



# IT001 - NHẬP MÔN LẬP TRÌNH

## CHƯƠNG 2: GIỚI THIỆU VỀ THUẬT TOÁN

Thuật toán là một trong những cách mô tả lời giải của một bài toán mà máy tính có thể thực hiện được. Người lập trình sẽ dựa trên thuật toán để cài đặt chương trình tương ứng (theo cú pháp quy ước của ngôn ngữ lập trình). Trong chương này, chúng ta sẽ tìm hiểu về thuật toán và các phương pháp để biểu diễn một thuật toán.

Khoa Khoa học máy tính



# NỘI DUNG

- 2.1 Khái niệm về vấn đề/ bài toán
- 2.2 Các bước giải quyết vấn đề/ bài toán bằng máy tính
- 2.3 Khái niệm về thuật toán
- 2.4 Các tiêu chuẩn của thuật toán
- 2.5 Các phương pháp biểu diễn thuật toán
- 2.6 Độ phức tạp thuật toán
- 2.7 Một số ví dụ về thuật toán
- Bài tập



## 2.1 Khái niệm về vấn đề/ bài toán



# Dẫn nhập

- Giải quyết vấn đề (**problem**) là tìm giải pháp (**solution**) cho vấn đề đó.
- Mục tiêu lập trình: Xây dựng chương trình máy tính để giải quyết vấn đề thay cho con người, có thể đưa ra lời giải (**solution**) tương tự như con người.
  - Con người sử dụng tri thức để giải quyết vấn đề và thường diễn đạt bằng ngôn ngữ tự nhiên.
  - Máy tính sẽ giải quyết vấn đề như thế nào?

PROBLEM & SOLUTION





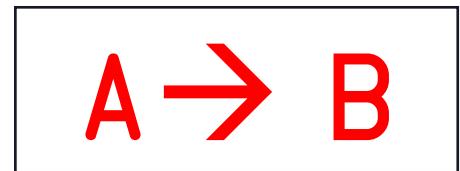
## 2.1 Khái niệm về vấn đề/ bài toán

- Một vấn đề (problem) trong ngũ cảnh của khoa học máy tính thường được định nghĩa là một trạng thái hay một tình huống cần giải quyết để tìm ra một lời giải, giải pháp hoặc câu trả lời.
- Mỗi vấn đề có thể được mô tả bằng một tập hợp các điều kiện ban đầu và một mục tiêu cụ thể muốn đạt được.
- Vấn đề thường đi kèm với một số ràng buộc (constraints) cần tuân theo, cũng như các điều kiện giới hạn và yêu cầu cụ thể.
- Ví dụ về một vấn đề là "tìm đường đi ngắn nhất từ điểm A đến điểm B trên một bản đồ có định sẵn các con đường và khoảng cách giữa chúng".
  - Đầu vào là bản đồ và các địa điểm A và B
  - Mục tiêu là tìm ra đường đi có chi phí ít nhất giữa hai điểm này.



## 2.1 Khái niệm về vấn đề/ bài toán

Mô hình vấn đề:



- **A**: giả thiết, điều kiện ban đầu (đầu vào – input)
- **B**: kết luận, mục tiêu cần đạt (đầu ra - output)
- Điều kiện và ràng buộc (Conditions and Constraints)
- Để giải quyết bài toán từ **A** thể dùng một số hữu hạn các bước suy luận để đạt được **B**. Quy trình giải bài toán  $A \rightarrow B$ :



Input bài toán

Giải bài toán

Xuất lời giải



## 2.1 Khái niệm về vấn đề/ bài toán

- Ví dụ 1: Tính diện tích của một tam giác biết chiều cao và cạnh đáy tương ứng (biết rằng các giá trị này là số thực nhỏ hơn 1 triệu, diện tích là một số thực với 2 số sau dấu thập phân).
  - Đầu vào: Nhập vào 2 số thực  $a$  và  $b$  ( $a$  là chiều cao,  $b$  là cạnh đáy tương ứng)
  - Đầu ra: Diện tích của tam giác
  - Điều kiện:
    - $a, b$  là số thực và nhỏ hơn 1 triệu
    - Đầu ra là một số thực với 2 số sau dấu thập phân
  - Giả định: Luôn nhập đúng theo yêu cầu



## 2.1 Khái niệm về vấn đề/ bài toán

Ví dụ 2: Bài toán tìm kiếm giá trị trong một dãy số (bất kỳ) cho trước.

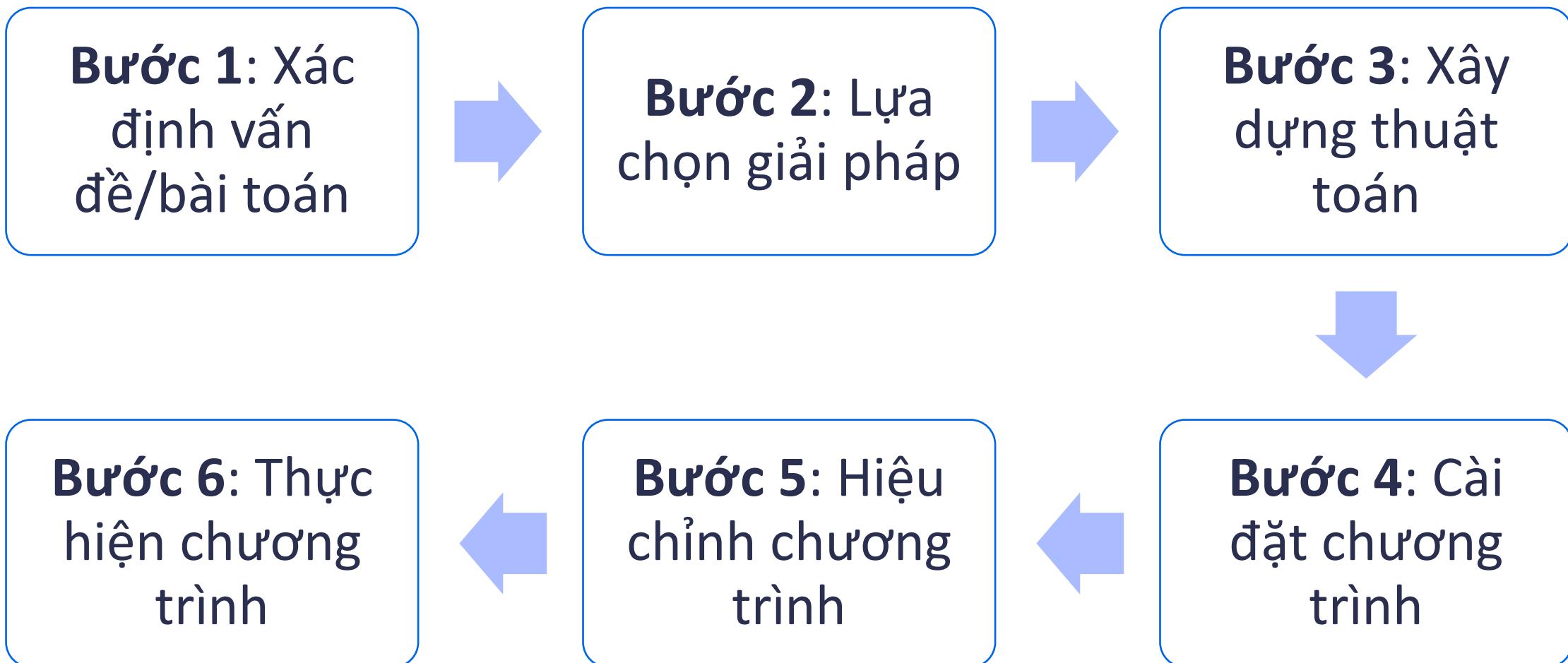
- **Đầu vào:** Nhập dãy số nguyên  $A=\{a_0, a_1, \dots, a_{n-1}\}$  có n phần tử và giá trị x cần tìm
  - **Đầu ra:** Xuất chỉ số i tương ứng  $x=A[i]$  hoặc NIL nếu không tìm thấy x
  - **Điều kiện:** Danh sách A có thể có thự tự hoặc không có thứ tự
- 
- Ví dụ 3: Tính nghiệm của phương trình  $ax+b=0$  (với  $a, b < 1$  tỷ).
    - **Input:** Đọc 2 số thực a và b nhỏ hơn 1 tỷ (giả định rằng người dùng luôn nhập đúng điều kiện)
    - **Output:** Nghiệm của phương trình tương ứng



## 2.2 Các bước giải quyết vấn đề/ bài toán bằng máy tính



## 2.2 Các bước giải quyết bài toán bằng máy tính





## 2.3 Khái niệm về thuật toán



# Dẫn nhập

- **Bài toán: Tìm Ước Số Chung Lớn Nhất**  
**(Finding the greatest common divisor)**
- **Định nghĩa:**
  - Ước số chung lớn nhất của 2 số a và b là số nguyên dương lớn nhất là ước của cả hai số đó (tiếng Anh: Greatest Common Factor hoặc Greatest Comon Divisor (GCF hay GCD)).
  - Nếu chỉ một trong hai số a, b bằng 0, số kia khác 0 thì GCF của chúng bằng giá trị tuyệt đối của số khác 0. Đặc biệt: UCLN(0, 0)=0.
  - Ví dụ, ước chung lớn nhất của 56 và 16 là 8.

=> **Vậy làm thế nào để tìm ra USCLN của 2 số?**



# Dẫn nhập

- Ý tưởng của cách giải bài toán này được nhà toán học người Hy Lạp cổ đại Euclid đưa ra như sau:
  - **Cách 1:** Dựa trên nguyên tắc: “UCLN của hai số nguyên không thay đổi khi thay số lớn hơn bằng hiệu của nó với số nhỏ hơn”. Ví dụ:  $\text{UCLN}(56, 16) = \text{UCLN}(40, 16) = 8$ , trong đó  $40 = 56 - 16$ . Lặp lại cho đến khi được hai số bằng nhau, cũng chính là UCLN của hai số ban đầu. *Lưu ý với phương pháp này cho phép đầu vào là hai số nguyên dương.*
  - **Cách 2:** Thế số lớn hơn bằng số dư của nó khi chia cho số nhỏ hơn, lặp lại thao tác trên cho đến khi số dư bằng 0. *Với phương pháp này thì đầu vào có thể là hai số nguyên bất kỳ.*



# Dẫn nhập

- Ta có thể thể hiện cách giải ở dạng ngắn gọn như sau:

- **Cách 1:** Lưu ý điều kiện ban đầu  $a > 0, b > 0$

$$\begin{cases} \text{UCLN}(a, a) = a \\ \text{UCLN}(a, b) = \text{UCLN}(a - b, b) \text{ nếu } a > b \\ \text{UCLN}(a, b) = \text{UCLN}(a, b - a) \text{ nếu } b > a \end{cases}$$

- **Cách 2:** (Điều kiện ban đầu:  $a \geq b$ , nên nếu  $a < b$  thì hoán đổi  $a, b$  trước khi thực hiện)

$$\begin{cases} \text{UCLN}(a, 0) = a \\ \text{UCLN}(a, b) = \text{UCLN}(b, a \bmod b) \end{cases}$$



# Dẫn nhập: Ví dụ

- **Cách 1:** Lưu ý điều kiện ban đầu  $a>0, b>0$

$$\begin{cases} \text{UCLN}(a, a)=a \\ \text{UCLN}(a, b)=\text{UCLN}(a-b, b) \text{ nếu } a > b \\ \text{UCLN}(a, b)=\text{UCLN}(a, b-a) \text{ nếu } b > a \end{cases}$$

- **Cách 2:**

$$\begin{cases} \text{UCLN}(a, 0)=a \\ \text{UCLN}(a, b)=\text{UCLN}(b, a \bmod b) \end{cases}$$

(Điều kiện ban đầu:  $a \geq b$ , nên nếu  $a < b$  thì hoán đổi  $a, b$  trước khi thực hiện)

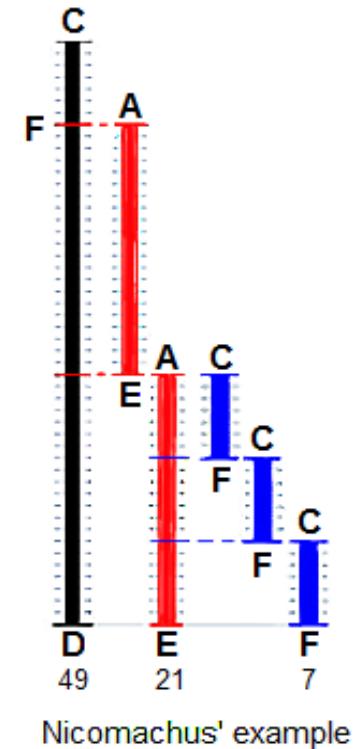
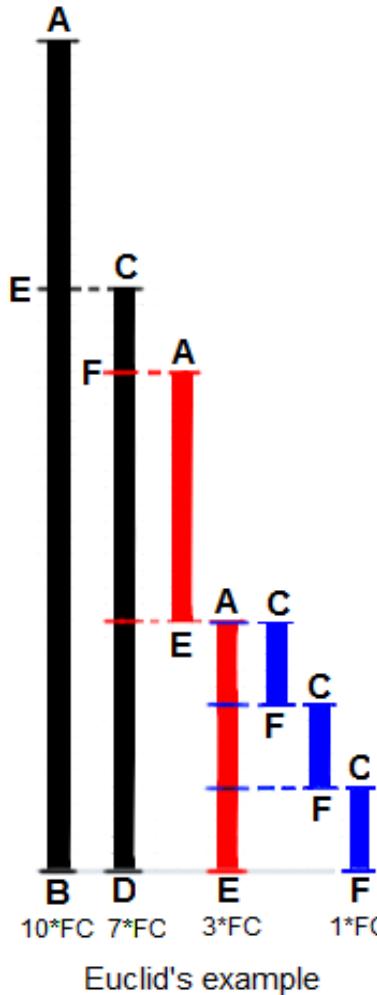
<b>Cách 1</b>	<b>a</b>	<b>b</b>	<b>Ghi chú</b>
Lần 1	154	35	$a > b \Rightarrow a=a-b$
Lần 2	119	35	$a > b \Rightarrow a=a-b$
Lần 3	84	35	$a > b \Rightarrow a=a-b$
Lần 4	49	35	$a > b \Rightarrow a=a-b$
Lần 5	14	35	$b > a \Rightarrow b=b-a$
Lần 6	14	21	$b > b \Rightarrow b=b-a$
Lần 7	14	7	$a > b \Rightarrow a=a-b$
Lần 8	7	7	$a=b \Rightarrow \text{Dừng}$

