# Chương I
# Một số khái niệm về lập trình và ngôn ngữ lập trình

- Khái niệm cơ sở về lập trình;
- Khái niệm và các thành phần của ngôn ngữ lập trình;
- Vai trò và phân loại chương trình dịch.

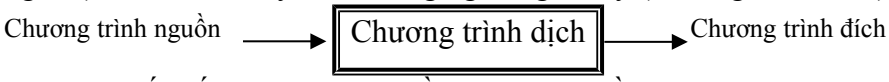
# §1. KHÁI NIỆM LẬP TRÌNH VÀ NGÔN NGỮ LẬP TRÌNH

Như đã biết, mọi bài toán có thuật toán đều có thể giải được trên máy tính điện tử. Khi giải bài toán trên máy tính điện tử, sau các bước xác định bài toán và xây dựng hoặc lựa chọn thuật toán khả thi là bước lập trình.

Lập trình là sử dụng cấu trúc dữ liệu và các câu lệnh của ngôn ngữ lập trình cụ thể để mô tả dữ liệu và diễn đạt các thao tác của thuật toán. Chương trình viết bằng ngôn ngữ lập trình bậc cao nói chung không phụ thuộc vào máy, nghĩa là một chương trình có thể thực hiện trên nhiều máy. Chương trình viết bằng ngôn ngữ máy có thể được nạp trực tiếp vào bộ nhớ và thực hiện ngay còn chương trình viết bằng ngôn ngữ lập trình bậc cao phải được chuyển đổi thành chương trình trên ngôn ngữ máy mới có thể thực hiện được.

Chương trình đặc biệt có chức năng chuyển đổi chương trình được viết bằng ngôn ngữ lập trình bậc cao thành chương trình thực hiện được trên máy tính cụ thể được gọi là *chương trình dịch*.

Chương trình dịch nhận đầu vào là chương trình viết bằng ngôn ngữ lập trình bậc cao (chương trình nguồn) thực hiện chuyển đổi sang ngôn ngữ máy (chương trình đích).




Xét ví dụ, bạn chỉ biết tiếng Việt nhưng cần giới thiệu về trường của mình cho đoàn khách đến từ nước Mĩ, chỉ biết tiếng Anh. Có hai cách để bạn thực hiện điều này.

*Cách thứ nhất:* Bạn nói bằng tiếng Việt và người phiên dịch giúp bạn dịch sang tiếng Anh. Sau mỗi câu hoặc một vài câu giới thiệu trọn một ý, người phiên dịch dịch sang tiếng Anh cho đoàn khách. Sau đó, bạn lại giới thiệu tiếp và người phiên dịch lại dịch tiếp. Việc giới thiệu của bạn và việc dịch của người phiên dịch luân phiên cho đến khi bạn kết thúc nội dung giới thiệu của mình. Cách dịch trực tiếp như vậy được gọi là *thông dịch*.

*Cách thứ hai:* Bạn soạn nội dung giới thiệu của mình ra giấy, người phiên dịch dịch toàn bộ nội dung đó sang tiếng Anh rồi đọc hoặc trao văn bản đã dịch cho đoàn khách đọc. Như vậy, việc dịch được thực hiện sau khi nội dung giới thiệu đã hoàn tất. Hai công việc đó được thực hiện trong hai khoảng thời gian độc lập, tách biệt nhau. Cách dịch như vậy được gọi là *biên dịch*.

Sau khi kết thúc, với cách thứ nhất không có một văn bản nào để lưu trữ, còn với cách thứ hai có hai bản giới thiệu bằng tiếng Việt và bằng tiếng Anh có thể lưu trữ để dùng lại về sau.

Tương tự như vậy, chương trình dịch có hai loại là *thông dịch* và *biên dịch*.

***a) Thông dịch***

Thông dịch (interpreter) được thực hiện bằng cách lặp lại dãy các bước sau:

① Kiểm tra tính đúng đắn của câu lệnh tiếp theo trong chương trình nguồn;
② Chuyển đổi câu lệnh đó thành một hay nhiều câu lệnh tương ứng trong ngôn ngữ máy;
③ Thực hiện các câu lệnh vừa chuyển đổi được.

Như vậy, quá trình dịch và thực hiện các câu lệnh là luân phiên. Các chương trình thông dịch lần lượt dịch và thực hiện từng câu lệnh. Loại chương trình dịch này đặc biệt thích hợp cho môi trường đối thoại giữa người và hệ thống. Tuy nhiên, một câu lệnh nào đó phải thực hiện bao nhiêu lần thì nó phải được dịch bấy nhiêu lần.

Các ngôn ngữ khai thác hệ quản trị cơ sở dữ liệu, ngôn ngữ đối thoại với hệ điều hành,... đều sử dụng trình thông dịch.

***b) Biên dịch***

Biên dịch (compiler) được thực hiện qua hai bước:

① Duyệt, kiểm tra, phát hiện lỗi, kiểm tra tính đúng đắn của các câu lệnh trong chương trình nguồn;
② Dịch toàn bộ chương trình nguồn thành một chương trình đích có thể thực hiện trên máy và có thể lưu trữ để sử dụng lại khi cần thiết.

Như vậy, trong thông dịch, không có chương trình đích để lưu trữ, trong biên dịch cả chương trình nguồn và chương trình đích có thể lưu trữ lại để sử dụng về sau.

Thông thường, cùng với chương trình dịch còn có một số dịch vụ liên quan như biên soạn, lưu trữ, tìm kiếm, cho biết các kết quả trung gian,... Toàn bộ các dịch vụ trên tạo thành một môi trường làm việc trên một ngôn ngữ lập trình cụ thể. Ví dụ, Turbo Pascal 7.0, Free Pascal 1.2, Visual Pascal 2.1,... trên ngôn ngữ Pascal, Turbo C++, Visual C++,... trên ngôn ngữ C++.

Các môi trường lập trình khác nhau ở những dịch vụ mà nó cung cấp, đặc biệt là các dịch vụ nâng cấp, tăng cường các khả năng mới cho ngôn ngữ lập trình.

