

# KỸ THUẬT LẬP TRÌNH

GIẢNG VIÊN: NGUYỄN VĂN GIANG

# CHƯƠNG 1

## CÁC KHÁI NIỆM CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH C/C++

- 1.1. Khái quát về ngôn ngữ lập trình C/C++
- 1.2. Các kiểu dữ liệu cơ sở
- 1.3. Biểu thức toán tử câu lệnh

# 1.1.KHÁI QUÁT VỀ NGÔN NGỮ LẬP TRÌNH C/C++

## 1.1.1. GIỚI THIỆU

### 1.1.1.1. Các khái niệm cơ bản

**Chương trình (program):** Tập hợp mã nguồn nhằm giải quyết một vấn đề cụ thể.

File chương trình viết bằng ngôn ngữ C: \*.cpp.  
Sau khi được biên dịch: \*.exe

**Lập trình (programming):** Viết chương trình dựa vào một giải thuật (algorithm) và được cài đặt bằng một ngôn ngữ lập trình (programming language).

**Phần mềm (software):** Hệ thống các chương trình liên kết với nhau để giải quyết một công việc nào đó

### 1.1.1.2. Giới thiệu ngôn ngữ lập trình C/C++

C là một ngôn ngữ lập trình cấp cao được phát triển bởi Brian Kernighan và Dennis Ritchie làm việc tại AT&T Bell Labs của Mỹ năm 1972.

1990: C được tổ chức tiêu chuẩn quốc tế ISO chính thức công nhận với tên gọi là Standard C. C++ là một sự mở rộng của C chuẩn với mô hình lập trình hướng đối tượng (object-oriented programming).

### 1.1.1.3. Môi trường lập trình C/C++

**DevCpp (4.9.9.2)** là phần mềm mã nguồn mở của hãng Bloodshed Software.

**Code::Blocks 17.12** là phần mềm mã nguồn mở của Code Blocks

**Microsoft Visual Studio.Net 2010/2015/2017/2019** của Microsoft

## 1.1.2. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH C/C++

### 1.1.2.1. Bộ ký tự của C

Bộ chữ cái Latin: A, B, C, ..., Z, ..., a, b, c, ..., z

Bộ chữ số thập phân: 0, 1, 2, ..., 9

Các ký hiệu toán tử: +, -, \*, /, %, =, !, ...

Các ký hiệu khác: , ; : \_ ! # [ ] ( ) , % , ? & . . .

Dấu cách (space) dùng để cách các từ.

## 1.1.2.2. Các từ khóa của C

Là các từ dành riêng của C, không được dùng nó vào các việc khác. Thông thường các từ hóa của C đều dùng chữ thường

Ví dụ :

if, else, switch, case, do, while, for, return, break, continuous, include, typedef, struct, int, char, float,...

### 1.1.2.3. Tên và cách đặt tên

Tên của C được bắt đầu bằng một chữ cái hoặc dấu gạch dưới , tiếp theo có thể là chữ cái, chữ số, dấu gạch nối

- Không được có khoảng trống trong tên
- C phân biệt chữ hoa chữ thường đối với tên
- Khi viết chương trình ta nên đặt tên sao cho chúng nói lên được ý nghĩa của đối tượng mà chúng biểu diễn.

## 1.1.2.4. Cách ghi chú thích trong chương trình

Khi viết chương trình nên đưa vào các câu chú thích (comment) để làm cho chương trình rõ ràng, dễ hiểu.

Cách 1:

```
//Tac gia chuong trinh: Nguyen Thanh Nam  
//Lop 19T4-LTM1  
int main ()  
... •
```

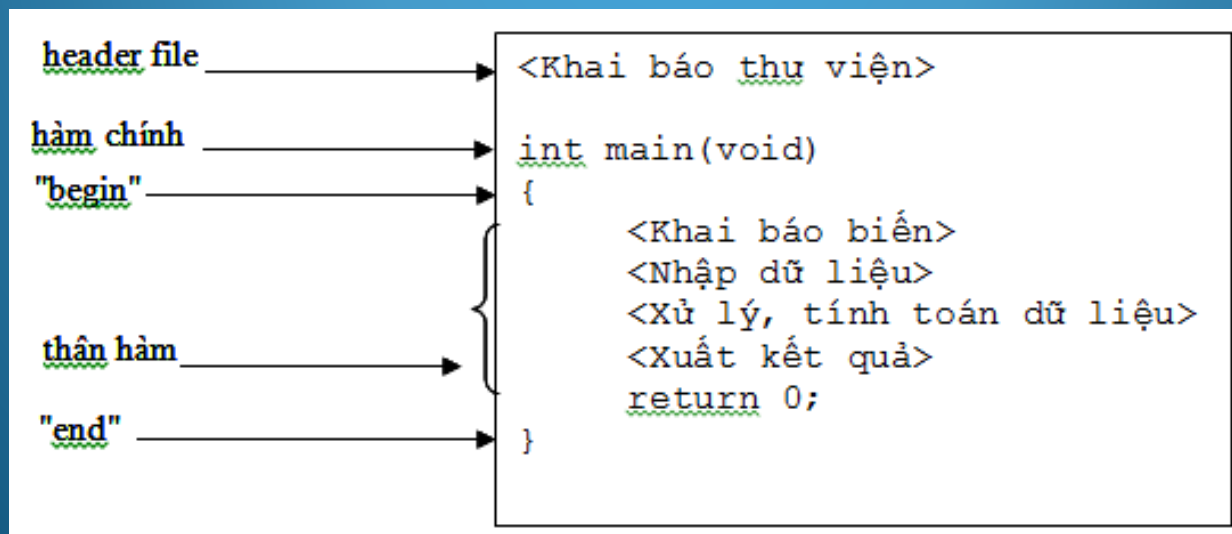
Cách 2:

```
/*Tac gia chuong trinh: Nguyen Thanh Nam  
Lop 19T4-LTM1*/  
int main ()  
... •
```

## 1.1.2.5. Câu lệnh và dấu chấm câu

Mỗi câu lệnh trong chương trình phải được kết thúc bằng dấu chấm phẩy (;) tuy nhiên có một số lệnh đặc biệt thì không có dấu chấm phẩy (;) ở cuối câu.

## 1.1.2.6. Cấu trúc của chương trình



## 1.1.2.7. Các bước lập trình cơ bản

1. Tìm hiểu và phân tích và mô hình hóa bài toán
2. Xây dựng thuật toán phù hợp để giải quyết bài toán
3. Cài đặt thuật toán bằng ngôn ngữ lập trình
4. Biên dịch chương trình và sửa lỗi
5. Chạy thử nghiệm và hoàn thiện chương trình
6. Tối ưu chương trình

## 1.1.3. CÁCH TẠO FILE MÃ NGUỒN



## 1.2. CÁC KIỂU DỮ LIỆU CƠ SỞ

### 1.2.1. Các kiểu dữ liệu cơ sở

#### 1.2.1.1. Kiểu ký tự (character)

Mỗi ký tự là một **ký hiệu** xác định được định nghĩa trong bảng mã chuẩn ASCII (xem phụ lục).

Mỗi ký tự có độ dài **1 byte** và tương ứng với một giá trị số là mã ASCII.

Mỗi ký tự được bao bằng dấu nháy đơn, ví dụ 'a', 'A', '1', '2', ..và có định dạng là **%c**.

