

TRƯỜNG CAO ĐẲNG KỸ THUẬT CAO THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN



**BÀI GIẢNG**

# Nhập môn lập trình

(Lưu hành nội bộ)

TP. HỒ CHÍ MINH, 2020

# MỤC LỤC

<b>MỤC LỤC.....</b>	<b>II</b>
<b>LỜI GIỚI THIỆU.....</b>	<b>V</b>
<b>CHƯƠNG 1 - TỔNG QUAN VỀ LẬP TRÌNH.....</b>	<b>6</b>
1.1. KHÁI NIỆM LẬP TRÌNH.....	6
1.2. THUẬT TOÁN VÀ BIỂU DIỄN THUẬT TOÁN .....	6
1.2.1. Thuật toán.....	6
1.2.2. Các tính chất của thuật toán.....	6
1.2.3. Biểu diễn thuật toán.....	6
1.3. CÁC BƯỚC XÂY DỰNG CHƯƠNG TRÌNH.....	11
<b>BÀI TẬP CHƯƠNG 1 .....</b>	<b>12</b>
<b>CHƯƠNG 2 – GIỚI THIỆU NGÔN NGỮ C++.....</b>	<b>13</b>
2.1. CẤU TRÚC MỘT CHƯƠNG TRÌNH C++ .....	13
2.2. LỆNH NHẬP/XUẤT .....	14
2.2.1. Lệnh xuất.....	14
2.2.2. Lệnh nhập.....	15
2.3. CÁC KIỂU DỮ LIỆU CƠ SỞ.....	16
2.3.1. Biến, hằng .....	17
2.3.2. Biến .....	17
2.3.3. Hằng.....	17
2.3.4. Định danh (Identifier).....	18
2.3.5. Chuyển đổi kiểu dữ liệu .....	19
2.4. CÁC TOÁN TỬ.....	20
2.4.1. Các toán tử số học.....	20
2.4.2. Toán tử gán.....	21
2.4.3. Các toán tử gán tắt.....	22
2.5. MỘT SỐ HÀM TOÁN HỌC .....	22

<b>BÀI TẬP CHƯƠNG 2 .....</b>	<b>23</b>
<b>CHƯƠNG 3 - CẤU TRÚC Rẽ NHÁNH.....</b>	<b>24</b>
3.1. CÁC PHÉP TOÁN QUAN HỆ.....	24
3.2. CÁC PHÉP TOÁN LOGIC .....	24
3.2.2. <i>Phép toán AND (&amp;&amp;)</i> .....	24
3.2.3. <i>Phép toán OR (  )</i> .....	25
3.2.4. <i>Phép toán NOT (!)</i> .....	25
3.3. CÂU LỆNH IF .....	25
3.3.1. <i>Dạng 1: if</i> .....	26
3.3.2. <i>Dạng 2: if ... else</i> .....	27
3.3.3. <i>Nhiều cấu trúc if ... else lồng nhau</i> .....	29
3.4. CÂU LỆNH SWITCH.....	32
3.5. TOÁN TỬ ĐIỀU KIỆN .....	34
<b>BÀI TẬP CHƯƠNG 3 .....</b>	<b>36</b>
<b>CHƯƠNG 4 – CẤU TRÚC LẶP .....</b>	<b>38</b>
4.1. TOÁN TỬ TĂNG, GIẢM .....	38
4.2. CÂU LỆNH WHILE.....	39
4.3. CÂU LỆNH DO... WHILE.....	40
4.4. CÂU LỆNH FOR .....	41
4.5. CÁCH SỬ DỤNG LỆNH LẶP.....	42
4.5.1. <i>Từ khóa break</i> .....	42
4.5.2. <i>Từ khóa continue</i> .....	43
<b>BÀI TẬP CHƯƠNG 4 .....</b>	<b>45</b>
<b>CHƯƠNG 5 – HÀM (FUNCTION).....</b>	<b>47</b>
5.1. ĐỊNH NGHĨA VÀ LỜI GỌI HÀM.....	47
5.2. KHAI BÁO HÀM.....	49
5.3. TẦM VỰC CỦA BIẾN.....	49
5.3.1. <i>Biến toàn cục</i> .....	49
5.3.2. <i>Biến cục bộ</i> .....	50
5.4. CÁC CÁCH TRUYỀN THAM SỐ.....	51

5.4.1. Truyền tham trị.....	51
5.4.2. Truyền địa chỉ.....	57
5.4.3. Truyền tham chiếu.....	58
5.5. Đệ QUI .....	62
<b>BÀI TẬP CHƯƠNG 5 .....</b>	<b>64</b>
<b>CHƯƠNG 6 - MẢNG (ARRAY) .....</b>	<b>66</b>
6.1. MẢNG MỘT CHIỀU.....	66
6.2. KHỞI TẠO MẢNG .....	67
6.3. TRUYỀN THAM SỐ MẢNG CHO HÀM.....	67
6.4. CHUỖI – MẢNG KÝ TỰ.....	70
6.4.1. Hằng chuỗi.....	70
6.4.2. Lưu trữ chuỗi trong mảng .....	70
6.4.3. Các hàm thư viện xử lý chuỗi .....	71
6.5. LỚP STRING.....	73
6.6. MẢNG 2 CHIỀU.....	76
<b>BÀI TẬP CHƯƠNG 6 .....</b>	<b>81</b>
<b>CHƯƠNG 7 – KIỂU DỮ LIỆU CẤU TRÚC (STRUCT).....</b>	<b>82</b>
7.1. GIỚI THIỆU .....	82
7.2. KHAI BÁO BIẾN CẤU TRÚC.....	83
7.3. TRUY XUẤT CÁC THÀNH PHẦN CẤU TRÚC.....	85
7.4. KHỞI TẠO CẤU TRÚC.....	85
7.5. MẢNG CẤU TRÚC.....	88
<b>BÀI TẬP CHƯƠNG 7 .....</b>	<b>91</b>
<b>CHƯƠNG 8 – CON TRỎ (POINTER).....</b>	<b>92</b>
8.1. BIẾN CON TRỎ.....	92
8.2. MẢNG CON TRỎ.....	94
8.3. CÁC BIẾN ĐỘNG .....	96
8.4. CON TRỎ VỚI KIỂU DỮ LIỆU CẤU TRÚC .....	97
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>103</b>

# LỜI GIỚI THIỆU

Nhập môn lập trình là môn học cung cấp cho sinh viên kiến thức và kỹ năng cơ bản nhất về lập trình máy tính. Môn học này được giảng dạy cho tất cả các chuyên ngành của ngành công nghệ thông tin trường Cao đẳng KT Cao Thắng.

Sau một thời gian tìm hiểu, làm việc và tham gia giảng dạy môn nhập môn lập trình cho sinh viên của ngành công nghệ thông tin. Chúng tôi quyết định biên soạn cuốn bài giảng này nhằm phục vụ công tác giảng dạy cũng như học tập của sinh viên ngành công nghệ thông tin. Nội dung bài giảng tập trung vào những kiến thức cơ bản về lập trình máy tính và ngôn ngữ lập trình C++.

Đây là lần đầu tiên xuất bản, nên chắc chắn không thể tránh khỏi những sai sót. Nhóm tác giả rất mong nhận được những ý kiến đóng góp của quý thầy cô, các đồng nghiệp và sinh viên để có thể hoàn thiện hơn bài giảng này phục vụ tốt hơn cho việc học tập của sinh viên.

Trân trọng cảm ơn./.

Tp. Hồ Chí Minh, tháng 8/2020

**Nhóm tác giả**

# CHƯƠNG 1 - TỔNG QUAN VỀ LẬP TRÌNH

## ❖ Mục tiêu:

- Ghi nhớ các khái niệm liên quan đến môn học.
- Giải thích được ưu, khuyết điểm của các phương pháp biểu diễn thuật toán.
- Áp dụng được các phương pháp biểu diễn thuật toán để biểu diễn thuật toán cho một số bài toán cụ thể.

## 1.1. Khái niệm lập trình

Lập trình là nghệ thuật cài đặt một hay nhiều thuật toán trừu tượng có liên quan với nhau bằng một ngôn ngữ lập trình để tạo ra một chương trình máy tính.

## 1.2. Thuật toán và biểu diễn thuật toán

### 1.2.1. Thuật toán

Thuật toán là dãy các bước xử lý nhằm giải quyết một vấn đề nào đó. Thuật ngữ này được dịch từ từ tiếng Anh “Algorithm”.

### 1.2.2. Các tính chất của thuật toán

Một thuật toán cần bảo đảm các tính chất sau:

- Tính hữu hạn: Thuật toán phải kết thúc thực thi sau một số lượng hữu hạn các bước xử lý
- Tính xác định: Mỗi bước xử lý phải được mô tả rõ ràng, chính xác, không nhập nhằng.
- Tồn tại dữ liệu đầu vào: Thuật toán phải có dữ liệu đầu vào hợp lệ, được mô tả rõ ràng.
- Tính có kết quả: Thuật toán phải cho ra kết quả đúng trên cơ sở dữ liệu đầu vào hợp lệ.
- Tính hiệu quả: Mỗi bước xử lý phải đơn giản với thời gian thực thi hữu hạn, trong thực tế điều này có nghĩa là phải thực thi trong một khoảng thời gian có thể chấp nhận được.
- Tính phổ dụng: Thuật toán có thể áp dụng để xử lý một họ các bài toán.

### 1.2.3. Biểu diễn thuật toán

#### 1.2.3.1. Sử dụng ngôn ngữ tự nhiên

Sử dụng ngôn ngữ giao tiếp hàng ngày để diễn đạt các bước thực hiện của thuật toán.

- **Ưu điểm: đơn giản, dễ sử dụng**
- **Khuyết điểm: thường là rườm rà và dễ bị nhập nhằng trong diễn đạt.**

**Ví dụ 1:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán tính tổng hai số nguyên  $a, b$ .

- ❖ Đầu vào: 2 số nguyên  $a, b$
- ❖ Đầu ra: Tổng của 2 số nguyên  $a, b$ .
- ❖ Thuật toán:
  - Bước 1: Nhập giá trị của  $a, b$ .
  - Bước 2: Tính Tổng  $= a + b$ .
  - Bước 3: Thông báo kết quả Tổng
  - Bước 4: Kết thúc.


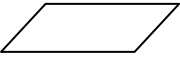
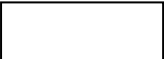
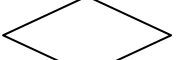


**Ví dụ 2:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán giải phương trình bậc nhất  $ax + b = 0$  ( $a, b \in \mathbb{R}$ )

- ❖ Đầu vào: 2 số thực  $a, b \in \mathbb{R}$
- ❖ Đầu ra: Nghiệm của phương trình bậc nhất  $ax + b = 0$ .
- ❖ Thuật toán:
  - Bước 1: Nhập giá trị của  $a, b$ .
  - Bước 2: Nếu  $a = 0$  thì
    - Bước 2.1: Nếu  $b = 0$  thì
      - Bước 2.1.1: Thông báo “Phương trình vô số nghiệm”
      - Bước 2.1.2: Kết thúc
    - Bước 2.2: Ngược lại ( $b \neq 0$ )
      - Bước 2.2.1: Thông báo “Phương trình vô nghiệm”
      - Bước 2.2.2: Kết thúc
  - Bước 3: Ngược lại ( $a \neq 0$ )
    - Bước 3.1: Tính  $x = \frac{-b}{a}$
    - Bước 3.2: Thông báo nghiệm là  $x$
    - Bước 3.3: Kết thúc.

### 1.2.3.2. Sử dụng lưu đồ (flow chart)

Lưu đồ được sử dụng thông dụng trong việc trình bày các bước cần thiết để giải quyết vấn đề qua các hình khối khác nhau và dòng dữ liệu giữa các bước được chỉ định đi theo các đường mũi tên

Một số qui ước ký hiệu lưu đồ:

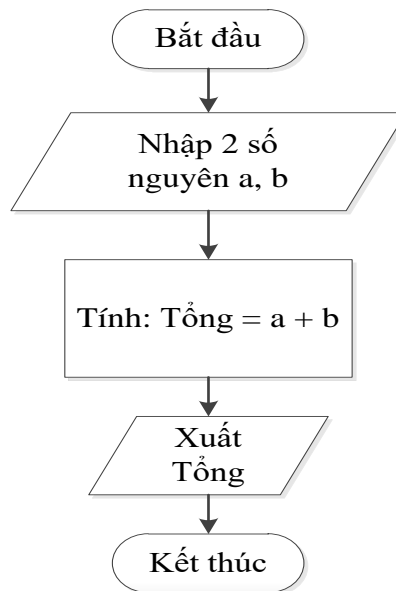
Ký hiệu	Mô tả
	Điểm bắt đầu và kết thúc một thuật toán.
	Thao tác nhập hay xuất dữ liệu.
	Khối xử lý công việc.
	Khối quyết định chọn lựa.
	Dòng tính toán, thao tác của chương trình.
	Khối lệnh gọi hàm (chương trình con)

- **Ưu điểm:** tránh được hầu hết vấn đề nhập nhằng trong diễn đạt.
- **Khuyết điểm:** Khó có thể chỉnh sửa các ký hiệu, phần lớn được chuẩn hóa.

**Ví dụ 1.1:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán tính tổng hai số nguyên a, b.

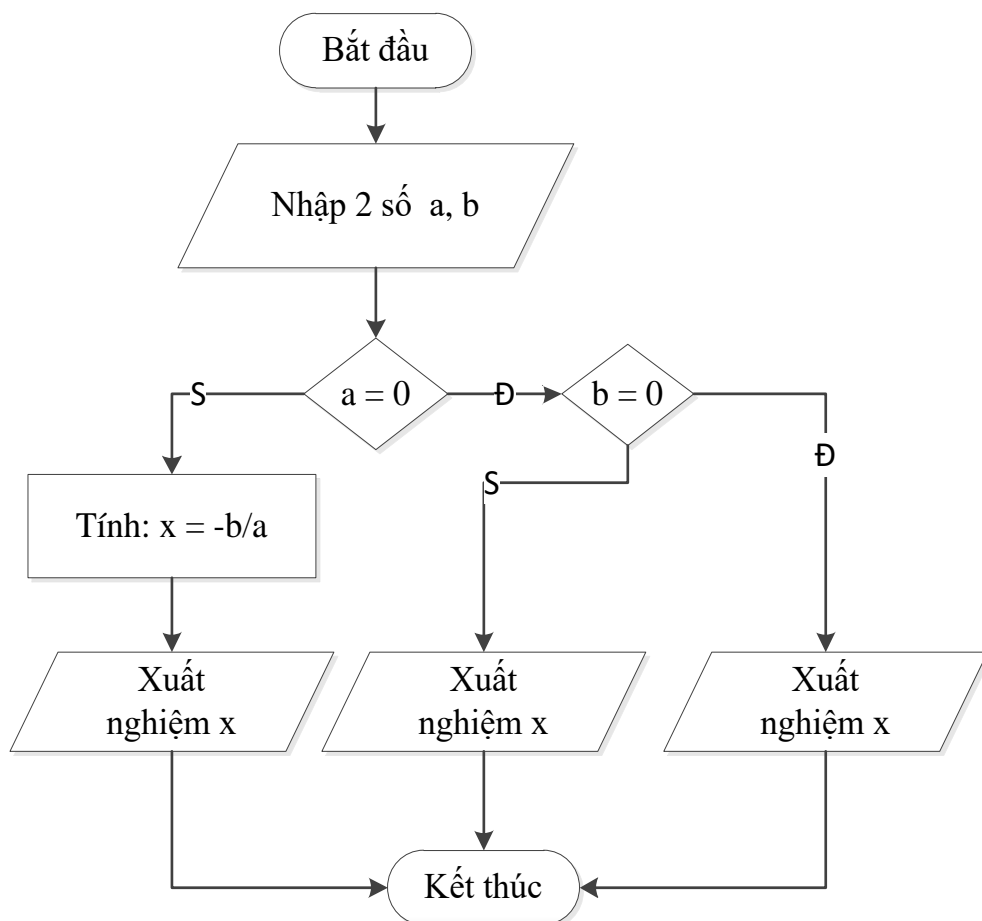
- ❖ Đầu vào: 2 số nguyên a, b
- ❖ Đầu ra: Tổng của 2 số nguyên a, b.
- ❖ Thuật toán:





**Ví dụ 1.2:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán giải phương trình bậc nhất  $ax + b = 0$  ( $a, b \in \mathbb{R}$ )

- ❖ Đầu vào: 2 số thực  $a, b$
- ❖ Đầu ra: Nghiệm của phương trình bậc nhất  $ax + b = 0$
- ❖ Thuật toán:



### 1.2.3.3. Sử dụng mã giả (pseudo-code)

Mã giả là một ngôn ngữ hình thức giúp các lập trình viên phát triển thuật toán. Mã giả cũng tương tự như ngôn ngữ giao tiếp hàng ngày, nó thuận lợi và thân thiện mặc dù nó không phải là ngôn ngữ lập trình máy tính.

Chương trình mã giả thì không thực thi được trên máy tính. Thay vào đó, chúng chỉ giúp bạn nghĩ ra một chương trình trước khi cố gắng viết nó bằng một ngôn ngữ lập trình như ngôn ngữ C.

**Ví dụ 1.3:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán tính tổng hai số nguyên a, b.

- ❖ Đầu vào: 2 số nguyên a, b
- ❖ Đầu ra: Tổng của 2 số nguyên a, b.
- ❖ Thuật toán:

---

```
Read a, b
Tong ← a + b
Write Tong
```

---

**Ví dụ 1.4:** Sử dụng ngôn ngữ tự nhiên để biểu diễn thuật toán cho bài toán giải phương trình bậc nhất  $ax + b = 0$  ( $a, b \in \mathbb{R}$ )

- ❖ Đầu vào: 2 số thực a, b
- ❖ Đầu ra: Nghiệm của phương trình bậc nhất  $ax + b = 0$
- ❖ Thuật toán:

---

```
Read a, b
If a = 0 then
    If b = 0 then
        Write "Phương trình vô số nghiệm!"
    Else
        Write "Phương trình vô nghiệm!"
Else
    X ← -b/a
    Write X
```

---

- **Ưu điểm:** tránh được hầu hết các vấn đề nhập nhằng trong diễn đạt.
- **Khuyết điểm:** Giống như hầu hết các ngôn ngữ lập trình là mơ hồ, không thống nhất về cú pháp.

### 1.3. Các bước xây dựng chương trình

Để xây dựng hoàn chỉnh một chương trình, ta thường trải qua 6 bước sau:

- Xác định bài toán/ vấn đề
- Lựa chọn phương pháp giải
- Xây dựng thuật toán/ thuật giải
- Cài đặt
- Kiểm thử
- Vận hành/ bảo trì.

## BÀI TẬP CHƯƠNG 1

### ❖ Bài tập lý thuyết:

Bài 1-1. Hãy trình bày khái niệm về lập trình?

Bài 1-2. Hãy trình bày khái niệm về thuật toán và các tính chất của thuật toán?

Bài 1-3. Hãy cho biết các phương pháp biểu diễn thuật toán và ưu, khuyết điểm của từng phương pháp?

### ❖ Bài tập áp dụng:

Hãy biểu diễn các thuật toán cho các bài toán sau đây bằng phương pháp ngôn ngữ tự nhiên và lưu đồ.

Bài 1-4. Nhập ba số bất kỳ. Hãy tính tổng ba số vừa nhập.

Bài 1-5. Nhập vào ba số bất kỳ. Hãy tính trung bình ba số vừa nhập.

Bài 1-6. Nhập vào ba số bất kỳ. Hãy tìm giá trị lớn nhất trong 3 số vừa nhập.

Bài 1-7. Nhập các hệ số  $a, b, c$  của phương trình bậc 2  $ax^2 + bx + c = 0$ . Hãy giải phương trình bậc 2 trên theo  $a, b, c$  vừa nhập.

## CHƯƠNG 2 – GIỚI THIỆU NGÔN NGỮ C++

### ❖ Mục tiêu:

- Ghi nhớ được cấu trúc một chương trình C++ đơn giản.
- Ghi nhớ được cú pháp của lệnh nhập/ xuất.
- Ghi nhớ được các kiểu dữ liệu được C++ cung cấp.
- Trình bày được các khái niệm về: biến, hằng và định danh.
- Giải thích được ý nghĩa của chuyển đổi kiểu dữ liệu.
- Ghi nhớ được các toán tử và chọn được các toán tử phù hợp cho từng bài toán cụ thể.