# Em có biết

## AI LÀ LẬP TRÌNH VIÊN ĐẦU TIÊN?

Đó là một phụ nữ, bà Ada Augusta Byron Lovelace, con gái của nhà thơ nổi tiếng thời đó Lord Byron. Ada là một trong những nhân vật ấn tượng nhất trong lịch sử Tin học. Bà sinh ngày 10/12/1815 và là người cùng thời với Charles Babbage, người đầu tiên đưa ra đề án thiết kế chiếc máy tính điều khiển theo chương trình có tên là Analytical Engine (máy giải tích).

Từ nhỏ, bà đã nổi tiếng là một người thông minh, có khả năng đặc biệt về toán học.

Ngay từ khi thiết kế máy giải tích còn ở trên giấy, Ada đã đề xuất với Babbage một kế hoạch chi tiết để máy giải tích tính các số Bernouilli. Ngày nay người ta coi kế hoạch này là *chương trình máy tính đầu tiên* và bà được gọi là *lập trình viên đầu tiên*.

Các ghi chép được công bố của Ada cho tới nay vẫn đặc biệt có ý nghĩa đối với các lập trình viên. Giáo sư J. Von Neumann đã viết rằng các quan sát của Ada "chứng tỏ bà đã hiểu được các nguyên tắc lập trình máy tính trước thời đại của mình hàng thế kỉ".

Như một nhà toán học, Ada đánh giá cao khả năng tự động hoá các công việc tính toán nặng nhọc. Nhưng bà quan tâm hơn đến ***các nguyên tắc của việc lập trình*** các thiết bị đó. Ngay khi máy giải tích còn chưa được xây dựng, Ada đã thí nghiệm viết những dãy lệnh. Bà nhận ra giá trị của một vài thủ thuật đặc biệt trong nghệ thuật mới này và điều thú vị là những thủ thuật này hiện giờ vẫn còn là cơ bản đối với các ngôn ngữ lập trình hiện đại, đó chính là ***chương trình con, vòng lặp và các phép chuyển điều khiển***.

Thay cho việc viết các dãy lệnh lặp đi lặp lại nhiều lần, ta có thể viết chúng dưới dạng các ***chương trình con*** để dùng nhiều lần. Các chương trình con ngày nay là thành phần không thể thiếu được của mọi ngôn ngữ lập trình.

Máy giải tích và các máy tính số thực hiện rất tốt các tính toán nhiều lần một cách nhanh chóng. Thời kì đó, các bìa đục lỗ được sử dụng để đưa dữ liệu và các lệnh vào máy. Bằng việc phát minh ra các lệnh thực hiện việc chuyển thiết bị đọc bìa về một bìa xác định trước nó, sao cho dãy các lệnh có thể được thực hiện một số lần nhất định, Ada đã phát minh ra ***vòng lặp*** – một trong những cấu trúc điều khiển quan trọng trong các ngôn ngữ lập trình.

Khả năng lôgic của Ada đã phát huy với ***phép chuyển điều khiển có điều kiện***. Bà nghĩ ra một loại lệnh để thao tác với thiết bị đọc bìa, nhưng thay cho việc quay lại và lặp lại dãy bìa, lệnh này cho phép thiết bị đọc bìa chuyển tới một bìa khác tại bất kì vị trí nào trong dãy, NẾU một điều kiện nào đó được thoả mãn. Việc thêm chữ NẾU đó vào danh sách các lệnh số học thuần tuý trước đây có nghĩa là chương trình có thể làm nhiều hơn là tính toán đơn thuần. Ở dạng thô sơ nhưng về tiềm năng là rất có ý nghĩa, máy giải tích có thể thực hiện các *quyết định*.

Ada mất năm 1852, khi mới qua tuổi 36. Nếu như bà không qua đời sớm như vậy, chắc chắn khoa học lập trình của thế kỉ XIX đã có thể tiến nhanh hơn nhiều.

Để tưởng nhớ công lao của Ada, một ngôn ngữ lập trình do Bộ Quốc phòng Mĩ tạo ra năm 1979 mang tên bà.

**Ada** Augusta Byron Lovelace

## §2. CÁC THÀNH PHẦN CỦA NGÔN NGỮ LẬP TRÌNH

### 1. Các thành phần cơ bản

Mỗi ngôn ngữ lập trình thường có ba thành phần cơ bản là *bảng chữ cái, cú pháp* và *ngữ nghĩa.*

***a) Bảng chữ cái*** là tập các kí tự được dùng để viết chương trình. Không được phép dùng bất kì kí tự nào ngoài các kí tự quy định trong bảng chữ cái.

Trong C++, bảng chữ cái bao gồm các kí tự:

- Các chữ cái thường và các chữ cái in hoa của bảng chữ cái tiếng Anh:
  a b c d e f g h i j k l m n o p q r s t u v w x y z
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- 10 chữ số thập phân Ả Rập: 0 1 2 3 4 5 6 7 8 9
- Các kí tự đặc biệt:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| + | - | * | / | = | | < | > | [ | ] | . | , (dấu phẩy) |
| ; | # | ^ | $ | @ | & | ( | ) | { | } | : | ' (dấu nháy) |
| dấu cách (mã ASCII 32) | | | | | | | | _ (dấu gạch dưới) | | | |

Bảng chữ cái của các ngôn ngữ lập trình nói chung không khác nhau nhiều. Ví dụ, bảng chữ cái của ngôn ngữ lập trình C++ chỉ khác Pascal là có sử dụng thêm các kí tự như dấu nháy kép ("), dấu sổ ngược (\), dấu chấm than (!).

***b) Cú pháp*** là bộ quy tắc để viết chương trình. Dựa vào chúng, người lập trình và chương trình dịch biết được tổ hợp nào của các kí tự trong bảng chữ cái là hợp lệ và tổ hợp nào là không hợp lệ. Nhờ đó, có thể mô tả chính xác thuật toán để máy thực hiện.

