

# *Kỹ thuật lập trình*

- Học phần này nhằm cung cấp cho sinh viên các kiến thức căn bản về lập trình theo phương pháp cấu trúc (chia module).
- Để có thể nắm bắt các kiến thức trình bày trong học phần này, sinh viên cần có kiến thức về tin học đại cương.
- Ngôn ngữ lập trình được chọn để minh họa các kiến thức trên là C++.
- Các kiến thức này sẽ tạo điều kiện cho học viên tiếp tục dễ dàng nắm bắt các kiến thức các học phần tin học về sau như: cấu trúc dữ liệu và giải thuật, lập trình hướng đối tượng, phân tích và thiết kế thuật toán, đồ họa, hệ điều hành, trí tuệ nhân tạo, ...
- **3LT (45 tiết) + 2 TH (60 tiết)**

# Nội dung: gồm 4 chương

1. Trình bày các khái niệm cơ bản trong lập trình: chương trình, dữ liệu, thuật toán, ... và cách thức biểu diễn dữ liệu và thuật toán trong một chương trình máy tính.
2. Giới thiệu ngôn ngữ C++ và cách viết các chương trình đơn giản với các kiểu dữ liệu cơ sở trên C++.
3. Trình bày phương pháp lập trình module với hàm, kỹ thuật đệ quy.
4. Giới thiệu các kiểu dữ liệu có cấu trúc như mảng, chuỗi, struct và các thuật toán thường gặp trên chúng.

# TÀI LIỆU THAM KHẢO

- [1] Hoàng Kiếm, *Giải bài toán trên máy tính như thế nào*, Tập 1, Nhà xuất bản Giáo dục, 2001.
- [2] Nguyễn Tiến Huy, Trần Hạnh Nhi, *Giáo trình Kỹ thuật lập trình*, Trường Đại học Khoa học Tự nhiên Tp.Hồ Chí Minh, 1997.
- [3] Trần Tuấn Minh, *Giáo trình Nguyên lý lập trình*, Khoa CNTTin, Đại học Đà Lạt.
- [4] Võ Tiến, *Giáo trình Nhập môn lập trình*, Đại học Đà Lạt, 1997.
- [5] **Giáo trình Kỹ thuật lập trình – Trần Ngọc Anh, Khoa Toán – Tin học, Đại học Đà Lạt, 2016**

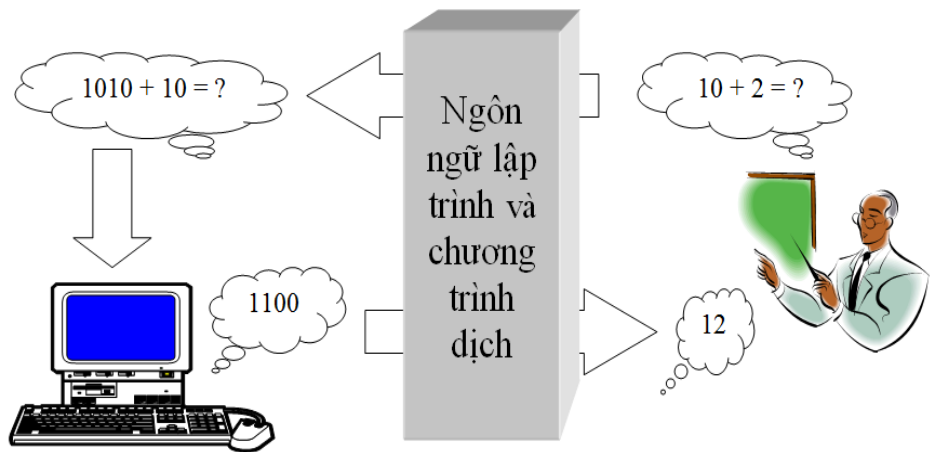
# Chương 1: DỮ LIỆU VÀ THUẬT TOÁN

- 1.1 Các khái niệm cơ bản
- 1.2 Thuật toán
  - 1.2.1 Định nghĩa
  - 1.2.2 Các đặc trưng của thuật toán
  - 1.2.3 Các ngôn ngữ biểu diễn thuật toán
- 1.3 Biểu diễn dữ liệu
  - 1.3.1 Biến
  - 1.3.2 Hằng
- 1.4 Biểu diễn thuật toán
  - 1.4.1 Cấu trúc tuần tự
  - 1.4.2 Cấu trúc rẽ nhánh
  - 1.4.3 Cấu trúc lặp
- BÀI TẬP

# 1.1. Các khái niệm cơ bản

- Trên thực tế có nhiều bài toán có phương pháp giải quyết đơn giản nhưng quá trình thực hiện lại cồng kềnh lặp đi lặp lại nhiều lần.
- Việc lặp lại các bước giải làm cho người giải nhàm chán, mệt mỏi dẫn đến mất tập trung gây ra sai sót.
- So với con người, máy tính có khả năng tính toán một khối lượng lớn các phép toán với độ chính xác cao trong khoảng thời gian cực ngắn (và không biết mệt).
- Do đó, từ lúc ra đời (khoảng năm 1940), máy tính đã trở thành một công cụ đa năng trong việc giải quyết các bài toán thuộc nhiều lĩnh vực: quân sự, quản lý, khoa học, tư vấn, thương mại, ...

# 1.1. Các khái niệm cơ bản (tt)



- **Chương trình (CT)** là một tập các mô tả, các câu lệnh được viết theo một trình tự nhất định bởi **lập trình viên (LTV)** nhằm hướng dẫn máy tính giải một bài toán đặt ra.
- Trong các thế hệ máy tính đầu tiên, LTV phải viết trực tiếp CT bằng ngôn ngữ máy. Công việc này mất nhiều thời gian vì ngôn ngữ máy rất khó sử dụng. Đến giai đoạn sau, các **ngôn ngữ lập trình (NNLT)** ra đời và thay thế dần ngôn ngữ máy.
- NNLT là hệ thống hữu hạn các ký hiệu, quy ước về ngữ pháp (quan hệ giữa các ký hiệu) và ngữ nghĩa (ý nghĩa của các ký hiệu) dùng để xây dựng các CT.
- LTV viết CT trên một NNLT chọn trước, sau đó sử dụng chương trình dịch để dịch và thực thi CT.

# 1.1. Các khái niệm cơ bản (tt)

- Có hai loại chương trình dịch:
  - **thông dịch**: dịch *từng câu lệnh* của CT ra ngôn ngữ máy và thực thi cho đến khi kết thúc CT;
  - **biên dịch**: dịch *toàn bộ* CT sang ngôn ngữ máy và thực thi CT.
- Mỗi NNLT phù hợp với các loại bài toán khác nhau. Các NNLT được phân loại theo độ “gần” với ngôn ngữ máy (hay theo mức độ trừu tượng).
  - Các NNLT “gần” với máy được gọi là các **NNLT bậc thấp**.
  - Các NNLT “xa” với máy được gọi là các **NNLT bậc cao**.
  - Viết CT trên các NNLT bậc cao sẽ tự nhiên hơn trên các NNLT bậc thấp.
- Các NNLT bậc cao được sử dụng rộng rãi trong thực tế là:
  - Pascal, Basic, dòng C, Java, Python.
  - **C** được phát triển (bởi Dennis Ritchie tại phòng thí nghiệm Bell Telephone vào năm 1972) từ **B** (do Ken Thomson tạo ra).
  - C++/C# được nâng cấp từ C, hỗ trợ thêm phương pháp lập trình hướng đối tượng. Hiện nay, C++/C# được sử dụng rộng rãi.

# 1.2. Thuật toán - 1.2.1. Định nghĩa

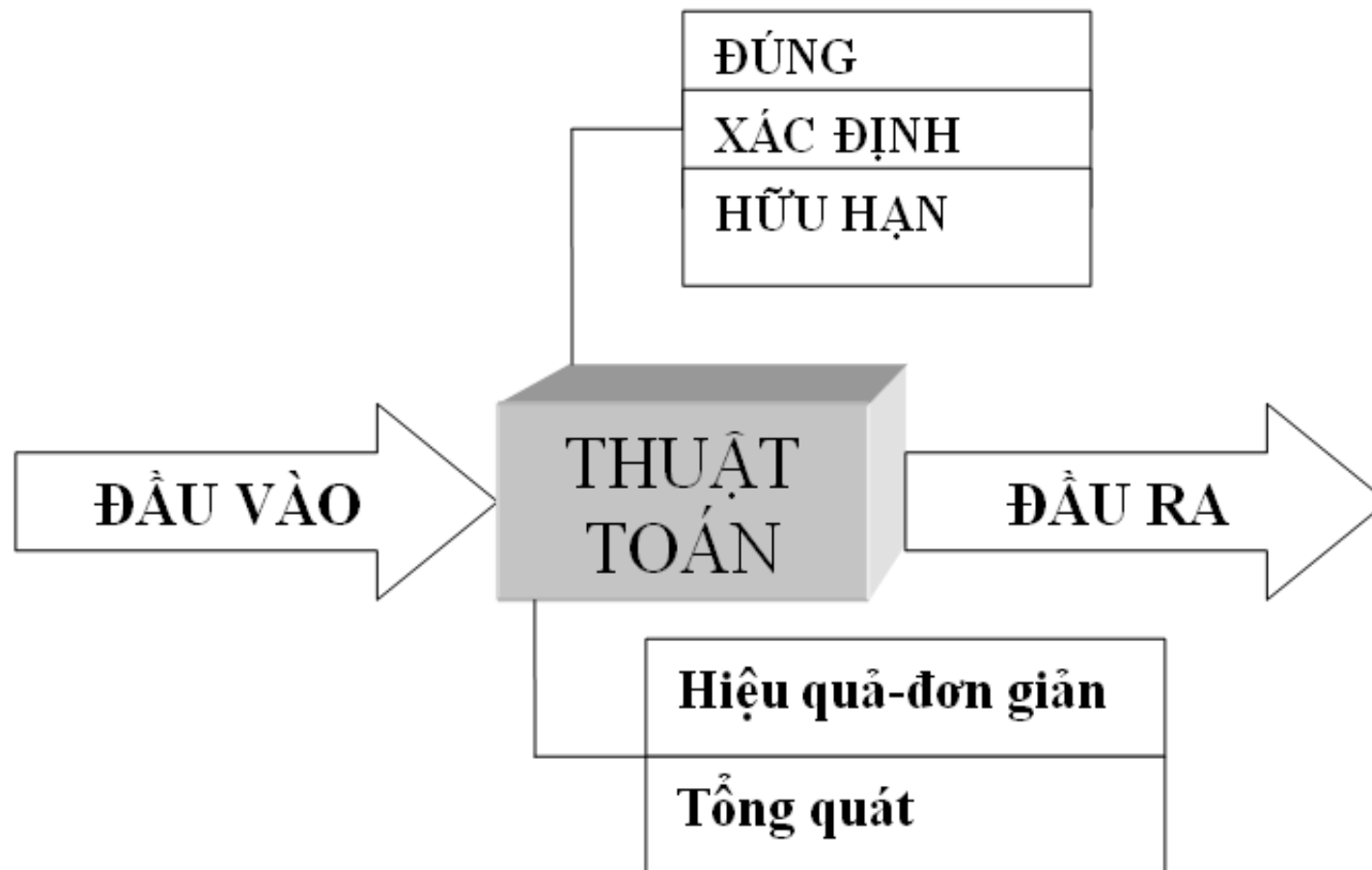
- Để biểu diễn lời giải cho một bài toán (dù là rất đơn giản) trên máy tính, ta phải làm rõ tất cả mọi chi tiết của cách giải, hướng dẫn cụ thể từng bước một thì máy tính mới có thể thi hành được. Cách biểu diễn lời giải bài toán một cách rõ ràng, chi tiết, có thể thi hành được trên máy tính được gọi là thuật toán.
- Thuật toán là một khái niệm cơ bản của Toán học và Tin học được dịch từ Algorithm, xuất phát từ một nhà Toán học Trung Á, sống ở khoảng thế kỷ IX. Vào thời kỳ này các nhà Toán học thường viết những sách dạy Toán, trong đó trình bày cách giải mà theo đó người ta có thể giải được các bài toán nhất định trong cuộc sống như: đo diện tích một mảnh đất, đo thể tích của một vật, hay đo gián tiếp khoảng cách giữa hai điểm.
- Trong các trường phổ thông, học sinh được hướng dẫn cách giải phương trình bậc nhất, bậc hai, ....
- Trong đời sống, ta thường gặp các khái niệm như: hướng dẫn cách nấu ăn, chương trình của một đại hội, cách vận hành một máy, ... Các khái niệm, các hướng dẫn đó rất gần với khái niệm thuật toán.



