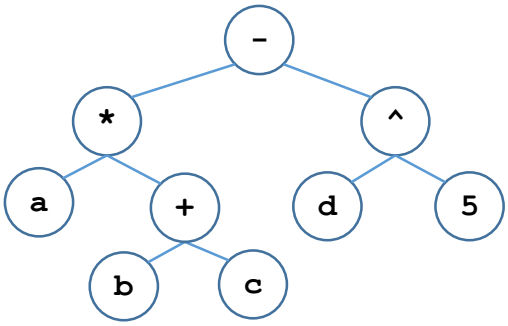
1 2 + 3 4 - *	= ?
3 -1 *	= -3

10.4.3. *Cây biểu diễn biểu thức*

10.4.3.1. Biểu diễn biểu thức bằng cây nhị phân

Ví dụ: với biểu thức

```
a * (b + c) - d ^ 5
```

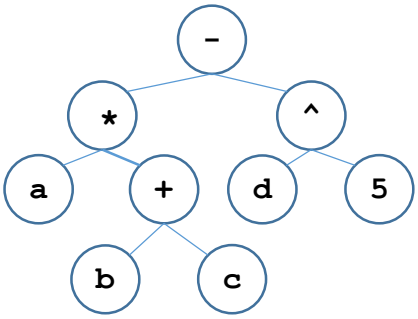


Được thực hiện theo sơ đồ biểu diễn bởi cây nhị phân sau, trong đó:

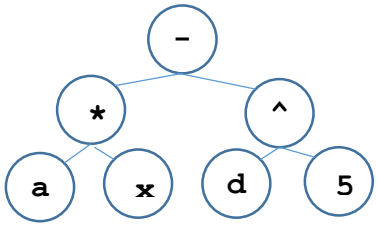
- Các biến và hằng luôn được biểu diễn bằng các lá.
- Các toán tử luôn biểu diễn bởi các nút trong.

10.4.3.2. Mô tả quá trình đọc, ghi nhận giá trị và thực hiện phép tính

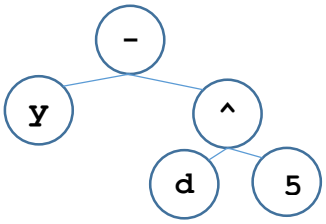
- **Theo dạng ký pháp *Infix* (LNR):**
  - Theo cách này, mỗi khi duyệt một đỉnh:
    - Nếu là lá bên trái (biến hoặc hằng) thì ghi nhận giá trị của biến
    - Nếu là toán tử thì lưu dạng của toán tử
    - Nếu là con phải và lá thực hiện phép tính theo toán tử đã lưu ở đỉnh cha giữa các giá trị đã lưu tại con phải và giá trị mới đọc tại con trái, ghi kết quả vào đỉnh cha.



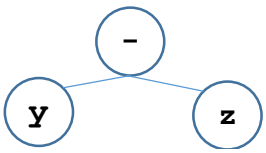
- *Thứ tự đọc: a * b + c*
- *Sau khi đọc đỉnh bên phải (c) của toán tử cộng, tính b+c (=x), lưu kết quả vào đỉnh của phép cộng (+)*



- *Thứ tự đọc trở thành: a * x*
- *Tính a*x (=y), lưu kết quả vào đỉnh của phép nhân (*)*



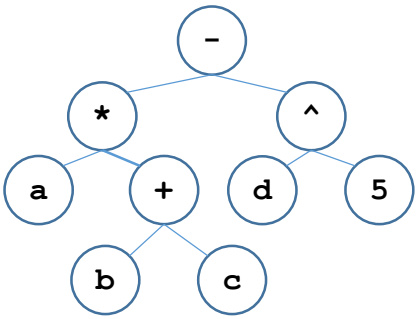
- *Thứ tự đọc trở thành: y*
- *Quá trình duyệt cây tiếp tục, để có y – d ^ 5*
- *Do 5 là đỉnh bên phải của đỉnh chứa phép lũy thừa ⇨ thực hiện phép tính d ^ 5 (=z)*