***c) Ngữ nghĩa*** xác định ý nghĩa thao tác cần phải thực hiện, ứng với tổ hợp kí tự dựa vào ngữ cảnh của nó.

***Ví dụ***

Phần lớn các ngôn ngữ lập trình đều sử dụng dấu cộng (+) để chỉ phép cộng. Xét các biểu thức:

A + B (1)

I + J (2)

Giả thiết *A*, *B* là các đại lượng nhận giá trị thực và *I*, *J* là các đại lượng nhận giá trị nguyên. Khi đó dấu "+" trong biểu thức (1) được hiểu là cộng hai số thực, dấu "+" trong biểu thức (2) được hiểu là cộng hai số nguyên. Như vậy, ngữ nghĩa dấu "+" trong hai ngữ cảnh khác nhau là khác nhau.

Tóm lại, cú pháp cho biết cách viết một chương trình hợp lệ, còn ngữ nghĩa xác định ý nghĩa của các tổ hợp kí tự trong chương trình.

Các lỗi cú pháp được chương trình dịch phát hiện và thông báo cho người lập trình biết. Chỉ có các chương trình không còn lỗi cú pháp mới có thể được dịch sang ngôn ngữ máy.

Các lỗi ngữ nghĩa khó phát hiện hơn. Phần lớn các lỗi ngữ nghĩa chỉ được phát hiện khi thực hiện chương trình trên dữ liệu cụ thể.

### 2. Một số khái niệm

***a) Tên***

Mọi đối tượng trong chương trình đều phải được đặt tên theo quy tắc của ngôn ngữ lập trình và từng chương trình dịch cụ thể.

Trong C++: tên là một dãy liên tiếp *không quá 127 kí tự bao gồm chữ số, chữ cái hoặc dấu gạch dưới và bắt đầu bằng chữ cái hoặc dấu gạch dưới*.

Trong chương trình dịch C++, tên có phân biệt chữ hoa, chữ thường.

***Quy tắc đặt tên lớp, biến, phương thức và hàm :***

- Các tên định kiểu, tên lớp bắt đầu với chữ hoa, viết liền các từ, ví dụ Node, EvenHandler,…
- Các tên biến và tên hàm, phương thức bắt đầu bằng chữ thường, các từ tiếp theo viết nối liền vào và bắt đầu bằng chữ hoa, ví dụ node, myVar, myMethod(), evenHandler, getName()… Các cụm từ viết tắt không dùng chữ hoa nếu còn từ khác ở sau nó, chẳng hạn thay vì đặt tên hàm exportHTMLSource() ta dùng tên exportHtmlSource().
- Các hàm và phương thức nên có tên chứa động từ cho biết mục đích cụ thể của nó, ví dụ với một phương thức kiểm tra xem chuỗi nhập vào có phải là chữ số hay không, tên gọi stringIsNumbers(char* testString) sẽ dễ hiểu hơn tên gọi có vẻ mơ hồ là checkString(char* testString)
- Tránh sử dụng tên bắt đầu bằng một hay hai dấu gạch dưới, vì dễ dẫn tới xung đột với các biến theo tiêu chuẩn của hệ thống nào đó.
- Với các project lớn, cần thêm tiền tố xác định phạm vi của biến:
    - m : biến thành viên của một lớp.
    - g : biến toàn cục.
    - c : biến tĩnh của một lớp (bao gồm cả hằng số)
    - <empty> : biến cục bộ.

Nhiều ngôn ngữ lập trình, trong đó có C++, phân biệt ba loại tên:

- Tên dành riêng;
- Tên chuẩn;
- Tên do người lập trình đặt.

**Tên dành riêng**

Một số tên được ngôn ngữ lập trình quy định dùng với *ý nghĩa xác định*, người lập trình không được sử dụng với ý nghĩa khác. Những tên này được gọi là *tên dành riêng* (còn được gọi là *từ khoá*).

***Ví dụ***. Một số tên dành riêng:

Trong C++: main, include, if, while, void.

**Tên chuẩn**

Một số tên được ngôn ngữ lập trình dùng với *ý nghĩa nào đó*. Những tên này được gọi là *tên chuẩn*. Tuy nhiên, người lập trình có thể khai báo và dùng chúng với ý nghĩa và mục đích khác.

Ý nghĩa của các tên chuẩn được quy định trong các *thư viện* của ngôn ngữ lập trình.

***Ví dụ.*** Một số tên chuẩn

- Trong C++:

cin cout getchar

**Tên do người lập trình đặt**

Tên do người lập trình đặt được dùng với ý nghĩa riêng, xác định bằng cách khai báo trước khi sử dụng. Các tên này không được trùng với tên dành riêng.

***Ví dụ***

Tên do người lập trình đặt:

A1
DELTA
CT_Vidu

### *b) Hằng và biến*

**Hằng**

*Hằng là các đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.*

Trong các ngôn ngữ lập trình thường có cáchằng số học, hằng lôgic, hằng xâu.

- Hằng số học là các số nguyên hay số thực (dấu phẩy tĩnh hoặc dấu phẩy động), có dấu hoặc không dấu.

- Hằng lôgic là giá trị *đúng* hoặc *sai* tương ứng với *true* hoặc *false*.
- Hằng xâu là chuỗi kí tự trong bảng chữ cái. Khi viết, chuỗi kí tự này được đặt trong cặp dấu nháy (C++ dùng dấu nháy kép).
- Hằng kí tự là một kí tự riêng biệt được viết trong cặp dấu nháy đơn. Mỗi kí tự tương ứng một giá trị trong bảng mã ASSCII. Hằng kí tự cũng được xem như trị số nguyên

***Ví dụ***

- Hằng số học: 2 0 -5 +18
  1.5 -22.36 +3.14159 0.5
  -2.236E01 1.0E-6
- Hằng lôgic:
  + Trong C++: true false
- Hằng xâu:
  + Trong C++: "Informatic" "TIN HOC"

**Biến**

*Biến là đại lượng được đặt tên, dùng để lưu trữ giá trị và giá trị có thể được thay đổi trong quá trình thực hiện chương trình*.