# 1.2.1. Định nghĩa

- Thuật toán theo nghĩa chính xác được định nghĩa bằng mô hình máy Turing, mô hình chuẩn Mabbob. Trên cơ sở các mô hình đó, người ta có thể xác định được các bài toán có thể giải được bằng thuật toán, phân lớp được các bài toán theo độ khó, .... Ở đây, ta chọn định nghĩa thuật toán theo nghĩa trực quan.
- **Thuật toán là một dãy hữu hạn các bước xác định (rõ ràng, không mập mờ, và thực thi được) để giải đúng (kết quả mong muốn) một bài toán.**
- Giả sử một người A được yêu cầu nấu một nồi chè bằng quy trình sau:
  - a) Rửa đậu, bắc lửa, đổ nước, đường vào nồi và chờ cho đến lúc sôi.
  - b) Nếm thử:
    - Nếu quá ngọt thì thêm nước.
    - Nếu quá nhạt thì thêm đường.
    - Nếu vừa thì đến bước c).
  - c) Chờ cho đến lúc vừa cạn nước thì: tắt lửa và bắc nồi xuống.
- Đây là một quy trình mập mờ, do đó A sẽ đặt ra rất nhiều câu hỏi (lượng đậu bao nhiêu, bắc lửa như thế nào, như thế nào là quá ngọt, làm sao biết là vừa cạn nước, ...) mà nếu không được trả lời thì A sẽ không thể nấu được một nồi chè tốt.

## 1.2.3. Các đặc trưng của thuật toán



Hình 1.3: Các tính chất và đặc trưng của thuật toán,

## 1.2.3. Các đặc trưng của thuật toán

- **Tính thực thi được** là một tính chất quan trọng. Chẳng hạn, A đã chỉ cho B cách tính nghiệm của phương trình bậc 2:

$$x_1 = -b + \sqrt{\Delta}/(2a), x_2 = -b - \sqrt{\Delta}/(2a).$$

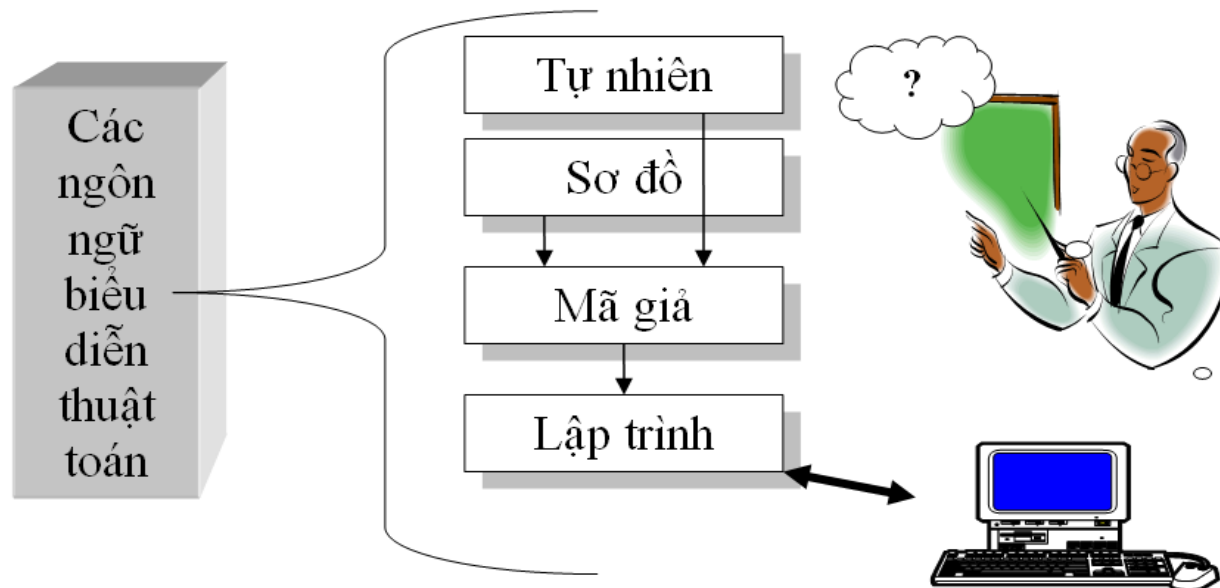
- Không phải lúc nào B cũng thực hiện được vì chỉ có thể lấy căn bậc hai của các số thực không âm.
- Nếu B tính phần sau trước rồi mới  $+/ -$  cho  $b$ , thì sẽ nhận được kết quả sai.
- **Tính đúng** là tính chất mà chúng ta đều muốn đạt nhưng không phải lúc nào cũng đạt được (phải tiến hành các chứng minh phức tạp). Trong thực tế, cần phải chạy thử nghiệm thuật toán trên nhiều bộ dữ liệu thử khác nhau để tăng độ tin cậy vào tính đúng của thuật toán.
- **Tính hữu hạn** là tính chất dễ bị vi phạm. Chẳng hạn, quy trình tính  $S$  sau không hữu hạn: a) cho  $S$  bằng 0, b) cộng  $S$  với 3, c) trừ  $S$  cho 3, d) nếu  $S > 1$  thì cho giá trị của  $S$ , ngược lại quay lại bước (b).

## 1.2.3. Các đặc trưng của thuật toán

- Bên cạnh ba tính chất cơ bản là xác định, hữu hạn và đúng, thuật toán còn có các đặc trưng khác:
  - Nhận dữ liệu *đầu vào*, tính toán và in ra kết quả (*đầu ra*).
  - Có thể có nhiều thuật toán giải đúng cùng một bài toán. Các thuật toán *tốn ít thời gian, không gian* (bộ nhớ) tính toán, và *đơn giản, dễ hiểu* sẽ được áp dụng.
  - Thuật toán phải áp dụng được trong nhiều trường hợp (tính *tổng quát*) chứ không phải chỉ áp dụng cho một số trường hợp riêng lẻ.
- Trong thực tế, khi thiết kế thuật toán thường người ta chỉ quan tâm đến các tính chất: đúng, xác định, hữu hạn.
  - Tính hiệu quả-đơn giản thường rất khó đạt được trọn vẹn: rất khó tìm được một thuật toán *tốn ít thời gian, không gian* tính toán mà lại *đơn giản và dễ hiểu*.
  - Tính tổng quát cũng khó đạt được khi gặp các bài toán phức tạp, đối với các bài toán này người ta sẽ xây dựng nhiều thuật toán, mỗi thuật toán giải một số trường hợp.

## 1.2.3. Các ngôn ngữ biểu diễn thuật toán

- Khi đã có thuật toán, ta có nhu cầu truyền đạt lại cho người khác cũng như thể hiện thành chương trình để máy tính thực thi. Phương tiện tự nhiên nhất để truyền đạt thuật toán là ngôn ngữ.
- Các loại ngôn ngữ thường được sử dụng là: ngôn ngữ tự nhiên, ngôn ngữ sơ đồ, ngôn ngữ mã giả, NNLT. Thực tế, chúng thường được kết hợp với nhau.