Ngoài ra, cũng có thể dùng kiểu ký tự để biểu diễn số nguyên

| Kiểu          | Số byte | Minimum | Maximum | Định dạng |
|---------------|---------|---------|---------|-----------|
| char          | 1       | -128    | 127     | %d, %i    |
| unsigned char | 1       | 0       | 255     | %d, %i    |

## 1.2.1.2. Kiểu số nguyên (integer number)

| Kiểu               | Số byte  | Minimum                    | Maximum              | Định dạng         |
|--------------------|----------|----------------------------|----------------------|-------------------|
| short              | 2        | -32.768                    | 32.767               | %hi, %hd          |
| unsigned short     | 2        | 0                          | 65.535               | %hu               |
| <b>int</b>         | <b>4</b> | <b>-2.147.483.648</b>      | <b>2.147.483.647</b> | <b>%i,<br/>%d</b> |
| unsigned int       | 4        | 0                          | 4.294.967.296        | %u                |
| long               | 4        | -2.147.483.648             | 2.147.483.647        | %li, %ld          |
| unsigned long      | 4        | 0                          | 4.294.967.245        | %lu               |
| long long          | 8        | -9.223.372.036.854.775.808 | 9223372036854775807  | %lld              |
| unsigned long long | 8        | 0                          | 18446744073709551616 | %llu              |

## 1.2.1.2. Kiểu số thực (real number)

| Kiểu               | Số byte | Minimum                 | Maximum                | Định dạng     |
|--------------------|---------|-------------------------|------------------------|---------------|
| <b>float</b>       | 4       | $-3.4 \times 10^{38}$   | $3.4 \times 10^{38}$   | %f, %e, %g    |
| <b>double</b>      | 8       | $-1.7 \times 10^{308}$  | $1.7 \times 10^{308}$  | %lf, %le, %lg |
| <b>long double</b> | 10      | $-3.4 \times 10^{4932}$ | $3.4 \times 10^{4932}$ | %Lf, %Le, %Lg |

## 1.2.2. KHÁI NIỆM VỀ HẰNG VÀ BIẾN

### 1.2.2.1. Hằng (Constant)

Một hằng có thể là một ký tự, số nguyên, số thực  
Giá trị của hằng không thay đổi trong phạm vi mà hằng được định nghĩa.

Khai báo hằng:

***#define <Tên hằng> <giá trị>***

(Không có “;” ở cuối câu lệnh)

Ví dụ: Khai báo hằng hằng PI có giá trị là 3.14159

***#define PI 3.14159***

## 2.2.2.2. Biến (Variable)

Biến là một tên gọi đại diện cho một địa chỉ xác định trong bộ nhớ.

Biến được khai báo phải thuộc một kiểu dữ liệu xác định

Khai báo biến:

*<Kiểu dữ liệu> <Tên biến>;*

Ví dụ:

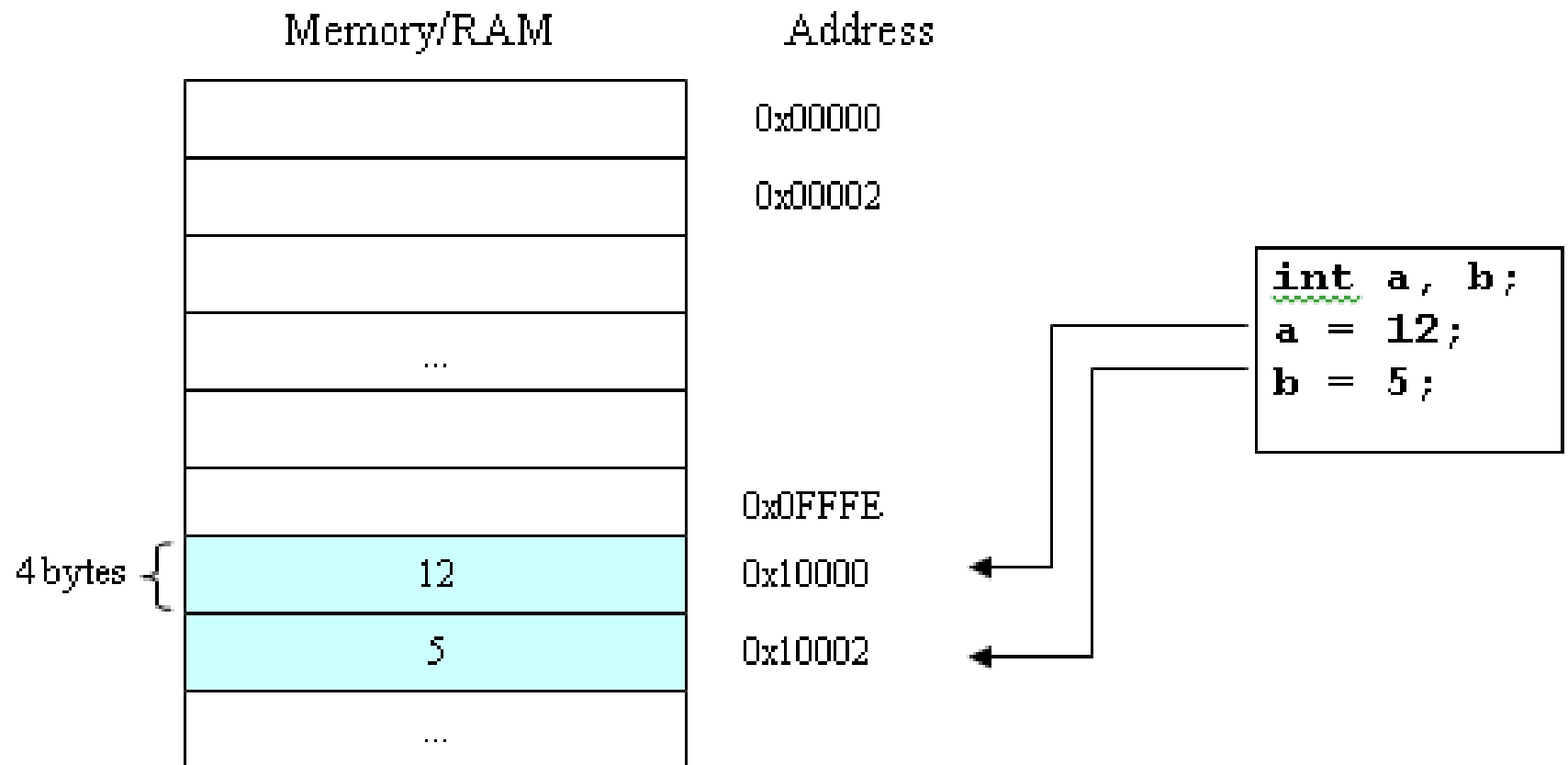
```
int n;
```

*<Kiểu dữ liệu> <Danh sách tên biến>;*

Ví dụ:

```
float x,y;
```

```
//float x;float y;
```



# 1.3. BIỂU THỨC – TÓÁN TỬ – CÂU LỆNH

## 1.3.1. BIỂU THỨC VÀ TÓÁN TỬ

### 1.3.1.1. Khái niệm về biểu thức

Biểu thức là một công thức tính toán bao gồm các phép toán, hằng, biến, hàm, và các dấu ngoặc.

Thứ tự ưu tiên:

1. Biểu thức trong ngoặc, phép gọi hàm,
2. !
3. / , \* , \ , % , &&
4. + , - , ||
5. < , > , == , <= , >= , !=

### 1.3.1.2. Toán tử

#### Toán tử số học:

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$  (chia lấy dư)

#### Toán tử quan hệ (so sánh):

$==$ ,  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $!=$  (không bằng)

Kết quả của phép so sánh là một giá trị logic true (=1) hoặc false (=0)

#### Toán tử logic: $!$ (NOT), $\&\&$ (AND), $\|\$ (OR)

Kết quả thực hiện phép toán logic là một giá trị logic true hoặc false.)

#### Toán tử tăng giảm: $++$ , $--$

Các toán tử này thực hiện việc tăng ( $++$ ) hay giảm ( $--$ ) giá trị của biến hay biểu thức 1 đơn vị.