Tuỳ theo cách lưu trữ và xử lí, C++ phân biệt nhiều loại biến. Các biến dùng trong chương trình đều phải khai báo. Việc khai báo biến sẽ được trình bày ở các phần sau.

***c) Chú thích***

Có thể đặt các đoạn chú thích trong chương trình nguồn. Các chú thích này giúp cho người đọc chương trình nhận biết ngữ nghĩa của chương trình đó dễ hơn. Chú thích không ảnh hưởng đến nội dung chương trình nguồn và được chương trình dịch bỏ qua.

Trong C++, chú thích được đặt giữa cặp dấu /* và */.

**TÓM TẮT**

- Cần có chương trình dịch để chuyển chương trình nguồn thành chương trình đích.
- Có hai loại chương trình dịch: thông dịch và biên dịch.
- Các thành phần của ngôn ngữ lập trình: bảng chữ cái, cú pháp và ngữ nghĩa.
- Mọi đối tượng trong chương trình đều phải được đặt tên:
  - Tên dành riêng: Được dùng với ý nghĩa riêng, không được dùng với ý nghĩa khác.
  - Tên chuẩn: Tên dùng với ý nghĩa nhất định, khi cần dùng với ý nghĩa khác thì phải khai báo.
  - Tên do người lập trình đặt: cần khai báo trước khi sử dụng.
- Hằng: Đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.
- Biến: Đại lượng được đặt tên. Giá trị của biến có thể thay đổi trong quá trình thực hiện chương trình.

## CÂU HỎI VÀ BÀI TẬP

**1.** Tại sao người ta phải xây dựng các ngôn ngữ lập trình bậc cao?

**2.** Chương trình dịch là gì? Tại sao cần phải có chương trình dịch?

**3.** Biên dịch và thông dịch khác nhau như thế nào?

**4.** Hãy cho biết các điểm khác nhau giữa tên dành riêng và tên chuẩn.

**5.** Hãy tự viết ra ba tên đúng theo quy tắc của C++ và có độ dài khác nhau.

**6.** Hãy cho biết những biểu diễn nào dưới đây không phải là biểu diễn hằng trong C++ và chỉ rõ lỗi trong từng trường hợp:

a) 150.0 b) -22 c) 6,23 d) "43"

e) A20 f) 1.06E-15 g) 4+6 h) "C
i) 'TRUE'

# Em có biết

## AI LÀ TÁC GIẢ CỦA NGÔN NGỮ C++?

**C++** là một loại ngôn ngữ lập trình. Đây là một dạng ngôn ngữ đa mẫu hình tự do có kiểu tĩnh và hỗ trợ lập trình thủ tục, dữ liệu trừu trượng, lập trình hướng đối tượng, và lập trình đa hình. Từ thập niên 1990, C++ đã trở thành một trong những ngôn ngữ thương mại phổ biến nhất trong khi đó.

Bjarne Stroustrup của Bell Labs đã phát triển C++ (mà tên nguyên thủy là "*C với các lớp*" trong suốt thập niên 1980 như là một bản nâng cao của ngôn ngữ C. Những bổ sung nâng cao bắt đầu với sự thêm vào của khái niệm lớp, tiếp theo đó là các khái niệm hàm ảo, chồng toán tử, đa kế thừa, tiêu bản, và xử lý ngoại lệ. Tiêu chuẩn của ngôn ngữ C++ đã được thông qua trong năm 1998 như là *ISO/IEC 14882:1998*. Phiên bản hiện đang lưu hành là phiên bản 2003,*ISO/IEC 14882:2003*. Hiện tại tiêu chuẩn mới nhất của ngôn ngữ C++ là C++11 (ISO/IEC 14882:2011).

Tổng quan về kĩ thuật:

Trong tiêu chuẩn 1998 của C++ có hai phần chính: phần ngôn ngữ cốt lõi và phần Thư viện chuẩn C++(STL - Standard Template Library). Phần thư viện này lại bao gồm hầu hết thư viện tiêu bản chuẩn và phiên bản có điều chỉnh chút ít của thư viện chuẩn C. Nhiều thư viện C++ hiện hữu không thuộc về tiêu chuẩn như là thư viện Boost. Thêm vào đó, nhiều thư viện không theo tiêu chuẩn được viết trong C một cách tổng quát đều có thể sử dụng trong các chương trình C++.

**Chức năng dẫn nhập trong C++**

So với C, C++ tăng cường thêm nhiều tính năng, bao gồm: khai báo như mệnh đề, chuyển kiểu giống như hàm,new/delete, bool, các kiểu tham chiếu, const, các hàm trong dòng (inline), các đối số mặc định, quá tải hàm, vùng tên (namespace), các lớp (bao gồm tất cả các chức năng liên quan tới lớp như kế thừa, hàm thành viên (phương pháp), hàm ảo, lớp trừu tượng, và cấu tử), sự quá tải toán tử, tiêu bản, toán tử ::, xử lí ngoại lệ, và sự nhận dạng kiểu trong thời gian thi hành.

C++ còn tiến hành nhiều phép kiểm tra kiểu hơn C trong nhiều trường hợp.

Câu lệnh chú giải bắt đầu với // nguyên là một phần của BCPL được tái sử dụng trong C++.

Một số thành phần của C++ sau này đã được thêm vào C, bao gồm const, inline, khai báo biến trong vòng lặpfor và chú giải kiểu C++ (sử dụng ký hiệu //). Tuy nhiên, C99 cũng bổ sung thêm một số tính năng không có trong C++, ví dụ như macro với số đối số động.

Vì được phát triển từ C, trong C++, thuật ngữ đối tượng có nghĩa là *vùng nhớ* như được dùng trong C, chứ không phải là một phiên bản của lớp như được hiểu trong phần lớn ngôn ngữ lập trình hướng đối tượng khác. Ví dụ như:

```
int i;
```

Dòng trên sẽ định nghĩa một đối tượng kiểu int (số nguyên), tức là một vùng nhớ sẽ được sử dụng để lưu giữ biến i.