### 2.1. Cấu trúc một chương trình C++

Một chương trình C++ đơn giản bao gồm hai phần như sau:

---

```
//Khai báo thư viện  
#include<iostream>  
using namespace std;  
  
//Hàm chính  
void main()  
{  
    //Các câu lệnh;  
}
```

Header

main

---

Ở thời điểm này, chúng ta có thể xem một chương trình C++ gồm hai phần chính: **tiêu đề (header)** và **hàm chính (main)**. **Tiêu đề**, hay còn gọi là **vùng toàn cục (global section)**, đưa ra các hướng dẫn tiền xử lý cho trình biên dịch. Nó chứa các ghi chú để mô tả cho mục đích của chương trình, cũng như thông tin các thư viện sẽ được sử dụng trong chương trình.

Mỗi chương trình C++ có **duy nhất một** hàm **main** cho biết điểm bắt đầu thực thi của chương trình. Mỗi hàm **main** phải bắt đầu bằng một dấu ngoặc nhọn { và kết thúc bằng dấu ngoặc nhọn }.

#### Ví dụ 2.1:

```
//Đây là chương trình minh họa một chương trình C++ đơn giản
//Nhập tên bạn ở đây
//Khai báo thư viện
#include<iostream>
using namespace std;
const double PI = 3.14; //Hằng số
//Hàm chính
void main()
{
    float fBanKinh = 4.0; //Khai báo biến và khởi tạo giá trị
    cout<<"PI = "<<PI<<endl;
    cout<<"Ban Kinh = "<<fBanKinh<<endl;
    cout<<"Chu vi = "<<2*PI*fBanKinh<<endl;
}
```

---

## 2.2. Lệnh nhập/xuất

### 2.2.1. Lệnh xuất

Lệnh xuất cho phép chương trình xuất lên màn hình nội dung được người lập trình chỉ định.

Cú pháp:

```
cout<< Nội_dung_xuất;
```

**Nội\_dung\_xuất** bao gồm:

- **Văn bản thường:** nội dung văn bản được đặt trong cặp ngoặc kép.
- **Tên biến:** xuất lên màn hình giá trị được lưu trong **Tên biến**.
- **Các từ khóa định dạng:**

Để sử dụng các định dạng này cần khai báo thư viện `<iomanip>` ở đầu chương trình bằng chỉ thị `#include <iomanip>`.

- + `endl`: Kết thúc một dòng.
- + `setw(n)`: Bình thường các giá trị được in ra bởi lệnh `cout <<` sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này quy định độ rộng dành để in ra các giá trị là `n` cột

màn hình. Nếu  $n$  lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.

- + *setprecision(n)*: Chỉ định số chữ số in ra là  $n$ . *setprecision(n)* kết hợp với *fixed* sẽ xuất số chữ số thập phân sau dấu chấm. Số sẽ được làm tròn trước khi in ra.
- + *showpoint, noshowpoint*: cho phép hay không cho phép xuất lên màn hình các chữ số **0 sau dấu chấm thập phân**.

### Ví dụ 2.2:

---

```
//Chương trình minh họa định dạng xuất
//Thêm tên của bạn ở đây
//Khai báo thư viện
#include <iostream> // để sử dụng cout <<
#include <iomanip> // để sử dụng các định dạng
using namespace std;
void main()
{
    cout << "CHI TIÊU" << endl << "=====" << endl ;
    cout << setprecision(2)<<fixed;
    cout << "Sách vở" << setw(20) << 123.456 << endl;
    cout << "Thức ăn" << setw(20) << 2453.6 << endl;
    cout << "Quan ao lạnh" << setw(15) << 3200.0 << endl;
}
```

---

### 2.2.2. Lệnh nhập

Lệnh nhập cho phép chương trình nhận dữ liệu được nhập từ bàn phím.

**Cú pháp:**

**cin>> Tên\_biến;**

Có thể nhập nhiều giá trị đồng thời theo cú pháp:

**cin>> Tên\_biến1>> Tên\_biến2>>...;**

### Ví dụ 2.3:

```
//Chương trình minh họa định dạng xuất
/*Chương trình minh họa câu lệnh nhập*/
//Khai báo thư viện
#include<iostream>
using namespace std;

//Hàm chính
void main()
{
    cout << "\\t\\tCHƯƠNG TRÌNH MINH HOA CAU LENH NHAP\\n";
    int iA, iB;
    int iTong;

    cout << "Hay nhap so nguyen a: ";
    cin >> iA;
    cout << "Hay nhap so nguyen b: ";
    cin >> iB;

    iTong = iA + iB;
    cout << "Tong 2 so vua nhap la: " << iTong;
}
```

### 2.3. Các kiểu dữ liệu cơ sở

Trong C++ có bốn kiểu dữ liệu cơ bản:

Kiểu dữ liệu	Từ khóa	Kích thước (bytes)	Phạm vi giá trị
Số nguyên	short	2	-32768 → +32767
	int	4	-2.147.483.648 → +2.147.483.647
	long	4	-2.147.483.648 → +2.147.483.647
Số thực	float	4	$\pm 3.4E \pm 38$
	double	8	$\pm 1.7E \pm 308$
Ký tự	char	1	0 → 255
Luận lý (Boolean)	bool		true/ false



### 2.3.1. Biến, hằng

#### 2.3.2. Biến

Các câu lệnh và dữ liệu của chương trình được lưu trữ tạm thời vào các vùng nhớ. Phần lớn chương trình lưu dữ liệu xuống và lấy dữ liệu từ **các vùng nhớ** và do đó bắt buộc người lập trình phải cho máy tính biết **tên và kiểu dữ liệu** của mỗi vùng nhớ mà người đó có ý định sử dụng.

##### a. Khai báo biến:

- Cú pháp:

*<Kiểu dữ liệu> <Tên biến>;*

*Hoặc*

*<Kiểu dữ liệu> <Tên biến 1>, <Tên biến 2>, ...;*

- Ví dụ:

```
//Khai báo x là biến chưa giá trị số nguyên (integer)
```

```
int x;
```

```
//Khai báo hai biến số thực (float)
```

```
float fpBanKinh, fpDienTich;
```

```
//Khai báo a là biến chưa ký tự (character)
```

```
char a;
```

##### b. Khởi tạo giá trị cho biến

Các biến cũng có thể được khởi tạo một giá trị khi khai báo, nhưng giá trị đó không vĩnh viễn mà có thể thay đổi.

*Ví dụ:*

---

```
int Tong = 0; //Khai báo và khởi tạo
```

```
int Tich = 1; //Khai báo và khởi tạo
```

---

### 2.3.3. Hằng

**Khả năng thay đổi hoặc không cho thay đổi dữ liệu** được lưu trữ có thể là thuộc tính thứ ba của những vùng nhớ. **Các vùng nhớ** của bộ nhớ mà ở đó

giá trị dữ liệu được lưu trữ **có thể thay đổi** trong suốt quá trình thực thi chương trình được gọi là **các biến**. Các biến này thông thường không nên khai báo trong vùng **tiêu đề (global section)** của chương trình. **Các vùng nhớ** của bộ nhớ mà ở đó giá trị dữ liệu được lưu trữ được khởi tạo một lần và **không bao giờ thay đổi** trong suốt quá trình thực thi của chương trình được gọi là **các hằng số**. Chúng thường được định nghĩa trong vùng toàn cục và được đặt trước bởi từ khóa **const** hoặc thông qua cú pháp **#define**.

- Cú pháp:

```
#define <Tên hằng> <Giá trị>
const <Kiểu dữ liệu> <Tên hằng> = <Giá trị>;
```

- Ví dụ:

```
const float PI = 3.14;
```

Hoặc

```
#define PI 3.14
```

#### 2.3.4. Định danh (Identifier)

Định danh được sử dụng để **đặt tên cho biến, hằng số và nhiều thành phần khác của chương trình**. Chúng bao gồm chủ yếu là các chữ cái, chữ số và ký tự gạch dưới **\_**. Chúng **không được bắt đầu bằng chữ số, không được có khoảng trắng và không được trùng với các từ khóa** dành riêng của C++ như *int* hoặc *if*. Tất cả các ký tự trong C++ là phân biệt hoa thường; do đó các vùng nhớ có tên là simple, Simple và SIMPLE là ba vùng nhớ khác nhau.


**Ví dụ 2.4:** Viết chương trình cho phép người dùng nhập vào bán kính hình tròn. Tính diện tích hình tròn với bán kính vừa nhập và xuất lên màn hình kết quả.

---

```
//Đây là chương trình tính diện tích hình tròn
//Khai báo thư viện
#include<iostream>
#include<cmath>
using namespace std;
const double PI = 3.14;    //Hằng số
```

---

```
//Ham chinh
void main()
{
    //Khai bao bien
    float fBanKinh, fDienTich;
    //Nhap du lieu
    cout<<"Hay nhap ban kinh hinh tron: "<<endl;
    cin>>fBanKinh;
    //Tinh toan
    fDienTich = PI*pow(fBanKinh,2);
    //Xuat du lieu
    cout<<"PI = "<<PI<<endl;
    cout<<"Ban Kinh = "<<fBanKinh<<endl;
    cout<<"Dien tich = "<<fDienTich<<endl;
}
```



---

### 2.3.5. Chuyển đổi kiểu dữ liệu

Bất cứ khi nào một biến số nguyên và một biến số thực hoặc hằng số xuất hiện trong cùng một biểu thức, số nguyên được thay đổi tạm thời thành số thực tương ứng. Sự chuyển đổi tự động này được gọi là sự ép kiểu ẩn.

Xem xét tình huống sau:

```
int Count;
Count = 7.8; //Count = 7
```

Chúng ta đang cố đưa một số thực vào một biến số nguyên. Trong C++ phần thực sẽ bị cắt bỏ, do đó, chúng ta sẽ bị mất thông tin.

Chuyển đổi kiểu có thể được thực hiện một cách tường minh bằng cách sử dụng cú pháp tổng quát: <kiểu dữ liệu>(Giá trị). Đây được gọi là **ép kiểu** hoặc **chuyển đổi kiểu**.

Ví dụ:

---

```
int count;

float sum;
```

---

---

```
count = 10.89; //Số thực sang số nguyên phép chuyển kiểu
               // 10 sẽ được lưu vào biến count

count = int(10.89); //Cũng chuyển số thực sang số nguyên;
                  // tuy nhiên đây là phép ép kiểu
```

---

Nếu hai số nguyên chia nhau kết quả sẽ là một số nguyên đã bỏ đi phần thực. Điều này có thể tạo ra kết quả không mong muốn.

Ví dụ:

---

```
int iSoSVDau = 10;

int iTongSoSV=50;

float fPhanTram;

fPhanTram = iSoSVDau/iTongSoSV;
```

---

Trong bài này chúng ta **mong muốn fPhanTram sẽ là 0.2** vì 10/50 là 0.2. Tuy nhiên vì cả iSoSVDau và iTongSoSV là các số nguyên, kết quả của phép chia sẽ là số nguyên bỏ đi phần thực. **Trong trường hợp này sẽ là 0.** Bất cứ khi nào, một số nguyên có giá trị nhỏ chia cho một số nguyên có giá trị lớn hơn kết quả luôn luôn là 0. Chúng ta có thể sửa bài này bằng cách ép kiểu:

```
fPhanTram = float(iSoSVDau)/iTongSoSV;
```

## 2.4. Các toán tử

### 2.4.1. Các toán tử số học

Bảng 2.1. Bảng các toán tử số học

Ký hiệu	Phép toán	Ví dụ	Kết quả
+	Cộng (Addition)	25 + 3	28
-	Trừ (Subtraction)	25.0 - 0.3	24.7
*	Nhân (Multiplication)	25 * 3	75
/	Chia (Division)	5.0 / 2.0	2.5
%	Chia dư (Remainder)	25 % 3	1

**Ví dụ 2.5:**

---

```
//Chương trình hiển thị thời gian
//Khai báo thư viện
#include<iostream>
using namespace std;

//Hàm main
void main()
{
    //Khai báo biến
    int TongSoGiay;
    int SoPhut, SoGiay;
    //Nhập dữ liệu
    cout << "Hay nhap tong so giay: ";
    cin >> TongSoGiay;
    //Tính toán
    SoPhut = TongSoGiay/60;//Tính số phút dựa trên tổng số giây
    SoGiay = TongSoGiay % 60;//Số giây còn lại
    //Xuất kết quả
    cout << TongSoGiay << " giay la " << SoPhut << " phut va "
         << SoGiay << " giay";
}
```

---

#### 2.4.2. Toán tử gán

Ký hiệu = trong C++ được gọi là toán tử gán. Nó gán cho biến ở bên trái **một giá trị ở bên phải**. Mặc dù ký hiệu này nhìn giống ký hiệu so sánh bằng, **không được nhầm lẫn nó với phép so sánh bằng. Phía bên trái phải luôn là một biến**. Ví dụ, `count = 8` là một phát biểu hợp lệ trong C++, tuy nhiên `8=count` sẽ phát sinh một lỗi cú pháp.

Các câu lệnh gán sẽ đặt các giá trị vào các vùng nhớ. Phía bên trái của câu lệnh gán chứa một và duy nhất một biến. Phía bên phải chứa một biểu thức. Một biểu thức có thể là một thao tác bất kỳ của các số thông thường, hoặc nội dung của hằng số và/hoặc biến. C++ sử dụng dấu = để phân chia giữa phía bên trái và phía bên phải của câu lệnh gán.

*Ví dụ:*

---

```
int count;
int total;
```

---

---

```
total = 10; //10 được lưu vào vùng nhớ có tên là total
count = 3 + 4; //Phía bên phải được tính là 7
           // count được gán giá trị của 7
total = total + count; //Phía bên phải được tính là 10 + 7
           // và 17 được đưa vào vùng nhớ có tên là total
```

---

### 2.4.3. Các toán tử gán tắt

Bảng 2.2. Các toán tử gán tắt

Ký hiệu	Phép toán	Ví dụ	Kết quả
<code>+=</code>	Gán cộng (Addition assignment)	<code>i += 3</code>	<code>i = i + 3</code>
<code>-=</code>	Gán trừ (Subtraction assignment)	<code>i -= 3</code>	<code>i = i - 3</code>
<code>*=</code>	Gán nhân (Multiplication assignment)	<code>i *= 3</code>	<code>i = i * 3</code>
<code>/=</code>	Gán chia (Division assignment)	<code>i /= 3</code>	<code>i = i / 3</code>
<code>%=</code>	Gán chia dư (Remainder assignment)	<code>i %= 3</code>	<code>i = i % 3</code>

### 2.5. Một số hàm toán học

C++ cung cấp một số hàm toán học cơ bản. Để sử dụng các hàm toán học này, chúng ta phải khai báo thư viện **cmath**.

Bảng 2.3. Bảng các hàm toán học

Hàm	Mô tả	Ví dụ	Kết quả
<code>pow(x, y)</code>	Tính giá trị $x^y$	<code>N = pow(2, 3)</code>	<code>N = 8</code>
<code>sqrt(x)</code>	Tính căn bậc 2 của $x$	<code>N = sqrt(4)</code>	<code>N = 2</code>
<code>cbrt(x)</code>	Tính căn bậc 3 của $x$	<code>N = cbrt(8)</code>	<code>N = 2</code>
<code>abs(x)</code>	Trị tuyệt đối của số nguyên $x$	<code>N = abs(-3)</code>	<code>N = 3</code>
<code>fabs(x)</code>	Trị tuyệt đối của số thực $x$	<code>N = fabs(-5.6)</code>	<code>N = 5.6</code>
<code>ceil(x)</code>	Làm tròn lên của $x$	<code>N = ceil(-8.8)</code>	<code>N = -8.0</code>
<code>floor(x)</code>	Làm tròn xuống của $x$	<code>N = floor(-8.8)</code>	<code>N = -9.0</code>

## BÀI TẬP CHƯƠNG 2

### ❖ Bài tập lý thuyết:

Bài 2-1. Cấu trúc một chương trình C++ đơn giản gồm mấy thành phần? Đó là những thành phần nào?

Bài 2-2. Cho biết cú pháp của câu lệnh xuất và cho các ví dụ?

Bài 2-3. Cho biết cú pháp của câu lệnh nhập và cho ví dụ?

Bài 2-4. Khi đặt tên cho các biến cần lưu ý những gì?

Bài 2-5. Hãy khai báo 2 biến số nguyên, 1 biến số thực, 3 biến ký tự.

Bài 2-6. Cho biết cú pháp khai báo hằng và cho ví dụ minh họa.

Bài 2-7. Viết chương trình cho phép người dùng nhập vào 3 số nguyên. Tính tổng của 2 số vừa nhập.

Bài 2-8. Viết chương trình cho phép người dùng nhập vào 3 số nguyên. Tính giá trị trung bình của 3 số vừa nhập.

Bài 2-9. Viết chương trình cho phép người dùng nhập vào 1 ký tự. Hãy cho biết ký tự vừa nhập là chữ hoa hay chữ thường?

Bài 2-10. Viết chương trình cho phép người dùng nhập vào bán kính của hình tròn. Tính chu vi và diện tích của hình tròn với bán kính vừa nhập.

## CHƯƠNG 3 - CẤU TRÚC RẼ NHÁNH

### ❖ Mục tiêu:

- Ghi nhớ cú pháp câu lệnh if và switch.
- Diễn tả được cách thức hoạt động của câu lệnh if và switch.
- Áp dụng được câu lệnh if và switch cho các bài toán cụ thể.

### 3.1. Các phép toán quan hệ

Bảng 3.1. Bảng các phép toán quan hệ

Ký hiệu toán học	Ý nghĩa	Ký hiệu C++	Ví dụ
=	So sánh bằng	==	3 == 4: Sai
≠	So sánh khác	!=	3 != 4: Đúng
>	So sánh lớn hơn	>	3 > 4: Sai
≥	So sánh lớn hơn hay bằng	>=	3 >= 4: Sai 4 >= 4: Đúng
<	So sánh nhỏ hơn	<	3 < 4: Đúng
≤	So sánh nhỏ hơn hay bằng	<=	3 <= 4: Đúng 4 <= 4: Đúng

### 3.2. Các phép toán logic

Bảng 3.2. Bảng các phép toán logic

Ký hiệu	Phép toán	Ví dụ
!	<b>NOT</b>	!b
&&	<b>AND</b>	(a > 1) && (a < 10)
	<b>OR</b>	a    b

#### 3.2.2. Phép toán AND (&&)

Phép AND chỉ đúng khi tất cả các mệnh đề đều đúng.



Bảng 3.3. Bảng chân trị phép AND

A	B	A&&B
true	true	true
true	false	false
false	true	false
false	false	false

### 3.2.3. Phép toán OR (||)

Phép OR chỉ sai khi tất cả các mệnh đề đều sai.

Bảng 3.4. Bảng chân trị phép OR

A	B	A  B
true	true	true
true	false	true
false	true	True
false	false	false

### 3.2.4. Phép toán NOT (!)

Phép NOT được sử dụng để phủ định một mệnh đề.

Bảng 3.5. Bảng chân trị phép toán NOT

A	!A
False	true
True	false

## 3.3. Câu lệnh if

Đôi lúc chúng ta chỉ muốn thực thi một đoạn lệnh với một điều kiện cụ thể. Để thực hiện việc này, chúng ta sử dụng **các câu lệnh điều kiện**. Một câu lệnh *if* là một loại câu lệnh điều kiện.