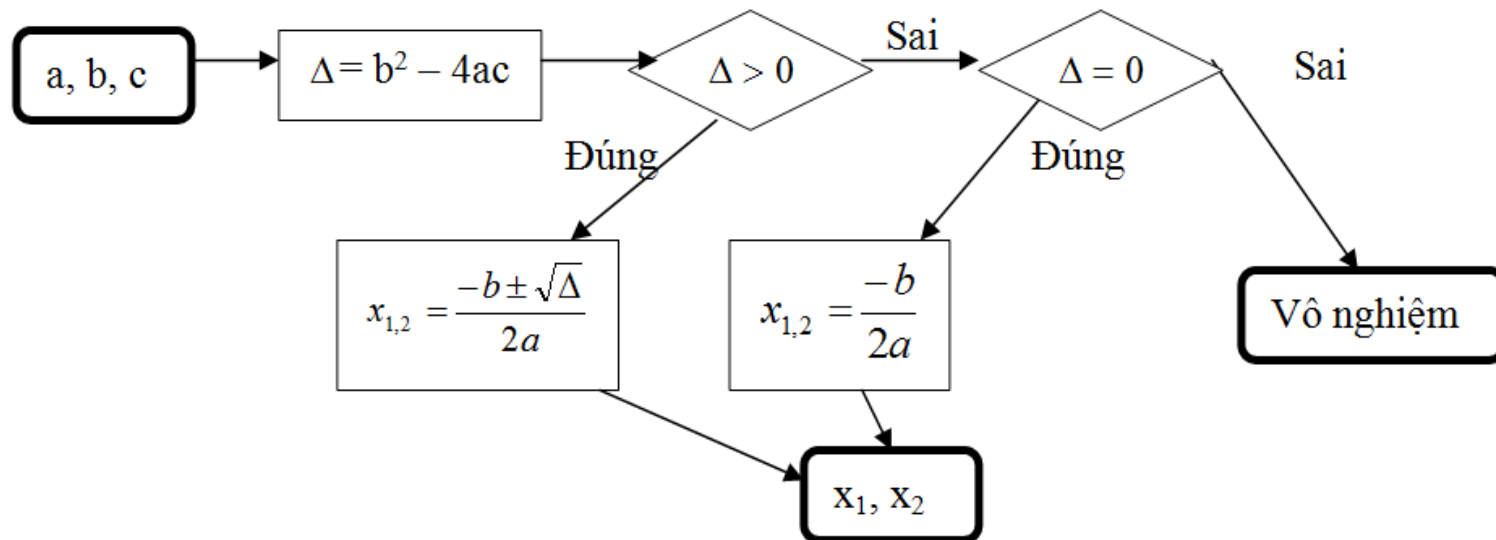


Hình 1.4: Các ngôn ngữ biểu diễn thuật toán

## 1.2.3. Các ngôn ngữ biểu diễn thuật toán

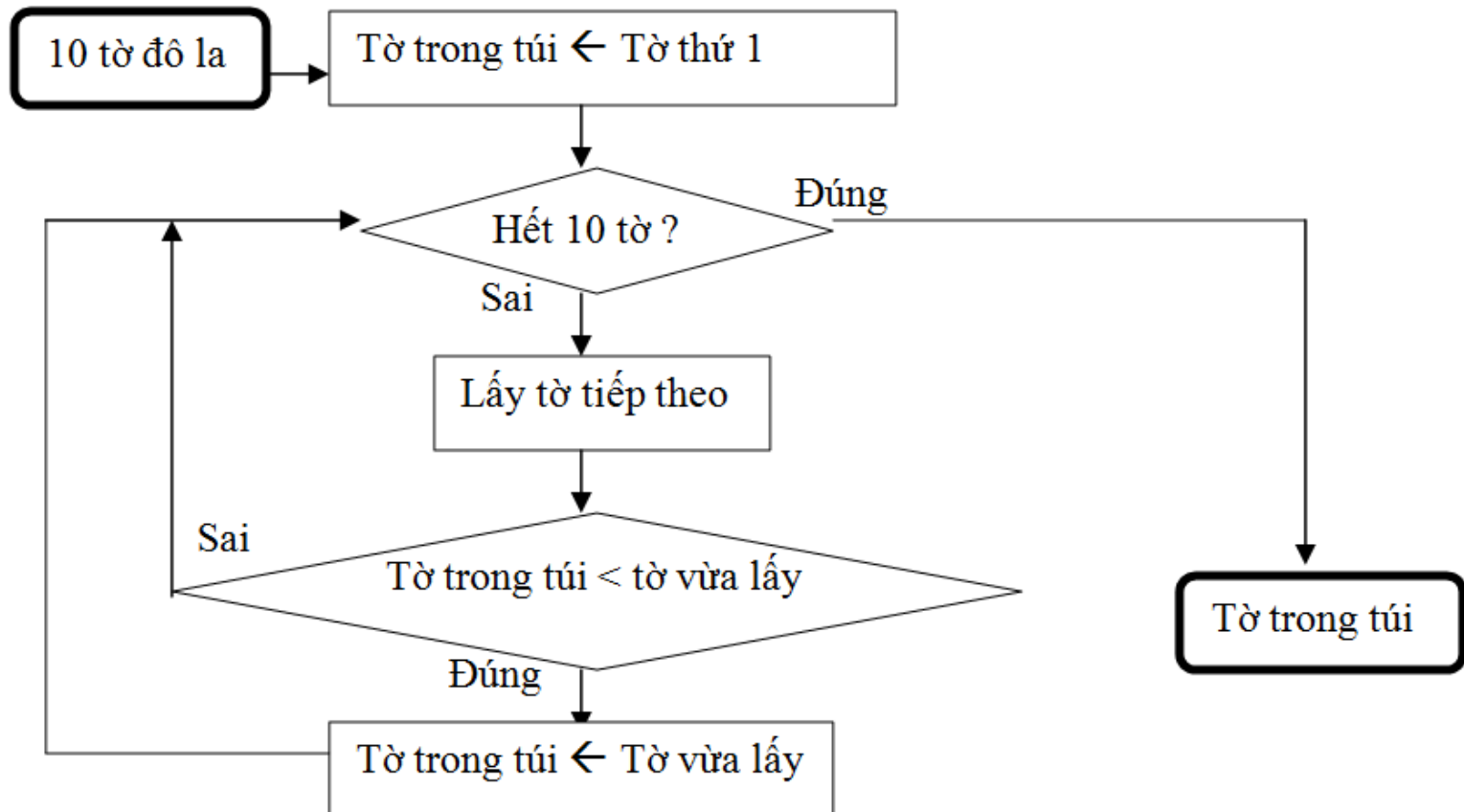
- **Ngôn ngữ tự nhiên** là ngôn ngữ diễn đạt của con người. Dùng ngôn ngữ tự nhiên biểu diễn thuật toán sẽ rất dài dòng, không thể hiện rõ cấu trúc thuật toán, đôi lúc còn gây hiểu lầm.
- **Ngôn ngữ sơ đồ** là công cụ trực quan để diễn đạt thuật toán, bao gồm tập các hình vẽ và các mũi tên với các quy ước: a) *hình thoi* bên trong chứa điều kiện, b) *hình chữ nhật* chứa nội dung xử lý, tính toán, c) *mũi tên* chỉ trình tự thực hiện, e) *hình oval* chỉ ra đầu vào và đầu ra, ...

Ví dụ 1.5: Thuật toán giải phương trình bậc hai ( $a \neq 0$ ).




## 1.2.3. Các ngôn ngữ biểu diễn thuật toán

Ví dụ 1.4: Thuật toán chọn tờ đô la có giá trị nhất trong 10 tờ đô la.



Tuy trực quan, nhưng các thuật toán biểu diễn bằng ngôn ngữ sơ đồ chiếm dụng không gian lớn, rất cồng kềnh.

## 1.2.3. Các ngôn ngữ biểu diễn thuật toán

- **Ngôn ngữ lập trình** là hệ thống các ký hiệu tuân theo các quy ước chặt chẽ về cú pháp và ngữ nghĩa, dùng để xây dựng các CT.
- **Ngôn ngữ mã giả** là ngôn ngữ vay mượn NNLT và ngôn ngữ tự nhiên. Dùng mã giả vừa tận dụng được các khái niệm trong NNLT vừa giúp LTV dễ dàng nắm bắt nội dung thuật toán.
-  *Đối với các bài toán đã có sẵn thuật toán giải, việc giải bài toán trên máy tính đơn thuần chỉ là biểu diễn dữ liệu của bài toán và thuật toán giải dưới dạng một NNLT nào đó.*
- *Tuy nhiên, quá trình này không phải lúc nào cũng dễ dàng. Nếu không nắm vững các quy tắc chuyển đổi hay các quy ước của NNLT thì CT máy tính sẽ cho kết quả sai lệch so với kết quả mong muốn.*



# 1.3. Biểu diễn dữ liệu

- Mọi bài toán từ đơn giản đến phức tạp đều dùng đến dữ liệu. Dữ liệu của bài toán (từ thế giới thực) thường rất phong phú và đa dạng.
- Tuy nhiên, trong máy tính, dữ liệu của CT đều là rời rạc, hữu hạn và chỉ được lưu trữ trong các biến (*không xét đến các lưu trữ trên bộ nhớ ngoài*).
- LTV phải biến đổi dữ liệu của bài toán cho phù hợp với cách biểu diễn trong máy tính.
- **1.3.1. Biến:** là nơi lưu trữ giá trị.
  - Mỗi biến đều có một **tên** dùng để phân biệt với các biến khác.
  - **Giá trị** mà biến lưu trữ có thể là số nguyên, số thực, ký tự, dòng chữ, .... Một biến chỉ có thể lưu trữ một loại giá trị nhất định. Loại giá trị mà biến có thể lưu trữ được gọi là **kiểu biến**.
  - Giá trị mà biến lưu trữ thường xuyên bị **biến đổi** trong quá trình CT thi hành.

## 1.3.1. Biến

Ví dụ 1.6:

Tên biến	Giá trị hiện tại	Kiểu biến	Ý nghĩa
STTu	100	Số nguyên	Số thứ tự
DiemTBinh	7.5	Số thực	Điểm trung bình
TenMHoc	'Toan'	Dòng ký tự ( <b>chuỗi</b> )	Tên một môn học

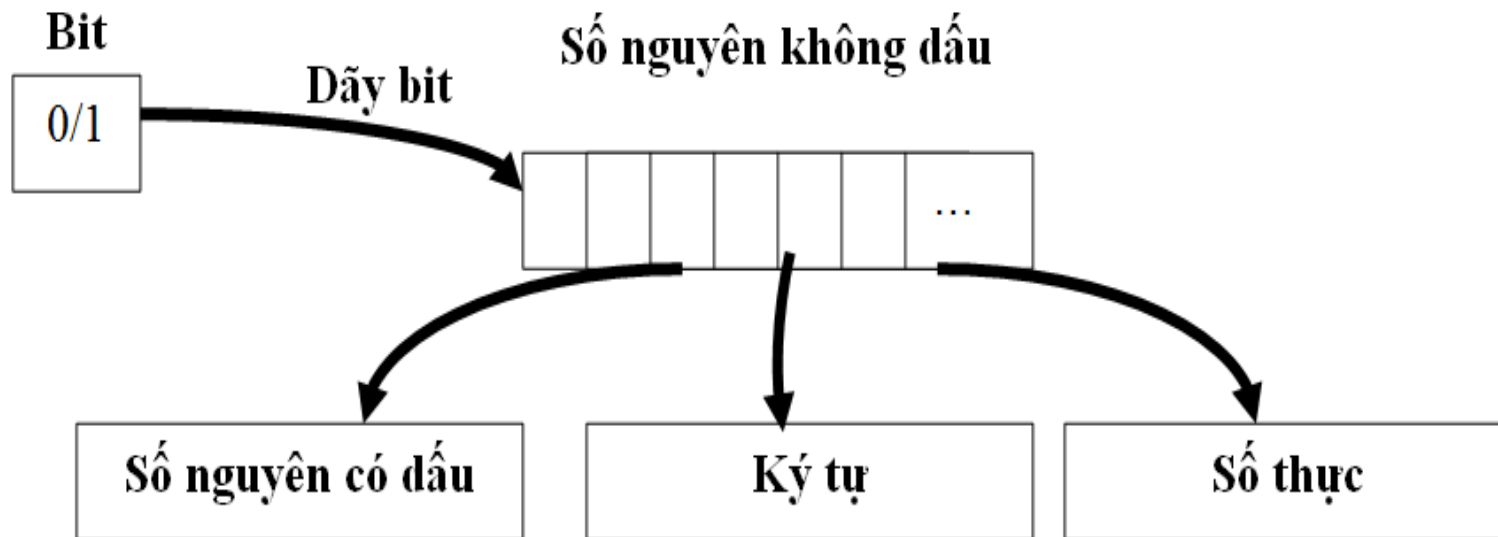
- Bộ nhớ máy tính chia thành nhiều ô nhớ, mỗi ô nhớ được xác định bằng một địa chỉ.
- Một biến chiếm một tập các ô nhớ kề nhau, địa chỉ biến là địa chỉ của ô nhớ đầu tiên.
- Để đọc giá trị của biến, máy tính tìm đến địa chỉ của biến và đọc nội dung của tất cả các ô nhớ để xác định giá trị.
- Việc nhớ địa chỉ của tất cả các biến gây khó khăn cho các LTV, do đó, các NNLT cho phép đặt tên cho biến. Khi thi hành CT, máy tính chuyển tên biến thành địa chỉ.

# 1.3.1.1. Tên biến

- Ý nghĩa của biến chỉ được hiểu bởi con người, do đó, tên biến phải gợi nhớ đến mục đích sử dụng để làm cho CT trong sáng, dễ sửa đổi.
- LTV phải chọn phong cách đặt tên biến của mình và nên tuân theo các quy tắc:
  - ✗ Tên biến phải liên quan đến ý nghĩa của biến.
  - ✗ Viết hoa các chữ cái đầu mỗi từ hoặc dùng dấu gạch dưới '\_' phân cách các từ.
  - ✗ Đừng đặt tên biến quá dài mà nên viết tắt sao cho khi nhìn vào tên tắt ta vẫn hiểu ngay được ý nghĩa của biến.
  - ✗ Tên biến phải có thêm tiền tố để biết được kiểu biến (nếu cần)
  - ✗ Tuân theo các quy tắc được sử dụng rộng rãi trong giới lập trình.

## 1.3.1.2. Kiểu biến

Dữ liệu trong thực tế thì đa dạng, nhưng trong máy tính dữ liệu chỉ thuộc về một số kiểu cơ bản (để có thể biểu diễn các đối tượng dữ liệu phức tạp trong thực tế, ta có thể dùng phương pháp cấu trúc hóa dữ liệu (xem chương 4)): số nguyên (tập con của tập số nguyên  $\mathbb{Z}$ ), số thực (tập con của tập số thực  $\mathbb{R}$ ), ký tự. Các kiểu dữ liệu cơ bản này được biểu diễn bằng dãy các bit 0/1.