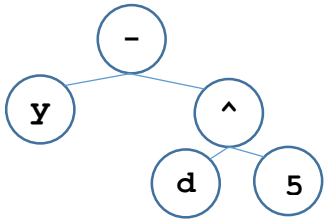
- *Thứ tự đọc trở thành: y - z*
- *Thực hiện phép tính y – z để cho kết quả*

- Thứ tự duyệt các đỉnh theo dạng Infix (LNR): a * b + c - d ^ 5

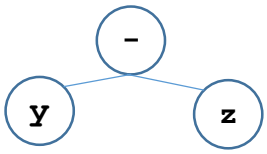
- **Theo dạng ký pháp *Postfix* (LRN)**
  - Theo cách này, mỗi khi duyệt một đỉnh:
    - Nếu là lá thì ghi nhận giá trị của biến
    - Nếu là đỉnh trong thì thực hiện toán tử ghi ở đỉnh này vào các giá trị ghi ở hai đỉnh con, ghi kết quả vào chính đỉnh này.



- *Thứ tự đọc: a b c +*
- *Sau khi đọc đỉnh trong (toán tử cộng), tính b+c (=x), lưu kết quả vào đỉnh của phép cộng (+)*
- *Thứ tự đọc trở thành: a x*



- *Đọc tiếp đến toán tử nhân, thứ tự đọc trở thành: a x **
- *Tính a*x (=y), lưu kết quả vào đỉnh của phép nhân (*)*



- *Thứ tự đọc trở thành: y*
- *Quá trình duyệt cây tiếp tục, để có y – d 5 ^*
- *Thực hiện phép tính d ^ 5 (=z)*
- *Thứ tự đọc trở thành: y z*
- *Đọc tiếp đến toán tử trừ, thứ tự đọc trở thành: y z -*
- *Thực hiện phép tính y-z để có kết quả sau cùng.*

  - Thứ tự duyệt các đỉnh khi đó sẽ là: `a b c + * d 5 ^ -`

10.4.3.3. Nhận xét

- Dạng *Infix*: nếu không dùng đến dấu ngoặc, có thể có hai cây biểu thức khác nhau cho cùng một dãy đỉnh khi duyệt thứ tự giữa.
- Dạng *Postfix*: luôn dựng lại được cây biểu thức duy nhất theo giải thuật sau: *Khi độ dài dãy lớn hơn 1, duyệt từ bên trái sang, nếu gặp một phần tử là toán tử, thì lấy hai phần tử đứng trước nó ra khỏi dãy chuyển thành hai con của phần tử ấy (theo đúng thứ tự)*. Vì thế biểu thức viết theo dạng Postfix là hoàn toàn xác định.

10.4.4. *Ký pháp Ba Lan nghịch đảo (Reverse Polish Notation -RPN)*

10.4.4.1. Giới thiệu

Ký pháp nghịch đảo Balan (RPN), còn được gọi là Postfix, do Charles Hamblin đề xuất vào những năm 1950s

Ký pháp này lấy ý tưởng của Polish notation, được đề xuất vào năm 1920 của nhà toán học người Balan có tên Jan Łukasiewicz (trong một số tài liệu gọi là ký pháp Łukasiewicz), giúp chuyển đổi từ biểu thức bình thường sang ký pháp Ba Lan

Việc tính giá trị một biểu thức viết dưới dạng Postfix rất thuận tiện như trên, tuy nhiên, theo thói quen thông thường, việc nhập biểu thức đó vào lại không dễ, người ta thường nhập vào một công thức dưới dạng thông thường (dạng Infix) rồi dùng chương trình chuyển đổi nó sang dạng Postfix.

Tại sao nên sử dụng RPN?

- RPN cho phép giảm thời gian trong việc tính một biểu thức. Người dùng không cần quan tâm đến dấu ngoặc trong biểu thức.
- RPN cho phép thấy kết quả ngay sau phép toán, do đó, người sử dụng có thể kiểm tra kết quả dễ hơn, nhanh hơn.
- Với RPN, việc thực hiện trên máy tính tỏ ra hiệu quả hơn

- Với cách viết này, việc tính toán dựa trên thứ tự của biểu thức, kết hợp với thứ tự ưu tiên của phép toán.
- RPN có tính logic cao vì người dùng đưa biểu thức, sau đó đưa phép tính cần thực hiện.