<b>Cách 2</b>	<b>a</b>	<b>b</b>	<b>Ghi chú</b>
Lần 1	154	35	$154 \% 35 = 14$
Lần 2	35	14	$35 \% 14 = 7$
Lần 3	14	7	$14 \% 7 = 0$
Lần 4	7	0	=> Kết thúc



# Dẫn nhập

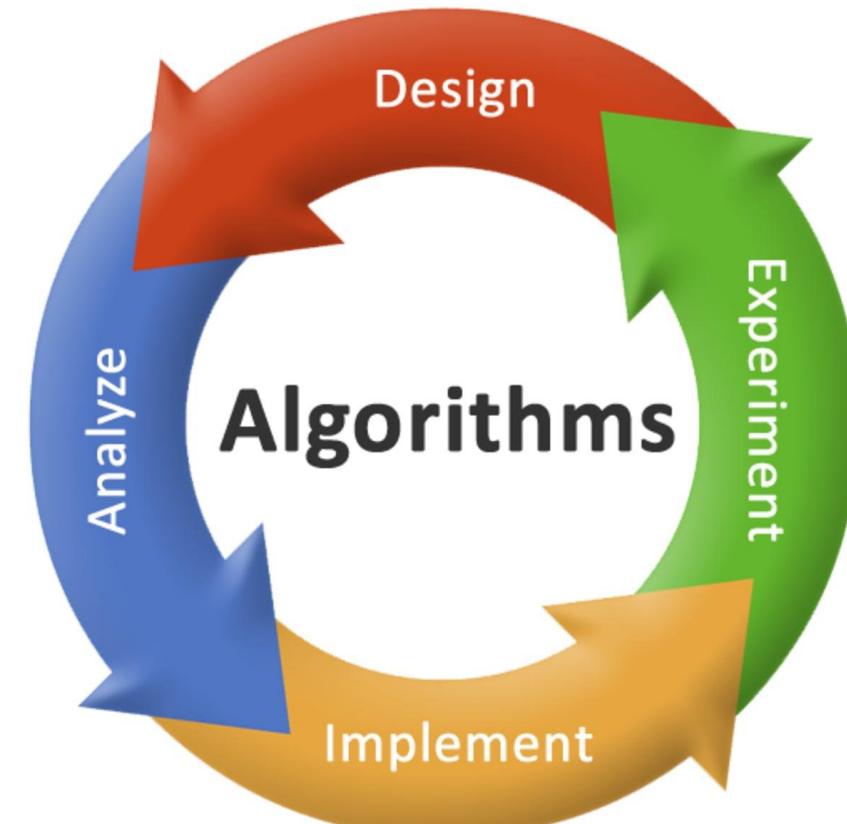
- Cách giải bài toán “Tìm Ước Số Chung Lớn Nhất” trình bày ở trên ta gọi là **Giải thuật Euclid**, hay thuật toán Euclid. Đây là một trong những thuật toán lâu đời nhất được sử dụng rộng rãi đến ngày nay.
- Từ “Thuật toán” (algorithm) từ bắt nguồn từ nhà toán học thế kỷ thứ 9: Muḥammad ibn Mūsā al-Khwārizmī, tên ông được Latinh hóa thành Algoritmi. (theo wikipedia)





## 2.3 Khái niệm về thuật toán

- **Thuật toán – Algorithm:** Là một dãy hữu hạn các thao tác cần thực hiện nhằm giải quyết một bài toán cụ thể nào đó.
- Mở rộng (máy tính): Là tập hợp (dãy, bước) **hữu hạn** các chỉ thị (hành động) được **định nghĩa rõ ràng** và **có thể thực thi** nhằm giải quyết một bài toán cụ thể nào đó. Quá trình hành động theo các bước này **phải dừng** và cho được **kết quả như mong muốn**.





# Sự cần thiết của thuật toán

- Tại sao sử dụng máy tính để xử lý dữ liệu?
  - Nhanh hơn.
  - Nhiều hơn.
  - Giải quyết những bài toán mà con người không thể hoàn thành được.
- Làm sao đạt được những mục tiêu đó?
  - Nhờ vào sự tiến bộ của kỹ thuật
  - Nhờ vào các thuật toán hiệu quả

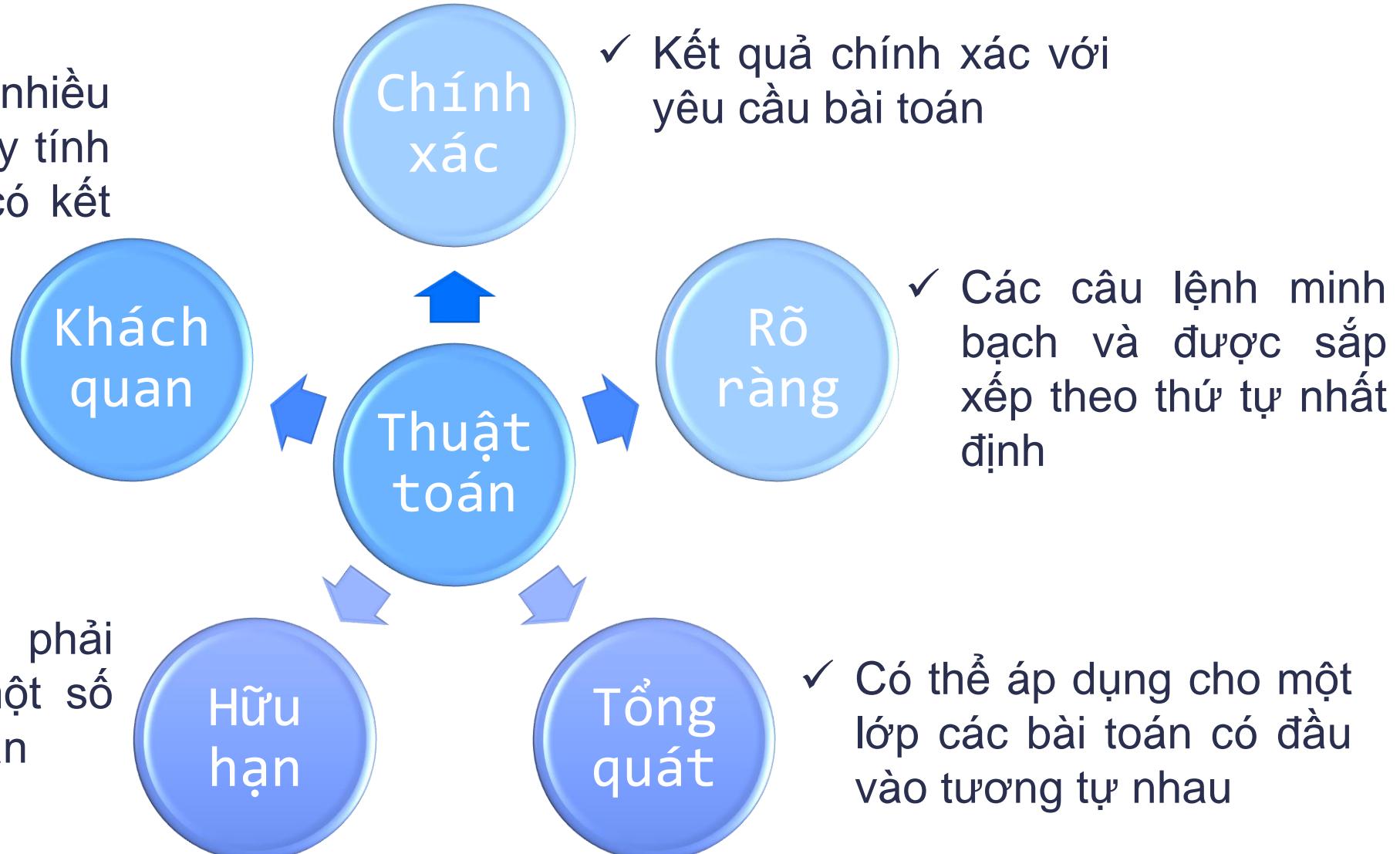


## 2.4 Các tiêu chuẩn của thuật toán



## 2.4 Các tiêu chuẩn của thuật toán

- ✓ Được thực thi nhiều lần bởi nhiều máy tính khác nhau đều có kết quả như nhau



- ✓ Kết quả chính xác với yêu cầu bài toán
- ✓ Các câu lệnh minh bạch và được sắp xếp theo thứ tự nhất định
- ✓ Có thể áp dụng cho một lớp các bài toán có đầu vào tương tự nhau



# Mở rộng khái niệm thuật toán

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán với các đặc tính sau:

Thường tìm được lời giải tốt gần đúng trong phạm vi cho phép

Thuật giải Heuristic thường thể hiện khá tự nhiên với cách suy nghĩ và hành động của con người.

Heuristic

Thời gian thực hiện nhanh hơn so với thuật toán tối ưu.



## 2.5 Các phương pháp biểu diễn thuật toán



## 2.5 Các phương pháp biểu diễn thuật toán

2.5.1 Dùng ngôn ngữ tự nhiên

2.5.2 Dùng lưu đồ - sơ đồ khối  
(Flowchart)

2.5.3 Dùng mã giả (Pseudo code)

2.5.4 Dùng ngôn ngữ lập trình





## 2.5 Các phương pháp biểu diễn thuật toán

### 2.5.1 Dùng ngôn ngữ tự nhiên



## 2.5.1 Sử dụng ngôn ngữ tự nhiên

- Sử dụng ngôn ngữ thường ngày để liệt kê các bước của thuật toán.
- Phương pháp biểu diễn này không yêu cầu người viết thuật toán cũng như người đọc thuật toán phải nắm các quy tắc.
- Tuy vậy, cách biểu diễn này:
  - Thường dài dòng
  - Không thể hiện rõ cấu trúc của thuật toán
  - Đôi lúc gây hiểu lầm hoặc khó hiểu cho người đọc
- Gần như không có một quy tắc cố định nào trong việc thể hiện thuật toán bằng ngôn ngữ tự nhiên.



## 2.5.1 Sử dụng ngôn ngữ tự nhiên

Ví dụ: Tìm nghiệm của phương trình bậc nhất  $ax+b=0$

- Đầu vào:  $a, b$  thuộc  $\mathbb{R}$
- Đầu ra: nghiệm phương trình  $ax + b = 0$

1. Nhập 2 số thực  $a$  và  $b$ .
2. Nếu  $a = 0$  thì
  - 2.1. Nếu  $b = 0$  thì
    - 2.1.1. Phương trình vô số nghiệm
    - 2.1.2. Kết thúc thuật toán.
  - 2.2. Ngược lại
    - 2.2.1. Phương trình vô nghiệm.
    - 2.2.2. Kết thúc thuật toán.
3. Ngược lại
  - 3.1. Phương trình có nghiệm.
  - 3.2. Giá trị của nghiệm đó là  $x = -\frac{b}{a}$
  - 3.3. Kết thúc thuật toán.



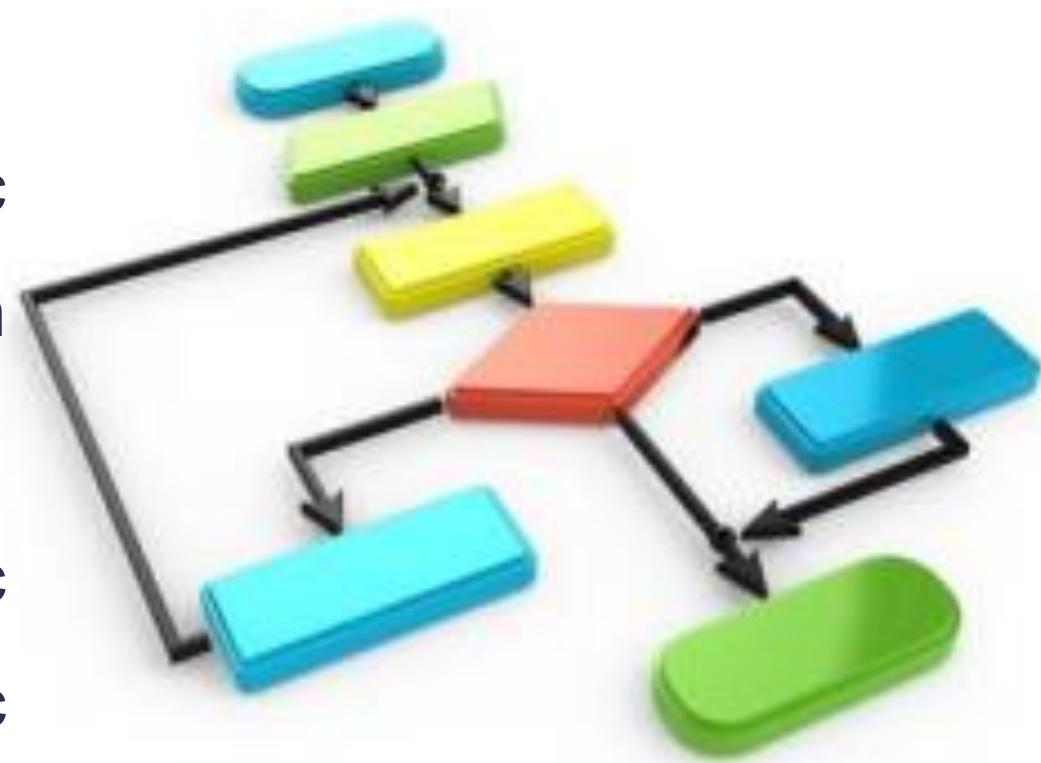
## 2.5 Các phương pháp biểu diễn thuật toán

### 2.5.2 Sử dụng lưu đồ - sơ đồ khối Flowchart



## 2.5.2 Sử dụng lưu đồ - sơ đồ khối Flowchart

- Lưu đồ là một cách biểu diễn trực quan của thuật toán trong lập trình.
- Lưu đồ sử dụng các hình khối khác nhau để biểu diễn các bước thực hiện của thuật toán.
- Lưu đồ giúp người lập trình, người đọc dễ dàng theo dõi và hiểu các bước trong thuật toán.