**Thư viện C++**

Thư viện chuẩn C++ dùng lại thư viện chuẩn C với một số điều chỉnh nhỏ để giúp nó hoạt động tốt hơn với ngôn ngữ C++. Một bộ phận lớn khác của thư viện C++ dựa trên Thư viện tiêu bản chuẩn (hay còn gọi là STL - viết tắt từ chữ *Standard Template Library*). Thư viện này có nhiều công cụ hữu dụng như là các thùng chứa (thí dụ như vector, danh sách liên kết và biến lặp (tổng quát hóa từ khái niệm con trỏ) để cung cấp những thùng chứa này sự truy cập giống như là truy cập mãng. Xa hơn nữa, bảng (đa) ánh xạ (mảng kết hợp) và (đa) tập, tất cả được cung cấp để có thể xuất ra các giao diện tương thích. Do đó, có thể dùng tiêu bản để viết các thuật toán tổng quát mà chúng làm việc được với bất kì thùng chứa nào hay với bất kì dãy nào được định nghĩa bởi biến lặp. Giống như C, các tính năng của thư viện này thì được truy cập bởi việc sử dụng lệnh dẫn

hướng#include để bao gồm một tập tin tiêu đề chuẩn. C++ cung ứng 69 tiêu đề chuẩn, trong đó có 19 tiêu đề không còn hiệu lực nữa.

Vì thư viện chuẩn được thiết kế bởi những chuyện gia hàng đầu và đã được chứng minh trong toàn bộ lịch sử kĩ nghệ, các thành phần của thư viện này được khuyến cáo sử dụng thay vì dùng những phần viết tay bên ngoài hay những phương tiện cấp thấp khác. Thí dụ, dùng std:vector hay std::string thay vì dùng kiểu mãng đơn thuần sẽ không những là cho "đời sống dễ thở hơn", mà còn là một cách hữu hiệu để viết phần mềm được an toàn và linh hoạt hơn.

STL nguyên là một thư viện của hãng HP và sau đó là của SGI, trước khi nó được nhận vào thành chuẩn C++. Tiêu chuẩn thì không tham chiếu nó bằng cái tên "STL", khi đa phần nó chỉ là bộ phận tiêu chuẩn. Tuy vậy, nhiều người vẫn dùng khái niệm "STL" này để phân biệt nó với phần còn lại của thư viện C++ như là IOstream, quốc tế hóa (kí tự và ngôn ngữ trình bày), chẩn đoán, thư viện C, v.v..

Một đề án mang tên STLPort, dựa cơ sở trên SGI STL, bảo trì các thiết lập mới của STL, IOStream và string. Các đề án khác cũng có những xây dựng đặc thù riêng của thư viện chuẩn với các mục tiêu thiết kế khác nhau. Mỗi nơi sản xuất hay phổ biến nhà trình dịch C++ đều bao gồm một sự thiết lập của thư viện, vì đây là phần quan trọng của tiêu chuẩn và lại là kỳ vọng của người lập trình.

C++ Các chức năng hướng đối tượng

C++ dẫn nhập thêm một số chức năng hướng đối tượng (OO) lên C. Nó cung cấp các lớp mà có 4 chức năng thông dụng trong các ngôn ngữ OO: tính trừu tượng, tính bao đóng, tính đa hình, và tính kế thừa.

Lưu ý: trong phần này các từ "hàm nội tại", "phương pháp", hay "hàm" đều có cùng một nghĩa là "phương pháp thuộc về một lớp".

**Tính đóng gói**[sửa | sửa mã nguồn]

C++ xây dựng tính đóng bằng cách cho phép mọi thành viên của một lớp có thể được khai báo bằng các từ khoá public, private, hay protected. (xem thêm các khái niệm cơ bản trong ngôn ngữ OOP). Một thành viên private chỉ có thể được truy cập từ các phương pháp (hàm nội tại) là thành viên của chính lớp đó hay được truy cập từ các hàm và các lớp được đặc biệt cho phép sử dụng bằng cách dùng từ khóa friend. Một thành viên protected của một lớp sẽ có thể truy cập được từ các thành viên (nào đó) của các lớp có tính kế thừa của nó hay cũng có thể truy cập được từ các thành viện của chính lớp đó và của mọi thành viên friend.

Nguyên lý của OOP là mọi và chỉ có các hàm là có thể truy cập được đến các giá trị nội tại của cùng lớp thì nên có tính đóng. C++ có hỗ trợ đặc tính này (qua các hàm thành viên và các hàm friend), nhưng C++ lại không là yêu cầu bắt buộc: người lập trình có thể khai báo các phần hay tất cả các giá trị nội tại là công cộng (*public*), và cũng cho phép làm cho toàn bộ lớp trở thành công cộng. Lí do là vì C++ hỗ trợ không chỉ lập trình hướng đối tượng mà còn hỗ trợ các mẫu hình yếu hơn như là lập trình mô-đun.

Một thói quen tốt cần có trong thực hành là khai báo mọi dữ liệu đều là riêng tư (*private*), hay ít nhất ở dạng bảo tồn, và sau đó, tạo ra một giao diện nhỏ (thông qua các phương pháp) cho người dùng của lớp này dấu đi các chi tiết thiết lập bên trong.

**Tính đa hình**

Khái niệm đa hình được dùng khá rộng rãi và là khái niệm bị lạm dụng cũng như không được định nghĩa rõ ràng.

Nói chung tính đa hình trong lập trình hướng muốn nói đến 1 đoạn code nhưng trong 2 trường hợp khác nhau có thể xuất ra 2 kết quả khác nhau. Vì tính chất ra nhiều kết quả khác nhau này nên nó được gọi là đa hình.

Trong trường hợp của C++, khái niệm này thường được nối kết với các tên của các hàm thành viên. Các hàm thành viên này có cùng tên, sự khác nhau chỉ có thể được dựa vào một hay cả hai yếu tố sau:

1. Số lượng và kiểu của các đối số (tức là nguyên mẫu của hàm) -- Tính chất này gọi là **đa hình tĩnh** (*static polymorphism*)

2. Kiểu lớp mà thực thể thực sự thuộc vào. Tính chất này được dùng khi hàm thành viên được định nghĩa là hàm *ảo* qua từ khóa virtual—tính chất này gọi là **đa hình động** (*dynamic polymorphism*)

Khi được gọi thì chương trình sẽ tùy theo hai yếu tố trên để xác định chính xác hàm nào phải được thực thi trong số các hàm cùng tên.

