TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN – TIN HỌC

Trần Ngọc Anh

Kỹ thuật lập trình

Đà lạt, 2 - 2016

LỜI MỞ ĐẦU

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 hoạ, hệ điều hành, trí tuệ nhân tạo, ...

Nội dung giáo trình gồm 4 chương:

- 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.
- Chương 2: Giới thiệu ngôn ngữ C++ và cách viết các chương trình đơn giản với các kiểu dữ liệu cơ sở trên C^{++}.
- Chương 3: Trình bày phương pháp lập trình module với hàm, kỹ thuật đệ quy.
- Chương 4: Giới thiệu các kiểu dữ liệu có cấu trúc như mảng, chuỗi, struct và các thuật toán thường gặp trên chúng.

Chắn chắn rằng trong giáo trình sẽ còn nhiều khiếm khuyết, tác giả mong muốn nhận được và rất biết ơn các ý kiến đóng góp quí báu của đồng nghiệp cũng như bạn đọc để bài giảng này có thể hoàn thiện hơn nữa về mặt nội dung cũng như hình thức trong lần tái bản sau.

Đà lạt, 2 - 2016

Tác giả

MỤC LỤC

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.

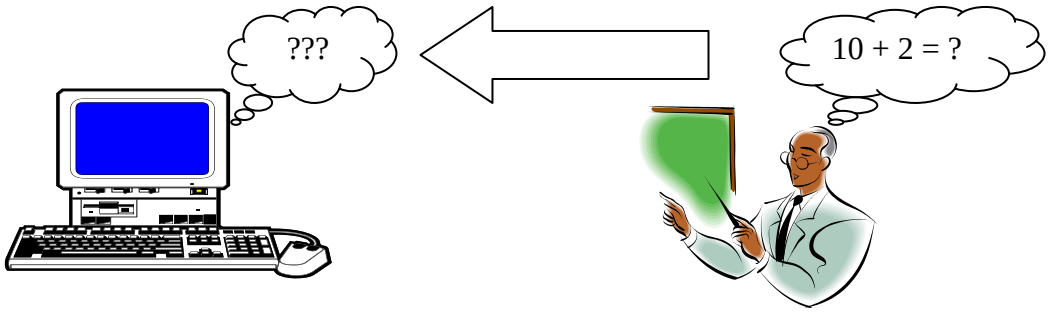
[4] Võ Tiến, *Giáo trình Nhập môn lập trình*, Đại học Đà Lạt, 1997.

CHƯƠNG 1: DỮ LIỆU VÀ THUẬT TOÁN

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 làm cho người giải bài toán 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 lãnh vực: quân sự, quản lý, khoa học, tư vấn, thương mại, …

Chương trình (CT) là một tập các mô tả, các câu lệnh được viết (bởi lập trình viên) theo một trình tự nhất định nhằm hướng dẫn máy tính giải một bài toán đặt ra. Trong các thế hệ máy tính đầu tiên, **lập trình viên** (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. Có hai loại chương trình dịch: a) **trình thông dịch**: dịch *từng câu lệnh* của CT ra ngôn ngữ máy và thực thi cho đến khi kết thúc CT; b) **trình biên dịch** (TBD): dịch *toàn bộ* CT sang ngôn ngữ máy và thực thi CT.

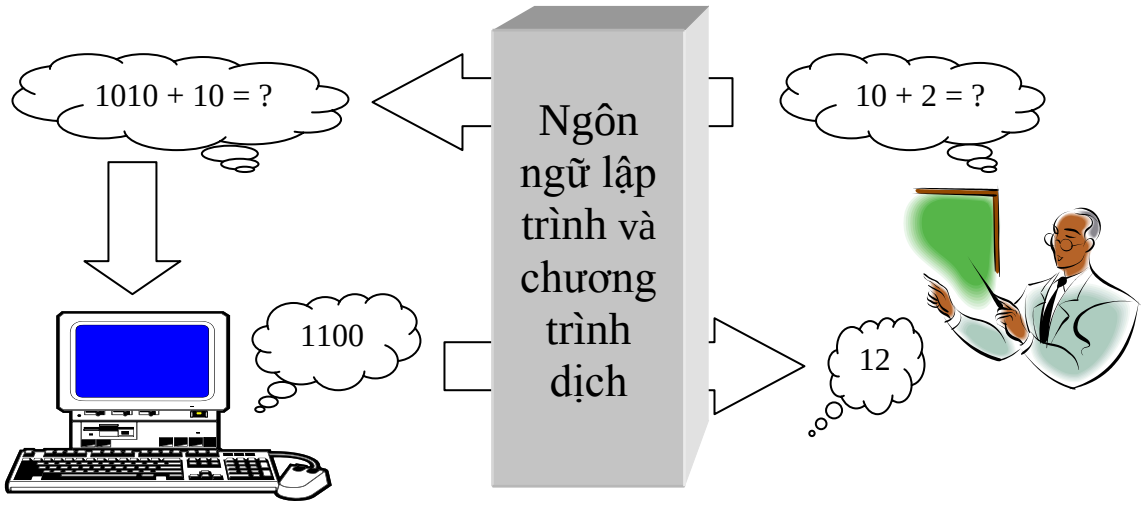


Hình 1.1: Máy tính chỉ hiểu 0 và 1

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 (mức trừu tượng thấp) được gọi là các **NNLT bậc thấp**, các NNLT "xa" với máy (có mức trừu tượng cao) được gọi là các **NNLT bậc cao**. Việc viết CT trên các NNLT bậc cao sẽ nhanh và tự nhiên hơn trên các NNLT bậc thấp.

Các NNLT bậc cao được sử dụng rộng rãi trong thực tế là: Pascal, Basic, C. **C** được phát triển (bởi Dennis Ritchie tại phòng thí nghiệm Bell Telephone vào năm 1972) từ **B** (do Ken Thomson tạo ra). C^{++} được nâng cấp từ C, hỗ trợ thêm phương pháp lập trình hướng đối tượng. Hiện nay, C^{++} là ngôn ngữ được sử dụng rộng rãi.

1.2 Thuật toán



Hình 1.2: NNLT và CT dịch giúp máy tính hiểu được các lệnh giống ngôn ngữ tự nhiên

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.

1.2.1 Định nghĩa:

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.

Thuật toán theo nghĩa chính xác được định nghĩa bằng mô hình máy Turing, mô hình chuẩn Mabkob. 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.

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 $\sqrt{\Delta}/(2a)$ 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).

1.2.2 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:

- Thuật toán nhận dữ liệu *đầu vào*, tính toán và cho 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 cho mọi trường hợp của bài toán (tính *tổng quát*) chứ không phải chỉ áp dụng cho một số trường hợp riêng lẽ.

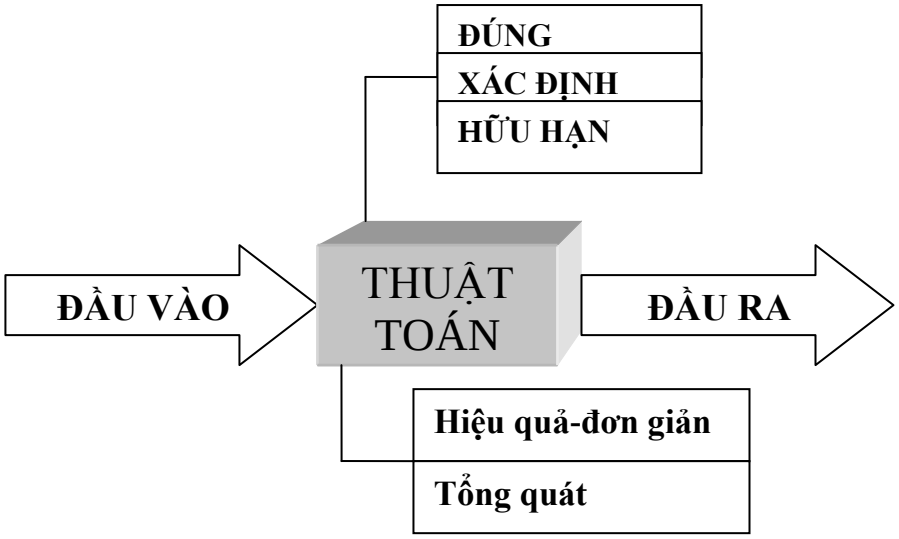
Trong thực tế, khi thiết kế thuật toán thường người ta chỉ quan tâm đến các tính chất: đúng, xác định, hữu hạn. Tính hiệu quả-đơn giản thường rất khó đạt được trọn vẹn: rất khó tìm được một thuật toán tốn ít thời gian, không gian tính toán mà lại đơn giản và dễ hiểu. Tính tổng quát cũng khó đạt được 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.

- **Ngôn ngữ tự nhiên** là ngôn ngữ diễn đạt của con người.

Ví dụ 1.1: Thuật toán tìm ước số chung lớn nhất của hai số nguyên dương



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

- Đầu vào: hai số nguyên dương a, b.
- Đầu ra: ước số chung lớn nhất.

  Bước 1: Nếu a > b thì trừ a một lượng bằng b,
  Nếu b > a thì trừ b một lượng bằng a.
  Bước 2: Nếu a = b thì *ước số chung lớn nhất là a*.
  Bước 3: Ngược lại quay lại Bước 1.

Ví dụ 1.2: Thuật toán tìm tờ đô la có giá trị nhất trong 10 tờ đô la

- Đầu vào: 10 tờ đô la có giá trị khác nhau.
- Đầu ra: tờ đô la có giá trị lớn nhất.

  Bước 1: Lấy tờ đô la đầu tiên bỏ túi.
  Bước 2: Lấy tờ đô la thứ hai.
  Bước 3: Nếu giá trị tờ đô la thứ hai lớn hơn tờ trong túi thì: lấy tờ trong túi trả lại, bỏ tờ thứ hai vào túi.
  Ngược lại, giữ nguyên tờ trong túi.

  …

  Bước 19: Lấy tờ đô la thứ mười.
  Bước 20: Nếu giá trị tờ đô la thứ mười lớn hơn tờ trong túi thì: lấy tờ trong túi trả lại, bỏ tờ thứ mười vào túi.
  Ngược lại, giữ nguyên tờ trong túi.

  Tờ lớn nhất là tờ trong túi.

Ví dụ 1.3: Thuật toán giải phương trình bậc hai $ax^2 + bx + c = 0$, với $a \neq 0$

- Đầu vào: a ($\neq 0$), b, c.
- Đầu ra: các nghiệm nếu có.

  Bước 1: Tính $\Delta = b^2 - 4ac$.
  Bước 2: Biện luận theo Δ

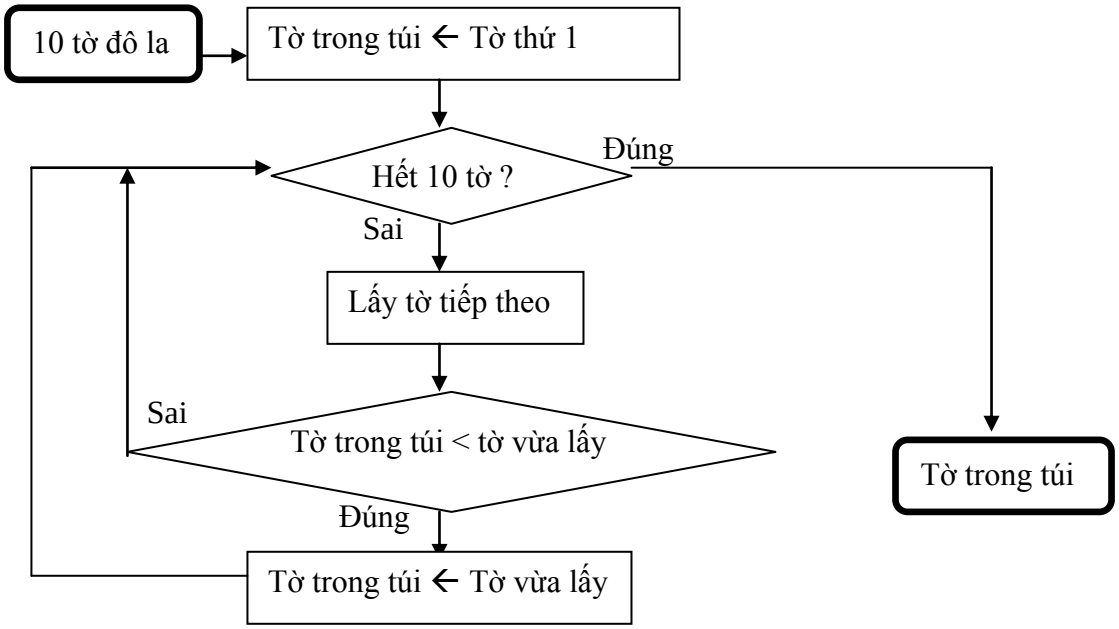
  - Nếu $\Delta > 0$ thì: - Tính $x_1 = \frac{-b-\sqrt{\Delta}}{2a}, x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.
    - Xuất ra x_1, x_2.

- Nếu delta = 0 thì: - Tính $x_0 = \frac{-b}{2a}$.
  - Xuất ra nghiệm kép x_0.
- Nếu delta < 0 thì: Xuất ra "Vô nghiệm".

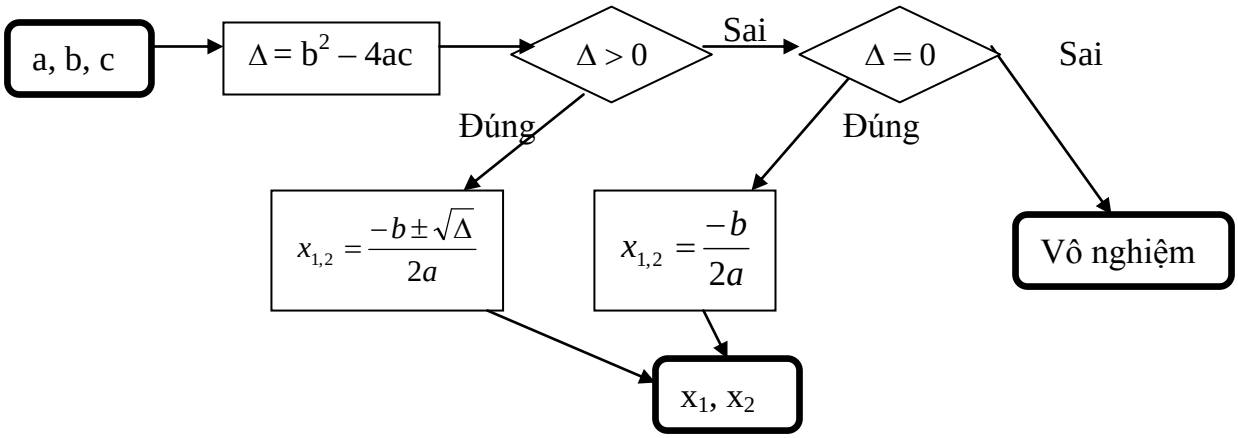
Dùng ngôn ngữ tự nhiên biểu diễn thuật toán sẽ rất dài dòng, không thể hiện rõ cấu trúc thuật toán, đôi lúc còn gây hiểu lầm, khó hiểu.

- **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 chọn lựa, b) *hình chữ nhật* bên trong chứa nội dung xử lý, tính toán, c) *mũi tên* chỉ trình tự thực hiện các thao tác, e) *hình oval* chỉ ra khởi đầu (đầu vào) và kết thúc (đầu ra), ...

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

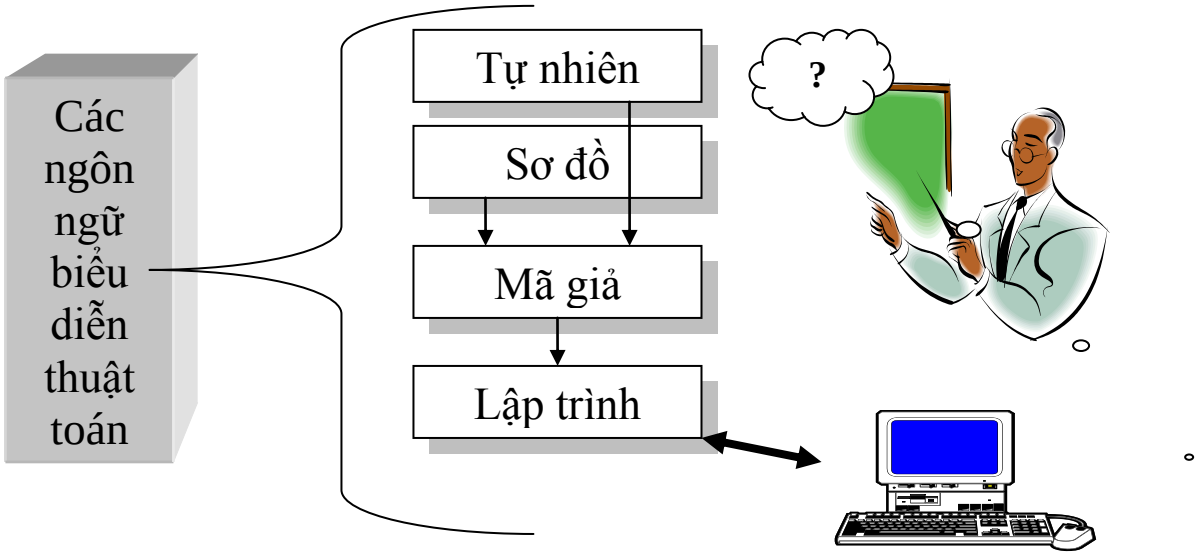


Ví dụ 1.5: Thuật toán giải phương trình bậc hai (a ≠ 0).



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

- **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.



Hình 1.4: Các ngôn ngữ biểu diễn 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:

Biến là nơi lưu trữ giá trị. Mỗi biến đều có một **tên** riêng 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.

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 (**chuỗi**)	Tên một môn học

1.3.1.1. Tên biến:

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.

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 sau:

- **Tên biến phải liên quan đến ý nghĩa của biến.** Lấy ví dụ, các biến lưu trữ các điểm Toán, Lý, Hóa nên được đặt với các tên "DiemToan", "DiemLy", "DiemHoa" chứ không nên là "a", "b", "c" vì chẳng hạn, *biểu thức tính điểm trung bình (a*3 + b*2 + c)/6 gây khó hiểu*.

- **Viết hoa các chữ cái đầu mỗi từ hoặc dùng dấu gạch dưới '_' phân cách các từ.** Chẳng hạn: "HetFile" hoặc "het_file", "TimDuoc" hoặc "tim_duoc".

- **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.** Lấy ví dụ, biến lưu mã số sinh viên sẽ có tên "MSSVien" chứ không phải "MaSoSinhVien", biến lưu trữ diện tích hình chữ nhật nên có tên "S_HCNhat" thay vì "dien_tich_hinh_chu_nhat".

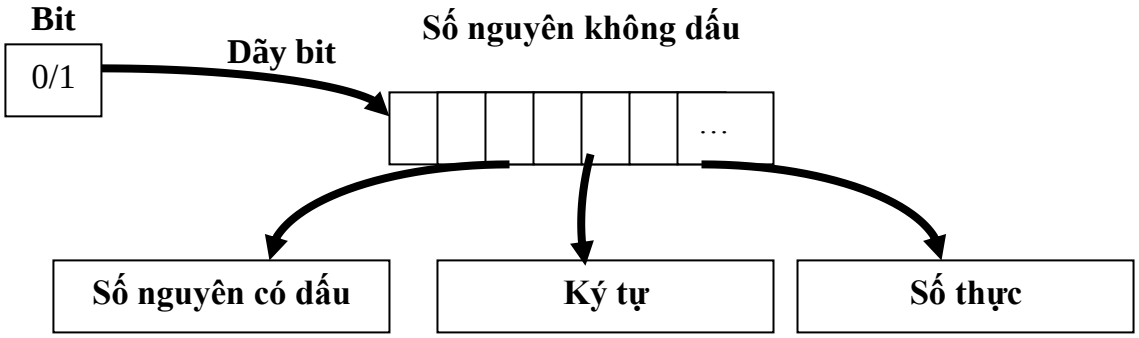
- **Tên biến phải có thêm tiền tố để biết được kiểu biến** *(nếu cần)*. Chẳng hạn, biến lưu trữ mã số sinh viên có kiểu là số nguyên nên có tên là "nguyen_MSSVien", biến lưu trữ mã hóa đơn có kiểu là chuỗi nên có tên là "chuoi_MaHDon".

- **Tuân theo các quy tắc được sử dụng rộng rãi trong giới lập trình.** Lấy ví dụ, các biến số thực nên có tên là "x", "y", "z", …; các biến kiểm soát vòng lặp nên là "i", "j", "k", ….; các biến chỉ số lượng phần tử của tập hợp (số sinh viên, số hóa đơn, …) nên là "m", "n", …

1.3.1.2. Biểu diễn dữ liệu, 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 $\Re$), 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

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, .., 255] cần đến 1 ô nhớ, [0, .., 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… 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’, .., ‘9’, …, ‘A’, ‘B’, .., ‘Z’, …, ‘a’, ‘b’, .., ‘z’, …, ‘*’, ‘+’, ‘@’, …. } 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.

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			$3.4*10^{-38}$ .. $3.4*10^{38}$	4

Lớn	$1.7*10^{-308}$ .. $1.7*10^{308}$	8
Rất lớn	$3.4*10^{-4932}$ .. $1.1*10^{4932}$	10

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 ← *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.

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: +, –.

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)
tong_tien – 2	– 2	Nguyên
(a + b) * 2	72	Nguyên
2.5	2.5	Thực
'W'	'W'	Ký 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.

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

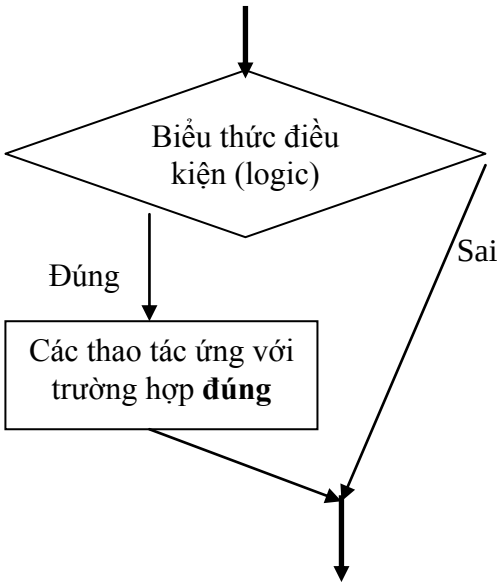
Ví dụ 1.8: Giả sử giá trị hiện hành của các biến a, b, lần lượt là 2, 3; kết quả của một số biểu thức logic cho trong bảng sau:

Biểu thức logic	Trị logic của biểu thức
(5 > 1) hoặc (6 < 1)	Đúng
(2 = 1) và (5 = 5)	Sai

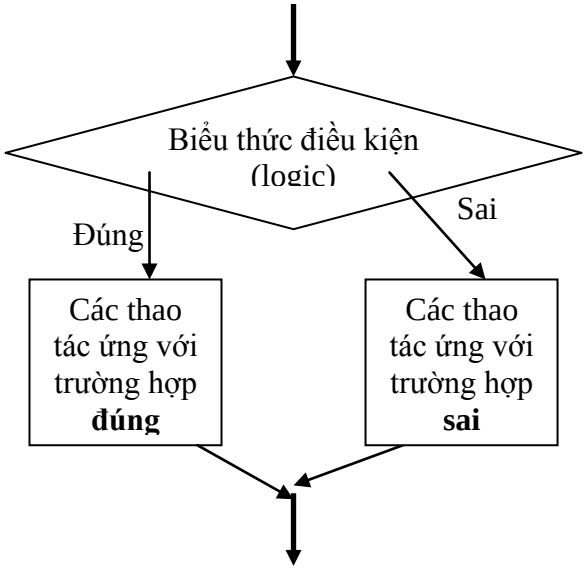
phủ định (5 = 4)	Đúng
(a + b) < PI	Sai (xem 1.3.2)

1.4.2. Cấu trúc rẽ nhánh:

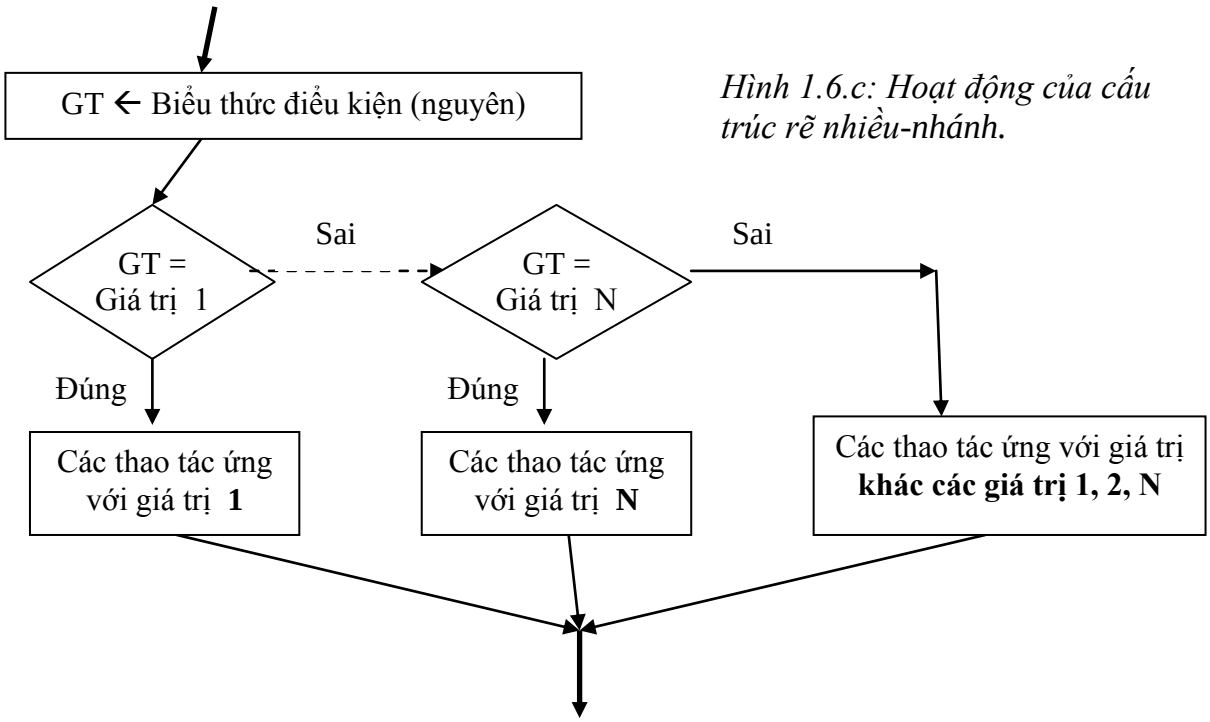
Cấu trúc rẽ nhánh phá vỡ trình tự thi hành tuần tự của CT dựa trên kết qủa của biểu thức điều kiện (biểu thức logic hoặc biểu thức nguyên). Rẽ nhánh dựa trên kết quả của biểu thức logic gồm có: rẽ nhánh-đơn, rẽ nhánh-đôi. Rẽ nhánh dựa trên kết quả của biểu thức nguyên được gọi là rẽ nhiều-nhánh. Có thể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



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



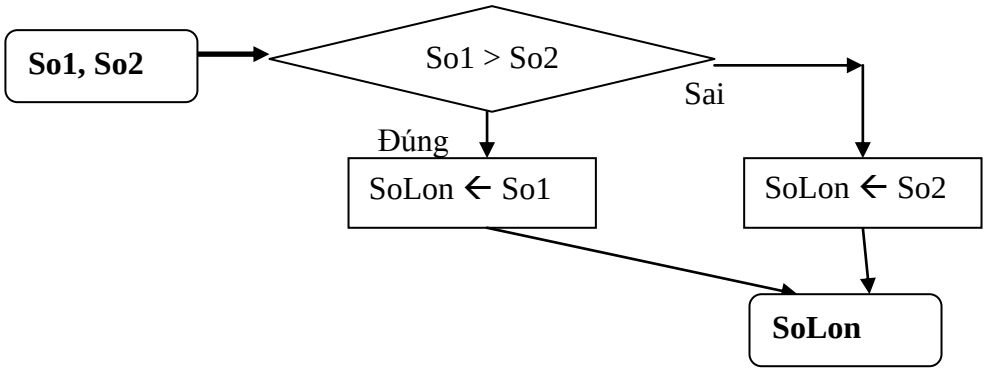
Hình 1.6.b: Hoạt động của cấu trúc rẽ nhánh-đôi.



Hình 1.6.c: Hoạt động của cấu trúc rẽ nhiều-nhánh.

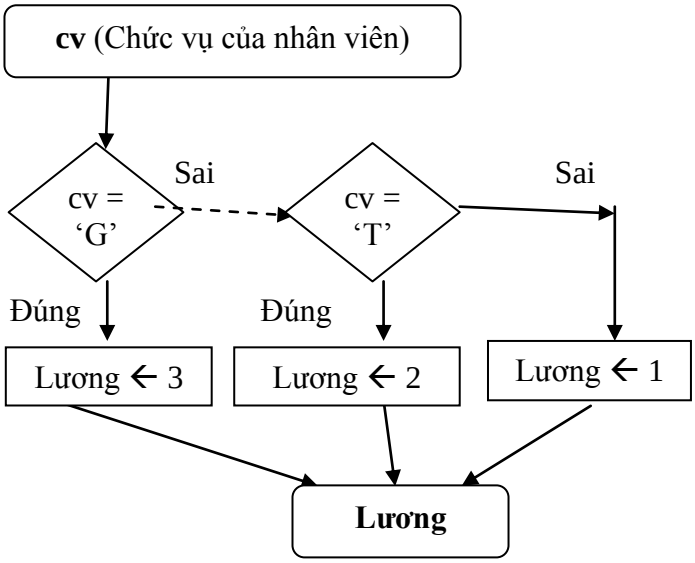
- **Cấu trúc rẽ nhánh-đơn**: CT tính giá trị của biểu thức điều kiện và rẽ nhánh nếu biểu thức đúng, ngược lại CT thực hiện tiếp lệnh sau cấu trúc rẽ nhánh.
- **Cấu trúc rẽ nhánh-đôi**: CT tính giá trị của biểu thức điều kiện và rẽ theo một trong hai nhánh ứng với giá trị đúng hoặc sai.
- **Cấu trúc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

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

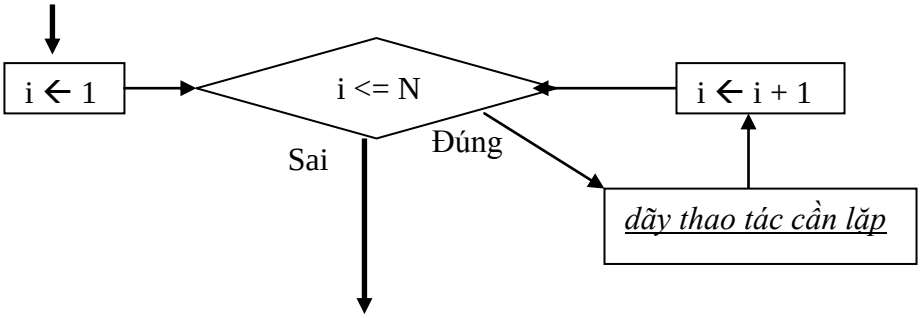
Ký hiệu	Ý nghĩa	Lương
G	Giám đốc	3
T	Trưởng phòng	2
Khác	Chức vụ khác	1



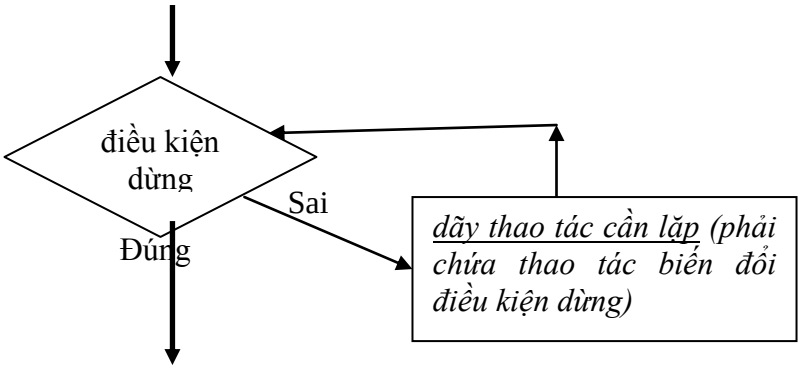
1.4.3. Cấu trúc lặp:

Cấu trúc lặp điều khiển việc lặp đi lặp lại các thao tác. Có hai loại cấu trúc lặp.

- Lặp xác-định: biết trước số lần lặp.
- Lặp không-xác-định: không biết (khó tính) số lần lặp.

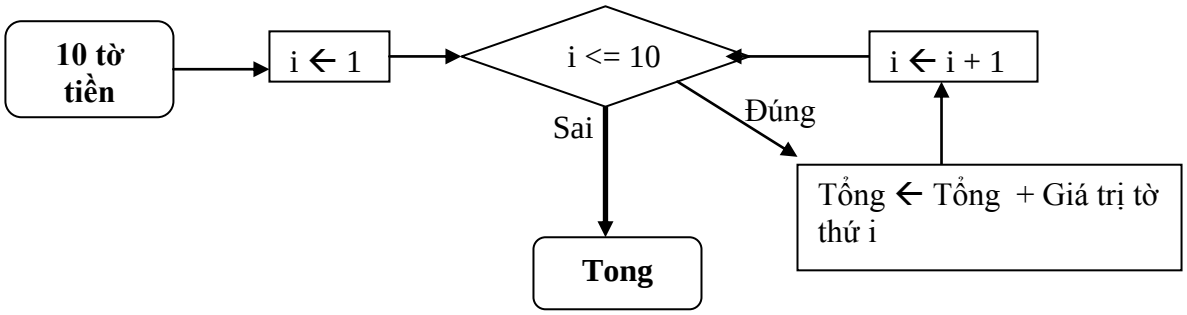


Hình 1.7.a: Hoạt động của cấu trúc lặp xác-định
(trong đó: N là số lần lặp, i là chỉ số của bước lặp hiện tại).

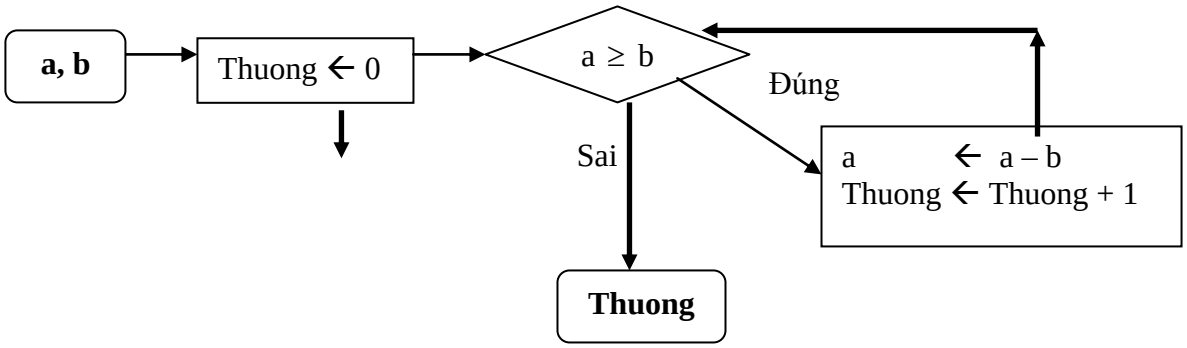


Hình 1.7.b: Hoạt động của cấu trúc lặp không-xác-định.