## 2.5.2 Sử dụng lưu đồ - sơ đồ khối Flowchart

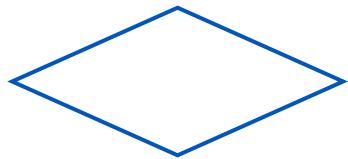
- Các ký hiệu cơ bản của lưu đồ:



Terminal: Khối giới hạn  
Chỉ thị bắt đầu và kết thúc.



Input/output: Khối vào ra  
Nhập/Xuất dữ liệu.



Decision: Khối lựa chọn  
Tùy điều kiện sẽ rẽ nhánh.



Process: Khối thao tác  
Ghi thao tác cần thực hiện.



Flowline: Đường đi  
Chỉ hướng thao tác tiếp theo.



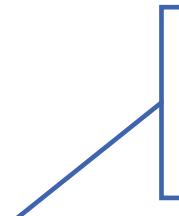
On-page Connector: Điểm nối cùng trang  
Điểm tập kết dòng điều khiển (flowline)  
trên một trang



Off-page Connector: Điểm nối sang trang  
Điểm tập kết của dòng điều khiển  
(flowline) từ trang khác



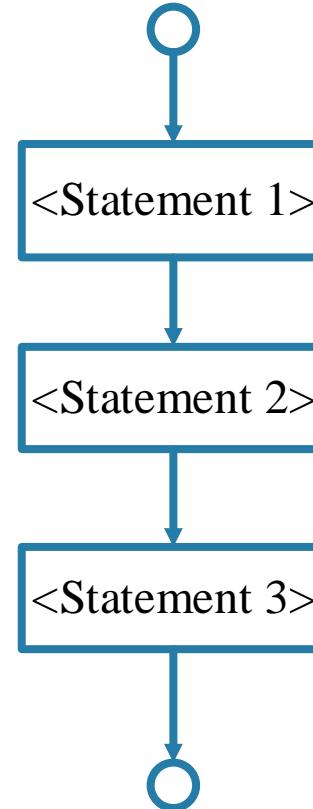
Predefined Process/Function  
Đại diện cho một nhóm các câu  
lệnh thực hiện một tác vụ.



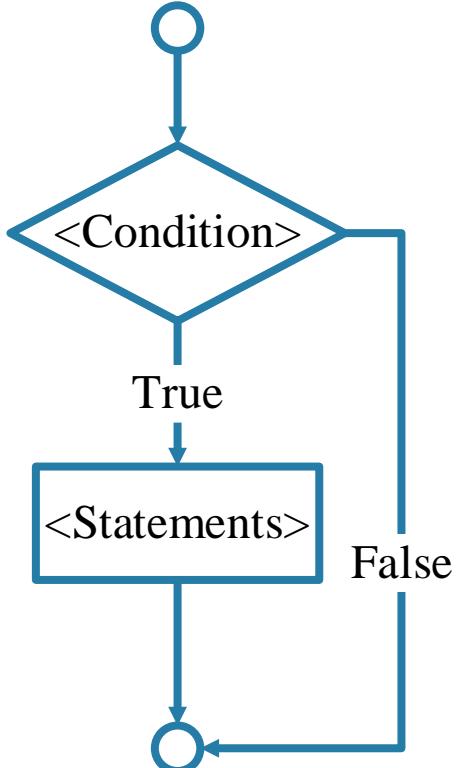
Annotation  
Giải thích



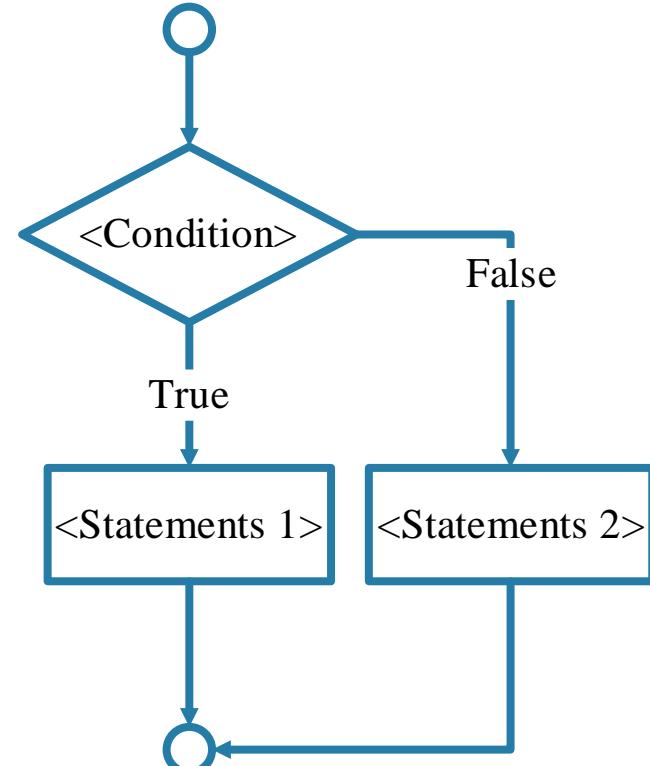
# Ví dụ các cấu trúc thường gặp khi vẽ lưu đồ



Các lệnh tuần tự

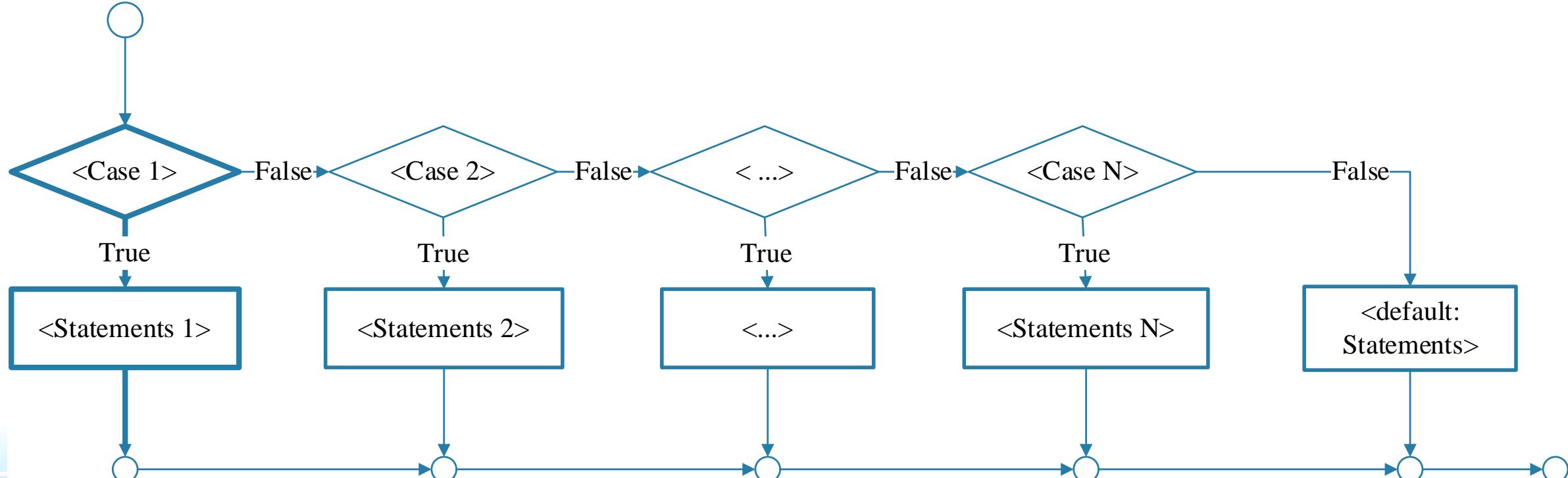


Cấu trúc điều kiện rẽ nhánh





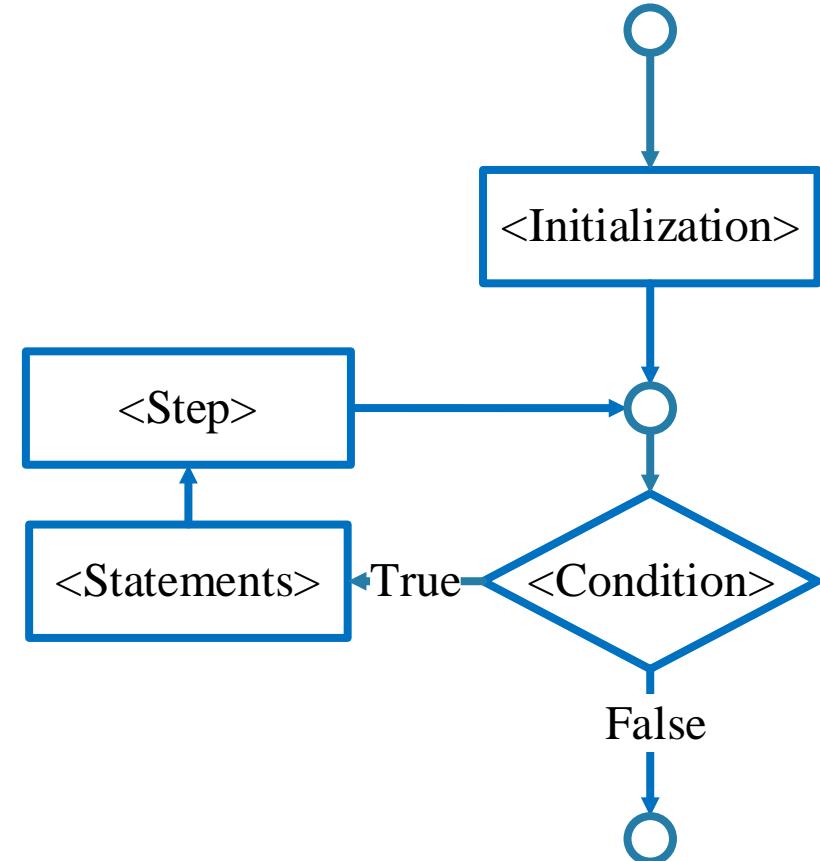
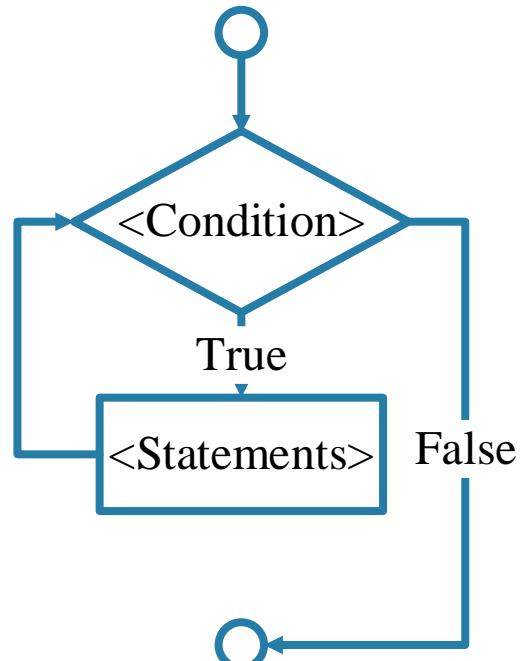
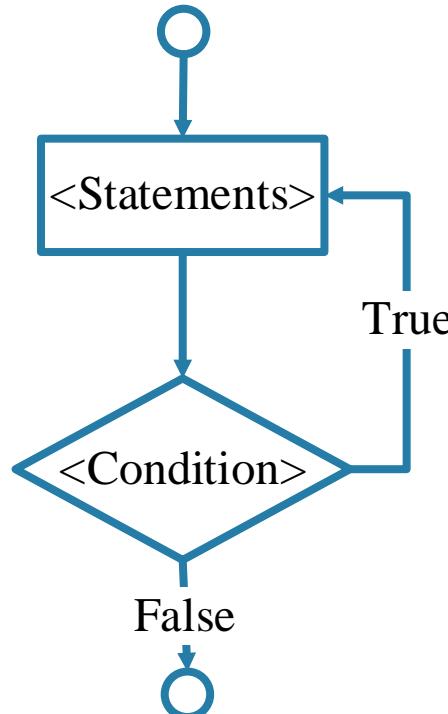
# Ví dụ các cấu trúc thường gặp khi vẽ lưu đồ