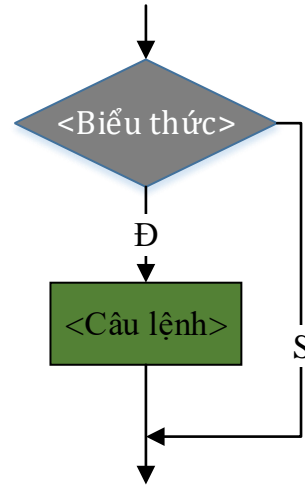
Nếu bạn muốn thực hiện một cách có điều kiện một số câu lệnh bằng câu lệnh *if*, cú pháp sau đây là bắt buộc:

### 3.3.1. Dạng 1: **if**

**Cú pháp:**

```
if (<Biểu thức>
    <Câu Lệnh>;
Next_Statement;
```

**Lưu đồ:**

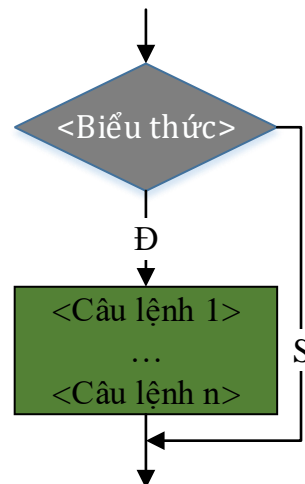


**Hoặc:**

**Cú pháp:**

```
if (<Biểu thức>)
{
    <Câu Lệnh 1>;
    ...
    <Câu Lệnh n>;
}
Next_Statement;
```

**Lưu đồ:**



<Biểu thức> cho kết quả đúng (true) hoặc sai (false). Nếu <biểu thức> là đúng (true), <câu lệnh> được thực thi, sau đó chương trình tiếp tục thực thi câu lệnh Next\_Statement. Nếu <biểu thức> là sai (false), <câu lệnh> được bỏ qua và tiếp tục thực thi ngay lập tức Next\_Statement.

**Ví dụ 3.1:**

```
//Day la chuong trinh xuat len "Ban dau" neu diem trung binh
//cua sinh vien lon hon hay bang 5.0 va xuat len "Ban rot"
//trong cac truong hop nguoc lai
//Khai bao thu vien
#include<iostream>
using namespace std;
//Ham chinh
void main()
{
    //Khai bao bien
    float average;
    //Nhap du lieu
    cout<<"Hay nhap diem trung binh cua ban: "<<endl;
    cin>>average;
    if(average>=5) //Chu y su dung toan tu quan he
        cout<<"Ban dau"<<endl;
    if(average<5)
        cout<<"Ban rot"<<endl;
}
```

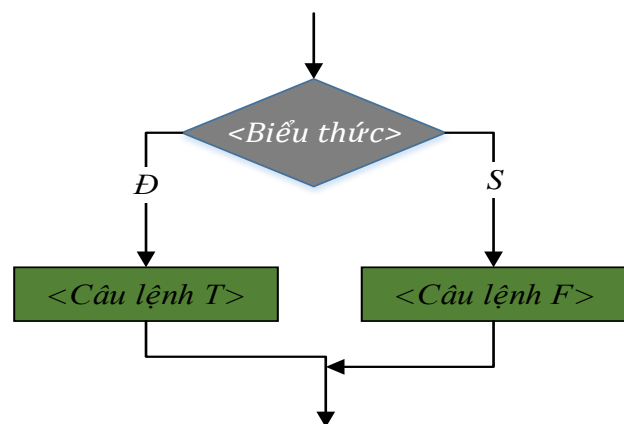
Chú ý không thể có trường hợp chương trình xuất lên màn hình cả hai thông báo “Ban dau” và “Ban rot”. Chỉ có một câu lệnh *if* được thực thi.

### 3.3.2. Dạng 2: **if ... else**

**Cú pháp:**

```
if (<Biểu thức>)
    <Câu Lệnh T>;
else
    <Câu Lệnh F>;
Next_Statement;
```

**Lưu đồ:**

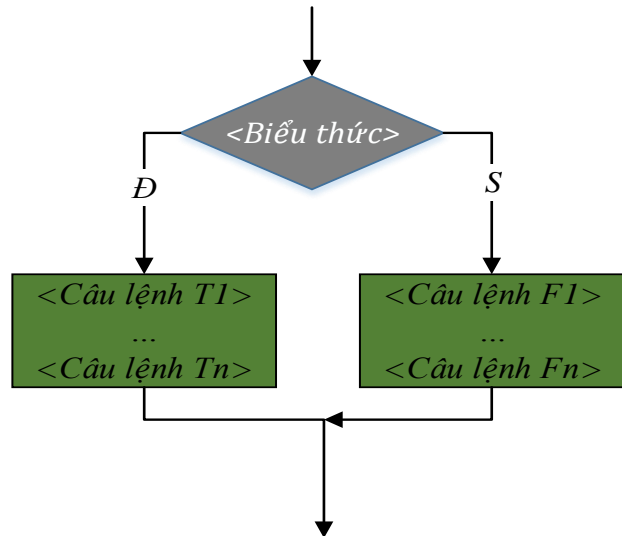


**Hoặc:**

**Cú pháp:**

```
if (<Biểu thức>)
{
    <Câu Lệnh T1>;
    ...
    <Câu Lệnh Tn>;
}
else
{
    <Câu Lệnh F1>;
    ...
    <Câu Lệnh Fn>;
}
Next_Statement;
```

**Lưu đồ:**



Nếu <biểu thức> là đúng (true), <câu lệnh T> được thực thi và chương trình thực thi tiếp Next\_Statement.

Nếu <biểu thức> là sai (false), <câu lệnh F> theo ngay sau từ khóa else được thực thi, và chương trình tiếp tục thực thi với Next\_Statement.

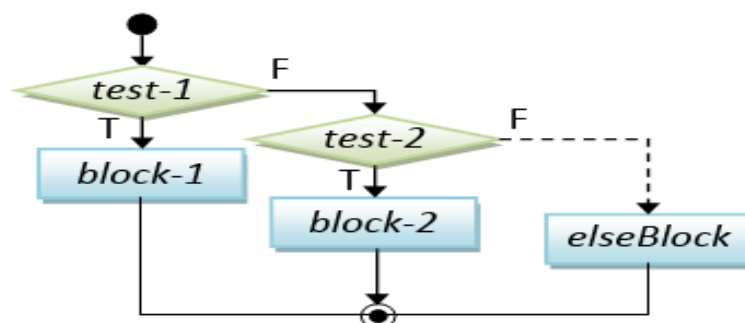
**Ví dụ 3.2:** Giải phương trình bậc nhất:  $ax + b = 0$

```
#include <iostream>
using namespace std;
void main()
{
    int a, b;
    cout << "Nhap hai so a, b: ";
    cin >> a >> b;
    if (a == 0)
    {
        if (b == 0)
            cout<<"Phuong trinh VSN";
        else
            cout<<"Phuong trinh VN";
    }
    else
        cout<<"x = ", float(-b) / a;
}
```

### 3.3.3. Nhiều cấu trúc **if ... else** lồng nhau

```
if ( booleanExpr-1 )
{
    block-1 ;
}
else
{
    if ( booleanExpr-2 )
    {
        block-2 ;
    }
    else
    {
        if ( booleanExpr-3 )
        {
            .....
        }
        else
        {
            elseBlock ;
        }
    }
}
```

*Lưu đồ:*



**Ví dụ 3.3:**

```
if (mark >= 80)
{
    cout<<"A";
}
else
{
    if (mark >= 70)
    {
        cout << "B";
    }
    else
    {
        if (mark >= 60)
        {
            cout << "C";
        }
        else
        {
            if (mark >= 50)
            {
                cout << "D";
            }
            else
            {
                cout << "F";
            }
        }
    }
}
```

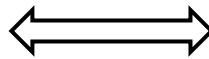
**Chú ý:**

– Mện

h đề *else* gắn  
với mệnh đề  
*if* gần nhất  
trong cùng  
một khối.

```
int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        cout<<"A";
else
    cout<<"B";
```

```
int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        cout<<"A";
else
    cout<<"B";
```



Nên dùng

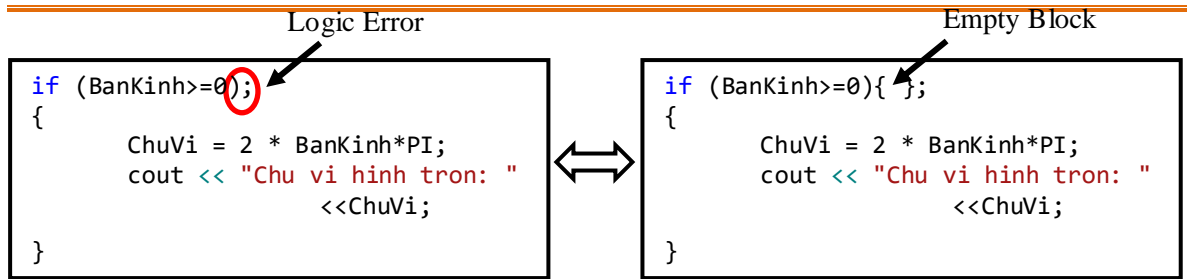
– Đoạn lệnh trên sẽ không in ra gì cả. Để bắt mệnh đề *else* gắn với mệnh đề *if* đầu tiên, phải thêm một cặp ngoặc {}:

```
int i = 1, j = 2, k = 3;
if (i > j)
{
    if (i > k)
        cout << "A";
}
else
    cout<<"B";
```

Đoạn lệnh trên sẽ in ra ký tự **B**.

**Lỗi phổ biến:**

Thêm một dấu chấm phẩy ở cuối mệnh đề *if*.



*Lỗi này rất khó tìm, vì nó không phải là lỗi biên dịch hay lỗi chạy chương trình, nó là một lỗi logic.*

```
if (BanKinh>=0)

    ChuVi = 2 * BanKinh*PI;
    cout << "Chu vi hình tron: "
        <<ChuVi;
```

– Quên cặp dấu {} cho khối lệnh.

**Sai**

```
if (BanKinh>=0)
{
    ChuVi = 2 * BanKinh*PI;
    cout << "Chu vi hình tron: "
        <<ChuVi;
}
```

**Đúng**

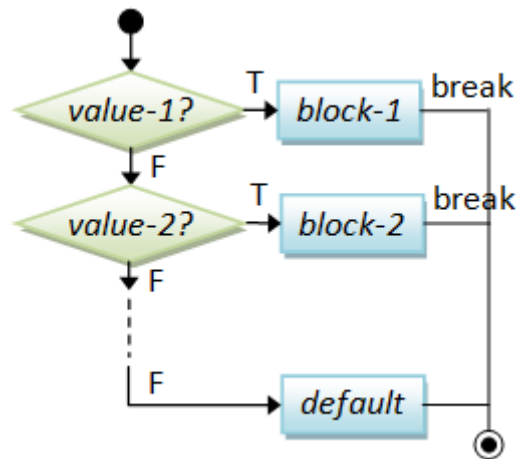
### 3.4. Câu lệnh **switch**

*Cú pháp:*

```
switch ( selector ) {
    case value-1:    block-1;    break;
    case value-2:    block-2;    break;
    case value-3:    block-3;    break;
    .....
    case value-n:    block-n;    break;
    default:         default-block;
}
```

*Lưu đồ:*





Hình 3.1 Lưu đồ cấu trúc *switch ... case*

**Ví dụ 3.4:**

```

int thang = m, nam = y; //Cho truoac gia tri x, y nguyen
switch (thang) {
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12: cout<<"Thang co 31 ngay!";    break;
case 4: case 6: case 9:
case 11: cout<<"Thang co 30 ngay!";    break;
case 2:
    if ((nam % 100 != 0 && nam % 4 == 0) || (nam % 400 == 0))
        cout<<"Thang co 29 ngay!";
    else
        cout<<"Thang co 28 ngay!";
    break;
default:
    cout<<"Gia tri thang x khong hop le!";
}
  
```

**Quy tắc lệnh *switch*:**

- Biểu thức **switch** phải sinh ra một giá trị kiểu **char**, **short**, hoặc **int**,... và phải luôn được bao trong cặp dấu ngoặc tròn.
- **gtri1**,...,**gtriN** phải có cùng kiểu dữ liệu với giá trị của biểu thức **switch**.
- Từ khóa **break** là tùy chọn, nhưng nên được sử dụng cuối mỗi trường hợp để thoát khỏi phần còn lại của lệnh **switch**. Nếu không có lệnh **break**, lệnh **case** tiếp theo sẽ được thực hiện.
- Trường hợp **default** là tùy chọn, có thể sử dụng để thực hiện các lệnh khi không có trường hợp nào ở trên là đúng.
- Thứ tự của các trường hợp (gồm cả trường hợp **default**) là không quan trọng. Tuy nhiên, phong cách lập trình tốt là nên theo một trình tự logic của các trường hợp và đặt trường hợp **default** cuối cùng.

**Lưu ý:**

- Đừng quên dùng lệnh **break** khi cần thiết.
- Ví dụ đoạn mã sau luôn hiển thị **"Sai so nam!"** dù giá trị **sonam** là bao nhiêu. Giả sử **sonam** bằng **15**. Lệnh **laisuatnam = 8.5** được thực hiện, tiếp theo là lệnh **laisuatnam = 9.0**, và cuối cùng là lệnh **cout<<"Sai so nam!"**.

```
switch (sonam)
{
    case 5:  laisuatnam = 6.5;
    case 10: laisuatnam = 7.5;
    case 15: laisuatnam = 8.5;
    case 30: laisuatnam = 9.0;
    default: cout << "Sai so nam!";
}
```

### 3.5. Toán tử điều kiện

**Cú pháp:**

```
(bt_logic) ? bt1 : bt2;
```

Thực hiện **bt1** khi **bt\_logic** đúng, thực hiện **bt2** khi **bt\_logic** sai.

**Ví dụ 1:** `abs = (a > 0) ? a : -a;`

**Ví dụ 2:** Thay vì sử dụng *if*

```
if (x > 0)
```

```
    y = 1;
```

```
else
```

```
    y = -1;
```

Có thể dùng toán tử ?

```
y = (x > 0) ? 1 : -1;
```

**Ví dụ 3:**

```
if (so % 2 == 0)
```

```
    cout<<so << " la so chan!";
```

```
else
```

```
    cout<<so << " la so le!";
```

**Tương đương với:**

```
(so % 2 == 0) ?cout<<so<<" la so chan!":cout<<so<< " la so le!";
```

## BÀI TẬP CHƯƠNG 3

### ❖ Bài tập lý thuyết:

Bài 3-1. Hãy cho biết cú pháp của câu lệnh if và cách thức hoạt động của câu lệnh này.

Bài 3-2. Hãy cho biết cú pháp của câu lệnh switch và cách thức hoạt động của câu lệnh này.

### ❖ Bài tập thực hành:

Bài 3-3. Nhập 3 cạnh *a*, *b*, *c*. Kiểm tra xem có lập thành tam giác không, nếu có thì là *tam giác gì* và tính *chu vi*, *diện tích* tam giác đó?

Bài 3-4. Viết chương trình nhập vào tháng và năm (dương lịch). Xuất ra màn hình tháng vừa nhập có bao nhiêu ngày? Biết rằng:

*Tháng 1/3/5/7/8/10/12 có 31 ngày. Tháng 4/6/9/11 có 30 ngày. Tháng 2 có 28 ngày nếu năm không nhuận, 29 ngày nếu năm nhuận.*

*Điều kiện năm nhuận:  $((\text{năm} \% 100 \neq 0 \ \&\& \ \text{năm} \% 4 == 0) \ || \ \text{năm} \% 400 == 0)$*

Bài 3-5. Viết chương trình nhập vào 2 số nguyên *chiSoCu*, *chiSoMoi* ứng với chỉ số điện cũ và chỉ số điện mới.

Tính *soKwTieuThu* = *chiSoMoi* - *chiSoCu*, và xuất ra màn hình tiền điện phải trả, biết rằng:

Số kw tiêu thụ	Đơn giá (đồng)
Dưới 50 kw	1.750
Từ 50 kw đến dưới 100 kw	2.225
Từ 100 kw đến dưới 200 kw	2.750
Từ 200 kw trở lên	3.250

Bài 3-6. Viết chương trình nhập vào 3 điểm  $A(x_1, y_1)$ ,  $B(x_2, y_2)$ ,  $C(x_3, y_3)$ . Kiểm tra xem 3 điểm A, B, C có tạo thành tam giác hay không? Nếu có, thì là tam giác gì? Tính và xuất diện tích tam giác ứng với 3 đỉnh vừa nhập ra màn hình.

**Hướng dẫn:**

- Ba điểm tạo thành tam giác khi 3 điểm đó không thẳng hàng.
- Đường thẳng qua 2 điểm A,B:  $y = m*x + b$ , với  $m = \frac{y_B - y_A}{x_B - x_A}$ ;  $b = y_A - m*x_A$ .
- Điểm C không thuộc về AB  $\Rightarrow$  A, B, C không thẳng hàng.

$$dienTich = \sqrt{p(p - s_{12})(p - s_{13})(p - s_{23})}; p = \frac{(s_{12} + s_{13} + s_{23})}{2}$$

$$\text{với } s_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

## CHƯƠNG 4 – CẤU TRÚC LẶP

### ❖ Mục tiêu:

- Phân biệt được sự khác nhau giữa tiền tố và hậu tố của toán tử tăng/ giảm.
- Ghi nhớ được cú pháp của các lệnh while, do..while và for.
- Diễn tả được cách thức hoạt động của các lệnh while, do..while và for.
- Phân biệt được sự khác nhau giữa tiền tố và hậu tố của toán tử tăng/ giảm.
- Áp dụng được các lệnh while, do..while và for để cài đặt các bài toán thực tế.

### 4.1. Toán tử tăng, giảm

Bạn có thể cộng hoặc trừ 1 từ một giá trị số nguyên cho trước:

```
count = count + 1;  
count += 1;
```

Cả hai câu lệnh này đều **tăng** giá trị của **count** lên 1 đơn vị. Nếu bạn thay dấu “+” bằng dấu “-” trong đoạn lệnh trên, thì cả hai câu lệnh đều **giảm** giá trị của **count** đi 1 đơn vị. C++ cũng cung cấp một **toán tử tăng ++** và một **toán tử giảm --** để thực hiện các nhiệm vụ này. Có hai chế độ có thể được sử dụng:

```
count++; //toán tử tăng ở chế độ hậu tố  
count--; //toán tử giảm ở chế độ hậu tố  
  
++count; //toán tử tăng ở chế độ tiền tố  
--count; //toán tử giảm ở chế độ tiền tố
```

Cả hai câu lệnh tăng đều thực thi chính xác như nhau. Cũng như vậy với các toán tử giảm. Vậy, mục đích của chế độ tiền tố và hậu tố là gì? Để trả lời câu hỏi này, hãy xem xét ví dụ sau:

```
int age = 18;  
if(age++ > 18)  
    cout<<"Chúc mừng bạn đã là người trưởng thành "<<endl;
```

Trong đoạn code trên, câu lệnh `cout` sẽ không được thực thi. Lý do là trong chế độ hậu tố sự so sánh giữa `age` và `18` được thực hiện trước. Sau đó giá trị của `age` được tăng lên một đơn vị. Vì `18` không lớn hơn `18`, điều kiện của `if` là sai (`false`). Mọi thứ sẽ khác nhau rất nhiều nếu bạn thay toán tử hậu tố bằng toán tử tiền tố.

```
int age = 18;
if(++age > 18)
    cout<<"Chúc mừng bạn đã là người trưởng thành "<<endl;
```

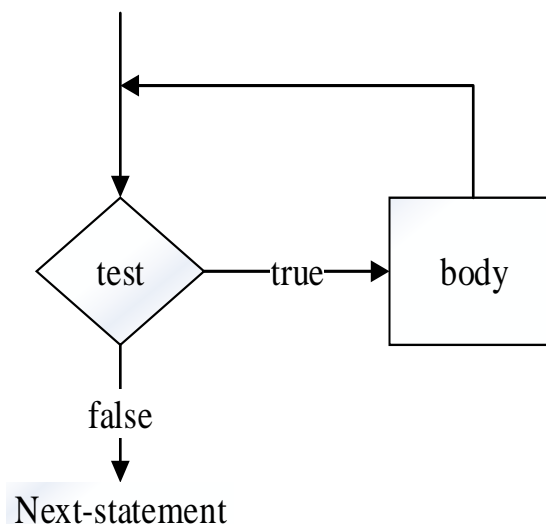
Trong đoạn code này `age` được tăng trước. Vì vậy, giá trị của nó là `19` khi phép so sánh được thực hiện. Câu lệnh điều kiện là đúng (`true`) và câu lệnh `cout` được thực thi.