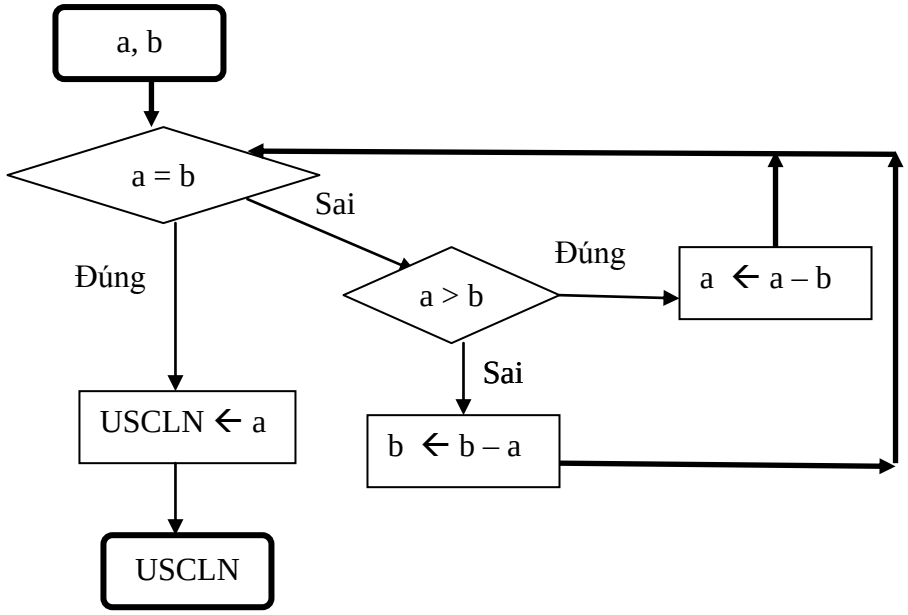
Ví dụ 1.11: Thuật toán tính tổng giá trị 10 tờ tiền dùng cấu trúc lặp xác-định



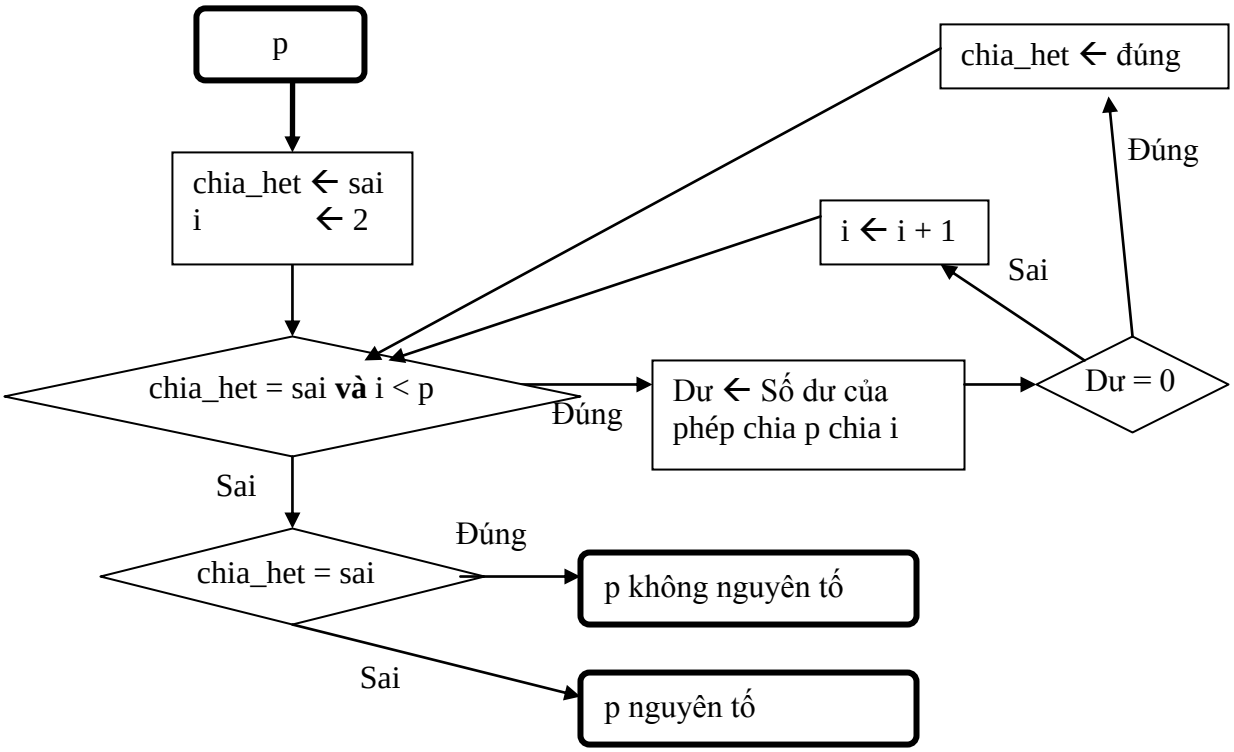
Ví dụ 1.12: Thuật toán chia a cho b lấy thương bằng các phép trừ dùng cấu trúc lặp không-xác-định



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.



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



BÀI TẬP

Viết các thuật toán sau (bằng các ngôn ngữ tự nhiên, sơ đồ)

1) Cho số nguyên dương n

   a. Tính $S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + ... + \frac{(-1)^{n+1}}{n}$, n >1.

   b. Tính $A = \left(1 + \frac{1}{1^2}\right)\left(1 + \frac{1}{2^2}\right)...\left(1 + \frac{1}{n^2}\right)$.

   c. Tính $A = \sqrt{2 + \sqrt{2 + ... + \sqrt{2}}}$ (n dấu căn).

2) Viết thuật toán nhập x, epsilon từ bàn phím

   a. Tính $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ... + \frac{(-1)^n x^{2n+1}}{(2n+1)!}$ sao cho $\left|\frac{x^{2n+1}}{(2n+1)!}\right| < \text{epsilon}$.

   b. Tính $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$ sao cho $\left|\frac{x^n}{n!}\right| <= \text{epsilon}$.

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

4) Tìm số nguyên k lớn nhất thỏa $P^k < n$.

5) Giải và biện luận phương trình bậc nhất.

6) Biết số tiền gởi ngân hàng của một người là 60000 USD, lãi suất một năm là 1.2%. Hãy tính số tiền người đó sẽ có sau Y năm.

7) Cho số nguyên dương N

   a. Tính số chữ số của N.

   b. Đảo ngược N. Ví dụ: N = 195 → 591.

   c. Tính số Fibonacci thứ N. Số Fibonacci được định nghĩa như sau:
   $F_0=F_1=1$
   $F_i = F_{i-1} + F_{i-2}$ (i>1).

   d. Phân tích N thành tích các thừa số nguyên tố. Ví dụ: $60 = 2^2 * 3 * 5$.

   e. Xuất tất cả các cặp số nguyên dương x, y sao cho x + 2y = N.

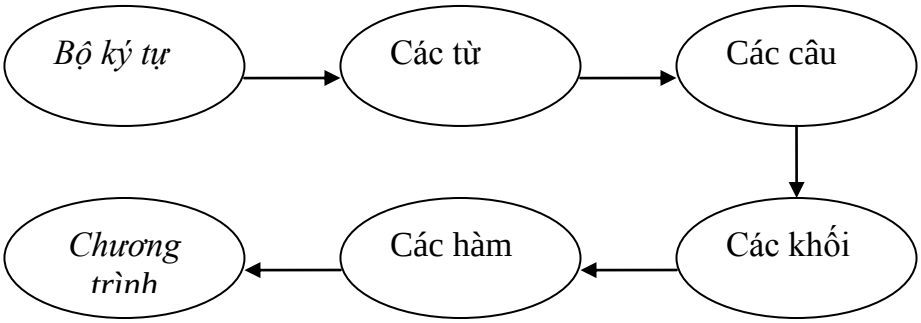
8) Hiển thị tất cả các phương án đổi tờ tiền 100000 thành các tờ tiền 20000, 10000, 5000. Ví dụ: 2 tờ 20000, 0 tờ 10000 và 0 tờ 5000 là một phương án.

9) Giải hệ phương trình bậc nhất hai ẩn.

CHƯƠNG 2: LẬP TRÌNH CƠ BẢN

2.1 Các thành phần cơ bản của ngôn ngữ C^{++}

*Thành phần cơ bản nhất của ngôn ngữ C^{++} gồm bộ ký tự. Các ký tự tạo thành các từ, các từ tạo thành các câu, các câu tạo thành các khối, các khối tạo thành các hàm, và các hàm tạo thành **CT**.*



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: (,), {, }, ….

2.1.2. 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.

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

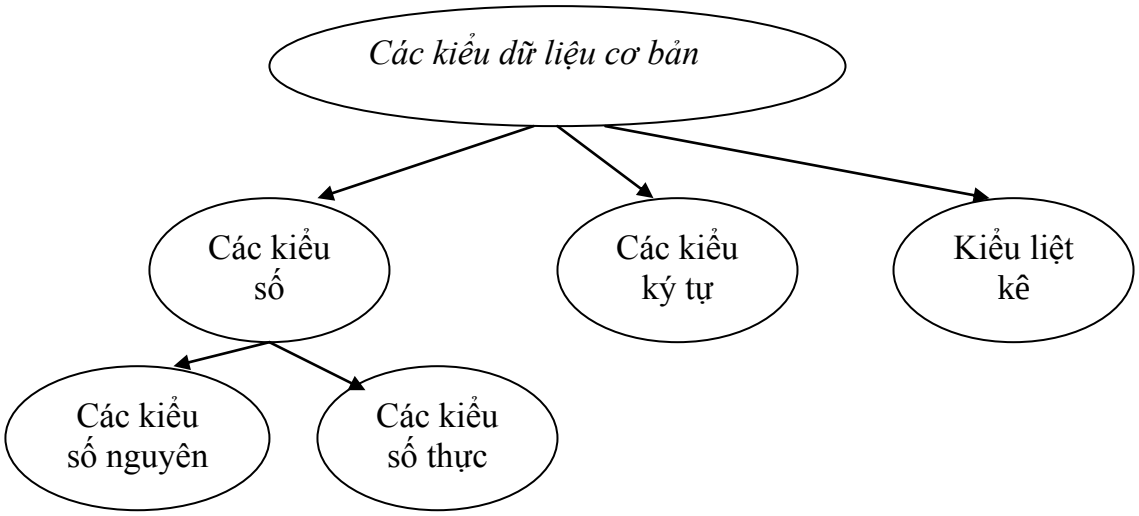
Tổ hợp các từ hình thành các câu lệnh. Các câu lệnh bao gồm: lệnh gán, lệnh điều khiển, lệnh gọi hàm, …

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.

2.2 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

2.2.1 Các kiểu số:

Tên các kiểu số được liệt kê trong bảng 2.1.

Bảng 2.1: Tên các kiểu số trong C++

Kiểu số nguyên			**Tên kiểu trong C++**
Nhỏ	Ngắn	Không dấu	unsigned char
		Có dấu	char

	Dài	Không dấu	unsigned int
		Có dấu	int
Lớn		Không dấu	unsigned long
		Có dấu	long
Kiểu số thực			**Tên kiểu trong C^{++}**
Nhỏ			float
Lớn			double
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*).

✍ Cần cẩn thận trong việc áp dụng phép toán so sánh = = trên số thực vì "số thực trong Toán học là vô hạn, liên tục còn trong máy tính thì nó rời rạc, hữu hạn". Nên thay so sánh "a = = b" bởi "$|a-b| < \text{EPSILON}$" (EPSILON là một hằng số có giá trị nhỏ được định nghĩa trước, chẳng hạn là 0.000001).

Ví dụ 2.1: Biểu thức

$$\frac{10}{\sqrt{5}} = \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}}$$

là đúng về mặt Toán học, nhưng trong máy tính biểu thức

$$\frac{10}{\sqrt{5}} == \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}}$$

cho ra kết quả sai vì sai số ở vế trái (một lần chia và lấy căn bậc 2) ít hơn vế phải (10 lần). Biểu thức trên phải được thay bằng:

$$\left| \frac{10}{\sqrt{5}} - \left(\frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} + \frac{1}{\sqrt{5}} \right) \right| < \text{EPSILON}.$$

✋ Tràn số và biểu diễn "quay vòng"

Khi giá trị của một số vượt quá chặn trên miền xác định của kiểu số (tràn số) thì xảy ra tình trạng "quay vòng" tính từ chặn dưới của miền xác định

Giả sử ta gán giá trị 32770 cho một biến a kiểu int có miền xác định [-32768..32767]. Vì 32770 vượt quá chặn trên miền xác định của kiểu int (32767) nên xảy ra tràn số. Lúc

này, sẽ diễn ra sự "quay vòng": 32768 (= chặn trên + 1) sẽ trở thành -32768 (chặn dưới), 32769 (= chặn trên + 2) trở thành -32767 (= chặn dưới + 1),32770 trở thành −32766. Do đó, a sẽ có giá trị -32766.

2.2.2 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.

Bảng 2.2: Biểu diễn ký tự bằng hai kiểu số nguyên ngắn: unsigned char, char.

Mã ASCII	Giá trị nguyên tương ứng khi lưu trữ ký tự bằng kiểu:	
	unsigned char (miền xác định: [0..255])	char (miền xác định: [-**128**..*127*])
0	0	0
1	1	1
..	..	..
127	127	127
128 (=*127* + 1)	128	-128 (= **-128**)
129 (=*127* + 2)	129	-127 (= **-128** + 1)
..	..	..
255 (=*127* + 128)	255	-1 (**-128** + 127)

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.

2.3 Khai báo hằng và kiểu liệt kê

2.3.1 Khai báo hằng:

Tên khai báo hằng (thường được viết hoa) tuân theo phong cách đặt tên biến. Các hằng được khai báo trước khi sử dụng, theo hai cách sau:

#define tên_hằng giá_trị

hoặc ***const kiểu_dữ_liệu tên_hằng = giá_trị;***

Ví dụ 2.2: một số khai báo hằng

```
#define   PI              3.1416
#define   EPSILON         0.000001
#define   GIAY_TREN_GIO   3600
#define   GIAY_TREN_PHÚT  60
```

```
#define        KY_TU                    'a'
```

hoặc:

```
const   float   PI                          =       3.1416;
const   float   EPSILON                     =       0.000001;
const   unsigned int GIAY_TREN_GIO          =       3600;
const   unsigned int GIAY_TREN_PHÚT         =       60;
const   char            KY_TU               =       'a';
```

Các khai báo hằng sai:

```
#define   PI   3.1416;
const      float PI = 3.1416,  EPSILON= 0.000001;
```

do: a) không có dấu ';' sau khai báo hằng bằng từ khóa #define, b) không thể khai báo cùng lúc nhiều hằng.

Bảng 2.3: Một số hằng số và hằng ký tự

Cách viết	**Giá trị**	**Cách viết**	**Giá trị**
'\''	Ký tự '	'\0'	Ký tự \0 (null)
'\"'	Ký tự "	'\t'	Ký tự Tab
'\\'	Ký tự \	'\b'	Ký tự Backspace
'\n'	Ký tự \n (xuống dòng)	*'\141'*	Ký tự 'a': mã ASCII là 97 đổi sang hệ bát phân (hệ 8) là $(141)_8$
0345	Số 229 (biểu diễn trong hệ 8 là 345, 229 = $3*8^2 + 4*8^1 + 5$)	1234L	Số nguyên long 1234
0xA5	Số 165 (biểu diễn trong hệ 16 là A5, 165 = $A*16^1 + 5$; với A là 10, B là 11, .., F là 15)	1.5e+23	$1.5 * 10^{23}$

2.3.2 Kiểu liệt kê:

Một kiểu liệt kê là một tập hợp các hằng định danh số nguyên có cùng ý nghĩa. Kiểu liệt kê là kiểu con của kiểu số nguyên.

Cú pháp định nghĩa kiểu liệt kê:

enum tên_kiểu_liệt_kê { tên_hằng_1 = giá_trị_1, tên_hằng_2 = giá_trị_2, .., tên_hằng_n = giá_trị_n };

Nếu không xác định *giá_trị_1* thì *giá_trị_1* được mặc định là 0. Nếu không xác định *giá_trị_i* thì *giá_trị_i* được mặc định là: *giá trị của hằng được liệt kê trước đó cộng 1*.

Ví dụ 2.3: một số khai báo kiểu liệt kê

Khai báo	**Các giá trị tương ứng các hằng**
enum color { RED, **BLUE**, GREEN };	0, **1**, 2
enum color { RED = 100, BLUE = 222, GREEN = 1};	100, 222, 1
enum color { RED = 100, BLUE, GREEN = 1};	100, 101, 1
enum cac_ngay { C_NHAT, T_HAI, T_BA, T_TU, T_NAM, T_SAU, T_BAY };	0, 1, 2, 3, 4, 5, 6

2.4 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
```

✍ 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).*

2.4.2 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	||

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) || (6 < 1)		
(**5** && (4 > 2))		
(**5** && (4 < 2))		
! (5 = = 4)		
(a + b) < 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?*).

2.4.3 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***.

Lấy ví dụ, biểu thức *1/100* có kết quả là *0* (do hiểu '/' 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.

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)*.

2.5 Cấu trúc một chương trình C^{++} đơn giản

2.5.1. Các lệnh nhập/xuất căn bản:

Hầu hết mọi CT đều cần dữ liệu đầu vào và đầu ra. Thông thường dữ liệu đầu vào được nhập từ bàn phím và dữ liệu đầu ra được xuất lên màn hình. Chẳng hạn, trong CT giải phương trình bậc hai, giá trị các hệ số a, b, c sẽ được nhập vào từ bàn phím và kết quả nghiệm được xuất ra màn hình. Lệnh nhập từ bàn phím là scanf, lệnh xuất ra màn hình là printf, có cú pháp như sau:

scanf (chuỗi định dạng, danh sách địa chỉ các biến);

printf (chuỗi định dạng, danh sách các biểu thức);

Để lấy địa chỉ của biến, ta dùng toán tử & trước tên biến.

Chuỗi định dạng của câu lệnh nhập (không chứa khoảng trắng và các ký tự phân cách) được cấu thành từ các ký tự chuyển dạng cho trong bảng 2.6 (mỗi ký tự chuyển dạng luôn có ký hiệu '%' ở trước).

Bảng 2.6: Các ký tự chuyển dạng nhập

Ký tự	Chuyển dạng
d	Số nguyên kiểu ***int*** (số nguyên int)
ld	Số nguyên long
o	Số nguyên int ở hệ bát phân
lo	Số nguyên long ở hệ bát phân
c	***Ký tự***
x	Số nguyên int hệ thập lục phân.
lx	Số nguyên long hệ thập lục phân.

f hoặc e	Số thực ***float***
lf hoặc le	Số thực double
s	Chuỗi ký tự (không chứa: khoảng trắng, tab, xuống dòng, …)

Các ký tự chuyển dạng trong bảng 2.7 và các ký tự khác (để giúp làm rõ ý nghĩa của dữ liệu xuất) tạo ra chuỗi định dạng cho câu lệnh xuất.

Bảng 2.7: Các ký tự chuyển dạng xuất

Ký tự	**Chuyển dạng**
d	Số nguyên hệ thập phân có dấu
u	Số nguyên hệ thập phân không dấu
o	Số nguyên hệ bát phân có dấu
c	Ký tự
x	Số nguyên hệ thập lục phân có dấu (các ký tự trong hệ viết thường).
X	Số nguyên hệ thập lục phân có dấu (các ký tự trong hệ viết hoa).
f	Số thực có dấu
s	Chuỗi ký tự

Ví dụ 2.7: Đoạn CT (a) nhập xuất một số nguyên, một số thực và một ký tự; (b) tính tổng và tích hai số thực a, b

```
int i;
float       fp;
char        c;
printf (“\n\n Hay nhap mot: so nguyên, \t mot  so thuc va \t mot ky tu. \n”);
scanf (“%d%f%c", &i, &fp, &c);
printf (“Xuat ra: so nguyen = %d \t, so thuc = %f \t, ky tu = %c”, i, fp, c);
int    a, b;
printf (“a = ? ”);  scanf (“%d”, &a);
printf (“b = ? ”);  scanf (“%d”, &b);
printf (“%d + %d = %d.\n”, a, b, a+b);
printf (“%d * %d = %d.”, a, b, a*b);
```

2.5.2. Cấu trúc một CT C^{++} đơn giản:

Cấu trúc một CT C^{++} đơn giản:

```
Nạp các thư viện
Khai báo các hằng
int      main (int argc, char* argv[])
```

```
{
    Các khai báo biến
    Các lệnh nhập dữ liệu
    Các lệnh tính toán và xử lý
    Các lệnh xuất dữ liệu
    return 0;
}
```

gồm: khai báo sử dụng các thư viện, các khai báo hằng, và hàm main.

CT sẽ bắt đầu bằng việc nạp vào các thư viện, khai báo các hằng, và thực thi các lệnh trong hàm main: nhập dữ liệu, tính toán xử lý trên dữ liệu và xuất dữ liệu kết quả. Ở đây, ta chấp nhận cú pháp khai báo hàm main: "**int main (int argc, char* argv[])"** và câu lệnh "**return 0;**" (xem chương 3).

Thư viện là tập các thao tác đã được viết sẵn, mỗi thao tác có một tên. Các thao tác này thường phức tạp nhưng hay được sử dụng, do đó, chúng được viết sẵn để cho LTV sử dụng. Chẳng hạn, trước khi kết thúc, CT thường chờ người sử dụng nhập vào một phím. Thao tác này là **getch()** được viết sẵn trong thư viện **conio**, do đó, ta phải khai báo thư viện **conio.h**: #include "conio.h". Thư viện **stdafx.h** là cần thiết để thực thi CT. Lệnh printf và scanf được cho trong thư viện "**stdio.h**".

Ví dụ 2.8: CT tính điểm trung bình Toán, Lý cho học sinh

- Nhập điểm Toán, Lý (Toan, Ly)
- Tính điểm trung bình (TB)
- Xuất TB

```
#include   "stdafx.h"
#include   "conio.h"
#include   "stdio.h"

#define      HESOTOAN      3
#define      HESOLY        2

int main(int argc, char* argv[])
{
      float  Toan;
      float  Ly;
      float  TB;

      printf ("Diem Toan = ?");
      scanf ( "%f", &Toan);
      printf ("Diem Ly = ?");
      scanf ( "%f", &Ly);
      TB = (Toan * HESOTOAN + Ly * HESOLY) / (HESOTOAN + HESOLY);
      printf( "Diem trung binh = %f", TB);
```

```
        getch (); return 0;
}
```

CT trên là đúng, nhưng không trong sáng: các biến, hằng được khai báo tùy tiện, rải rác trong CT. ✍ Để tạo ra một CT trong sáng, ta nên đặt các khai báo hằng bên trên hàm main (sau các chỉ thị #include), các khai báo biến tập trung ở đầu thân hàm main. *Hãy viết lại CT trên cho trong sáng hơn, bài tập.*

Ví dụ 2.9: CT đổi thời gian ở dạng giây sang dạng giờ:phút:giây

- Nhập thời gian ở dạng giây (giay_nhap_vao)
- Đổi ra giờ (gio), và lấy giây dư (giay_du)
- Đổi giây dư ra phút và giây (phut, giay)
- Xuất gio, phut, giay

```
/*  Chương trình đổi  thời gian ở dạng giây
    sang  dạng giờ:phút:giây.
*/
#include "stdafx.h"                 // cần để chạy CT
#include “conio.h”                 // chứa thao tác getch()
#include “stdio.h”                 // chứa các thao tác nhập, xuất
#define     GIAY_TREN_PHUT      60      // Số giây quy đổi từ một phút
#define     GIAY_TREN_GIO       3600    // Số giây quy đổi từ một giờ
int main(int argc, char* argv[])    /* Hàm chính của chương trình */
{
        int     giay_nhap_vao, gio, phut, giay, giay_du;
        printf( “Nhap vao so giay:\t” );    scanf ( “%d”, &giay_nhap_vao);
        gio             = giay_nhap_vao  /  GIAY_TREN_GIO;
        giay_du         = giay_nhap_vao % GIAY_TREN_GIO;        // Phép chia dư
        phut            = giay_du   /   GIAY_TREN_PHUT;
        giay            = giay_du  %  GIAY_TREN_PHUT;
        printf ( “\nThoi  gian quy doi la: %d h: %d m : %d s ”, gio, phut, giay);
        getch ();                                   // Dừng và chờ ấn một phím
        return 0;
}
```

Trong đoạn CT trên, ta thấy có các dòng chữ xuất hiện sau (trong) cặp dấu “//” (“/*” “*/”) . Đó là các chú thích nhằm mô tả ý nghĩa các dòng lệnh, các khai báo … Các chú thích sẽ bị bỏ đi khi biên dịch CT.

Bài tập: Viết CT đổi thời gian ở dạng dạng giờ:phút:giây sang dạng giây.

✋ Từ đây đến cuối giáo trình này, ta quy ước “#include …” thay thế cho các chỉ thị nạp các thư viện cần thiết đã biết ý nghĩa, khi cần nạp thêm thư viện mới chưa biết ý nghĩa ta sẽ viết chỉ thị nạp thư viện đó ở bên dưới.

Ví dụ 2.10: CT tính tổng diện tích hình chữ nhật và hình vuông

- Nhập cạnh hình vuông (canh) và chiều dài, chiều rộng của hình chữ nhật (c_dai, c_rong)
- Tính diện tích hình vuông (dtich_hvuong)
- Tính diện tích hình chữ nhật (dtich_hcn)
- Tính tổng diện tích (tong_dtich)
- Xuất tong_dtich

```
#include …
int main(int argc, char* argv[])
{
        // Khai báo các biến.
        int      c_dai, c_rong, canh, dtich_hcn, dtich_hvuong, tong_dtich;

        // Nhập dữ liệu: chiều dài, chiều rộng hình chữ nhật, cạnh hình vuông.
        printf( "Nhap vao chieu dai va chieu rong hinh chu nhat:\n" );
        scanf ( "%d %d", &c_dai, &c_rong);
        printf( "Nhap vao canh hinh vuong:\t" );
        scanf ( "%d", &canh);
        dtich_hcn      = c_dai * c_rong;              // tính diện tích hình chữ nhật
        dtich_hvuong= canh  * canh;                   // tính diện tích hình vuông
        tong_dtich     = dtich_hcn + dtich_hvuong;    // tính tổng diện tích

        // Xuất diện tích hình chữ nhật, hình vuông và tổng diện tích.
        printf( "\nS_hcnhat dai %d, rong %d la: \t%d", c_dai, c_rong, dtich_hcn);
        printf( "\nS_hvuong canh %d la: \t %d", canh, dtich_hvuong);
        printf( "\nTong dien tich la: \t%d", tong_dtich);

        getch (); return 0;
}
```

Bài tập: Viết CT: (a) tính tổng chu vi của hình chữ nhật và hình vuông, hình thoi và hình tròn; (b) tính tổng diện tích của hình thoi và hình tròn.

2.6 Các lệnh điều khiển

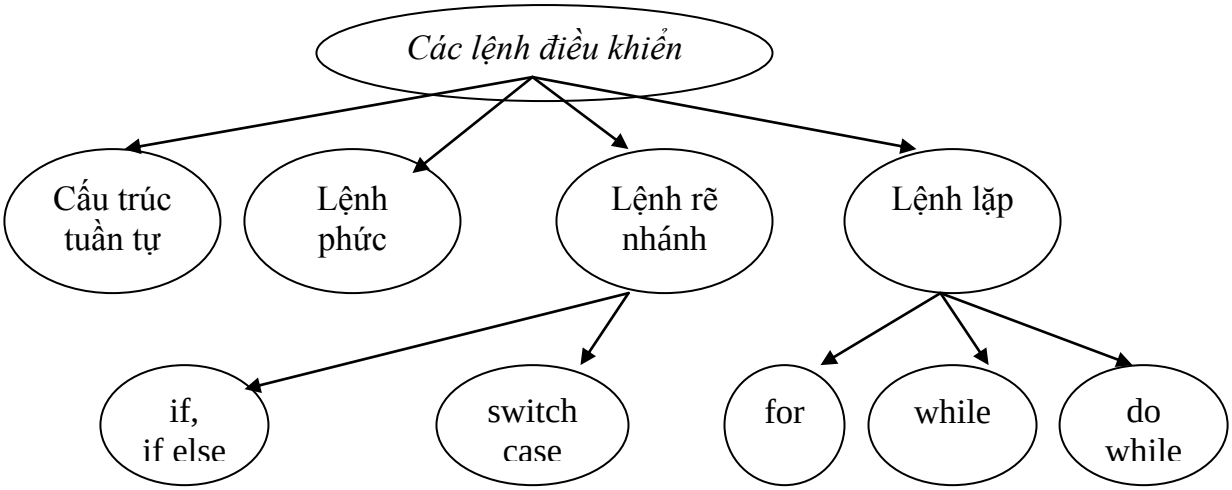
Mục này, ta xét đến các lệnh điều khiển trong C^{++}. Chúng là các thể hiện trên C^{++} của ba cấu trúc cơ bản: tuần tự, rẽ nhánh và lặp.

2.6.1 Cấu trúc tuần tự:

Đây là cấu trúc mặc định (được hiểu ngầm) trong C++. Các câu lệnh được thực hiện tuần tự từ trên xuống dưới theo đúng trình tự xuất hiện của chúng trong CT.

2.6.2 Lệnh phức (khối lệnh):

Lệnh phức là tập các câu lệnh được nhóm lại thành khối trong cặp '{', '}'.



Hình 2.3 Các lệnh điều khiển

Cú pháp lệnh phức:

```
{       câu_lệnh_1;
        …
        câu_lệnh_N;
}
```

Có thể xem lệnh phức là một lệnh duy nhất. Từ đây về sau khi viết "lệnh" ta hiểu đó là lệnh đơn hay lệnh phức; lệnh gán, hay lệnh điều khiển.

2.6.3 Lệnh rẽ nhánh:

2.6.3.1. Lệnh rẽ nhánh-đơn:

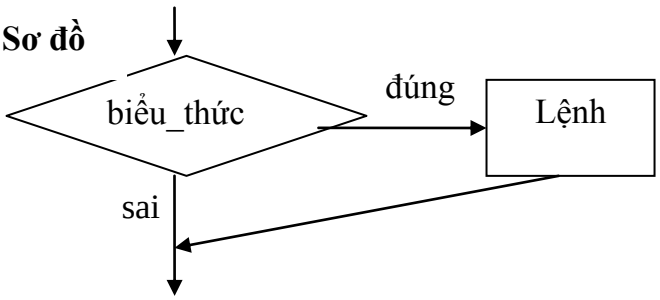
Cú pháp

if (biểu_thức) Lệnh;

Ý nghĩa

Nếu biểu_thức có giá trị đúng thì **Lệnh** được thực hiện.

Sơ đồ

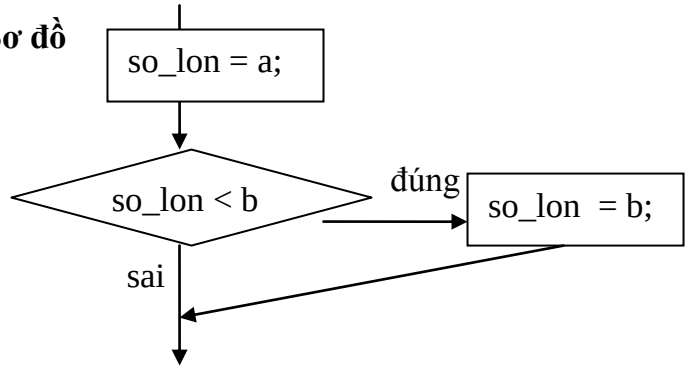


Ví dụ 2.11: Tìm số lớn nhất trong hai số a, b

Các câu lệnh

```
so_lon = a;
if (so_lon < b) so_lon = b;
```

Sơ đồ



✋ Ta so sánh hai đoạn CT (A, B) tìm số lớn nhất và nhỏ nhất trong ba số a, b, c.

```
// A
if (b <= c) && (c <= a))     {
        so_lon  = a;
        so_nho = b;
}
if (c <= b) && (b <= a))     {
        so_lon  = a;
        so_nho = c;
}
if (a <= c) && (c <= b))     {
        so_lon  = b;
        so_nho = a;
}
if (c <= a) && (a <= b))     {
        so_lon  = b;
        so_nho = c;
}
if (a <= b) && (b <= c))     {
        so_lon  = c;
        so_nho = a;
}
if (b <= a) && (a <= c))     {
        so_lon  = c;
        so_nho = b;
}
```

```
// B
so_lon  = a;
if (so_lon < b) so_lon = b;
if (so_lon < c) so_lon = c;
so_nho  = a;
if (so_nho > b) so_nho = b;
if (so_nho > c) so_nho = c;
```

Hiệu quả:

- A: 6 phép so sánh phức tạp
- B: 4 phép so sánh đơn giản

B trong sáng và ngắn gọn hơn A. Trong B có áp dụng chiến lược chia bài toán thành hai bài toán con:

- Tìm số lớn nhất trong 3 số:
  - Tìm số lớn nhất của hai số a, b (so_lon).
  - Tìm số lớn nhất của số so_lon và c.
- Tìm số nhỏ nhất trong 3 số:
  - Tìm số nhỏ nhất của hai số a, b (so_nho).
  - Tìm số nhỏ nhất của số so_nho và c.

Bài tập: Viết và so sánh các CT tìm số lớn nhất và nhỏ nhất của 4 số a, b, c, d theo hai dạng A, B.

Ví dụ 2.12: CT tìm điểm trung bình lớn nhất của hai học sinh khi biết điểm Toán, Lý của từng học sinh.

- Nhập điểm Toán, Lý của hai học sinh (Toan_1, Ly_1, Toan_2, Ly_2)
- Tính điểm trung bình của hai học sinh (TB_1, TB_2) [xem ví dụ 2.8]
- Xác định điểm trung bình lớn nhất (TB_Lon_Nhat) của TB_1, TB_2 [xem ví dụ 2.11]

```
#include …
#define     HESOTOAN     3
#define     HESOLY       2
int main(int argc, char* argv[]) {
      float   Toan_1, Ly_1, TB_1, Toan_2, Ly_2, TB_2, TB_Lon_Nhat;
      printf (“Diem Toan, Ly cua cac hoc sinh”);
```

```
        scanf ("%f%f%f%f", &Toan_1, &Ly_1, &Toan_2, &Ly_2);
        TB_1 = (Toan_1*HESOTOAN+Ly_1*HESOLY) / (HESOTOAN + HESOLY);
        TB_2 = (Toan_2*HESOTOAN+Ly_2*HESOLY) / (HESOTOAN + HESOLY);
        TB_Lon_Nhat = TB_1;
        if (TB_Lon_Nhat < TB_2)   TB_Lon_Nhat = TB_2;
        printf( "Diem trung binh lon nhat = %f", TB_Lon_Nhat);
        getch (); return 0;
}
```

✍ Từ ví dụ 2.11 và 2.12, ta thấy rằng, để giải quyết một bài toán trong thực tế (thường phức tạp) cần phải áp dụng chiến lược chia để trị "*chia bài toán thành các bài toán con, giải các bài toán con đó, rồi tổng hợp chúng lại"* (xem chương 3).

2.6.3.2. Lệnh rẽ nhánh-đôi:

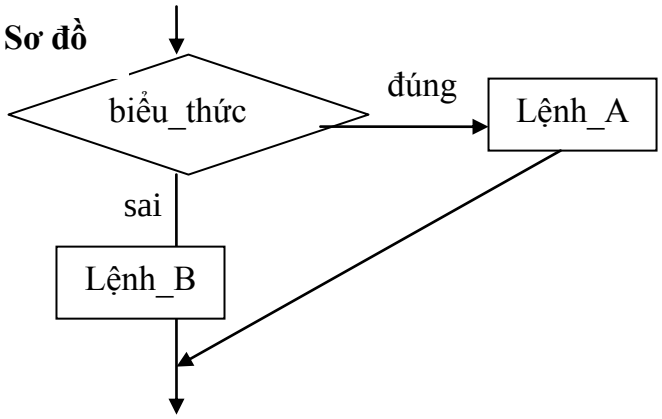
Cú pháp

```
if (biểu_thức)   Lệnh_A;
else             Lệnh_B;
```

Ý nghĩa

Nếu biểu_thức có giá trị đúng thì Lệnh_A được thực hiện, ngược lại (biểu_thức có giá trị sai) thì Lệnh_B được thực hiện.

Sơ đồ

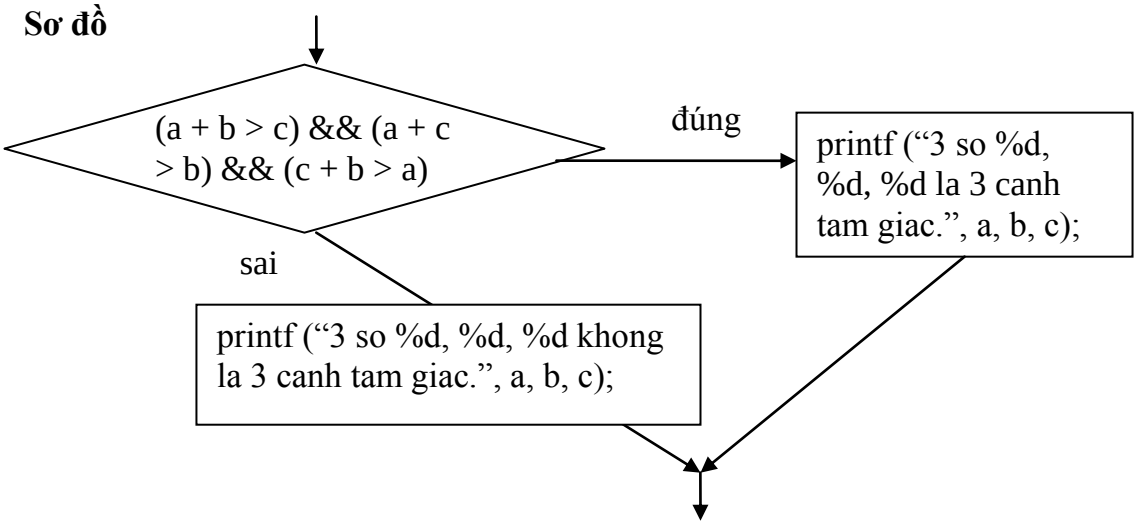


Ví dụ 2.13: Kiểm tra 3 số a, b, c có phải là 3 cạnh của tam giác không ?

Câu lệnh

```
if ((a + b > c) && (a + c > b) && (c + b > a))
        printf ("3 so %d, %d, %d la 3 canh tam giac.", a, b, c);
else
        printf ("3 so %d, %d, %d khong la 3 canh tam giac.", a, b, c);
```

Sơ đồ



Bài tập: Cho a, b, c là 3 cạnh của một tam giác. Hãy cho biết đó là tam giác gì (thường, vuông, cân, vuông cân hay đều)?

✍ ***else** mặc định được gắn với **if** (chưa có else) gần nhất*. Xét 3 đoạn CT được viết bởi 3 LTV C, D, E:

```
// C                    // D                    // E
if (a < 5)              if (a < 5) {            if (a >= 5)
    if (b < 10 )            if (b < 10 )            a = a + 1;
        a = b + 1;              a = b + 1;      else    // a < 5
else                    }                           if (b < 10 )
    a = a + 1;          else  a = a + 1;                a = b + 1;
```

Cả C, D, E đều muốn lệnh "*a = a + 1*" thực hiện khi a >= 5. Đoạn của C thực hiện không đúng ý đồ vì **else** được hiểu với **if** gần nhất, còn đoạn của D (dùng lệnh phức để tách ***else*** ra khỏi ***if*** gần với nó), và E là đúng.

✍ *Trong nhiều trường hợp các lệnh if lồng nhau có thể được đơn giản hóa bằng phép toán && (và).* Chẳng hạn, nên thay *if (a < 5) if (a > 1)* bằng *if ((a < 5) && (a > 1)).*

✍ *Khi phải xét nhiều trường hợp có chung điều kiện, nên dùng các lệnh if lồng nhau để CT hiệu quả và trong sáng hơn (thấy rõ sự phân cấp các trường hợp).*

Ví dụ 2.14: So sánh 3 đoạn CT *F, G, H* xác định xếp loại học sinh (Loai) dựa trên điểm trung bình (DTB) theo quy định: 'G' (giỏi) nếu $8.0 \le DTB \le 10.0$, 'K' (khá) nếu $6.5 \le DTB < 8.0$, 'T' (trung bình) nếu $5.0 \le DTB < 6.5$, 'Y' (yếu) nếu $0.0 \le DTB < 5.0$.