Cấu trúc điều kiện rẽ nhánh



# Ví dụ các cấu trúc thường gặp khi vẽ lưu đồ

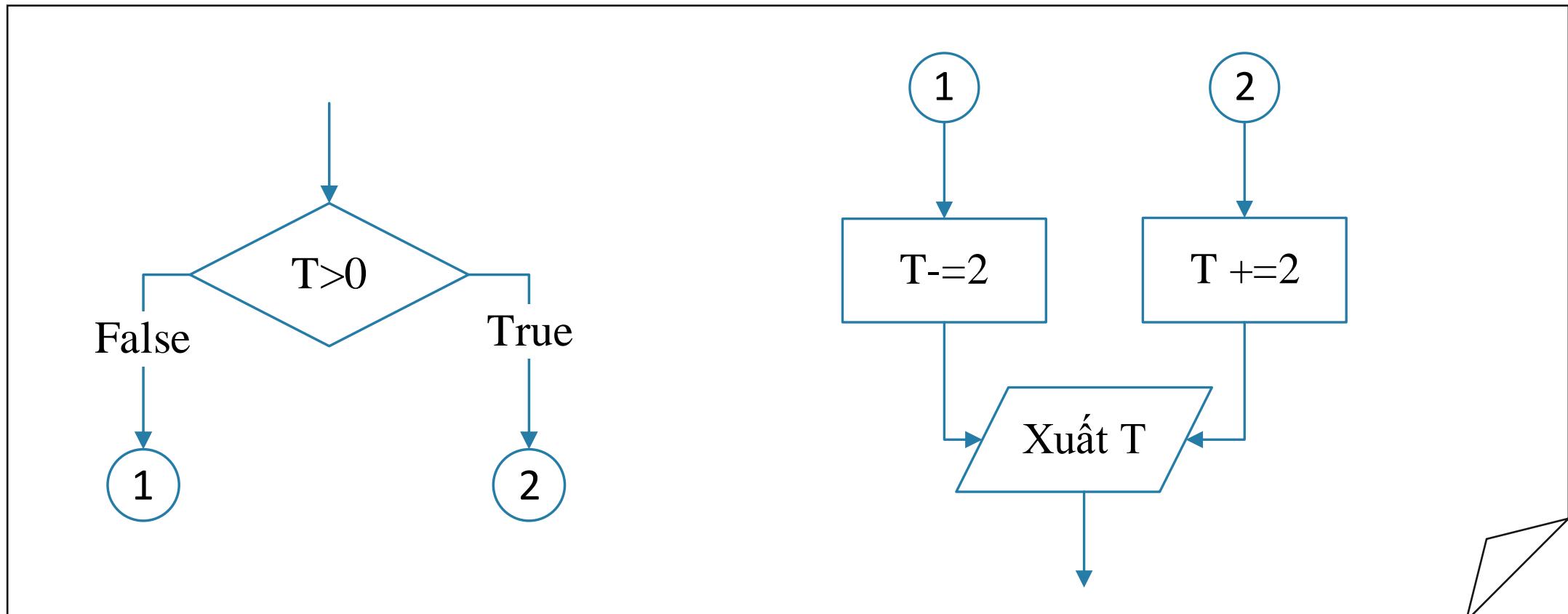


Cấu trúc vòng lặp



# Ví dụ ký hiệu On-page Connector

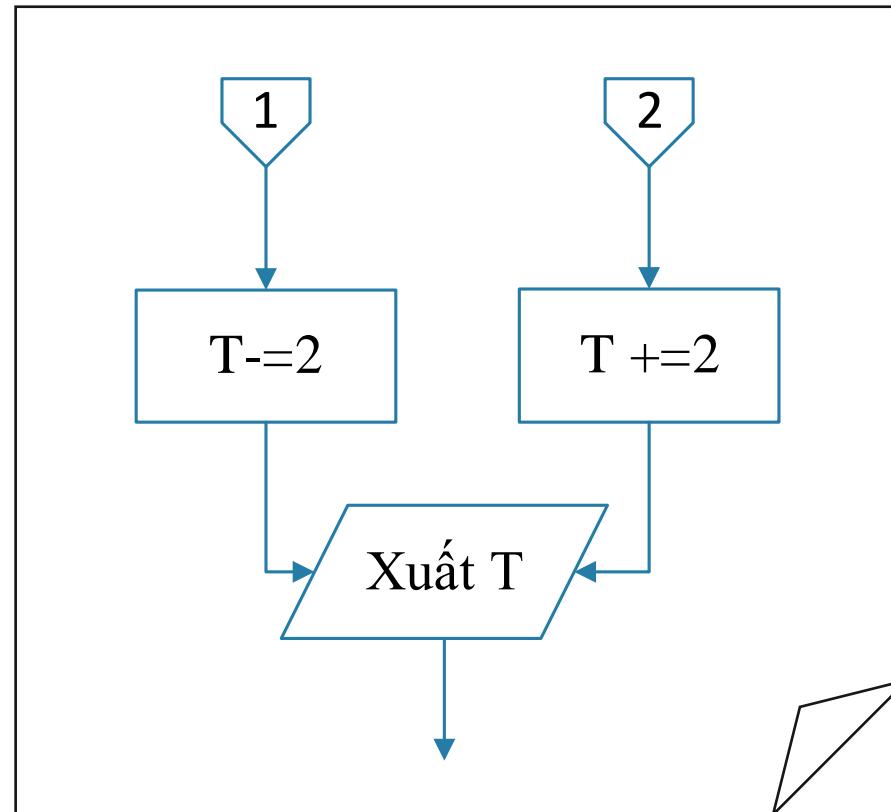
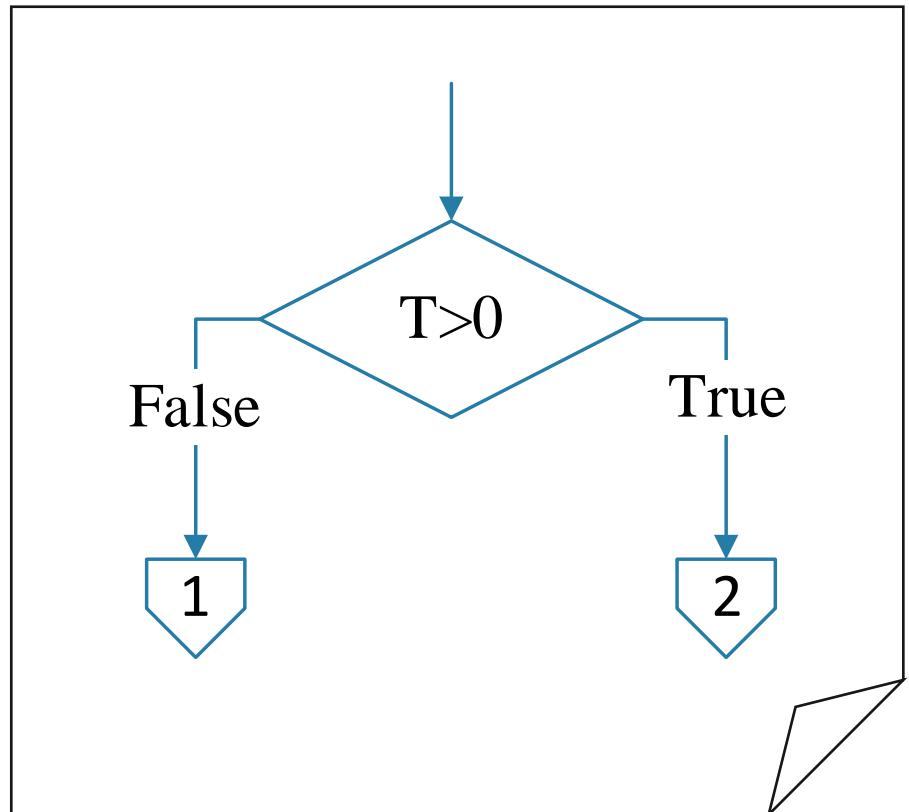
- **On-page Connector:** Điểm tập kết dòng điều khiển (flowline) trên một flowchart trong cùng một trang.





# Ví dụ ký hiệu Off-page Connector

- **Off-page Connector:** Điểm tập kết của dòng điều khiển (flowline) từ trang khác.

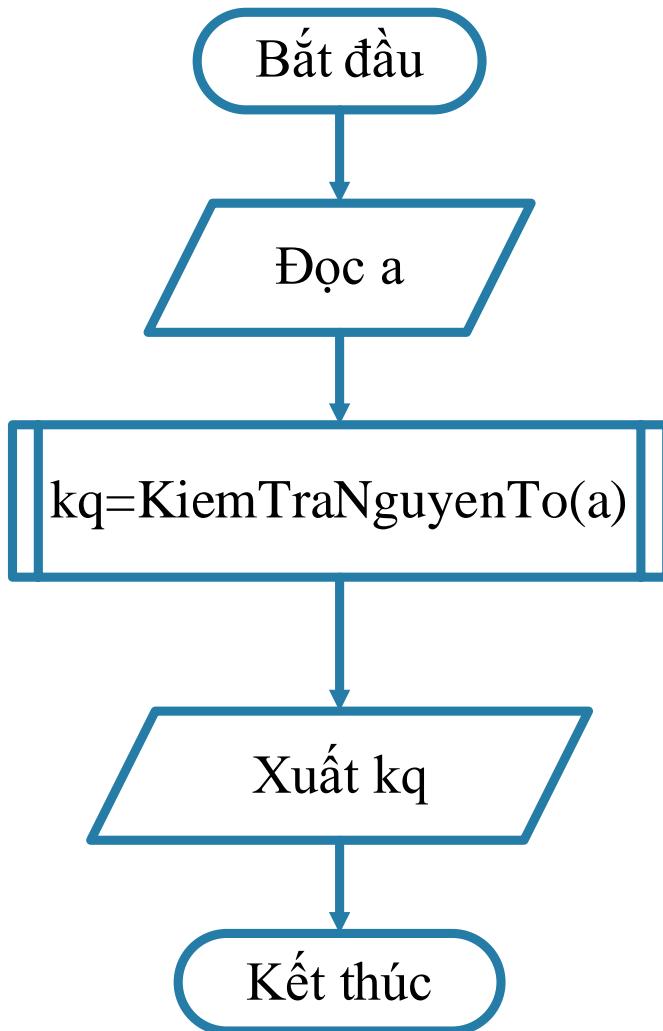




# Ví dụ ký hiệu Predefined Process/Function

- **Predefined Process/Function:**

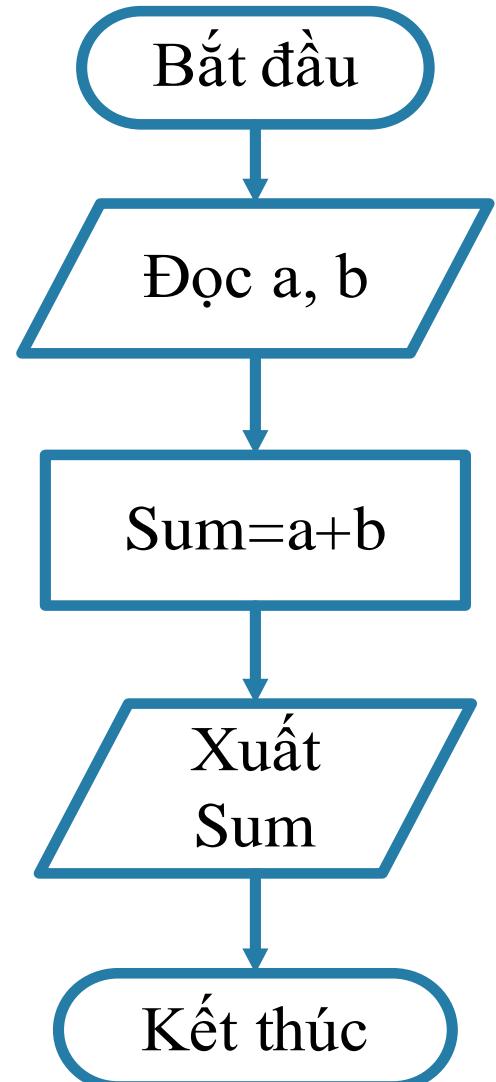
Đại diện cho một nhóm các câu lệnh thực hiện một tác vụ.