## 1.3.2. Lệnh và khối lệnh

**Lệnh đơn** là một khai báo hay phát biểu được ghi trên 1 dòng và kết thúc bằng dấu chấm phẩy (;).

**Khối lệnh** là tập hợp các lệnh đơn và được bao trong cặp ngoặc nhọn ({}).

```

    int count=5; // Lệnh đơn

    printf("Cac so tu 5 den 0: \n"); // Lệnh đơn

    while(count>=0)
    {
        printf("%d, ", count); // Lệnh đơn
        count--; // Lệnh đơn
    }
  
```

**Khối lệnh** {

### 1.3.3. Chuyển đổi kiểu dữ liệu

(*<Kiểu dữ liệu đích>*) *<Biểu thức>*

# CHƯƠNG 2

## CÁC HÀM NHẬP XUẤT

2.1. HÀM printf() & scanf()

2.2. HÀM getche() & getch()

## 2.1. HÀM printf() & scanf()

### 2.1.1. Hàm printf() (#include <stdio.h>)

**printf("<Định\_dạng>", <Các\_biến>);**

*Công dụng: Xuất dữ liệu ra màn hình*

| Kiểu định dạng | Ý nghĩa                       |
|----------------|-------------------------------|
| %c             | Định dạng ký tự               |
| %d             | Định dạng số nguyên           |
| %f             | Định dạng số thực             |
| \t             | Cách khoảng tab               |
| \n             | Xuống dòng                    |
| %nd            | Dành n vị trí để in số nguyên |
| %.nf           | In số thực với n số lẻ        |

## 2.1.1. Hàm scanf() (#include <stdio.h>)

*scanf("<Định\_dạng>", <Địa\_chỉ\_các\_biến>);*

*Công dụng: Nhập dữ liệu từ bàn phím*

*Ví dụ 1:*

```
float x; int n;
```

```
scanf("%f", &x); scanf("%d", &n);
```

*Có thể dùng (nhưng không khuyến khích !):*

```
scanf("%f%d", &x, &n);
```

*Không được để khoảng trắng trước định dạng biến nhập:*

```
scanf(" %f", &x); // phải nhập khoảng trắng trước
```

*Nên: xuất câu thông báo trước khi yêu cầu người dùng nhập dữ liệu*

```
printf("Nhập số thực x:") scanf("%f", &x);
```

## 2.2. HÀM `getche()` và `getch()`

### 2.2.1. Hàm `getche()` (`#include <conio.h>`)

*Công dụng: Dùng để đọc một ký tự từ bàn phím và trả về ký tự đó. (ký tự nhập sẽ xuất hiện trên màn hình tại vị trí con trỏ)*

Ví dụ 1:

```
printf("Nhap ky tu bat ky: ");  
char k = getche();  
//k sẽ lưu ký tự mà người dùng nhập
```

Cách khác (không khuyến khích)

```
scanf("%c", &k);
```

## 2.2. HÀM getch() và getche()

### 2.2.1. Hàm getch() (#include <conio.h>)

*Công dụng: Dùng để đọc một ký tự từ bàn phím và trả về ký tự đó. (ký tự nhập sẽ **KHÔNG** xuất hiện trên màn hình tại vị trí con trỏ)*

Ví dụ 1:

```
int main()  
{ ...  
    getch();  
    return 0;  
}
```

Dùng getch() ở cuối hàm main() để chờ người dùng nhấn phím bất kỳ rồi mới thực hiện lệnh return 0; để thoát khỏi chương trình

# CHƯƠNG 3

## CẤU TRÚC ĐIỀU KHIỂN

3.1. Cấu trúc rẽ nhánh

3.2. Vòng lặp

## 3.1. CẤU TRÚC RỄ NHÁNH

### 3.1.1. Lệnh if...else

#### 3.1.1.1. Cú pháp

```
if(<điều kiện>)  
{  
    <phát biểu> ;  
}  
if(<điều kiện>)  
{  
    <phát biểu 1>;  
}  
else  
{  
    <phát biểu 2>;}
```

Nếu <điều kiện> có giá trị true thì thực hiện <phát biểu>

Nếu <điều kiện> có giá trị true thì thực hiện <phát biểu 1>, ngược lại thực hiện <phát biểu 2>

## 3.1.2. Lệnh switch...case

### 3.1.2.1. Cú pháp

**switch**(<biểu thức>)

{

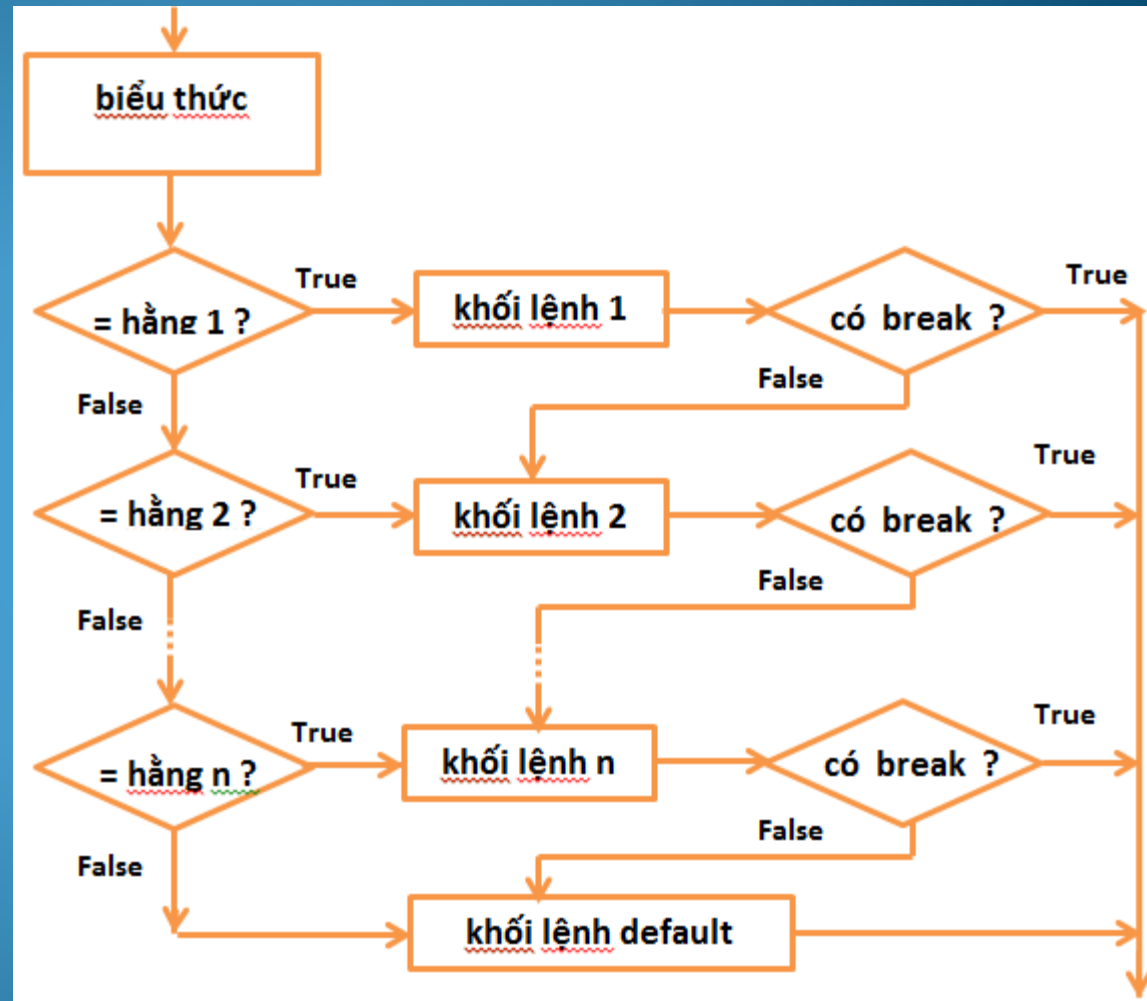
**case** <hằng 1>:  
    <khối lệnh 1>;  
    [break;]

**case** <hằng 2>:  
    <khối lệnh 2>;  
    [break;]

...