## EM BIẾT GÌ VỀ CÁC NGÔN NGỮ LẬP TRÌNH

Đã có hàng ngàn ngôn ngữ lập trình được thiết kế và mỗi năm lại có thêm nhiều ngôn ngữ lập trình mới xuất hiện. Các ngôn ngữ thường được nhắc đến là: *Ada*, *Algol*, *APL*, *Assembly*, *BASIC*, **C**, **C++**, *C#*, *COBOL*, *Delphi*, *DHTML*, *Fortran*, **Java**, *JavaScript*, *Lisp*, *Logo*, **Pascal**, *Perl*, *PHP*, *PL/SQL*, *Prolog*, *Python*, *Ruby*, *Visual Basic*, *Visual Foxpro*,… Sự phát triển của ngôn ngữ lập trình gắn liền với sự phát triển của tin học. Mỗi loại ngôn ngữ phù hợp hơn với một số lớp bài toán nhất định. Cùng với tên các ngôn ngữ lập trình, các thuật ngữ thường được nhắc tới như "lập trình cấu trúc", "lập trình hướng đối tượng", "lập trình web",…

Những ngôn ngữ lập trình hiện nay thường cung cấp các thư viện bao gồm nhiều hàm hỗ trợ giao diện người dùng và các thiết bị đầu cuối. Cập nhật dữ liệu theo thời gian thực là một hướng phát triển nhằm đáp ứng các nhu cầu đồng bộ hoá nhanh dữ liệu dùng chung cho nhiều nơi hay là để thoả mãn nhu cầu cần đồng bộ hoá dữ liệu của các dịch vụ (như trong ngân hàng, hàng không và quân sự). Ngoài việc hỗ trợ cho các giao diện, ngày nay hầu hết các hệ điều hành (UNIX/Linux, Netware và Windows) đều có khả năng đa luồng(*multithreading*) hay đa nhiệm (*multitasking*) nâng cao hiệu quả của máy tính. Do đó, các ngôn ngữ thường có thêm các hàm, thủ tục hay các biến cho phép người lập trình tận dụng điều này.

Dưới đây giới thiệu một số ngôn ngữ lập trình thông dụng: Algol, Basic, C, C++, Cobol, Fortran, Pascal, Java, Visual,…

- **Fortran** (hay **FORTRAN**) là một ngôn ngữ lập trình được phát triển từ những năm 1950 và vẫn được dùng nhiều trong tính toán khoa học cho đến hơn nửa thế kỉ sau. Tên gọi này xuất phát từ việc ghép các từ tiếng Anh ***For**mula **Tran**slator* nghĩa là *dịch công thức*. Các phiên bản đầu có tên chính thức là FORTRAN. Điểm yếu của FORTRAN là thiếu hỗ trợ trực tiếp cho các kết cấu có cấu trúc, kiểu dữ liệu còn nghèo, không thuận lợi cho xử lí xâu. Fortran được phát triển ban đầu như là một ngôn ngữ thủ tục. Tuy nhiên các phiên bản mới của Fortran đã có các tính năng hỗ trợ lập trình hướng đối tượng.
- **COBOL** ra đời năm 1959, được chấp nhận dùng cho các ứng dụng xử lí dữ liệu thương mại, kinh doanh.
- **ALGOL** do Uỷ ban các nhà tin học Châu Âu và Hoa Kì (Committee of EU & USA computer scientists) tạo ra năm 1958, là ngôn ngữ tiên phong đưa ra tập các thủ tục, định kiểu dữ liệu cực kì phong phú,... và có ảnh hưởng mạnh tới các ngôn ngữ ra đời sau.
- **BASIC** là ngôn ngữ được phát triển năm 1963 bởi John Kemeny và Thomas Kurtz. BASIC là ngôn ngữ còn nhiều hạn chế như thực hiện câu lệnh chủ yếu là tuần tự từ trên xuống, điều khiển chương trình chỉ nhờ lệnh IF...THEN và GOSUB.
- **LISP** do John McCarthy của MIT (Học viện Kĩ thuật Massachusetts - Massachusetts Institute of Technology) tạo ra ngôn ngữ LISt Processing (LISP) năm 1958, là ngôn ngữ đặc biệt thích hợp cho thao tác kí hiệu và xử lí danh sách thường gặp trong các bài toán tổ hợp. Đặc biệt thích hợp cho việc chứng minh định lí, gần đây được dùng để phát triển hệ chuyên gia và các hệ thống dựa trên tri thức.
- **PASCAL** do Giáo sư Niclaus Wirth phát triển dựa trên Algol năm 1970. Pascal là tên nhà toán học và triết học Blaise Pascal. Pascal là ngôn ngữ đặc biệt thích hợp cho kiểu lập trình cấu trúc. Cho đến nay, Pascal vẫn được dùng để giảng dạy về lập trình trong nhiều trường trung học và đại học trên thế giới. Đó là ngôn ngữ cho phép mô tả thuật toán thuận tiện nhất. Pascal cũng phục vụ nhiều ứng dụng kĩ nghệ khoa học và lập trình hệ thống. Phần lớn hệ điều hành Macintosh được viết bằng Pascal. Hệ sắp chữ TeX được Donald Knuth viết bằng ngôn ngữ mang nhiều yếu tố của Pascal. Trình biên dịch Free Pascal được viết bằng Pascal là một trình biên dịch mạnh mẽ có khả năng biên dịch cả ứng dụng cũ và mới (phân phối miễn phí dưới giấy phép GNU) hỗ trợ nhiều hệ điều hành.
- **C** là ngôn ngữ được xây dựng bởi Dennis Ritchie năm 1972 dùng trong hệ điều hành UNIX. Từ đó C còn được dùng trong nhiều hệ điều hành khác và trở thành một trong những ngôn ngữ phổ dụng nhất. C rất hiệu quả và được ưa chuộng nhất để viết các phần mềm hệ thống, mặc dù nó cũng được dùng cho việc viết các ứng dụng. Ngoài ra, C cũng thường được dùng làm ngôn ngữ giảng dạy lập trình. Ngày nay, C được phát triển và mang nhiều tính năng mới làm cho nó mềm dẻo thêm. Có nhiều câu chuyện về sự ra đời của C và hệ điều hành Unix, trong đó có chuyện cho rằng sự phát triển của C là kết quả của các lập trình viên muốn chơi trò *Space Travel* (du lịch vũ trụ). Để chơi Space Travel thuận lợi hơn, Thompson và Ritchie đã viết hệ điều hành bằng hợp ngữ cho máy PDP-7 đang để không trong văn phòng. Tiếp tục, muốn xuất hệ điều hành này sang máy PDP-11 của văn phòng, họ quyết định dùng một ngôn ngữ bậc cao để hệ điều hành có thể xuất được dễ dàng từ máy tính này sang máy khác. Vậy nên họ đã sáng tạo ra một ngôn ngữ mới là C. Cho đến năm 1973, C đã trở nên đủ mạnh để dùng viết một số thành phần quan trọng của hệ điều hành Unix (trước đây viết bằng Assembly trong