Hình 1.5: Biểu diễn dữ liệu trong máy tính

## 1.3.1.2. Kiểu biến

Mỗi **bit** chứa một trong hai giá trị 0/1, một **ô nhớ** (byte) 8 bit có thể chứa  $2^8$  (256) giá trị khác nhau. Một biến lưu trữ một **số nguyên không dấu** thuộc miền xác định:  $[0, \dots, 255]$  cần đến 1 ô nhớ,  $[0, \dots, 4,294,967,295]$  cần đến 4 ô nhớ (32 bit), .... **Số nguyên có dấu** được biểu diễn qua số nguyên không dấu (xem các tài liệu nhập môn tin học).

Đối với **số thực**, khi biểu diễn trong máy tính, người ta không nói đến miền xác định mà nói đến độ chính xác (số thực *dấu chấm động*). Số thực có độ-chính-xác-dưới  $m$ , độ-chính-xác-trên  $n$  có tối đa  $m$  chữ số ở phần thập phân,  $n$  chữ số ở phần nguyên. Số thực trong máy tính là *rời rạc*. Chẳng hạn  $1/3$  không phải là  $0.3333\dots$  mà là số gần đúng với giá trị này, do đó,  $1$  là khác với  $(1/3)*3$ .

Để **biểu diễn tập 256 ký tự**  $\{ '0', '1', \dots, '9', \dots, 'A', 'B', \dots, 'Z', \dots, 'a', 'b', \dots, 'z', \dots, '*', '+', '@', \dots \}$  ta sẽ tương ứng mỗi ký tự với một số nguyên từ 0 đến 255. Số nguyên tương ứng với ký tự được gọi là mã ASCII của ký tự. Việc biểu diễn ký tự quy về việc biểu diễn một số nguyên.

## 1.3.1.2. Kiểu biến

*Bảng 1.1: Các kiểu dữ liệu cơ bản trong các NNLT*

Kiểu số nguyên			Miền xác định	Kích thước (số bytes)
Nhỏ	Ngắn	Không dấu	0 .. 255	1
		Có dấu	-128 .. 127	1
	Dài	Không dấu	0 .. 65,535	2
		Có dấu	-32,768 .. 32,767	2
Lớn		Không dấu	0 .. 4,294,967,295	4
		Có dấu	-2,147,483,648 .. 2,147,483,647	4
Kiểu số thực			Độ chính xác	Kích thước
Nhỏ			$3.4*10^{-38} \text{ .. } 3.4*10^{38}$	4
Lớn			$1.7*10^{-308} \text{ .. } 1.7*10^{308}$	8
Rất lớn			$3.4*10^{-4932} \text{ .. } 1.1*10^{4932}$	10

## 1.3.1.2. Kiểu biến

Khi viết CT, LTV phải cân nhắc việc chọn lựa kiểu biến phù hợp với mục đích sử dụng. Sau đây là một số quy tắc:

- ✎ *Đừng dùng biến có miền xác định quá lớn để biểu diễn cho các dữ liệu có miền xác định quá nhỏ.*
- ✎ *Đừng dùng biến có miền xác định nhỏ để biểu diễn cho các dữ liệu có miền xác định lớn.* Chẳng hạn, vào thời kỳ mới xuất hiện máy tính người ta chỉ biểu diễn năm bằng 2 chữ số vì lý do tiết kiệm (bộ nhớ lúc đó rất đắt): “75” sẽ biểu diễn năm 1975, hàng loạt CT ra đời dựa trên cách biểu diễn này: quản lý điều hành không lưu, tín dụng ngân hàng, .... Từ năm 2000 trở đi, các CT máy tính không phân biệt được năm 2000 với năm 1900 vì cả hai đều được biểu diễn bằng 2 chữ số “00” (sự cố Y2K) dẫn đến thiệt hại rất lớn.
- ✎ *Không nhất thiết phải dựa vào miền xác định của dữ liệu thực tế.* Ví dụ, để biểu diễn điểm trung bình (có một chữ số ở phần thập phân) ta không nên dùng số thực mà chỉ nên dùng số nguyên với quy ước: 105 là 10.5, 90 là 9.0, .... Cách biểu diễn này vừa tiết kiệm mà vừa giúp CT thực hiện nhanh hơn (máy tính xử lý số nguyên nhanh hơn số thực).



## 1.3.2. Hằng

Có hai loại hằng: a) hằng giá trị, b) hằng định danh.

**Hằng giá trị** là các giá trị, chẳng hạn: số nguyên 13, số thực 4.5, ký tự 'a', ký tự '+', .... Việc áp dụng các hằng giá trị làm cho CT khó đọc vì hằng giá trị không gợi ý nghĩa sử dụng của nó. Hơn nữa, việc thay đổi một hằng giá trị tồn tại ở nhiều nơi trong CT là mất thời gian và dễ sai sót.

Để tránh các bất lợi của việc dùng hằng giá trị, các NNLT cho phép định nghĩa **hằng định danh** tương ứng với giá trị: định danh sẽ thay cho giá trị. *Dùng hằng định danh làm cho CT trong sáng và dễ sửa đổi*: khi cần sử dụng giá trị ta sẽ viết định danh, khi cần thay đổi giá trị ta chỉ chỉnh sửa một lần ở vị trí định nghĩa.

Chẳng hạn, có thể định nghĩa hằng định danh PI tương ứng với giá trị 3.14. Khi cần tăng độ chính xác của PI chỉ cần thay đổi một lần lúc khai báo PI, chứ không thay thế hàng loạt các giá trị 3.14 thành 3.1416 ở nhiều nơi trong CT, vừa mất thời gian, vừa dễ sai sót.



## 1.4. Biểu diễn thuật toán

Bên cạnh việc biểu diễn dữ liệu của bài toán vào CT, ta còn phải biểu diễn các bước thực hiện của thuật toán vào CT. Các thao tác tính toán ở thế giới thực không phức tạp như dữ liệu mà chỉ được hình thành từ ba cấu trúc cơ bản: cấu trúc tuần tự, cấu trúc rẽ nhánh và cấu trúc lặp.

### 1.4.1. Cấu trúc tuần tự:

Các thao tác tuần tự thực hiện từ trên xuống dưới theo đúng trình tự xuất hiện của chúng trong thuật toán. Trong cấu trúc tuần tự, lệnh (thao tác) gán là lệnh thường gặp nhất, nó có dạng:

*Tên biến*  $\leftarrow$  *Biểu thức tính toán*;

Biểu thức tính toán chứa các *toán hạng*, các *phép toán*, và các *ký hiệu gom nhóm*. Các *toán hạng* bao gồm các biến, hằng, các giá trị và có thể là các lời gọi hàm (xem chương 3, ở đây tạm hiểu là các hàm toán học). Các *phép toán* thường là các phép toán hay gặp trong toán học (có thể được ký hiệu khác đi):  $+$ ,  $-$ ,  $*$ ,  $/$ , ... Các *ký hiệu gom nhóm* trong các NNLT hạn chế hơn trong toán học, chẳng hạn ký hiệu “(”, “)” được dùng thay cho cả “[”, “]” và “{”, “}”.

Một *biểu thức* trả về một giá trị thuộc một kiểu dữ liệu nhất định. Biểu thức trả về giá trị số nguyên/số thực/ký tự được gọi là biểu thức nguyên/thực/ký tự.

# 1.4.1. Cấu trúc tuần tự

Một biểu thức trả về một giá trị thuộc một kiểu dữ liệu nhất định. Biểu thức trả về giá trị số nguyên/số thực/ký tự được gọi là biểu thức nguyên/thực/ký tự.

Trình tự tính toán giá trị biểu thức trong các NNLT cũng tương tự như trong toán học: các thành phần có độ ưu tiên cao (thấp) sẽ được thực hiện trước (sau), các thành phần có cùng độ ưu tiên lần lượt được thực hiện từ trái sang phải.

Bảng 1.2: Độ ưu tiên của các thành phần trong biểu thức

Độ ưu tiên	Thành phần
5	Gọi hàm
4	Biểu thức con chứa trong dấu ngoặc ( )
3	Toán tử một ngôi
2	Các phép toán: *, /,
1	Các phép toán: +, −.

# 1.4.1. Cấu trúc tuần tự

Ví dụ 1.7: Giả sử giá trị hiện hành của các biến `tong_tien`, `a`, `b`, lần lượt là 0, 30, 6; kết quả của một số biểu thức cho trong bảng sau:

Biểu thức	Kết quả	Loại biểu thức
$((1.0 + 9.0) / 5.0) - (PI + PI)$	- 4.28	Thực
'b' - 'a'	1	Nguyên (xem 2.2.2)
<code>tong_tien</code> - 2	- 2	Nguyên
$(a + b) * 2$	72	Nguyên
2.5	2.5	Thực
'W'	'W'	Ký tự

## 1.4.1. Cấu trúc tuần tự

✎ Đối với các biểu thức dài và phức tạp, nên dùng (nhưng đừng lạm dụng) các biến trung gian để thay thế các biểu thức con. Chẳng hạn, để tính nghiệm của phương trình bậc 2:

- Cách 1:

$$x_1 \leftarrow (-b - \sqrt{b*b - 4*a*c}) / (2*a)$$

$$x_2 \leftarrow (-b + \sqrt{b*b - 4*a*c}) / (2*a)$$

- Cách 2: dùng biến trung gian `can_delta`

$$can\_delta \leftarrow \sqrt{b*b - 4*a*c}$$

$$x_1 \leftarrow (-b - can\_delta) / (2*a)$$

$$x_2 \leftarrow (-b + can\_delta) / (2*a)$$

- Cách 3: lạm dụng việc dùng các biến trung gian

$$can\_delta \leftarrow \sqrt{b*b - 4*a*c}$$

$$tru\_b \leftarrow -b$$

$$hai\_a \leftarrow 2*a$$

$$x_1 \leftarrow (tru\_b - can\_delta) / hai\_a$$

$$x_2 \leftarrow (tru\_b + can\_delta) / hai\_a$$

So với cách 1, cách 2 ngắn gọn hơn và nhanh hơn (chỉ tính căn thức một lần); còn cách 3 thì dùng quá nhiều biến trung gian.

## 1.4.1. Cấu trúc tuần tự - Biểu thức logic

Nếu trong biểu thức có các phép toán so sánh ( $\geq, \leq, >, <, =, \neq$ ) thì giá trị của biểu thức là đúng hoặc sai, biểu thức sẽ được gọi là biểu thức logic. Các biểu thức logic kết hợp với nhau bằng các phép toán logic *và, hoặc, phủ định* để tạo thành các biểu thức logic mới.