## Ví dụ 1

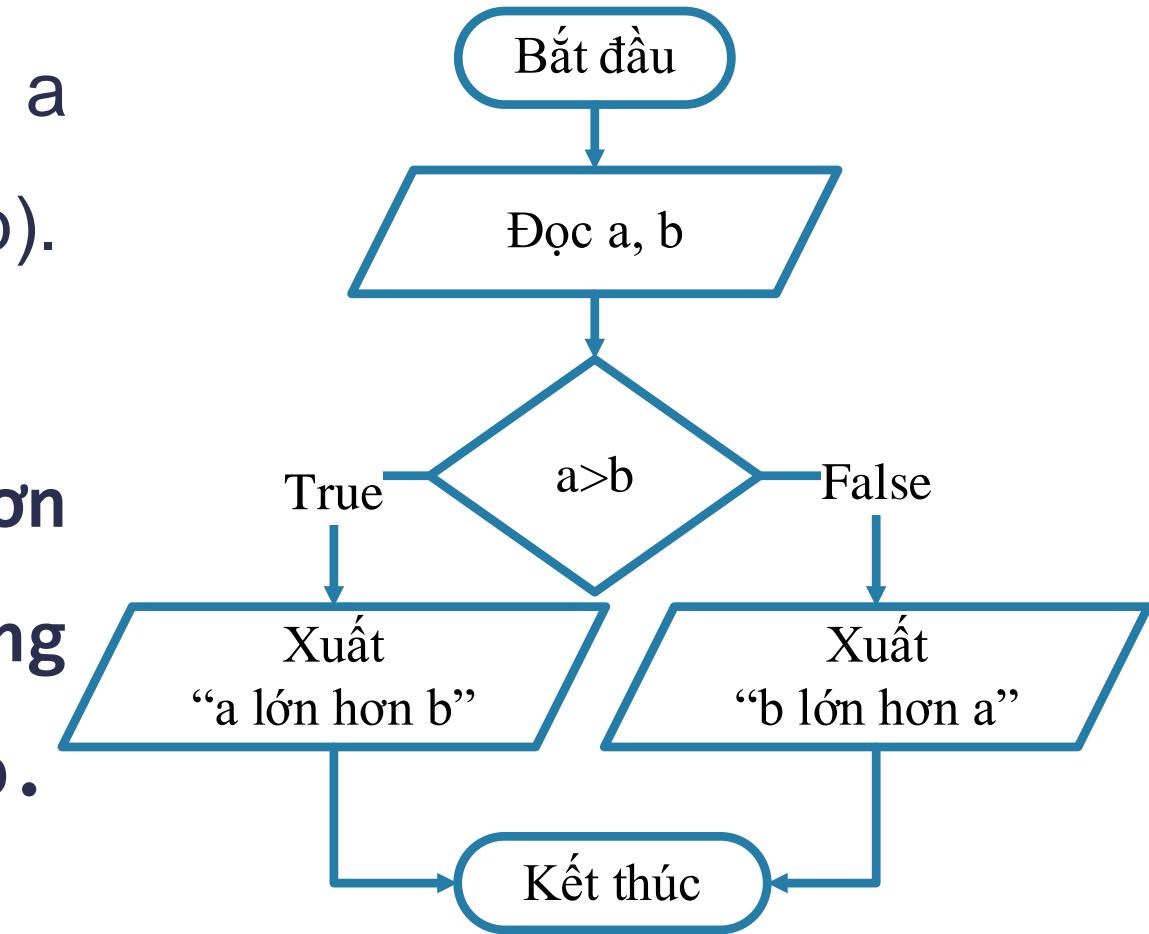
- **Đề bài:** Xuất tổng của 2 số tự nhiên a và b nhập từ bàn phím (không cần xét điều kiện nhập)
  - **Input:** Số tự nhiên a và b
  - **Output:** Kết quả tổng của a và b





## Ví dụ 2

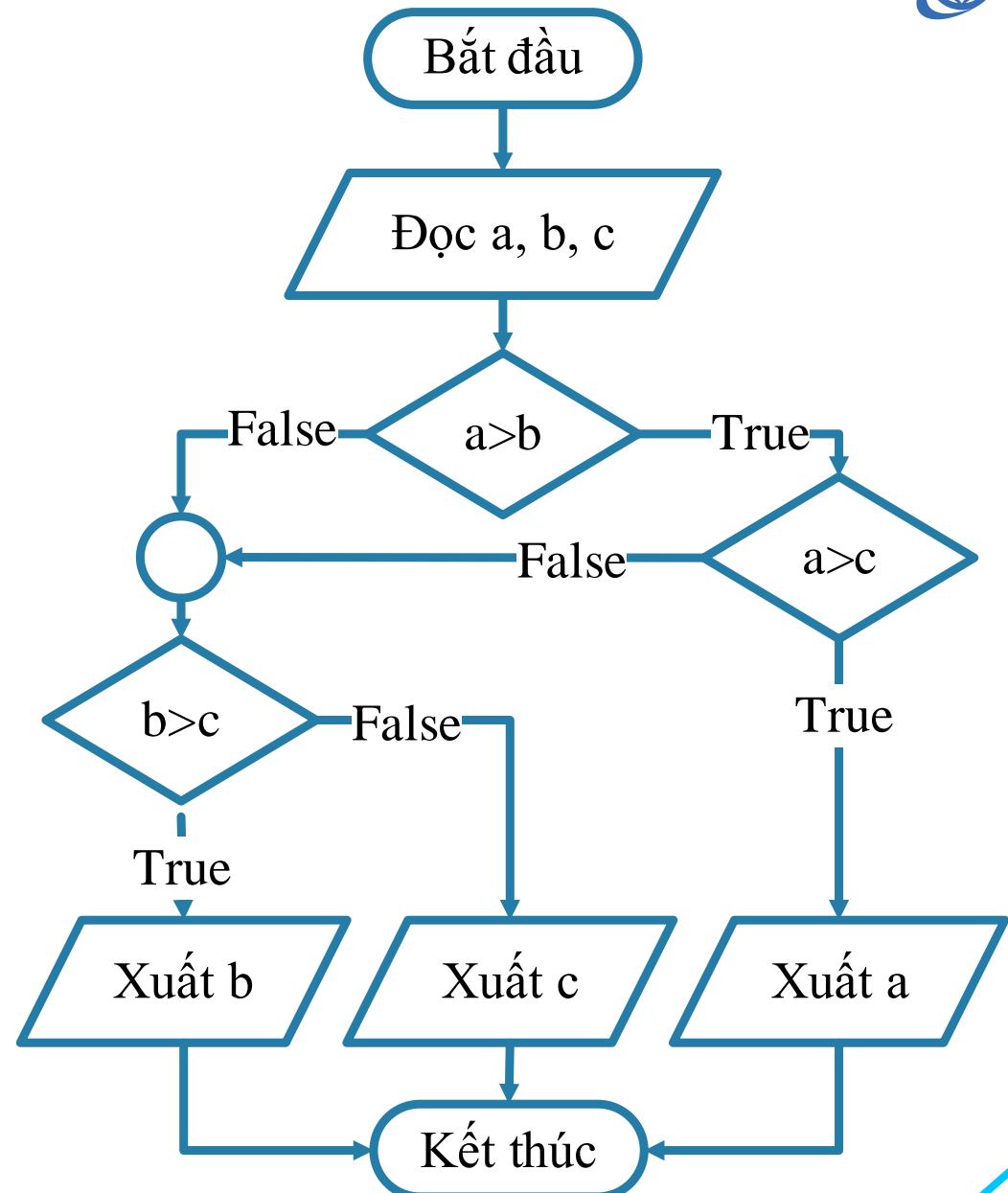
- **Đề bài:** Tìm số lớn hơn trong 2 số a và b (không cần xét điều kiện nhập).
- **Input:** Đọc 2 số a và b
- **Output:** Kết luận “a lớn hơn b” hoặc “b lớn hơn a” tương ứng với phép so sánh a và b.





## Ví dụ 3

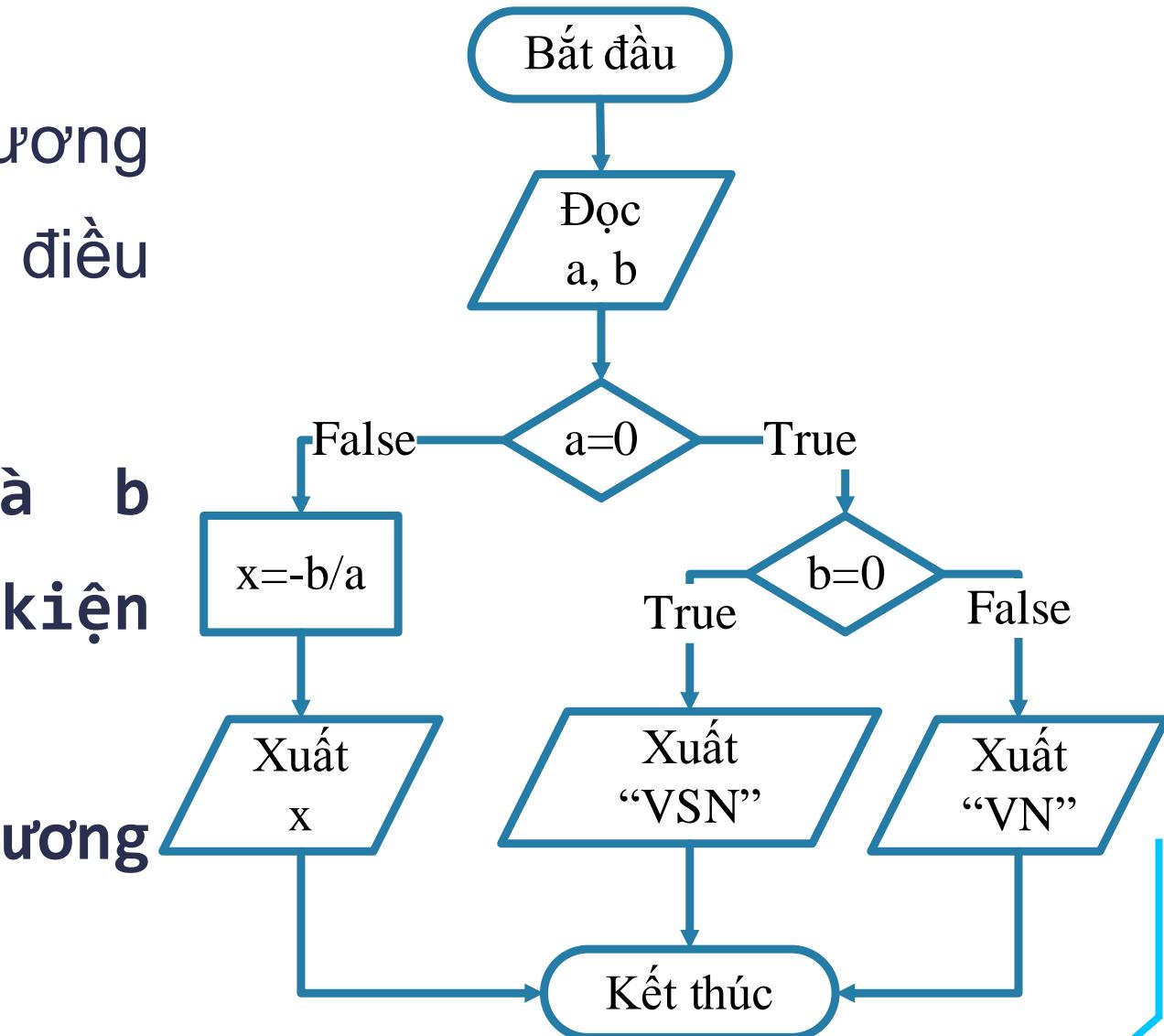
- **Đề bài:** Tìm số lớn nhất trong 3 số a, b và c (không cần xét điều kiện nhập).
- **Input:** Đọc 3 số a, c và c (không cần xét điều kiện nhập)
- **Output:** Xuất ra số lớn nhất trong 3 số





## Ví dụ 4

- **Đề bài:** Tính nghiệm của phương trình  $ax+b=0$  (không cần xét điều kiện nhập).
- **Input:** Đọc 2 số a và b (không cần xét điều kiện nhập)
- **Output:** Nghiệm của phương trình tương ứng





## 2.5 Các phương pháp biểu diễn thuật toán

### 2.5.3 Sử dụng Mã giả (Pseudo code)



## 2.5.3 Sử dụng Mã giả (Pseudo code)

- Vay mượn các cú pháp của ngôn ngữ lập trình (C, Pascal, ...) mà không cần tuân thủ nghiêm ngặt cú pháp của ngôn ngữ đó, có thể kết hợp dùng một phần ngôn ngữ tự nhiên.
- Dùng kết hợp với các ký hiệu toán học, biến, hàm, cú pháp trong ngôn ngữ lập trình, giúp người cài đặt dễ dàng nắm bắt nội dung thuật toán.
- **Ưu điểm:**
  - Dễ hiểu
  - Tập trung vào logic, ...
- **Nhược điểm:** Không trực quan bằng lưu đồ khối...