```
// F                                    // G                                    // H
if(DTB < 5.0)                           if(DTB < 5.0)                           if (DTB<5.0)
        Loai = 'Y';                             Loai = 'Y';                             Loai = 'Y';
if((DTB>=5.0)&&(DTB<6.5))               else if((DTB>=5.0)&&(DTB<6.5))          else if(DTB<6.5)
        Loai = 'T';                             Loai = 'T';                             Loai = 'T';
if((DTB>=6.5)&&(DTB<8.0))               else if((DTB>=6.5)&&(DTB<8.0))          else if(DTB<8.0)
        Loai = 'K';                             Loai = 'K';                             Loai = 'K';
if(DTB>=8.0)                            else                                    else
        Loai = 'G';                             Loai = 'G';                             Loai = 'G';
```

- Số phép so sánh của *F* lớn hơn của *G*:
  - Để xác định lần lượt xếp loại 'Y', 'T', 'K', 'G', *G* cần 1, 2, 3, 3 phép so sánh. Số phép so sánh trung bình là 9/4.
  - Để xác định xếp loại, *F* luôn tiến hành 4 phép so sánh. Số phép so sánh trung bình là 4 (lớn hơn 9/4).
- *H* (hiệu quả bằng) nhưng trong sáng hơn *G* vì ta thấy rõ sự phân cấp các trường hợp.

Ví dụ 2.15: CT giải phương trình bậc hai $ax^2 + bx + c = 0$, a, b, c là các số thực.

```
#include …
#include "math.h"
int main(int argc, char* argv[])
{
        float   a, b, c, delta, x1, x2, so_nghiem;
        printf (“Nhap cac he so a, b, c. \n”);
        scanf ( “%f%f%f”, &a, &b, &c);
        delta = (b* b) – 4 * a * c;
        if (delta < 0)
                so_nghiem = 0;
        else {
                if (delta > 0) {
                        x1 = (-b + sqrt (delta)) / (2 * a);    // (1)
                        x2 = (-b – sqrt (delta))  / (2 * a);    // (2)
                }
                else
                        x1 = x2 = (-b) / (2 * a);
                so_nghiem = 2;
        }
        if (so_nghiem == 0)   printf (“Vo nghiem”);
        else                  printf (“Nghiem cua phuong trinh la: %f, %f.”, x1, x2);
        getch (); return 0;
}
```

Hàm sqrt để tính căn bậc 2 chứa trong thư viện “math.h”.

Bài tập: Hãy viết lại hai câu lệnh (1) và (2) hiệu quả hơn (xem 1.4.1).

✋ Ta có thể phát triển CT giải phương trình bậc 2 trên để giải phương trình trùng phương $ax^4 + bx^2 + c = 0$. Để giải phương trình này bằng Toán, ta *đặt ẩn phụ* $T = x^2$ để đưa việc giải phương trình trên về việc giải phương trình bậc 2:

$$aT^2 + bT + c = 0.$$

Tuy nhiên, khi viết CT, nhiều LTV đã rất lúng túng để *đặt ẩn phụ* $T = x^2$ (chẳng hạn, dùng lệnh gán T = x * x). Khó khăn này nảy sinh vì máy tính không hiểu và xử lý được ký hiệu (*trừ phi ta đã lập trình*). Cách giải quyết rất đơn giản:

(1) giải phương trình $aT^2 + bT + c = 0$ để tìm T,

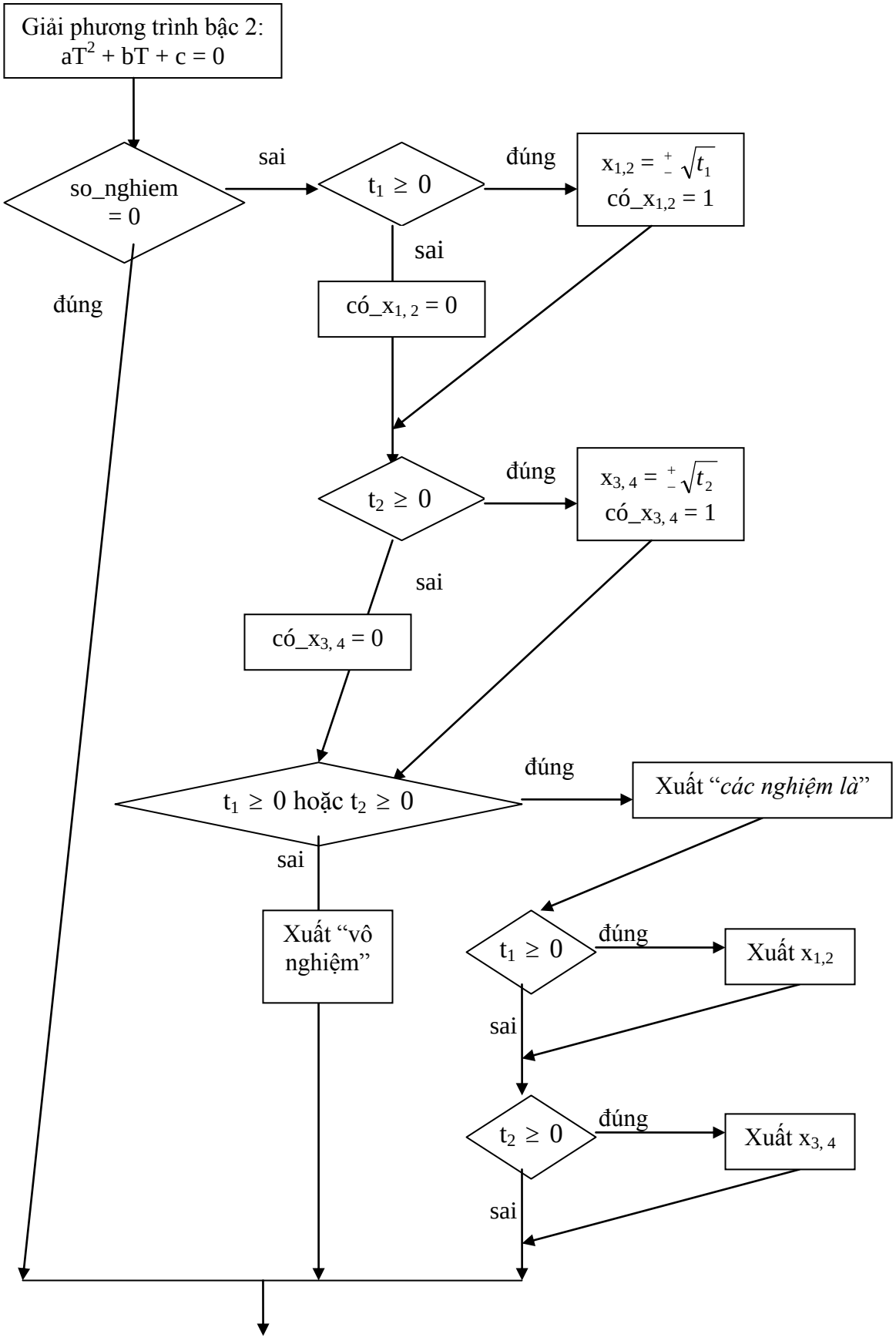
(2) xác định x từ T qua công thức: $x = \pm\sqrt{T}$.

Từ sơ đồ thuật toán giải phương trình trùng phương cho trong hình 2.4 hãy viết CT. *Bài tập.*

2.6.3.3. Lệnh tam phân:

Cú pháp:

(biểu_thức ? biểu_thức_1 : biểu_thức_2)



Hình 2.4 Sơ đồ thuật toán giải phương trình trùng phương

Ý nghĩa: nếu biểu_thức đúng kết quả của lệnh tam phân là giá trị của **biểu_thức_1**, ngược lại kết quả của lệnh tam phân là giá trị của **biểu_thức_2**.

Chẳng hạn:

- *so_lon = (a > b ? a : b);* dùng để tìm số lớn nhất trong hai số a, b
- *Loại = (DTB < 5.0 ? 'Y' : (DTB < 6.5 ? 'T' : (DTB < 8.0 ? 'K' : 'G'))));* dùng để tính xếp loại học sinh dựa vào điểm trung bình (xem ví dụ 2.14).

2.6.3.4 Lệnh rẽ nhiều nhánh:

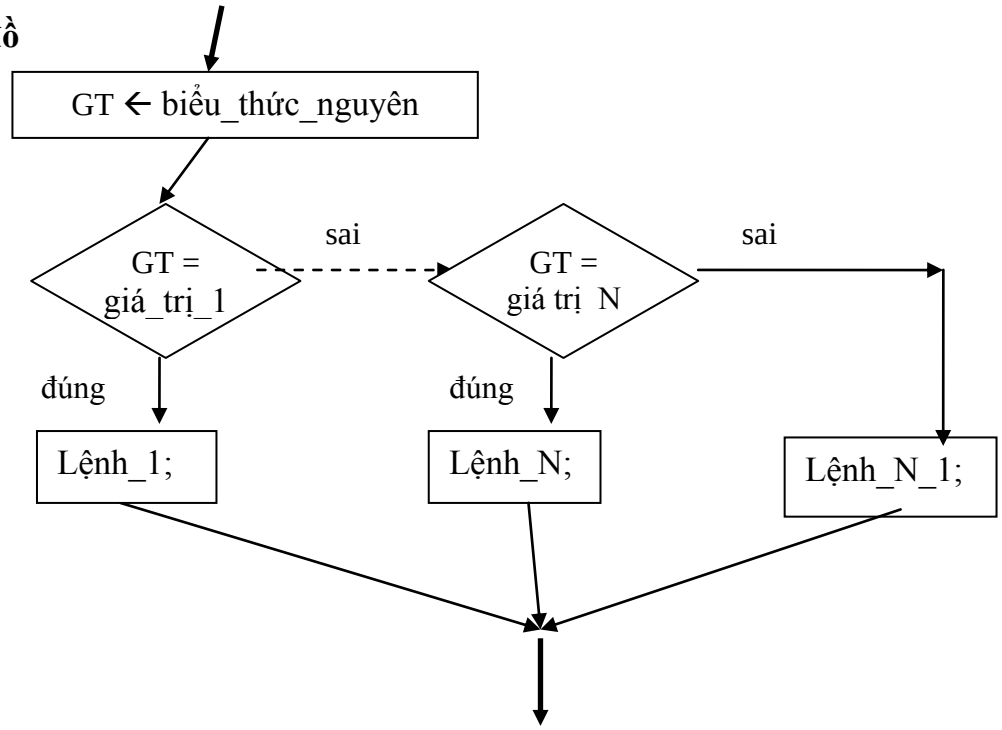
Cú pháp

```
switch (biểu_thức_nguyên)
{
        case    giá_trị_1:
                Lệnh_1;
                break;
        …
        case    giá_trị_N:
                Lệnh_N;
                break;
        default:
                Lệnh_N_1;    // Lệnh N + 1
                break;
}
```

Ý nghĩa

- Thực thi **Lệnh_i** nếu **biểu_thức_nguyên** có kết quả là **giá_trị_i** (i = 1.. N).
- Nếu kết quả không thuộc {**giá_trị_1, .., giá_trị_N**} thì **Lệnh_N_1** được thực hiện.
- Chú ý: các giá trị quyết định (**giá_trị_1, .., giá_trị_N**) cũng phải là giá trị nguyên.

Sơ đồ



Ví dụ 2.16: Tính lương nhân viên dựa vào chức vụ (xem ví dụ 1.10)

```
switch (cv)
{
        case  'G':   luong = 3;  break;
        case  'T':   luong = 2;  break;
        default:     luong = 1;  break;
}
```

✍ **Trong *switch* có thể không có *default***. Ví dụ, đoạn CT đổi một chữ số (chu_so) ra từ tiếng Anh tương ứng:

```
switch (chu_so)
{
        case  '0':   printf ("Zero");   break;
        case  '1':   printf ("One");    break;
        case  '2':   printf ("Two");    break;
        case  '3':   printf ("Three");  break;
        case  '4':   printf ("Four");   break;
        case  '5':   printf ("Five");   break;
        case  '6':   printf ("Six");    break;
        case  '7':   printf ("Seven"); break;
        case  '8':   printf ("Eight");  break;
        case  '9':   printf ("Nine");   break;
}
```

Bài tập: viết lệnh tính toán +, -, *, / giống máy tính bỏ túi.

✍ **Trong *case* có thể không có *break***. Chẳng hạn, nếu không có *break* ở *case '1'*, khi chạy đoạn CT với *chu_so* bằng *1* ta có kết quả là *One Two*. Lý do là, sau khi thực hiện *printf ("One");* máy tính chuyển xuống thực hiện *printf ("Two");* tương ứng với *case 2*. Tận dụng đặc điểm này, ta có một thể hiện khác của lệnh rẽ nhiều nhánh.

Cú pháp

```
switch (biểu_thức_nguyên)
{
   …
   case gt_i1: … case gt_ij: … case gt_imi:
                Lệnh_i;
                break;
   …
   default:
                Lệnh_N_1;    // Lệnh N + 1
                break;
}
```

Ý nghĩa

Thực thi **Lệnh_i** nếu **biểu_thức_nguyên** có kết quả là **gt_ij** (j = 1.. m_i).

Chẳng hạn ta có đoạn lệnh tính số ngày của một tháng trong năm:

```
switch (thang) {
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                        so_ngay = 31; break;
        case 4: case 6: case 9: case 11:
                        so_ngay = 30; break;
        case 2:
             if  (((nam % 4 == 0) &&  (nam % 100 != 0)) || (nam % 400 == 0))
                        so_ngay = 29;           // năm nhuần
             else       so_ngay = 28;
             break;
}
```

Bài tập: Viết CT kiểm tra ba số a, b, c có phải là ngày tháng năm của một ngày nào đó trong năm hay không?

2.6.4 Lệnh lặp:

C^{++} cung cấp ba lệnh lặp: ***for, while*** *và* ***do while*** để biểu diễn các cấu trúc lặp xác-định và không-xác-định.

2.6.4.1. Cú pháp và hoạt động:

Cú pháp:

- for (bthức_khởi_đầu; bthức_điều_kiện; bthức_bước_nhảy)
  Lệnh;
- while (bthức_điều_kiện)
  Lệnh;
- do
  Lệnh;
  while (bthức_điều_kiện);

Hoạt động của for, while, do while cho trong hình 2.5.

2.6.4.2. Biểu diễn các cấu trúc lặp bằng các lệnh lặp for, while, do while:

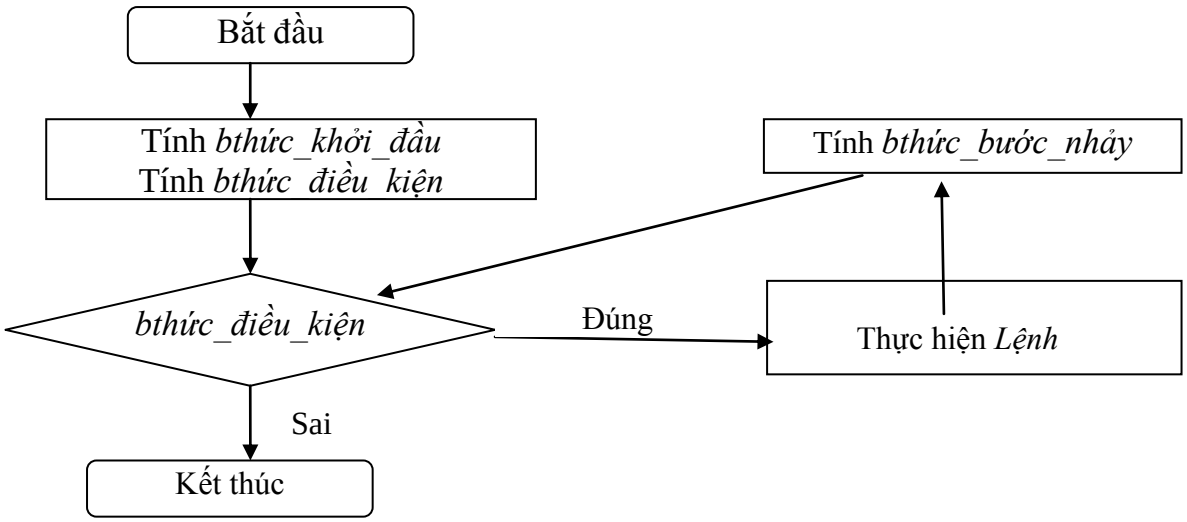
Gọi "Lệnh" là "dãy thao tác cần lặp", ta có thể biểu diễn các cấu trúc lặp (xác-định và không-xác-định trong hình 1.7) bằng các lệnh lặp for, while, do .. while…

2.6.4.2.1. Thể hiện cấu trúc lặp xác định N (N > 1) lần:

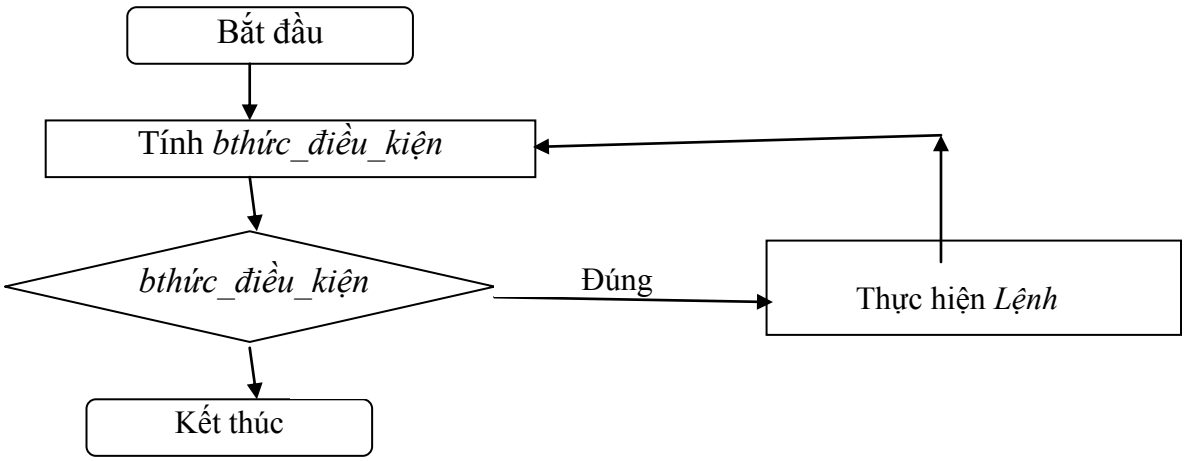
```
for ( i =1;  i <= N;  i=i+1)
                Lệnh;
```

```
int i = 1;
do {    Lệnh;
        i = i + 1;
} while (i <= N);
```

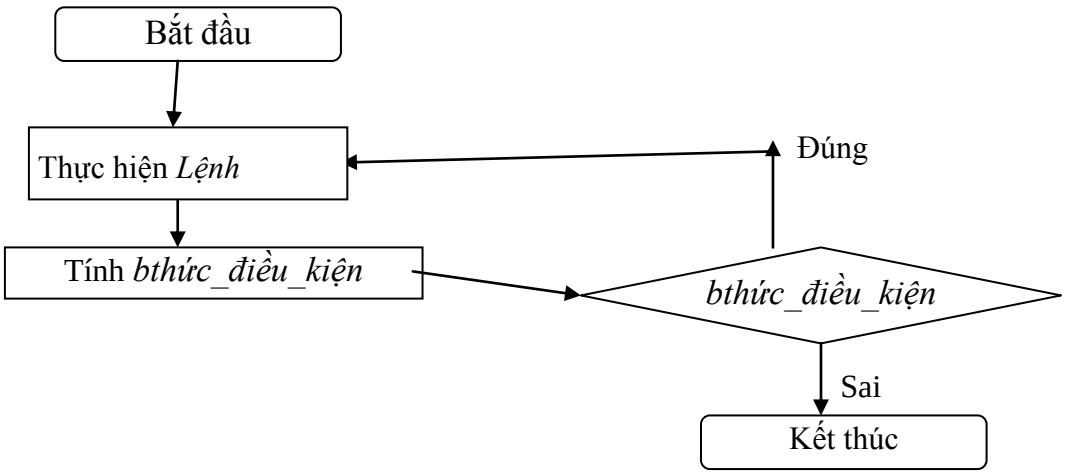
```
int i = 1;
while (i <= N)  {
        Lệnh;
        i = i+1;
}
```



Hình 2.5.a: Hoạt động của for



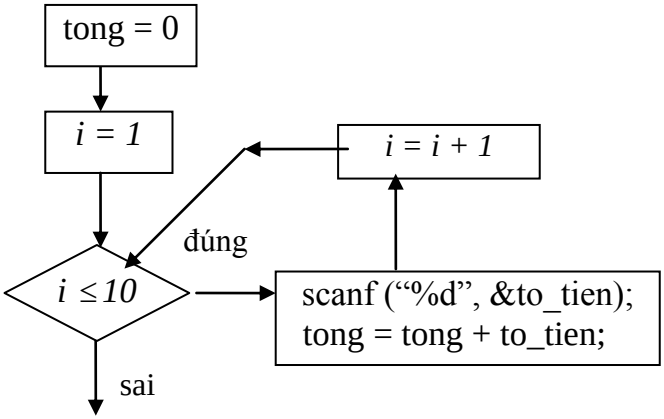
Hình 2.5.b: Hoạt động của while



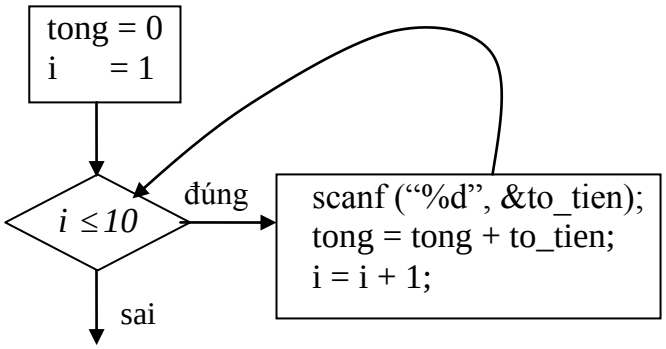
Hình 2.5.c: Hoạt động của do while

Ví dụ 2.17: Tính tổng giá trị 10 tờ tiền

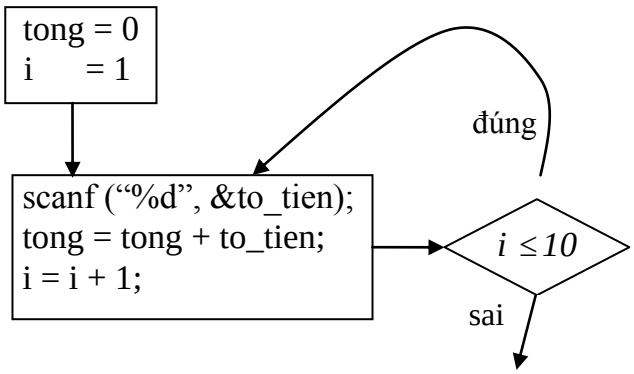
```
ln = 0;
for ( i =1;  i <= 10;  i=i+1)
 {
    scanf ("%d", &to_tien);
    if  (to_tien > ln)
         ln = to_tien;
}
```




```
ln = 0;  i =1;
while (i <= 10)
 {
    scanf ("%d", &to_tien);
    if  (to_tien > ln)
         ln = to_tien;
    i=i+1;
}

Tong = 0;
For(i=1; i<=n; i++)
         Tong = Tong + i;
```




```
tong = 0;
i = 1;
do {
     scanf ("%d", &to_tien);
     tong = tong + to_tien;
     i = i + 1;
} while (i <= 10);
```



Ví dụ 2.18: Tính tổng S = 1 + 3 + 5 + .. + (2M – 1) với M > 1 cho trước

```
S = 0;
for ( i =1;  i <= M;  i++)
      S = S + (2*i – 1);
```

```
i = 1; S = 0;
while (i <= M) {
    S = S + (2*i – 1);
    i++;
}
```

```
i = 1; S = 0;
do {
      S = S + (2*i – 1);
      i++;
}
while (i <= M);
```

Trong ví dụ 2.18 ta thấy lệnh *i++*. Đó là cách viết gọn của lệnh *i = i + 1*. ✍ C++ cho phép ta viết tắt các lệnh gán có dạng

tên_biến = **tên_biến toán_tử** biểu_thức;

thành

tên_biến toán_tử= biểu_thức;

Bảng 2.8: Một số kiểu viết tắt các lệnh gán

Lệnh gán	***Viết tắt***		***Lệnh gán***	***Viết tắt***		***Lệnh gán***	***Viết tắt***
x = x + y;	*x += y;*		*x = x – y;*	*x -= y;*		*x = x * y;*	*x *= y;*
x = x / y;	*x /= y;*		*x = x% y;*	*x %= y;*		*i = i-1;*	*i--;*

Ví dụ 2.19: Xuất ra các số từ 5 đến 20 và 10 câu "C++"

```
for ( i =5;  i <= 20;  i++)
      printf ("%d", i);
for ( i =1;  i <= 10;  i++)
      printf("C++");
```

```
i = 5;
while (i <= 20) {
      printf ("%d", i);
      i++;
}
i = 1;
while (i <= 10) {
      printf("C++");
      i++;
}
```

```
i = 5;
do {
      printf ("%d", i);
      i++;
}while (i <= 20);
i = 1;
do {
      printf("C++");
      i++;
} while (i <= 10);
```

2.6.4.2.2. Thể hiện cấu trúc lặp không-xác-định:

```
for ( ;  ! điều_kiện_dừng ;  )
        Lệnh;
```

```
do {
        Lệnh;
} while (! điều_kiện_dừng);
```

```
while (! điều_kiện_dừng)
        Lệnh;
```

✋ Đến một thời điểm hữu hạn, "*Lệnh*" phải làm cho **điều_kiện_dừng** được thỏa mãn. Nếu không vòng lặp sẽ đến vô hạn.

Ví dụ 2.20: Tìm ước số chung lớn nhất của hai số a, b (xem ví dụ 1.13)

```
for ( ;  a != b ;  )
     if (a > b) a = a – b;
    else        b = b – a;
USCLN = a;
```

```
while (a != b)
        if (a > b)      a = a – b;
        else            b = b – a;
USCLN = a;
```

```
do {
        if (a > b)      a = a – b;
        else            b = b – a;
 } while (! (a == b));
 USCLN = a;
// Đoạn CT này sai ? Vì sao ?
```

✍ ***Khi sử dụng lệnh lặp do..while, ta phải chú ý***: do..while thực hiện lần lặp đầu tiên trước khi kiểm tra điều kiện dừng. Chẳng hạn, đoạn CT viết bằng do..while trong ví dụ trên là sai khi a bằng b. *Vì sao?, hãy sửa lại cho đúng, bài tập.*

Ví dụ 2.21: Thực hiện phép chia nguyên a cho b bằng các phép trừ (xem ví dụ 1.12):

```
thuong = 0;
for ( ;  a >= b;  )  {
        a = a – b;
        thuong++;
}
```

```
thuong = 0;
while (a >= b) {
        a = a – b;
        thuong++;
 }
```

```
if (a == b) thuong = 1;
else {
        thuong = 0;
         do {
                a = a – b;
                thuong++;
        } while (! (a < b));
}
```

2.6.4.3. Câu lệnh break và continue:

Lệnh break và continue thường được lồng trong một lệnh if ở bên trong thân của vòng lặp nhằm làm tăng tính linh hoạt của nó: a) **break:** ngắt vòng lặp giữa chừng.; b) **continue**: bỏ qua các lệnh (trong thân vòng lặp) còn lại bên dưới, tính **bthức_bước_nhảy** chỉ khi lệnh lặp là for, chuyển sang bước lặp kế tiếp nếu **bthức_điều_kiện** đúng.

Khi biểu thức đúng vòng lặp kết thúc

```
Thân vòng lặp
 {
  Lệnh_1;
  ..
 if(biểu_thức) break;
  ..
  Lệnh_N;
 }
```

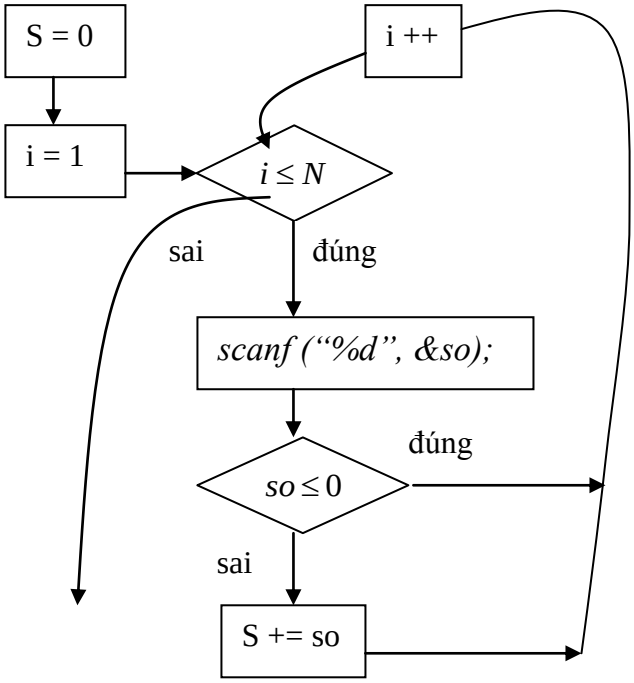
```
Thân vòng lặp
{
  Lệnh_1;  …
  if(biểu_thức) continue;
  Lệnh_i;
  ..
  Lệnh_N;
}
```

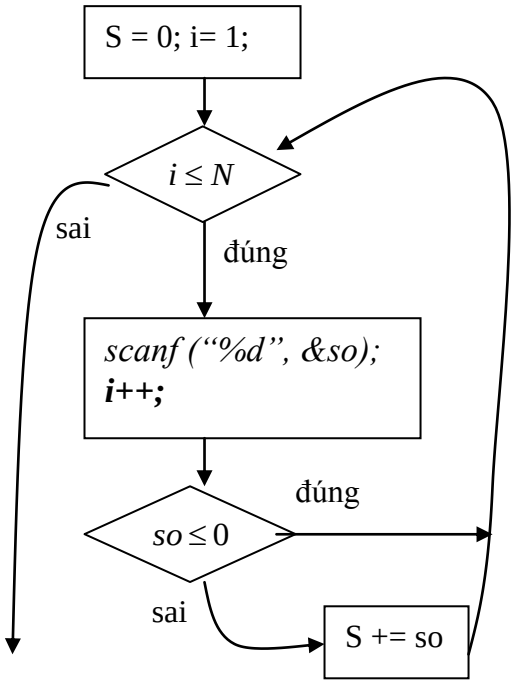
Khi biểu thức đúng, Lệnh_i, .., Lệnh_N bị bỏ qua để sang bước lặp kế tiếp

Ví dụ 2.22 Tính tổng các số dương trong N số nguyên nhập vào

```
S = 0;
for (i=1; i<=N; i++)
{
        scanf ("%d", &so);
        if (so <= 0)  continue;
        S += so;
}
```

```
S = 0; i = 1;
while (i <= N)  {
        scanf ("%d", &so);
        i++;
        if (so <= 0)  continue;
        S += so;
}
```





☞ Trong hai đoạn CT trên

```
if (so <= 0)  continue;
S += so;
```

tương đương với

```
if (so <= 0)  continue;
else          S += so;
```

Vì sao?

Ví dụ 2.23 Kiểm tra tính nguyên tố của số nguyên p > 1 (xem ví dụ 1.14):

```
chia_het = 0; i = 2;
do {
        if (p % i == 0)  {
                chia_het = 1; break;
        }
        i++;    // Thay i++ bằng else i++ được không? tại sao?
} while (i < p);
if (i = = p)  printf ("Nguyen to");
else          printf ("Khong nguyen to");
```

✋ Nếu thay `if (i = = p)  printf ("Nguyen to");`
`else  printf ("Khong nguyen to");`

trong đoạn CT trên thành `if (!chia_het) printf ("Nguyen to");`
`else  printf ("Khong nguyen to");`

thì kết quả còn đúng không? *Tại sao ?*

Bài tập: dùng lệnh for, while viết lại đoạn CT trên và so sánh chúng.

2.6.4.4. Tối ưu vòng lặp:

Lệnh lặp là lệnh chiếm dụng nhiều thời gian nhất trong CT, do đó, khi viết các vòng lặp, ta nên tối ưu chúng. Một số gợi ý: a) tận dụng kết quả tính toán ở các bước trước, b) không tính hằng (so với vòng lặp) trong vòng lặp, trải vòng lặp, giảm các vòng lặp lồng nhau.

Ví dụ 2.24: Tính e^x bằng công thức $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + .. + \frac{x^n}{n!}$. Để tính e^x ta sử dụng cấu trúc lặp không xác định: dừng khi $\frac{x^n}{n!} \le \varepsilon$. Đoạn CT1 (không tối ưu, ở dưới trái) dùng hai cấu trúc lặp xác định để tính x^i, i!. Đoạn *CT2* (*tối ưu hơn đoạn CT1*, ở dưới phải) *tận dụng kết quả tính* $\frac{x^{n-1}}{(n-1)!}$ *ở bước trước* để tính $\frac{x^n}{n!}$ (nhân thêm với $\frac{x}{n}$).

```
#define       EPSILON     0.0001
…
long   GiaiThua;
int    i, n;
float  LuyThua, a_n, e_mu_x, x;
…
e_mu_x = n = a_n = 1;
while (a_n > EPSILON) {
    LuyThua = x;
     for (i=2; i<=n; i++)   LuyThua *= x;
    GiaiThua = 1;
     for (i=2; i<=n; i++)   GiaiThua *= i;
    a_n      =  LuyThua / GiaiThua;
    e_mu_x  +=  a_n;
    n++;
}
```

```
#define  EPSILON     0.000001
…
long  GiaiThua;
int   i, n;
float LuyThua, a_n, e_mu_x, x;
…
e_mu_x = n = a_n = 1;
while (a_n > EPSILON)
{
      a_n      =   a_n * (x / n);
      e_mu_x  +=  a_n;
      n++;
}
```

Ví dụ 2.25: Để tính tổng thể tích tất cả các hình cầu có bán kính R lần lượt là 1mm, 2mm, .., 1000mm; một cách máy móc, ta viết vòng lặp thể hiện công thức $S = \sum_{R=1}^{1000} \frac{4 \prod R^3}{3}$. Vòng lặp này *lặp lại* 999 lần *tính giá trị* $\frac{4\prod}{3}$ không cần thiết vì đây là

hằng đối với vòng lặp. Chuyển $\frac{4\Pi}{3}$ ra khỏi S, ta có $S = \frac{4\Pi}{3}\sum_{R=1}^{1000} R^3$. Đoạn CT sau chỉ tính giá trị $\frac{4\Pi}{3}$ một lần:

```
#include "math.h"
#define   PI   3.14
…
S = 0;
for (R=1; R<=1000; R++)
     S += pow(R, 3);     // pow(R, 3) ≡ R^3, thư viện math.h
S *= (4*PI) / 3;
```

✋ Đối với các vòng lặp xác định có N bước lặp, ta có thể áp dụng kỹ thuật trải vòng lặp để giảm số bước lặp (dĩ nhiên phải thực hiện nhiều tính toán hơn trong mỗi bước lặp). Có thể tăng tốc câu lệnh for (giảm 1/2 số bước lặp, *có thể giảm 1/4 số bước lặp được không? bài tập*?) trong đoạn CT trên theo hai cách:

```
for (R=1; R<=500; R++)
{
     S += pow(2*R-1, 3);
     S += pow(2*R,   3);
}
```

```
for (R=1; R<1000; R+=2)
{
    S += pow(R,   3);
    S += pow(R+1, 3);
}
```

✍ ***Sau các câu lệnh điều khiển ta không đặt dấu ';'*** (trừ khi ta muốn thực hiện một lệnh rỗng (không làm gì)). Đây là lỗi thường gặp. Chẳng hạn, để tính $S = \sum_{=1}^{10} i$ một LTV đã viết đoạn CT:

```
S = 0;
for (i=1; i<= 10; i++);
        S += i;
```

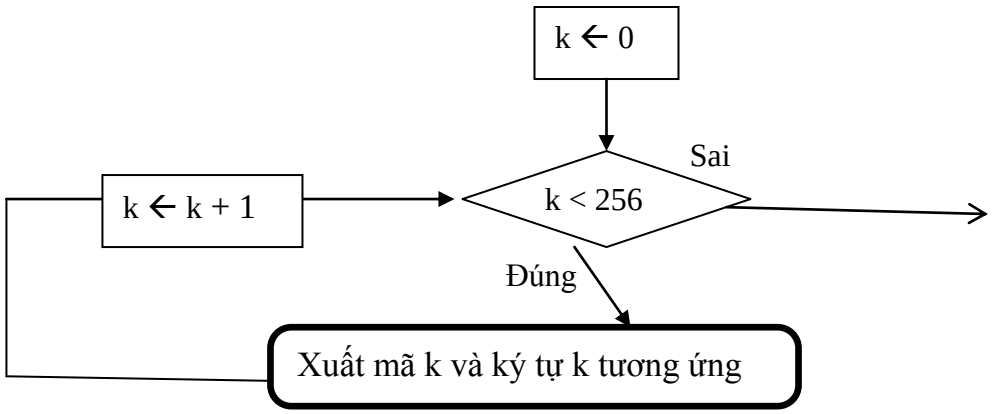
Phân tích: Do có ';' sau câu lệnh for nên máy lặp 10 lần lệnh rỗng. Khi vòng lặp kết thúc, i mang giá trị 11. Sau khi thực hiện lệnh "S += i", *S có giá trị là 11, trong khi đó giá trị đúng của S là 55*.

2.7 Phân tích một số chương trình đơn giản

Trong phần này, ta sẽ phân tích quá trình viết một số chương trình giải các bài toán đơn giản.

2.7.1 Xuất bảng mã ASCII

Bảng mã ASCII chứa mã (số nguyên) của 256 ký tự được đánh số từ 0 đến 255. Ta sẽ duyệt k từ 0 đến 255, ứng mỗi k ta xuất ra k và ký tự có mã ASCII là k tương ứng.