*Bảng 1.3: Quy tắc của các phép toán logic*

A	B	A và B	A hoặc B	Phủ định A
Đúng	Đúng	Đúng	Đúng	Sai
Đúng	Sai	Sai	Đúng	Sai
Sai	Đúng	Sai	Đúng	Đúng
Sai	Sai	Sai	Sai	Đúng

Ví dụ 1.8: Giả sử giá trị hiện hành của các biến a, b, lần lượt là 2, 3; kết quả của một số biểu thức logic cho trong bảng sau:

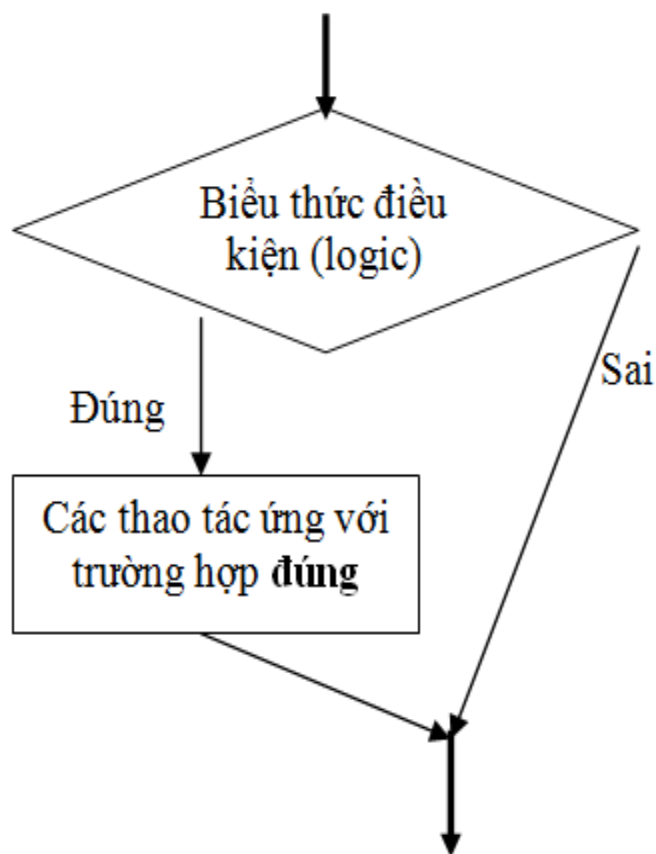
Biểu thức logic	Trị logic của biểu thức
$(5 > 1)$ hoặc $(6 < 1)$	Đúng
$(2 = 1)$ và $(5 = 5)$	Sai
phủ định $(5 = 4)$	Đúng
$(a + b) < \text{PI}$	Sai (xem 1.3.2)

## 1.4.2. Cấu trúc rẽ nhánh

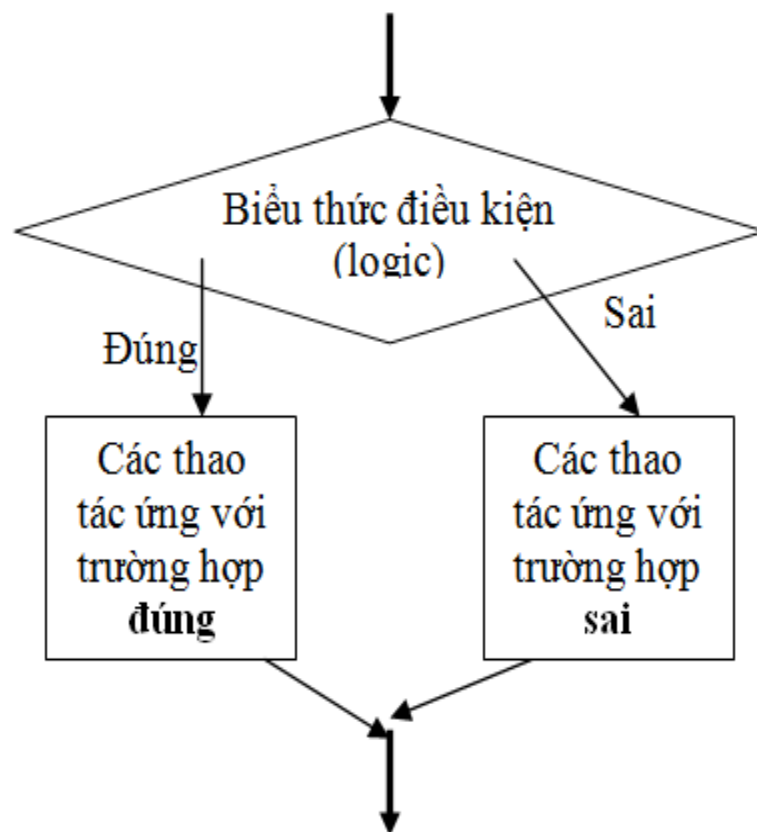
Cấu trúc rẽ nhánh phá vỡ trình tự thi hành tuần tự của CT dựa trên kết quả của biểu thức điều kiện (biểu thức logic hoặc biểu thức nguyên). Rẽ nhánh dựa trên kết quả của biểu thức logic gồm có: rẽ nhánh-đơn, rẽ nhánh-đôi. Rẽ nhánh dựa trên kết quả của biểu thức nguyên được gọi là rẽ nhiều-nhánh. Có thể biểu diễn (nhưng bất tiện) rẽ nhiều-nhánh thông qua rẽ nhánh-đôi (bài tập).

- **Cấu trúc rẽ nhánh-đơn:** CT tính giá trị của biểu thức điều kiện và rẽ nhánh nếu biểu thức đúng, ngược lại CT thực hiện tiếp lệnh sau cấu trúc rẽ nhánh.
- **Cấu trúc rẽ nhánh-đôi:** CT tính giá trị của biểu thức điều kiện và rẽ theo một trong hai nhánh ứng với giá trị đúng hoặc sai.
- **Cấu trúc rẽ nhiều-nhánh:** CT tính giá trị của biểu thức điều kiện và rẽ theo nhánh tương ứng với kết quả của biểu thức.

## 1.4.2. Cấu trúc rẽ nhánh

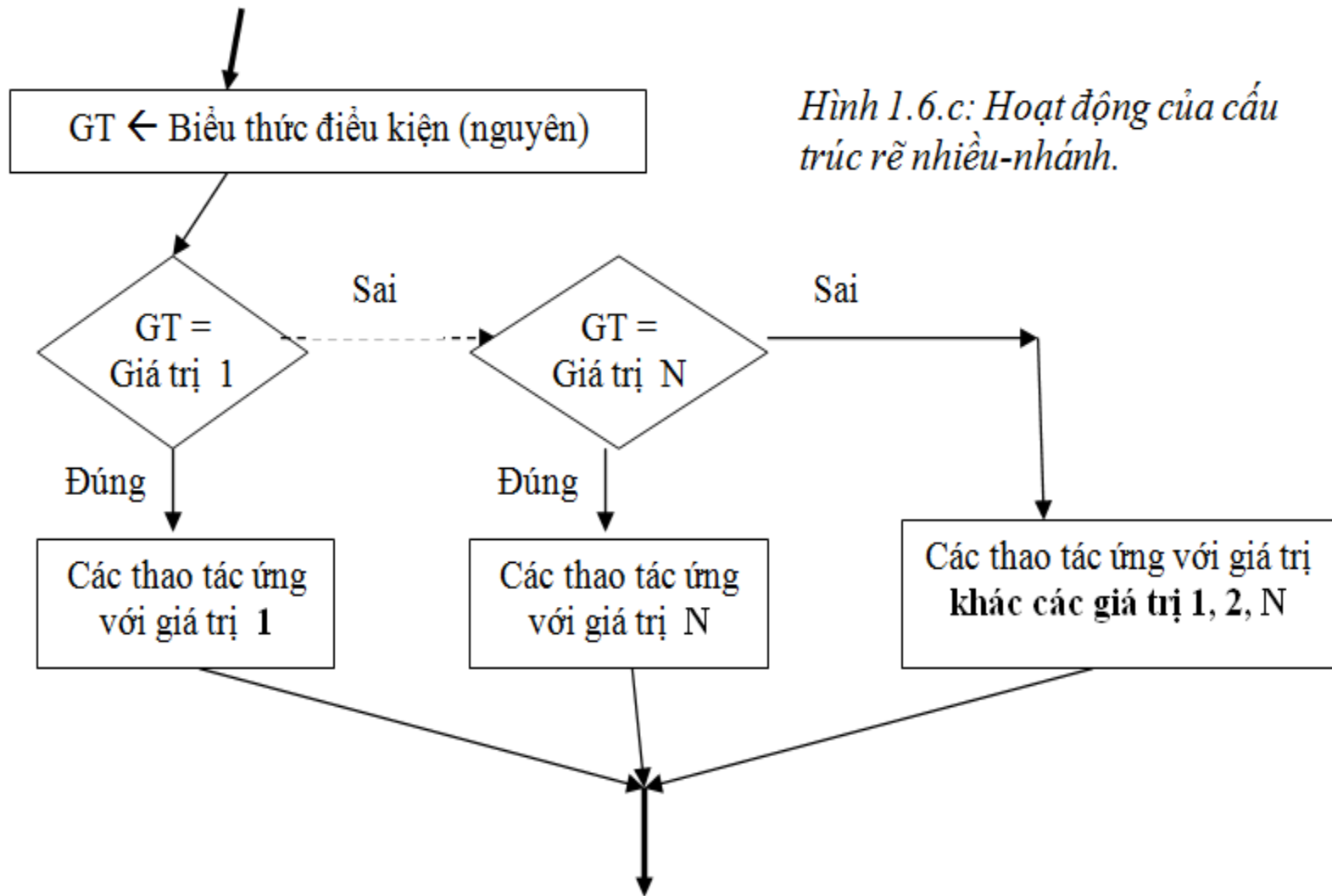


Hình 1.6.a: Hoạt động của cấu trúc rẽ nhánh-đơn.



Hình 1.6.b: Hoạt động của cấu trúc rẽ nhánh-đôi.