10.4.4.2. Ví dụ

Với biểu thức `Q = a*(b+c)-d^5`

Ký hiệu biểu thức ghi dưới dạng Postfix là *P*. Trong quá trình chuyển đổi ta dùng một *stack S* để lưu các phần tử trong *P* chưa sử dụng đến.

Trong stack này, nếu toán tử (m) định đưa vào có độ ưu tiên thấp hơn toán tử sẵn có trên đỉnh stack (n) thì lấy toán tử n ra đưa vào stack, rồi đưa toán tử m vào stack.

Khi đọc từ trái sang phải biểu thức Q, lần lượt có kết quả như sau:

(i). Đọc và ghi nhận giá trị a, ghi giá trị a vào P. ⇒P= "*a*".

(ii). Đọc toán tử "*". Đưa toán tử này vào stack S. ⇒S= "*"

(iii). Đọc dấu ngoặc mở "(", đưa dấu ngoặc này vào stack ⇒S= "**(**".

(iv). Đọc toán hạng b, đưa b vào P. ⇒*P*= "*a b*"

(v). Đọc toán tử "+", đặt "+" vào stack. ⇒*S*= "*(+"

(vi). Đọc toán hạng *c*, đưa *c* vào cuối *P*. ⇒*P*= "*a b c*"

(vii). Đọc dấu ngoặc đóng ")". Lần lượt lấy các toán tử ở cuối stack ra khỏi stack đặt vào cuối P cho đến khi gặp dấu ngoặc mở "(" trong stack thì giải phóng nó.
⇒*S*= "*"; *P*="*a b c +*"

(viii). Đọc toán tử "-". Cuối stack *S* có toán tử "*" có mức ưu tiên lớn hơn toán tử "-", ta lấy toán tử "*" ra khỏi stack, đặt vào cuối P, đặt toán tử "-" vào stack.
⇒*S*="-"; *P*=" *a b c + * "

(ix). Đọc toán hạng *d*, đưa *d* vào cuối *P*. ⇒*P*= "*a b c + * d*"

(x). Đọc toán tử "^", đưa toán tử "^" vào cuối stack. ⇒ *S*="-^"

(xi). Đọc hằng số *5*, đưa *5* vào cuối *P*. ⇒ *P*="*a b c + * d 5*"

(xii). Đã đọc hết biểu thức *Q*, lần lượt lấy các phần tử cuối trong stack đặt vào *P* cho đến hết. ⇒*P*="*a b c + * d 5 ^ -*"; S=“”

📖 Một cách trình bày khác khi chuyển từ *Infix* sang *Postfix*:

Thứ tự	Đọc	S	P
1	**a**		a
2	*****	*	a
3	**(**	*(	a
4	**b**	*(	a b
5	**+**	*(+	a b
6	**c**	*(+	a b c

Thứ tự	Đọc	S	P
7	**)**	*	a b c +
			a b c +
8	**-**		a b c + *
		-	a b c + *
9	**d**		a b c + * d
10	^	- ^	a b c + * d
11	**5**		a b c + * d 5
		-	a b c + * d 5 ^
			a b c + * d 5 ^ -

10.4.4.3. Tính giá trị biểu thức dạng Postfix

- **Bước 1:** Đọc lần lượt các phần tử của biểu thức P (từ trái qua phải).
  - Nếu gặp toán hạng thì đẩy nó vào Stack.

- Nếu gặp phép toán thì lấy hai phần tử liên tiếp trong Stack thực hiện phép toán, kết quả được đẩy vào trong Stack.