```
/*/     Chương trình hiện bảng mã ASCII
/*/

// Các thư viện
#include "stdafx.h"
#include “conio.h”          // Chứa getch() để dừng CT xem kết quả
#include “stdio.h”          // Chứa getch() để dừng CT xem kết quả
// Hàm chính
int     main(int argc, char* argv[])
{
        // Khai báo các biến
        int     k;

        printf (“----------------------------[ MaASCII, KyTu]----------------------------\n”);
        for (k = 0; k < 256; k++)
                printf (“  %c %d”, k, k);

        getch ();
        return 0;
}
```

2.7.2 Tính số tổ hợp chập K của N phần tử

Công thức tính số tổ hợp chập k của n phần tử $C_n^k = \frac{(k+1)*..*n}{(n-k)!}$.

1) Nhập các giá trị n, k.
2) Tính *n*! và lưu trữ kết quả vào biến có tên *n_gt*.
3) Tính k! và lưu trữ kết quả vào biến có tên *k_gt*.

4) Tính (n-k)! và lưu trữ kết quả vào biến có tên *nk_gt* (ta không thể dùng tên n-k_gt được ? vì sao?).
5) Tính giá trị biểu thức *n_gt / (k_gt * nk_gt)* và lưu kết quả vào biến có tên to_hop.
6) Xuất giá trị của biến to_hop.

Để tính giai thừa của n, ta dùng vòng lặp **for** có n bước lặp. Ở bước lặp thứ i, ta lấy kết quả phép nhân i với *n_gt* (ban đầu *n_gt* được gán bằng 1) và gán lại vào *n_gt*. Tương tự, ta có thể tính giai thừa của k, n-k.

Các biến: n, k, n_gt, k_gt, nk_gt, to_hop có kiểu số nguyên.

```
/*/     Chương trình tính số tổ hợp chập k của n phần tử
/*/

// Các thư viện
#include "stdafx.h"
#include “conio.h”          // Chứa getch() để dừng CT xem kết quả
#include “stdio.h”          // Chứa các thao tác nhập xuất

// Hàm chính
int     main(int argc, char* argv[])
{
        // Khai báo các biến
        int     n, k,
                n_gt, k_gt, nk_gt,          // Lưu kết quả tính n!, k!, (n-k)!
                to_hop,                     // Lưu kết quả tính tổ hợp
                i;                          // Biến điều khiển vòng lặp

        // 1. Nhập n, k
        printf (“N = \t”);  scanf (“%d”, &n);
        printf (“K = \t”);  scanf (“%d”, &k);

        // 2. Tính N!
        n_gt = 1;
        for (i = 1; i <= n; i = i + 1)          n_gt = n_gt * i;

        // 3. Tính K!
        k_gt = 1;
        for (i = 1; i <= k; i = i + 1)          k_gt = k_gt * i;

        // 4. Tính (N – K)!
        nk_gt = 1;
        for (i = 1; i <= n – k; i = i + 1)      nk_gt = nk_gt * i;

        // 5. Tính tổ hợp
        to_hop = n_gt / (k_gt * nk_gt);
```

```
        // 6. Xuất giá trị của biến to_hop
        printf ("\nC(%d, %d) = %d", n, k, to_hop);

        getch (); return 0;
}
```

2.7.3 Tìm ước số chung lớn nhất của hai số

1) Nhập a, b
2) Tìm USCLN của a và b theo thuật toán chia Euclide lưu kết quả vào biến uscln
3) Xuất uscln

Để tìm ước số chung lớn nhất (USCLN) ta dùng thuật toán chia Euclide (*có thể dùng thuật toán trong ví dụ 1.13, bài tập*). Hoạt động của quá trình tìm USCLN của 16 và 24 (đặt c = 16 và d = 24) của thuật toán chia Euclide được minh họa qua bảng:

c	d	Phần nguyên (nguyen)	Dư (du)
16	24	0	16
24	16	1	8
16	8	2	0
8	**0**		

- Thuật toán dừng khi d bằng 0.
- Giá trị hiện tại của c chính là USCLN của 16 và 24.

Dùng vòng lặp không xác định có điều kiện dừng là "b = 0". Trong mỗi bước lặp, ta thực hiện phép chia c cho d lấy du, sau đó, gán giá trị của d cho c, và của du cho d.

Các biến: a, b, uscln, du có kiểu số nguyên.

```
/*/     Chương trình tìm ước số chung lớn nhất của hai số nguyên a, b
/*/

// Các thư viện
#include …

// Hàm chính
int     main(int argc, char* argv[])
{
        int     a, b,
                c, d, du,       // Các biến trung gian tìm USCLN
                uscln;          // Lưu USCLN của a và b

        // 1. Nhập a, b
```

```
        printf ("Nhập vào 2 so nguyen a, b.\n"); scanf ("%d%d", &a, &b);

        // 2. Tìm USCLN của a và b
        c = a; d = b;                       // Gán a và b vào hai biến c, d
        while (d > 0) {
                du      =  c  % d;          // Lấy phần dư của phép chia c cho d
                c       =  d;               // Gán d vào c
                d       = du;               // Gán du vao d
        }
        uscln = c;                          // USCLN của

        // 3. Xuất uscln
        printf ("USCLN cua %d và %d la = %d.", a, b, uscln);

        getch (); return 0;
}
```

2.7.4 Tìm số lớn nhất trong n số thực dương nhập vào (n > 1)

1) Nhập n > 1
2) Giả sử số lớn nhất (so_ln) là 0.0.
3) Trong khi (chưa nhập đủ N số thực)
    a. Nhập số thực dương thứ i (ban đầu đặt i = 1, sau mỗi bước lặp tăng i lên 1) vào biến so_nhap
    b. Nếu so_nhap > so_ln, thay giá trị hiện tại của so_ln bằng giá trị của so_nhap
4) Xuất so_ln

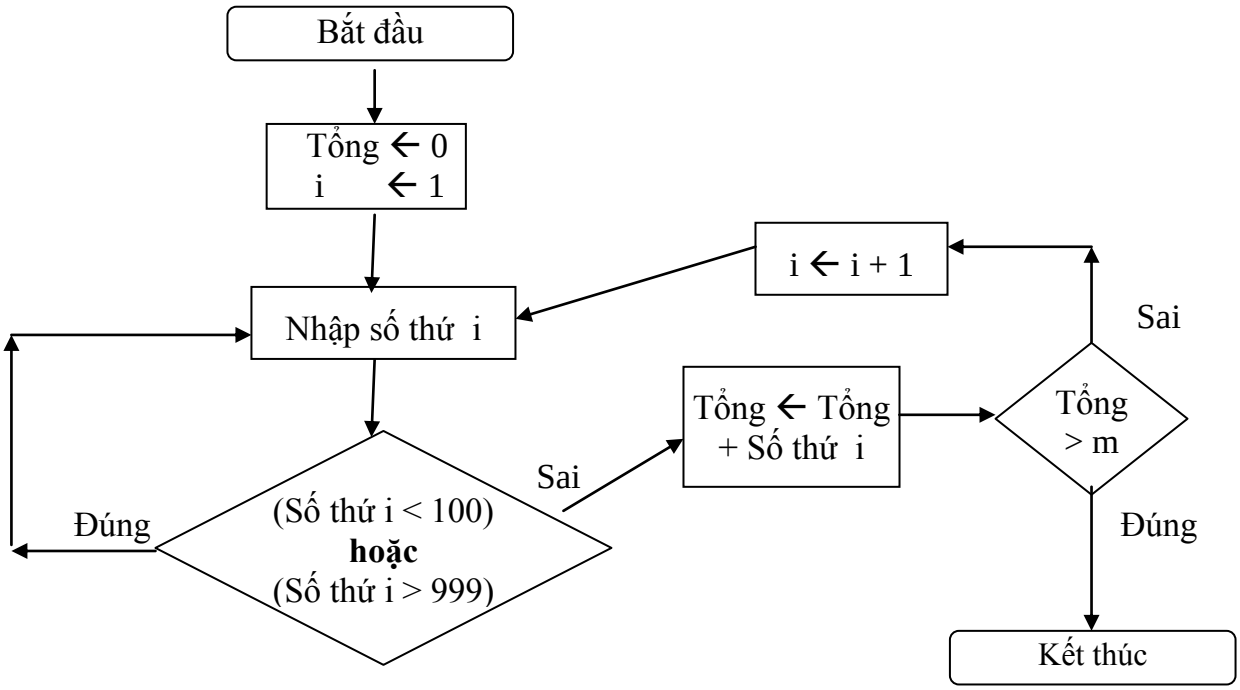
Các biến so_nhap, so_ln có kiểu số thực. Các biến n, i có kiểu số nguyên.

```
/*/     Chương trình tìm số lớn nhất trong n số thực dương (n > 1)
/*/

// Các thư viện
#include …

// Hàm chính
int main(int argc, char* argv[])
{
        int     n, i;
        float   so_nhap,            // Biến chứa số thực dương nhập vào
                so_ln;              // Biến chứa số thực lớn nhất hiện thời

        // 1. Nhập n
        printf( "\n n (>1) = " ); scanf ( "%d", &n);
```

```
        // 2. Giả sử số lớn nhất (so_ln) là 0.0.
        so_ln = 0.0;

        // 3. Tìm số lớn nhất trong n số thực nhập vào
        i = 1;
        do {    printf ("So thu %d = \t", i);  scanf ("%f", &so_nhap);
                so_ln = (so_ln < so_nhap ? so_ln : so_nhap);
                i = i + 1;
        } while (i <= n);

        // 4. Xuất so_ln
        printf( "So lon nhat cua %d so la %f.", n, so_ln);

        getch (); return 0;
}
```

2.7.5 Nhập vào dãy các số nguyên dương 3 chữ số cho đến khi tổng của chúng lớn hơn m cho trước, xuất ra tổng và số lượng số nhập vào

Thuật toán:




```
/*/  CT nhập vào dãy các số nguyên dương 3 chữ số cho đến khi tổng của chúng lớn
     hơn m cho trước, xuất ra tổng và số lượng số nhập vào.
/*/

// Các thư viện
#include …
```

```
// Hàm chính
int     main(int argc, char* argv[])
{
        int     m,
                i     = 1,  // Biến đếm số lượng các số nguyên dương 3 chữ số đã nhập vào
                tong = 0, // Biến chứa tổng các số nguyên dương 3 chữ số đã nhập vào
                so;         // Biến chứa số nguyên nhập vào

        printf( "Tong cho truoc m = ? ");  scanf ("%d", &m);

        while (1)       // hoặc for(;;) thể hiện một vòng lặp không điều kiện
        {
                printf( "Nhap so nguyen 3 chu so thu %d = ", i);
                scanf ("%d", &so);
                if ((so < 100)  || (so > 999))
                        continue;      // Bỏ qua các số không là số nguyên dương 3 chữ số
                tong = tong + so;
                if (tong > m)           // Khi tổng lớn hơn m thì thoát vòng lặp
                        break;
                i++;
        }

        printf( "Tong cua %d so nguyen duong 3 chu so nhap vao la %d", i, tong);

        getch (); return 0;
}
```

2.7.6 Vẽ hình chữ nhật đặc các ký tự '*'

Với chiều dài (c_dai) bằng 6, chiều rộng (c_rong) bằng 4, ta cần vẽ ra màn hình hình chữ nhật sau:

```
******
******
******
******
```

✋ Ta sẽ vẽ lần lượt từng dòng (từ dòng 1 đến dòng c_dai). Ở mỗi dòng, ta xuất ra màn hình c_rong ký tự '*' và chuyển con trỏ xuống dòng dưới.

Dùng hai vòng lặp lồng nhau: a) vòng lặp ngoài điều khiển các dòng, b) vòng lặp trong điều khiển việc vẽ các ký tự trên dòng.

```
/*/  CT vẽ hình chữ nhật đặc các ký tự '*'.
/*/

// Các thư viện
```

```
#include …

// Các hằng
#define       KY_TU       ‘*’

// Hàm chính
int     main(int argc, char* argv[])
{
        int     c_dai, c_rong,          // Biến lưu chiều dài và chiều rộng hình chữ nhật
                dong, cot;              // Biến điều khiển dòng, cột

        printf( “Nhập chiều dài và chiều rộng của hình chữ nhật cần vẽ\n”);
        scanf (“%d%d”, &c_dai, &c_rong);

        for (dong = 1; dong <= c_dai; dong++)
        {
                for (cot = 1; cot <= c_rong; cot++)
                        printf (“%c”, KY_TU);
                printf (“\n”);
        }

        getch (); return 0;
}
```

Viết CT vẽ hình chữ nhật ký tự với ký tự vẽ do người sử dụng nhập vào. *Bài tập.*

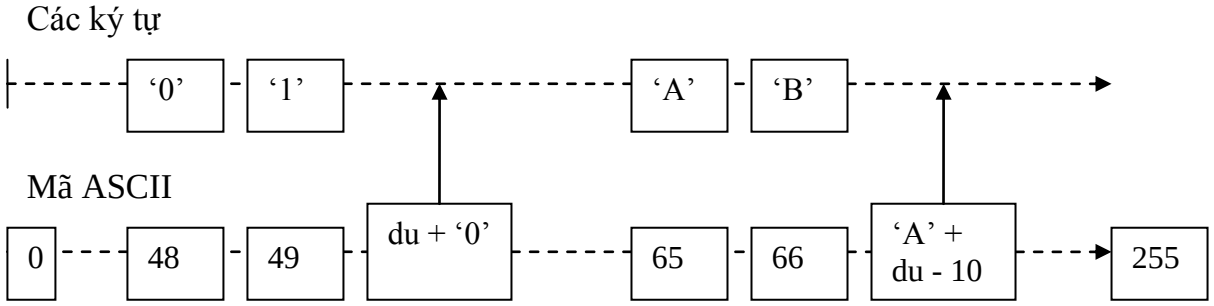
2.7.7 Đổi số từ hệ 10 sang hệ 2, hệ 16

Đổi số sang hệ 2: chia số cho 2, in số dư và lấy thương số cho 2, .., tiếp tục cho đến khi thương số bằng 0. Chẳng hạn, số 30 ở hệ 10 là số 11110 ở hệ 2. Quá trình đổi cho trong bảng:

Số đổi (b)	Hệ đổi	Thương số (*thuong*)	Dư (*du*)
30	2	15	0
15	2	7	1
7	2	3	1
3	2	1	1
1	2	**0**	1

Dùng vòng lặp không xác định có điều kiện dừng là “*thuong* = 0”. Ở mỗi bước lặp ta chia b (ban đầu được đặt bằng số cần đổi) cho 2 lấy *thuong* và *du*, sau đó xuất *du* và gán *thuong* vào b.

Để đổi số sang hệ 16, thay vì chia 2 ta sẽ chia cho 16. Đối với các số dư lớn hơn 10, ta thay chúng bằng các ký tự theo quy ước (dư 10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F'). Cách đổi số dư sang ký tự tương ứng trong hệ 16 cho trong hình 2.7.



Hình 2.7 Đổi số dư sang ký tự

Ví dụ:

Số đổi (b)	Hệ đổi	Thương Số (thuong)	Dư (du)	Quy đổi
30	16	1	14	E
1	16	**0**	1	1

```
/*/     Chương trình đổi số từ hệ 10 sang các hệ 2, 16
/*/

// Các thư viện
#include …

// Hàm chính
int main(int argc, char* argv[])
{
        int     so, du, b;

        printf( "Nhap so nguyen can doi sang he 2 va he 16:\t"); scanf ("%d", &so);

        // Đổi số sang hệ 2
        b = so;
        printf( "Ket qua doi %d sang he %d (theo thu tu dao nguoc): ", so, 2);
        do {
                du  =  b % 2;
                b   =  b /  2;
                printf( "%d", du);
        } while (b != 0);

        // Đổi số sang hệ 16
        b = so;
        printf( "\nKet qua doi %d sang he %d (theo thu tu dao nguoc): ", so, 16);
        do {
```

```
            du  = b % 16;
            b   =  b /  16;
            switch (du) {
                  case 0: case 1: case 2: case 3: case 4: case 5:
                  case 6: case 7: case 8: case 9:
                        printf( "%c", du + '0'); break;
                  default:
                        printf( "%c", 'A' + du – 10); break;
            }
      } while (b != 0);

      getch (); return 0;
}
```

BÀI TẬP

1. Có bao nhiêu loại chú thích? Tại sao phải sử dụng chú thích?
2. Chỉ thị #include dùng để làm gì?
3. Sự khác nhau giữa biến nguyên và biến thực là gì?
4. Tại sao phải sử dụng hằng định danh thay thay cho hằng giá trị?
5. Các ký tự nào được phép xuất hiện trong tên biến? Các quy tắc nào để đặt tên cho biến? Các quy tắc nào để đặt tên cho hằng?
6. Cho biết giá trị nhỏ nhất mà một biến kiểu int có thể lưu giữ.
7. Hãy xác định các kiểu biến tối ưu để lưu trữ các giá trị sau: tuổi một người, bán kính hình tròn, giá của một món hàng, nhiệt độ (^{o}C). Đặt các tên biến thích hợp để lưu trữ các giá trị đó.
8. Chỉ ra các tên biến nào là hợp lệ:
main, 123variable, x, if, tong_thu_nhap, trong_luong_#s, const, one, gross-cost, typedef, int, Radius, ban-kinh, define, tienthu_duoc, void.
9. Câu lệnh sau gọi là câu lệnh gì, và nó có ý nghĩa gì?
   x = 5 + 8;
10. Biểu thức là gì ? Trong một biểu thức chứa nhiều phép toán, cái gì quyết định trật tự tính toán các phép toán?
11. Biểu thức 10 % 3 sẽ nhận giá trị là bao nhiêu?
12. Biểu thức 5 + 3 * 8 / 2 nhận giá trị gì? Sử dụng các dấu ngoặc, hãy viết lại biểu thức trên sau cho nó vẫn nhận cùng một giá trị.
13. Có thể xem một biểu thức có kết qủa 0/1 là biểu thức logic có trị sai/đúng?

14. Có cách nào để viết đúng các biểu thức khi không biết độ ưu tiên của các phép toán hay không?

15. Cấu trúc tam phân là gì? Cấu trúc này có ích lợi gì?

16. Đoạn chương trình sau được viết có tốt không?

```
int main(int argc, char* argv[])   {int x, y; printf ("\nNhap 2 so");
 scanf ("%d", &x);scanf ("%d", &y);
        printf ("\nSo lon la : %d", (x>y ? x :   y));getch();return 0; }
```

Hãy viết lại đoạn chương trình trên cho dễ đọc hơn.

17. Viết một câu lệnh if để: gán giá trị của x cho biến y chỉ khi x nằm trong khoảng từ 1 đến 20, không thay đổi giá trị của y nếu x không thuộc khoảng trên. Sử dụng phép toán ba ngôi để thực hiện cùng yêu cầu trên.

18. Viết lại câu lệnh sau, chỉ sử dụng một lệnh if đơn giản và một phép toán quan hệ:

```
if ( x <  1)
    if ( x > 10 )
        câu_lệnh;
```

19. Các biểu thức sau nhận giá trị gì?
*(1 + 2 * 3), 10 % 3 * 3 – (1 + 2), ((1 + 2) * 3), (5 = = 5), (x = 5).*

20. Nếu x = 4, y = 6, z = 2, thì giá trị của các biểu thức sau là đúng hay sai?

```
if ( x = = 4 )
if ( x != y – z )
if ( z = 1 )
if ( y )
```

21. Các câu viết được bọc trong chú thích, các khoảng trống, dòng trống có ảnh hưởng đến chương trình khi chạy không?

22. Sự khác nhau giữa một lệnh và một khối lệnh?

23. Các biến long có thể giữ các số rất lớn. Tại sao chúng ta không luôn sử dụng chúng thay cho các biến int?

24. Điều gì xảy ra nếu ta gán một giá trị lớn vào một biến có miền giá trị nhỏ hơn?

25. Sự khác nhau giữa phép toán một ngôi và phép toán hai ngôi ? Phép toán trừ (-) là phép toán một ngôi hay hai ngôi?

26. Ta có thể lồng bao nhiêu vòng lặp vào nhau, cần lưu ý gì khi sử dụng các vòng lặp lồng vào nhau?
27. Sự khác nhau giữa các vòng lặp: for, while, do while ? Có thể sử dụng while thay cho for, for thay cho while trong mọi trường hợp được không?
28. x và y sẽ nhận giá trị bao nhiêu sau khi các câu lệnh sau được thực hiện:

```
for (x = 0 ; x < 100 ; x++ )
    ;      // ';' là câu lệnh rỗng !!!
for (y = 2 ; y <= 10 ; y=y+3)
    ;      // ';' là câu lệnh rỗng !!!
```

29. Bao nhiêu chữ X sẽ được in ra màn hình nếu thực hiện các câu lệnh sau:

```
int x, y;

for (x = 0 ; x < 10 ; x++)
    for (y = 5 ; y > 0 ; y--)
        printf ("X");
```

30. Hãy viết câu lệnh for, while đếm một biến từ 1 đến 100 với bước nhảy 3.
31. Tìm lỗi của các đoạn chương trình sau:

a. Đoạn CT 1

```
mau_tin = 0;
while (mau_tin < 100)
{
    printf ("\n Mau tin thu : %d", mau_tin);
    printf ("\n Lay mau tin ke tiep ...")
 }
```

b. Đoạn CT 2

```
for (dem = 1 ; dem < MAX ; dem++);      // MAX là hằng số với gia trị 100
    printf ("\n Dem : %d", dem);
```

32. Đoạn chương trình dưới đây sai khi nào ?

```
i = 0; S = 0;
printf (" n = ');       scanf ("%d", &n);
do {
        printf (" x = ');       scanf ("%d", &x);
        S = S + x;
        i = i + 1;
} while (i < n);
printf (' Tong cac so danh vao la : %d', S);
```

Viết các CT:

33. Tính: chu vi hình vuông, diện tích của hình thoi, hình tròn, tam giác (theo công thức Herông).
34. Kiểm tra tính chất cân, vuông, vuông cân, đều của một tam giác.
35. Tính lương nhân viên dựa vào chức vụ:

    + Chức vụ = 1 → Lương = 300

+ Chức vụ = 2 → Lương = 200
+ Chức vụ = 3 → Lương = 100

36. Máy tính bỏ túi mở rộng trên các số nguyên:
'+', '-', '*': các phép toán cộng, trừ, nhân.
'D': phép chia lấy phần nguyên.
'M': phép mod.
'L': phép lấy số lớn.
'N': phép lấy số nhỏ.

37. Giải và biện luận phương trình bậc 2, hệ phương trình bậc nhất hai ẩn.

38. Tính trung bình cộng của 10 số thực.

39. Vẽ các tam giác sau:

```
AAAA          A          A
AAA          AAA         AA
AA          AAAAA        AAA
A         AAAAAAA        AAAA
```

40. Nhập vào số thực X, dãy số thực dương (kết thúc khi nhập vào số âm), xuất ra trung bình cộng và kiểm tra xem có X trong dãy không?

41. Tính số chỉnh hợp chập K của N phần tử.

42. Kiểm tra tính nguyên tố của một số nguyên > 1.

43. Giải phương trình trùng phương: $ax^4 + bx^2 + c = 0$.

44. Xác định thứ trong tuần của ngày trong năm: "(ngày + 2*tháng + 3*(tháng+1) div 5 + năm + năm div 4 + 1) mod 7 = 0 → thứ bảy, 1 → chủ nhật, 2 → thứ hai, …".

45. Đọc vào 1 số nguyên n>0. Hãy tính:
a) Tổng của n số chẵn đầu tiên.
b) Tổng -1-2+3+4-5 ... n.
c) Tổng $1^2+2^2+ . . . + n^2$.

46. Tính X^n với $n \in Z$, $x \in R$.

47. Nhập vào 1 số nguyên dương. Hãy cho biết:
a) Chữ số đơn vị của nó.
b) Chữ số hàng trăm và hàng chục của nó.
c) Tổng các chữ số của nó với giả sử số nhập vào nhỏ hơn 1000 và lớn hơn hay bằng 100.

48. Nhập vào 4 số nguyên a, b, c, d. Tính và in ra phân số kết quả ở 2 dạng phân số thường và phân số tối giản của: $\frac{a}{b} + \frac{c}{d}$

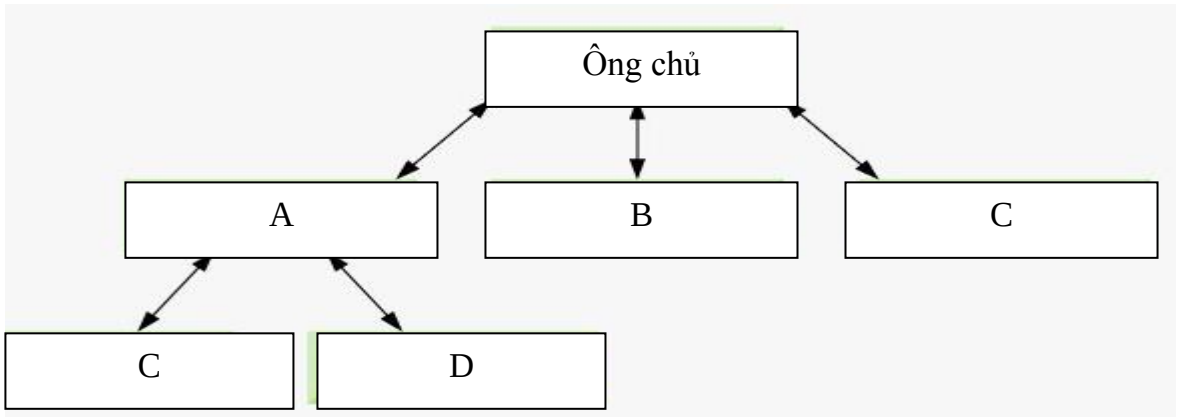
49. Nhập vào một số tiền và cho biết có bao nhiêu cách đổi:
a) Số giấy bạc 100 đồng và số tiền còn dư.
b) Số giấy bạc 100 đồng và 50 đồng và số tiền còn dư.

50. Tính tổng các số lẻ từ m đến n.
51. Nhập vào 1 số nguyên n > 1 và cho biết từ 1 đến n có bao nhiêu số thỏa cả 2 hoặc thỏa một trong hai tính chất: (a) tận cùng bằng chữ số 6, (b) Chia hết cho 4.
52. Hãy cho biết số nguyên n > 0 có bao nhiêu chữ số.
53. Đọc vào số của năm và cho biết can, chi tương ứng của năm. Chẳng hạn, năm 1991 là năm Tân Mùi.
54. Đọc vào 3 số nguyên dương và xác định xem 3 số trên có thể là ngày,tháng và năm của 1 ngày nào đó không ?
55. Đọc vào 3 số biểu thị ngày, tháng và năm. Hãy cho biết đó là ngày thứ mấy trong năm. Chẳng hạn, ngày 1 tháng 3 năm 1972 là ngày thứ 61 trong năm.
56. Đọc vào 3 số biểu thị ngày, tháng và năm của năm 1992. Hãy cho biết đó là ngày thứ mấy (trong tuần lễ) tương ứng.
57. Đọc vào 10 số, xác định: (a) tổng, (b) số lớn nhất, (c) số nhỏ nhất.
58. Tìm tất cả các số nguyên tố từ 2 đến n.

CHƯƠNG 3: LẬP TRÌNH MODULE

3.1. Phương pháp lập trình module

✋ Lấy ví dụ, một ông chủ yêu cầu các công nhân (A, B, C, …), mỗi công nhân thực hiện một công việc và báo cáo kết quả sau khi làm xong. Ông chủ không biết cụ thể từng công nhân làm thế nào để hoàn thành công việc được giao. Mỗi công nhân (chẳng hạn A) có thể phân chia công việc thành các công việc nhỏ hơn để giao cho các công nhân khác (C, D). Các công nhân này không biết ông chủ là ai, họ chỉ biết làm công việc của mình và báo cáo cho người công nhân A đã giao việc cho họ. Trong công ty này, mỗi người thực hiện một nhiệm vụ khác nhau nhưng đều hướng tới mục đích giúp công ty hoàn thành đúng hợp đồng với khách hàng.



Hình 3.1 Quan hệ ông chủ, công nhân

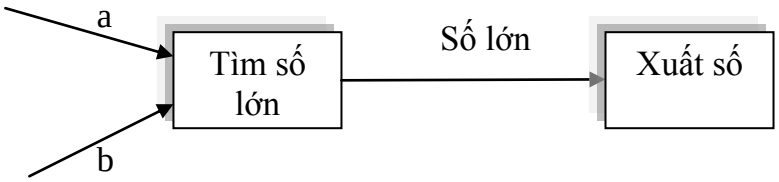
Trong lập trình cũng vậy. Khi giải một bài toán phức tạp, ta thường chia bài toán thành các bài toán con để việc giải bài toán ban đầu trở nên dễ dàng hơn. Phương pháp lập trình module chia nhỏ bài toán cần giải quyết thành các bài toán con, các bài toán con lại được chia thành các bài toán con nhỏ hơn, ... Việc chia nhỏ được lặp lại cho đến khi mọi bài toán con đều đơn giản và có thể giải quyết được, việc giải quyết bài toán lớn ban đầu sẽ quy về việc giải nhiều bài toán con.

✍ Việc chia nhỏ "từ trên xuống" giúp LTV tập trung vào giải quyết từng bài toán cụ thể (nhỏ) mà không phải quán xuyến tất cả các công việc mà bài toán đặt ra. Hơn nữa, có thể giao cho các LTV khác giải giúp các bài toán con. Trong thực tế, các bài toán rất phức tạp đòi hỏi phải tập trung sức lực của rất nhiều LTV. Nhờ phương pháp lập trình module, một nhóm LTV có thể cùng tham gia giải quyết một bài toán: trưởng nhóm sẽ thiết kế sơ đồ giải quyết bài toán và xác định các bài toán con cần giải quyết, mỗi LTV sẽ nhận nhiệm vụ giải quyết một bài toán con (công việc sẽ độc lập và đơn giản hơn).

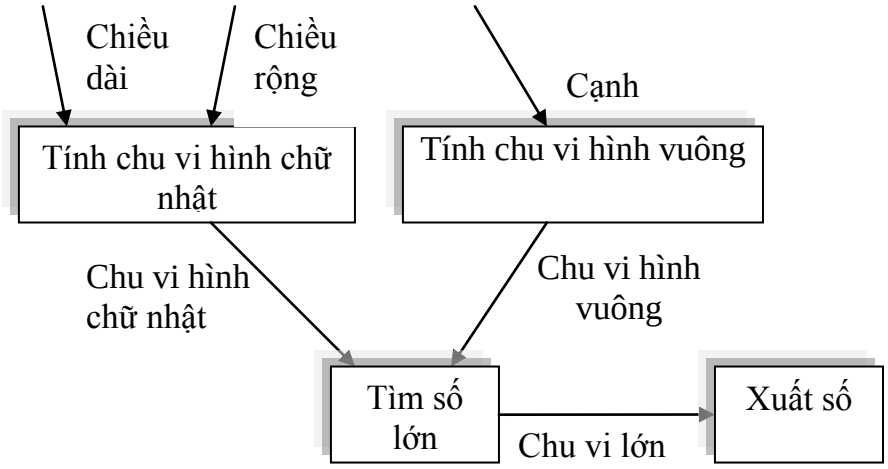
CT sẽ được chia ra thành các module (CT con), các module này tiếp tục được phân thành các module nhỏ hơn, …. Khi thực thi, CT chính (hàm main) sẽ gọi các module chính để yêu cầu chúng thực thi. Đến lượt mình các module này lại gọi đến các module con khác, ….

3.2. Hàm

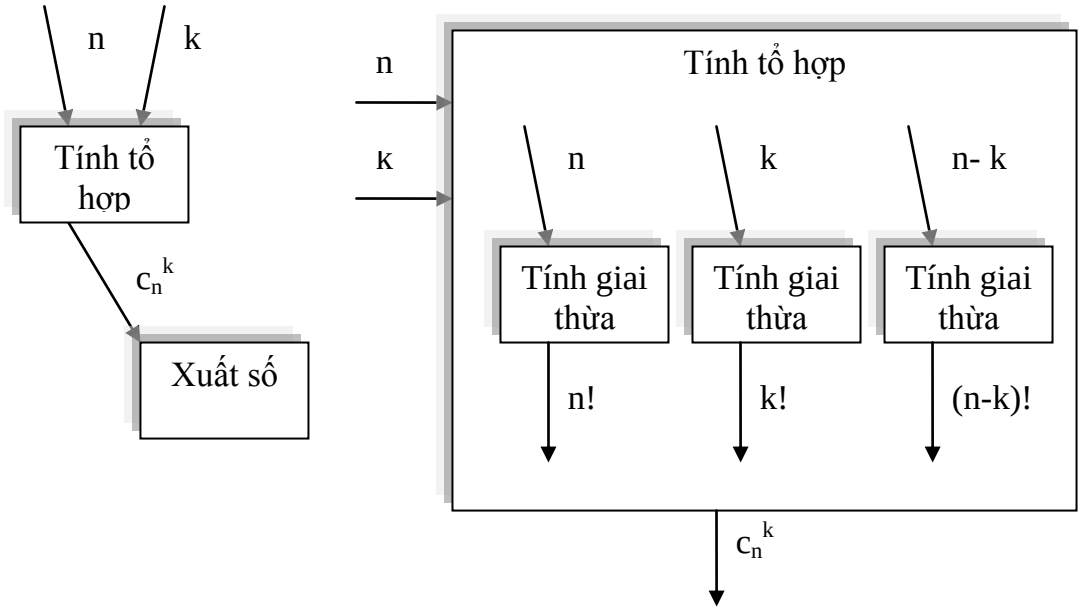
Xét sơ đồ các CT: (CT1) tìm số lớn nhất trong hai số nguyên (hình 3.2), (CT2) xuất ra chu vi lớn nhất trong chu vi hình chữ nhật và chu vi hình vuông (hình 3.3), (CT3) tính số tổ hợp chập k của n phần tử (hình 3.4).



Hình 3.2: Sơ đồ giải bài toán tìm số lớn nhất trong hai số



Hình 3.3: Sơ đồ giải bài toán tìm chu vi lớn nhất của chu vi hình chữ nhật, hình vuông



Hình 3.4: Sơ đồ giải bài toán tìm chu vi lớn nhất của chu vi hình chữ nhật, hình vuông

Ta thấy, bên trong các CT này tồn tại các module cơ bản: tìm số lớn trong hai số, tính giai thừa của một số, tính tổ hợp, tính chu vi hình chữ nhật, tính chu vi hình vuông, xuất một số.

✍ C^{++} đưa vào khái niệm hàm để hỗ trợ lập trình module. Mỗi module tương ứng với một hàm. Hàm là một khối lệnh được đặt tên, thực hiện một nhiệm vụ xác định (một hoặc nhiều lần trong CT). Hàm nhận các đối số vào (hoặc không) và trả về giá trị (hoặc không: **void**) bằng câu lệnh **return** (có cú pháp **return gia_tri_tra_ve;**). Lệnh return sẽ chấm dứt quá trình xử lý của hàm.

3.2.1 Xác định tên hàm:

Tên của hàm cũng tuân theo quy tắc đặt tên cho hằng, biến: đặt tên gợi nhớ, mang tính thống nhất, … Tất nhiên, hàm không được trùng tên với một biến, hằng đã có. Sau đây là một số gợi ý:

1) Tên hàm nên bắt đầu bằng động từ đặc trưng cho mục đích của hàm. Chẳng hạn: tính_giai_thừa, tìm_file, đóng_csdliệu, …
2) Tên hàm trả về kiểu logic nên bắt đầu bằng từ "**Là**". Chẳng hạn, LàNguyênTố, LàĐóngFile, …

3.2.2 Xác định tên và trình tự các đối số:

Tên tham số nên gợi nhớ đến ý nghĩa của đối số. Các đối số nên được để theo trình tự tự nhiên của nó (chẳng hạn, hàm tính tổ hợp nên để đối số n trước k, nên giữ tên n, k giống như trong toán học), các đối số quan trọng để trước, các đối số ít quan trọng để sau, các đối số lấy dữ liệu trả về để sau (chẳng hạn, hàm tính giờ, phút giây nên để các đối số theo thứ tự sau: giay_doi, &gio, &phut, &giay), …

3.2.3 Tiêu đề hàm:

Kiểu trả về của hàm, tên hàm, các đối số, kiểu tương ứng và hình thức truyền kết hợp lại hình thành tiêu đề hàm (prototype).

Tiêu đề các hàm trong CT1, CT2, CT3 cho trong bảng

Hàm	**Tiêu đề**
Tìm số lớn trong hai số	int tim_so_lon (int a, int b);
Xuất số	void xuat_so (int so);
Tính giai thừa của một số	int tinh_giai_thua (int so);
Tính chu vi hình chữ nhật	int tinh_chu_vi_hcnhat (int chieu_dai, int chieu_rong);
Tính chu vi hình vuông	int tinh_chu_vi_hvuong (int canh);
Tính tổ hợp	int tinh_to_hop(int n, int k);

3.2.4 Gọi hàm:

Khi CT gọi một hàm (bằng tên hàm) các đối số thực sẽ được truyền cho các đối số của hàm (nếu hàm có đối số), khối lệnh bên trong hàm được thực hiện và hàm trả về giá trị (nếu hàm có giá trị trả về).

Ví dụ 3.1: CT tìm số lớn nhất trong ba số