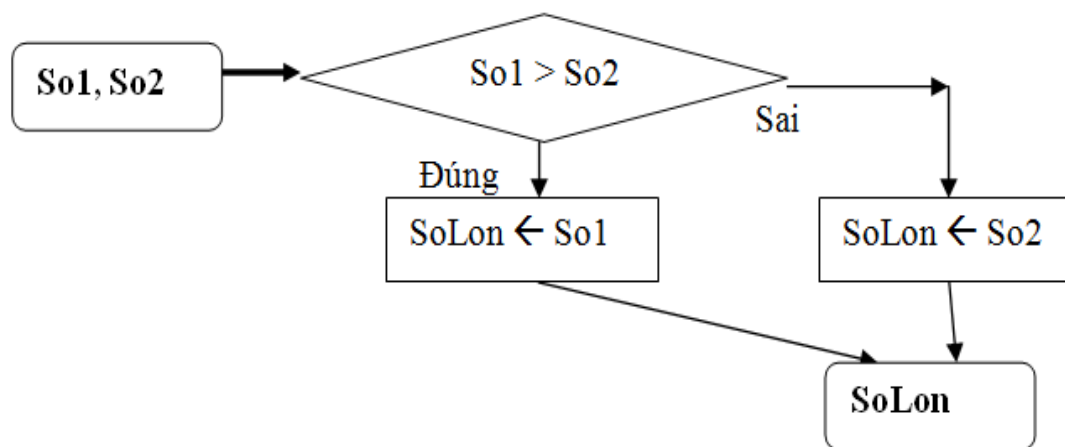
## 1.4.2. Cấu trúc rẽ nhánh





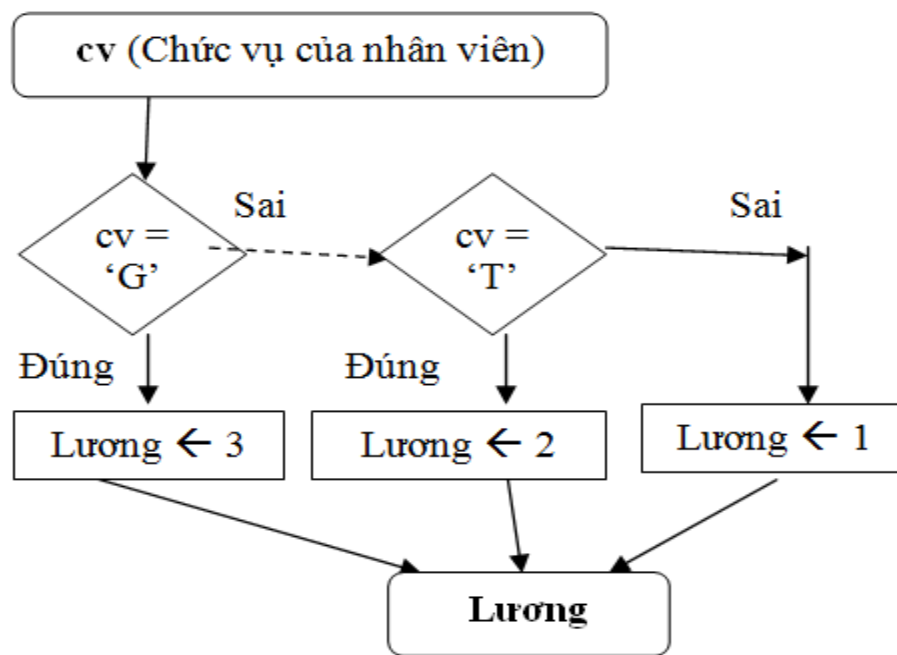
## 1.4.2. Cấu trúc rẽ nhánh

Ví dụ 1.9: Thuật toán tìm số lớn nhất trong hai số dùng cấu trúc rẽ nhánh đôi



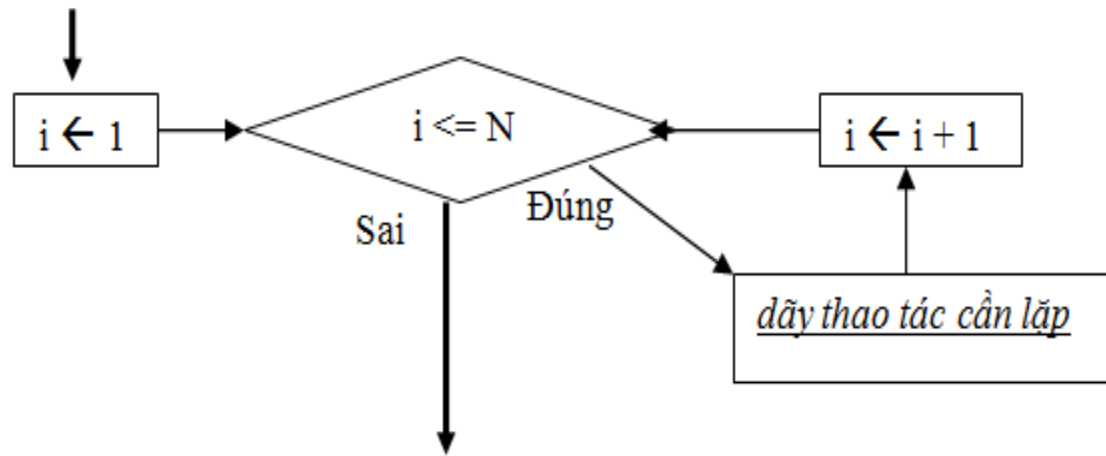
Ví dụ 1.10: Thuật toán tính lương nhân viên (Lương) dựa vào chức vụ cv theo bảng dùng cấu trúc rẽ nhiều nhánh

Ký hiệu	Ý nghĩa	Lương
G	Giám đốc	3
T	Trưởng phòng	2
Khác	Chức vụ khác	1

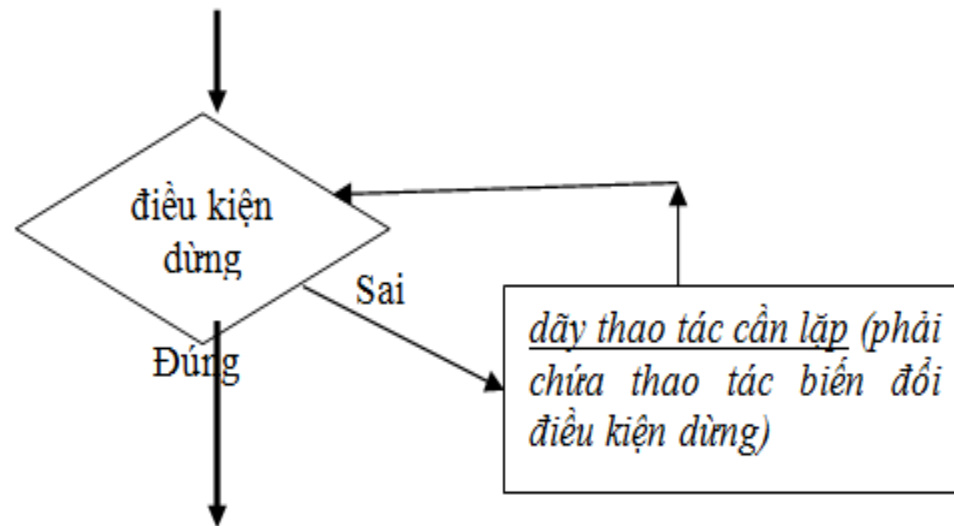


## 1.4.3. Cấu trúc lặp

- Cấu trúc lặp điều khiển việc lặp đi lặp lại các thao tác.
- Lặp xác-định: biết trước số lần lặp.
- Lặp không-xác-định: không biết (khó tính) số lần lặp.



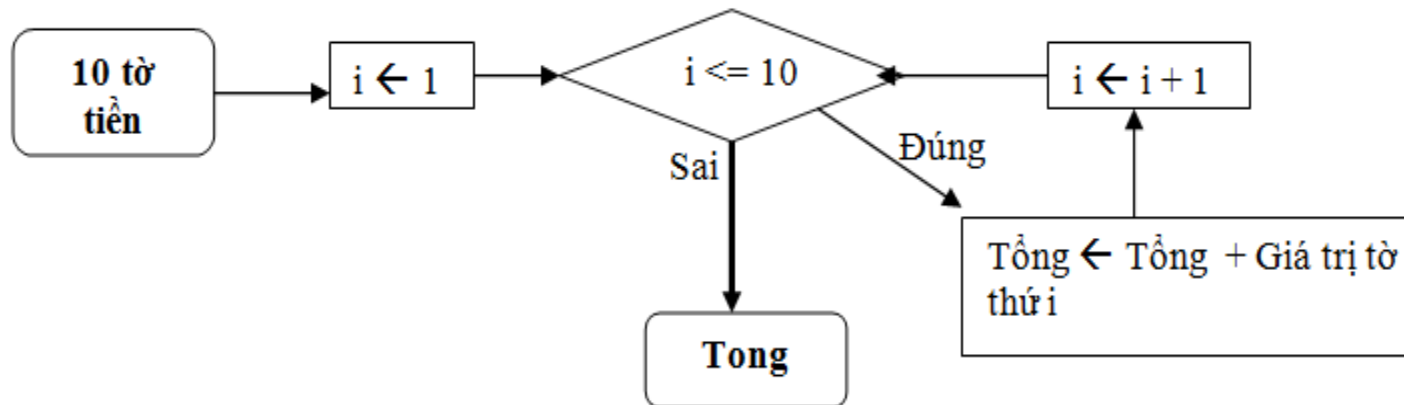
Hình 1.7.a: Hoạt động của cấu trúc lặp xác-định  
(trong đó:  $N$  là số lần lặp,  $i$  là chỉ số của bước lặp hiện tại).



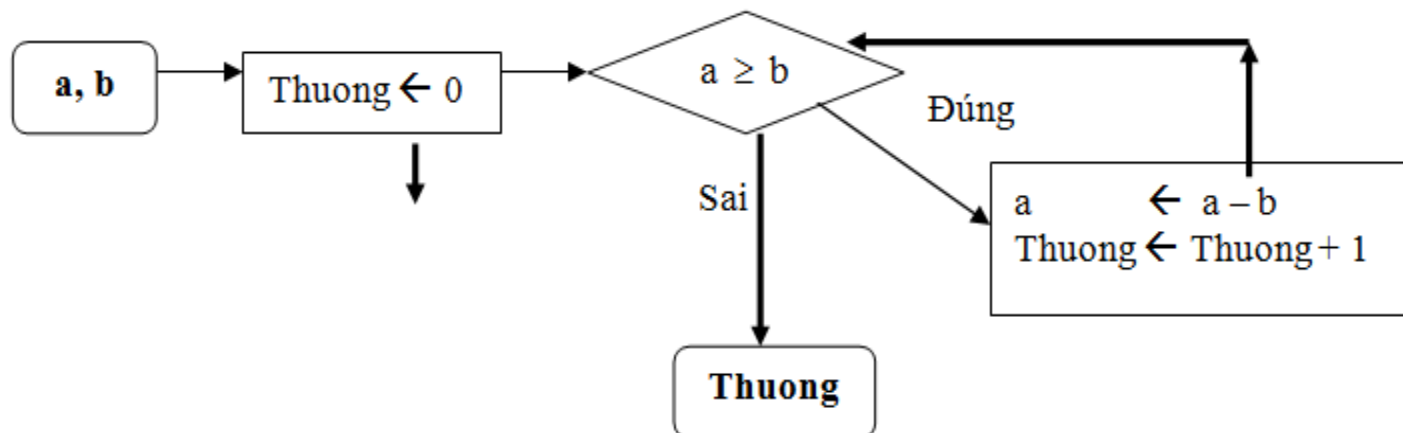
Hình 1.7.b: Hoạt động của cấu trúc lặp không-xác-định.

## 1.4.3. Cấu trúc lặp

Ví dụ 1.11: Thuật toán tính tổng giá trị 10 tờ tiền dùng cấu trúc lặp xác-định

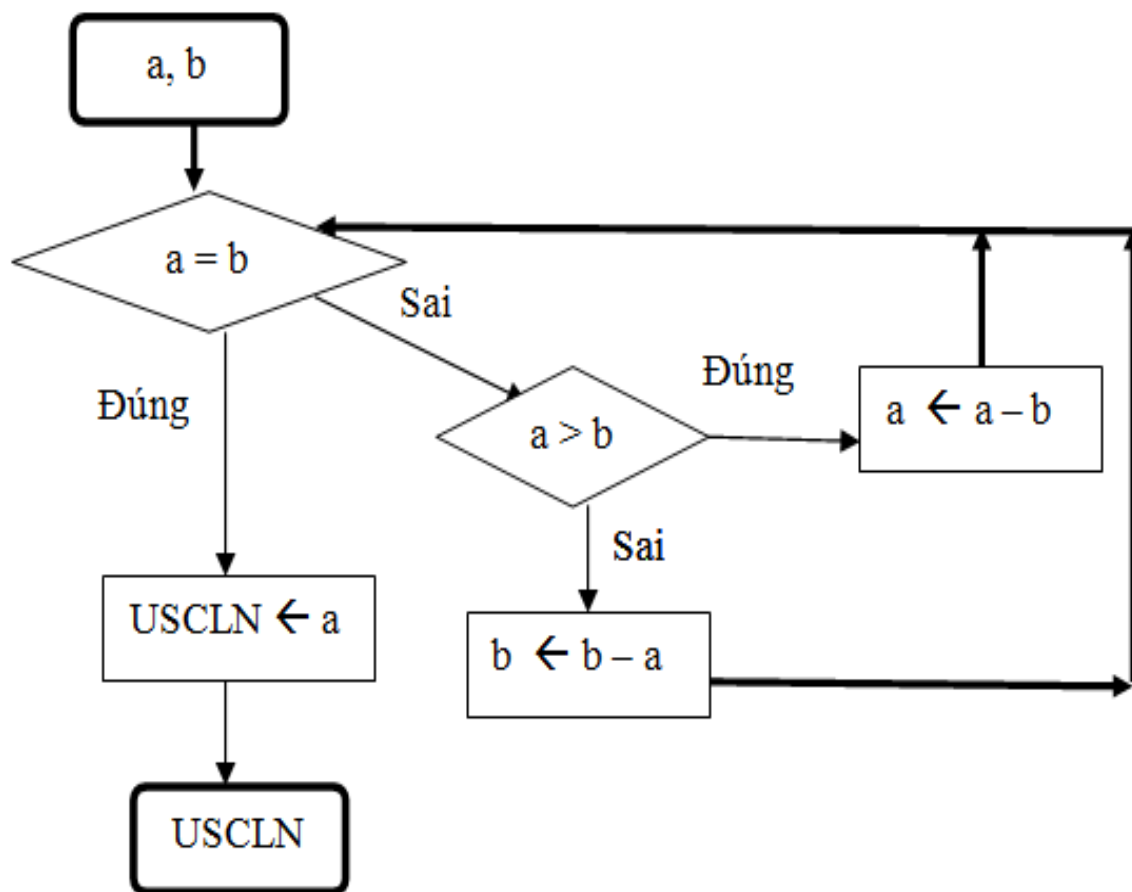


Ví dụ 1.12: Thuật toán chia a cho b lấy thương bằng các phép trừ dùng cấu trúc lặp không-xác-định