**case** <hằng n>:  
    <khối lệnh n>;  
    [break;]

**[default: <khối lệnh default>;]**



} Lưu ý: <biểu thức> có giá trị là số nguyên hoặc ký tự

**Bài tập :** Viết chương tạo menu cho phép người dùng chọn thực hiện việc tính diện tích hoặc chu vi tam giác như sau:

## CHƯƠNG TRÌNH TÍNH DIỆN TÍCH VÀ CHU VI TAM GIÁC

Nhập độ dài 3 cạnh của tam giác:

$$a = 3$$

$$b = 4$$

$$c = 5$$

Bạn muốn thực hiện chức năng nào sau đây:

1. Kiểm tra xem tam giác vuông/cân/đều
2. Tính chu vi tam giác
3. Tính diện tích tam giác
0. Thoát

Bạn chọn: 2

Chu vi tam giác = 12

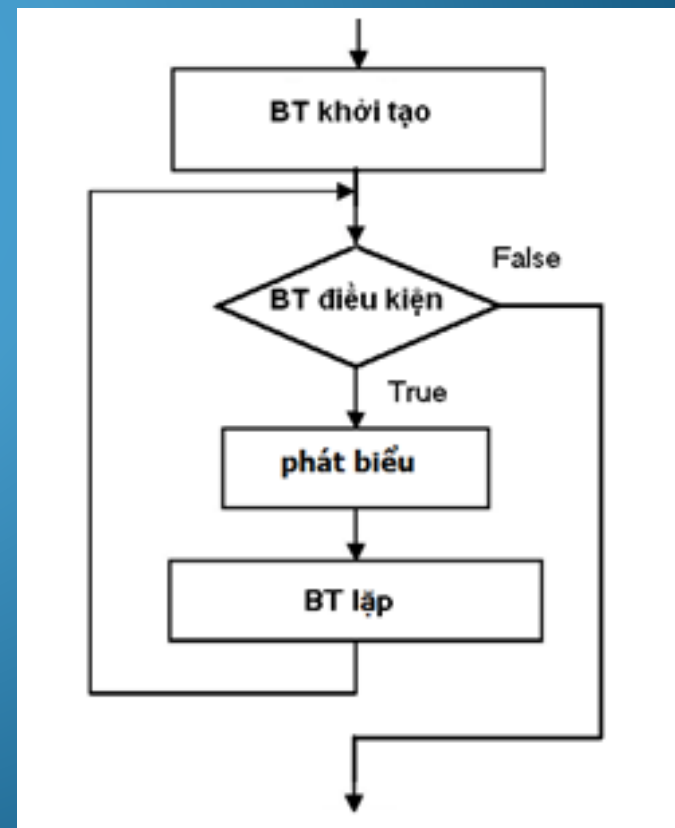
Hoàn thiện chương trình: Kiểm tra dữ liệu nhập phải là 3 cạnh của tam giác.