các máy từ PDP-11 đến PDP-20). Đây là lần đầu tiên, nhân của một hệ điều hành được viết bằng một ngôn ngữ khác Assembly.

Dennis Ritchie

- **C++** là ngôn ngữ lập trình hỗ trợ lập trình cấu trúc (thủ tục, dữ liệu trừu trượng), lập trình hướng đối tượng. Từ thập niên 1990, C++ đã trở thành một trong những ngôn ngữ phổ biến nhất. C++ góp phần xây dựng những ứng dụng lớn nhất hiện nay như hệ điều hành Windows, trình duyệt và máy tìm kiếm Google,… Người phát triển C++ là Bjarne Stroustrup của Bell Labs năm 1983. Ý tưởng tạo ra một ngôn ngữ mới bắt nguồn từ khi ông viết luận án tiến sĩ. Trong suốt thập niên 1980 "C với các lớp" được coi là một bản nâng cao của ngôn ngữ C. Tên C++ được đặt ra bởi Rick Mascitti (giữa năm 1983) và lần đầu tiên được dùng trong tháng 12/1983. Cái tên C++ cho biết C++ là ngôn ngữ được phát triển trên cơ sở ngôn ngữ C.
- **Java** được khởi đầu bởi James Gosling và các đồng nghiệp ở Sun Microsystems năm 1991 là một phần của *Dự án Xanh*. Ban đầu ngôn ngữ này được gọi là Oak (có nghĩa là cây sồi, do bên ngoài cơ quan của ông Gosling có trồng nhiều loại cây này). Họ dự định phát triển ngôn ngữ này thay cho C++. Công ti Sun Microsystems đang giữ bản quyền và phát triển Java thường xuyên. Java được phát hành vào năm 1994, rồi nó trở nên nổi tiếng khi Netscape tuyên bố tại hội thảo *SunWorld năm 1995* là trình duyệt Navigator của họ sẽ hỗ trợ Java. Java có thể tương thích với nhiều họ máy như PC, Macitosh, tương thích với nhiều hệ điều hành như Windows, Linux. Người ta nói Java là ngôn ngữ lập trình một lần (trên một máy) nhưng có thể chạy nhiều lần (trên nhiều máy). Java được sử dụng chủ yếu để lập trình trên môi trường mạng và Internet.

James Gosling

# Chương II
# Chương trình đơn giản

- Cấu trúc chương trình;
- Các kiến thức cơ bản về kiểu dữ liệu, phép toán, biểu thức, câu lệnh gán, tổ chức vào/ra đơn giản;
- Cách thực hiện chương trình trong môi trường C++.

# §3. CẤU TRÚC CHƯƠNG TRÌNH

## 1. Cấu trúc chung

Nói chung, chương trình được viết bằng một ngôn ngữ lập trình bậc cao thường gồm *phần khai báo* và *phần thân.* Phần thân chương trình nhất thiết phải có. Phần khai báo có thể có hoặc không tuỳ theo từng chương trình cụ thể.

Khi diễn giải cú pháp của ngôn ngữ lập trình người ta thường sử dụng ngôn ngữ tự nhiên. Các diễn giải bằng ngôn ngữ tự nhiên được đặt giữa cặp dấu < và >. Các thành phần của chương trình có thể có hoặc không được đặt trong cặp dấu [ và ].

Với quy ước trên, cấu trúc của một chương trình có thể được mô tả như sau:

[< *phần khai báo* >]

<*phần thân*>

## 2. Các thành phần của chương trình

### *a) Phần khai báo*

Có thể có các khai báo cho: thư viện, không gian tên,hằng, biến và chương trình con.

**Khai báo thư viện**

Mỗi ngôn ngữ lập trình thường có sẵn một số thư viện cung cấp một số chương trình thông dụng đã được lập sẵn. Để sử dụng các chương trình đó cần khai báo thư viện chứa nó.

***Ví dụ.*** Khai báo thư viện

- Trong Pascal:

```
uses crt;
```

- Trong C++:

```
#include <stdio.h>
```

Thư viện *crt* trong Pascal hoặc *stdio.h* trong C++ cung cấp các chương trình có sẵn để làm việc với màn hình văn bản và bàn phím. Ví dụ, muốn xoá những gì đang có trên màn hình,:

- Trong Pascal, sau khi khai báo thư viện *crt*, ta dùng lệnh:

```
clrscr;
```

- Trong C++, sau khi khai báo thư viện *stdio.h*, ta dùng lệnh:

```
clrscr();
```

**Khai báo hằng**

***Ví dụ.*** Khai báo hằng

- Trong Pascal:

```
const MaxN = 1000;
      PI = 3.1416;
      KQ = 'Ket qua:';
```

- Trong C++:

```
const int MaxN = 1000;
const float PI = 3.1416;
```

Khai báo hằng thường được sử dụng cho những giá trị xuất hiện nhiều lần trong chương trình.