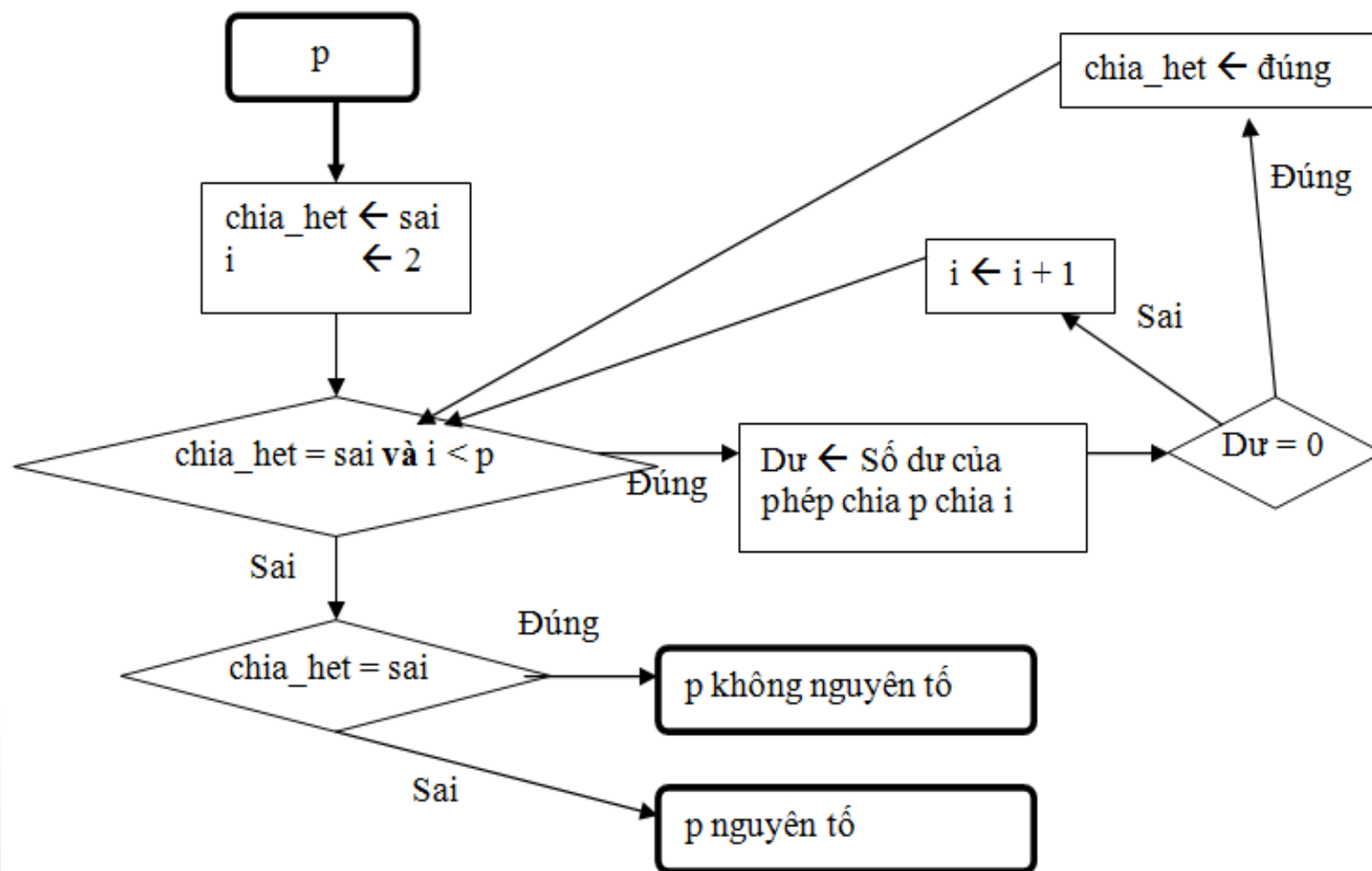
## 1.4.3. Cấu trúc lặp

Ví dụ 1.13: Thuật toán tìm ước số chung lớn nhất (USCLN) của hai số nguyên  $a, b$  dùng cấu trúc lặp không-xác-định.



## 1.4.3. Cấu trúc lặp

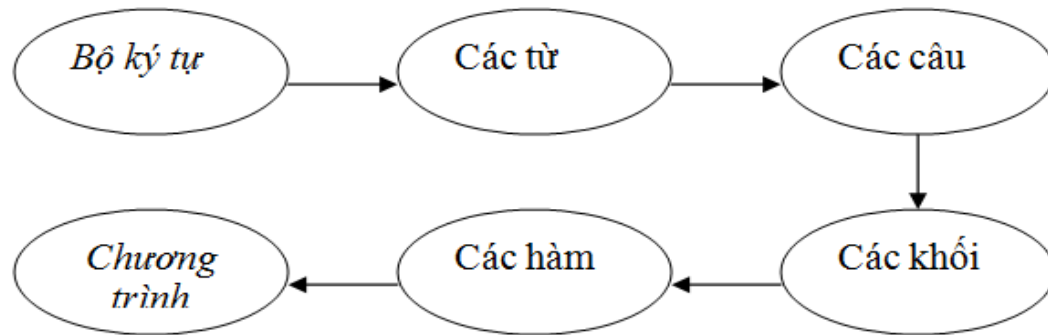
Ví dụ 1.14: Thuật toán kiểm tra tính nguyên tố của số nguyên  $p$  dùng cấu trúc lặp không-xác-định



# CHƯƠNG 2: LẬP TRÌNH CƠ BẢN

- 2.1 Các thành phần cơ bản của ngôn ngữ C++
- 2.2 Các kiểu dữ liệu cơ bản và các phép toán trên chúng
- 2.3 Khai báo hằng và kiểu liệt kê
- 2.4 Biến, lệnh gán, biểu thức
- 2.5 Cấu trúc một chương trình C++ đơn giản
- 2.6 Các lệnh điều khiển
- 2.7 Phân tích một số chương trình đơn giản

# 1. Các thành phần cơ bản của ngôn ngữ C++



Hình 2.1 Các thành phần cơ bản của C++

## 2.1.1. Bộ ký tự:

- Các chữ cái: A, B, ..., Z, a, b, ..., z.
- Ký tự gạch thấp: \_.
- Bộ chữ số: 0, 1, ..., 9.
- Các ký hiệu toán học: +, -, \*, /, <=, >, ...
- Một số ký tự khác: (, ), {, }, ....

## 2.1.3. Các câu, các khối, các hàm, chương trình:

Tổ hợp các từ hình thành các câu lệnh. Các câu lệnh bao gồm: lệnh gán, lệnh điều khiển, lệnh gọi hàm, ...

Tổ hợp các câu lệnh ta có các khối lệnh. Mỗi khối lệnh thực hiện một nhiệm vụ xác định trong CT.

CT bao gồm các hàm độc lập nhau, mỗi hàm chứa nhiều khối lệnh.

# 1. Các thành phần cơ bản của ngôn ngữ C++

## 2.1.2. Các từ:

Sự tổ hợp các ký tự tạo ra các từ (định danh). Có những từ do C++ tạo ra và dành sẵn – các từ khóa, có những từ do LTV đặt (không trùng với các từ khóa). C++ phân biệt chữ hoa và chữ thường.

### a. Các từ khóa:

#### ▪ Khai báo:

*main, #define, typedef, struct,  
int, char, integer, float, ....*

#### ▪ Các cấu trúc điều khiển:

*if, else, switch, for, while, do, ....*

### b. Các từ do LTV đặt:

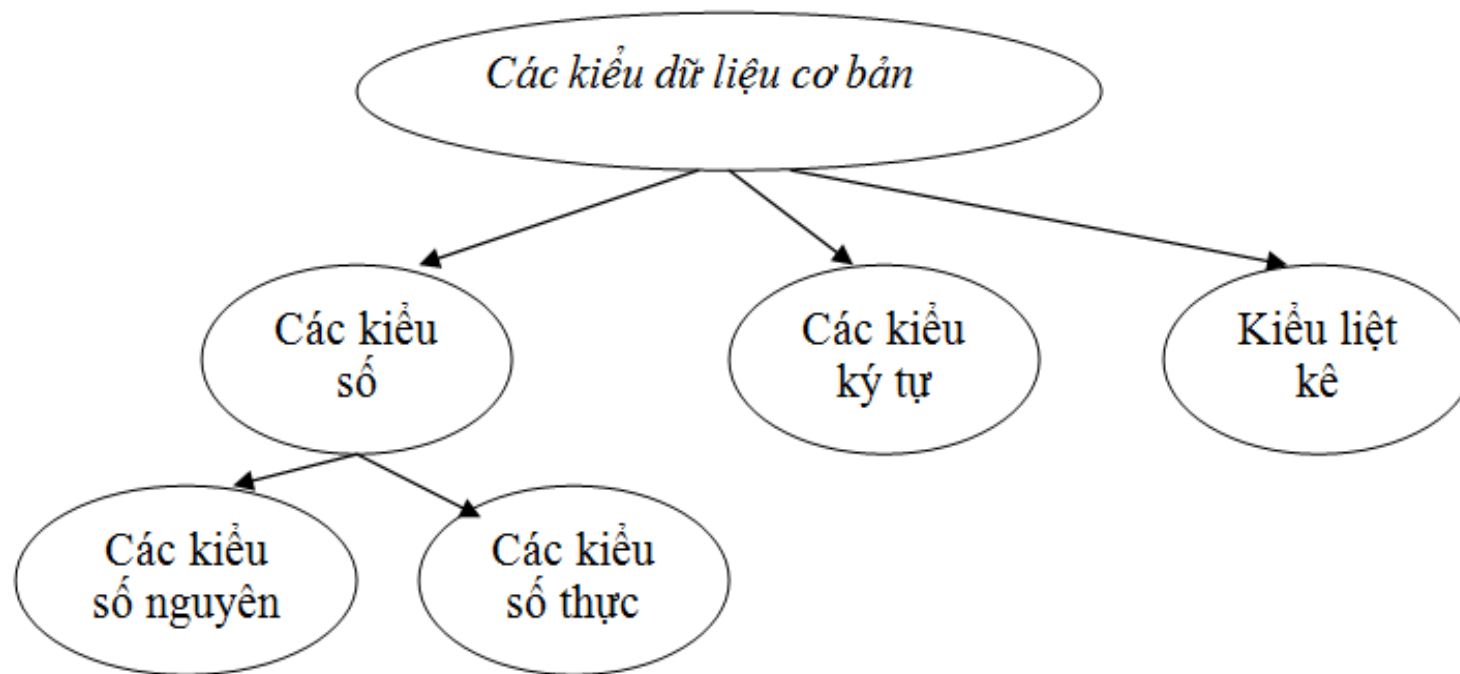
LTV sẽ tạo ra các từ (định danh) với mục đích đặt tên cho: hằng, biến, kiểu dữ liệu, hàm, .... Chúng phải bắt đầu bằng một chữ cái, sau đó là các chữ số, dấu gạch thấp, ... Chẳng hạn, các tên sau không hợp lệ: *a#b* (chứa ký tự #), *float* (trùng với từ khóa), *9tong* (bắt đầu bằng chữ số), *a+b* (có ký tự +), *a b* (có khoảng trắng), ....