- **Bước 2:** Nếu chưa đọc hết biểu thức trong P thì lặp lại bước 1, ngược lại chuyển sang bước 3.
- **Bước 3:** Kết thúc. Lúc này đỉnh của Stack chứa giá trị của biểu thức cần tính

Ví dụ: Cho biểu thức (các hằng số được xem như là số nguyên)

```
P = 4 5 + 2 3 + * 6 + 8 7 + /
```

Thứ tự	*Đọc*	*Stack*	*Giải thích*
1	**4**	4	
2	**5**	4 5	
3	+	9	9=4+5
4	**2**	9 2	
5	**3**	9 2 3	
6	+	9 5	5=2+3

Thứ tự	*Đọc*	*Stack*	*Giải thích*
7	*	45	45=9*5
8	**6**	45 6	
9	+	51	51=45+6
10	**8**	51 8	
11	**7**	51 8 7	
12	+	51 15	15=8+7
13	/	3	3=51/15

10.5. BÀI TẬP

10.5.1. *Bài tập cơ bản*

10.5.1.1. Cho cây nhị phân các số nguyên với giả sử các giá trị trong cây không trùng nhau. Hãy viết các hàm thực hiện các yêu cầu sau:

10.5.1.1.1. Kỹ thuật nhập, xuất, liệt kê

1. Viết hàm xuất các giá trị trong cây.
2. Viết hàm xuất các giá trị chẵn trong cây
3. Viết xuất địa chỉ các nút trên cây có giá trị (khóa) lớn hơn x và nhỏ hơn y.
4. Viết hàm xuất các số hoàn thiện trong cây.
5. **(*)** Viết hàm xuất tất cả các nút trên tầng thứ k của cây.
6. **(*)** Viết hàm xuất tất cà các nút trên cây theo thứ tự tăng dần từ 0 đến tầng h -1 của cây (với h là chiều cao của cây).
7. Viết hàm cho người dùng chọn lựa cách nhập các phần tử vào cây NPTK theo một trong các cách sau:
   i. Nhập trực tiếp từ bàn phím
   ii. Phát sinh ngẫu nhiên.
   iii. Nhập từ mảng 1 chiều
   iv. Nhập từ 1 danh sách liên kết (DSLK)
8. Viết hàm thực hiện liệt kê:
   a. Các nút lá
   b. Các nút có 1 cây con
   c. Các nút có 2 cây con.
   d. Các nút lá có khoá < x (x là tham số đầu vào của hàm). Thực hiện tương tự cho hàm đếm số nút có khoá >x.

10.5.1.1.2. Kỹ thuật đếm

1. Tổng số nút có trên cây.
2. Số nút lá (node bậc 0)
3. Đếm số lượng nút có đúng một con (trái hoặc phải đều được).

4. Đếm số lượng nút chỉ có 1 cây con trái
5. Số node chỉ có 1 cây con phải
6. Đếm số lượng nút có đúng hai con.
7. Đếm số lượng nút có giá trị là số lẻ (tương tự cho trường hợp số chẵn).
8. Đếm số lượng nút có đúng một con mà thông tin tại nút đó là số nguyên tố.
9. Đếm số lượng nút có đúng hai con mà thông tin tại nút đó là số chính phương.
10. Đếm số lượng nút trên tầng thứ k của cây.
11. Đếm số lượng nút nằm ở tầng thấp hơn tầng thứ k của cây.
12. Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k của cây.
13. Số nút lá có khoá < x (x là tham số đầu vào của hàm). Thực hiện tương tự cho hàm đếm số nút có khoá >x
14. Số node trên từng mức của cây

10.5.1.1.3. Kỹ thuật tính toán

15. Tính tổng các nút có trong cây.
16. Tính tổng các nút lá trong cây.
17. Tính tổng các nút có một con.
18. Tính tổng các nút có đúng hai con.
19. Tính tổng các nút lẻ.
20. Tính tổng các nút lá mà thông tin tại đó là giá trị chẵn.
21. Tính tổng các nút có đúng một con mà thông tin tại nút đó là số nguyên tố.
22. Tính tổng các nút có đúng hai con mà thông tin tại nút đó là số chính phương.
23. Tính tổng các nút lá có khoá < x (x là tham số đầu vào của hàm). Thực hiện tương tự cho hàm đếm số nút có khoá >x
24. Tính chiều cao của cây
25. Chiều cao của cây.
26. Độ dài đường đi từ gốc đến node x
27. Độ lệch lớn nhất trên cây. Biết rằng:
- Độ lệch lớn nhất trên cây là độ lệch của nút có độ lệch lớn nhất.
- Độ lệch trên 1 nút là chênh lệcch giữa chiều cao của cây con trái và chiều cao của cây con phải.