```
#include …
// Các hàm
float   tim_so_lon (float   a, float  b)
{
        return (a > b  ?  a : b);
}

void  xuat_so (float so)    // so: tham trị !!!
{
        printf ("%f", so);
}
// Hàm chính
int     main(int argc, char* argv[])
{
        float   so_1, so_2, so_3, so_lon_cua_3_so, so_lon_cua_2_so;

        printf( "Nhap vao ba so can tim so lon nhat:\n" );
        scanf ( "%f  %f  %f ", &so_1, &so_2, &so_3);
        so_lon_cua_2_so = tim_so_lon (so_1,    so_2);
        so_lon_cua_3_so = tim_so_lon (so_lon_cua_2_so, so_3);
        printf("\nSo lon nhat cua (%f,%f,%f):\t", so_1, so_2, so_3);
        xuat_so (so_lon_cua_3_so);

        getch (); return 0;
}
```

a, b đối số của hàm – hình thức

so_1, so_2 đối số thực

so_lon_cua_2_so, so_3 đối số thực

Ví dụ 3.2: CT xuất ra chu vi lớn nhất trong chu vi hình chữ nhật và chu vi hình vuông

```
#include …
// Các hàm
// Hàm tìm số lớn trong hai số.
int    tim_so_lon (int a, int b)
{
        return (a > b  ?  a : b);
}

// Hàm xuất số ra màn hình.
void   xuat_so (int so)
{
        printf ("%d", so);
```

a, b đối số của hàm – hình thức

so đối số của hàm – hình thức

```
}

// Hàm tính chu vi hình chữ nhật.
int    chu_vi_hcnhat (int chieu_dai, int chieu_rong)
{
       return (chieu_dai + chieu_rong) * 2;
}

// Hàm tính chu vi hình vuông.
int    chu_vi_hvuong (int canh)
{
       return canh * 4;
}
```

chieu_dai, chieu_rong đối số của hàm – hình thức

canh đối số của hàm – hình thức

```
// Hàm chính
int     main(int argc, char* argv[])
{
        int cdai, crong, canh, cvi_hcnhat, cvi_hvuong, cvi_lon;

        printf( "Nhap chieu dai va chieu rong cua hinh chu nhat:\n" );
        scanf ( "%d  %d", &cdai, &crong);
        printf( "Nhap canh hinh vuong:\n" );
        scanf ( "%d  %d", &canh);
        printf("\nChu vi lon la:");
        cvi_hcnhat   = chu_vi_hcnhat(cdai, crong);
        cvi_hvuong = chu_vi_hvuong(canh);
        cvi_lon      = tim_so_lon(cvi_hcnhat, cvi_hvuong);
        xuat_so(cvi_lon);

        getch (); return 0;
}
```

Các đối số thực:
- cdai, crong.
- canh.
- cvi_hcnhat, cvi_hvuong.
- cvi_lon.

3.2.5 Biến toàn cục, biến cục bộ và cơ chế che dấu:

Biến toàn cục là biến được khai báo trước hàm main, và không thuộc bất cứ hàm nào. Biến cục bộ là biến được khai báo trong các hàm. Biến toàn cục được hiểu và sử dụng trong tất cả các hàm (kể cả hàm main). Biến cục bộ chỉ được hiểu và sử dụng trong hàm mà nó được khai báo. *Khi có biến cục bộ trùng tên với biến toàn cục thì trong phạm vi hàm: biến cục bộ sẽ được hiểu, biến toàn cục bị che (cơ chế che dấu)*. Biến cục bộ được tạo ra và phân phối bộ nhớ khi hàm được gọi; khi hàm thực hiện xong, các biến cục bộ sẽ bị hủy.

Ví dụ 3.3: minh họa biến toàn cục, cục bộ và cơ chế che dấu

```
#include …
int    x, y;                  // khai báo các biến toàn cục: x, y
void   thu_1 ()               // cơ chế che dấu
```

```
{
        int     x = 88, y = 100;    // khai báo các biến cục bộ: x, y
        printf ("\n Gia tri cua x trong thu_1 = %d", x);          // "Gia tri cua x = 88"
        printf ("\n Gia tri cua y trong thu_1 = %d", y);          // "Gia tri cua y = 100"
}
void    thu_2 ()
{
        x = 88, y = 100;
        printf ("\n Gia tri cua x trong thu_2 = %d", x);          // "Gia tri cua x = 88"
        printf ("\n Gia tri cua y trong thu_2 = %d", y);          // "Gia tri cua y = 100"
}
int     main(int argc, char* argv[])
{
        x = 1; y = 2;
        printf ("\n Gia tri cua x truoc thu_1 = %d", x);          // "Gia tri cua x = 1"
        printf ("\n Gia tri cua y truoc thu_1= %d\n", y);         // "Gia tri cua y = 2"
        thu_1 ();
        printf ("\n Gia tri cua x sau thu_1 = %d", x);            // "Gia tri cua x = 1"
        printf ("\n Gia tri cua y sau thu_1= %d\n", y);           // "Gia tri cua y = 2"
        thu_2 ();
        printf ("\n Gia tri cua x sau thu_2 = %d", x);            // "Gia tri cua x = 88"
        printf ("\n Gia tri cua y sau thu_2= %d\n", y);           // "Gia tri cua y = 100"

        getch(); return 0;
}
```

✍ *Khi viết CT, LTV: nên hạn chế sử dụng biến toàn cục vì rất khó kiểm soát chúng, hơn nữa tính độc lập của các hàm bị vi phạm; b) không nên tiết kiệm các biến cục bộ (vì chúng chỉ tồn tại tạm thời).*

Một lỗi lầm phổ biến khi sử dụng hàm là tận dụng biến toàn cục để tiết kiệm biến cục bộ và đối số của hàm (*thực chất đối số của hàm được truyền bằng tham trị cũng là biến cục bộ*). Chẳng hạn, trong CT tính tổng $1^1 + 2^2 + .. + 100^{100}$, do tiết kiệm biến cục bộ i trong hàm lũy thừa LuyThua mà CT sẽ bị lặp vô tận.

```
#include …

int     i, n;           // Các biến toàn cục

long    LuyThua (int a, int n)
{
        int ket_qua = 1;
        for (i = n; i>= 1; i--)
                ket_qua *= a;
         return ket_qua;
}
```

Khi thoát vòng lặp giá trị của i là 0

```
int     main(int argc, char* argv[])
{
        long S = 0;
        for (i=1; i<=100; i++)
                S += LuyThua(i, i);
        printf ("S = %d", S);
       getch(); return 0;
}
```

*Ở lần lặp đầu tiên, giá trị của i là **1**.*

*Lệnh i++ sẽ làm cho giá trị của i là **1**. Từ đó, dẫn đến vòng lặp vô tận.*

Khi CT lớn (chứa rất nhiều hàm) ta nên liệt kê tất cả các tiêu đề hàm lên đầu CT (trước hàm main) để tiện cho việc quản lý các hàm, cài đặt các hàm để ở dưới hàm main. Hơn nữa, điều đó còn giúp ta tránh lỗi "sử dụng (gọi) một hàm mà hàm đó chưa được định nghĩa".

Ví dụ 3.4: CT tính số tổ hợp chập K của N phần tử

```
#include …
        // Các tiêu để hàm được liệt kê đầu CT
        int     tinh_giai_thua (int so);
        int     tinh_to_hop    (int n, int k);

int     main(int argc, char* argv[])
{
        int n, k, to_hop;

        printf ("N = \t");  scanf ("%d", &n);
        printf ("K = \t");  scanf ("%d", &k);

        to_hop = tinh_to_hop (n, k);

        printf ("\nC(%d, %d) = %d", n, k, to_hop);

        getch (); return 0;
}
        int     tinh_giai_thua (int so)
        {
                // Khai báo các biến cục bộ
                int     giai_thua = 1, i;

                for (i=1; i<=so; i++)
                        giai_thua = giai_thua * i;

                return giai_thua;
        }

        int     tinh_to_hop    (int n, int k)
```

Đối số của hàm (hình thức)

```
{
        // Khai báo các biến cục bộ
        int      n_gt, k_gt, nk_gt;

        n_gt   = tinh_giai_thua (n);
        k_gt   = tinh_giai_thua (k);
        nk_gt = tinh_giai_thua (n-k);

        return   n_gt / (k_gt * nk_gt);
}
```

Các đối số thực

3.2.6 Các phương pháp truyền đối số:

Có hai kiểu truyền đối số thực cho hàm: (a) **tham trị**: *giá trị của đối số thực sẽ được gán cho đối số của hàm*, (b) **tham chiếu**: *tham chiếu của đối số thực được gán cho đối số của hàm* (có thể đồng nhất đối số của hàm với đối số thực, thay đổi trên đối số của hàm đồng nghĩa với thay đổi trên đối số thực). Để quy định cách truyền **tham chiếu** cho đối số của hàm, khi cài đặt hàm ta phải thêm ký hiệu **&** trước tên đối số.

✋ Các đối số trong các hàm của các CT trong các ví dụ 3.1, 3.2, 3.4 được truyền bằng tham trị.

Ví dụ 3.5: CT so sánh hình thức truyền tham chiếu và tham trị

```
#include …
        void  hoan_vi_1 (float &a, float &b);    // a, b: tham chiếu
        void  hoan_vi_2 (float a, float b);      // a, b: tham trị
        void  hoan_vi_3 (float a, float &b);     // a: tham trị, b: tham chiếu
int     main(int argc, char* argv[])
{
        float   so_1_1 = 1.5, so_2_1 = 2.5, so_1_2 = 1.5, so_2_2 = 2.5,
                so_1_3 = 1.5, so_2_3 = 2.5;

        printf ("\n");
        printf ("\nSo 1_1, 2_1 truoc hoan_vi_1: %f, %f", so_1_1, so_2_1);
        hoan_vi_1 (so_1_1, so_2_1);
        printf ("\nSo 1_1, 2_1 sau hoan_vi_1: %f, %f", so_1_1, so_2_1);
        printf ("\n");
        printf ("\nSo 1_2, 2_2 truoc hoan_vi_2: %f, %f", so_1_2, so_2_2);
        hoan_vi_2 (so_1_2, so_2_2);
        printf ("\nSo 1_2, 2_2 sau hoan_vi_2: %f, %f", so_1_2, so_2_2);
        printf ("\n");
        printf ("\nSo 1_3, 2_3 truoc hoan_vi_3: %f, %f", so_1_3, so_2_3);
        hoan_vi_3 (so_1_3, so_2_3);
        printf ("\nSo 1_3, 2_3 sau hoan_vi_3: %f, %f", so_1_3, so_2_3);

        getch (); return 0;
```

```
}
```

```
void  hoan_vi_1 (float &a, float &b)
                    // a, b: tham chiếu
{
      float  tam = a; a = b; b = tam;
}
```

a, b là tham chiếu đến so_1_1, so_2_1. Có thể xem a là so_1_1, b là so_2_1.

```
void  hoan_vi_2 (float a, float b)
                    // a, b: tham trị
{
      float  tam = a; a = b; b = tam;
}
```

a, b là các biến cục bộ nhận giá trị khởi đầu là so_1_2, so_2_2.

```
void  hoan_vi_3 (float a, float &b)
          // a: tham trị, b: tham chiếu
{
      float  tam = a; a = b; b = tam;
}
```

a là tham chiếu đến so_1_3, b là biến cục bộ nhận giá trị khởi đầu là so_2_3.

✋ Để hoán vị hai số nguyên, ta phải dùng hàm hoan_vi_1.

3.2.7. Xác định kiểu trả về cho hàm:

Khi nào hàm có giá trị trả về, khi nào không? Dưới đây là một số gợi ý:

1) Khi muốn thể hiện các đoạn CT có tính chất giống như một hàm toán học và trả về một giá trị, ta dùng hàm có giá trị trả về. Chẳng hạn: tính giai thừa, tổ hợp, chỉnh hợp, tính chu vi hình vuông, tìm số lớn trong hai số, …
2) Dùng hàm không trả về giá trị đối với các đoạn CT có tính chất thực hiện một hành động nào đó (chẳng hạn, hoán vị hai số) hoặc đối với các đoạn CT có tính chất tính toán nhưng trả về cùng lúc nhiều giá trị (chẳng hạn: a) tìm số lớn và nhỏ trong hai số: trả về số lớn và số nhỏ; b) giải phương trình bậc hai: có thể trả về đến hai nghiệm); … ✍ **Để trả về nhiều giá trị, ta dùng con đường truyền tham chiếu (&)**.

Dùng hàm trả về giá trị khi đoạn CT cần phải trả lại một giá trị cho biết tình trạng thực hiện của nó như: chạy thành công, các tình huống lỗi, … Trong các trường hợp đó, người ta thường trả về giá trị kiểu int. Chẳng hạn, đoạn CT đọc một file trên đĩa sẽ được viết bằng một hàm trả về: 1: nếu đọc thành công, 0: nếu không tìm thấy file, -1: nếu đầu đọc bị lỗi, …

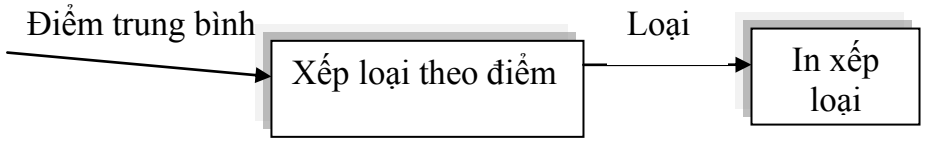
Ví dụ 3.6: CT đổi thời gian dạng giây thành dạng giờ:phút:giây

```
#include …
#define     GIAY_TREN_PHUT      60
#define     GIAY_TREN_GIO       3600
```

```
void   doi_thoi_gian (int giay_doi, int &gio, int &phut, int &giay);
// Hàm này trả về ba giá trị: gio, phut, giay.

int     main(int argc, char* argv[])
{
        int     giay_nhap_vao, gio, phut, giay;
        printf( "Nhap vao so giay:\t" );
        scanf ( "%d", &giay_nhap_vao);
        doi_thoi_gian (giay_nhap_vao, gio, phut, giay);
        printf ( "\nThoi  gian quy đoi la: %d h: %d m : %d s ", gio, phut, giay);
        getch (); return 0;
}

void   doi_thoi_gian (int giay_doi, int &gio, int &phut, int &giay)
{
        int giay_du;
        gio             = giay_doi  /  GIAY_TREN_GIO;
        giay_du         = giay_doi % GIAY_TREN_GIO;
        phut            = giay_du   /   GIAY_TREN_PHUT;
        giay            = giay_du  %  GIAY_TREN_PHUT;
}
```

Ví dụ 3.7: CT xếp loại học sinh dựa trên điểm trung bình

Sơ đồ:




```
#include …
char   xep_loai_theo_diem (float diem);     // Trả về xếp loại tương ứng với diem.
void   in_xep_loai (char loai);             // Không trả về giá trị.

int main(int argc, char* argv[])
{
        char    xep_loai;       float   diem_trung_binh;

        printf( "Nhap vao diem trung binh cua hoc sinh can xep loai:\t" );
        scanf ( "%f ", &diem_trung_binh);
        loai = xep_loai_theo_diem (diem_trung_binh);
        in_xep_loai (loai);

        getch (); return 0;
}
```

```
char    xep_loai_theo_diem (float diem)
{
        char    loai;
        if (diem < 5.0)                 loai = 'Y';
        else   if (diem  < 6.5)         loai = 'T';
               else   if (diem  < 8.0)  loai = 'K';
                      else              loai = 'G';
        return loai;
}
void    in_xep_loai (char  xep_loai)
{
        switch (xep_loai) {
                case    'Y':    printf ("Hoc sinh xep loai Yeu."); break;
                case    'T':    printf ("Hoc sinh xep loai Trung binh."); break;
                case    'K':    printf ("Hoc sinh xep loai Kha."); break;
                case    'G':    printf ("Hoc sinh xep loai Gioi."); break;
        }
}
```

3.2.8 Phân chia chương trình thành những đơn vị logic (module):

Những CT tốt là những CT dễ hiểu, nghĩa là cấu trúc logic của thuật toán phải được thể hiện ngay lên CT. Cách tốt nhất để thể hiện cấu trúc logic của thuật toán là dùng CT con (module).

✍ Một phương pháp tốt để phân chia CT cho hợp lý là đóng khung những đơn vị logic của CT. Trong một CT mang tính module cao thì những đơn vị logic này sẽ có tính độc lập cao.

Một trong những tiêu chuẩn để đánh giá sự độc lập của một module chính là sự liên hệ về mặt dữ liệu của module đó với toàn bộ CT hoặc với những module khác. Một CT con (hàm) là một module độc lập khi nó chỉ nhận dữ liệu vào từ các tham số vào và xuất kết quả ra các tham số ra mà trong quá trình tính toán không tác động đến một module khác hoặc một biến toàn cục nào đó. Điều này có nghĩa là, module đó có thể hoạt động mà không phụ thuộc vào môi trường xung quanh. Nếu module hoạt động phụ thuộc vào môi trường thì nó sẽ không thể tự tách biệt với những sự thay đổi của môi trường, và có thể chịu những hậu quả không mong muốn.

✋ Chính vì vậy, ngay khi thiết kế CT, hãy bình tĩnh chọn lựa và phân chia thật hợp lý các đơn vị logic của CT với tính độc lập cao nhất có thể được. Trong trường hợp có những module phụ thuộc lẫn nhau (*chia xẻ chung các biến toàn cục*) ta tìm cách theo dõi được các ảnh hưởng qua lại giữa các hàm thông qua các biến toàn cục này:

1) *Đối với các hàm có sử dụng, thao tác đến biến toàn cục:*

a) Ghi lại tất cả các tác động của các hàm này lên biến toàn cục như: đọc, tăng giá trị lên 1 khi n nhỏ hơn 5, ...;

b) Miền giá trị hợp lệ của biến toàn cục đối với hàm này. Chẳng hạn, chỉ thi hành khi biến có giá trị là 5, …

2) Đối với các biến toàn cục:

a) Lập sơ đồ theo dõi quá trình biến đổi giá trị của biến toàn cục dưới tác động của hàm.

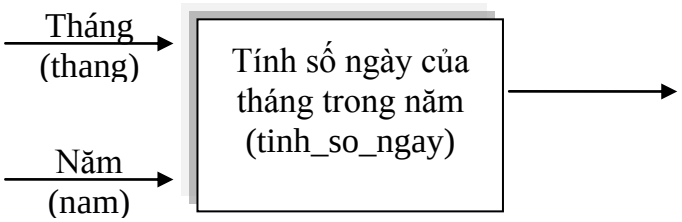
b) Lập bảng theo dõi tất cả tác động của hàm lên biến toàn cục. Chẳng hạn, bảng theo dõi có dạng sau:

Hàm	**Đọc**	**Ghi**	**Tác động**
Ham1		x	Khởi tạo giá trị bằng 1
Ham2	x		Ghi tập tin "file.txt" vào đĩa nếu giá trị bằng 2
…			

3.3. Phân tích một số CT giải một số bài toán bằng phương pháp lập trình module

3.3.1 Xác định số ngày của một tháng trong năm

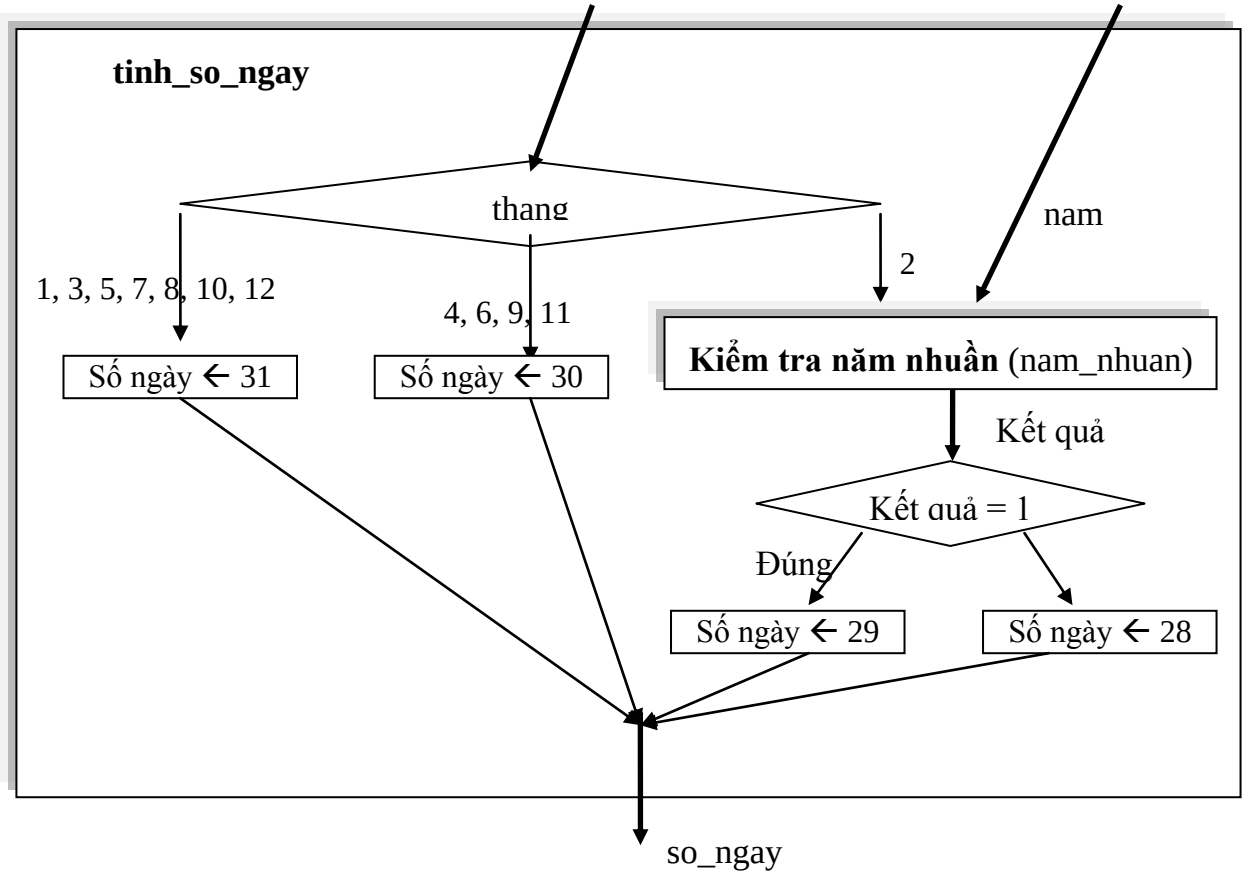
Sơ đồ CT cho trong hình 3.5.a.



Hình 3.5.a Sơ đồ CT giải bài toán 3.3.1

Hàm tính số ngày nhận vào tháng và năm trả về số ngày. Trong hàm tinh_so_ngay, ta cần đến hàm kiểm tra năm nhuần (nam_nhuan) để xác định số ngày của tháng 2 – xem hình 3.5.b.

```
#include …
int     nam_nhuan   (int nam);                  // Kiểm tra năm nhuần
int     tinh_so_ngay (int thang, int nam);      // Tính số ngày trong tháng
int main(int argc, char* argv[])
{
        int  thang, nam, so_ngay;
        // Nhập tháng và năm  ???
        so_ngay = tinh_so_ngay (thang, nam);

        // Thông báo số ngày so_ngay trong tháng thang của năm nam ???
```

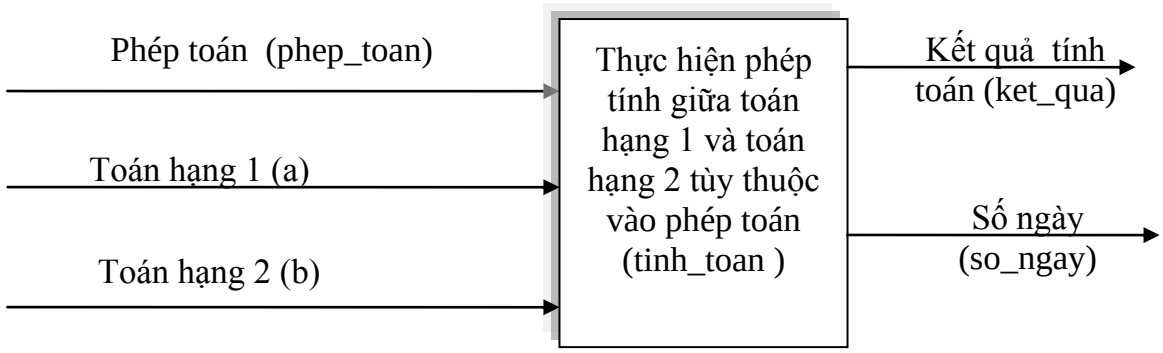


Hình 3.5.b Sơ đồ hàm tính số ngày của tháng trong năm

```
        getch (); return 0;
}
int     nam_nhuan (int nam)                 // Kiểm tra năm nhuần
{
        int  co_nhuan;
        // ???
        return co_nhuan;
}
int     tinh_so_ngay (int thang, int nam)           // Tính số ngày trong tháng
{
        int so_ngay;
        // ???
        return so_ngay;
}
```

3.3.2 Thực hiện các phép tính +, -, *, /

Sơ đồ CT:



✋ Giá trị trả về của hàm là kết quả có thực hiện được phép toán hay không. Do đó, để lấy kết quả thực hiện phép toán (trong trường hợp thực hiện được) ta dùng tham chiếu.

```
#include …
int     tinh_toan (char phep_toan, float a, float b, float &ket_qua);
int     main(int argc, char* argv[])
{
        char  phep_toan;  float  toan_hang_1, toan_hang_2, ket _qua;

        // Nhập toan_hang_1, toan_hang_2 và phep_toan
        // ???

        if (tinh_toan (phep_toan, toan_hang_1, toan_hang_2, ket _qua))
            printf( "%f %c %f = %f", toan_hang_1, phep_toan, toan_hang_2, ket_qua);
        else
            printf( "Khong the %f %c %f.", toan_hang_1, phep_toan, toan_hang_2);

        getch (); return 0;
}

int     tinh_toan(char phep_toan, float a, float b, float &ket_qua)
{
        int tinh_duoc = 1;

        // ???

        return tinh_duoc;
}
```

3.3.3 Giải phương trình bậc 1

Nếu a != 0 thì gọi hàm tính nghiệm (tim_nghiem) để xác định nghiệm của phương trình
Ngược lại:

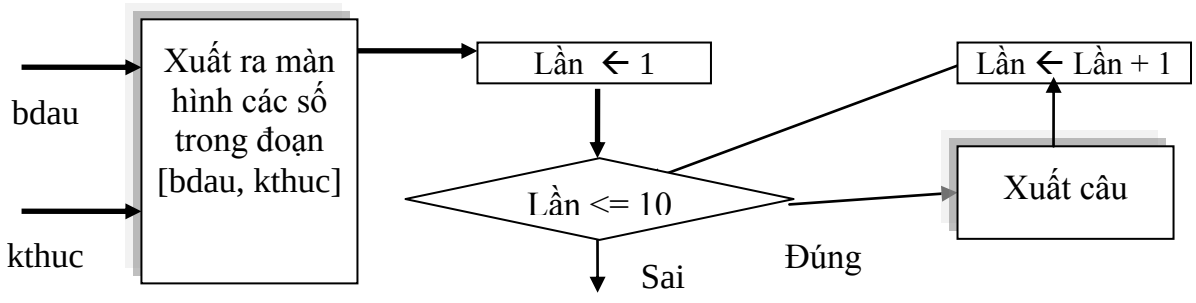
Nếu b != 0 thì thông báo phương trình vô nghiệm.

Ngược lại, thông báo phương trình có vô số nghiệm.

✍ Hàm tính nghiệm nhận hai hệ số a, b của phương trình bậc 1 ax + b = 0, tính toán và trả về nghiệm.

```
#include …
float   tim_nghiem (float a, float b);          // Tính nghiệm phương trình bậc 1
int     main(int argc, char* argv[])
{
        float a, b;

        // Nhap vao cac he so cua phương trình bậc 1
        // ???

        if (a != 0)
                printf( "Ptrinh %f x + %f = 0 co nghiem %f", a, b, tim_nghiem(a, b));
        else    if (b != 0)   printf( "Phuong trinh %f x + %f = 0 vo nghiem", a, b);
                else          printf( "Phuong trinh %f x + %f = 0 co vo so nghiem", a, b);

        getch (); return 0;
}
float   tim_nghiem (float a, float b)           // Tính nghiệm phương trình bậc 1
{
        float   nghiem;

        // ???

        return  nghiem;
}
```

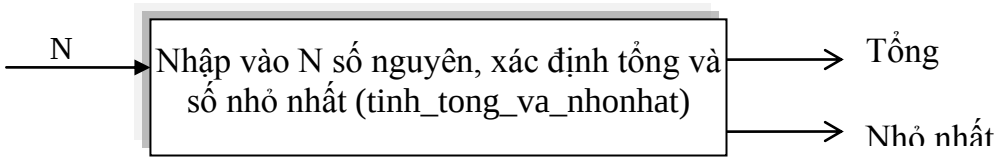
3.3.4 Xuất ra màn hình các số trong đoạn và 10 câu "AAAAAAAAA"




```
#include …
#define     SO_LAN          10
#define     CAU_XUAT        "AAAAAAAAA"
void    xuat_cau ();                // Xuất câu CAU_XUAT
void    xuat_so_trong_doan (int bdau, int kthuc);
```

```
int     main(int argc, char* argv[])
{
        int     dau_doan, cuoi_doan;

        // Nhap doan chua cac so can xuat
        // ???

        xuat_so_trong_doan (dau_doan, cuoi_doan);

        int lan = 1;
        while (lan <= SO_LAN) { xuat_cau (); lan ++; }

        getch (); return 0;
}
void    xuat_so_trong_doan (int bdau, int kthuc)    // xuất các số trong [bdau, kthuc]
{
        // ???
}
void    xuat_cau ()                                 // xuất câu CAU_XUAT
{
        // ???
}
```

3.3.5 Nhập vào N số nguyên, xác định tổng và số nhỏ nhất

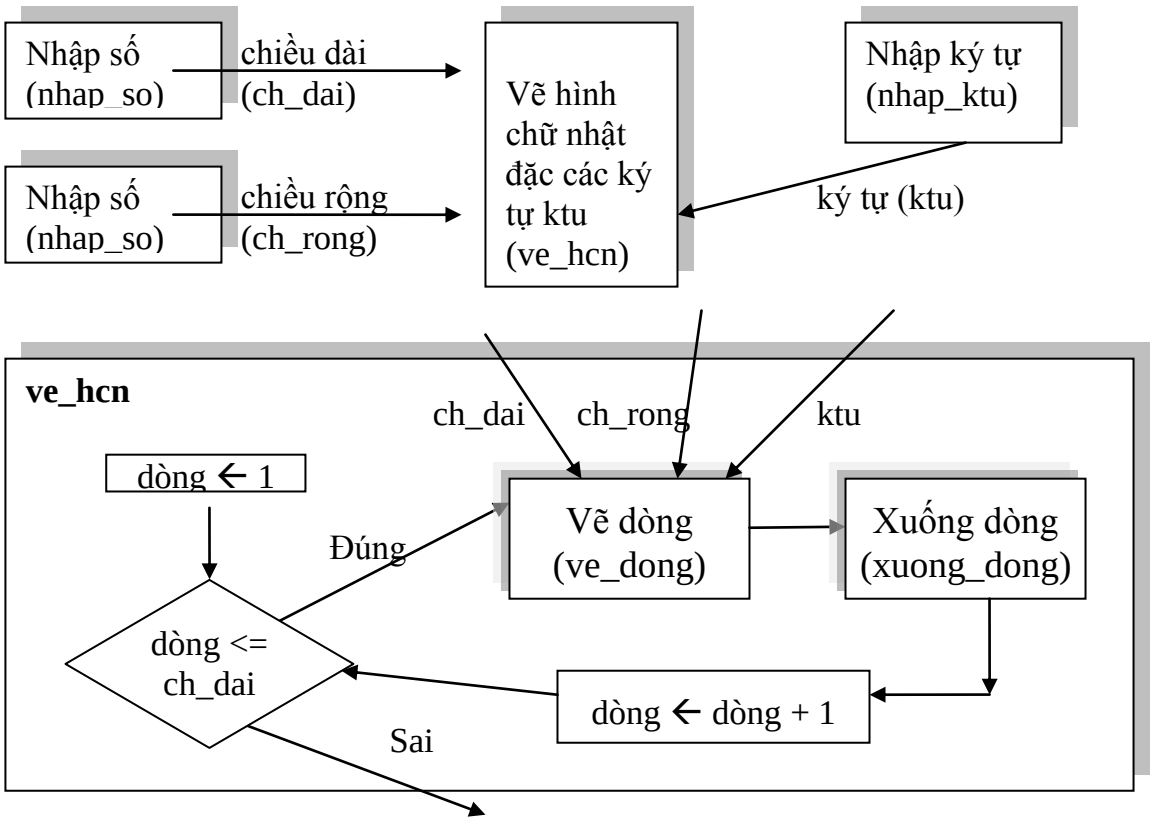


✋ Hàm tinh_tong_va_nhonhat nhận vào số nguyên N, nhập N số nguyên và xác định tổng và số nhỏ nhất. Kết quả tổng và số nhỏ nhất sẽ được lấy ra bằng con đường tham chiếu.

```
#include …
#define     MAX_GIA_TRI     32767
void    tinh_tong_va_nhonhat (int so_lan, int &tong, int &nho_nhat);
int     main(int argc, char* argv[])
{
        int N, tong_tri_gia, to_nho_nhat;

        // Nhập N ???
        tinh_tong_va_nhonhat (N, tong_tri_gia, to_nho_nhat);
        // Xuất tong_tri_gia, to_nho_nhat ???
        getch (); return 0;
}
void    tinh_tong_va_nhonhat (int so_lan, int &tong, int &nho_nhat)
```

```
{
        // ???
}
```

3.3.6 Vẽ hình chữ nhật đặc các ký tự



Hình 3.6 Sơ đồ CT vẽ hình chữ nhật đặc các ký tự

- ✓ Hàm nhap_so trả ra số nhập được bằng con đường tham chiếu, không trả về giá trị.
- ✓ Hàm nhap_ky_tu trả về ký tự nhập được (bằng con đường return).
- ✓ Hàm ve_dong nhận vào ktu và số cột, không trả về giá trị.
- ✓ Hàm ve_hcn nhận vào ch_dai, ch_rong, ktu; không trả về giá trị.

```
#include …
void    nhap_so       (int &so);
char    nhap_ky_tu  ();
void    ve_dong       (int ch_dai, char ky_tu);
void    xuong_dong ();
void    ve_hcn        (int ch_dai, int ch_rong, char ktu);
int main(int argc, char* argv[])
{
        int chieu_dai, chieu_rong, kytu;
        // Nhập chieu_dai, chieu_rong, kytu ???
```

```
        ve_hcn (chieu_dai, chieu_rong, kytu);
        getch (); return 0;
}
void    nhap_so       (int &so)
{
        // ???
}
char    nhap_ky_tu ()
{
        // ???
}
void    ve_dong       (int ch_dai, char ky_tu)
{
        // ???
}
void    xuong_dong ()
{
        // ???
}
void    ve_hcn         (int ch_dai, int ch_rong, char ktu)
{
        for (int dong = 1; dong <= ch_dai; dong++)
        {
                // ???
        }
}
```

3.3.7 Nhập vào một dãy các số nguyên dương (kết thúc khi nhập vào số âm) và tính tích của chúng

✋ Hàm nhập số lấy ra số dương nhập được (bằng con đường tham chiếu) và trả về giá trị 1, trong trường hợp nhập số âm hàm trả về giá trị 0.