## 4.2. Câu lệnh `while`

*Cú pháp:*

```
while (test)
{
    body;
}
next-statement;
```

*Lưu đồ:*



**Ví dụ:**

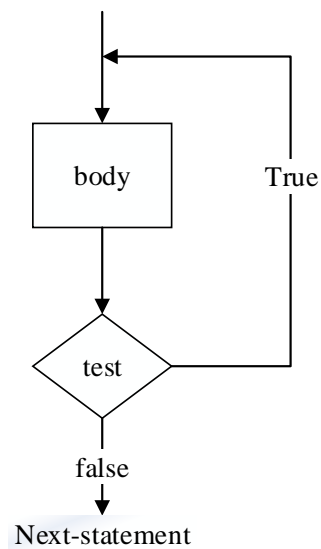
```
int count = 0;
while (count < 100)
{
    cout<<"Welcome to C++!";
    count++;
}
```

### 4.3. Câu lệnh do...while

**Cú pháp:**

```
do
{
    loop-body;
} while (test);
next-statement;
```

**Lưu đồ:**



**Ví dụ:**

```
int count = 0;
do
{
    cout<<"Welcome to C++!";
```



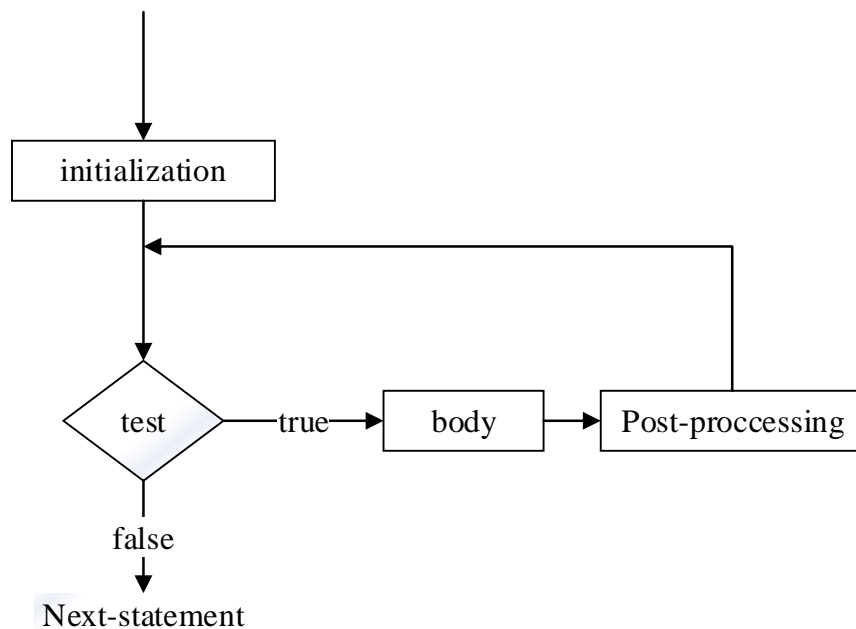
```
count++;  
} while (count < 100);
```

#### 4.4. Câu lệnh for

*Cú pháp:*

```
for (initialization; test; post-processing)  
{  
    loop-body;  
}  
next-statement;
```

*Lưu đồ:*



*Ví dụ:*

```
int i;  
for (i = 0; i < 100; i++)  
{  
    cout<<"Welcome to C++!";  
}
```

**Lưu ý:**

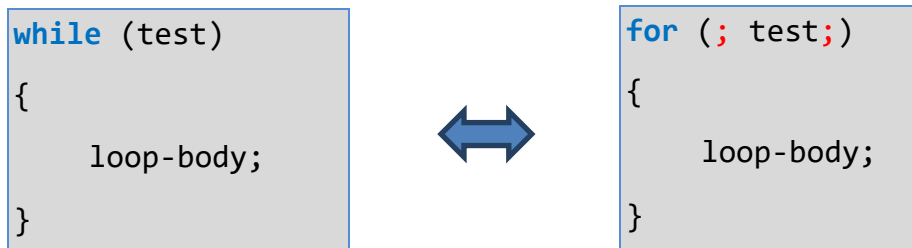
- Các trường hợp sau đây là đúng:

```
for (int i = 1; i < 1000; cout<<i++);
```

```
for (int i = 0, j = 0; (i + j < 10); i++, j++)  
{  
    //Do something  
}
```

#### 4.5. Cách sử dụng lệnh lặp

- Ba lệnh lặp **while**, **do...while**, và **for** là tương đương nhau trong nhiều trường hợp; nghĩa là có thể viết một vòng lặp bằng một dạng bất kỳ trong 3 dạng trên.
- Lệnh lặp **for** có thể sử dụng khi biết trước số lần lặp, ví dụ khi bạn muốn in ra một thông báo 100 lần.
- Lệnh lặp **while** có thể sử dụng khi không biết trước số lần lặp, ví dụ như trong trường hợp đọc vào các số đến khi gặp số 0.
- Lệnh lặp **do...while** có thể sử dụng thay lệnh **while** khi thân vòng lặp phải được thực hiện trước khi kiểm tra điều kiện tiếp tục lặp.
- Sự tương đương của các lệnh lặp:



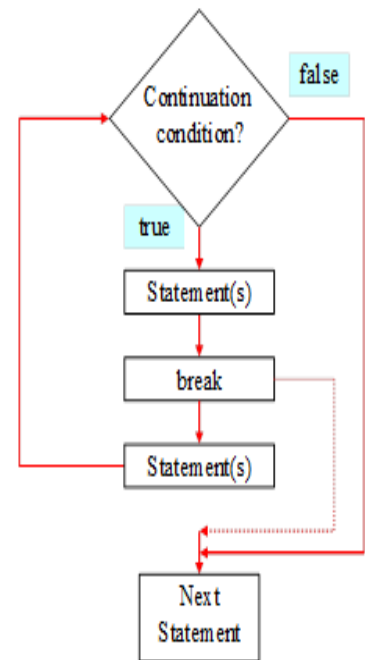
##### 4.5.1. Từ khóa break

- Chúng ta đã sử dụng từ khóa **break** trong câu lệnh **switch**. Chúng ta cũng có thể sử dụng từ khóa **break** trong lệnh lặp để kết thúc ngay vòng lặp.

**Ví dụ:**

```
#include<iostream>
using namespace std;

void main()
{
    int sum = 0;
    int number = 0;
    while (number<20)
    {
        number++;
        sum += number;
        if (sum>100)
            break;
    }
    cout << "The number is: " << number<<endl;
    cout << "The sum is: " << sum << endl;
    system("pause");
}
```



#### 4.5.2. Từ khóa continue

– Chúng ta cũng có thể dùng từ khóa *continue* trong vòng lặp. Khi gặp từ khóa *continue*, nó sẽ kết thúc vòng lặp hiện tại. Nói cách khác, *continue* thoát khỏi một vòng lặp, còn *break* thì kết thúc một lệnh lặp.

**Ví dụ:**

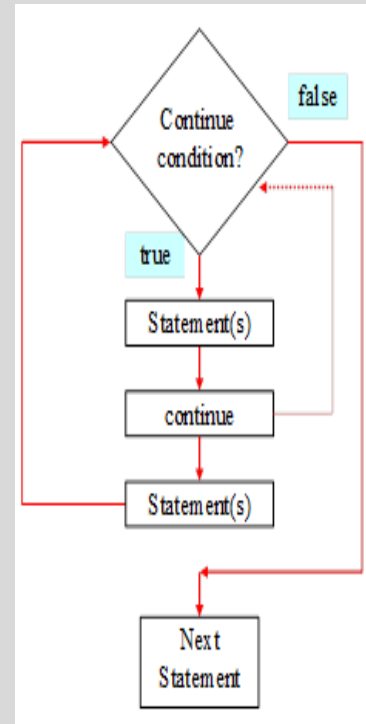
```
#include<iostream>
using namespace std;

void main()
```

```

{
    int sum = 0;
    int number = 0;
    //Tính tổng từ 1 đến 10 là 55
    while (number < 10)
    {
        number++;
        if (number == 5 || number == 6)
            //bỏ qua 5 và 6 nên tổng là 44
            continue;
        sum += number;
    }
    cout << "The number is: " << number << endl;
    cout << "The sum is: " << sum << endl;
    system("pause");
}

```



## BÀI TẬP CHƯƠNG 4

### ❖ Bài tập lý thuyết:

Bài 4-1. Hãy cho biết cú pháp và cách thức hoạt động của câu lệnh while?

Bài 4-2. Hãy cho biết cú pháp và cách thức hoạt động của câu lệnh do..while?

Bài 4-3. Hãy cho biết cú pháp và cách thức hoạt động của câu lệnh for?

### ❖ Bài tập thực hành:

Bài 4-4. Viết chương trình cho phép người dùng nhập vào số nguyên dương N. Hãy tính tổng các giá trị từ 1 tới N theo công thức:

$$s = 1 + 2 + \dots + N$$

Bài 4-5. Viết chương trình cho phép người dùng nhập vào một số nguyên dương N. Hãy tính tổng theo công thức:

$$s = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N}$$

Bài 4-6. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số nguyên tố hay không?

**Biết: số nguyên tố là số lớn hơn 1 chỉ chia hết cho 1 và cho chính nó.**

Bài 4-7. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số chính phương hay không?

**Biết: Số chính phương** hay còn gọi là **số hình vuông** là **số tự nhiên** có căn bậc hai là một **số tự nhiên**, hay nói cách khác, **số chính phương** bằng bình **phương** (lũy thừa bậc 2) của một **số tự nhiên**

Bài 4-8. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số hoàn hảo (hoàn thiện) hay không?

**Biết: Số hoàn thiện** (hay còn gọi là **số hoàn chỉnh**, **số hoàn hảo** hoặc **số hoàn thành**) là một số nguyên dương mà tổng các ước nguyên dương chính thức của nó (số nguyên dương bị nó chia hết ngoại trừ nó) bằng chính nó. Ví dụ: 6, 28, ...

Bài 4-9. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số nguyên tố?

Bài 4-10. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số chính phương?

Bài 4-11. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số hoàn hảo?

Bài 4-12. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số nguyên tố thuộc đoạn từ 1 đến  $N$ ?

Bài 4-13. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số chính phương thuộc đoạn từ 1 đến  $N$ ?

Bài 4-14. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số hoàn hảo thuộc đoạn từ 1 đến  $N$ ?

## CHƯƠNG 5 – HÀM (FUNCTION)

### ❖ Mục tiêu:

- Ghi nhớ cú pháp của hàm.
- Phân biệt được các cách truyền tham số cho hàm.
- Viết được một số hàm đệ quy đơn giản.
- Áp dụng được hàm vào các bài toán thực tế.

### 5.1. Định nghĩa và lời gọi hàm

Một yếu tố quan trọng của các chương trình có cấu trúc đó là tính mô đun hóa: chia mã lệnh thành các đơn vị nhỏ hơn. Các đơn vị này, hoặc các mô đun không trả về giá trị được gọi là các thủ tục trong hầu hết các ngôn ngữ lập trình và được gọi là hàm void trong C++.

Hàm *void main()* trong các chương trình thực hành trước là một hàm chỉ có một mô đun mã lệnh. Chúng ta cũng đã sử dụng các hàm được định nghĩa trước như *pow* và *sqrt* được định nghĩa trong các thư viện tiện ích và “được nhập khẩu” và chương trình của chúng ta với khai báo *#include<cmath>*.

Trong nhiều chương trình đơn giản hầu hết các chức năng được gọi bởi hàm chính. Việc gọi hàm về cơ bản nghĩa là bắt đầu thực thi các hướng dẫn trong mô đun đó. Đôi khi một chức năng có thể cần thông tin “được truyền qua” để thực hiện nhiệm vụ được chỉ định.

Nếu một hàm tính căn bậc hai của một số nguyên, thì nó cần số đó truyền cho hàm bởi chức năng gọi hàm. Thông tin được truyền qua một hàm được gọi là các tham số. Các tham số là các thành phần truyền thông giữa các hàm. Một số hàm thực hiện nhiệm vụ rất đơn giản như xuất lên màn hình nội dung cơ bản. Đây có thể chỉ là hướng dẫn cho người dùng hoặc chỉ là tài liệu hướng dẫn về những gì chương trình sẽ thực hiện. Các hàm này thường được gọi là hàm không tham số vì chúng không yêu cầu bất cứ thứ gì được truyền qua khi gọi hàm.

**Ví dụ 5.1:**

---

```
// Chương trình minh họa hàm
// Khai báo thu viện
#include<iostream>
using namespace std;
// Khai báo hàm
void XuatMoTa (); //Nguyen mau ham

//Ham chinh
void main()
{
    cout<<"Chao mung den voi chuong trinh tinh luong." <<endl;
    XuatMoTa(); //Loi goi ham
    cout<<"Hi vong ban thich chuong trinh nay."<<endl;

}
//Dinh nghĩa hàm
//*****
// Ten ham: XuatMoTa()
// Tac vu: Ham nay xuat len man hinh mo ta ve chuong trinh
// Dau vao: Khong
// Dau ra: Khong
//*****
void XuatMoTa() //Tieu de cua ham
{
    cout<<"*****"<<endl;
    cout<<"Chương trình nay nhan hai so (ty le chi tra va
        gio)"<<endl;
    cout<<"va xuat ra luong phai tra."<<endl;
    cout<<"*****"<<endl;
}
```

---

Hãy chú ý đến các phần được tô đậm trong chương trình trên. Phần tô đậm cuối cùng được gọi là phần định nghĩa hàm. Tiêu đề của hàm *void*



*XuatMoTa()* chứa tên của hàm được đứng trước bởi từ *void*. Từ *void* có nghĩa rằng hàm không có giá trị trả về cho các mô đun gọi nó. Sau tên hàm là cặp dấu ngoặc. Cũng giống như hàm chính, tất cả các hàm bắt đầu với dấu ngoặc nhọn trái và kết thúc với dấu ngoặc nhọn phải. Ở giữa các dấu ngoặc nhọn là các hướng dẫn của hàm.

Chú ý rằng hàm này nằm phía dưới hàm *main*. Hàm này được kích hoạt như thế nào? Nó phải được gọi bởi hàm *main* hoặc hàm khác trong chương trình. Hàm này được gọi bởi hàm *main* với một hướng dẫn đơn giản *XuatMoTa()*.

Chú ý lời gọi hàm chỉ có tên hàm (không có từ *void* phía trước) được theo sau bởi cặp dấu ngoặc và một dấu chấm phẩy.

## 5.2. Khai báo hàm

Cú pháp:

---

```
<KDL> <Tên hàm>([Danh sách tham số])
{
    <Các câu lệnh>;
    [return <Giá trị>/Biến/<Biểu thức>;]
}
```

---

## 5.3. Tầm vực của biến

### 5.3.1. Biến toàn cục

Phạm vi của một đối tượng (biến, hằng số, hàm, ...) là nơi mà nó có thể được truy xuất trong chương trình. Phần tiêu đề (phần phía trên hàm *main*) thường được đề cập đến như là phần toàn cục. Mọi đối tượng được định nghĩa hay khai báo trong phần này được gọi là **phạm vi toàn cục**, nghĩa là nó có thể được truy xuất tại bất kỳ thời điểm nào trong suốt quá trình thực thi chương trình. Mọi đối tượng được định nghĩa bên ngoài khối bao của tất cả các hàm có

phạm vi toàn cục. Mặc dù hầu hết các hằng số và tất cả các hàm được định nghĩa toàn cục, các biến không bao giờ định nghĩa như vậy.

### 5.3.2. Biến cục bộ

**Phạm vi cục bộ** đề cập đến các đối tượng được định nghĩa bên trong một khối lệnh. Chúng chỉ hoạt động bên trong vùng bao của một khối cụ thể. Trong C++ khối lệnh được bắt đầu bằng dấu { và kết thúc bằng dấu }. Vì tất cả các hàm (bao gồm cả hàm main) bắt đầu và kết thúc với một cặp ngoặc nhọn, thân của một hàm là một khối lệnh. Các biến được định nghĩa bên trong một hàm được gọi là **các biến cục bộ** (đối lập với **các biến toàn cục** trong phạm vi toàn cục).

#### Ví dụ 5.2:

---

```
//Chương trình minh họa phạm vi của biến
//Khai báo thu viên
#include<iostream>
using namespace std;
//Khai báo biến toàn cục
const float PI = 3.14;
//Khai báo nguyên mẫu hàm
void XuatTieuDe();

//Hàm chính
void main()
{
    float fpCircle;
    cout<<"fpCircle có phạm vi cục bộ trong hàm main." <<endl;
    {
        float fpSquare;
        cout<<"fpSquare có phạm vi cục bộ hoạt động chỉ "
            <<"trong một phần của hàm main."<<endl;
        cout<<"Cả fpSquare và fpCircle có thể được "
            <<"truy xuất ở đây cùng như "
            <<"hằng số toàn cục PI."<<endl;
    }
}
```

---

---

```

        cout<<"fpCircle hoạt động o day, "
            <<"nhưng fpSquare thì không."<<endl;
        XuatTieuDe();
    }
    void XuatTieuDe()
    {
        int iTriangle;
        cout<<"Hàng số toàn cục PI hoạt động o day "
            <<" cùng nhu biến cục bộ iTriangle."<<endl;
    }

```

---

Các luật về phạm vi:

1. Phạm vi của một đối tượng toàn cục, mọi đối tượng được khai báo hoặc định nghĩa bên ngoài tất cả các hàm
2. Các hàm được định nghĩa toàn cục. Điều đó có nghĩa rằng mọi hàm có thể gọi bất kỳ một hàm khác tại mọi thời điểm.
3. Phạm vi của một đối tượng cục bộ là từ vị trí nó được định nghĩa đến phần kết thúc khối lệnh nó được định nghĩa.
4. Phạm vi của các tham số hình thức cũng giống như phạm vi của các biến cục bộ được định nghĩa tại điểm bắt đầu của hàm.

Tại sao các biến hầu như không được định nghĩa toàn cục? Các cấu trúc lập trình tốt đảm bảo rằng tất cả sự truyền thông giữa các hàm sẽ rõ ràng thông qua việc dùng các tham số. Các biến toàn cục có thể bị thay đổi bởi bất kỳ hàm nào. Trong những dự án lớn, với nhiều lập trình viên có thể làm việc trên cùng một chương trình, các biến toàn cục là không đáng tin vì giá trị của chúng có thể bị thay đổi bởi bất kỳ hàm nào hoặc bất kỳ lập trình viên nào.

## 5.4. Các cách truyền tham số

### 5.4.1. Truyền tham trị

Chương trình dưới đây, ví dụ 81.1b, là một mở rộng của ví dụ 81.1 ở trên. Chương trình sẽ nhận về một tỷ lệ chi trả và số giờ làm việc và tính ra lương phải trả dựa trên các giá trị nhận về. Điều này có thể được thực hiện trong một hàm khác được gọi là hàm *TinhLuong*.

**Ví dụ 5.3:**

---

```
//Khai bao thu vien
#include<iostream>
using namespace std;

//Khai bao nguyen mau ham
void XuatMoTa();
void TinhLuong(float, int);

//Ham chinh
void main()
{
    //Khai bao bien
    float fpTyLeChiTra;
    int iSoGio;

    cout<<"CHAO MUNG DEN VOI CHUONG TRINH TINH LUONG"<<endl;
    XuatMoTa(); //Loi goi ham

    cout<<"Hay nhap vao so tien tren gio: "<<endl;
    cin>>fpTyLeChiTra;
    cout<<"Hay nhap vao so gio lam viec: "<<endl;
    cin>>iSoGio;

    TinhLuong(fpTyLeChiTra, iSoGio); //Goi ham tinh luong
    cout<<"Hi vong ban thich chuong trinh nay."<<endl;
}

//Dinh nghia ham
//*****
// Ten ham: XuatMoTa()
// Tac vu: Ham nay xuat len man hinh mo ta ve chuong trinh
```

---

---

```

// Dau vao: Khong
// Dau ra: Khong
//*****
void XuatMoTa() //Tieu de cua ham
{
    cout<<"*****"<<endl;
    cout<<"Chương trình nay nhan hai so (ty le chi tra va
        gio)"<<endl;
    cout<<"va xuat ra luong phai tra."<<endl;
    cout<<"*****"<<endl;
}
//*****
// Ten ham: TinhLuong(float, int)
// Tac vu: Ham nay tinh va xuat ra luong
// Dau vao: Ty le chi tra va so gio
// Dau ra: Khong
//*****
void TinhLuong(float TyLe, int SoGio)
{
    float fpLuong;
    fpLuong = TyLe * SoGio;
    cout<<"Luong la: "<<fpLuong<<endl;
}

```

---

Những phần tô đậm của chương trình này cho thấy việc phát triển một hàm khác. Hàm này có một chút khác biệt trong đó nó có các tham số bên trong ngoặc của lời gọi hàm, tiêu đề và nguyên mẫu. Các tham số là các thành phần truyền thông giữa các hàm. Hàm `TinhLuong` cần thông tin từ nơi gọi nó. Để tính lương phải trả nó cần tỷ lệ chi trả trên giờ và số giờ làm việc được truyền tới nó. Lời gọi hàm cung cấp thông tin này bằng các tham số bên trong các dấu ngoặc của lời gọi hàm `TinhLuong(fpTyLeChiTra, iSoGo);`. Cả `TyLeChiTra` và `iSoGio` được gọi là **các tham số thực**. Chúng trùng khớp 1-1 tương ứng với các tham số trong phần tiêu đề hàm mà chúng được gọi là `TyLe` và `SoGio`.

```
void TinhLuong(float TyLe, int SoGio)
```

Các tham số trong phần tiêu đề hàm được gọi là **các tham số hình thức**.