10.5.1.1.4. Kỹ thuật đặt cờ hiệu

28. Kiểm tra cây nhị phân T có phải là "cây nhị phân tìm kiếm" hay không ?
29. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng" hay không ?
30. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng hoàn toàn" hay không?

10.5.1.1.5. Kỹ thuật lính canh

31. Tìm giá trị lớn nhất trong cây.
32. Tìm giá trị nhỏ nhất trong cây.
33. Tìm độ lệch lớn nhất trên cây.
34. Tìm địa chỉ nút mà giá trị tại nút đó là lớn nhất.
35. Tìm địa chỉ của nút trong cây mà giá trị của nút đó bằng giá trị x. Nếu không thấy hàm trả về giá trị NULL.

10.5.1.2. Cây nhị phân các phân số

Cho cây nhị phân các phân số với giả sử các giá trị trong cây không trùng nhau. Hãy viết các hàm thực hiện các yêu cầu sau.

10.5.1.2.1. Kỹ thuật nhập, xuất, liệt kê.

36. Viết hàm xuất các phân số dương trong cây.

37. Viết hàm xuất tất cả phân số dương trên tầng thứ k của cây.

38. Viết hàm xuất tất cả các nút trên cây theo thứ tự từ 0 đến tầng h -1 của cây (với h là chiều cao của cây).

10.5.1.2.2. Kỹ thuật đếm

39. Đếm số lượng nút có đúng một con và giá trị tại đó là phân số tối giản.

40. Đếm số lượng nút có đúng hai con và giá trị tại nút đó là phân số âm.

41. Đếm số lượng nút âm trên tầng thứ k của cây.

42. Đếm số lượng nút nằm ở tầng thấp hơn tầng thứ k của cây.

43. Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k của cây.

10.5.1.2.3. Kỹ thuật tính toán

44. Bài 910. Tính tổng các nút trong cây.

45. Bài 911. Tính tổng các nút lá trong cây.

46. Bài 912. Tính tổng các nút có đúng một con.

47. Bài 913. Tính tổng các nút có đúng hai con.

48. Bài 914. Tính chiều cao của cây.

10.5.1.2.4. Kỹ thuật đặt cờ hiệu

49. Kiểm tra cây nhị phân T có phải là "cây nhị phân tìm kiếm" hay không ?

50. Kiểm tra cây nhị phân T có phải là "cânhị phân cân bằng" hay không ?

51. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng hoàn toàn" hay không?

10.5.1.2.5. Kỹ thuật lính canh

52. Tìm giá trị lớn nhất trong cây.

53. Tìm giá trị nhỏ nhất trong cây.

54. Tìm độ lệch lớn nhất trên cây. Độ lệch của một nút trong cây được định nghĩa là chiều cao của cây con trái trừ chiều cao cây con phải lấy trị tuyệt đối.

55. Tìm địa chỉ nút mà giá trị tại nút đó là giá trị lớn nhất.

10.5.1.3. Các bài tập khác

56. Cho cây nhị phân T trong đó thông tin tại mỗi nút trong cây biểu diễn các thành phần thông tin của một tỉnh(TINH). Biết rằng một tỉnh gồm những thành phần như sau:

- Mã tỉnh : kiểu số nguyên 2 byte.
- Tên tỉnh : chuỗi tối đa 30 ký tự.
- Dân số : kiểu số nguyên 4 byte.
- Diện tích : kiểu số thực.

a. Tìm địa chỉ một node mà dân số tại node đó là lớn nhất trong cây.

b. Liệt kê các tỉnh trong cây có diện tích lớn hơn 100.000.

c. Đếm số lượng node lá có diện tích nhỏ hơn 700 và dân số lớn hơn 1.000.000.