Nên tạo ra phong cách riêng của mình khi đặt tên các từ để dễ theo dõi CT.



## 2. Các kiểu dữ liệu cơ bản và các phép toán trên chúng

*Một kiểu dữ liệu là một quy định chung về hình dạng, cấu trúc, giá trị cũng như cách biểu diễn và xử lý. LTV phải chọn các kiểu dữ liệu thích hợp để có thể giải tốt bài toán đặt ra. Một NNLT chỉ chấp nhận các kiểu dữ liệu tuân theo (hoặc được xây dựng trên) quy định của nó. Trong C++, các kiểu dữ liệu cơ bản gồm: số nguyên, số thực, ký tự, liệt kê, .....*



Hình 2.2 Các kiểu dữ liệu cơ bản

## 2. Các kiểu dữ liệu cơ bản và các phép toán trên chúng

Bảng 2.1: Tên các kiểu số trong C++

Kiểu số nguyên			Tên kiểu trong C++
Nhỏ	Ngắn	Không dấu	unsigned char
		Có dấu	<b>char</b>
	Dài	Không dấu	unsigned int
		Có dấu	<u><b>int</b></u>
Lớn		Không dấu	unsigned long
		Có dấu	<u><b>long</b></u>
Kiểu số thực			Tên kiểu trong C++
Nhỏ			<u><b>float</b></u>
Lớn			<u><b>double</b></u>
Rất lớn			long double

Các phép toán trên các kiểu số gồm:

- Các phép toán số học: +, -, \*, % (*phép chia giữa hai số nguyên lấy phần dư, chẳng hạn  $5 \% 3 = 2$ ), / ((a) nếu hai vế của phép chia đều là số nguyên thì / là phép chia lấy phần nguyên (ví dụ,  $5 / 3 = 1$ ), (b) ngược lại, / là phép chia có kết quả là số thực (ví dụ,  $5.0 / 3 = 1.66$ )).*
- Các phép toán so sánh: <, <=, >, >=, == (*so sánh bằng*), != (*so sánh khác*).

## 2. Các kiểu dữ liệu cơ bản và các phép toán trên chúng

Bảng 2.2: Biểu diễn ký tự bằng kiểu số nguyên ngắn

Mã ASCII	Giá trị nguyên tương ứng
	unsigned char (miền xác định: [0..255])
0	0
1	1
..	..
127	127
128 ( $=127 + 1$ )	128
129 ( $=127 + 2$ )	129
..	..
255 ( $=127 + 128$ )	255

### Các kiểu ký tự:

Như đã biết (trong 1.3.1.2), một ký tự được biểu diễn bằng một số nguyên có giá trị là mã ASCII của ký tự. Tập mã ASCII của các ký tự được đánh số từ 0 đến 255 và có thể lưu trữ trong hai kiểu số nguyên ngắn: *char* và *unsigned char* cho trong bảng 2.2.

*Các phép toán trên kiểu ký tự có thể xem là các phép toán trên mã ASCII của các ký tự, hay là các phép toán trên kiểu số nguyên.*

## 4. Biến, lệnh gán, biểu thức

### 2.4.1 Biến và lệnh gán:

Cú pháp khai báo biến (*trước khi sử dụng*):

*kiểu\_dữ\_liệu*      *tên\_biến\_1* = *giá\_trị\_ban\_đầu\_1*,  
                         *tên\_biến\_2* = *giá\_trị\_ban\_đầu\_2*,  
                         ...;

Nếu trong khai báo biến ta không dùng “= *giá\_trị\_ban\_đầu*” thì giá trị ban đầu của biến là một giá trị ngẫu nhiên (tùy thuộc vào giá trị hiện tại của ô nhớ được cấp phát cho biến).

Ví dụ 2.4: Một số khai báo biến

int	tong_tien, to_trong_tui, to_vua_lay, uscln, a, b;
float	dtich_hcn, dtich_hvuong, tong_dtich, cdai, crong, canh;
char	xep_loai, chuc_vu;
color	mau_nen, mau_chu;
cac_ngay	ngay_lam_viec;

Các khai báo biến sai (thừa, thiếu dấu ‘,’ hoặc thiếu dấu ‘;’) thường gặp:

int	tong_tien, ;
float	dtich_hcn dtich_hvuong;
char	xep_loai, chuc_vu

## 4.1. Biến, lệnh gán

✎ Để gán giá trị của một biểu thức cho biến ta dùng lệnh gán '=' với cú pháp:

`tên_biến = biểu_thức;`

hoặc `tên_biến_1 = tên_biến_2 = .. = tên_biến_N = biểu_thức;`

để gán một giá trị cho nhiều biến khác nhau.

Ví dụ 2.5: một số lệnh gán

```
ngay_lam_viec = T_HAI;      chuc_vu = 'L';  
mau_nen = mau_chu = BLUE; a = b = 12;      tong_tien = tong_no = 0;  
gio          = giay_nhap_vao / GIAY_TREN_GIO;      (1)  
giay_du      = giay_nhap_vao % GIAY_TREN_GIO;      (2)  
phut         = giay_du / GIAY_TREN_PHUT;          (3)  
giay         = giay_du % GIAY_TREN_PHUT;          (4)
```

Trong ví dụ 2.2, ta đã định nghĩa các hằng GIAY\_TREN\_GIO, GIAY\_TREN\_PHUT.  
*Hãy cho biết ý nghĩa của các lệnh từ (1) đến (4).*

## 4.2. Biểu thức

Các biểu thức trong C++ tương tự trong các NNLT khác (xem 1.4.1). Đối với biểu thức logic thì có một số khác biệt.

### Biểu thức logic

C++ dùng biểu thức số học để biểu diễn biểu thức logic. Một biểu thức số học trong C++ có kết quả là 0 (khác 0) nếu hiểu theo nghĩa logic sẽ là sai (đúng).

*Bảng 2.4: Các phép toán logic*

Phép toán logic	Ký hiệu
Phủ định	!
Và	&&
Hoặc	

## 4.2. Biểu thức

Ví dụ 2.6: Giá trị của một số biểu thức logic (với  $a = 3$ ,  $b = 5$ )

Biểu thức	Trị số học	Trị logic
$(5 == 5) \&\& (6 != 2)$		
$(5 > 1) \parallel (6 < 1)$		
$(5 \&\& (4 > 2))$		
$(5 \&\& (4 < 2))$		
$!(5 == 4)$		
$(a + b) < \text{PI}$		

✎ Có thể sử dụng các phép toán số học với biểu thức logic (mặc dù điều này vô nghĩa về mặt Toán học). Chẳng hạn, biểu thức  $(1 < 2) - (3 < 2) + (4 < 5)$  có giá trị 2 (vì *sao?*).

## 4.3. Quá trình tính toán biểu thức của C++

*Bảng 2.5: Thứ tự thực hiện của các thành phần trong biểu thức*

Độ ưu tiên	Thành phần của biểu thức
0	Lời gọi hàm
1	Biểu thức con chứa trong dấu ngoặc ( )
2	!
3	– (phép toán đổi dấu)
4 (4.5)	*, /, % (+, -)
5	<, <=, >, >=
6	==, !=
7	&&,

- Các thành phần trong biểu thức có độ ưu tiên nhỏ được thực hiện trước, trong trường hợp hai thành phần có cùng độ ưu tiên thì thành phần bên trái thực hiện trước. Khi viết các biểu thức, ta nên dùng nhiều dấu mở, đóng ngoặc.
- Trước khi tính toán các biểu thức số, C++ tiến hành một số thao tác tối ưu (chẳng hạn, chuyển đổi các thành phần của biểu thức về kiểu dữ liệu có kích thước nhỏ hơn để giảm bớt thời gian tính toán, ...) do đó, khi muốn thể hiện một biểu thức số vào C++, ta cần thận trọng.



## 4.3. Quá trình tính toán biểu thức của C++

Lấy ví dụ, biểu thức  $1/100$  có kết quả là  $0$  (do hiệu ‘/’ là phép chia nguyên) chứ không phải  $0.01$  (theo nghĩa phép chia thực). Nếu muốn có kết quả  $0.01$  ta phải viết  $1/100.0$ , hoặc  $1.0/100$  hoặc  $1.0/100.0$ . Nhưng giả sử  $i$  và  $j$  là hai biến nguyên lưu giá trị  $1$  và  $100$ . Để biểu thức  $i/j$  thực hiện phép chia thực, ta có hai cách sau:

Cách 1: nhân tử hoặc mẫu cho  $1.0$ , chẳng hạn  $i/(j*1.0)$ . Chú ý, phải có dấu ngoặc bao  $j$  và  $1.0$ , vì nếu không, biểu thức vẫn cho ra kết quả sai (là  $0.0$ , vì sao?).

Cách 2: dùng phép toán ép kiểu để buộc C++ hiểu biến nguyên  $i$  có kiểu thực “ $float(i)/j$ ”.

Để ép kiểu, ta dùng cú pháp:

`tên_kiểu_mới` (*biến hoặc biểu thức cần ép kiểu*)

hoặc

`(tên_kiểu_mới)` (*biến hoặc biểu thức cần ép kiểu*).

✎ Một trong những tác dụng của kỹ thuật ép kiểu là tránh tràn số.

## 4.3. Quá trình tính toán biểu thức của C++

Xét đoạn CT:

```
int      i, j, k2;  
long     k1;  
i = 1000;  
j = 50;  
k1 = i * j;  
k2 = i * j / 10;
```

a) Về mặt toán học,  $i*j$  cho kết quả 50000 và có thể lưu vào k1. Tuy nhiên, vì i, j thuộc kiểu int nên  $i*j$  là biểu thức int có kết quả là -15536 (sai, vì sao?). Nếu ép kiểu “*long (i\*j)*” thì quá chậm vì *phép nhân thực hiện trước phép ép kiểu*. Trong trường hợp này, ta phải ép kiểu i trước khi thực hiện phép nhân: “*long(i) \* j*”. Khi đó, giá trị của i có kiểu long,  $i*j$  là biểu thức long, và k1 nhận giá trị đúng.

b) Tương tự, ta cho rằng  $i*j/10$  có kết quả 5000 vì (về mặt toán học)  $i * \frac{j}{10} \equiv \frac{i * j}{10}$ .

Tuy nhiên, phép nhân i với j được thực hiện trước phép chia nguyên cho 10, do đó k2 sẽ nhận giá trị -1553 (sai). Do đó, ta phải viết “*long(i)\*j/10*” hoặc “*i \* (j / 10)*”.

Nhận xét: để tránh tràn số, ta nên viết biểu thức sao cho *phép chia (trừ) thực hiện trước phép nhân (cộng)*.