Một so sánh quan trọng giữa lời gọi hàm và tiêu đề hàm

Lời gọi hàm	Tiêu đề hàm
<i>TinhLuong(fpTyLeChiTra, iSoGio)</i>	<i>void TinhLuong(float TyLe, int SoGio)</i>

1. Lời gọi hàm không có bất kỳ từ nào đứng trước, còn tiêu đề hàm có từ `void` đứng trước.
2. Lời gọi hàm không có kiểu dữ liệu trước các tham số thực trong khi tiêu đề hàm phải có kiểu dữ liệu cho các tham số hình thức.
3. Mặc dù các tham số hình thức có thể có cùng tên với các tham số thực tương ứng, chúng không bắt buộc phải trùng tên. Tham số thực đầu tiên, *fpTyLeChiTra*, là tương ứng với *TyLe*, tham số hình thức đầu tiên. Điều này có nghĩa là giá trị của *fpTyLeChiTra* được gán cho *TyLe*. Tham số thực thứ hai, *iSoGio*, tương ứng với *SoGio*, tham số hình thức thứ hai, và gán giá trị cho *SoGio* giá trị của nó. Các tham số tương ứng phải có cùng kiểu dữ liệu.
4. Các tham số thực (*fpTyLeChiTra* và *iSoGio*) truyền các giá trị của chúng cho các tham số hình thức tương ứng. Đây là kiểu **truyền tham trị (pass by value)**. Điều này có nghĩa rằng *fpTyLeChiTra* và *TyLe* là hai vùng nhớ khác biệt. Nghĩa là giá trị trong *fpTyLeChiTra* tại thời điểm gọi sẽ được đặt vào vùng nhớ của *TyLe* như là giá trị khởi tạo. Cần lưu ý rằng nếu hàm *TinhLuong* đã thay đổi giá trị của *TyLe*, nó sẽ không ảnh hưởng giá trị của *fpTyLeChiTra* trong hàm chính. Về bản chất, truyền giá trị cũng giống như sao chép giá trị trong *fpTyLeChiTra* và đặt nó vào *TyLe*.

Một so sánh quan trọng nữa, đó là so sánh giữa nguyên mẫu hàm và tiêu đề hàm.

Nguyên mẫu hàm	Tiêu đề hàm
<b>Void</b> <b>TinhLuong(float,</b>	<b>void</b> <b>TinhLuong(float    TyLe,</b>

<code>int) ;</code>	<code>int SoGio)</code>
---------------------	-------------------------

1. Nguyên mẫu hàm có một dấu chấm phẩy ở cuối còn tiêu đề hàm thì không có.
2. Nguyên mẫu hàm chỉ liệt kê ra kiểu dữ liệu của các tham số và không có tên các tham số. Tuy nhiên, nguyên mẫu hàm cũng có thể liệt kê ra cả hai và như vậy nó chính xác giống như tiêu đề hàm ngoại trừ dấu chấm phẩy.

Chúng ta cùng nhìn lại ba phần – nguyên mẫu hàm, lời gọi hàm và tiêu đề hàm.

1. Tiêu đề phải có cả kiểu dữ liệu và tên cho tất cả **các tham số hình thức**.
2. Nguyên mẫu phải có kiểu dữ liệu và có thể có tên cho **các tham số hình thức**.
3. Lời gọi hàm phải có tên nhưng không được có kiểu dữ liệu cho **các tham số thực**.

**Ví dụ 5.4:** Chương trình tính lập phương của một biến, sử dụng cách truyền giá trị.

```
/*Chương trình tính lap phuong cua mot bien*/
#include<iostream>
using namespace std;

int TinhLapPhuongTruyenGiaTri(int n);

void main()
{
    cout<<"\t\tCHUONG TRINH TINH LAP PHUONG\n";
    int iN = 5;

    cout << "Gia tri ban dau cua n la " << iN << endl;
    /*Truyen gia tri iN cho ham tinh lap phuong va nhan
    ket qua tra ve */
    iN = TinhLapPhuongTruyenGiaTri(iN);
    cout<<"Gia tri n lap phuong la "<< iN<<endl;
}

int TinhLapPhuongTruyenGiaTri(int n)
```

```
{
    return n*n*n; //Lap phuong cua bien cuc bo n va tra ve ket
qua
}
```

Bước 1: Trước khi main gọi hàm TinhLapPhuongTruyenGiaTri

```
void main()
{
    int iN = 5;
    iN = TinhLapPhuongTruyenGiaTri(iN);
}
```

iN

5

```
int TinhLapPhuongTruyenGiaTri(int n)
{
    return n*n*n;
}
```

n

Chưa  
định  
nghĩa

Bước 2: Hàm TinhLapPhuongTruyenGiaTri được gọi thực thi

```
void main()
{
    int iN = 5;
    iN = TinhLapPhuongTruyenGiaTri(iN);
}
```

iN

5

```
int TinhLapPhuongTruyenGiaTri(int n)
{
    return n*n*n;
}
```

n

5

Bước 3: Sau khi hàm TinhLapPhuongTruyenGiaTri tính lập phương của n và trước khi hàm TinhLapPhuongTruyenGiaTri trả về cho hàm main

```
void main()
{
    int iN = 5;
    iN = TinhLapPhuongTruyenGiaTri(iN);
}
```

iN

5

```
int TinhLapPhuongTruyenGiaTri(int n)
{
    return n*n*n;
}
```

125

n

5

Bước 4: Sau khi hàm TinhLapPhuongTruyenGiaTri trả về main và trước khi gán kết quả cho iN

```
void main()
{
    int iN = 5;
    iN = TinhLapPhuongTruyenGiaTri(iN);
}
```

iN

5

125

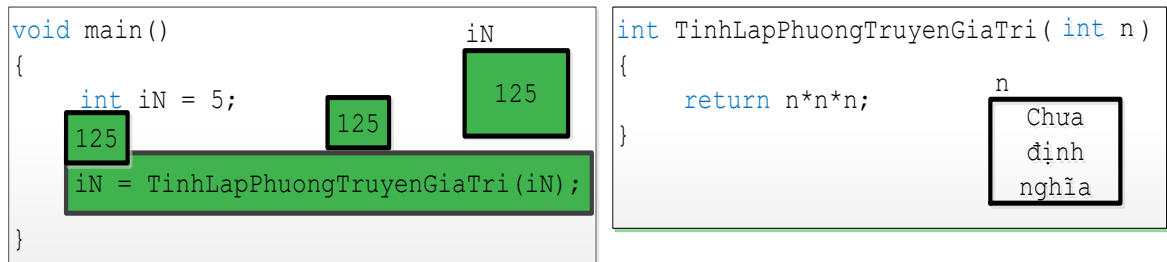
```
int TinhLapPhuongTruyenGiaTri(int n)
{
    return n*n*n;
}
```

n

Chưa  
định  
nghĩa

Bước 5: Sau khi main hoàn tất việc gán giá trị cho iN





#### 5.4.2. Truyền địa chỉ

Hàm nhận một địa chỉ như là một đối số phải định nghĩa một tham số con trỏ để nhận địa chỉ này.

**Ví dụ 5.5:** Chương trình lập phương của một biến, sử dụng cách truyền con trỏ.

```

/*Chương trình tính lap phuong cua mot bien*/
#include<iostream>
using namespace std;

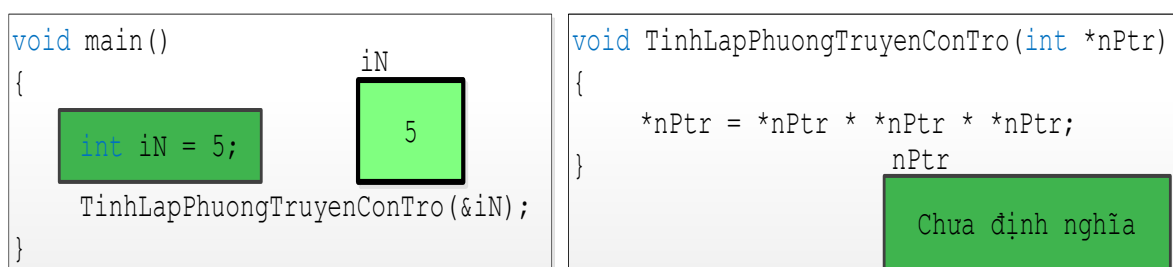
void TinhLapPhuongTruyenConTro(int *nPtr);

void main()
{
    cout<<"\t\tCHUONG TRINH TINH LAP PHUONG\n";
    int iN = 5;

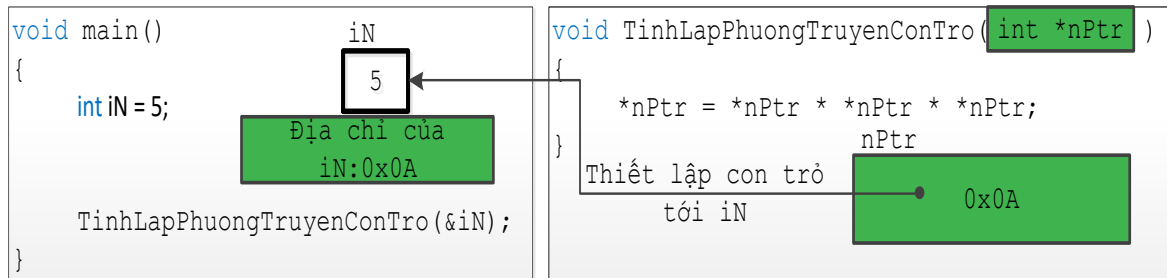
    cout << "Gia tri ban dau cua n la " << iN << endl;
    /*Truyen dia chi cua iN cho ham TinhLapPhuongTruyenConTro*/
    TinhLapPhuongTruyenConTro(&iN);
    cout<<"Gia tri n lap phuong la "<< iN<<endl;
    system("pause");
}

void TinhLapPhuongTruyenConTro(int *nPtr)
{
    *nPtr = *nPtr * *nPtr * *nPtr;
}
    
```

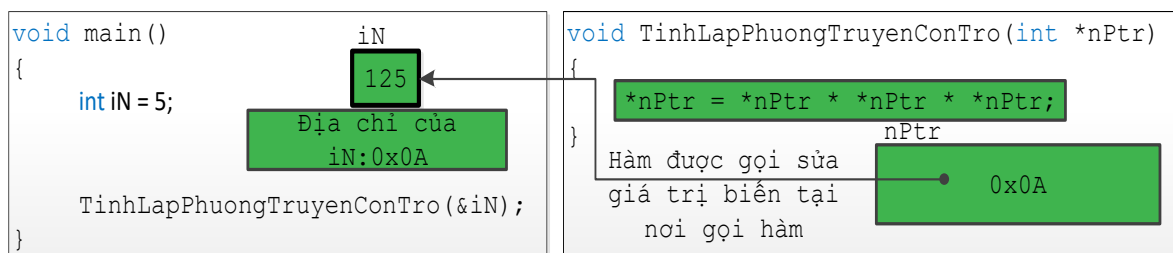
Bước 1: Trước khi main gọi hàm TinhLapPhuongTruyenConTro



Bước 2: Sau khi hàm `TinhLapPhuongTruyenConTro` được gọi và trước khi `*nPtr` tính lập phương



Bước 3: Sau khi `*nPtr` tính lập phương và trước khi chương trình trả điều khiển về cho main



### 5.4.3. Truyền tham chiếu

Giả sử chúng ta muốn hàm `TinhLuong` chỉ tính lương phải trả và sau đó truyền giá trị này trở lại hàm gọi mà không xuất nó lên màn hình. Chúng ta cần một tham số khác, không phải để nhận thông tin từ nơi gọi mà cung cấp thông tin về nơi gọi. Tham số đặc biệt này có thể không được truyền kiểu tham trị vì bất kỳ thay đổi nào được thực hiện trong một hàm tới một tham số hình thức được truyền tham trị không ảnh hưởng đến tham số thực tương ứng của nó. Thay vào đó, tham số này được **truyền bằng tham chiếu**, có nghĩa là hàm gọi sẽ đưa cho hàm được gọi vị trí ô nhớ của tham số thực thay vì sao chép giá trị được lưu trong ô nhớ đó. Điều này sau đó cho phép hàm được gọi đi vào ô nhớ và thay đổi giá trị của tham số thực.

#### Ví dụ 5.6:

---

```
/* Chương trình tính lương dựa vào số giờ làm việc trừ 20% thuế
*/
```

---

```
//Khai bao thu vien
#include<iostream>
using namespace std;
//Khai bao nguyen mau ham
void XuatMoTa();
void TinhLuong(float, int, float&);

//Ham chinh
void main()
{
    //Khai bao bien
    float fpTyLeChiTra;
    int iSoGio;
    float fpLuong, fpThucLanh;
    cout<<"CHAO MUNG DEN VOI CHUONG TRINH TINH LUONG" <<endl;
    XuatMoTa(); //Loi goi ham

    cout<<"Hay nhap vao so tien tren gio: "<<endl;
    cin>>fpTyLeChiTra;

    cout<<"Hay nhap vao so gio lam viec: "<<endl;
    cin>>iSoGio;

    TinhLuong(fpTyLeChiTra, iSoGio, fpLuong);

    fpThucLanh = fpLuong - (fpLuong*0.2);
    cout<<"Luong thuc lanh: "<<fpThucLanh<<endl;
    cout<<"Hi vong ban thich chuong trinh nay."<<endl;
}

//Dinh nghia ham
//*****
// Ten ham: XuatMoTa()
// Tac vu: Ham nay xuat len man hinh mo ta ve chuong trinh
// Dau vao: Khong
// Dau ra: Khong
```

---

---

```

//*****
void XuatMoTa() //Tieu de cua ham
{
    cout<<"*****"<<endl;
    cout<<"Chương trình nay nhan hai so (ty le chi tra va
        gio)"<<endl;
    cout<<"va xuất ra lương phải trả."<<endl;
    cout<<"*****"<<endl;
}

//*****
// Ten ham: TinhLuong(float, int, float &)
// Tac vu: Ham nay tinh va xuất ra lương
// Dau vao: Ty le chi tra va so gio
// Dau ra: Lương (thay doi tham so thuc tuong ung)
//*****
void TinhLuong(float TyLe, int SoGio, float &Luong)
{
    Luong = TyLe * SoGio;
}

```

---

Lưu ý rằng hàm *TinhLuong* bây giờ có ba tham số. Hai tham số đầu, *TyLe* và *SoGio*, được truyền tham trị trong khi tham số thứ ba có một dấu **&** ngay sau kiểu dữ liệu chỉ ra rằng nó được **truyền tham chiếu**. Tham số thực *fpLuong* cùng cặp với *Luong* vì cả hai đều là tham số thứ ba trong các danh sách tương ứng. Nhưng vì cặp này là truyền tham chiếu, cả hai tên *fpLuong* và *Luong* cùng chỉ một vùng nhớ. Do đó, hàm thực hiện việc gì trên tham số hình thức *Luong* thì cũng thay đổi giá trị của *fpLuong*. Sau khi hàm *TinhLuong* tính được giá trị của *Luong*, điều khiển trở về hàm chính và lúc này *fpLuong* đã được gán giá trị. Hàm main tiếp tục tính lương thực lãnh, bằng cách trừ đi 20% lương, và xuất kết quả.

**Ví dụ 5.7:** Chương trình tính lập phương của một biến, sử dụng cách truyền tham chiếu

```

/*Chương trình tính lap phuong cua mot bien*/
#include<iostream>

```

```
using namespace std;

void TinhLapPhuongTruyenThamChieu(int &nPtr);


void main()
{
    cout<<"\t\tCHUONG TRINH TINH LAP PHUONG\n";
    int iN = 5;

    cout << "Gia tri ban dau cua n la " << iN << endl;
    /*Truyen dia chi cua iN cho ham
    TinhLapPhuongTruyenThamChieu*/
    TinhLapPhuongTruyenThamChieu(iN);
    cout<<"Gia tri n lap phuong la "<< iN<<endl;
    system("pause");
}

void TinhLapPhuongTruyenThamChieu(int &n)
{
    n = n * n * n;
}
```

Bước 1: Trước khi main gọi hàm TinhLapPhuongTruyenThamChieu

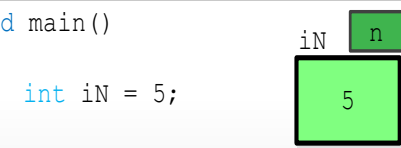
```
void main()
{
    int iN = 5;
    TinhLapPhuongTruyenThamChieu(iN);
}
```



```
void TinhLapPhuongTruyenThamChieu(int &n)
{
    n = n * n * n;
} /*Không có ô nhớ cho n trong hàm này*/
```

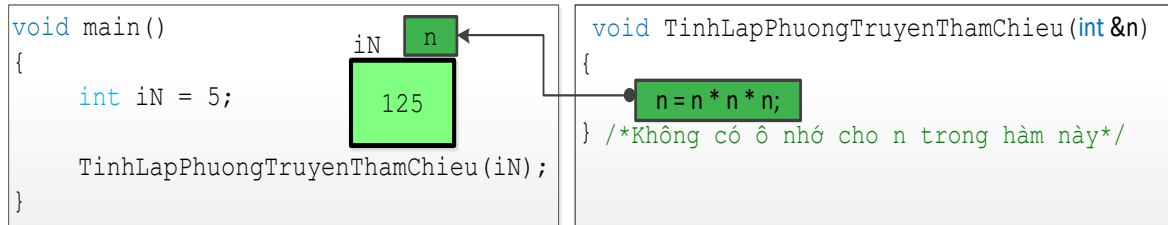
Bước 2: Sau khi main gọi hàm TinhLapPhuongTruyenThamChieu và trước khi hàm TinhLapPhuongTruyenThamChieu tính giá trị lập phương của n (Vùng nhớ của n trong TinhLapPhuongTruyenThamChieu cũng chính là vùng nhớ của biến tại nơi gọi hàm)

```
void main()
{
    int iN = 5;
    TinhLapPhuongTruyenThamChieu(iN);
}
```



```
void TinhLapPhuongTruyenThamChieu(int &n)
{
    n = n * n * n;
} /*Không có ô nhớ cho n trong hàm này*/
```

Bước 3: Sau khi tính lập phương của n và trả điều khiển về cho main



## 5.5. đệ quy

Một hàm gọi một hàm khác là bình thường, nhưng một hàm gọi lại chính nó người ta gọi là **hàm đệ quy**. Một hàm đệ quy là một hàm gọi lại chính nó trực tiếp hoặc gián tiếp thông qua hàm khác. Đệ quy là một chủ đề phức tạp được thảo luận nhiều trong các khóa đào tạo khoa học máy tính. Trong phần này ta chỉ xem xét các ví dụ đơn giản liên quan đến đệ quy.

Ví dụ tính giai thừa theo phương pháp đệ quy

Giai thừa của một số nguyên dương  $n$ , được viết là  $n!$  là tích của  $n*(n-1)*...*1$ , với  $1!$  là 1 và  $0!$  là 1. Ví dụ  $5! = 5*4*3*2*1 = 120$ .