57. Cho cây nhị phân T trong đó thông tin tại mỗi nút trong cây biểu diễn các thành phần thông tin của một đọc giả (DOCGIA). Biết rằng một đọc giả gồm những thành phần như sau:

- Mã đọc giả : chuỗi tối đa 5 ký tự.
- Tên đọc giả : chuỗi tối đa 30 ký tự.
- Ngày sinh : kiểu ngày.
- Địa chỉ : chuỗi tối đa 30 ký tự.
- Ngày lập thẻ : kiểu ngày

a. Tìm địa chỉ một node mà đọc giả tại node đó là lớn tuổi nhất trong cây.

b. Liệt kê các đọc giả trong cây sinh sau năm 1975.

c. Đếm số lượng node có đủ hai con có ngày thành lập thẻ trong ngày 07/05/2004.
d. Tìm kiếm địa chỉ một node trong cây theo mã đọc giả.

58. Cho cây nhị phân T trong đó thông tin tại mỗi nút trong cây biểu diễn các thành phần thông tin của một học sinh gồm những thành phần như sau:

Họ tên : chuỗi tối đa 30 ký tự.
Giới tính : kiểu số nguyên 2 byte.
Ngày sinh : kiểu dữ liệu ngày.
Địa chỉ : chuỗi tối đa 50 ký tự.
E mail : chuỗi tối đa 30 ký tự

a. Tìm địa chỉ một node mà học sinh tạo node đó là lớn tuổi nhất trong cây.
b. Liệt kê các học sinh trong cây

10.5.2. *Bài tập nâng cao*

59. Cho cây nhị phân tìm kiếm T các số thực, hãy cho biết:

a. Đặc điểm thứ tự các nút được xuất ra trong cây khi duyệt cây theo thứ tự LNR.
b. Đặc điểm thứ tự các nút được xuất ra trong cây khi duyệt cây theo thứ tự RNL.
c. Phải duyệt cây như thế nào để lưu cây trên tập tin sao cho khi đọc dữ liệu từ tập tin ta sẽ tạo lại cây nhu ban đầu.
d. Tìm cây con có tổng lớn nhất.

60. Tạo cây cân bằng từ danh sách liên kết kép có thứ tự tăng dần.

TÀI LIỆU THAM KHẢO

[1]. Hoàng Kiếm – Giải Một Bài Toán Trên Máy Tính Như Thế Nào – Tập 1 & Tập 2- Nhà Xuất Bản Giáo Dục -2001

[2]. Dương Anh Đức – Trần Hạnh Nhi – Giáo Trình Cấu Trúc Dữ Liệu Trừng Dại Học Khoa Học Tự Nhiên Tp.Hồ Chí Minh – 2003

[3]. Trần Đan Thư – Giáo Trình Lập Trình C – Tập 1& Tập 2 – Nhà Xuất Bản Đại Học Quốc Gia – 2001

[4]. Lê Hoài Bắc – Lê Hoàng Thái – Nguyễn Tấn Trần Minh Khang – Nguyễn Phương Thảo – Giáo Trình Lập Trình C – Nhà Xuất Bản Đại Học Quốc Gia Tp Hồ Chí Minh – 2003

[5]. Nguyễn Thanh Hùng – Các Bài Tập Tin Học Chọn Lọc – Nhà Xuất Bản Giáo Dục – 1996.

[6]. Nguyễn Tiến Huy – Tập Bài Giảng Kỹ Thuật Lập Trình – 2003.

[7]. Nguyễn Tấn Trần Minh Khang – Tập Bài Giảng Kỹ Thuật Lập Trình – 2003

[8]. Bùi Việt Hà – Lập Trình Pascal – Nhà Xuất Bản Giáo Dục – 2001

[9]. Phạm Văn ất – Kỹ Thuật Lập Trình C Cơ Sở Và Nâng Cao – Nhà Xuất Bản Khoa Học Và Kỹ Thuật – 1996.

[10]. Các Đề Thi Tốt Nghiệp Cao Đẳng Trường Đại Học Khoa Học Tự Nhiên Tp.Hồ Chí Minh.