## 2.5.3 Sử dụng Mã giả (Pseudo code)

- Ví dụ: Tìm Ước số chung lớn nhất của 2 số nguyên dương a và b.
  - Đầu vào: Nhập 2 số nguyên dương a và b
  - Đầu ra: Xuất ra một số nguyên là ước chung lớn nhất của a và b

```
1 UCLN_CACH1(a, b)
2 BEGIN
3   WHILE a ≠ b DO
4     IF a > b THEN
5       a:= a - b
6     ELSE
7       b:= b - a
8     END IF
9   END WHILE
10  RETURN a
11 END
```

```
1 UCLN_CACH2(a, b)
2 BEGIN
3   WHILE b ≠ 0 DO
4     t:= b
5     b:= a MOD b
6     a:= t
7   END WHILE
8   RETURN a
9 END
```

```
1 FUNCTION UCLN_CACH3(a, b)
2 BEGIN
3   IF b = 0 THEN
4     RETURN a
5   ELSE
6     RETURN UCLN_CACH3(b, a%b)
7   END IF
8 END FUNCTION
9
10 BEGIN
11   INPUT a, b
12   result = UCLN_CACH3(a, b)
13   OUTPUT "UCLN LA: ", result
14 END
```



## 2.5.2 Sử dụng Mã giả (Pseudo code)

- Ví dụ: Tìm nghiệm của phương trình bậc nhất  $ax+b=0$ 
  - Đầu vào: Nhập  $a, b$  thuộc  $\mathbb{R}$
  - Đầu ra: Xuất kết quả Nghiệm phương trình  $ax + b = 0$

GiaiPT()

```
1      BEGIN
2          INPUT a, b
3          IF a = 0 THEN
4              IF b = 0 THEN
5                  OUTPUT "Phương trình vô số nghiệm"
6              ELSE
7                  OUTPUT "Phương trình vô nghiệm"
8              END IF
9          ELSE
10             x = -b / a
11             OUTPUT "Nghiệm của phương trình là x = ", x
12         END IF
13     END
```



## 2.5.3 Sử dụng Mã giả (Pseudo code)

- Đề bài: Bài toán tìm kiếm giá trị trong một dãy số được nhập từ bàn phím. (Cách 1)
- Đầu vào: Nhập dãy số  $A=\{a_0, a_1, \dots, a_{n-1}\}$  và giá trị  $x$  cần tìm.
- Đầu ra: Xuất chỉ số  $i$  tương ứng nếu tìm thấy  $x=A[i]$ , ngược lại kết luận không tìm thấy.

```
1 BEGIN
2     INPUT n // số lượng phần tử trong dãy
3     DECLARE array[n] // khai báo mảng chứa n phần tử
4     FOR i FROM 0 TO n-1 DO
5         INPUT array[i] // nhập từng phần tử vào mảng
6     END FOR
7     INPUT x // giá trị cần tìm kiếm
8     DECLARE found = FALSE
9     // biến kiểm tra trạng thái tìm kiếm, khởi tạo là FALSE
10    FOR i FROM 0 TO n-1 DO
11        IF array[i] = x THEN
12            found = TRUE
13            OUTPUT "Giá trị được tìm thấy tại vị trí ", i
14            BREAK // thoát vòng lặp nếu tìm thấy
15        END IF
16    END FOR
17    IF found = FALSE THEN
18        OUTPUT "Giá trị không được tìm thấy trong dãy"
19    END IF
20 END
```



## 2.5.3 Sử dụng Mã giả (Pseudo code)

Đề bài: Bài toán tìm kiếm giá trị trong một dãy số cho trước. (Cách 2)

- Đầu vào: Dãy  $A=\{a_0, a_1, \dots, a_{n-1}\}$  và giá trị  $x$
- Đầu ra: Xuất chỉ số i tương ứng  $x=A[i]$  hoặc NIL nếu không tìm thấy x.

**TimKiem2(A, x)**

- 1 Set  $A[n]$  to  $x$
- 2 Set  $i$  to 0
- 3 Repeat this loop:
  - 4     If  $A[i] = x$ ,  
               exit the loop
  - 5     Set  $i$  to  $i + 1$
  - 6 If  $i=n$  Return NIL
  - 7 Else Return  $i$



## 2.5 Các phương pháp biểu diễn thuật toán

### 2.5.4 Sử dụng ngôn ngữ lập trình



## 2.5.4 Sử dụng ngôn ngữ lập trình

- Dùng ngôn ngữ lập trình cụ thể để cài đặt thuật toán
- Chương trình viết bằng ngôn ngữ lập trình có thể thực thi được trên máy tính
- Kỹ năng lập trình đòi hỏi cần học tập và thực hành



## 2.5.4 Sử dụng ngôn ngữ lập trình

- Ví dụ: Tìm nghiệm của phương trình bậc nhất  $ax+b=0$ 
  - Đầu vào: Nhập các hệ số  $a$  và  $b$
  - Đầu ra: Xuất nghiệm của phương trình

Đoạn mã sau viết bằng ngôn ngữ C:

```
#include <stdio.h>
#include <conio.h>
int main() {
    int a, b;
    printf("Nhập a, b: ");
    scanf("%d%d", &a, &b);
    if (a == 0)
        if (b == 0) printf("Phương trình VSN");
        else printf("Phương trình VN");
    else
        printf("x = %.2f", -float(b)/a);
}
```



## 2.5.4 Sử dụng ngôn ngữ lập trình

- Ví dụ: Bài toán tìm kiếm giá trị trong một dãy số cho trước.
  - Đầu vào: Mảng 1 chiều  $A=\{a_0, a_1, \dots, a_{n-1}\}$  với size phần tử và giá trị  $x$  cần tìm
  - Đầu ra: Xuất chỉ số  $i$  tương ứng  $x=A[i]$  hoặc  $-1$  nếu không tìm thấy  $x$ .

Đoạn mã sau viết bằng ngôn ngữ C++:

```
int linearsearch(int A[], int size, int x) {  
    int index = 0;  
    while (index < size && A[index] != x)  
        index++;  
    if(index <= size) return index;  
    return -1;  
}
```

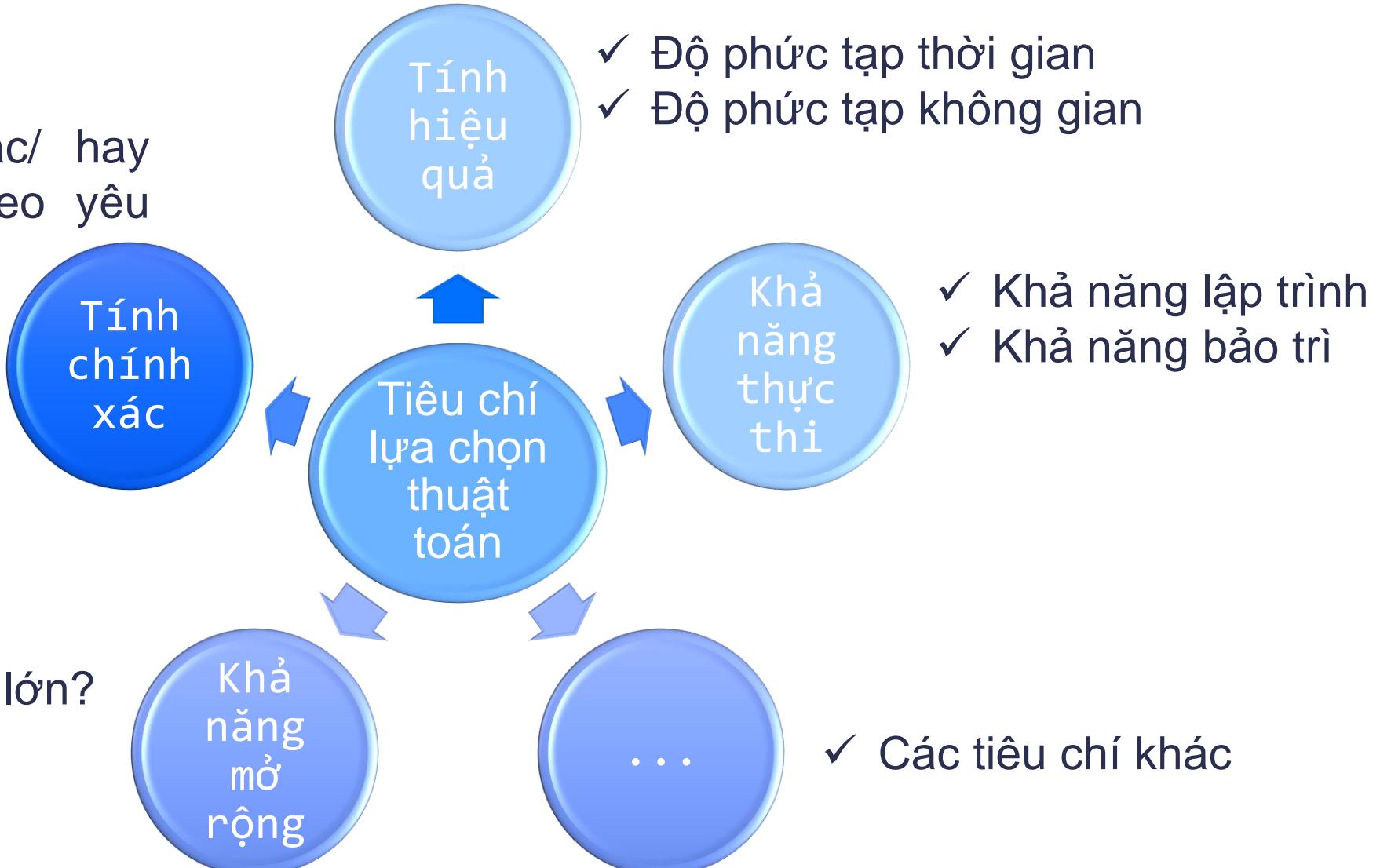


## 2.6 Độ phức tạp của thuật toán



# Tiêu chí lựa chọn thuật toán

- ✓ Độ chính xác/ hay gần đúng theo yêu cầu



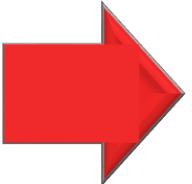


# Đánh giá thời gian thực hiện giải thuật

Không phụ thuộc cấu hình  
máy tính, ngôn ngữ lập  
trình, chương trình dịch



Không cần triển khai cài  
đặt và thực thi chương  
trình



Đánh giá giải thuật  
dựa trên mối liên quan  
giữa **dữ liệu đầu vào**  
và **chi phí các phép  
tính sơ cấp** cần thiết.

Chỉ dựa vào phân tích bản  
thân thuật toán



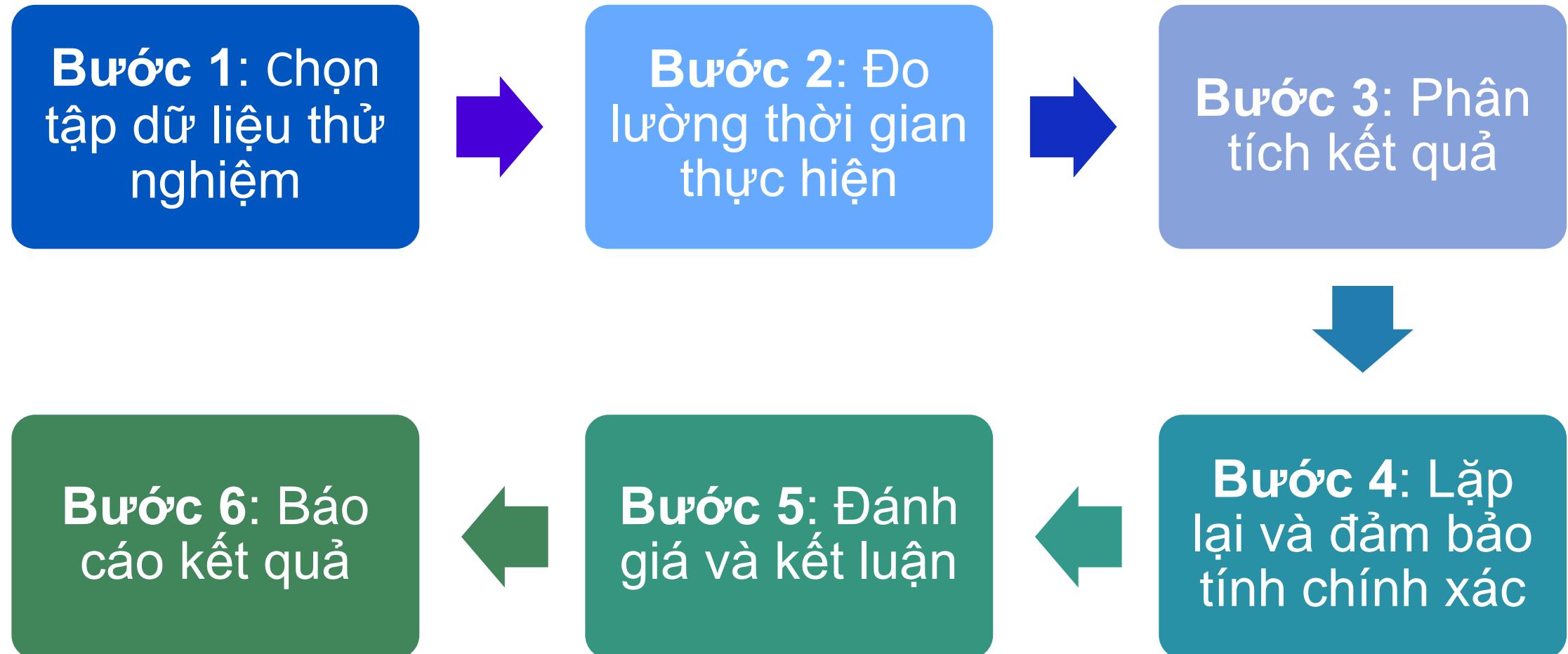
# Đánh giá thời gian thực hiện giải thuật

- Có 2 cách đánh giá Thời gian chạy của thuật toán:
  - **Phương pháp thực nghiệm:** Là một cách tiếp cận thực tế để xác định hiệu suất của thuật toán bằng cách chạy nó trên các tập dữ liệu cụ thể và đo lường thời gian thực hiện hoặc sử dụng tài nguyên.
  - **Phương pháp xấp xỉ toán học:** Đây là phương pháp sử dụng các công cụ toán học để ước lượng thời gian chạy của thuật toán.



# Phương pháp thực nghiệm

- Các bước đánh giá thuật toán bằng Phương pháp thực nghiệm:





# Phương pháp thực nghiệm

- **Ưu điểm:**

- Hiện thực và trực quan
- Không yêu cầu kiến thức toán học cao
- Khám phá các yếu tố ảnh hưởng (bộ nhớ, kiến trúc phần cứng, ...)
- Kiểm tra được khả năng thực thi của thuật toán

- **Nhược điểm:**

- Phụ thuộc vào ngôn ngữ lập trình và khả năng của người lập trình
- Phụ thuộc vào dữ liệu
- Tốn thời gian và tài nguyên
- Phụ thuộc vào cấu hình máy tính, các tiến trình đang chạy, ...
- Không cung cấp cái nhìn tổng quát về độ phức tạp lý thuyết của thuật toán
- ...