```
#include …
int     nhap_so (int thu, int &so);           // Nhập số
int main(int argc, char* argv[])
{
        int so, i, tich = 1;
        for (i = 1; nhap_so (i, so); i++)
                tich = tich * so;
        printf( “Tich cua %d so nguyen duong nhap vao la %d”, i–1, tich);
        getch (); return 0;
}
int     nhap_so (int thu, int &so)            // Nhập số
{
```

```
        int     nhap_duoc_so_duong;
        // ???
        return nhap_duoc_so_duong;
}
```

3.3.8 Nhập vào dãy các số nguyên dương 3 chữ số cho đến khi tổng của chúng lớn hơn M cho trước, xuất ra tổng và số lượng số nhập vào

```
#include …
#define   EVER              ;;
#define   NHAP_LAI          1  // Số vừa nhập không là số nguyên dương 3 chữ số
#define   TIEP_TUC          2  // Số vừa nhập là số nguyên dương 3 chữ số, tong <= M
#define   NGUNG             3  // Số vừa nhập là số nguyên dương 3 chữ số, tong > M
int     nhap_tinh_va_kiem_tra (int so, int &tong, int m);
int     main(int argc, char* argv[])
{
        int so, i=1, tong = 0, M, ket_qua;
        // Nhập M ???

        for (EVER) {
                // Nhập số thứ i ???
                ket_qua = nhap_tinh_va_kiem_tra (so, tong, M);
                // Xử lý ket_qua: continue, hoặc i++, hoặc break
        }

        // Xuất tong ???
        getch (); return 0;
}
int     nhap_tinh_va_kiem_tra (int so, int &tong, int m)
{
        int     ket_qua;

        // ket_qua = NHAP_LAI so không là số nguyên dương 3 chữ số
        // ket_qua = TIEP_TUC khi so là số nguyên dương 3 chữ số, tong <= M
        // ket_qua = NGUNG khi so là số nguyên dương 3 chữ số, tong > M
        // ???

        return ket_qua;
}
```

3.3.9 Tìm bội số chung nhỏ nhất của hai số nguyên dương

```
#include …
int     tinh_uscln (int x, int y);      // tính ước số chung lớn nhất của x và y
int     tinh_bscnn (int x, int y);      // tính bội số chung nhỏ nhất của x và y
```

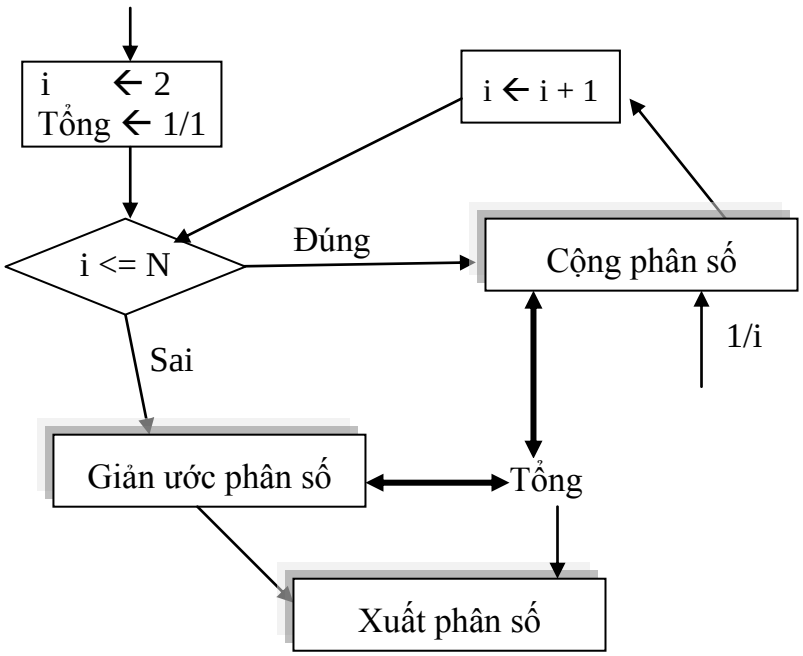
```
int main(int argc, char* argv[])
{
        int so_1, so_2;
        // Nhap hai so nguyen can tim boi so chung nho nhat ???
        bscnn = tinh_bscnn (so_1, so_2);
        // Xuất bscnn ???
        getch (); return 0;
}
int     tinh_uscln (int x, int y)       // tính ước số chung lớn nhất của x và y
{
        int     uscln;
        // ???
        return uscln;
}
int     tinh_bscnn (int x, int y)       // tính bội số chung nhỏ nhất của x và y
{
        int     bscnn;

        return bscnn;
}
```

3.3.10 Đổi số từ hệ 10 sang hệ b

```
#include …
void    xuat_ky_so (int ky_so);      // xuất ký số trong hệ he
void    doi_so (int so, int he);             // đổi số so sang hệ he
int main(int argc, char* argv[])
{
        int so, b;
        // Nhập số ở hệ 10 so và hệ cần đổi b ???
        doi_so (so, b);
        getch (); return 0;
}
void    xuat_ky_so (int ky_so)       // xuất ký số trong hệ he
{
        // ???
}
void    doi_so (int so, int he)              // đổi số so sang hệ he
{
        // ???
}
```

3.3.11 Tính tổng

$$1+\frac{1}{2}+\ldots+\frac{1}{n}$$



Hình 3.7: Sơ đồ CT tính tổng $1+\frac{1}{2}+\ldots+\frac{1}{n}$

```
#include …
void    cong_hai_PS (int& tu, int& mau, int tu_moi, int mau_moi);
int     tinh_uscln (int x, int y);              // tính ước số chung lớn nhất của x và y
void    gian_uoc (int& tu, int& mau);       // giản ước phân số tu/mau
void    xuat_phan_so (int tu, int mau);     // xuất phân số tu/mau
int     main(int argc, char* argv[])
{
        int N, tu_cua_tong, mau_cua_tong, i;
        // Nhập N ???
        tu_cua_tong = mau_cua_tong = 1;
        for  (i = 2; i<=N; i++)
                cong_hai_PS (tu_cua_tong, mau_cua_tong, 1, i);
        gian_uoc (tu_cua_tong, mau_cua_tong);
        xuat_phan_so (tu_cua_tong, mau_cua_tong);
        getch (); return 0;
}
void    cong_hai_PS (int& tu, int& mau, int tu_moi, int mau_moi)
// Cộng hai phân số: tu/mau + tu_moi/mau_moi gán vào tu/mau
{
        // ???
}
int     tinh_uscln (int x, int y)       // tính ước số chung lớn nhất của x và y
{
        int uscln, du;
        // ???
        return uscln;
}
```

```
void   gian_uoc (int& tu, int& mau)      // giản ước phân số tu/mau
{
       // ???
}
void   xuat_phan_so (int tu, int mau)    // xuất phân số tu/mau
{
       // ???
}
```

3.4 Hàm đệ quy:

Thuật toán đệ quy là sự mở rộng (tính xác định) khái niệm thuật toán. Để giải một bài toán bằng đệ quy ta đưa bài toán về bài toán đồng dạng ở cấp độ thấp hơn (chẳng hạn: độ lớn dữ liệu nhỏ hơn, giá trị cần tính toán nhỏ hơn, …). Quá trình này tiếp tục cho đến khi đạt đến các bài toán ở cấp độ nhỏ nhất có thể giải được. Giải các bài toán ở cấp độ nhỏ và lần ngược quá trình cho đến khi giải được bài toán ban đầu.

Ví dụ 3.8: Một số định nghĩa đệ quy:

- Một số nguyên dương lẻ hoặc là số 1 hoặc bằng số nguyên dương lẻ khác cộng 2.
- Người giàu là người kiếm được nhiều tiền hay bố mẹ họ là "người giàu".
- Tổ hợp chập k của N (N $\geq k \geq 0$): $C_n^k = \begin{cases} 1, k = 0 \\ 0, k > n \\ C_{n-1}^{k-1} + C_{n-1}^{k}, k > 0 \end{cases}$
- Giai thừa của một số nguyên dương: $\begin{cases} 0! = 1, \\ n! = n*(n-1)! \end{cases}$;
- Số fibonacci: $\begin{cases} fibo(0) = 0,\ fibo(1) = 1, \\ fibo(n) = fibo(n-1) + fibo(n-2) \end{cases}$

✍ C^{++} cho phép hàm đệ quy. Có hai dạng hàm đệ quy: a) trực tiếp: hàm gọi trực tiếp đến chính nó, b) gián tiếp: hàm gọi đến nó thông qua một hàm khác.

Để giải quyết bài toán bằng đệ qui ta tiến hành theo 3 bước:

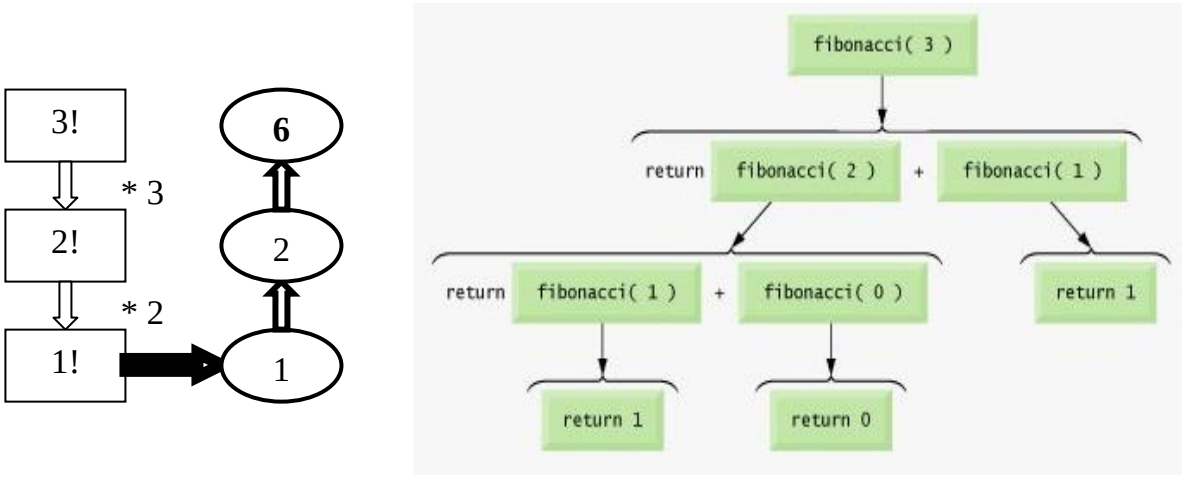
a) Thông số hóa bài toán: Tìm các thông số cần thiết cho bài toán, đặc biệt thông số chỉ kích thước của bài toán, một đại lượng mà nội dung của nó giảm đi sau mỗi lần gọi đệ qui.
b) Tìm các trường hợp tầm thường và giải các trường hợp này: Đây là phần cơ sở của giải thuật, được thực hiện không đệ qui, thường ứng với trường hợp kích thước bài toán nhỏ nhất (bằng 0, 1,...)
c) Phân rã bài toán tổng quát: Tìm phương án đúng để đưa bài toán về bài toán có kích thước nhỏ hơn.

Ví dụ 3.9: Các hàm đệ quy tính n!, số fibonacci và C_n^k:

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

```
int Fib(int n) {
        int kq;
        if  (n == 0) kq = 0;
        else if  (n == 1) kq = 1;
        else  kq = Fib(n -1) + Fib(n -2);
        return kq;
}
```

```
int ToHop(int n, int k)
{
    int kq;
    if  (k == 0) kq = 1;
    else if (k > n)     kq = 0;
    else kq = ToHop(n-1, k-1) + ToHop(n-1, k);
    return kq;
}
```



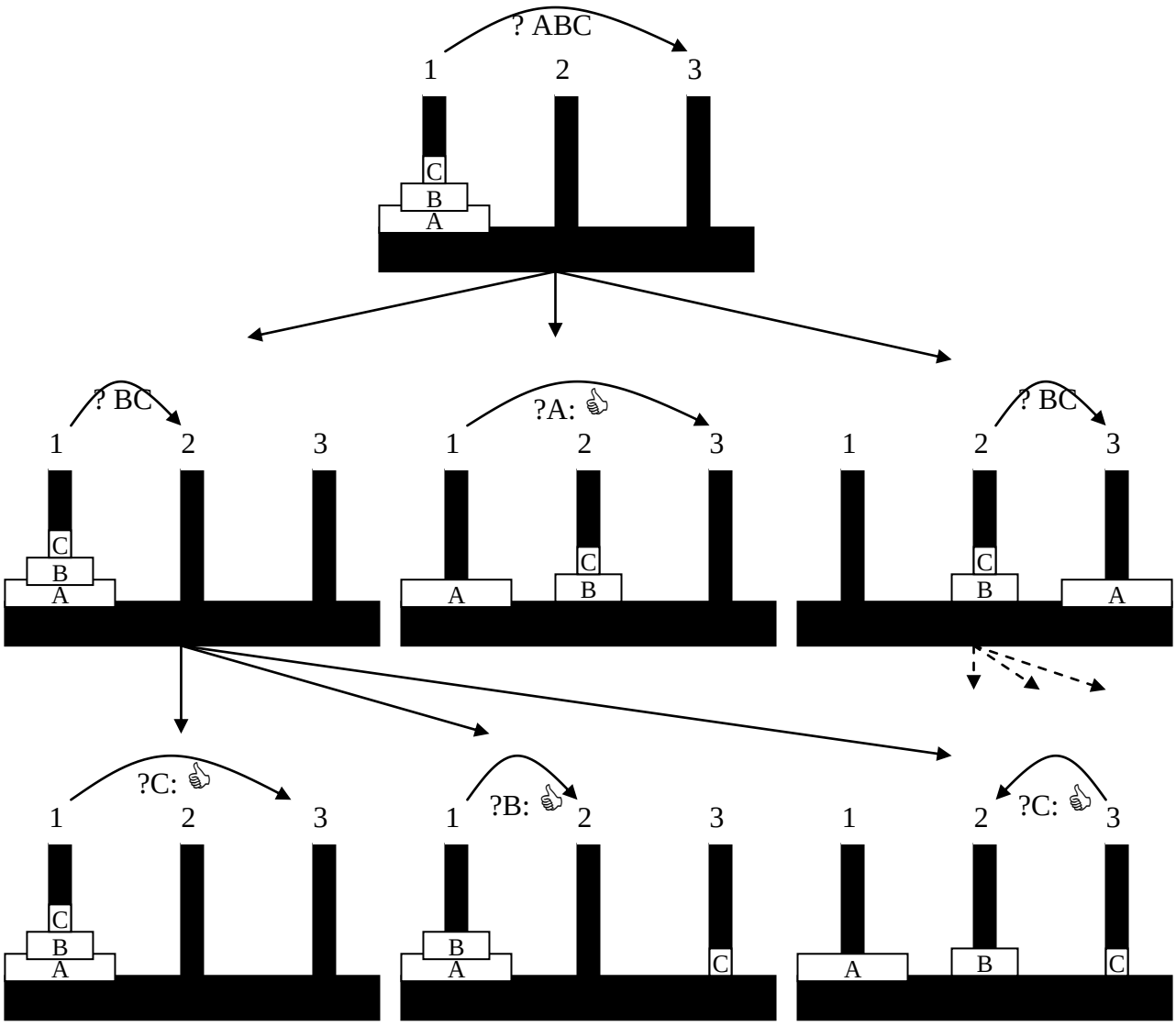
Hình 3.8 Quá trình tính 3! và số fibonacci thứ 3

Ví dụ 3.10: CT giải bài toán tháp Hà Nội

Bài toán tháp Hà Nội: "Có 3 cột '1', '2', '3' và chồng n đĩa đặt ở cột '1' có kích thước khác nhau, đĩa lớn ở dưới, đĩa nhỏ ở trên. Hãy chuyển các đĩa từ '1' sang '3' có thể dùng cột '2' làm trung gian, sao cho: a) mỗi lần chỉ chuyển được 1 đĩa, b) ở các cột luôn luôn đĩa lớn ở dưới, đĩa nhỏ ở trên.

Một phần quá trình giải bài toán tháp Hà Nội với n = 3 cho trong hình 3.9.

```
#include …
void    chuyen_1_dia(char x, char y);
void    thn(int n, char x, char y, char z);
```



Hình 3.9: Một phần quá trình giải bài toán tháp Hà Nội với n = 3

```
int     main (int argc[], char* arv[])
{
        int     n;
        char    x, y, z;
        printf (“So dia = ” ); scanf (“%d”, &n);
        thn (n, '1', ‘2', '3');
        getch(); return 0;
}
void    chuyen_1_dia(char x, char y)
{
        printf (“ Chuyen dia %c sang %c”, x, y);
}
void    thn(int n, char x, char y, char z)
{
        if (n = = 0)    return;
```

```
        thn (n-1,x,z,y);
        chuyen_1_dia (x,y);
        thn (n-1,z,y,x);
}
```

BÀI TẬP

1. Một hàm có thể trả về bao nhiêu giá trị? Nếu ta muốn một hàm trả về nhiều giá trị thì sao? Nếu một hàm không trả về giá trị, ta phải khai báo kiểu của nó như thế nào?
2. Dòng đầu tiên của định nghĩa hàm là gì, và nó chứa những thông tin gì? Cách đặt tên hàm như thế nào là tốt ?
3. Biến cục bộ là gì? Biến toàn cục là gì? Biến cục bộ và biến toàn cục khác nhau như thế nào? Tại sao không sử dụng tất cả các biến toàn cục?
4. Tìm điểm sai trong đoạn chương trình sau:

```
#include <iostream.h>
void  in_thong_diep ()
{
        printf (“Day la mot thong diep.”);
        return   0;
}
void main ()
{
         in_thong_diep (“Day la mot thong diep.”);
}
```

5. Tìm điểm sai trong định nghĩa hàm sau:

```
int  nhan_doi (int  y);
{
        return   (2 * y);
}
```

6. Viết một hàm nhận hai số nguyên int làm đối số, và trả về tích của hai đối số đó.
7. Viết một hàm nhận hai số nguyên int làm đối số, xuất tỷ số của đối số thứ nhất với đối số thứ hai. Nếu trường hợp không chia được (đối số thứ hai bằng 0) thì cho một câu thông báo “Không chia cho 0 được.”.
8. Viết các CT giải các bài toán tính giai thừa, tính số fibonacci thứ n, tháp hà nội, tính số tổ hợp chập k của n phần tử. Yêu cầu: dùng hàm đệ quy.
9. **Giải các bài tập trong chương 2 từ bài 33 đến bài 58 bằng phương pháp lập trình module (dùng hàm).**

CHƯƠNG 4: LẬP TRÌNH VỚI DỮ LIỆU CÓ CẤU TRÚC

Các bài toán trong thực tế thường phức tạp, đòi hỏi phải thao tác trên các loại dữ liệu là sự ghép nối (tổng hợp) của các kiểu dữ liệu đơn giản. Khi viết các CT chúng "không thể" được tách rời mà phải gộp chung lại mới có ý nghĩa. Kiểu dữ liệu có cấu trúc là nền tảng quan trọng để trừu tượng hóa và đóng gói dữ liệu.

Chẳng hạn, trong thực tế ta hay gặp các loại dữ liệu có tính cấu trúc như: học sinh (gồm: điểm trung bình, mã số, tuổi tác), số điện thoại (dãy các chữ số), ngày tháng năm (gồm: ngày, tháng, năm), …

✍ Kiểu dữ liệu có cấu trúc là kiểu dữ liệu mà mỗi giá trị của nó gồm nhiều thành phần, các thành phần có thể cùng hoặc khác kiểu, *kiểu thành phần có thể là kiểu đơn giản hoặc thậm chí là kiểu có cấu trúc khác*. Việc truy cập đến các thành phần có thể thông qua tên (kiểu struct) hoặc chỉ số (kiểu mảng).

Trong chương này, ta sẽ lập trình với các kiểu dữ liệu: mảng (một, hai chiều), struct.

4.1. Mảng một chiều

4.1.1 Khai báo, định nghĩa và một số thao tác đơn giản :

Mảng một chiều là dãy các thành phần (phần tử) cùng kiểu dữ liệu có quan hệ với nhau, chẳng hạn: mảng số nguyên (thực, ký tự) chứa các thành phần có kiểu số nguyên (số thực, ký tự).

C^{++} chỉ trang bị phép toán truy cập đến từng thành phần của mảng, LTV phải tự thiết kế các thao tác trên kiểu dữ liệu mảng.

✍ Để định nghĩa kiểu mảng một chiều, ta dùng từ khóa typedef:

typedef kiểu_phần_tử tên_kiểu_mảng [kích_thước_kiểu_mảng];

✍ Để khai báo một biến mảng (bien_mang) có kiểu mảng **(tên_kiểu_mảng)**:

tên_kiểu_mảng bien_mang;

Mỗi phần tử của mảng là một biến có kiểu_phần_tử. Các phần tử của mảng được đánh số thứ tự từ 0 đến kích_thước – 1. Để truy cập đến phần tử thứ i (i phải có kiểu nguyên), ta dùng cú pháp:

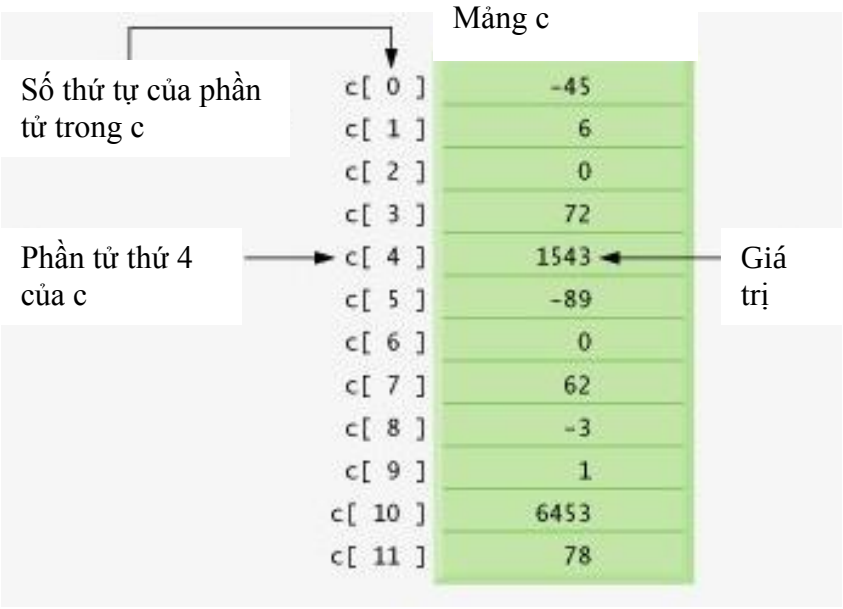
tên_biến_mảng [i]

✋ Hình 4.1 chỉ ra mảng số nguyên c chứa 12 phần tử: c[0], c[1], .., c[11]. Giá trị của chúng lần lượt là: -45, 6, 0, 72, 1543, -89, 0, … Để khai báo c, trước hết ta định nghĩa kiểu mảng nguyên (mang_nguyen) có kích thước lớn hơn hoặc bằng 12, sau đó khai báo c thuộc kiểu mang_nguyen.

- *typedef int mang_nguyen[100];*
- *mang_nguyen c;*

Mảng c chứa 99 phần tử, nhưng ta chỉ sử dụng 12 phần tử đầu tiên. Do đó, ta phải duy trì thêm biến n_c (có giá trị là 12) chỉ ra số phần tử đang sử dụng:

- *int n_c;*



Hình 4.1 Mảng số nguyên c chứa 12 phần tử

Có thể khai báo trực tiếp c như sau:

```
int     c[12];
```

Ví dụ 4.1: tạo mảng d chứa 10 số thực 0.0 và xuất d ra màn hình

```
typedef   float   mang_thuc[100];                    // (1)
mang_thuc   d;                                       // (2)
int           n_d = 10;     // số phần tử d đang sử dụng  (3)
for (int i=0; i<n_d; i++)   d[i] = 0.0;              // (4)
for (int i=0; i<n_d; i++)   printf("%f", d[i]);      // (5)
```

✍ Có thể vừa tiến hành khai báo mảng và gán các giá trị ban đầu cho một số (các phần tử còn lại có giá trị mặc định là 0.0) các phần tử của mảng theo cú pháp:

ten_kieu_mang bien_mang = {gia_tri_phan_tu_0, gia_tri_phan_tu_1, ..};

Chẳng hạn, nếu muốn vừa khai báo d vừa gán các giá trị ban đầu cho 5 phần tử đầu tiên của d (ở ví dụ 4.1) có giá trị là 0.0, 1.0, 2.0, 3.0, 4.0 ta thay lệnh (2) bằng lệnh (2'):

```
mang_thuc   d = { 0.0, 1.0, 2.0, 3.0, 4.0 };
```

Lúc đó, nếu bỏ câu lệnh (4), kết quả (thực hiện câu lệnh (5)) xuất d ra màn hình là: *0.0, 1.0, 2.0, 3.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0.*

Có thể dùng dạng khai báo trực tiếp d (thay lệnh (1) và (2) bằng lệnh (2')):

```
float   d[12] = { 0.0, 1.0, 2.0, 3.0, 4.0 };
```

✋ Nếu khai báo trực tiếp mảng e mà không chỉ ra số lượng phần tử của e thì bắt buộc phải đặt giá trị ban đầu cho các phần tử của e. Lúc đó, số lượng phần tử của e là số giá trị được liệt kê. Chẳng hạn, câu lệnh

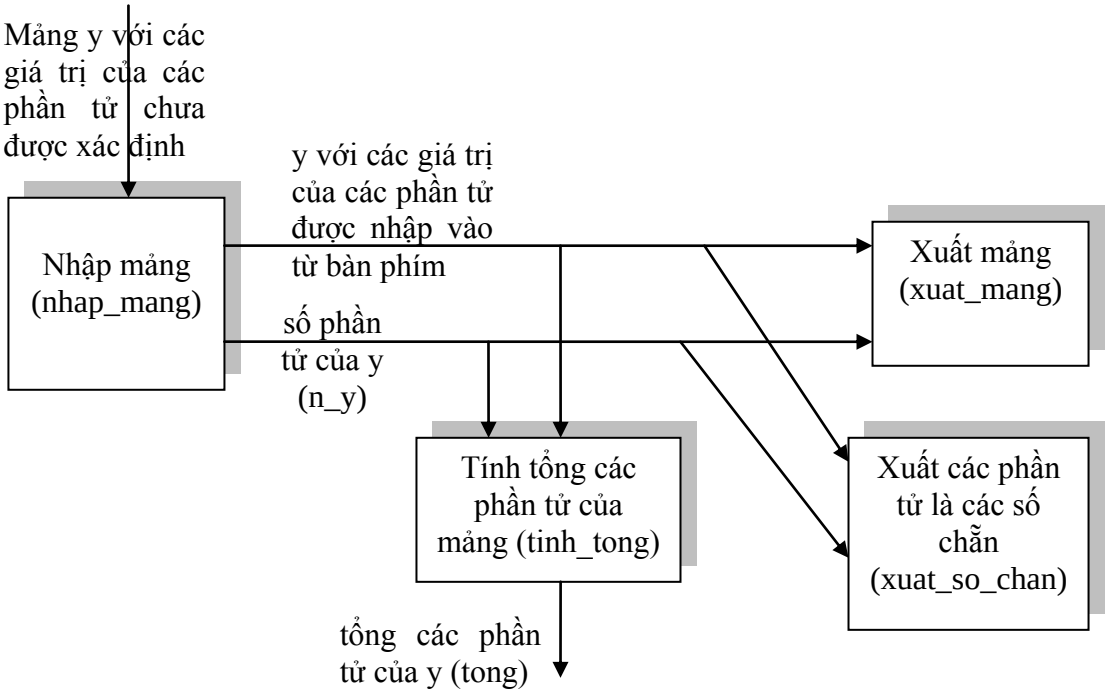
float e[] = { 0.0, 1.0, 2.0, 3.0, 4.0 }; // (6)

sẽ tạo ra một mảng 5 phần tử với các giá trị ban đầu của các phần tử là *0.0, 1.0, 2.0, 3.0, 4.0*. Câu lệnh (6) tương đương với câu lệnh (7) sau:

float e[5] = { 0.0, 1.0, 2.0, 3.0, 4.0 }; // (7).

Ví dụ 4.2: Một phần CT minh họa một số thao tác cơ bản trên mảng các số nguyên: nhập mảng chứa n số nguyên, xuất mảng, tính tổng các phần tử của mảng, xuất ra các phần tử là các số chẵn trong mảng.

Sơ đồ CT:




```
#include …
#define   KICH_THUOC   100
typedef   int   mang_nguyen [KICH_THUOC];

void   nhap_mang   (mang_nguyen  y, int &n_y);
void   xuat_mang    (mang_nguyen  y, int n_y);
int    tinh_tong     (mang_nguyen  y, int n_y);
void   xuat_so_chan (mang_nguyen  y, int n_y);
int    la_so_chan    (int i);

int    main(int argc, char* argv[])
{
       mang_nguyen a;   int  n, tong;
       nhap_mang (a, n);   xuat_mang  (a, n);
       tong  = tinh_tong (a, n); printf ("\n Tong cac phan tu trong mang la: %d", tong);
```

```
        xuat_so_chan (a, n);
        getch (); return 0;
}
void   nhap_mang    (mang_nguyen  y, int &n_y)
/*/
```

✋ Người đọc có thể biện luận rằng: "Các giá trị của các phần tử trong *a* (đối số thực) chưa được xác định trước khi thực hiện hàm nhap_mang. Sau khi thực hiện hàm nhap_mang thì các giá trị của các phần tử trong a là các số nguyên được nhập vào từ bàn phím. Do đó, phải truyền tham chiếu đến a cho y: *mang_nguyen &y*."

✍ Thực ra, **ta chỉ cần truyền tham trị từ a đến y** (y sẽ nhận địa chỉ của mảng *a* – địa chỉ của phần tử đầu tiên của *a*) để có thể nhập các giá trị của các phần tử trong *a* thông qua y.

Dĩ nhiên, n_y phải là tham chiếu đến n (đối số thực) để nhập được n từ hàm nhap_mang thong qua (tên hình thức) n_y.

```
/*/
{
        printf ("\n Nhap so phan tu cua mang : ");
        scanf ("%d", &n_y);
        for (i=0 ; i < n_y; i++) {
                printf ("\n Nhap phan tu thu %d:", i);
                scanf ("%d", &y[i]);
        }
}

void   xuat_mang     (mang_nguyen  y, int n_y)
{
        for (i=0 ; i < n_y; i++)        printf ("\t%d", y[i]);
}
int     tinh_tong (mang_nguyen  y, int n_y)
{
        int  tong = 0;
        for (i=0 ; i < n_y; i++)     tong = tong + y[i];
        return tong;
}
int     la_so_chan (int i)              // hàm kiểm tra số chẵn
{
        int     kq;     // kq = 1: i là số chẵn, kq = 0: i là số lẽ
        // ???
        return  kq;
}
void   xuat_so_chan (mang_nguyen  y, int n_y)
{
        // ???
}
```

Ví dụ 4.3: Một phần CT kiểm tra tần suất xuất hiện của mỗi mặt khi gieo một con xúc xắc 6000000 lần.

```
#include …
#include "stdlib.h"   // Chứa các hàm: srand, rand
#include "time.h"     // Lấy thời gian hệ thống

int      main(int argc, char* argv[])
{
        const  int  kich_thuoc       = 7;            // không dùng phần tử thứ 0
        int  tan_suat[ kich_thuoc ] = { 0 };         // đặt các giá trị 0 cho các phần tử

        // ✍ Khởi tạo bộ sinh số ngẫu nhiên (mỗi lần chạy CT mỗi khác): hạt giống của
        // bộ sinh số phụ thuộc vào thời gian của hệ thống.
        srand( time(0) );

        int      mat_xuat_hien;
        for (lan_tung = 1; lan_tung <= 6000000; lan_tung++ )
        {
                mat_xuat_hien =  rand() % 6  + 1;
                // ✍ Hàm rand() tạo một số nguyên ngẫu nhiên.
                // mat_xuat_hien là một số nguyên ngẫu nhiên trong đoạn [1, 6].

                tan_suat [mat_xuat_hien] ++;
        }

        for (mat = 1; mat < kich_thuoc; mat ++ )
              printf ("\nMat %d xuat hien %d lan", mat, tan_suat[mat]);

        getch(); return 0;
}
```

Kết quả chạy CT:

Lần 1

```
Mat 1 xuat hien 1000389 lan.
Mat 2 xuat hien 1000702 lan.
Mat 3 xuat hien 999793 lan.
Mat 4 xuat hien 1001797 lan.
Mat 5 xuat hien 999900 lan.
Mat 6 xuat hien 997419 lan._
```

Lần 2

```
Mat 1 xuat hien 1000199 lan.
Mat 2 xuat hien 1000424 lan.
Mat 3 xuat hien 999325 lan.
Mat 4 xuat hien 998813 lan.
Mat 5 xuat hien 1000586 lan.
Mat 6 xuat hien 1000653 lan.
```

Lần 3

```
Mat 1 xuat hien 999692 lan.
Mat 2 xuat hien 1000799 lan.
Mat 3 xuat hien 999702 lan.
Mat 4 xuat hien 999850 lan.
Mat 5 xuat hien 1000458 lan.
Mat 6 xuat hien 999499 lan.
```

Lần 4

```
Mat 1 xuat hien 1000256 lan.
Mat 2 xuat hien 999638 lan.
Mat 3 xuat hien 998927 lan.
Mat 4 xuat hien 1001208 lan.
Mat 5 xuat hien 1000695 lan.
Mat 6 xuat hien 999276 lan._
```

4.1.2 Các thao tác thường gặp trên mảng

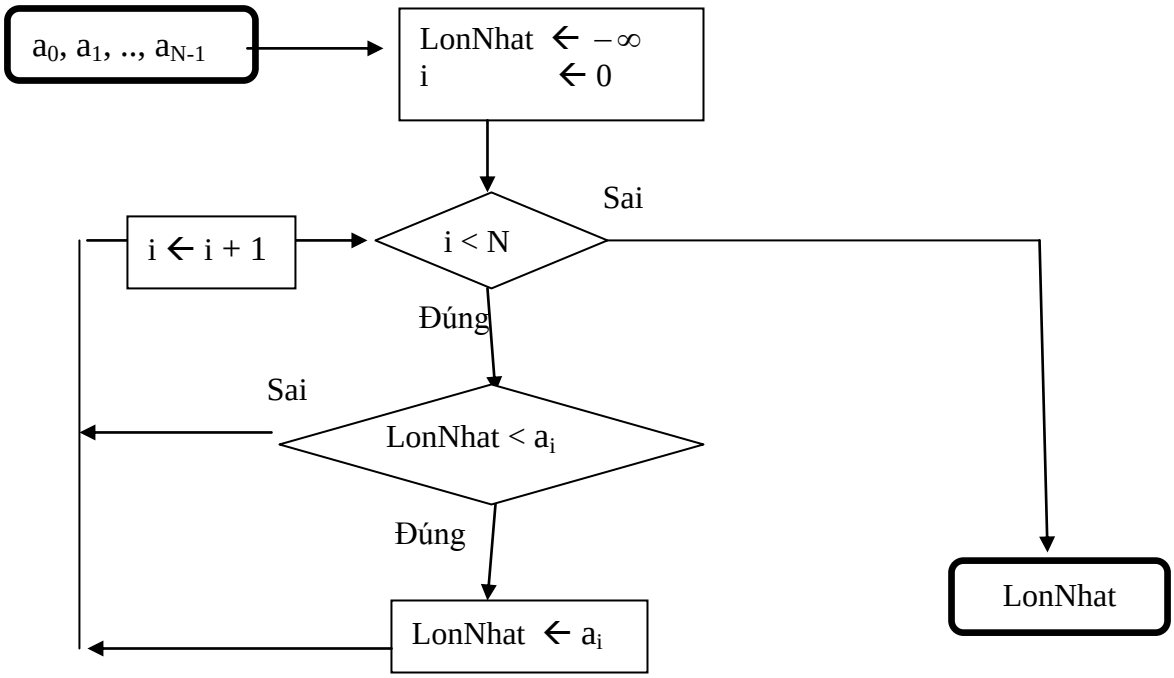
Các thao tác thường gặp trên mảng là tìm vị trí của phần tử lớn nhất (nhỏ nhất), tìm vị trí xuất hiện của phần tử trong mảng, sắp xếp mảng (tăng dần hoặc giảm dần).

Trong mục này, chúng ta sẽ phân tích và cài đặt chúng thành các hàm. Không mất tổng quát ta sẽ thao tác trên mảng số nguyên (mang_nguyen).

4.1.2.1 Tìm phần tử lớn nhất trong mảng

Đầu vào: Mảng $a_0, a_1, .., a_{N-1}$.

Đầu ra: LonNhat = max $\{a_0, a_1, .., a_{N-1}\}$.



Hình 4.2: Thuật toán tìm phần tử lớn nhất trong một mảng.

Giá trị $-\infty$ là giá trị vô hạn không thể biểu diễn trong máy tính (hữu hạn). Ta định nghĩa hằng VoCung mang giá trị bé nhất mà máy tính (NNLT) hiểu được, chẳng hạn: là -2,147,483,648 nếu $a_1, a_2, .., a_N$ là các số nguyên được lưu trữ bởi kiểu số nguyên lớn có dấu; là $-1.7*10^{-308}$ nếu $a_1, a_2, .., a_N$ là các số thực được lưu trữ bởi kiểu số thực lớn.

Để phạm vi sử dụng của hàm lớn hơn, ta sẽ tìm vị trí của phần tử lớn nhất trong mảng con {a[dau], .., a[cuoi]} của mảng a. Hàm được cài đặt như sau:

```
int     tim_vt_ptlonnhat (mang_nguyen  a, int  dau, int  cuoi)
{
        int  vt_ln = dau;
        for (int j=dau+1; j<=cuoi; j++)
                if (a[j] > a[vt_nn])  vt_ln = j;
        return  vt_ln;

}
```

Bài tập: Viết hàm tìm vị trí của phần tử nhỏ nhất trong mảng con {a[dau], .., a[cuoi]} của mảng a.

4.1.2.2 Sắp xếp giảm mảng

Đầu vào: Mảng a_0, a_1, .., a_{N-1}.

Đầu ra: Mảng được sắp xếp giảm.

Có nhiều thuật toán sắp xếp mảng (trong các giáo trình "Cấu trúc dữ liệu và thuật toán"), ở đây ta chỉ tìm hiểu thuật toán chọn trực tiếp.

Thuật toán trải k bước (k=0, .., N-2). Bước k: tìm vị trí của phần tử lớn nhất trong mảng [k..N-1], đổi giá trị với phần tử ở vị trí k.

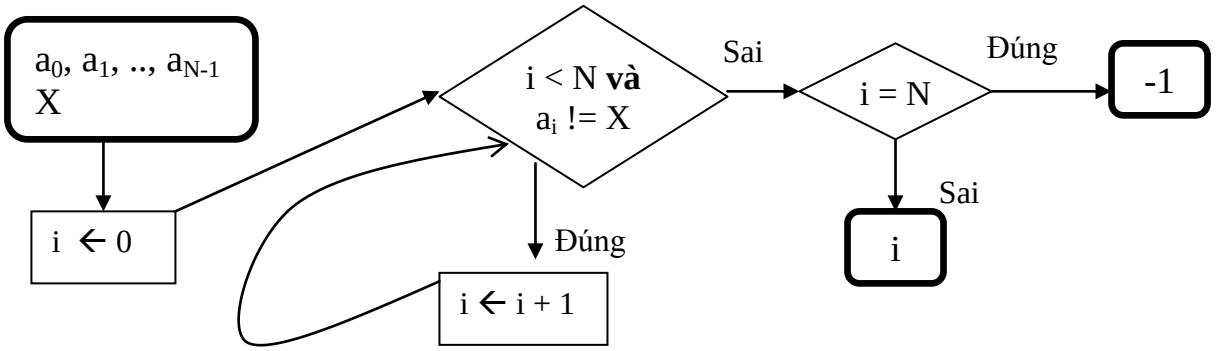
Hàm:

```
void   sx_chontructiep (mang_nguyen  a, int  n)
{
        for (int k=0; k<n-1; k++)
        {
                int  vt_ln = tim_vt_ptlonnhat (a, k, n-1);
                if (vt_ln!= k)          // Đổi giá trị của a[k] và a[vt_ln]
                        hoan_vi (a[k], a[vt_ln]);
        }
}
```

Bài tập: Viết hàm sắp xếp tăng mảng con {a[dau], .., a[cuoi]} của mảng a.

4.1.2.3 Tìm kiếm vị trí của phần tử X trong mảng a_0, a_1, .., a_{N-1}

Nếu không có X trong mảng thì kết qủa là -1; ngược lại, trả ra vị trí của X trong a.



Hình 4.3: Thuật toán tìm kiếm vị trí của phần tử trong mảng.

Cài đặt:

```
int     tim_kiem   (mang_nguyen  a, int  n, int  x)
{
        int i = 0, vt;
        while (i<n && a[i] != x)  i++;
```

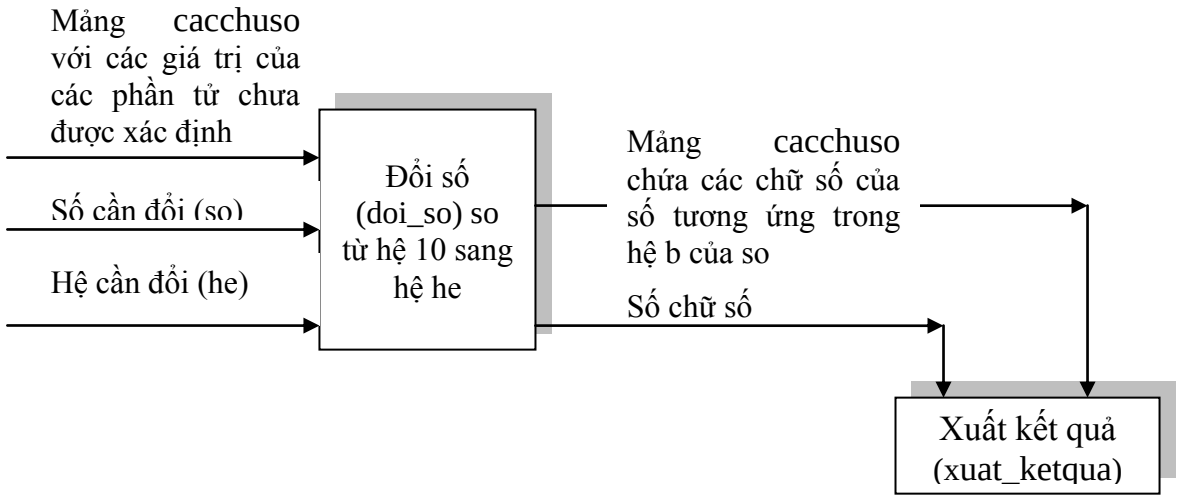
```
        if (i = = n)    vt = -1;        // Không có x trong a
        else            vt = i;         // Có x trong a ở vị trí i
        return vt;
}
```

4.1.3 Phân tích một số CT ứng dụng đơn giản:

4.1.3.1 Đổi số từ hệ 10 sang hệ cơ số b

Trong 3.3.10 ta đã phân tích CT đổi số từ hệ 10 sang hệ b. Hạn chế của CT này là kết quả số trong hệ cơ số b bị đảo ngược (chẳng hạn kết quả đổi số 4 trong hệ 10 của CT là 001 (kết quả đúng phải là 100)). Lý do là lúc đó (ở chương 3) ta chưa có kiểu dữ liệu mảng để lưu các chữ số (số dư) nhận được trong quá trình đổi. Nếu đã lưu trữ các chữ số vào mảng ta có thể duyệt mảng theo chiều từ phải sang trái (từ phần tử cuối cùng lùi lại đến phần tử đầu tiên) để nhận được kết quả đúng như mong muốn.

Sơ đồ CT:




```
#include …
typedef   int   mang_nguyen[100];
void    xuat_ky_so (int ky_so, int  he);
void    doi_so (int so, int he, mang_nguyen  cacchuso, int  &sochuso);
void    xuat_ketqua (mang_nguyen  cacchuso, int  sochuso, int  he);
int     main(int argc, char* argv[])
{
        int so, b;
        mang_nguyen  cacchuso;  int  sochuso;

        // Nhập số ở hệ 10 so và hệ cần đổi b ???
        doi_so (so, b, cacchuso, sochuso);
        xuat_ketqua (cacchuso, sochuso, b);

        getch (); return 0;
```

```
}
void   xuat_ky_so (int  ky_so, int  he)
{
        // ???
}
void   doi_so (int so, int he, mang_nguyen  cacchuso, int  &sochuso)
{
        sochuso = 0;
        do {
                cacchuso[sochuso] = so % he;
                sochuso++;   so /= he;
        } while (so != 0);
}
void   xuat_ketqua (mang_nguyen  cacchuso, int  sochuso, int  he)
{
        for(i=sochuso-1; i>=0; i--)
                xuat_ky_so (cacchuso[i], he);
}
```

4.1.3.2 Tìm các số nguyên tố bé hơn N bằng sàng Erastosthene

Erastosthene đã vẽ trên cát N – 1 ô đánh số từ 2 đến N (hay sàng) để tìm các số nguyên tố từ 2 đến N như sau:

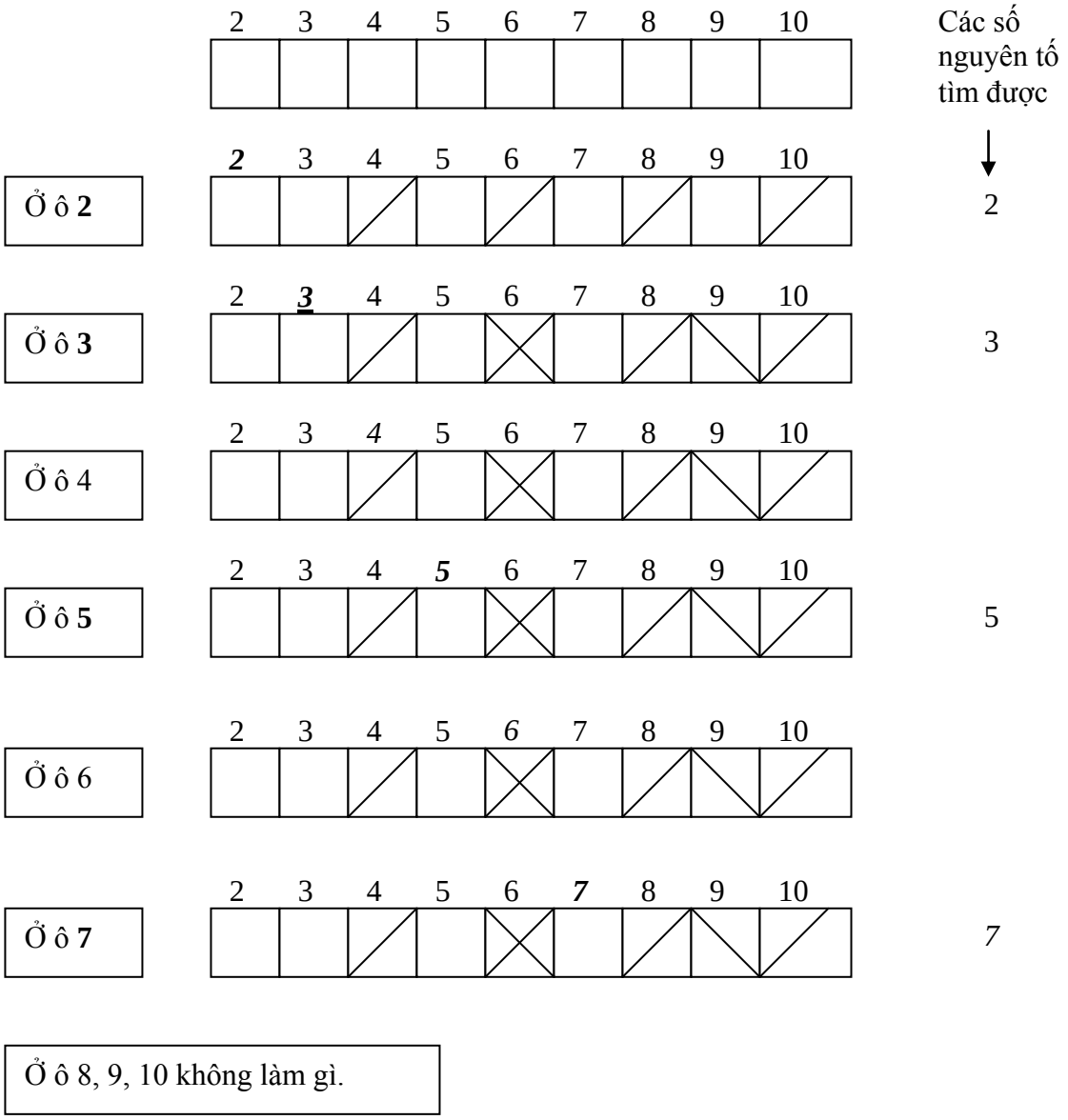
- gặp một ô chưa bị đánh dấu bỏ thì số thứ tự tương ứng với ô đó là số nguyên tố (2 là số nguyên tố đầu tiên)
- khi tìm được số nguyên tố i, đánh dấu chéo bỏ hết các ô có số thứ tự tương ứng là bội của i.

Quá trình tìm các số nguyên tố bé hơn 10 cho trong hình 4.4.

Sàng S là một mảng các số nguyên N+1 phần tử (bỏ phần tử thứ 0 và 1) với quy ước sau khi sàng xong: S[i] = 0 nếu i không là số nguyên tố, S[i] = 1 nếu i là số nguyên tố. .

```
#include …
typedef   int   mang_nguyen [100];
void            tao_sang_Erastosthene (mang_nguyen S, int N);
void            bo_boi (mang_nguyen S, int N, int i);
void            sang_Erastosthene  (mang_nguyen S, int N);
void            xuat_sang_Erastosthene  (mang_nguyen S, int N);
int     main(int argc, char* argv[])
{
        int N; mang_nguyen S;
        printf (“In cac so nguyen to tu 1 den N bang sang Erastosthene, N = ? ”);
        scanf (“%d”, &N);
        tao_sang_Erastosthene (S, N);
        sang_Erastosthene  (S, N);
```

```
        printf ("Cac so nguyen to tu 2 den %d bang sang Erastosthene:", N);
        xuat_sang_Erastosthene  (S, N);
        getch (); return 0;
}
void    tao_sang_Erastosthene (mang_nguyen S, int N)
{
        // S[i] = 1, với i = 2, .., N ???
}
void    bo_boi (mang_nguyen S, int N, int i)
{
        // Đánh dấu 0 cho các phần tử của S có chỉ số là bội số của i ???
}
```

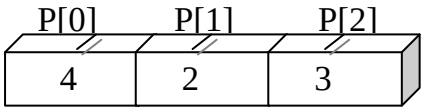


Hình 4.4 Quá trình tìm các số nguyên tố bé hơn 10

```
void    sang_Erastosthene  (mang_nguyen S, int N)
{
        for (i=2; i<=N; i++)
                // (a) nếu S[i] = 1 thì cho S[j] = 0 với j = 2*i, 3*i, 4*i, …,
                // (b) nếu S[i] = 0 thì chuyển sang bước i+1
                // ???
}
void    xuat_sang_Erastosthene  (mang_nguyen S, int N)
{
        // Xuất các chỉ số i có S[i] bằng 1 ???
}
```

4.1.3.3 Thao tác trên đa thức: nhập, xuất, tính đạo hàm cấp 1, định trị

Để lưu trữ đa thức $P = a_0x + a_1x^1 + \ldots + a_nx^n$ ta dùng mảng một chiều các số thực P với P[i] lưu hệ số a_i. Chẳng hạn: đa thức $P = 3x^2+2x+4$ được lưu bằng mảng các số thực P như sau:



Hình 4.5: Hình ảnh trong bộ nhớ lưu đa thức $3x^2+2x+4$.

Bảng phân tích các hàm:

STT	Nhiệm vụ	Đối số vào	Trả về
1	Nhập đa thức	Đa thức (mảng lưu trữ, cấp)	void
2	Xuất đa thức	Đa thức	void
3	Tính đạo hàm cấp 1	Đa thức	void
4	Định trị đa thức	Đa thức, giá trị x	float
5	Tính x mũ k	X, k	float

```
#include …
#define   KICH_THUOC    100
typedef   float    da_thuc [KICH_THUOC];

void    nhap_da_thuc (da_thuc p, int &cap);
void    xuat_da_thuc (da_thuc p, int cap);
void    dao_ham_cap_1 (da_thuc p, int &cap);
float   dinh_tri (da_thuc p, int cap, float x);
float   x_mu_k (float x, int k);

int     main(int argc, char* argv[])
{
```

```
        // Các khai báo biến ???
        nhap_da_thuc (p, cap);  xuat_da_thuc (p, cap);
        scanf ("%f", &x);
        px = dinh_tri (p, cap, x); printf ("\np(%f) = %f", x, px);
        dao_ham_cap_1 (p, cap); xuat_da_thuc (p, cap);
        getch (); return 0;
}
void    nhap_da_thuc (da_thuc p, int &cap)
{
        // Nhập cấp cap; sau đó nhập từng hệ số p[j], j=0, .., cap          ???
}
void    xuat_da_thuc (da_thuc p, int cap)
{
        for (j=0; j<=cap; j++)          printf ("%f * x^%d + ", p[j], j);
}
void    dao_ham_cap_1 (da_thuc p, int &cap)
{
        // ???
}
float   x_mu_k (float x, int k)        // Tính x^k
{
        float    kq = 1.0;
        // ???
        return kq;
}
float   dinh_tri (da_thuc p, int cap, float x)
{
        float    px = 0.0;
        for (j=0; j<=cap; j++)
                px += p[j] * x_mu_k (x, j);
        return px;
}
```

✋ Hàm xuất đa thức xuat_da_thuc trên cho kết quả không tốt? Hãy sửa đổi nó để có kết quả tốt hơn. *Bài tập.*

4.2. Chuỗi ký tự

4.2.1 Khai báo, định nghĩa và một số thao tác đơn giản:

Chuỗi ký tự là mảng một chiều các ký tự. Để quản lý chuỗi ký tự ta không lưu số ký tự có trong chuỗi mà dùng ký tự null '\0' để đánh dấu hết chuỗi.

Ví dụ 4.4: một số lệnh định nghĩa kiểu chuỗi, khai báo chuỗi

Khai báo	*Ý nghĩa*

char s1[] = "hello";	Khai báo chuỗi s1 chứa 5 ký tự theo thứ tự: 'h', 'e', 'l', 'l', 'o', '\0'.
char s2[] = {'h', 'e', 'l', 'l', 'o', '\0'};	Khai báo chuỗi s2 chứa 5 ký tự theo thứ tự: 'h', 'e', 'l', 'l', 'o', '\0'.
char s3[20];	Khai báo chuỗi s3 có thể chứa 20 ký tự (tính luôn cả ký tự '\0'). Các ký tự hiện có trong s3 là các ký tự ngẫu nhiên, s3 chưa có ký tự '\0' ở cuối.
typedef char chuoi[10];	Định nghĩa kiểu chuỗi có thể chứa 10 ký tự.
chuoi s4;	Khai báo chuỗi s4 có thể chứa 10 ký tự.

✍ Thư viện *string.h* chứa sẵn rất nhiều các hàm xử lý chuỗi. Tuy nhiên, do mục đích của giáo trình là trình bày các kỹ thuật lập trình, nên ta sẽ tự thiết kế các hàm xử lý trên chuỗi. Để nhập chuỗi ta dùng hàm **gets(biến_chuỗi)** trong thư viện *stdio.h*, để xuất chuỗi ta dùng lệnh **printf** với định dạng xuất %s.

Ví dụ 4.5: CT nhập vào một chuỗi s chứa tối đa 10 ký tự và xuất s ra màn hình.

Để chuỗi s có thể chứa tối đa 10 ký tự do người sử dụng nhập vào s phải được khai báo sao cho có thể chứa 11 ký tự vì ta phải dành một chỗ để chứa ký tự '\0'.

```
#include "stdio.h"
#include "conio.h"
typedef   char   chuoi [11];
int main(int argc, char* argv[])
{
        chuoi  s;
        printf ("\n Nhap chuoi: "); gets (s);
        printf ("\n Chuoi moi nhap la: %s", s);
        getch(); return 0;
}
```

4.2.2 Các thao tác thông thường trên chuỗi:

Trong phần này, ta sẽ thao tác trên các biến chuỗi có tên là s, t. Các hàm trong phần này được viết ở dạng mã giả.

4.2.2.1 Tính độ dài chuỗi

Duyệt qua các ký tự của chuỗi cho đến khi gặp ký tự '\0', trong mỗi bước tăng độ dài lên 1.

```
int     dodai (s)                                   // Hàm (1)
{
        dd = 0;
        for (i=0; s[i] != '\0'; i++)
                dd++;
        return dd;
```

}

4.2.2.2 So sánh hai chuỗi

Kết quả so sánh hai chuỗi s và t xảy ra ba trường hợp: =, <, >

- Chuỗi s = chuỗi t nếu:
  - s có độ dài (L) bằng t *và*
  - với mọi k = 0, .., L–1: s[k] = t[k].
- Chuỗi s < chuỗi t nếu:
  - s xuất hiện ở phần đầu của t, *hoặc*
  - Với mọi k = 0, .., m: s[k] = t[k]; s[m+1] < t[m+1]; m >= 0
- Ngược lại: chuỗi s > chuỗi t.

Ví dụ: "le van **b**" > "le van **a**", "tran **t**hinh" < "tran **v**an thinh", "a" = "a".

Thuật toán:

B1) So sánh tương ứng từng ký tự trên s và t theo thứ tự từ trái sang phải, nếu:

- ký tự trên s < ký tự trên t thì **kết luận s < t.**
- ký tự trên s > ký tự trên t thì **kết luận s > t.**
- ký tự trên s = ký tự trên t thì tiếp tục so sánh.

B2) Kết thúc so sánh, nếu:

- độ dài của chuỗi s > chuỗi t thì **kết luận s > t.**
- độ dài của chuỗi s = chuỗi t thì **kết luận s = t.**
- độ dài của chuỗi s < chuỗi t thì **kết luận s < t.**

Cài đặt:

Vào: hai chuỗi s, ta cần so sánh.

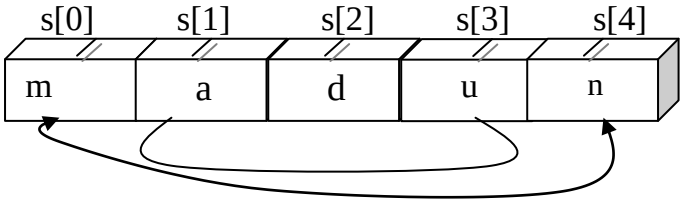
Ra: -1 nếu s < t, 0 nếu s = t, 1 nếu s > t.

```
int  so_sanh_chuoi (s, t)                    // Hàm (2)
{
    for (is = 0, it = 0; is<dodai(s) && it < dodai(t); i++, j++)
    {
        if (s[is] < t[it])  return -1;
        if (s[is] > t[it])  return  1;
    }
    if (dodai(s) == dodai(t))        return 0;
    else if (dodai(s) <  dodai(t))   return -1;
    else                             return 1;
}
```

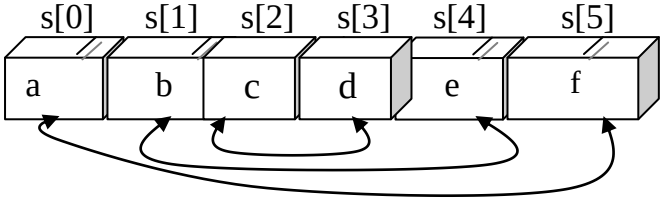
4.2.2.3 Đảo ngược chuỗi

Để đảo ngược chuỗi ta hoán đổi các cặp ký tự: đầu và cuối; ký tự sau ký tự đầu và ký tự kế cuối; ..; cặp ký tự ở giữa.

Ví dụ: s = "madun" có chuỗi đảo ngược là "nudam"



s = "abcdef" có chuỗi đảo ngược là "fedcba"



Thuật toán: L ← dodai(s);

- Đổi s[0] với s[L-1];
- Đổi s[1] với s[L-2];
- ..
- Đổi s[(L-1)/2] với s[L/2];

Cài đặt: Vào: chuỗi s cần đảo.

Ra: s đã đảo ngược.

```
void   dao_nguoc_chuoi (s)                    // Hàm (3)
{
        L = dodai(s);
        for(j = 0, k=L-1;   j < k;   j++, k--)
            đổi (s[k], s[j]);      // Đổi nội dung hai biến ký tự s[k] và s[j]
}
```

4.2.2.4 Xóa ký tự xuất hiện trong chuỗi

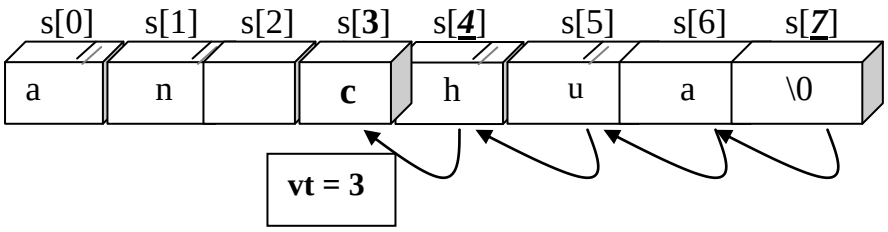
Ví dụ: chuỗi "an **c**hua" sau khi xóa ký tự **c** sẽ thành "an hua".

Để xóa ký tự kt trong chuỗi s ta thực hiện hai bước:

B1) Tìm vị trí **vt** xuất hiện của ký tự kt trong s.

B2) Dời các ký tự vt+1, vt + 2, .., dodai(s) sang trái một đơn vị.

Minh họa:



Cài đặt:

Vào: chuỗi s, ký tự kt cần xóa

Ra: chuỗi s đã xóa ký tự kt (xuất hiện đầu tiên)

```
int   tim_vt_ktu (s, kt)   // Hàm (4)
{
    for(i=0; i<dodai(s); i++)
        if (s[i] == kt) return i;
    return -1;   // khong co kt trong s
}
```

```
void  xoa_ktu (s, kt)          // Hàm (5)
{
        vt = tim_vt_ktu (s, kt);
        if (vt == -1) return;
        for(i=vt+1; i<=dodai(s); i++)
                s[i-1] = s[i];
}
```

Để xóa mọi ký tự kt xuất hiện trong chuỗi s, ta lặp lại việc xóa ký tự kt trong s cho đến khi không còn tìm thấy ký tự kt trong s. Cài đặt thuật toán như sau:

```
int  tim_vt_ktu (s, kt, bd)                    // Hàm (6)
{
        // tìm vị trí xuất hiện của ký tự kt trong s bắt đầu từ bd
        for(i=bd; i<dodai(s); i++)
                if (s[i] == kt)  return i;
        return -1;      // khong co kt trong s
}
int     xoa_ktu (s, kt, bd)                    // Hàm (7)
{
        // xóa ký tự kt đầu tiên xuất hiện trong s bắt đầu từ bd
        vt = tim_vt_ktu (s, kt, bd);
        if (vt == -1) return -1;       // không còn kt trong s
        for(i=vt+1; i<=dodai(s); i++)
                s[i-1] = s[i];
        return vt;                     // đã xóa kt trong s ở vị trí vt
}
void    xoa_moi_kytu (s, kt)                   // Hàm (8)
{
        bd = 0;
        while (bd >= 0)
                bd = xoa_ktu (s, kt, bd);
}
```

4.2.2.5 Chèn ký tự vào chuỗi

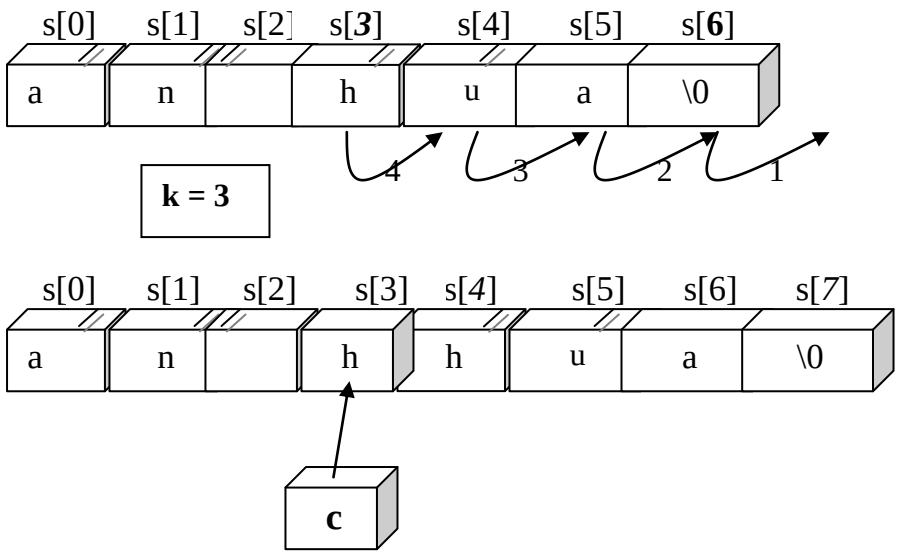
Để chèn ký tự kt vào chuỗi s ở vị trí k ta thực hiện hai bước:

B1) Dời các ký tự k, k+1, .., dodai(s) sang phải một đơn vị.

B2) Ghi kt vào s ở vị trí k.

Ví dụ: chèn ký tự **c** vào chuỗi "an hua" ở vị trí 3 cho ra chuỗi "an **c**hua".

Minh họa:



Cài đặt: Vào: chuỗi s, ký tự kt cần chèn, vị trí k cần chèn.

Ra: chuỗi s đã chèn thêm ký tự kt vào vị trí k

```
void chen_ktu (s, kt, k)                    // Hàm (9)
{
        for(i=dodai(s); i >= k; i--)
                s[i+1] = s[i];
        s[k] = kt;
}
```

4.2.2.6 Trích chuỗi con

Ta có thể trích ra chuỗi con t chứa n ký tự xuất hiện từ vị trí bđ của chuỗi s:

s = { s[0] s[1] … **s[bd] s[bd+1] .. s[bd+n-1]** … '\0' }

t = { **s[bd] s[bd+1] .. s[bd+n-1]** '\0'}

```
void trich_chuoi_con (s, bd, n, t)          // Hàm (10)
{
        for (i=0; i<n; i++)
        {
                t[i] = s[bd+i];
                if (s[bd+i] == '\0')  break;
        }
        t[i] = '\0';
}
```

4.2.2.7 Tìm chuỗi con xuất hiện trong chuỗi

Ví dụ: chuỗi con "*chua an*" xuất hiện ở vị trí thứ 8 trong chuỗi "toi con *chua an* com".

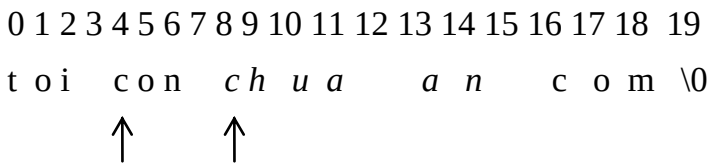
Từ nhận xét trên, ta có thuật toán tìm vị trí xuất hiện của chuỗi con t trong chuỗi s:

BĐ = 0; l_t = độ dài của t;

Trong khi (còn tìm được trong s (từ BĐ) vị trí **vt** *đầu tiên xuất hiện t[0])*

- *Trích chuỗi con r từ s chứa l_t ký tự bắt đầu từ vt.*
  - *nếu độ dài của r < l_t thì dừng thuật toán với kết luận không có t trong s.*
  - *nếu r bằng t thì dừng thuật toán với kết luận vị trí xuất hiện t trong s là **vt**.*
  - *Ngược lại, BĐ = vt+1.*

Minh họa:



Cài đặt:

```
int  tim_chuoi_con (s, t)                          // Hàm (11)
{
        bd = 0, l_t = dodai(t);
        vt = tim_vt_ktu (s, t[0], bd);
        while (vt != -1)
        {
                trich_chuoi_con (s, vt, l_t, r);
                if (l_t > dodai(r))
                        return -1;      // không có t trong s
                if (so_sanh_chuoi (t, r) == 0)
                        return vt;
                bd = vt+1;
                vt = tim_vt_ktu (s, t[0], bd);
        }
        return -1;      // không có t trong s
}
```

Ví dụ 4.6: Trích ra tên, họ và chữ lót từ họ và tên đầy đủ. Chẳng hạn với họ và tên đầy đủ "nguyen van teo", ta có kết quả trích gồm ba chuỗi ho = "nguyen", lot = "van", ten = "teo".

Để trích họ, chữ lót, tên từ chuỗi họ và tên s ta cần thực hiện bốn bước:

(B1) Tìm ra vị trí hai khoảng trắng phân cách: họ và chữ lót (vtkt_dautien), chữ lót và tên (vtkt_cuoicung).

(B2) Trích chuỗi con xuất hiện trong s tính từ vị trí thứ 0 đến vị trí vtkt_dautien-1 đưa vào ho.

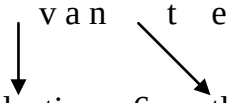
(B3) Trích chuỗi con xuất hiện trong s tính từ vị trí vtkt_dautien+1 đến vị trí vtkt_cuoicung-1 đưa vào lot.

(B4) Trích chuỗi con xuất hiện trong s tính từ vị trí vtkt_cuoicung+1 đến vị trí cuối cùng đưa vào ten.

Minh họa:

0 1 2 3 4 5 **6** 7 8 9 ***10*** 11 12 13 14

n g u y e n v a n t e o '\0'



(B1) vtkt_dautien = 6 vtkt_cuoicung = 10

(B2) trích chuỗi con [0, 5] vào họ.

(B3) trích chuỗi con [7, 9] vào lót.

(B4) trích chuỗi con [11, 13] vào tên.

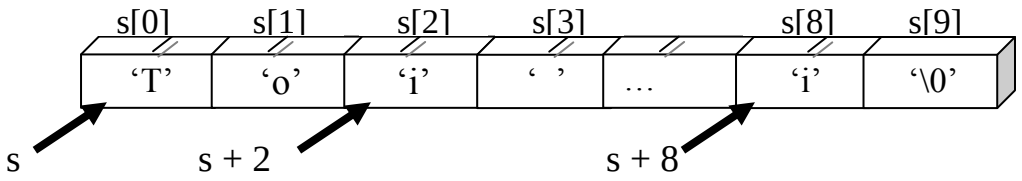
Hàm (6) giúp ta tìm được vtkt_dautien, vtkt_cuoicung. Hàm (10) giúp ta trích chuỗi con từ chuỗi mẹ. Để trích chuỗi con xuất hiện trong đoạn từ [a, b], ta gọi *trich_chuoi_con (s, a, b-a+1, t).*

```
#include …
typedef   char   chuoi [100];
int       trich_ho_lot_ten (chuoi s, chuoi h, chuoi l, chuoi t);
int       tim_vt_ktu (chuoi s, char kt, int bd);                    // Hàm (6)
void      trich_chuoi_con (chuoi s, int bd, int n, chuoi t);        // Hàm (10)
int       main(int argc, char* argv[])
{
          chuoi s, ho, lot, ten;
          // Nhập chuỗi s chứa họ và tên đầy đủ ???
          trich_ho_lot_ten (s, ho, lot, ten);
          // Xuất ho, lot, ten ???
          getch (); return 0;
}
int       tim_vt_ktu (chuoi s, char kt, int bd)
{
          // ???
}
void      trich_chuoi_con (chuoi s, int bd, int n, chuoi t)
{
          // ???
}
int       trich_ho_lot_ten (chuoi s, chuoi h, chuoi l, chuoi t)
{
          vtkt_dautien    =  tim_vt_ktu (s, ' ', 0);
          if (vtkt_dautien = = -1)
                    return 0;        // Không trích được họ, lót, tên từ s vì s không có ký tự ' '
          vtkt_cuoicung =  tim_vt_ktu (s, ' ', vtkt_dautien+1);
          if (vtkt_cuoicung = = -1)
                    return 0;        // Không trích được họ, lót, tên từ s vì s không có phần lót
          trich_chuoi_con (s, 0, vtkt_dautien-1, h);
          trich_chuoi_con (s, vtkt_cuoicung+1, dodai(s)-1, h);
          return 1;
```

```
}
```

Hàm trich_ho_lot_ten được cài đặt như trên không xử lý được hai trường hợp. Thứ nhất, khi s không có chữ lót, hàm phải trả về chuỗi l rỗng. Thứ hai, khi s có nhiều chữ lót, hàm phải trả về tất cả các chữ lót (vào trong một mảng các chuỗi). Hãy cải tiến hàm trên. *Bài tập*.

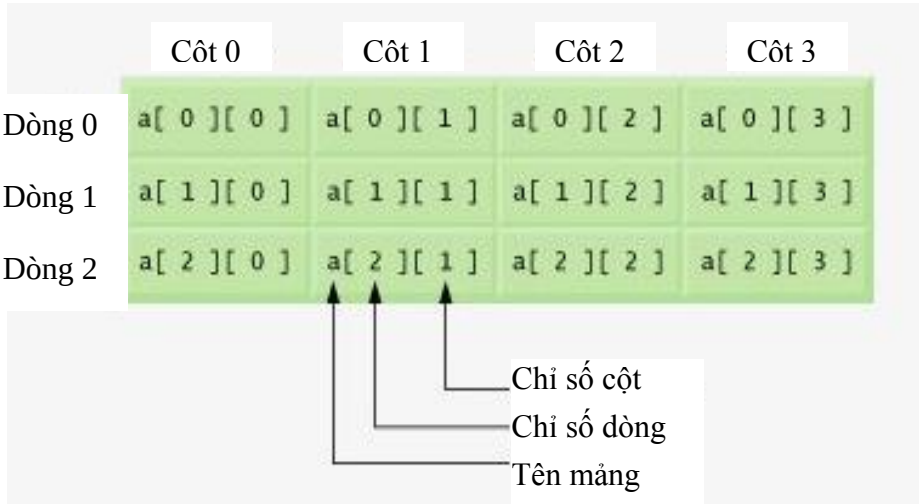
✍ Chú ý: có thể lấy chuỗi con bắt đầu từ ký tự thứ j đến ký tự cuối cùng của chuỗi s bằng cú pháp: **s + j.**



Hình 4.6: Cách truy cập đến chuỗi con trong chuỗi.

4.3. Mảng nhiều chiều

Mảng hai chiều là mảng có nhiều phần tử, mỗi phần tử là một mảng một chiều khác. Có thể xem mảng hai chiều là một bảng (các hàng, các cột chỉ định các ô). Mảng ba chiều là mảng có nhiều phần tử, mỗi phần tử là một mảng hai chiều. Trong thực tế, mảng hai chiều được sử dụng rộng rãi.



Hình 4.7 Minh họa một mảng hai chiều với 3 dòng và 4 cột

✍ Để định nghĩa kiểu mảng hai chiều các phần tử thuộc kiểu dữ liệu KDL với số dòng tối đa SD và số cột tối đa SC ta viết:

typedef KDL TênKiểuMảngHaiChiều [SD] [SC];

Tương tự với trường hợp mảng một chiều, thường ta không sử dụng hết kích thước của mảng hai chiều mà chỉ sử dụng các dòng, và các cột đầu tiên, ta cần khai báo các biến nguyên lưu trữ số dòng, số cột thực sự này.

Khai báo mảng hai chiều:

TênKiểuMảngHaiChiều tên_biến;

Có thể khai báo trực tiếp (và đặt giá trị ban đầu) mảng hai chiều tương tự trong trường hợp mảng một chiều.

Ví dụ 4.7: Một số khai báo một mảng hai chiều và ý nghĩa

- int b[2][2] = { { 1, 2 }, { 3, 4 } };
  b có 2 dòng và hai cột (dòng 0 chứa các số nguyên 1, 2; dòng 1 chứa các số nguyên 3, 4).
- int b[2][2] = { { 1 }, { 3, 4 } };
  b[0][0] = 1, b[0][1] = **0**, b[1][0] = 3 và b[1][1] = 4.
- int b[2][3] = { 1, 2, 3, 4, 5 };
  b[0][0] = 1, b[0][1] = 2, b[0][2] = 3, b[1][0] = 4, b[1][1] = 5, và b[1][2] = **0**.

✋ Trong thực tế, mảng hai chiều có nhiều ứng dụng: cài đặt các thao tác, phép toán trên ma trận; cài đặt cấu trúc bảng dùng trong quy hoạch động (xem giáo trình Thiết kế và đánh giá thuật toán), …

Ví dụ 4.8: Một số thao tác (được viết dưới dạng mã giả) trên ma trận thực nhập, xuất, tính tổng và tích của hai ma trận thực; kiểm tra ma trận tam giác trên.

Định nghĩa kiểu ma trận thực:

```
typedef   float   ma_tran_thuc [100][100];
```

✓ Nhập ma trận a:

```
void   nhap_ma_tran (a, &m, &n)
{
        // Nhập số dòng m, số cột n ???
        for (i=0; i<m; i++)
                for (j=0; j<n; j++) {
                        printf ("[%d][%d] = ? ", i, j);
                        scanf ("%f", a[i][j]);
                }
}
```

Ở đây ta không cần truyền tham chiếu đến a? Vì sao?

✓ Xuất ma trận a:

```
void   xuat_ma_tran (a, m, n)
{
        for (i=0; i<m; i++) {
                printf ("\n");
                for (j=0; j<n; j++)
                        printf ("%4.1f", a[i][j]);
        }
}
```

Chuỗi "%4.1f" định dạng xuất số thực trong phạm vi 4 cột canh lề phải với một số lẻ.

✓ Tính tổng hai ma trận (c = a + b):

```
void    tong_ma_tran (a, m, n, b, c)
{
        for (i=0; i<m; i++)
                for (j=0; j<n; j++)
                        c[i][j] = a[i][j] + b[i][j];
}
```

✓ Tính tích hai ma trận (c = a * b):

```
int     tich_ma_tran (a, ma, na, b, mb, nb, c, &mc, &nc)
{
        if (na != mb)
                return 0;       // Không thể nhân a với b.
        mc = ma; nc = nb;
        for (i=0; i<mc; i++)
                for (j=0; j<nc; j++)  {
                        c[i][j] = 0;
                        for (k=0; k<na; k++)
                                c[i][j] += a[i][k] + b[k][j];
                }
        return 1;
}
```

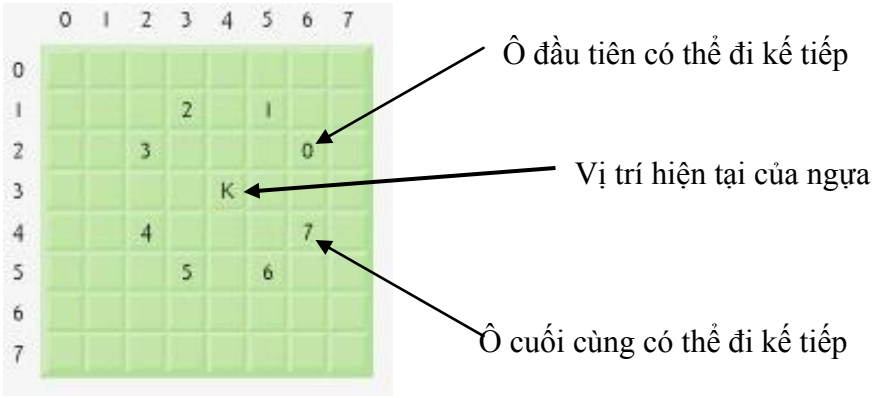
✓ Kiểm tra a có phải là ma trận tam giác trên không?

a là tam giác trên khi tất cả các phần tử của a nằm dưới đường chéo chính đều bằng 0 hay tồn tại một phần tử của a nằm dưới đường chéo chính khác 0.