Giai thừa của một số nguyên  $\text{Number} \geq 0$  có thể được viết theo dạng vòng lặp như sau:

---

```

long GiaiThua(long Number)
{
    long KetQua = 1;
    for(int i = Number; i>=1; --i)
        KetQua *= i;
    return KetQua;
}
    
```

---

Mặt khác  $n!$  cũng được định nghĩa đệ quy dựa trên quan hệ sau:

$$n! = n*(n-1)!$$

Như vậy, với định nghĩa đệ quy ta có thể tính  $n!$  thông qua  $(n-1)!$  Do đó hàm tính giai thừa ở trên có thể được viết theo dạng đệ quy như sau:

---

```

long GiaiThua(long Number)
{
    if(Number <= 1) //Truong hop suy bien
    
```

---

---

```
        return 1;
    else //buoc de quy
        return Number*GiaiThua(Number - 1);
}
```

---

### Lớp các bài toán giải được bằng đệ quy

Phương pháp đệ quy thường được dùng để giải các bài toán có đặc điểm:

- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp suy biến hay cơ sở, trong trường hợp này hàm được tính bình thường mà không gọi lại chính nó.
- Đối với trường hợp tổng quát (bước đệ quy), bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham số khác có kích thước nhỏ hơn tham số ban đầu và sau một số bước hữu hạn biến đổi cùng dạng bài toán đưa về trường hợp suy biến.

Từ đặc điểm trên ta có cú pháp chung của một hàm đệ quy như sau:

---

```
if(<trường hợp suy biến>)
{
    <trình bày cách giải>;
}
else //buoc de quy
{
    <Gọi lại hàm với đối số nhỏ hơn>;
}
```

---

## BÀI TẬP CHƯƠNG 5

### ❖ Bài tập lý thuyết:

Bài 5-1: Hãy cho biết cú pháp của hàm và ý nghĩa các thành phần trong cú pháp đó.

Bài 5-2. Hãy giải thích cách truyền tham trị và cho ví dụ minh họa.

Bài 5-3. Hãy giải thích cách truyền tham chiếu và cho ví dụ minh họa.

Bài 5-4. Hãy giải thích cách truyền địa chỉ và cho ví dụ minh họa.

Bài 5-5. Độ quy là gì? Cho ví dụ minh họa.

### ❖ Bài tập thực hành:

Bài 5-6. Viết chương trình cho phép người dùng nhập vào số nguyên dương N. Hãy tính tổng các giá trị từ 1 tới N theo công thức:

$$s = 1 + 2 + \dots + N$$

Bài 5-7. Viết chương trình cho phép người dùng nhập vào một số nguyên dương N. Hãy tính tổng theo công thức:

$$s = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N}$$

Bài 5-8. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số nguyên tố hay không?

**Biết: số nguyên tố là số lớn hơn 1 chỉ chia hết cho 1 và cho chính nó.**

Bài 5-9. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số chính phương hay không?

**Biết: Số chính phương** hay còn gọi là **số hình vuông** là **số** tự nhiên có căn bậc hai là một **số** tự nhiên, hay nói cách khác, **số chính phương** bằng bình **phương** (lũy thừa bậc 2) của một **số** tự nhiên

Bài 5-10. Viết chương trình cho phép người dùng nhập vào số nguyên N. Hãy cho biết số N vừa nhập có phải số hoàn hảo (hoàn thiện) hay không?



**Biết:** Số hoàn thiện (hay còn gọi là số hoàn chỉnh, số hoàn hảo hoặc số hoàn thành) là một số nguyên dương mà tổng các ước nguyên dương chính thức của nó (số nguyên dương bị nó chia hết ngoại trừ nó) bằng chính nó. Ví dụ: 6, 28, ...

Bài 5-11. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số nguyên tố?

Bài 5-12. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số chính phương?

Bài 5-13. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy cho biết các giá trị thuộc đoạn từ 1 đến  $N$  có bao nhiêu số hoàn hảo?

Bài 5-14. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số nguyên tố thuộc đoạn từ 1 đến  $N$ ?

Bài 5-15. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số chính phương thuộc đoạn từ 1 đến  $N$ ?

Bài 5-16. Viết chương trình cho phép người dùng nhập vào số nguyên dương  $N$ . Hãy xuất các số hoàn hảo thuộc đoạn từ 1 đến  $N$ ?

## CHƯƠNG 6 - MẢNG (ARRAY)

### ❖ Mục tiêu:

- Ghi nhớ cú pháp khai báo mảng một chiều, hai chiều.
- Mô tả được cách thức tổ chức bộ nhớ của mảng một chiều, hai chiều.
- Có khả năng tổng quát hóa các thao tác xử lý trên mảng một chiều, hai chiều.
- Ghi nhớ các thao tác trên mảng ký tự và chuỗi.
- Sử dụng được mảng để giải các bài toán thực tế.

### 6.1. Mảng một chiều

Trước giờ chúng ta đã thảo luận về một biến như là một vị trí đơn trong bộ nhớ máy tính. Có thể có một tập các vị trí vùng nhớ, tất cả có cùng kiểu dữ liệu, được nhóm lại cùng nhau dưới cùng một tên. Như là một tập hợp được gọi là một mảng. Giống như mọi biến khác, một mảng phải được định nghĩa để máy tính có thể cung cấp số lượng vùng nhớ tương ứng. Số lượng này dựa trên kiểu dữ liệu lưu trữ và số vùng nhớ.

#### Ví dụ 6.1:

---

```
//Khai bao thu vien
#include<iostream>
#include<cmath>
using namespace std;

//Khai bao kích thước mảng
const int SIZE = 100;

//Ham chính
void main()
{
    int arA[SIZE]; //Khai bao mảng với 100 phần tử
    ...
}
```

---

Mỗi phần tử của mảng, được truy xuất bởi tên mảng và vị trí trong mảng (subscript). Trong C++ vị trí trong mảng, đôi khi được đề cập như là chỉ mục (index), được đặt trong cặp ngoặc vuông. Số của vị trí trong mảng luôn bắt đầu từ 0 đến tổng số vùng nhớ trừ 1.

0	1	2	3	4	5	...	97	98	99

## 6.2. Khởi tạo mảng

Chúng ta có thể khởi tạo giá trị của mảng theo nhiều cách:

Khởi tạo mảng với tất cả các giá trị bằng 0:

```
int arA[SIZE] = {0};
```

Khởi tạo mảng với các giá trị được liệt kê, các phần tử còn lại bằng 0:

```
int arA[SIZE] = {1, 2, 4, 6, 9};
```

Mảng tự xác định kích thước:

```
int arA[] = {1, 2, 4, 6, 9}; //Kích thước mảng bằng 5
```

Khởi tạo mảng dùng vòng lặp:

```
int arA[SIZE];
for(int i= 0; i< SIZE; i++)
    arA[i] = 0;
```

Khởi tạo mảng với giá trị do người dùng nhập:

```
int arA[SIZE];
for(int i= 0; i< SIZE; i++)
    cin>>arA[i];
```

## 6.3. Truyền tham số mảng cho hàm

Mảng có thể được truyền như là các đối số (tham số) tới các hàm. Mặc dù các biến có thể được truyền tham trị hoặc tham chiếu, các mảng luôn được

**truyền con trỏ (pass by pointer)**, nó tương tự truyền tham chiếu. Điều này có nghĩa là các mảng có thể thay đổi giá trị bởi lời gọi hàm. Tuy nhiên, không bao giờ có dấu & đứng giữa kiểu dữ liệu và tên, như là truyền tham chiếu các tham số.

**Ví dụ 6.2:**

---

```
/*Thông tin CT & TG
Chương trình minh họa truyền mảng cho hàm
*/
//Khai báo thu viện
#include<iostream>
using namespace std;

//Khai báo kích thước mảng
const int MAXSIZE = 100;
//Khai báo nguyên mẫu hàm
void NhapDuLieu(int ar[], int &Size);
float TinhTrungBinh(const int ar[], int Size);

//Hàm chính
void main()
{
    int arA[MAXSIZE]; //Khai báo mảng với 100 phần tử
    int Size = 0; //Số lượng thực dụng
    float TrungBinh;

    NhapDuLieu(arA, Size);
    TrungBinh = TinhTrungBinh(arA, Size);

    cout<<"Trung bình của "<<Size<<" phần tử là:
    "<<TrungBinh<<endl;
}
/*****
- Tên hàm: NhapDuLieu
- Công việc: Hàm này nhập và lưu giá trị dữ liệu vào mảng
```

---

```
- Du lieu vao: Khong
- Du lieu ra: mang chua cac gia tri va so luong phan tu
*****/
void NhapDuLieu(int ar[], int &Size)
{
    int pos = 0;
    int Value;

    cout<<"Hay nhap gia tri hoac -99 de ket thuc: "<<endl;
    cin>>Value;
    while(Value != -99)
    {
        ar[pos] = Value; //luu du lieu vao mang
        pos++;           //tang chi muc cua mang

        cout<<"Hay nhap gia tri hoac -99 de ket thuc: "<<endl;
        cin>>Value;
    }
    Size = pos; //den khi vong lap ket thuc
                // pos se giu so luong phan tu da nhap
}
*****/
- Ten ham: TinhTrungBinh
- Cong viec: Ham nay tinh va tra ve trung binh cac gia tri
- Du lieu vaonhT: Mang cac so nguyen va kich thuoc
- Du lieu ra: trung binh cac gia tri trong mang
*****/
float TinhTrungBinh(const int a[], int Size)
{
    int sum = 0;
    for(int pos = 0; pos < Size; pos++)
        sum += a[pos]; //Cong cac gia tri vao tong
    return float(sum)/Size;
}
```

---

## 6.4. Chuỗi – mảng ký tự

### 6.4.1. Hằng chuỗi

Một hằng chuỗi là một chuỗi được đặt trong dấu ngoặc kép. Ví dụ, “chao ban”, “hoc C++” là các hằng chuỗi. Khi chúng được lưu vào bộ nhớ máy tính, ký tự rỗng được tự động thêm vào cuối. Chuỗi “hay nhap mot ky tu” được lưu trữ như sau:

H	a	y		n	h	a	p		m	o	t		k	y		t	u	\0
---	---	---	--	---	---	---	---	--	---	---	---	--	---	---	--	---	---	----

Khi một hằng chuỗi được sử dụng trong C++, địa chỉ vùng nhớ thực của nó được truy xuất. Trong câu lệnh:

---

```
cout<<"hay nhap mot ky tu";
```

---

Địa chỉ vùng nhớ được truyền tới đối tượng cout. cout sau đó hiển thị các ký tự liên tiếp cho đến khi gặp ký tự rỗng.

### 6.4.2. Lưu trữ chuỗi trong mảng

Thông thường chúng ta cần truy xuất đến những phần của chuỗi hơn là toàn bộ chuỗi. Ví dụ, chúng ta có thể muốn thay đổi các ký tự trong một chuỗi hoặc thậm chí so sánh hai chuỗi. Trong trường hợp này, một hằng chuỗi không phải là cái chúng ta cần. Thay vào đó, một mảng ký tự là sự lựa chọn thích hợp. Khi sử dụng mảng ký tự, không gian vùng nhớ phải đủ để lưu trữ chuỗi và cả ký tự rỗng. Ví dụ:

---

```
char cChuoi[10];
```

---

Câu lệnh này định nghĩa một mảng gồm 10 ký tự có tên là cChuoi. Tuy nhiên, mảng này chỉ có thể lưu trữ không nhiều hơn 9 ký tự khác rỗng vì một ô nhớ được dành riêng cho ký tự rỗng. Cùng xem xét đoạn lệnh sau:

---

```
char cChuoi[10];

cout<<"hay nhap mot chuoi khong qua 9 ky tu:";
```

---

---

```
cin>>cChuoi;
```

---

Hãy nhớ rằng máy tính thực sự xem cChuoi như là địa chỉ bắt đầu của mảng. Có một vấn đề có thể nảy sinh khi sử dụng cin trên một mảng ký tự. cin không biết cChuoi chỉ có 10 phần tử. Nếu người dùng nhập một chuỗi dài hơn 9 ký tự, thì cin sẽ ghi qua phần cuối của mảng. Chúng ta có thể giải quyết vấn đề này bằng cách sử dụng hàm getline. Nếu bạn sử dụng:

---

```
cin.getline(cChuoi, 10);
```

---

Thì máy tính biết rằng chiều dài tối đa của chuỗi, bao gồm cả ký tự rỗng, là 10. Do đó, cin sẽ đọc cho đến khi người dùng gõ ENTER hoặc cho đến khi đọc được 9 ký tự, tùy theo cái nào đến trước. Khi chuỗi là một mảng, nó có thể được xử lý theo từng ký tự.

#### 6.4.3. Các hàm thư viện xử lý chuỗi

C++ cung cấp nhiều hàm cho việc kiểm tra và thao tác với chuỗi. Dưới đây là một số hàm thông dụng:

Hàm	Ý nghĩa	Ví dụ
strlen(char s[])	Nhận về chiều dài chuỗi s	<pre>char cLine[40] = "Have a nice day!"; int iLength; iLength = strlen(cLine);</pre>
strcat(char s1[], char s2[])	Nối chuỗi s2 vào cuối chuỗi s1	<pre>char str1[25] = "Hello "; char str2[11] = "World"; strcat(str1, str2);</pre>
cin.get(char ch)	Đọc một ký tự và lưu vào biến ch	<pre>char firstchar, ch, secondchar; cin.get(firstChar);</pre>

		<pre>cin.get(ch); cin.get(secondChar);</pre>
<code>cin.get(char s[], int n)</code>	Đọc n ký tự và lưu vào chuỗi s	<pre>char strName[21]; cin.get(strName,21)</pre>
<code>cin.ignore(int n, char ch)</code>	Bỏ qua n ký tự hoặc gặp ký tự ch	<pre>cin.ignore(200, '\n');</pre>
<code>cin.getline(char s[], int n)</code>	Đọc từ 1 tới n ký tự và lưu vào s	<pre>char cLine[50]; cin.getline(cLine,50);</pre>
<code>isalpha(char ch)</code>	Kiểm tra giá trị trong ch có phải là chữ cái hay không	
<code>isdigit(char ch)</code>	Kiểm tra giá trị trong ch có phải chữ số hay không	
<code>isspace(char ch)</code>	Kiểm tra giá trị trong ch có phải khoảng trắng hay không	
<code>strlwr(char s[]);</code>	Đổi tất cả ký tự của s sang chữ thường	
<code>strupr(char s[])</code>	Đổi tất cả ký tự của s sang chữ	



	hoa	
strcmp(char s[],char t[])	So sánh 2 chuỗi s và t, giá trị âm nếu s<t, 0 nếu s bằng t, giá trị dương nếu s>t	
strcpy(char s[],char t[])	Gán nội dung của chuỗi t cho chuỗi s	

### 6.5. Lớp string

C++ có một thư viện string cung cấp nhiều cách thức thuận lợi để xử lý văn bản. Để sử dụng string chúng ta cần khai báo thư viện: **#include<string>**

Chúng ta có thể khai báo các biến string để lưu trữ các giá trị string. Chúng ta có thể gán một hằng string, hoặc các giá trị string khác vào một biến string. Chúng ta có thể nối hai string bằng toán tử "+".

---

```
string str1 = "I love";
string str2 = "You";
cout<<str1 + str2<<endl;
```

---

Các string trong C++ cũng có thể được so sánh theo thứ tự từ điển bằng các toán tử quan hệ. Các chữ cái thường lớn hơn chữ cái in hoa.

---

```
string str1 = "Apple"; string str2 = "apple";
string str3 = "apples"; string str4 = "orange";
bool t1 = (str1 == str2); // t1 = false
bool t2 = (str1 < str2); //t2 = true
bool t3 = (str2 < str3); //t3 = true
bool t4 = (str3 != str4); //t4 = true
bool t5 = (str4 > str3); // t5 = true
```

---

Trong thực tế, một string là một tổ hợp của một chuỗi các ký tự. Các ký tự trong một string có thể được truy xuất thông qua vị trí của chúng. Nghĩa là, một string với chiều dài  $n$ , những vị trí của các ký tự của nó có phạm vi từ 0 tới  $n - 1$ . Với một biến string, chúng ta có thể truy xuất đến một ký tự của nó thông qua toán tử [ $<\text{vị trí}>$ ].

---

```
string myMsg = "How are you doing?";
cout<<myMsg[4]<<" "<<myMsg[12]
<<myMsg[13]<<myMsg[16]<<endl; // a dog
```

---

### Các hàm thành viên trong lớp string

Thư viện string rất linh hoạt cho việc cung cấp nhiều phép toán liên quan tới thao tác xử lý chuỗi/ văn bản. Ví dụ: tìm chiều dài chuỗi, so sánh hai chuỗi, tìm kiếm/ rút trích chuỗi con, nối chuỗi, ... Mỗi biến string được kết hợp với một số hàm thành viên, như `s.length()`, `s.substr(...)`. Dưới đây là một số hàm thành viên thông dụng của string.

Hàm	Mô tả
<code>s.length()</code>	Trả về số lượng ký tự trong string <code>s</code>
<code>s.substr(x, y)</code>	Rút trích một chuỗi chuỗi con với chiều dài <code>y</code> bắt đầu tại vị trí <code>x</code> . Nếu không có <code>y</code> , một chuỗi con từ vị trí <code>x</code> tới cuối chuỗi sẽ được rút trích
<code>s.find(r)</code>	Kiểm tra chuỗi (string) <code>r</code> có xuất hiện trong chuỗi <code>s</code> hay không. Nếu có hàm trả về vị trí bắt đầu xuất hiện
<code>s.erase(x, n)</code>	Xóa <code>n</code> ký tự bắt đầu tại vị trí <code>x</code>
<code>s.replace(x, n, str)</code>	Thay thế <code>n</code> ký tự tại vị trí bắt đầu là <code>x</code> bằng <code>str</code> . <b>Lưu ý: chiều dài của <code>str</code> có thể lớn hơn <code>n</code>.</b>
<code>s1.compare(s2)</code>	So sánh chuỗi <code>s1</code> với <code>s2</code> . Giá trị trả về là âm

	nếu $s1 < s2$ , bằng 0 nếu $s1$ giống $s2$ , là dương nếu $s1 > s2$
<code>s1.swap(s2)</code>	Hoán đổi nội dung hai chuỗi
<code>s1.insert(index, s2)</code>	Thêm chuỗi $s2$ vào $s1$ sau vị trí $index$
<code>s.begin()</code>	Trả về vị trí bắt đầu chuỗi
<code>s.end()</code>	Trả về vị trí kết thúc chuỗi

**Ví dụ 6.3:**

---

```

/*Thông tin CT & TG
Chương trình minh họa xử lý chuỗi có kiểu string
*/
//Khai báo thư viện
#include<iostream>
#include<string>
using namespace std;

//Khai báo nguyên mẫu hàm
int DemSoKyTuSo(string str);

//Hàm chính
void main()
{
    string str;
    int iNumDigits = 0;

    cout<<"Hay nhập một chuỗi: "<<endl;
    getline(cin, str);

    iNumDigits = DemSoKyTuSo(str);

    cout<<"Số ký tự số trong chuỗi: "<<iNumDigits<<endl;

```

---

```

}
/*****
- Ten ham: DemSoKyTuSo
- Cong viec: Ham nay dem so luong chu so trong chuoai
- Du lieu vao: Mang ky tu
- Du lieu ra: So luong chi so trong mang ky tu
*****/

int DemSoKyTuSo(string s)
{
    int KQ = 0;
    int pos = 0;

    while(pos < s.length())
    {
        if(isdigit( s[pos]))
            KQ++;
        pos++;
    }
    return KQ;
}

```

---

## 6.6. Mảng 2 chiều

Mảng hai chiều lưu trữ dữ liệu như là tập hợp các dòng và cột. Mỗi phần tử của mảng hai chiều được xác định qua một cặp chỉ số dòng và chỉ số cột. Tất cả các phần tử có cùng một kiểu dữ liệu. Mỗi phần tử được truy xuất thông qua chỉ số dòng và chỉ số cột.