## 3.2. VÒNG LẶP

### 3.2.1. Vòng lặp for

#### 3.2.1.1. Cú pháp

```
for (<BT khởi tạo>;<BT điều kiện>;<BT lặp>)  
{  
    <Phát biểu> ;  
}
```

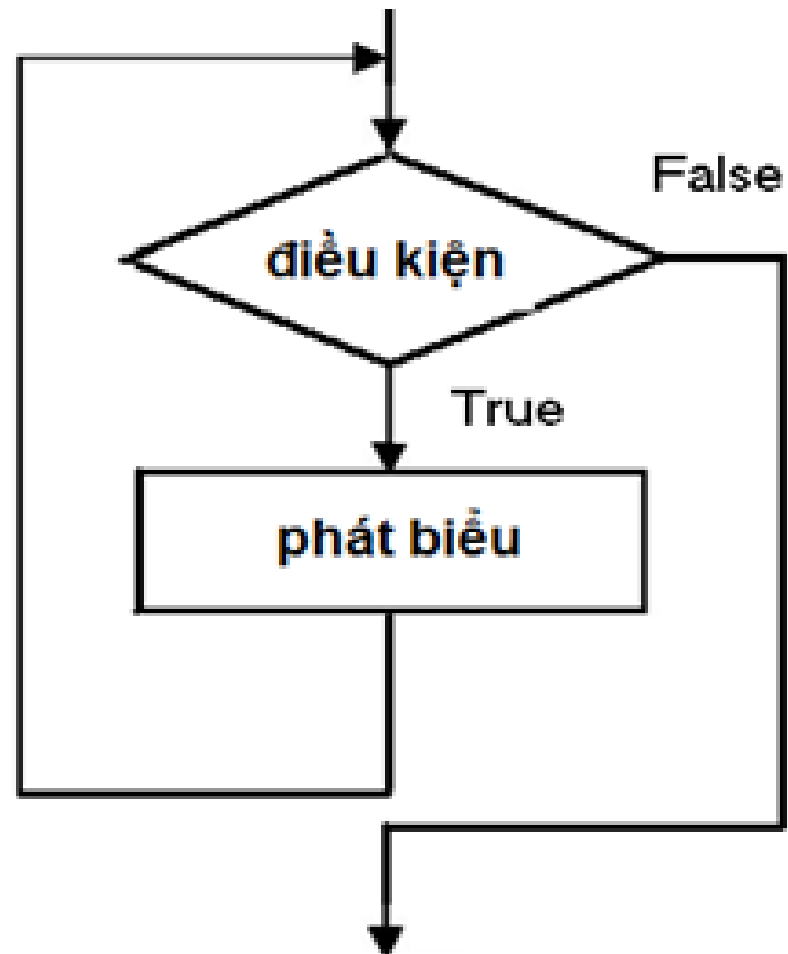


## 3.2. VÒNG LẶP

### 3.2.2. Vòng lặp while

#### 3.2.2.1. Cú pháp

```
while(<Điều kiện>)  
{  
    <Phát biểu> ;  
}
```

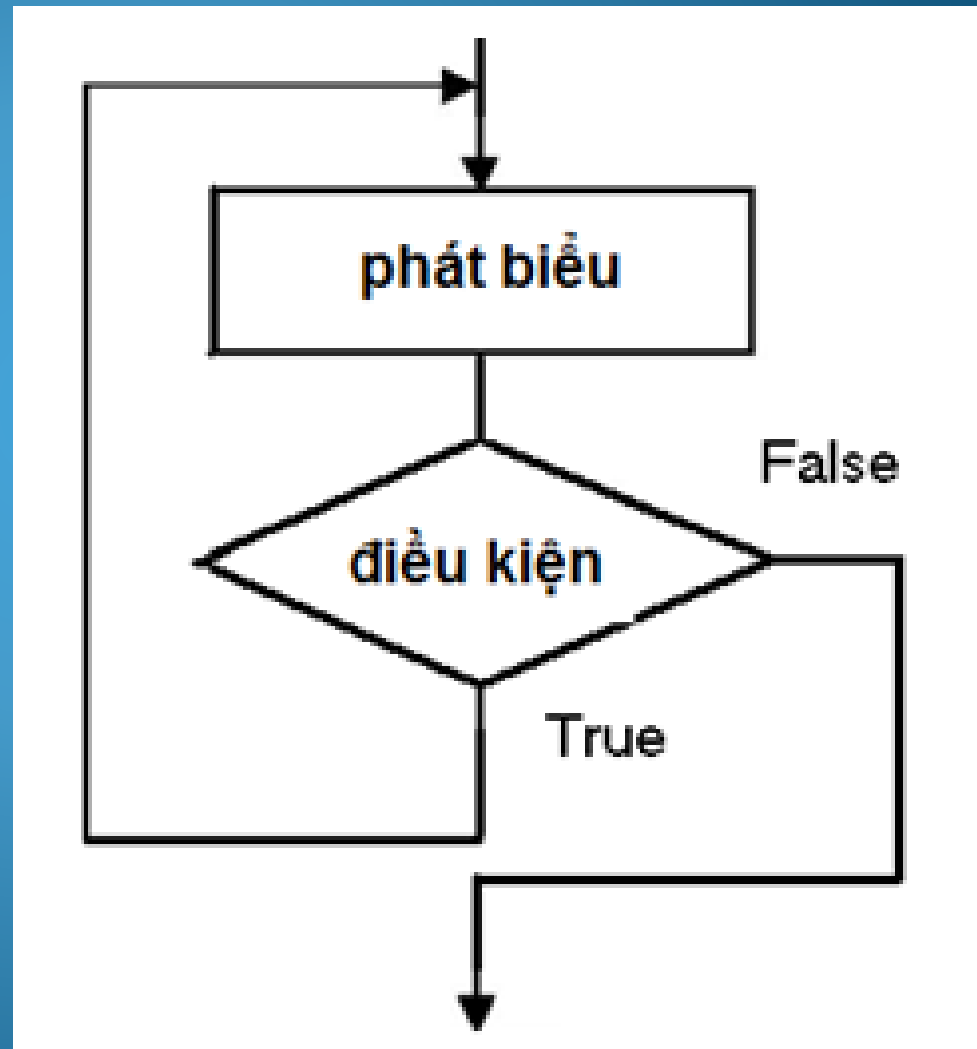


## 3.2. VÒNG LẶP

### 3.2.3. Vòng lặp do...while

#### 3.2.3.1. Cú pháp

```
do  
{  
    <Phát biểu> ;  
}while(<Điều kiện>);
```



# CHƯƠNG 4

## HÀM

- 4.1. Tổng quan về hàm trong C/C++
- 4.2. Tầm vực
- 4.3. Đệ quy

## 4.1. TỔNG QUAN VỀ HÀM TRONG C/C++

### 4.1.1. Khái niệm hàm (function)

#### 4.1.1.1. Khái niệm

Hàm là một chương trình con cho phép định nghĩa trước các phát biểu cần thực hiện nhằm giải quyết một nhiệm vụ xác định.

Trong một chương trình C, tối thiểu phải có một hàm, đó là hàm `main()`.

#### 4.1.1.2. Công dụng của hàm

- Làm cho việc phân tích, sửa lỗi, nâng cấp chương trình thuận tiện hơn.
- Tiết kiệm được thời gian lập trình do một hàm có thể được gọi tới nhiều lần trong một chương trình và có thể được dùng trong các chương trình khác nhau.
- Dễ dàng phân công công việc cho các lập trình viên cùng tham gia

## 4.1.2. Cấu trúc của hàm

### 4.1.2.1. Cấu trúc

Phần khai báo và mô tả hàm (prototype)

**<Kiểu DL trả về> <Tên hàm> ([<DS tham số>]);**

Ví dụ: Prototype của hàm tính Chu vi hình tròn:

**float ChuViHinhTron (float r);**

## Phần cài đặt hàm (Implementation)

```
<Kiểu DL trả về> <Tên hàm>([<DS tham số>])  
{  
    <Thân hàm>  
}
```

Ví dụ: Hàm tính Chu vi hình tròn:

```
float ChuViHinhTron (float r)  
{  
    return 2*Pi*r;  
}
```

#### 4.1.2.2. Cách trình bày phần khai báo và cài đặt hàm

##### Cách 1:

```
float ChuViHinhTron (float r)
{
    return 2*Pi*r;
}

int main()
{
    float r;
    printf(«Nhap ban kinh: ») ;
    scanf(«%f», &r) ;
    printf(«Chu vi=%.1f», ChuViHinhTron(r) ) ;
    getch() ;
    return 0;
}
```

## Cách 2:

```
float ChuViHinhTron (float r);
int main()
{
    ...
    printf («Chu vi=% .1f», ChuViHinhTron (r) );
    ...
}
float ChuViHinhTron (float r)
{
    return 2*Pi*r;
}
```

### 4.1.2.3. Lệnh *return* và *exit*

#### Lệnh *return*

`return [<Biểu thức>];`

#### Lưu ý:

<Biểu thức>: Có giá trị thuộc kiểu trả về của hàm.

Hàm có trả về giá trị thì trong thân hàm luôn có câu lệnh :

`return <biểu thức>;`

Gọi thực  
 hiện hàm add(),  
 điều khiển của  
 hàm main()  
 được chuyển  
 cho hàm add()

```
int main()
{
    float a=5, b=8, c;

    c=add(a,b) ;

    printf("%d", c) ;
    getch() ;
    return 0;
}
```

```
float add(float a, float b)
{
    return a+b;
}
```

Trả điều  
 khiển và giá  
 trị biểu thức  
a+b cho hàm  
 gọi là main()

### 4.1.2.3. Lệnh *return* và *exit*

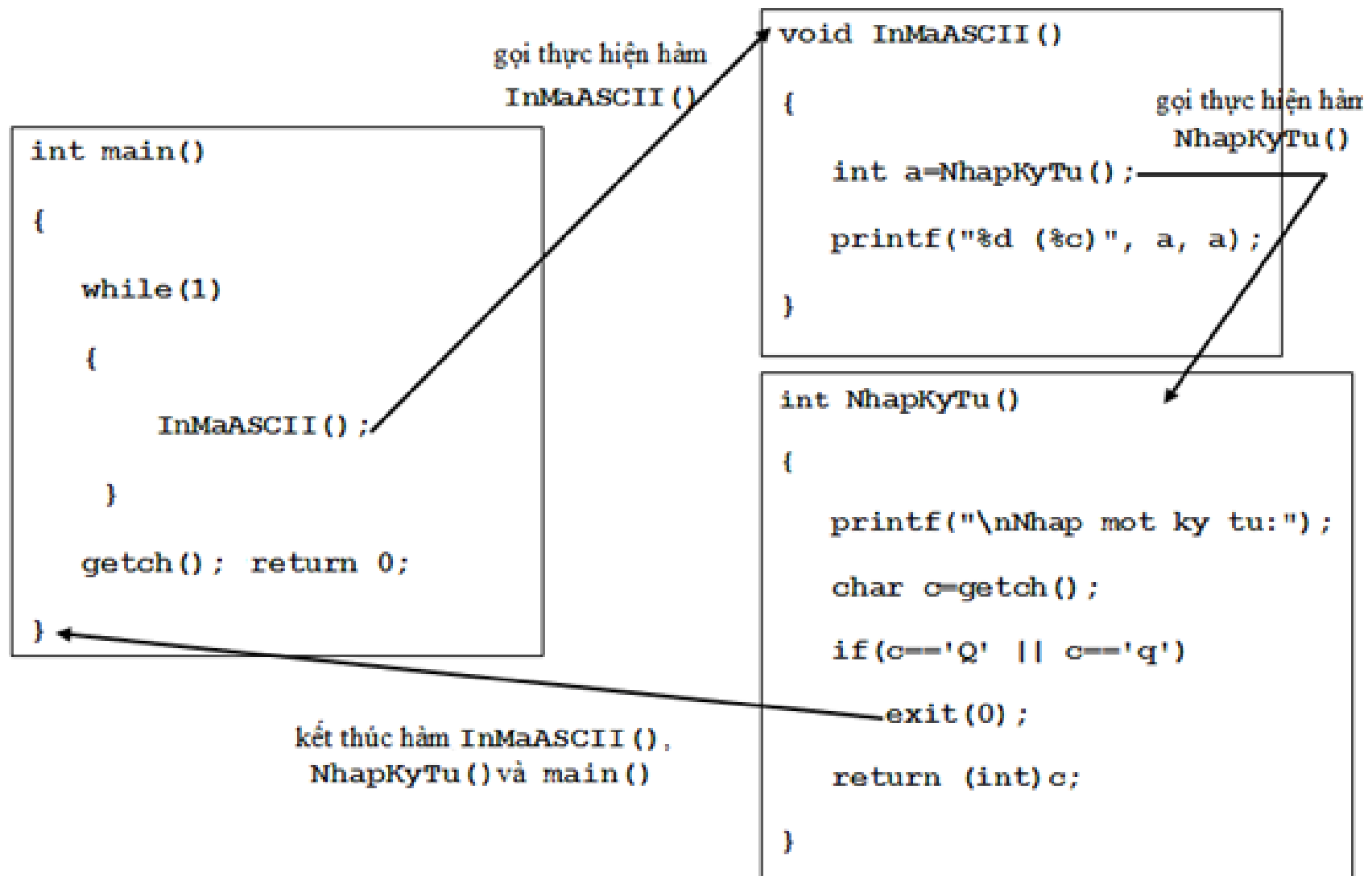
#### Hàm `exit()`

Dùng để kết thúc chương trình.