```
int     ktra_mtran_tgiactren (a, m, n)
{
        if (m != n)             // a không phải ma trận vuông
                return 0;       // a không là ma trận tam giác trên
        int  tgt = 1;
        for (i=0; i<m; i++)
                for (j=0; j<i; j++)
                        if (a[i][j] != 0.0) {
                                tgt = 0;
                                break;
                        }
        return  tgt;
}
```

Hãy viết CT cài đặt tất cả các thao tác trên ma trận: chuyển vị, kiểm tra ma trận đối xứng, ... *Bài tập*.

Ví dụ 4.8: Xác định các ô có thể đi kế tiếp của ngựa (mã) theo luật cờ vua.



Hình 4.8 Các ô có thể đi kế tiếp của ngựa

Ngựa có thể đến tối đa 8 kế tiếp có chênh lệch về dòng và cột so với vị trí hiện tại lưu trong hai mảng lech_dong, lech_cot:

```
int    lech_dong [8] = {-1,  -2,  -2,  -1, +1, +2, +2, +1};
int    lech_cot   [8] = {+2, +1, -1,  -2,  -2,  -1, +1, +2};
```

Với vị trí hiện tại là (i, j), ô thứ c có thể đi kế tiếp của ngựa là (i+lech_dong[c], j+lech_cot[c]). Dĩ nhiên, ô (i+lech_dong[c], j+lech_cot[c]) phải thuộc bàn cờ.

Dùng mảng hai chiều a lưu bàn cờ với cách sau:

- a[i][j] = -1, nếu (i, j) là vị trí hiện tại của ngựa
- a[i][j] = c, nếu (i, j) là ô kế tiếp thứ c của ngựa
- a[i][j] = -2, nếu ngược lại

CT xác định các ô có thể đi kế tiếp của ngựa được xây dựng dựa trên 2 hàm sau:

✓ Hàm xác định các ô có thể đi kế tiếp của ngựa:

Ban đầu, $a[i][j] = 0, i = \overline{0,7}, j = \overline{0,7}$.

```
int     xdinh_cacoke_ngua (a, dong_htai, cot_htai)
{
        a[dong_htai][cot_htai] = -1;
        o_hop_le_thu = 0;
        for (c=0; c<8; c++) {
                dong_ke = dong_htai + lech_dong[c];
                cot_ke  = cot_htai  + lech_cot  [c];
                if ((dong_ke < 0 || dong_ke > 7) || (cot_ke < 0 || cot_ke > 7)) {
                        a[dong_ke][cot_ke] = o_hop_le_thu;
                        o_hop_le_thu++;
                }
        }
        return o_hop_le_thu;        // Trả về số ô có thể đi kế tiếp
}
```

- ✓ Hàm xuất ra bàn cờ thể hiện vị trí hiện tại của ngựa và các ô có thể đi kế tiếp của ngựa

```
void    xuat_ban_co (a)
{
        for (i=0; i<8; i++)
                for (j=0; j<8; j++)
                        switch (a[i][j]) {
                                case -2:  printf ("%3c", '_');      break;
                                case -1:  printf ("%3c", 'K');      break;
                                default:  printf ("%3d", a[i][j]); break;
                        }
}
```

Chuỗi "%3c" ("%3d"): định dạng xuất ký tự (số nguyên) trong phạm vi 3 cột canh lề phải.

Bài tập:

- Có thể tổng quát CT trên trên bàn cờ kích thước nxn được không?
- Viết CT xác định các ô bị chiếm dụng bởi một hậu trên bàn cờ vua.

☞ Mảng 3 chiều là mảng mà mỗi phần tử là một mảng 2 chiều. Trong thực tế, mảng ba chiều ít được sử dụng. Ví dụ 4.9 minh họa thao tác trên mảng 3 chiều bằng CT tạo sinh một mảng 3 chiều ngẫu nhiên và xuất ra màn hình.

Ví dụ 4.9: Một phần CT sinh và xuất ra một mảng 3 chiều ngẫu nhiên

```
#include …
#include "stdlib.h"   // Chứa các hàm: srand, rand
#include "time.h"     // Lấy thời gian hệ thống

typedef  int                mang_2chieu[10][10];
typedef  mang_2chieu   mang_3chieu[10];
// hoặc thay 2 dòng typedef trên đây bằng "typedef  int  mang_3chieu[10][10][10];"

void   sinh_mang3chieu_nnhien (mang_3chieu a, int &m, int &n, int &o);
void   xuat_mang3chieu (mang_3chieu a, int m, int n, int o);

int    main(int argc, char* argv[])
{
       // Các khai báo biến mảng 3 chiều a và kích thước của a ???

       // ✍ Khởi tạo bộ sinh số ngẫu nhiên (mỗi lần chạy CT mỗi khác): hạt giống của
       // bộ sinh số phụ thuộc vào thời gian của hệ thống.
       srand( time( 0 ) );

       sinh_mang3chieu_nnhien (a, m, n, o);  xuat_mang3chieu (a, m, n, o);

       getch (); return 0;
```

```
}
void   sinh_mang3chieu_nnhien (mang_3chieu a, int &m, int &n, int &o)
{
        m = rand() % 10;  n = rand() % 10;  o = rand() % 10;
        for (i = 0 ; i < m ; i++)
                for (j = 0 ; j < n ; j++)
                        for (k = 0 ; k < o ; k++)
                                x[i][j][k] = rand ()%100;
}
void   xuat_mang3chieu (mang_3chieu a, int m, int n, int o)
{
        for (i = 0 ; i < m ; i++) {
                printf ("\n\n[%d]", i);
                for (j = 0 ; j < n ; j++) {
                        printf("\n");
                        for (k = 0 ; k < o ; k++)   printf ("%3d", x[i][j][k]);
                }
        }
}
```

4.4. Kiểu struct

Struct là một kiểu dữ liệu gồm nhiều thành phần, các thành phần có thể cùng hoặc khác kiểu (kiểu thành phần có thể là đơn giản hoặc có cấu trúc). Để định nghĩa một kiểu struct, ta viết:

```
struct        ten_kieu_struct  {
        kieu_tphan_1              ten_tphan_1;
        kieu_tphan_2              ten_tphan_2;
        …
        kieu_tphan_N              ten_tphan_N;
};
```

Hình 4.9 chỉ ra định nghĩa (và minh họa) một số kiểu dữ liệu có kiểu struct.

✍ Khai báo một biến thuộc kiểu struct:

```
ten_kieu_struct     ten_bien;
```

✍ Để truy cập đến một thành phần của struct, ta sử dụng tên biến struct và tên thành phần cần truy cập. Cú pháp:

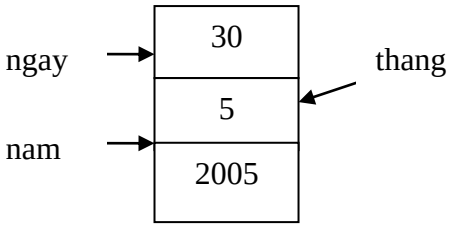
```
ten_bien.ten_thanh_phan
```

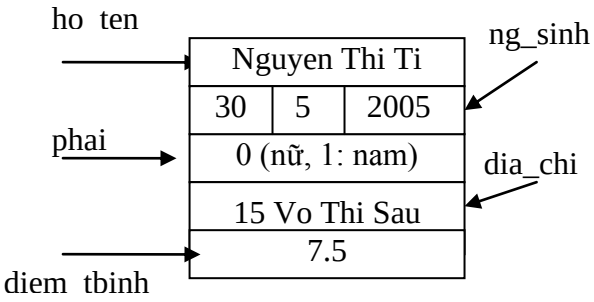
Thành phần của một struct là một biến thuộc kiểu tương ứng.

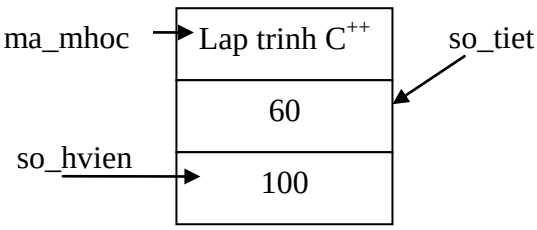
✋ C^{++} không trang bị định dạng để nhập xuất giá trị của một biến có kiểu struct. LTV phải viết từng câu lệnh nhập, xuất mỗi thành phần của biến struct.

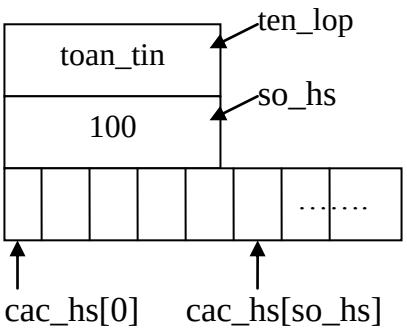
Sử dụng kiểu struct cho phép đóng gói dữ liệu nhằm giúp cho LTV dễ quản lý các đối tượng dữ liệu trong CT. Chẳng hạn:

Kiểu	Định nghĩa	Minh họa

▪ Ngày tháng năm	struct n_t_nam
	{
	int ngay;
	int thang;
	int nam;
	};
		

▪ Học sinh	struct hoc_sinh
	{
	chuoi ho_ten;
	n_t_nam ng_sinh;
	int phai;
	chuoi dia_chi;
	float diem_tbinh;
	};
		

▪ Môn học	struct mon_hoc
	{
	chuoi ma_mhoc;
	int so_tiet;
	int so_hvien;
	};
		

▪ Lớp học	typedef hoc_sinh mang_hs[100];
	struct lop_hoc
	{
	chuoi ten_lop;
	int so_hs;
	mang_hs cac_hs;
	};
		

Hình 4.9 Định nghĩa và minh họa một số kiểu dữ liệu struct

(a) để quản lý dữ liệu của một đa thức p: thay vì cần một mảng lưu các hệ số (he_so_p) và một biến lưu bậc (bac_p), ta sẽ đóng gói các hệ số và bậc vào một biến có kiểu struct (biến đa thức p).

(b) để quản lý phân số a: thay vì cần hai biến lưu tử số (tu_a) và mẫu số (mau_a), ta sẽ đóng gói tử số và mẫu số vào một biến có kiểu struct (biến phân số a).

Ví dụ 4.10: Các thao tác trên đa thức: nhập, xuất, đạo hàm cấp 1, định trị. *Hãy so sánh CT này với CT trong 4.1.3.3*.

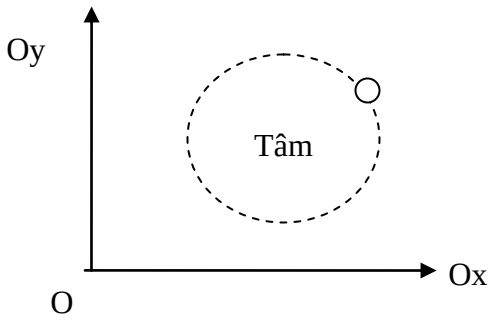
```
#include …
#define   KICH_THUOC   100
```

```
struct    da_thuc {
        float   he_so [KICH_THUOC];  // Các hệ số của đa thức
        int     bac;                 // Bậc của đa thức
};
void    nhap_da_thuc (da_thuc &p);
void    xuat_da_thuc (da_thuc p);
void    dao_ham_cap_1 (da_thuc &p);
float   dinh_tri (da_thuc p, float x);
int     main(int argc, char* argv[])
{
        da_thuc p;     float x, px;
        nhap_da_thuc (p);    xuat_da_thuc (p);
        scanf ("%f", &x);     px = dinh_tri (p, x);
        printf ("\nDinh tri cua da thuc tren x = %f la: %f", x, px);
        dao_ham_cap_1 (p); xuat_da_thuc (p);
        getch (); return 0;
}
void    nhap_da_thuc (da_thuc &p)                  // nhập đa thức
{
        printf ("Cap = "); scanf ("%d", &p.bac);
        for (j=0; j<=p.bac; j = j + 1) {
                // Nhập hệ số thứ j của đa thức p p.he_so[j] ???
        }
}
void    xuat_da_thuc (da_thuc p)             // xuất đa thức
{
        for (j=0; j<=p.bac; j = j + 1)
                printf ("%f * x^%d + ", p.he_so[j], j);
}
void    dao_ham_cap_1 (da_thuc &p)       // đạo hàm cấp 1
{
        p.bac --;
        for (j=0; j<=p.bac; j = j + 1)
                p.he_so[j] = p.he_so[j+1] * (j + 1);
}
float   x_mu_k (float x, int k)                // tính x mũ k
{
        float    kq = 1.0;
        // ???
        return kq;
}
float   dinh_tri (da_thuc p, float x)          // định trị đa thức
{
        float    px = 0.0;
        for (j=0; j<=p.bac; j = j + 1)
```

```
            px += p.he_so[j] * x_mu_k (x, j);
        return px;
}
```

Ví dụ 4.11: Minh họa ứng dụng kiểu struct quản lý hình tròn qua CT nhập và tính diện tích hình tròn.

Một hình tròn được xác định bởi tâm và một điểm trên cung tròn:




```
#include …

#define     PI      3.1416
struct  diem {              // Định nghĩa kiểu điểm
        float   x;
        float   y;
};
struct  h_tron      {       // Định nghĩa kiểu hình tròn
        diem   tam;
        diem   diem_tren_cung;
};

void    nhap_diem     (diem&);
void    xuat_diem     (diem);
void    nhap_htron    (h_tron&);
void    xuat_htron    (h_tron);
float   tinh_khoang_cach  (diem, diem);
float   tinh_dtich_htron (h_tron);

int     main(int argc, char* argv[])
{
        h_tron    c;

        nhap_htron (c);  xuat_htron  (c);
        printf ("\n Dien tich hinh tron là %d.", tinh_dtich_htron (c));

        getch(); return 0;
}
```

Tiêu đề hàm có thể không chứa kiểu dữ liệu của các đối số.

```
void   nhap_diem (diem &p)                    // nhập toạ độ điểm
{
       scanf ("%f", &p.x);
       scanf ("%f", &p.y);
}
void   xuat_diem (diem p)                     // xuất tọa độ điểm
{
       printf ("(%6.2f, %6.2f)", p.x, p.y);
}
void   nhap_htron     (h_tron &a)             // nhập hình tròn
{
       // Nhập tâm và điểm trên cung của a
}
void   xuat_htron     (h_tron a)              // xuất hình tròn
{
       // Xuất tâm và điểm trên cung của a
}
float  tinh_khoang_cach  (diem p1, diem p2)   // tính khoảng cách giữa hai điểm
{
       dx = p1.x – p2.x;     dy = p1.y – p2.y;
       return   sqrt (pow(dx , 2.0) + pow(dy , 2.0));
}
float  tinh_dtich_htron (h_tron  a)           // tính diện tích hình tròn
{
       r = tinh_khoang_cach (a.tam, a.diem_tren_cung);
       return  (PI * pow (r, 2.0));
}
```

Ví dụ 4.12: CT quản lý danh sách những người có điện thoại, biết thông tin của mỗi người gồm: họ (chuỗi), tên (chuỗi) và số điện thoại

```
#include …
#define  MAX_DANH_SACH  100
struct  MotNguoi
{
       char  Ho[31];
       char  Ten[9];
       char  SoDienThoai[8];
};
typedef  MotNguoi  DanhSachNguoi[MAX_DANH_SACH];
void  NhapMotNguoi (MotNguoi &x);
void  NhapDanhSach (DanhSachNguoi   ds,  int  &n);
void   XuatMotNguoi (MotNguoi  x);
void  XuatDanhSach (DanhSachNguoi   ds,  int  n);
int main(int argc, char* argv[])
{
```

```
        DanhSachNguoi     DanhSach;
        int               SoNguoi;
        NhapDanhSach (DanhSach,  SoNguoi);
        XuatDanhSach (DanhSach,  SoNguoi);
        getch (); return 0;
}
void  NhapMotNguoi (MotNguoi &x)     // nhập thông tin một người
{
        fflush(stdin);                // Làm sạch bộ đệm bàn phím !!!
        printf ("\n Ho              : "); gets (x.Ho);
        printf ("\n Ten             : ");  gets (x.Ten);
        printf ("\n So dien thoai : ");  gets (x.SoDienThoai);
}
void  NhapDanhSach (DanhSachNguoi  ds,  int   &n)   // nhập danh sách người
{
        printf ("\n So nguoi la bao nhieu : ");  scanf ("%d", &n);
        for (int  i = 0 ; i < n ; i++)
                NhapMotNguoi(ds[i]);
}
void   XuatMotNguoi (MotNguoi  x)       // xuất thông tin một người
{
        printf ("%s", x.Ho);
        printf ("%s", x.Ten);
        printf ("%s", x.SoDienThoai);
}
void  XuatDanhSach (DanhSachNguoi  ds,  int  n)       // xuất danh sách người
{
        for (int  i = 0 ; i < n ; i++) {
                XuatMotNguoi (ds[i]);
                printf ("\n");
        }
}
```

✍ Có thể vừa khai báo kiểu vừa gán giá trị ban đầu cho một biến struct. Chẳng hạn, khai báo hai biến số phức a, b nhận giá trị ban đầu lần lượt là 2+3i và 5-4i như sau:

```
struct so_phuc {
        float   thuc;
        float   ao;
};
so_phuc   a = {2.0, 3.0}, b = {5.0, -4.0};
```

✍ Hai biến thuộc kiểu struct có thể được gán cho nhau bằng câu lệnh gán. Phép gán hai biến kiểu struct thực ra là tập hợp các phép gán từng thành phần.

Ví dụ 4.13: Một phần CT tính tổng $1+\frac{1}{2}+\ldots+\frac{1}{n}$ (so sánh với 3.3.11).

```
#include …
struct   phan_so
{
        int tu_so;
        int mau_so;
};
void    nhap_phan_so (phan_so &ps);
void    xuat_phan_so (phan_so ps);
void    tong_phan_so (phan_so ps1, phan_so ps2, phan_so &ps3);
void    toi_gian_phan_so (phan_so &ps1);
int     tinh_uscln (int x, int y);
int     main(int argc, char* argv[])
{
        phan_so  s = {1, 1};          // Khởi gán giá trị ban đầu cho s

        // Nhập n ???
        for (i=2; i<=n; i++) {
                phan_so  ps_i = {1, i};
                tong_phan_so (s, ps_i, c);
                toi_gian_phan_so (c);
                s = c;                  // Gán nội dung biến c cho s
        }
        xuat_phan_so(s);
        getch(); return 0;
}
void    xuat_phan_so (phan_so ps)                   // xuất phân số
{
        printf ("%d / %d", ps.tu_so, ps.mau_so);
}
void    tong_phan_so (phan_so ps1, phan_so ps2, phan_so &ps3)      // tổng hai phân số
{
        ps3.tu_so       = ps1.tu_so * ps2.mau_so + ps1.mau_so * ps2.tu_so;
        ps3.mau _so  = ps1.mau_so * ps2.mau_so;
}
void    toi_gian_phan_so (phan_so &ps)          // tối giản phân số
{
        int    uscln  = tinh_uscln (ps.tu_so, ps.mau_so);
        // ???
}
int     tinh_uscln (int x, int y)               // tính ước số chung lớn nhất của x, y
{
        int     usc;
        // ???
        return usc;
}
```

Hãy bổ sung vào một phần CT trên các thao tác: nhập, -, *, /. Hãy viết CT thao tác trên số phức. *Bài tập*.

Ví dụ 4.14: CT hiện ra bảng chọn lựa:

1) Nhập danh sách học sinh (mã học sinh: chuỗi ký tự, có gia đình: ký tự C/K, điểm trung bình: float, tuổi: integer).
2) Xuất danh sách học sinh.
3) Sắp xếp tăng danh sách học sinh theo điểm trung bình.
4) Tìm kiếm thông tin học sinh theo mã học sinh.
5) Liệt kê các học sinh chưa có gia đình.

```
#include …
#define     CHAO            "Xin chao cac ban"
#define     TAM_BIET        "Xin tam biet cac ban"
#define     TOI_DA          50
typedef     char    chuoi[100];
struct      HocSinh {
        chuoi       MaHS;
        char        CoGDinh;
        float       DTBinh;
        int         Tuoi;
};
typedef     HocSinh    DayHocSinh [TOI_DA];
struct  LopHoc
{
        DayHocSinh      CacHocSinh;
        int             SiSo;
};
int     MenuVaChonLua ();
HocSinh     Nhap1HSinh ();
void        Xuat1HSinh (HocSinh hs);
LopHoc      NhapDSachLop ();
void        XuatDSachLop (LopHoc lop);
void        HoanVi (HocSinh &hs1, HocSinh &hs2);
void        SapXepTangTheoDTB(LopHoc &lop);
int         TkiemHSinhTheoMaSo (LopHoc lop, chuoi ma_tim, int &vt);
void        LietKeChuaCoGDinh (LopHoc lop);
int main(int argc, char* argv[])
{
        LopHoc lop10A;
        int     vi_tri, so_chon, ngung;
        chuoi   ma_can_tim;
        ngung = 0; printf ("%s", CHAO);
        while (! ngung)
        {
                so_chon = MenuVaChonLua ();
```

```
                if ((so_chon < 1) || (so_chon > 5) )
                {
                        ngung = 1;     continue;
                }
                switch (so_chon)
                {
                case 1:   lop10A = NhapDSachLop (); break;
                case 2:   XuatDSachLop (lop10A);  break;
                case 3:   SapXepTangTheoDTB(lop10A); break;
                case 4:   printf (“Nhap ma hoc sinh can tim thong tin = ”);
                          fflush (stdin); gets (ma_can_tim);
                          if (! TkiemHSinhTheoMaSo(lop10A, ma_can_tim, vi_tri))
                                printf (“Khong co hoc sinh do.”);
                          else
                                Xuat1HSinh (lop10A.CacHocSinh[vi_tri]);
                          break;
                case 5:   LietKeChuaCoGDinh (lop10A);
                }
                printf (“\nẤn phím bất kỳ để tiếp tuc…\n”); getch ();
        }
        printf (“%s”, TAM_BIET);
        getch (); return 0;
}
int     MenuVaChonLua ()
{
        int     kq_chon;
        printf (“Chọn một trong các mục sau:\n”);
        printf (“1) Nhập danh sách học sinh.\n”);
        printf (“2) Xuất danh sách học sinh.\n”);
        printf (“3) Sắp xếp tăng danh sách học sinh theo điểm trung bình.\n”);
        printf (“4) Tìm kiếm thông tin học sinh theo mã học sinh.\n”);
        printf (“5) Liệt kê các học sinh chưa có gia đình.\n”);
        printf (“?) Thoát.\n”);
        printf (“Chọn = ”);  scanf (“%d”, &kq_chon);
        return kq_chon;
}
HocSinh       Nhap1HSinh ()
{
        HocSinh hs;
        fflush (stdin);
        printf (“\n Mã số: ”); gets (hs.MaHS);
        printf (“\n Có gia đình (C/K): ”); scanf (“%c”, &hs.CoGDinh);
        printf (“\n Điểm trung bình : ”); scanf (“%f”, &hs.DTBinh);
        printf (“\n Tuổi: ”); scanf (“%d”, &hs.Tuoi);
        return hs;
```

```
}
void    Xuat1HSinh (HocSinh hs)
{
        printf ("\n Mã số: %s", hs.MaHS);
        printf ("\n Có gia đình (C/K): %c", hs.CoGDinh);
        printf ("\n Điểm trung bình : %f", hs.DTBinh);
        printf ("\n Tuổi: %d", hs.Tuoi);
}
LopHoc      NhapDSachLop ()
{
        LopHoc lop;
        int j;
        printf ("\n Sĩ số = "); scanf ("%d", &lop.SiSo);
        for (j=0; j<lop.SiSo; j++)
                lop.CacHocSinh[j] = Nhap1HSinh ();
        return lop;
}
void    XuatDSachLop (LopHoc lop)
{
        int j;
        printf ("\n Sĩ số = %d", lop.SiSo);
        for (j=0; j<lop.SiSo; j++)
        {
                printf ("\n Học sinh thứ %d:\n", j);
                Xuat1HSinh (lop.CacHocSinh[j]);
        }
}
void    HoanVi (HocSinh &hs1, HocSinh &hs2)
{
        HocSinh tam;
        tam = hs1;  hs1 = hs2; hs2 = tam;
}
void    SapXepTangTheoDTB(LopHoc &lop)
{
        int j, k;
        for (j=0; j<lop.SiSo-1; j = j + 1)
                for (k=j+1; k<lop.SiSo; k = k + 1)
                        if (lop.CacHocSinh[j].DTBinh > lop.CacHocSinh[k].DTBinh)
                                HoanVi (lop.CacHocSinh[j], lop.CacHocSinh[k]);
}
int     TkiemHSinhTheoMaSo (LopHoc lop, chuoi ma_tim, int &vt)
{
        int kq, k;
        kq = 0;
        for (k=0; k<lop.SiSo; k = k + 1)
```

```
                if (strcmp(lop.CacHocSinh[k].MaHS, ma_tim) == 0)
                {
                        kq = 1; vt = k;
                        break;
                }
        return kq;
}
void    LietKeChuaCoGDinh (LopHoc lop)
{
        int k;
        for (k=0; k<lop.SiSo; k = k + 1)
                if (lop.CacHocSinh[k].CoGDinh == 'C')
                        Xuat1HSinh (lop.CacHocSinh[k]);
}
```

BÀI TẬP

1) Tìm lỗi của hai đoạn CT sau:

a)

```
        typedef  int  mang_1_chieu[5][4];
        mang_1_chieu         Arr;
        int    i, j;
for (i = 0 ; i < 4 ; i++)
        for (int   j = 0 ; j < 5 ; j++)
              Arr[i][j]  =  i+j;
```

b)

```
        typedef  int  mang_1_chieu[2][3];
        mang_1_chieu         Arr;
        for (int   i = 0 ; i <= 2 ; i++)
                    for (int   j = 0 ; j <= 3 ; j++)
                  Arr[i][j]  =  0;
```

2) Viết các hàm:

a) xoa_chuoi_con (chuoi, vt, n) → xoá n ký tự trong chuỗi kể từ vi_tri vt.

Ví dụ: *ho_ten ← "123456789"*

delete (ho_ten, 2, 3) → ho_ten = "156789".

b) chen_chuoi_con (chuoi_them, chuoi_goc, vt) → chèn chuỗi chuoi_them vào chuoi_goc ở vị trí vt.

Ví dụ: *ho_ten ← "123456789"*

insert ("abc", ho_ten, 8) → ho_ten = "15678abc9".

c) kiem_tra_doi_xung (s) → kiểm tra chuỗi s có đối xứng hay không ?

d) tim_chuoi_con (s, s1) → tìm vị trí đầu tiên xuất hiện chuỗi s1 trong s

Ví dụ : S1="AAB", S ="12AAB67AAB" sẽ in ra 3

e) so_thanh_chuoi (so, chuoi) → biến số thành chuỗi

Ví dụ: STR(1234:5,s) → s = "1234"

STR(2.5,s) → "2.500"

f) chuoi_thanh_so (chuoi, so) → biến chuỗi thành số

g) dao_nguoc_chuoi(chuoi) → đảo ngược một chuỗi ký tự

3) Viết các CT:

1) Nhập mảng các số nguyên, xuất mảng, tích các phần tử trong mảng, trung bình cộng các phần tử, xuất ra các phần tử là: số lẻ, số nguyên tố, tìm vị trí của phần tử có giá trị nhỏ nhất, và xuất ra dãy được sắp xếp giảm.

2) Cài đặt các thao tác trên vector: +, tích vô hướng, lấy chuẩn, tính khoảng cách giữa hai vector.

3) Nhập 1 mảng gồm n phần tử. Viết CT kiểm tra xem một số x có trong mảng trên không. Nếu có hãy cho biết có bao nhiêu số x như vậy, và cho biết các số thứ tự tương ứng của nó trong mảng .

4) Nhập 1 mảng có n phần tử. Hãy cho biết trong mảng có bao nhiêu phần tử < x, = x, và > x.

5) Có 1 mảng gồm n phần tử. Kiểm tra xem mảng trên có được sắp theo thứ tự giảm không.

6) Có 1 mảng gồm n phần tử. Biết mảng có m($0 \le m \le n$) phần tử đầu tiên chứa các số theo thứ tự tăng dần. Nhập vào 1 số x, và tìm cách chèn x vào đúng vị trí trong mảng.

7) Cho mảng gồm n phần tử. Hãy cho biết giá trị lớn nhất và giá trị nhỏ nhất trên mảng và vị trí tương ứng của chúng.

8) Cho 1 mảng gồm n phần tử đã được sắp theo thứ tự tăng. Hãy tìm xem có phần tử x trong mảng không và tìm theo phương pháp dò tìm nhị phân.

9) Cho mảng gồm n phần tử. Hãy tìm phần tử lặp lại nhiều lần nhất, số lần lặp lại của nó, trong 2 trường hợp :

a) Mảng chưa có thứ tự.

b) Mảng đã được sắp thứ tự không giảm.

10) Viết CT thực hiện các thao tác trên đa thức: tính đạo hàm cấp k, tính tích 2 đa thức.

11) Viết chương trình nhập một dãy các phần tử là các số nguyên. In ra các giá trị khác nhau của dãy đó.

(Ví dụ : dãy : 2 3 2 10 6 5 3 , in ra 2 3 10 6 5)

12) Viết chương trình nhập một dãy các phần tử là các số thực. In ra các giá trị khác nhau theo giá trị tăng dần.

13)Cho dãy số $X = \{x_1, x_2, \ldots, x_n\}$ và số C

a. Đếm số lần xuất hiện của số C trong X.

b. Tính tổng các số chia hết cho C trong X.

c. Đếm số đường chạy trong X. Biết rằng x_i và x_{i+1} thuộc cùng một đường chạy nếu $x_i <= x_{i+1}$. Ví dụ: X = {1, 2, 5, -3, -4, 5, 7, 9} có 3 đường chạy: {1, 2, 5}, {-3}, {-4, 5, 7, 9}.

d. Kiểm tra tính đối xứng của X. Ví dụ: X = {1, 2, 3, 2, 1} có tính đối xứng.

e. Tìm số chẵn nhỏ nhất trong X chia hết cho 4 (nếu có).

f. Tính tổng các phần tử của X.

g. Tính trung bình cộng các số dương chẵn và tổng các số âm lẻ trong X.

14)Xuất ra phần giao, hiệu của hai dãy số X = {x_1, x_2, …, x_n} và Y = {y_1, y_2, …, y_m}. Ví dụ: X = {1, 2, 3, 5}, Y = {-4, 1, 5, 7} có phần giao là {1, 5}, có phần hiệu là {2, 3}. Kiểm tra xem X có chứa Y không ?

15) Vẽ các biểu đồ sau:

a) Ngang : ví dụ dãy là : 3 5 8 2 thì vẽ :

```
* * *
* * * * *
* * * * * * * *
* *
```

b) Đứng : ví dụ dãy là : 3 5 8 2 thì vẽ :

```
  *
  *
  *
 * *
 * *
* * *
* * * *
* * * *
```

16) Viết chương trình nắn tên: cắt bỏ các khoảng trắng chỉ giữ lại các khoảng trắng ngăn cách họ, chữ lót, tên; ký tự đầu của họ, chữ lót, tên được viết hoa; các ký tự còn lại viết thường. Ví dụ: *" nguYen Van BA "* → "Nguyen Van Ba".

17)Viết chương trình cài đặt chuỗi ký tự theo một trong hai cách (giả sử kiểu chuỗi chưa có sẵn trong ngôn ngữ lập trình bạn đang dùng):

a. phần tử đầu chỉ số ký tự của chuỗi;

b. chuỗi được kết thúc bởi ký tự có mã ASCII bằng 0.

Sau đó viết lại các thao tác cơ bản trên chuỗi (tính chiều dài chuỗi, nối, sao chép một phần của chuỗi, chặt ngắn chuỗi, kiểm tra chuỗi con, ...).

18) Viết chương trình thực hiện các thao tác trên ma trận: nhập ma trận, xuất ma trận, tìm các phần tử của ma trận lớn hơn một số Z cho trước, tính tổng, tích các phần tử của ma trận, tính định thức, chuyển vị ma trận, tính ma trận nghịch đảo.

19) Viết chương trình kiểm tra một ma trận có phải là ma trận chéo, tam giác (trên, dưới), đơn vị, đối xứng.

20)Nhập vào thông tin hai học sinh. Xuất ra thông tin về học sinh có điểm trung bình lớn hơn.

21)Nhập vào thông tin hai môn học. Xuất ra thông tin về môn học có số lượng người học nhiều hơn.

22)Nhập vào thông tin ba học sinh. Xuất ra thông tin về học sinh có điểm trung bình lớn nhất.

23)Thực hiện các phép toán +, -, *, lấy số liên hiệp của các số phức.

24)Nhập vào dãy một chiều gồm N học sinh. Xuất ra màn hình:

0. thông tin học sinh có điểm trung bình cao nhất.
1. tìm xem có học sinh: điểm trung bình 3.0, tuổi 14, mã số 1234 không ?
2. dãy N học sinh được sắp xếp tăng theo mã số.

25)Quản lý N đội bóng của một giải bóng đá. Thông tin về một đội bóng bao gồm:

i. tên đội bóng có kiểu chuỗi.
ii. số trận thắng, số trận thua, số trận hòa: cùng có kiểu số nguyên.
iii. điểm số: có kiểu số thực.

Hãy:

1. Nhập danh sách các đội bóng gồm: tên đội bóng, số trận thắng, số trận thua, số trận hòa.
2. Tính điểm (thắng: 3, hòa: 1, thua: 0) cho từng đội.
3. Sắp thứ tự các đội theo điểm số.
4. Xuất thông tin về đội đứng đầu.
5. Tìm thông tin về đội chưa thắng trận nào.

26) Viết chương trình nhập hồ sơ các học sinh của lớp học gồm: tên, tuổi, điểm trung bình. In ra tuổi lớn nhất, tuổi nhỏ nhất của lớp, học sinh nào có điểm trung bình lớn nhất, học sinh nào có điển trung bình nhỏ nhất.

27) Viết chương trình nhập danh sách của một thư viện. Hồ sơ về một quyển sách gồm: tên sách, tên tác giả, năm xuất bản, tên người mượn. Sau đó in ra các sách của thư viện với các thông tin như trên và in ra có bao nhiêu cuốn sách đã cho mượn.

28) Viết chương trình tính điểm của một lớp:

- Nhập các thông tin sau cho mỗi sinh viên: tên, năm sinh, điểm trung bình HK 1, HK 2.
- In ra danh sách các học sinh của lớp theo thứ tự giảm dần của điểm trung bình toàn năm (TBtoàn năm = (TB kỳ1 +TBkỳ)/2) và cho biết hạng của học sinh.

29) Viết chương trình nhập hồ sơ của một lớp. Hồ sơ mỗi học sinh gồm: tên, năm sinh, điểm trung bình. In ra danh sách lớp theo thứ tự giảm dần của điểm.

30) Viết chương trình tính điểm cho học kỳ của một lớp:

- Đối với mỗi học sinh nhập: tên, điểm văn, toán, lý, hóa, ngoại ngữ.
- In ra danh sách của lớp theo thứ tự giảm dần của điểm trung bình và cho biết hạng của mỗi học sinh. Biết rằng văn và toán có hệ số 3, lý, hóa, ngoại ngữ có hệ số 2.

31) Nhập hồ sơ của một tập thể người. Hồ sơ gồm 2 thành phần là: tên, ngày sinh. Ngày là một kiểu record gồm 3 thành phần: ngày, tháng, năm. Sau khi nhập xong in ra danh sách theo thứ tự giảm dần của tuổi.

32) Viết chương trình hiện ra menu chọn lựa:

1. Nhập vào dãy N số thực.
2. xuất ra các số thực < 5.0.
3. sắp xếp tăng dần dãy số.
4. xuất ra dãy số.
5. nhập vào một số thực X, kiểm tra xem có số thực X trong dãy hay không? nếu có hãy chỉ ra vị trí của X.
6. tính và xuất ra trung bình cộng của dãy.
7. tính và xuất ra trung bình nhân của dãy.

Yêu cầu:

- thủ tục: nhập dãy, xuất dãy, xuất các số < 5.0.
- hàm: kiểm tra X có trong dãy hay không, tính trung bình cộng, nhân.