Khai báo mảng và khởi tạo mảng hai chiều:

Ví dụ:

```

int Mang2Chieu[3][6]; // khai bao mang 2 chieu

int temp[4][3] = {50, 70, 60, 48, 75, 62, 51,
                  69, 60, 52, 78, 63};

int    temp[4][3]    =    {{50,    70,    60},    {48,    75,    62},

```

---

---

```
        {51, 69, 60}, {52, 78, 63}}};  
  
int    t2[7][4]    =    {{50,    70,    60},    {48,    75,    62},  
                        {51, 69, 60}, {52, 78, 63}}};  
  
int    temp[][3]    =    {{50,    70,    60},    {48,    75,    62},  
                        {51, 69, 60}, {52, 78, 63}}};  
  
int    temp[][3]    =    {50,    70,    60,    48,    75,    62,    51,  
                        69, 60, 52, 78, 63};
```

---

### **Xử lý mảng hai chiều**

Để xử lý các giá trị dữ liệu được lưu trong mảng ta có thể sử dụng cú pháp sau:

---

```
int Mang2Chieu[3][6]; // khai bao mang 2 chieu  
  
for(int Dong = 0; Dong < SoDong; Dong++) //Voi moi dong  
    for(int Cot = 0; Cot < SoCot; Cot++) //Voi moi cot  
        if(Mang2Chieu[Dong][Cot] thỏa điều kiện)  
            Xử lý Mang2Chieu[Dong][Cot];
```

---

### **Ví dụ 6.4:**

---

```
/*Thông tin CT & TG  
Chương trình minh họa xử lý mảng 2 chiều  
*/  
  
//Khai bao thu vien  
#include<iostream>  
using namespace std;  
  
//Định nghĩa số dòng, số cột tối đa  
#define MAX_ROWS 50  
#define MAX_COLS 100  
  
  
//Khai bao nguyên mẫu hàm  
//Nhập mảng  
voidNhapMang2Chieu(int a[][MAX_COLS], int &m, int &n);
```

---

```
//Tinh toan
int TinhTongCacPhanTu(int a[][MAX_COLS], int m, int n);
//Tim kiem
int TimPhanTuNhoNhat(int a[][MAX_COLS], int m, int n);

//Ham chinh
void main()
{
    //Khai bao bien
    int arMang2Chieu[MAX_ROWS][MAX_COLS];
    int iNumRow, iNumCol;

    //Nhap lieu
    NhapMang2Chieu(arMang2Chieu, iNumRow, iNumCol);

    //Tinh toan
    int Sum = TinhTongCacPhanTu(arMang2Chieu, iNumRow, iNumCol);
    cout<<"Tong cac phan tu cua mang: "<<Sum<<endl;
    //Tim kiem
    int Min = TimPhanTuNhoNhat(arMang2Chieu, iNumRow, iNumCol);
    cout<<"Gia tri nho nhat trong mang: "<<Min<<endl;
}
/*****
- Ten ham: NhapMang2Chieu
- Cong viec: Ham nay nhap vao so dong, so cot va gia tri
              cac phan tu trong mang
- Du lieu vao:Khong
- Du lieu ra: mang cac gia tri va so dong, so cot
*****/
//Nhap mang
void NhapMang2Chieu(int a[][MAX_COLS], int &m, int &n)
{
    cout<<"Hay nhap so dong: "<<endl;
    cin>>m;
    cout<<"Hay nhap so cot: "<<endl;
```

```

        cin>>n;

        for(int row_pos = 0; row_pos < m; row_pos++)
            for(int col_pos = 0; col_pos < n; col_pos++)
            {
                cout<<"a["<<row_pos<<"]["<<col_pos<<"]="";
                cin>>a[row_pos][col_pos];
            }
    }

    /*****
    - Ten ham: TinhTongCacPhanTu
    - Cong viec: Ham nay nhan ve mot mang 2 chieu cac phan tu
                  va tinh tong cac phan tu co trong mang
    - Du lieu vao:mang 2 chieu cac so nguyen, so dong va so cot
    - Du lieu ra: Tong cac phan tu trong mang
    *****/
    int TinhTongCacPhanTu(int a[][MAX_COLS], int m, int n)
    {
        int Tong = 0;
        for(int row_pos = 0; row_pos < m; row_pos++)
            for(int col_pos = 0; col_pos < m; col_pos++)
                Tong = Tong + a[row_pos][col_pos];
        return Tong;
    }

    /*****
    - Ten ham: TinhTongCacPhanTu
    - Cong viec: Ham nay nhan ve mot mang 2 chieu cac phan tu
                  va tim gia tri nho nhat co trong mang
    - Du lieu vao:mang 2 chieu cac so nguyen, so dong va so cot
    - Du lieu ra: Gia tri nho nhat trong mang
    *****/
    int TimPhanTuNhoNhat(int a[][MAX_COLS], int m, int n)
    {

```

---

---

```
int minValue = a[0][0];
for(int row_pos = 0; row_pos < m; row_pos++)
    for(int col_pos = 0; col_pos < m; col_pos++)
        if(a[row_pos][col_pos] < minValue)
            minValue = a[row_pos][col_pos];
return minValue;
}
```

---



## BÀI TẬP CHƯƠNG 6

### ❖ Bài tập lý thuyết:

Bài 6-1. Hãy khai báo mảng một chiều gồm 100 phần tử các số nguyên.

Bài 6-2. Hãy khai báo mảng một chiều gồm 100 phần tử các số thực.

Bài 6-3. Hãy khai báo mảng 2 chiều các số nguyên gồm 50 dòng, 100 cột.

Bài 6-4. Hãy khai báo mảng 2 chiều các số thực gồm 100 dòng, 100 cột.

Bài 6-5. Hãy khai báo mảng 1 chiều chứa tối đa 50 ký tự.

Bài 6-6. Hãy cho biết các hàm xử lý mảng các ký tự đã được giới thiệu và cho ví dụ.

Bài 6-6. Hãy cho biết các hàm xử lý chuỗi (string) đã được giới thiệu và cho ví dụ.

### ❖ Bài tập thực hành:

Bài 6-7. Hãy viết các hàm nhập xuất mảng một chiều các số nguyên/ thực.

Bài 6-8. Hãy viết hàm cho biết vị trí của phần tử bằng x xuất hiện cuối cùng trong mảng. Với x nhập vào từ bàn phím.

Bài 6-9. Hãy viết hàm đếm các phần tử âm, dương trong mảng.

Bài 6-10. Tính giá trị trung bình của các số hoàn hảo trong mảng.

## CHƯƠNG 7 – KIỂU DỮ LIỆU CẤU TRÚC (STRUCT)

### ❖ Mục tiêu:

- Ghi nhớ cú pháp khai báo một kiểu dữ liệu cấu trúc.
- Phân biệt được kiểu dữ liệu cấu trúc và các biến cấu trúc.
- Áp dụng kiểu dữ liệu cấu trúc để giải các bài toán thực tế.

### 7.1. Giới thiệu

Trong phần này chúng ta tìm hiểu về cấu trúc. Giống như mảng, cấu trúc cho phép lập trình viên gom nhóm dữ liệu với nhau. Tuy nhiên, không giống như mảng, cấu trúc cho phép bạn gom nhóm các phần tử có kiểu dữ liệu khác nhau lại với nhau. Để minh họa cách sử dụng cấu trúc trong thực hành, ta cùng xem xét chương trình quản lý thông tin sinh viên. Thông tin mô tả một sinh viên gồm: họ tên, mã số sinh viên, địa chỉ, điểm trung bình toàn khóa học. Chúng ta có thể định nghĩa các biến như sau:

Các biến	Thông tin lưu trữ
char HoTen[25]	Họ tên sinh viên có chiều dài tối đa 24 ký tự
int MSSV	Giá trị số nguyên của mã số sinh viên
char DiaChi[50]	Địa chỉ của sinh viên
float DiemTrungBinh	Giá trị số thực của điểm trung bình toàn khóa

Tất cả các biến có liên quan với nhau bởi vì chúng có thể chứa thông tin về cùng một sinh viên. Chúng ta có thể đóng gói chúng với nhau bằng cách tạo ra một cấu trúc. Sau đây là phần khai báo:

---

```
struct SINHVIEN
{
    char HoTen[25];
    int MSSV;
    char DiaChi[50];
    float DiemTrungBinh;
}; //Chu ý dấu chấm phẩy ở đây
```

---

Nhãn là tên cấu trúc, trong trường hợp này là SINHVIEN. Nhãn được sử dụng giống như tên kiểu dữ liệu. Bên trong cặp ngoặc nhọn chúng ta định nghĩa các biến là các thành phần của cấu trúc. Vì vậy đoạn lệnh trên khai báo một cấu trúc có tên là SINHVIEN gồm bốn thành phần: HoTen, MSSV, DiaChi, DiemTrungBinh.

**Cú pháp khai báo kiểu dữ liệu cấu trúc:**

```
struct Tên_Kiểu_Cấu_trúc
{
    Các thành phần cấu trúc;
};
```

**Ví dụ:**

---

```
struct ngay
{
    int Ngay_Thu;
    char Ten_Thang[10];
    int Nam;
};
struct Nhan_Cong
{
    char Ten[25];
    char Dia_Chi[30];
    double Bac_Luong;
    ngay Ngay_Sinh;
    ngay Ngay_Vao_Co_Quan;
};
```

---

## 7.2. Khai báo biến cấu trúc

Lập trình viên phải nhận ra rằng khai báo cấu trúc không phải là khai báo biến. Thay vào đó nó cho phép trình biên dịch biết cấu trúc của nó là gì. Nghĩa là, khai báo một kiểu dữ liệu mới được gọi là kiểu dữ liệu cấu trúc. Bây giờ ta có thể định nghĩa các biến theo cú pháp như sau:

```
Tên_Kiểu_cấu_trúc Tên_biến_cấu_trúc;
```

**Lưu ý:**

- Các biến cấu trúc sẽ được cấp phát bộ nhớ một cách đầy đủ cho tất cả các thành phần của nó.

- Việc khai báo có thể được thực hiện đồng thời với việc định nghĩa cấu trúc. Muốn vậy cần đặt danh sách tên biến cấu trúc cần khai báo sau dấu } theo cú pháp sau:

```
struct Tên_kiểu_cấu_trúc  
{  
    Các thành phần cấu trúc;  
}Danh sách tên biến cấu trúc;
```

**Ví dụ:**

**SINHVIEN sv1, sv2;**

Cả sv1 và sv2 sẽ chứa bốn thành phần: HoTen, MSSV, DiaChi, DiemTrungBinh. Cả sv1 và sv2 được gọi là thể hiện (instances) của cấu trúc SINHVIEN.

---

```
struct SINHVIEN  
{  
    char HoTen[25];  
    int MSSV;  
    char DiaChi[50];  
    float DiemTrungBinh;  
}sv1, sv2;  
  
struct ngay  
{  
    int Ngay_Thu;  
    char Ten_Thang[10];  
    int Nam;  
}Ngay_Di, Ngay_Den;  
  
struct Nhan_Cong  
{  
    char Ten[25];  
    char Dia_Chi[30];  
    double Bac_Luong;  
    ngay Ngay_Sinh;  
    ngay Ngay_Vao_Co_Quan;  
}Nguoi_A, Nguoi_B;
```

---

### 7.3. Truy xuất các thành phần cấu trúc

Chắc chắn lập trình viên sẽ cần gán các giá trị cho các thành phần cấu trúc và cũng theo dõi giá trị mà các thành phần đang có. C++ cho phép bạn truy xuất các thành phần cấu trúc sử dụng toán tử chấm.

**Cú pháp truy xuất các thành phần cấu trúc:**

**Tên\_biến\_cấu\_trúc.Tên\_thành\_phần**

**Tên\_biến\_cấu\_trúc.Tên\_biến\_cấu\_trúc.Tên\_thành\_phần**

...

Cùng xem các ví dụ sau:

---

```
sv1.MSSV = 23;
```

---

Trong câu lệnh trên số nguyên 23 được gán cho thành phần MSSV của sv1. Toán tử chấm được sử dụng để kết nối tên thành phần với biến cấu trúc mà nó thuộc về.

---

```
cout << Nguoi_A.Ten;
```

---

Câu lệnh trên sẽ xuất lên màn hình tên của Nguoi\_A.

---

```
cout << Nguoi_A.Ngay_Sinh.Nam;
```

---

Câu lệnh trên sẽ đưa lên màn hình năm sinh của Nguoi\_A.

### 7.4. Khởi tạo cấu trúc

Chúng ta đã thấy các biến và mảng có thể được khởi tạo ngay khi khai báo. Các thành phần của cấu trúc cũng có thể được khởi tạo khi biến cấu trúc được khai báo. Giả sử ta có cấu trúc được định nghĩa như sau:

---

```
struct HINH_TRON
{
    float x;
    float y;
    float BanKinh;
};
```

---

Một biến cấu trúc Hinh1 có thể được khai báo và khởi tạo như sau:

---

```
HINH_TRON Hinh1 = {5.5, 3.2, 6};
```

---

Các giá trị trong danh sách được gán cho các thành phần theo thứ tự mà chúng xuất hiện. Do đó, 5.5 được gán cho Hinh1.x, 3.2 được gán cho Hinh1.y và 6 được gán cho Hinh1.BanKinh.

---

```
HINH_TRON Hinh1 = {5.5};
```

---

Câu lệnh này sẽ chỉ khởi tạo giá trị cho Hinh1.x và bỏ qua tất cả các thành viên còn lại không được khởi tạo.

Có một luật quan trọng khi khởi tạo các thành phần cấu trúc. Nếu một thành phần cấu trúc không được khởi tạo, thì tất cả các thành phần theo sau thành phần đó phải không được khởi tạo.

Cũng có một điểm quan trọng nữa là bạn không thể khởi tạo các thành phần cấu trúc trong khi bạn định nghĩa cấu trúc. Khai báo cấu trúc dưới đây là không hợp lệ:

---

```
struct HINH_TRON
{
    float x = 5.5; //Khong hop le
    float y = 3.2; //Khong hop le
    float BanKinh = 6; //Khong hop le
};
```

---

**Ví dụ 7.1:** Viết chương trình nhập vào tọa độ 2 điểm trong mặt phẳng OXY. Tính khoảng cách giữa 2 điểm vừa nhập.

---

```
//Chương trình tính khoảng cách 2 điểm

//Khai báo thư viện
#include<iostream>
#include<cmath>
using namespace std;

//Khai báo cấu trúc ĐIỂM
```

---

```
struct DIEM
{
    int x, y;
};
//Khai báo hàm
void NhapDiem(DIEM &d);
void XuatDiem(DIEM d);
float KhoangCach2Diem(DIEM d1, DIEM d2);

//Hàm chính
void main()
{
    cout << "\t\t CHUONG TRINH TINH KHOANG CACH 2 DIEM\n";
    //Khai báo biến
    DIEM d1, d2;
    float KhoangCach;

    cout << "Hay nhap toa do diem 1: " << endl;
    NhapDiem(d1);
    cout << "Hay nhap toa do diem 2: " << endl;
    NhapDiem(d2);

    KhoangCach = KhoangCach2Diem(d1, d2);

    cout << "Khoang cach giua diem ";
    XuatDiem(d1);
    cout << " va ";
    XuatDiem(d2);
    cout << " la: " << KhoangCach;
    system("pause");
}
void NhapDiem(DIEM &d)
{
    cout << "Hay nhap hoành do: ";
    cin >> d.x;
    cout << "Hay nhap tung do: ";
    cin >> d.y;
}
void XuatDiem(DIEM d)
{
    cout << "(" << d.x << ", " << d.y << ")";
}
float KhoangCach2Diem(DIEM d1, DIEM d2)
{
    float KQ = sqrt(pow(d1.x - d2.x, 2) + pow(d1.y - d2.y, 2));
    return KQ;
}
```

---

## 7.5. Mảng cấu trúc

Thay vì xác định mỗi một biến riêng biệt, chúng ta có thể sử dụng một mảng các cấu trúc. Một mảng các cấu trúc được định nghĩa giống như bất kỳ mảng nào khác. Ví dụ giả sử chúng ta đã có khai báo cấu trúc sau trong chương trình của chúng ta:

---

```
struct HINH_TRON
{
    float x;
    float y;
    float BanKinh;
    float ChuVi;
    float KhoangCachToiGocToaDo;
};
```

---

Sau đó câu lệnh dưới đây khai báo một mảng cấu trúc, arCircle, có 100 phần tử. Mỗi phần tử là một biến cấu trúc HINH\_TRON.

---

```
HINH_TRON arCircle[100];
```

---

Giống như các mảng đã được học trong các bài học trước, bạn có thể truy cập một phần tử mảng bằng cách sử dụng chỉ số của nó. Vì vậy, arCircle[0] là cấu trúc đầu tiên trong mảng, arCircle[1] là thứ hai, và vân vân. Cấu trúc cuối cùng trong mảng là arCircle[99]. Để truy cập một thành viên của một trong các phần tử mảng này, chúng ta vẫn sử dụng toán tử dấu chấm. Ví dụ, arCircle[9].ChuVi cung cấp cho ta thành phần ChuVi của arCircle[9]. Nếu chúng ta muốn hiển thị tâm và khoảng cách từ gốc tọa độ của 30 hình tròn đầu tiên, chúng ta có thể sử dụng như sau:

---

```
for (int count = 0; count < 30; count++)
{
    cout << arCircle[count].x<<endl;
    cout << arCircle[count].y<<endl;
    cout << arCircle[count].KhoangCachToiGocToaDo;
```

---



}

---

**Ví dụ 7.2:** Viết chương trình nhập vào tọa độ 3 điểm của một tam giác trong mặt phẳng OXY. Tính diện tích tam giác vừa nhập.

---

```
//Chương trình tính diện tích tam giác

//Khai báo thư viện
#include<iostream>
#include<cmath>
using namespace std;
//Khai báo cấu trúc ĐIỂM
struct DIEM
{
    int x, y;
};
struct TAM_GIAC
{
    DIEM d[3];
};

//Khai báo hàm
void NhapDiem(DIEM &d);
void XuatDiem(DIEM d);
float KhoangCach2Diem(DIEM d1, DIEM d2);
void NhapTamGiac(TAM_GIAC &t);
bool LaTamGiac(TAM_GIAC t);
float TinhDienTichTamGiac(TAM_GIAC t);
//Hàm chính
void main()
{
    cout << "\t\t\t CHUONG TRINH TINH DIEN TICH TAM GIAC\n";
    //Khai báo biến
    TAM_GIAC t;
    float DienTich;

    NhapTamGiac(t);
    if (LaTamGiac(t))
    {
        DienTich = TinhDienTichTamGiac(t);
        cout << "Diện tích tam giác vừa nhập là: "<< DienTich;
    }
    else
    {
        cout << "Tọa độ 3 điểm vừa nhập không là tam giác!";
    }

    system("pause");
}
```

---

```

void NhapDiem(DIEM &d)
{
    cout << "Hay nhap hoành do: ";
    cin >> d.x;
    cout << "Hay nhap tung do: ";
    cin >> d.y;
}
void XuatDiem(DIEM d)
{
    cout << "(" << d.x << ", " << d.y << ")";
}
float KhoangCach2Diem(DIEM d1, DIEM d2)
{
    float KQ = sqrt(pow(d1.x - d2.x,2) + pow(d1.y-d2.y, 2));
    return KQ;
}
void NhapTamGiac(TAM_GIAC &t)
{
    cout << "Hay nhap cac toa do cua tam giac:" << endl;
    for (int i = 0; i < 3; i++)
    {
        cout << "Hay nhap toa do thu " << i + 1 << ":"<<endl;
        NhapDiem(t.d[i]);
        cout << endl;
    }
}
bool LaTamGiac(TAM_GIAC t)
{
    bool KQ = false;
    float a, b, c;
    a = KhoangCach2Diem(t.d[0], t.d[1]);
    b = KhoangCach2Diem(t.d[0], t.d[2]);
    c = KhoangCach2Diem(t.d[2], t.d[1]);
    if (a*a+b*b>=c*c && a*a+c*c>=b*b && b*b + c*c >= a*a )
    {
        KQ = true;
    }
    return KQ;
}
float TinhDienTichTamGiac(TAM_GIAC t)
{
    float KQ, P, a, b, c;
    a = KhoangCach2Diem(t.d[0], t.d[1]);
    b = KhoangCach2Diem(t.d[0], t.d[2]);
    c = KhoangCach2Diem(t.d[2], t.d[1]);
    P = (a + b + c) / 2;
    KQ = sqrt(P*(P - a)*(P - b)*(P - c));
    return KQ;
}

```

---

## BÀI TẬP CHƯƠNG 7

### ❖ Bài tập lý thuyết:

Bài 7-1. Cho biết cú pháp khai báo kiểu dữ liệu cấu trúc và cho ví dụ minh họa?