Thư viện chứa hàm `<stdlib.h>`

`exit(<trạng thái>);`

`<trạng thái>` : Trạng thái kết thúc quá trình, thường thì giá trị là 0 nếu không có lỗi, ngược lại có giá trị khác 0 .



## 4.1.3. Sử dụng hàm

### 4.1.3.1. Hàm không trả về giá trị

Cách gọi:

<Tên hàm> (<Tên các tham số>);

Ví dụ: Với hàm

```
void Swap(float &a, float &b);
```

Cách gọi:

```
float a=2,b=3,c=4,d=5;
```

```
Swap(a,b);
```

```
Swap(c,d);
```

```
Swap(b,a);
```

### 4.1.3.2. Hàm trả về giá trị

Cách 1: Gọi hàm trong câu lệnh if, printf() hay cout

Ví dụ:

```
int KiemTraTamGiac(float a, float b, float c);  
float DienTichTamGiac(float a, float b, float c);
```

Goi hàm cách 1:

```
if (KiemTraTamGiac(a,b,c)==1)  
    printf("S =%.1f",DienTichTamGiac(a,b,c));  
else  
    printf("Day khong la 3 canh cua tam giac");
```

Cách 2: Gán giá trị trả về của hàm vào biến tạm

Ví dụ:

Goi hàm cách 2:

```
int t=KiemTraTamGiac(a,b,c) ;
```

```
float S=DienTichTamGiac(a,b,c) ;
```

```
if (t==1)
```

```
    printf("S =%.1f",S) ;
```

```
else
```

```
    printf("Day khong la 3 canh cua tam giac") ;
```

## 4.1.4. Truyền tham số

Tham số (parameter/argument): là biến hay giá trị cần truyền cho hàm để hàm xử lý, tính toán

### 4.1.4.1. Tham số thực

Là tham số được truyền thực sự cho hàm thông qua lời gọi hàm.

### 4.1.4.2. Tham số hình thức

Là tham số được khai báo trong prototype mô tả hàm.

- Tên của tham số chỉ là tên hình thức nên có thể khác tên với các tham số thực.

a,b: tham số hình thức

```
void Sub (float &a, float b);
```

a tham số hình thức biến

b tham số hình thức trị

```
int main()
```

```
{
```

```
float x=5,y=7;
```

x,y: tham số thực

```
Sub (x,y);
```

```
printf("\nx=%.1f\ty=%.1f",x,y);
```

```
getch();
```

```
return 0;
```

```
}
```

## Tham số hình thức trị:

- Là tham số được truyền cho hàm bằng giá trị của tham số thực.

Khi đó sự thay đổi giá trị của tham số trong hàm nếu có sẽ không làm thay đổi giá trị của tham số thực đó trong hàm gọi.

## Tham số hình thức biến:

- Là tham số được truyền cho hàm bằng địa chỉ của tham số thực.

- Khi đó sự thay đổi giá trị tương ứng của tham số trong hàm (nếu có) sẽ làm thay đổi giá trị của tham số thực đó ở ngoài hàm.

Viết hàm kiểm tra số  $n$  có là số nguyên tố không ?

*Số nguyên tố là số nguyên lớn hơn 1 mà chỉ chia hết cho 1 và chính nó*

Ví dụ: 2,3,5,7,11,13 là các số nguyên tố

*Nhận xét: nếu  $n$  chia hết cho một số bất kỳ trong phạm vi từ 2 đến  $n-1$  thì  $n$  không là số nguyên tố*

## 4.2. VẤN ĐỀ TẦM VỰC

### 4.2.1. Khái niệm tầm vực

- Tầm vực của một đối tượng là phạm vi mà đối tượng đó được biết đến và có thể sử dụng được.

### 4.2.2. Quy tắc cơ bản về tầm vực

#### 4.2.2.1. Tầm vực của hàm

- Khi một hàm được mô tả trong một file (.cpp hay .h) thì tầm vực của hàm này là chính nó và tất cả các hàm nằm sau phần khai báo hay cài đặt của hàm này trong file đó.
- Nếu hàm được mô tả trong file .h thì tầm vực của hàm này bao gồm cả các file có dẫn hướng `#include` tới file chứa hàm này.

```
// mylib.h
#include <stdio.h>
#include <conio.h>
#include "mylib.h"
float NhapSo();
int main(void)
{
    float r, C, S;
    r = NhapSo();
    C = ChuVi(r);
    S = DienTich(r);
    printf("\n\n");
    printf("C = %-7.2f\n", C);
    printf("S = %-7.2f\n", S);
    getch();
    return 0;
}
float NhapSo()
{
    float r;
    do
    {
        printf("Ban kinh:\n");
        scanf("%f", &r);
    }while (r<0);
    return r;
}
```

```
// mylib.h
#define PI 3.14159

float ChuVi(float r)
{
    return 2*PI*r;
}

float DienTich(float r)
{
    return PI*r*r;
}
```

## 4.2.2.2. Tầm vực của biến:

### **Biến cục bộ :**

- Biến được khai báo trong một cấu trúc điều khiển hay hàm.
- Tầm vực của biến cục bộ là phạm vi của cấu trúc điều khiển hay hàm đó.

### **Biến toàn cục :**

- Biến được khai báo bên ngoài tất cả các hàm.
- Tầm vực của biến toàn cục là toàn bộ file chứa khai báo đó.
- Nếu biến cục bộ trùng tên với biến toàn cục thì biến toàn cục cùng tên sẽ bị "che",

## 4.3. ĐỆ QUY

### 4.3.1. Khái niệm

#### 4.3.1.1. Khái niệm về đệ quy

Hàm đệ quy là hàm mà trong thân hàm này có lời gọi hàm tới chính nó.

Ví dụ: Hàm tính  $n!$  dùng phương pháp đệ quy

```
int GiaiThua(int n)
{
    if(n==0) return 1;
    else
        return n*GiaiThua(n-1);
}
```

```
int GiaiThua(int n)
{
    if(n==0) return 1; // Phần dừng
    return n*GiaiThua(n-1); //Phần đệ quy
}
```

#### 4.3.1.2. Cấu trúc hàm đệ quy

Gồm 2 phần:

- **Phần dừng** : là điều kiện để kết thúc giải thuật và trong đó không bao gồm lời gọi hàm tới chính nó.
- **Phần đệ quy** : chứa lời gọi hàm tới chính hàm này để chỉ ra các bước đệ quy sau.

# CHƯƠNG 5

## MẢNG VÀ CHUỖI

5.1. Mảng

5.2. Chuỗi

## 5.1. MẢNG

### 5.1.1. Khái niệm

Mảng là một tập hợp các phần tử có cùng một kiểu dữ liệu.

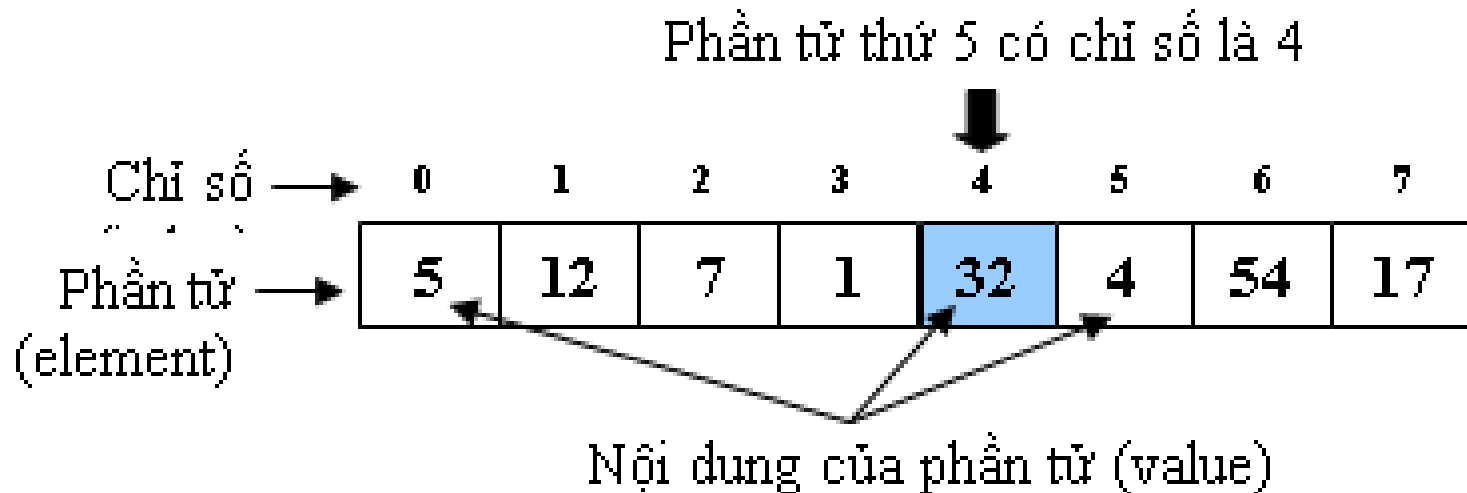
Trong C/C++, một mảng khi được khai báo sẽ được cấp một khối bộ nhớ liên tục.