# Phương pháp xấp xỉ toán học

- Đánh giá thuật toán theo hướng tiệm xấp xỉ tiệm cận sử dụng khái niệm **BigO** (ký hiệu: **O**). Ký hiệu BigO được sử dụng để mô tả độ phức tạp của một thuật toán, nó cho thấy cách thời gian thực thi của thuật toán tăng lên khi kích thước đầu vào tăng.
- Độ phức tạp thời gian** của một thuật toán là cách đo lường thời gian mà thuật toán tiêu tốn để thực hiện một tác vụ. Đơn vị đo Độ phức tạp **không phải là đơn vị đo thời gian như giờ, phút, giây** mà thường được xác định bởi số lần thực hiện một phép toán cơ bản (ví dụ: so sánh, gán giá trị, truy cập dữ liệu) hoặc số lần lặp **tùy thuộc vào kích thước của đầu vào**.



# Phương pháp xấp xỉ toán học

- Thời gian thực hiện chương trình không chỉ phụ thuộc vào Kích thước dữ liệu mà còn phụ thuộc vào Tính chất dữ liệu.
- Các trường hợp phân tích độ phức tạp thuật toán:
  - Trường hợp tốt nhất (Best case)
  - Trường hợp xấu nhất (Worst case)
  - Trường hợp trung bình (Average case)
- Ví dụ: Thuật toán sắp xếp Insertion sort
  - Trường hợp tốt nhất:  $O(n)$  (khi mảng đã có thứ tự như yêu cầu)
  - Trường hợp trung bình:  $O(n^2)$
  - Trường hợp xấu nhất:  $O(n^2)$  (khi mảng có thứ tự ngược với yêu cầu).



# Phương pháp xấp xỉ toán học

- Một số phân lớp độ phức tạp của thuật toán như bên dưới đây: (được sắp xếp tăng dần theo độ phức tạp)

Độ phức tạp	Ký hiệu
Độ phức tạp hằng số	$O(C)$
Độ phức tạp logarit	$O(\log n)$
Độ phức tạp tuyến tính	$O(N)$
Độ phức tạp $n \log n$	$O(n \log n)$
Độ phức tạp bậc 2	$O(n^2)$
Độ phức tạp bậc 3	$O(n^3)$
Độ phức tạp lũy thừa	$O(2^n)$
Độ phức tạp giai thừa	$O(n!)$



# Phương pháp xấp xỉ toán học

- **Ví dụ:** Thuật toán tìm tổng của một mảng số nguyên:

- **Mã giả:**

```
1  SUM_ARRAY(arr, n)
2      sum = 0
3      FOR i IN arr
4          sum = sum + i
5      RETURN sum
```

- **Phân tích:**

- Dòng 2: **sum = 0** thực hiện 1 lần
- Dòng 3: Vòng lặp **FOR i IN arr** thực hiện **n** lần
- Dòng 4: **sum = sum + i** thực hiện **n** lần
- Dòng 5: **RETURN sum** thực hiện 1 lần

- **Độ phức tạp thời gian:**

Tổng số bước =  $1 + n + n + 1 = 2n + 2$ . Khi  $n$  tăng, các hằng số không đáng kể, do đó độ phức tạp thời gian là  $O(n)$ .



# Phương pháp xấp xỉ toán học

- **Ưu điểm:**
  - Ít phụ thuộc vào phần cứng, NNLT và khả năng người lập trình
  - Khả năng dự đoán hiệu suất của thuật toán mà không cần chạy thực nghiệm
  - Dễ dàng so sánh độ hiệu quả so với thuật toán khác một cách nhanh chóng
- **Nhược điểm:**
  - Cần có kiến thức về toán học
  - Không phản ánh điều kiện thực thi thực tế, hoặc tối ưu hóa phần cứng



## 2.7 Một số ví dụ về thuật toán



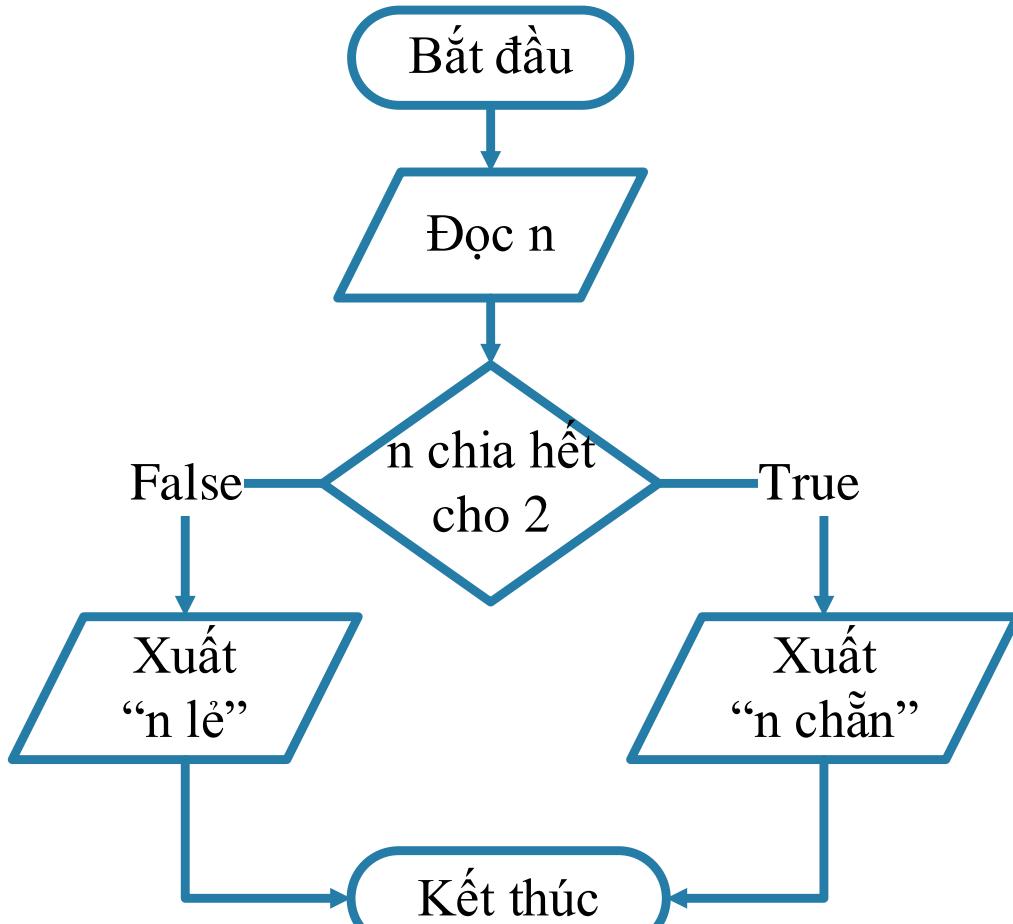
## 2.7 Một số ví dụ về thuật toán

- Ví dụ 1: Vẽ lưu đồ thuật toán Kiểm tra tính chẵn lẻ của một số nguyên.
- Ví dụ 2: Vẽ lưu đồ thuật toán Tính tổng các số nguyên dương lẻ từ 1 đến n.
- Ví dụ 3: Vẽ lưu đồ thuật toán Tìm nghiệm của phương trình bậc hai một ẩn.
- Ví dụ 4: Vẽ lưu đồ thuật toán kiểm tra một số được tạo từ các chữ số chẵn hay không?

# Ví dụ 1: Kiểm tra tính chẵn lẻ của một số nguyên



- **Input:** một số nguyên  $n$
- **Output:** Xuất kết quả là tính chẵn lẻ của số nguyên  $n$