Bài 7-2. Hãy khai báo cấu trúc điểm trong mặt phẳng OXY gồm hoành độ và tung độ và khai báo 2 điểm với giá trị khởi tạo ban đầu tùy ý.

Bài 7-3. Hãy khai báo kiểu dữ liệu cấu trúc phân số gồm tử số và mẫu số.

### ❖ Bài tập thực hành:

Bài 7-4. Viết chương trình nhập vào một phân số. Rút gọn phân số vừa nhập (nếu được).

Bài 7-5. Viết chương trình nhập vào hai phân số. Tính tổng, hiệu, tích và thương của hai phân số vừa nhập (các phân số kết quả phải ở dạng tối giản).

Bài 7-6. Viết chương trình nhập vào một phân số. So sánh 2 phân số, trả về 1 trong 3 giá trị: 0 – nếu 2 phân số bằng nhau, -1 – nếu phân số 1 nhỏ hơn phân số 2, 1 – nếu phân số 1 lớn hơn phân số 2.

Bài 7-7. Viết chương trình định nghĩa cấu trúc điểm trong mặt phẳng. Hãy viết chương trình nhập và tọa độ 2 điểm. Sau đó, tính khoảng cách giữa hai điểm vừa nhập.

## CHƯƠNG 8 – CON TRỎ (POINTER)

### ❖ Mục tiêu:

- Ghi nhớ cú pháp khai báo biến con trỏ.
- Phân biệt được biến con trỏ và biến thường.
- Áp dụng được biến con trỏ để xử lý các bài toán về mảng.

### 8.1. Biến con trỏ

Phải luôn phân biệt giữa địa chỉ bộ nhớ và dữ liệu được lưu trong đó. Cho đến thời điểm này chúng ta chỉ quan tâm đến dữ liệu được lưu trữ trong một biến chứ không phải địa chỉ của nó (nơi mà trong bộ nhớ chính biến được cấp phát). Trong phần này chúng ta sẽ xem xét các địa chỉ của các biến và các biến đặc biệt, được gọi là con trỏ, nó giữ các địa chỉ này. Địa chỉ của một biến có thể được đưa ra bằng cách thêm toán tử & trước tên biến.

---

```
int Tong;
```

```
cout<<&Tong; //Xuat len dia chi cua bien Tong
```

---

Toán tử & đặt trước tên biến Tong chỉ ra địa chỉ của nó, và nó không phải dữ liệu được lưu trong biến Tong. Trong hầu hết các hệ thống địa chỉ của biến sẽ được in lên màn hình là một giá trị thập lục phân thể hiện cho vị trí vật lý của biến. Để định nghĩa một biến con trỏ, ta đặt trước tên biến một dấu sao (\*).

Cú pháp khai báo biến con trỏ.

```
<kiểu dữ liệu> *<Tên biến con trỏ>;
```

Ví dụ:

---

```
int *ptr;
```

---

Dấu sao đằng trước biến ptr chỉ rằng ptr giữ địa chỉ của một vùng nhớ. Từ khóa int cho biết rằng vùng nhớ mà ptr trỏ tới có giá trị nguyên. ptr **KHÔNG** phải là một kiểu dữ liệu số nguyên, mà là một con trỏ giữ địa chỉ của một vị trí mà một giá trị số nguyên được lưu trữ. Sự khác biệt này là rất quan trọng!

Ví dụ sau sẽ cho thấy sự khác biệt này.

---

```
int Tong; //Tong se giu mot gia tri so nguyen

int *TongPtr; //TongPtr giu mot dia chi mot so nguyen
duoc luu
```

---

### ❖ Sử dụng ký hiệu &

Ký hiệu & cơ bản được sử dụng trong hai trường hợp:

Cách sử dụng phổ biến nhất chúng ta đã thấy là giữa kiểu dữ liệu và tên biến của một tham số truyền tham chiếu trong một tiêu đề/ nguyên mẫu. Đây được gọi là biến tham chiếu. Địa chỉ bộ nhớ của tham số được gửi tới hàm thay vì giá trị tại địa chỉ đó. Khi tham số được sử dụng trong hàm, trình biên dịch tự động truy cập vào vùng nhớ của biến. Tự động truy cập có nghĩa rằng vị trí của biến tham chiếu được truy cập để lấy ra hoặc lưu trữ một giá trị.

Chúng ta cùng nhìn lại hàm hoán vị để thấy rằng ký hiệu & được sử dụng trong các tham số là cần thiết để thực hiện hoán vị. Lý do là những giá trị này cần được thay đổi bởi hàm và do đó chúng ta cần cho địa chỉ (vị trí vùng nhớ) của những giá trị đó để hàm có thể ghi những giá trị mới vào bên trong chúng vì chúng cần được hoán đổi.

---

```
void HoanVi(int &a, int &b)
//Dau & chi ra rang cac tham so
//duoc truyen tham chieu
{
    int Tam = a; //Vi a la bien tham chieu,
                //trinh bien dich lay gia tri duoc luu
                //trong do va dat vao Tam
    a = b; //Gia tri moi duoc ghi truc tiep vao a
    b = Tam; //va trong b
}
```

---

Ký hiệu & cũng được sử dụng bất cứ khi nào chúng ta quan tâm đến địa chỉ của một biến hơn là nội dung bên trong nó.

---

```
void NhapMang1Chieu(int a[], int &n);
```

---

Sử dụng ký hiệu & để lấy địa chỉ của một biến có ích khi chúng ta gán giá trị cho các biến con trỏ.

### ❖ Sử dụng ký hiệu \*

Ký hiệu \* cũng cơ bản được sử dụng trong hai trường hợp:

Nó được sử dụng để định nghĩa các biến con trỏ:

---

```
int *ptr;
```

---

Nó cũng được sử dụng bất cứ khi nào chúng ta quan tâm đến nội dung của vùng nhớ được trỏ tới bởi một biến con trỏ, hơn là địa chỉ của nó. Khi được sử dụng theo cách này \* được gọi là toán tử dẫn xuất.

---

```
cout<<*ptr; //vi ptr la bien con tro, * dan xuat ptr,  
            //gia tri duoc luu tai vi tri ptr tro toi se duoc xuat  
            //len man hinh
```

---

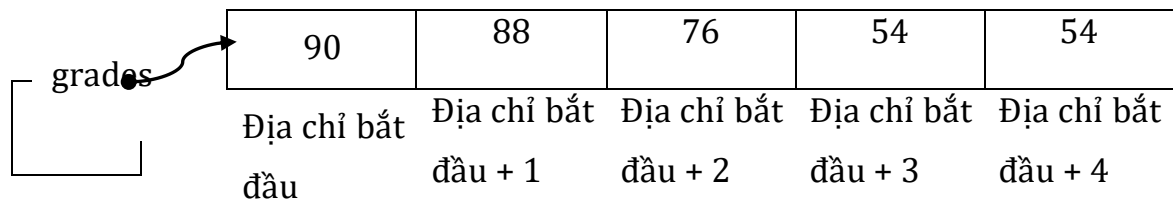
### ❖ Sử dụng \* và & cùng nhau:

Trong nhiều trường hợp \* và & có khả năng cùng xuất hiện. Dấu \* chỉ được sử dụng trước một biến con trỏ để chúng ta có thể lấy ra dữ liệu thực hơn là địa chỉ của biến. Ký hiệu & được sử dụng trên một biến để lấy địa chỉ của biến hơn là dữ liệu được lưu trữ trong nó.

## 8.2. Mảng con trỏ

Khi các mảng được truyền tới các hàm chúng được truyền bằng con trỏ. Tên một mảng là một con trỏ trỏ tới điểm bắt đầu của mảng. Các biến có thể chỉ lưu giữ một giá trị và do đó ta có thể tham chiếu tới giá trị đó chỉ bằng tên biến. Tuy nhiên, mảng lưu giữ nhiều giá trị. Tất cả các giá trị này không thể được tham chiếu chỉ bởi tên mảng. Con trỏ cho phép chúng ta truy cập tất cả các phần tử của mảng. Nhớ rằng tên mảng thực sự là một con trỏ lưu giữ địa chỉ

của phần tử đầu tiên trong mảng. Nếu grades là một mảng gồm 5 số nguyên, như hình bên dưới, grades thực chất là một con trỏ trỏ tới vị trí đầu tiên trong mảng, và grades[0] cho phép chúng ta truy xuất nội dung của vị trí đầu tiên đó.



Truy xuất một phần tử của mảng thông qua chỉ số được thực hiện bởi con trỏ số học. Chúng ta có thể truy xuất đến phần tử thứ hai của mảng với phát biểu grades[1], phần tử thứ ba với grades[2], vân vân. Cụm từ "Địa chỉ bắt đầu + 1" trong hình trên nghĩa là di chuyển tới một phần tử từ địa chỉ bắt đầu của mảng. Phần tử thứ ba được truy xuất bằng cách di chuyển qua hai phần tử và vân vân. Hai câu lệnh sau là tương đương:

---

```
cout<<grades[2];

cout<<*(grades + 2);
```

---

### **Ví dụ 8.1:**

---

```
/*
Chương trình minh họa con trỏ số học
*/
//Khai báo thu viện
#include<iostream>
using namespace std;

//Ham chinh
void main()
{
    int grades[] = {90, 88, 76, 54, 34};
    // Khai báo và khởi tạo mảng các số nguyên.
    // Vì grades là một tên mảng, nó thực sự là một con trỏ
```

---

---

```
// giữ địa chỉ đầu tiên của mảng.
cout << "grade đầu tiên là: "<< *grades << endl;
// Đầu * trước grades để lấy nội dung của phần tử đầu tiên
cout << "grade thứ hai là: "<< *(grades + 1) << endl;
cout << "grade thứ ba là: "<< *(grades + 2) << endl;
cout << "grade thứ tư là: "<< *(grades + 3) << endl;
cout << "grade thứ năm là: "<< *(grades + 4) << endl;
}
```

---

### 8.3. Các biến động

Khi khai báo mảng, lập trình viên phải ước lượng số lượng phần tử tối đa có thể được sử dụng vì mảng và kích thước của nó là tĩnh, nghĩa là, nó không thể thay đổi trong suốt quá trình thực thi của chương trình. Do đó, nếu mảng được định nghĩa lớn hơn cần thiết sẽ tốn bộ nhớ. Nếu được định nghĩa nhỏ hơn cần thiết, sẽ không đủ bộ nhớ để lưu trữ tất cả các phần tử. Sử dụng con trỏ (và hai toán tử new và delete trong phần sau) cho phép chúng ta cấp phát động đủ bộ nhớ cho mảng để tránh lãng phí bộ nhớ.

Điều này dẫn chúng ta đến các biến năng động. Con trỏ cho phép chúng ta sử dụng các biến động, có thể được tạo ra và tiêu hủy khi cần thiết trong một chương trình. Chúng tôi đã nghiên cứu các quy tắc phạm vi, xác định nơi mà một biến đang hoạt động. Liên quan đến điều này là khái niệm về thời gian sống, thời gian mà trong đó một biến tồn tại. Sự tồn tại của các biến động được điều khiển bởi chương trình thông qua các lệnh rõ ràng để cấp phát (tức là tạo) và hủy cấp phát (nghĩa là hủy) chúng. Toán tử new được sử dụng để cấp phát và toán tử delete được sử dụng để hủy cấp phát các biến động. Trình biên dịch theo dõi các biến không động trong bộ nhớ (các biến thảo luận trong các bài thực hành sách này) được cấp phát. Nội dung của chúng có thể được truy cập bởi tên của chúng. Tuy nhiên, trình biên dịch không theo dõi địa chỉ của một biến động. Khi lệnh new được sử dụng để cấp phát bộ nhớ cho một biến động, hệ thống trả về địa chỉ của nó và lập trình viên lưu trữ nó trong một biến con trỏ. Thông qua biến con trỏ chúng ta có thể truy cập vị trí bộ nhớ.



---

```

int *one; // one va two duoc khai bao la cac con tro
int *two; // tro toi kieu du lieu int
int result; // khai bao mot bien int luu gia tri tong
              // cua hai so kieu int.

one = new int; // cac cau lenh cap phat bo nho dong
two = new int; // du de luu tru gia tri kieu int
              // va gan dia chi cua chung cho cac bien con tro
              // one va two tuong ung.

*one = 10; // Nhung lenh gan gia tri 10
*two = 20; // vao vung nho duoc tro toi boi one
              // va 20 cho vung nho duoc tro toi boi two

result = *one + *two;
// Cong gia tri duoc luu trong vung nho
// duoc tro toi boi one va two
cout << "result = " << result << endl;
delete one; // Nhung lenh nay huy cap phat cac bien dong
delete two; // Nhung vung nho cua chung duoc giai phong va
              // chung khong con ton tai nua.

```

---

Bây giờ chúng ta sẽ sử dụng các biến động để cấp phát số lượng vùng nhớ thích hợp để lưu trữ một mảng. Bằng cách sử dụng toán tử new để tạo mảng, chúng ta có thể đợi cho đến khi chúng ta biết cần mảng lớn bao nhiêu trước khi tạo nó. Chương trình sau đây thể hiện ý tưởng này. Đầu tiên, người dùng được yêu cầu nhập vào số lượng các grades cần được xử lý. Sau đó, số lượng đó được sử dụng để cấp phát chính xác bộ nhớ đủ để giữ một mảng với số lượng yêu cầu của các phần tử cho các grades.

#### 8.4. Con trỏ với kiểu dữ liệu cấu trúc

Giống như bất kỳ kiểu nào khác, các cấu trúc có thể được trỏ tới bởi các kiểu con trỏ riêng của nó.

---

```

struct PHIM
{
    string TuaPhim;
    int NamSanXuat;

```

---

---

```
};
```

```
PHIM Phim1;  
PHIM *pPhim;
```

---

Ở đây Phim1 một biến của kiểu cấu trúc PHIM, và pPhim là một con trỏ tới các biến có kiểu cấu trúc PHIM. Do đó câu lệnh dưới đây là hợp lệ:

---

```
pPhim = &Phim1;
```

---

Giá trị của con trỏ pPhim sẽ được gán địa chỉ của của biến Phim1.

Bây giờ, chúng ta hãy xem một ví dụ khác kết hợp con trỏ và cấu trúc, và sẽ giới thiệu một toán tử mới: toán tử mũi tên (->).

### **Ví dụ 8.2:**

---

```
/*  
Chuong trình minh hoa con tro va cau truc  
Gioi thieu toan tu mui ten (->)  
*/  
  
//Khai bao thu vien  
#include <iostream>  
#include <string>  
#include <conio.h>  
  
using namespace std;  
  
//Dinh nghia kieu du lieu cau truc  
struct PHIM  
{  
    string TuaPhim;  
    int NamSanXuat;  
};
```

---

---

```

void main ()
{
    PHIM Phim1;
    PHIM *pPhim;
    pPhim = &Phim1;

    cout << "Hay nhap tua phim: ";
    getline (cin, pPhim->TuaPhim);
    cout << "Hay nhap nam san xuat: ";
    cin>>pPhim->NamSanXuat;

    cout << "\nBan da nhap:\n";
    cout << pPhim->TuaPhim;
    cout << " (" << pPhim->NamSanXuat << ")\n";
}

```

---

Toán tử mũi tên (->) là toán tử dereference (lấy từ (một con trỏ) địa chỉ của một mục dữ liệu được giữ ở một vị trí khác) được sử dụng riêng với các con trỏ tới các biến có các thành viên. Toán tử này phục vụ để truy cập trực tiếp thành viên của một biến từ địa chỉ của nó. Ví dụ, trong ví dụ trên:

---

```
pPhim->TuaPhim;
```

---

Câu lệnh này tương đương với:

---

```
*(pPhim).TuaPhim;
```

---

Cả hai câu lệnh trên, pPhim->TuaPhim và \*(pPhim).TuaPhim đều hợp lệ, và cả hai đều truy cập tới thành viên TuaPhim của cấu trúc dữ liệu được trỏ tới bởi con trỏ gọi là pPhim.

Bảng dưới đây tóm tắt các cách kết hợp có thể của toán tử cho các con trỏ và cho các thành viên cấu trúc:

Câu lệnh	Ý nghĩa	Câu lệnh tương đương
a.b	Thành viên b của biến a	

a->b	Thành viên b của biến được trỏ tới bởi con trỏ a	(*a).b
*a.b	Giá trị được trỏ tới của thành viên b của biến a.	*(a.b)

Một cấu trúc tự trỏ chứa một thành viên con trỏ mà trỏ tới chính cấu trúc đó. Ví dụ:

---

```
struct PHIM
{
    string TuaPhim;
    int NamSanXuat;
    PHIM *pNextPhim;
};
```

---

Định nghĩa một kiểu cấu trúc có tên là PHIM. Kiểu cấu trúc PHIM có ba thành viên – một thành viên TuaPhim có kiểu chuỗi, một thành viên NamSanXuat kiểu số nguyên và một con trỏ pNextPhim. Thành viên pNextPhim trỏ tới một cấu trúc có kiểu cấu trúc PHIM - một cấu trúc cùng kiểu với kiểu được khai báo ở đây, do đó ta có thuật ngữ "cấu trúc tự trỏ".

Các cấu trúc tự trỏ có thể được liên kết với nhau để hình thành các cấu trúc dữ liệu hữu ích như danh sách, hàng đợi, ngăn xếp và cây (các cấu trúc dữ liệu sẽ được giới thiệu trong các môn học sau). Một con trỏ NULL thường cho biết rằng kết thúc của một cấu trúc dữ liệu giống như các ký tự null (\0) chỉ ra sự kết thúc của một chuỗi.

### **Ví dụ 8.3:**

---

```
/*
Chương trình minh họa con trỏ và cấu trúc tự trỏ
*/
//Khai báo thu vien
#include <iostream>
#include <string>
using namespace std;
```

---

```
//Định nghĩa kiểu dữ liệu cấu trúc tu trỏ
struct PHIM
{
    string TuaPhim;
    int NamSanXuat;
    PHIM *pNextPhim; //con trỏ tới phim tiếp theo
};

void main ()
{
    PHIM *PhimDauTien = NULL; //Tạo danh sách với không nút
    PHIM *PhimHienTai = NULL;
    PHIM *PhimTruoc = NULL;

    char cContinue = 'Y';

    while(cContinue == 'Y' || cContinue == 'y')
    {
        PhimHienTai = new PHIM();

        if(PhimTruoc != NULL)
            PhimTruoc->pNextPhim= PhimHienTai;

        cout<<"Hay nhap ten phim: "<<endl;
        cin>>PhimHienTai->TuaPhim;
        cout<<"Hay nhap nam san xuat: "<<endl;
        cin>>PhimHienTai->NamSanXuat;
        PhimHienTai->pNextPhim = NULL;

        PhimTruoc = PhimHienTai;

        if(PhimDauTien == NULL)
            PhimDauTien = PhimHienTai;
```

---

---

```
        cout<<"Ban co muon them phim moi (y/n)?: ";
        cin>>cContinue;
    }
    //Xuat thong tin cac phim co trong danh sach
    PhimHienTai = PhimDauTien;
    while(PhimHienTai != NULL)
    {
        cout<<PhimHienTai->TuaPhim<<" ("<<PhimHienTai->
                                   NamSanXuat<<" ) ";
        PhimHienTai = PhimHienTai->pNextPhim;
        if(PhimHienTai != NULL)
            cout<<"-> ";
    }
}
```

---

## TÀI LIỆU THAM KHẢO

- [1]. D. DeFino, M. Bardzell, *Starting out with C++*, 6<sup>th</sup> Edition, Pearson. 2009.
- [2]. Y. Daniel Liang, *Introduction to Programming with C++*, 3<sup>rd</sup> Edition, Pearson, 2014.
- [3]. G.S Phạm Văn Ất, *Kỹ thuật lập trình cơ sở và nâng cao*, NXB Khoa học và kỹ thuật, 1999.