## 5.1.2. Mảng một chiều

Là danh sách các phần tử sắp liên tục sao cho mỗi phần tử ứng với 1 vị trí trong mảng gọi là chỉ số hay chỉ mục .

Chỉ số của mảng luôn được đánh số từ 0.

Mảng các số nguyên với 8 phần tử được đánh chỉ số từ 0 đến 7:



## 5.1.2.1. Khai báo mảng 1 chiều

<Kiểu dữ liệu> <Tên\_mảng>[<Kích thước>;

Ví dụ 5.2:

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

```
float b[50];
```

```
int c[]={1,5,7};
```

(Mảng c ban đầu có 3 phần tử là 1,5,7)

### 5.1.2.3. Truy xuất phần tử của mảng

**<Tên mảng>[<Chỉ số>]**

Ví dụ 5.4:

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

Gán giá trị 20 cho phần tử a[2]:

```
a[2]=20;
```

In phần tử thứ i:

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

Nhập phần tử thứ i:

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

# Ví dụ: Hàm các nhập, xuất mảng số thực

| Chỉ số  | i    | 0 | 1   | 2 | 3  | 4  | 5 | 6 | 7 (n-1) |
|---------|------|---|-----|---|----|----|---|---|---------|
| Giá trị | a[i] | 3 | 7.1 | 0 | -1 | -2 | 4 | 9 | 2       |

```

D:\GIANG\BaiGiang\KTLT\ViDu\Chuong 4\Mang 1 chieu.exe
Nhap so phan tu: 8
a[0] = 3
a[1] = 7.1
a[2] = 0
a[3] = -1
a[4] = -2
a[5] = 4
a[6] = 9
a[7] = 2

Mang vua nhap:
3.0      7.1      0.0      -1.0      -2.0      4.0      9.0      2.0
  
```

## Bài tập áp dụng:

- Viết chương trình thực hiện các công việc sau (mỗi công việc viết 1 hàm riêng):

1. Nhập mảng số nguyên
2. Xuất mảng số nguyên
3. Xuất các số chẵn trong mảng
4. Đếm xem có bao nhiêu số chẵn trong mảng
5. Tính tổng các số chẵn trong mảng
6. Tính giá trị trung bình của mảng
7. Tìm số nhỏ nhất trong mảng
8. Xóa tất cả các số âm trong mảng
9. Xóa số ở vị trí thứ k trong mảng
10. Chèn một số x vào vị trí thứ k trong mảng

## 5.1.3. Mảng 2 chiều

Mảng 2 chiều dùng để lưu và xử lý dữ liệu dạng bảng hoặc ma trận.

Nói cách khác mảng 2 chiều là một bảng gồm có m dòng và n cột. Mỗi dòng trong mảng 2 chiều là một mảng 1 chiều.

### 5.1.3.1. Khai báo mảng 2 chiều

**<Kiểu DL> <Tên mảng> [<Số dòng>] [<Số cột>];**

Ví dụ :

```
int a[4][5];
```

```
float b[10][20];
```

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

[illegible]

## 5.1.3. Tham khảo phần tử của mảng 2 chiều

**<Tên mảng>[<Chỉ số dòng>][<Chỉ số cột>;**

Ví dụ: Cho mảng a[] như hình

Ta có : `a[1][2]=0`

Gán giá trị 2 cho phần tử `a[2][3]`:

**`a[2][3]=2;`**

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

In phần tử dòng i, cột j: **`printf("%d",a[i][j]);`**

Nhập phần tử dòng i, cột j:

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

## Bài tập áp dụng:

- Viết chương trình thực hiện các công việc sau (mỗi công việc viết 1 hàm riêng):

1. Nhập ma trận số nguyên
2. Xuất ma trận
3. Xuất các số chẵn trong ma trận
4. Đếm xem có bao nhiêu số chẵn trong ma trận
5. Tính tổng các số chẵn trong ma trận
6. Tính giá trị trung bình của ma trận
7. Tìm số nhỏ nhất trong ma trận
8. Tìm số nhỏ nhất của từng cột
9. Tìm số lớn nhất của từng dòng
10. Tính tổng từng cột của ma trận

## 5.2.CHUỖI (STRING)

### 5.2.1. KHÁI NIỆM

**Chuỗi là một mảng các ký tự**

Chuỗi được kết thúc bằng ký tự '\0'

|   |   |   |   |  |   |   |   |   |    |
|---|---|---|---|--|---|---|---|---|----|
| K | h | o | a |  | C | N | T | T | \0 |
|---|---|---|---|--|---|---|---|---|----|

### 5.2.2. KHAI BÁO CHUỖI

**char <Tên\_chuỗi> [<Độ\_dài>];**

Ví dụ: char s[100];

**Khai báo và gán giá trị ban đầu:**

```
char s[100]={'K','h','o','a',' ','C','N','T','T','\0'};
```

```
char s[100]="Khoa CNTT";
```

```
char s[]="Khoa CNTT";
```

## 5.2.3. CÁC HÀM XỬ LÝ CHUỖI

### 5.2.3.1. Hàm nhập chuỗi

**gets(<biến\_chuỗi>);**      // <stdio.h>

Ví dụ:

```
char name[50];  
printf("Nhap chuoi ho ten: ");  
gets(name);
```

**Lưu ý:**

1. Nếu trước khi dùng hàm gets() có sử dụng hàm **scanf()** hay **cin** thì phải làm sạch bộ đệm bàn phím bằng câu lệnh:

– **fflush(stdin)**      // <stdlib.h>

2. Nếu dùng `scanf("%s",name);` hay `cin>>name;` sẽ không nhận được chuỗi có khoảng trắng

## 5.2.3.2. Hàm xuất chuỗi

```
puts(<chuỗi>; // <stdio.h>
```

Ví dụ:

```
char name[50]="Ly Tu Trong";
```

Cách 1:

```
puts(name);
```

Cách 2:

```
printf("%s", name);
```

**Lưu ý:** Sau khi xuất xong puts() tự động xuống dòng

### 5.2.3.3. Hàm tính chiều dài chuỗi

**int strlen(<chuỗi>);** //<string.h>

*Trả về chiều dài chuỗi*

Ví dụ:

```
char name[50]="Ly Tu Trong";
```

```
int len = strlen(name);
```

→ len = 11

### 5.2.3.4. Hàm sao chép chuỗi

**strcpy(<chuỗi\_đích>,<chuỗi\_nguồn>);** //<string.h>

*Chép <chuỗi\_nguồn> vào <chuỗi\_đích>*

Ví dụ:

```
char s1[]="Hello, World!", s2[30];
```

```
strcpy(s2, s1);
```

```
printf("%s", s2);
```

→ s2 = "Hello, World!"

### 5.2.3.5. Hàm nối chuỗi

**strcat(<chuỗi\_đích>, <chuỗi\_nguồn>);** //<string.h>

Nối <chuỗi\_nguồn> vào cuối <chuỗi\_đích>

Ví dụ:

```
char s1[30]="Hello ", s2[]="World!"
```

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

→ s1 = "Hello World!"