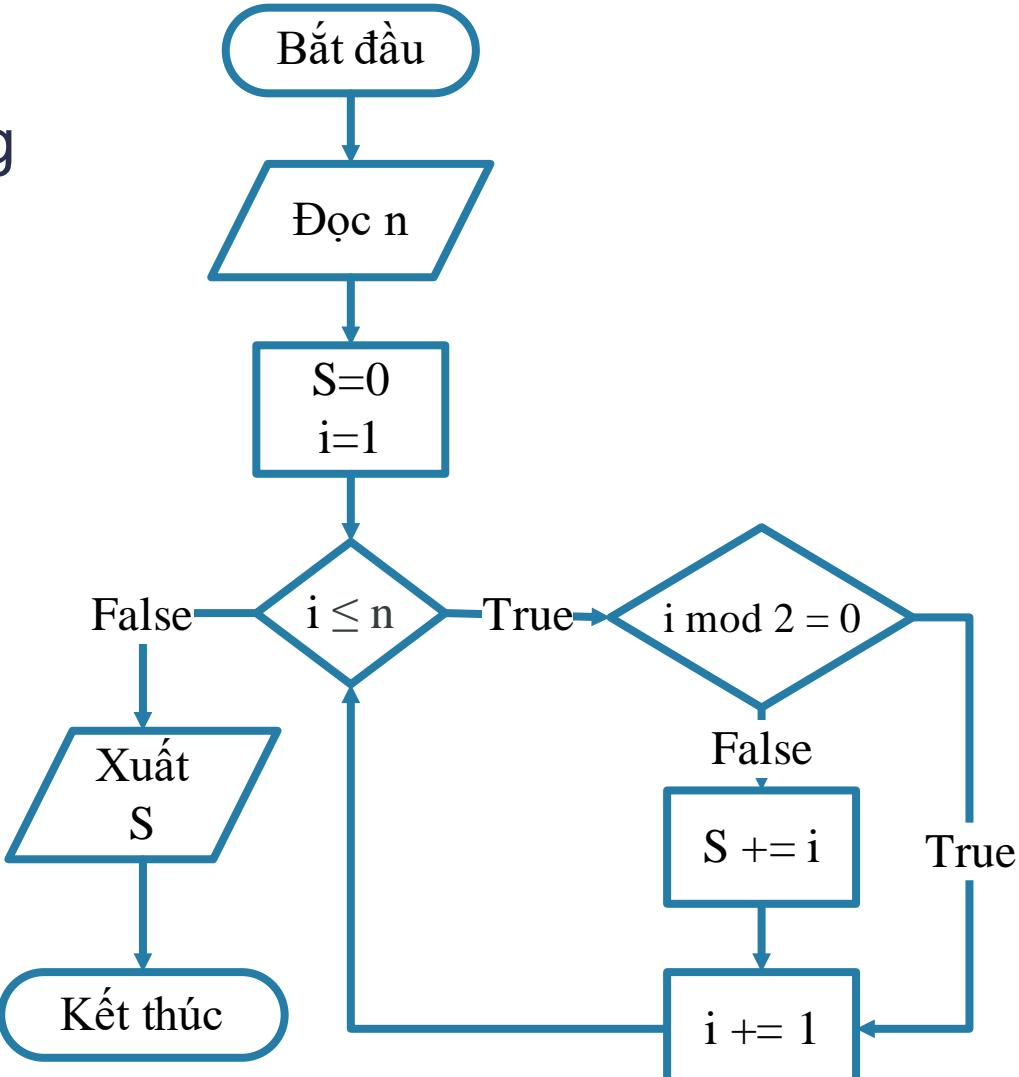




## Ví dụ 2: Tính tổng các số nguyên dương lẻ từ 1 → n

- **Input:** Một số nguyên n
- **Output:** Số nguyên dương tổng theo yêu cầu

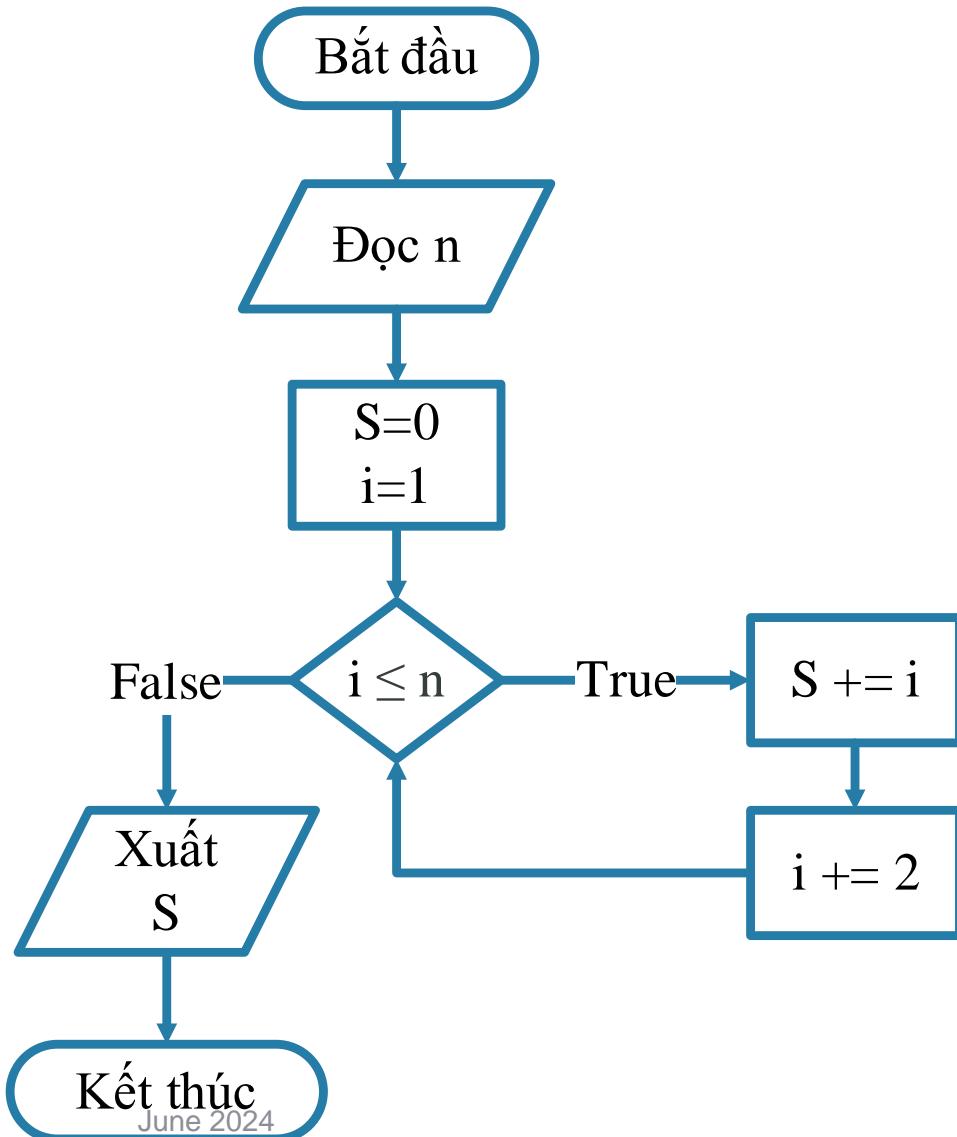
\* Cách giải 1:  $i = i + 1$





## Ví dụ 2: Tính tổng các số nguyên dương lẻ từ 1 → n

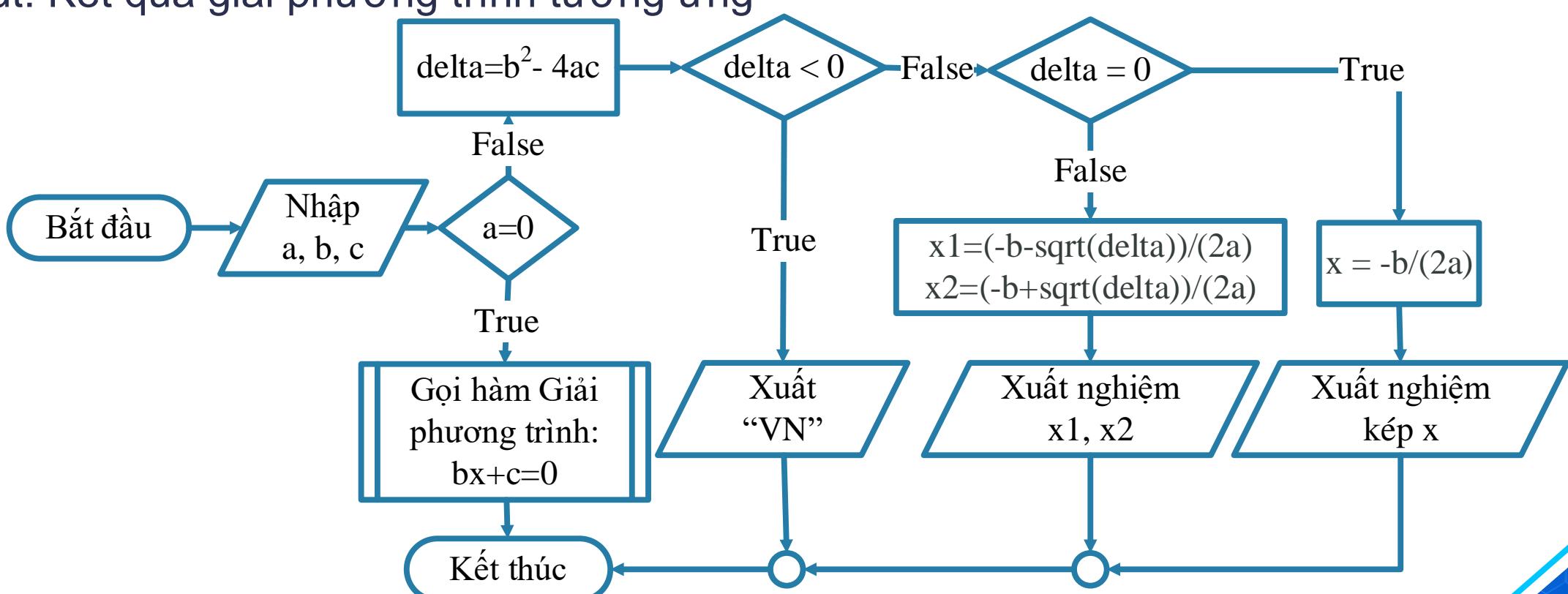
- **Input:** Một số nguyên n
  - **Output:** Số nguyên dương tổng  
theo yêu cầu
- \* Cách giải 2:  $i = i + 2$





# Ví dụ 3: Tìm nghiệm của phương trình bậc hai một ẩn

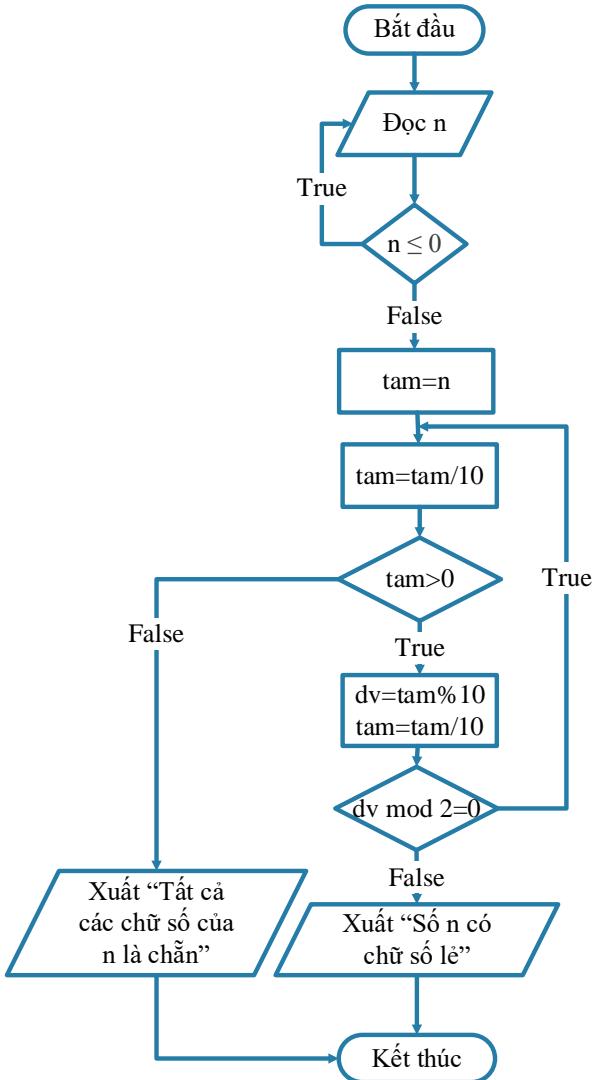
- Giải pt:  $ax^2+bx+c=0$
- Input: 3 số thực a, b, c
- Output: Kết quả giải phương trình tương ứng





# Ví dụ 4: Vẽ lưu đồ thuật toán kiểm tra một số được tạo từ các chữ số chẵn hay không?

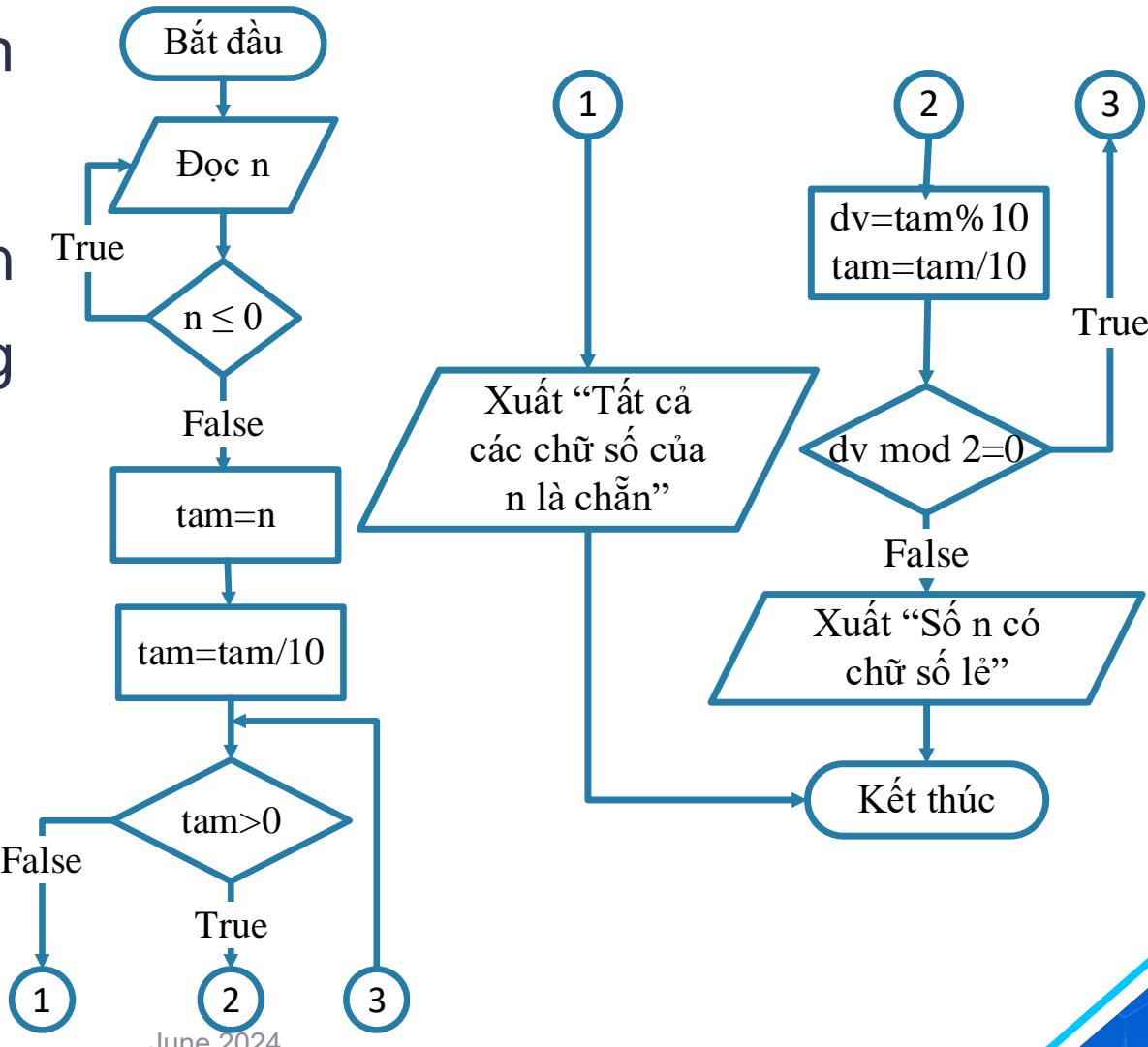
- **Input:** Số nguyên dương n (nhập đến khi thỏa điều kiện)
- **Output:** In ra “Tất cả các chữ số của n là chẵn” hoặc “Số n có chữ số lẻ” tương ứng với kết quả kiểm tra





**Ví dụ 4: Vẽ lưu đồ thuật toán kiểm tra một số nguyên dương được tạo từ các chữ số chẵn hay không?**

- **Input:** Số nguyên dương n (nhập đến khi thỏa điều kiện)
  - **Output:** In ra “Tất cả các chữ số của n là chẵn” hoặc “Số n có chữ số lẻ” tương ứng với kết quả kiểm tra





# Bài tập



# Bài tập lý thuyết

Câu 1: Thuật toán là gì? Trình bày các tính chất quan trọng của một thuật toán?

Câu 2: Các bước xây dựng chương trình?

Câu 3: Các cách biểu diễn thuật toán? Ưu và khuyết điểm của từng phương pháp? Cho ví dụ minh họa.

Câu 4: Độ phức tạp của thuật toán là gì? Sắp xếp các độ phức tạp của thuật toán được cho sau đây theo thứ tự hiệu quả giảm dần:  $O(N)$ ,  $O(N^3)$ ,  $O(\log_2 N)$ ,  $O(1)$ ,  $O(N!)$ ,  $O(N \log_2 N)$ ,  $O(2^N)$ ,  $O(N^2)$



# Bài tập thực hành

Vẽ lưu đồ thuật toán cho các bài toán sau:

1. Viết chương trình nhập và xuất ra tên của chính bạn.
2. Viết chương trình nhập vào một số nguyên. In ra số đối xứng. VD: Nhập 1, in ra -1; nhập -2, in ra 2.
3. Viết chương trình chuyển độ F sang độ C và ngược lại, biết công thức chuyển là:  
$$^{\circ}\text{F} = ( ^{\circ}\text{C} \times 1.8 ) + 32$$
$$^{\circ}\text{C} = ( ^{\circ}\text{F} - 32 ) / 1.8$$

4. Liệt kê các số chẵn từ 1 đến n.
5. Nhập vào 2 số nguyên a, b. Tính tổng hiệu tích thương.
6. Viết chương trình nhập vào ba số nguyên. Tìm số lớn nhất và số nhỏ nhất.



# Bài tập thực hành

6. Tìm ước số lớn nhất và nhỏ nhất của một số nguyên dương n mà không phải là chính số đó và số 1.
7. Tính giá trị biểu thức:

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

$$S = 1 + 2^2 + 3^3 + \dots + n^n$$

8. Liệt kê tất cả ước số của số nguyên dương n.
9. Nhập vào số nguyên dương n. Đếm số chữ số của số đó.
10. Nhập vào số xe (gồm 4 chữ số) của bạn. Cho biết số xe của bạn được mấy nút?



# Chúc các em học tốt !