### 5.2.3.6. Hàm so sánh chuỗi

**int strcmp(s1,s2);** //<string.h>

- Trả về 0 nếu s1 = s2
- Trả về số dương nếu s1 > s2
- Trả về số âm nếu s1 < s2

Ví dụ :

```
strcmp("Ly Tu Trong","ly tu trong")<0
```

```
strcmp("Ly Tu Trong","Tu trong")<0
```

### 5.2.3.7. Hàm đổi chuỗi sang chữ hoa

**strupr(<chuỗi>);**

*Đổi các ký tự trong <chuỗi> sang dạng viết hoa.*

Ví dụ :

```
char name[]="Ly TU Trong";
```

```
strupr(name);
```

→ name = "LY TU TRONG"

## 5.2.3.8. Hàm đổi chuỗi sang chữ thường

**strlwr(<chuỗi>);**

*Đổi các ký tự trong <chuỗi> sang dạng viết thường.*

Ví dụ :

```
char name[]="Ly TU Trong";
```

```
strlwr(name);
```

```
→ name = "ly tu trong"
```

# CHƯƠNG 6

## CON TRỎ

### 6.1. KHÁI QUÁT VỀ CON TRỎ

#### 6.1.1. khái niệm

Con trỏ là biến lưu địa chỉ bộ nhớ của một dữ liệu nào đó.

Dùng con trỏ ta có thể truy cập biến thông qua địa chỉ của biến.

## 6.1.2. Lý do dùng con trỏ

- Giúp cho việc sử dụng bộ nhớ tối ưu hơn. Các biến con trỏ sẽ được cấp phát một cách động trong vùng nhớ heap.
- Việc dùng con trỏ cho mảng , chuỗi , cấu trúc sẽ làm cho việc xử lý dễ dàng và linh hoạt hơn.
- Đối với các cấu trúc dữ liệu có độ phức tạp cao như danh sách liên kết và cây bắt buộc phải dùng con trỏ.

### 6.1.3. Khai báo biến con trỏ

<Kiểu dữ liệu> \*<Tên biến con trỏ>;

Ví dụ :

**int \*x,\*y;** //x,y là con trỏ lưu địa chỉ của một số nguyên

**float \*z;** //z là con trỏ lưu địa chỉ của một số thực

**char \*s;** //s là con trỏ lưu địa chỉ của một ký tự

### 6.1.4. Sử dụng biến con trỏ trong các biểu thức

**\*** (Tóan tử gián tiếp) : Giá trị của ô nhớ mà biến con trỏ lưu địa chỉ

**&** (Tóan tử địa chỉ) : Địa chỉ của biến thông thường

| <code>int i=234;</code>   | <code>int *p=&amp;i;</code> |
|---------------------------|-----------------------------|
| <code>i=234</code>        | <code>*p=234</code>         |
| <code>&amp;i=10120</code> | <code>p=10120</code>        |

| Bộ nhớ | Địa chỉ | Biến     |
|--------|---------|----------|
| ...    |         |          |
| 234    | 10120   | <u>i</u> |
|        | 10124   |          |
| ...    | ...     |          |
|        | 31200   |          |
|        | 31202   |          |
| 10120  | 31204   | <u>p</u> |
| ...    |         |          |

`p = &i;`

## 6.1.5. Cấp phát và giải phóng vùng nhớ

Cấp phát vùng nhớ cho <Biến con trỏ>

**<Biến con trỏ> = new <Kiểu dữ liệu>;**

Giải phóng vùng nhớ cho <Biến con trỏ>

**delete <Biến con trỏ>;**

Ví dụ:

```
int *p;  
p=new int;  
...  
delete p;
```

## 6.2. ỨNG DỤNG CỦA CON TRỎ

### 6.2.1. Hàm có tham số con trỏ

```
void add(int *num1, int num2)
{
    *num1 = *num1 + num2;
}

int main()
{
    int *n1, n2=3;
    n1=new int;
    *n1=5;
    add(n1, n2);
    printf("n1=%d, n2=%d\n", n1, n2);
    getch();
    return 0;
}
```

```

void main(void)
{
    int n1, n2;

    n1 = 5; n2 = 3;

    add(&n1, n2);

    printf("n1=%d, n2=%d\n", n1, n2);
}

```

n1  
 5  
 0x0012FED4

n2  
 3  
 0x0012FEC8

n1, n2: tham số thực  
 num1, num2: tham số hình thức

0x0012FED4

3

```

void add(int *num1, int num2)
{
    *num1 = *num1 + num2;
}

```

num1  
 0x0012FED4  
 0x0012FDE4

0x0012FED4  
~~5~~ 8

num2  
 3  
 0x0012FDE8

## 6.2.2. Con trỏ và mảng:

Ví dụ:

Khai báo mảng thông thường:

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

Khai báo mảng dùng con trỏ:

```
int *a, n=100;
```

```
a = new int[n];
```

| Thao tác                   | Ký hiệu thông thường   | Sử dụng con trỏ     |
|----------------------------|------------------------|---------------------|
| Tham khảo địa chỉ phần tử  | <code>&amp;a[i]</code> | <code>a+i</code>    |
| Tham khảo nội dung phần tử | <code>a[i]</code>      | <code>*(a+i)</code> |

Ví dụ:

```
void Input(float *&a, int &n)
{
    printf("Nhap vao so phan tu: ");
    scanf("%d", &n);
    a = new float[n];
    for(int i=0; i<n; i++)
    {
        printf("a[%d] =", i);
        scanf("%f", (a+i));
        //scanf("%f", &a[i]);
    }
}
```

# CHƯƠNG 7: KIỂU DỮ LIỆU DO NGƯỜI DÙNG ĐỊNH NGHĨA

## 7.1. KIỂU CẤU TRÚC (struct)

- struct là kiểu dữ liệu gồm nhiều thành phần, mỗi thành phần có thể có kiểu dữ liệu khác nhau.

### 7.1.1. Khai báo cấu trúc dữ liệu

```
typedef struct <Tên kiểu>
{
    <Mô tả các thành phần>;
};
```

```
typedef struct SV
```

```
{
```

```
    char    maSV[10];
```

```
    char    tenSV[50];
```

```
    int     NS;
```

```
    float   Diem;
```

```
};
```

Khai báo biến kiểu SV:

```
SV t;
```

```
SV *p;
```

```
SV a[100];
```

## 7.1.2. Tham khảo các thành phần của struct

Biến thông thường:

**<Tên biến>.<Thành phần>;**

Nếu là biến con trỏ:

**<Tên biến> -> <Thành phần>;**

## Bài tập vận dụng mảng struct:

Viết chương trình cho phép người dùng thực hiện các công việc sau:

1. Nhập danh sách các mặt hàng (MH) gồm mã hàng (MaMH), tên hàng (TenMH), đơn vị tính (DVT), số lượng tồn kho (SL), giá theo đơn là 1.000 đ (Gia)

2. Xuất danh sách các mặt hàng theo dạng bảng như sau:

| STT | Ma hang | Ten Hang    | DVT | SL  | Gia | Thanh Tien |
|-----|---------|-------------|-----|-----|-----|------------|
| 1   | MH01    | Duong trang | kg  | 100 | 20  | 2000       |
| 2   | MH02    | Dau an      | lit | 200 | 30  | 6000       |
| 3   | MH03    | Gao thom    | kg  | 100 | 25  | 2500       |
| ... |         |             |     |     |     |            |

3. Xuất các mặt hàng có đơn vị tính là “kg”

4. Tính tổng giá trị tồn kho của các mặt hàng có đơn vị tính là “kg”

5. Giảm giá 10% tất cả các mặt hàng có đơn vị tính là “kg”

6. Tìm mặt hàng có số lượng tồn kho nhỏ nhất

7. Tìm mặt hàng có thành tiền lớn nhất

8. Xóa tất cả các mặt hàng có số lượng tồn kho bằng 0