**Khai báo không gian tên**

Nhờ vào namespace, ta có thể nhóm các thực thể như lớp, đối tượng và các hàm dưới một tên gọi tương ứng với từ khóa namespace. Theo cách này, các phạm vi toàn cục lại được chia nhỏ ra thành các phạm vi toàn cục con, mà mỗi một phạm vi toàn cục con này có một tên gọi riêng.

Để khai báo namespace, ta sử dụng từ khóa namespace theo cú pháp sau

```
namespace tên_của_namespace
{
  các_thực_thể
}
```

Để truy cập đến các thực thể của namespace, ta sử dụng toán tử phạm vi ::

**Khai báo biến**

Tất cả các biến dùng trong chương trình đều phải đặt tên và phải khai báo cho chương trình dịch biết để lưu trữ và xử lí. Biến chỉ nhận một giá trị tại mỗi thời điểm thực hiện chương trình được gọi là *biến đơn*.

***Ví dụ***

Khi khảo sát phương trình đường thẳng $ax + by + c = 0$, các hệ số $a$, $b$, $c$ có thể được khai báo như những biến đơn.

Cách khai báo biến được trình bày riêng trong Đ3.

Khai báo và sử dụng chương trình con được trình bày trong chương VI.

***b) Phần thân chương trình***

Dãy lệnh trong phạm vi được xác định bởi cặp dấu hiệu mở đầu và kết thúc tạo thành thân chương trình**.**

***Ví dụ.*** Thân chương trình trong C++:

```
int main()
{
  cout << "Hello world!" << endl;
  return 0;
}
```

## 3. Ví dụ chương trình đơn giản

Dưới đây xét một vài ví dụ về những chương trình đơn giản.

***Ví dụ 1.*** Chương trình sau thực hiện việc đưa ra màn hình thông báo "*Xin chao cac ban!*".

| Trong Pascal | Trong C++ |
|---|---|
| **program** vi_du;<br>**begin**<br>  writeln('Xin chao cac ban!');<br>**end.** | #include <stdio.h><br>**main**()<br>{<br>  Cout<<("Xin chao cac ban!");<br>} |
| - Phần khai báo chỉ có khai báo tên chương trình gồm tên dành riêng *program* và tên chương trình là *vi_du.*<br>- Phần thân chương trình chỉ có một câu lệnh *writeln*, đưa thông báo ra màn hình. | - Phần khai báo chỉ có một câu lệnh *include* khai báo thư viện *stdio.h*.<br>- Phần thân chương trình chỉ có một câu lệnh cout đưa thông báo ra màn hình. |

## §4. MỘT SỐ KIỂU DỮ LIỆU CHUẨN

Các bài toán trong thực tế thường có dữ liệu vào và kết quả ra thuộc những kiểu dữ liệu quen biết như số nguyên, số thực, kí tự,... Khi lập trình cho những bài toán như vậy, khi cần người lập trình sử dụng các kiểu dữ liệu đó thường gặp một số hạn chế nhất định, phụ thuộc vào một số yếu tố như dung lượng bộ nhớ, khả năng xử lí của CPU,...

Vì vậy, mỗi ngôn ngữ lập trình thường cung cấp một số kiểu dữ liệu chuẩn cho biết phạm vi giá trị có thể lưu trữ, dung lượng bộ nhớ cần thiết để lưu trữ và các phép toán tác động lên dữ liệu. Dưới đây xét một số kiểu dữ liệu chuẩn thường dùng cho các biến đơn trong Pascal.

### 1. Kiểu nguyên

| Kiểu | Bộ nhớ lưu trữ một giá trị | Phạm vi giá trị |
|---|---|---|
| Short | 2 byte | signed: $-2^{15}$ -> $2^{15}-1$<br>unsigned: 0 -> $2^{16}-1$ |
| **Int** | 4 byte | signed: $-2^{31}$ -> $2^{31}-1$<br>unsigned: 0 -> $2^{32}-1$ |
| **Long** | 4 byte | signed: $-2^{31}$ -> $2^{31}-1$<br>unsigned: 0 -> $2^{32}-1$ |
| **Long long** | 8 byte | signed: $-2^{63}$ -> $2^{63}-1$<br>unsigned: 0 -> $2^{64}-1$ |

### 2. Kiểu thực

Có nhiều kiểu dùng để khai báo các đại lượng nhận giá trị là số thực. Thường dùng hơn cả là các kiểu được liệt kê trong bảng sau:

| Kiểu | Bộ nhớ lưu trữ một giá trị | Phạm vi giá trị |
|---|---|---|
| **float** | 4 byte | 7 số thập phân |
| **double** | 8 byte | 15 số thập phân |
| **Long double** | 8 byte | 15 số thập phân |

### 3. Kiểu kí tự

Ta hiểu kí tự là các kí tự thuộc bộ mã ASCII gồm 256 kí tự có mã ASCII thập phân từ 0 đến 255.

Ví dụ, kí tự *A* có mã ASCII là 65, kí tự *a* có mã ASCII là 97. Kí tự đặc biệt, dùng để thể hiện sự ngăn cách giữa hai từ viết liên tiếp trong các văn bản, là dấu cách. Dấu cách được gõ bằng phím **Space** - phím dài nhất trên bàn phím và có mã ASCII bằng 32.

| Kiểu | Bộ nhớ lưu trữ một giá trị | Phạm vi giá trị |
|---|---|---|
| **char** | 1 byte | 256 kí tự trong bộ mã ASCII |
| **wchar_t** | 2/4 byte | |

### 4. Kiểu lôgic

| Kiểu | Bộ nhớ lưu trữ một giá trị | Phạm vi giá trị |
|---|---|---|
| **bool** | 1 byte | true hoặc false |

***Ghi chú***

- Một số ngôn ngữ lập trình mô tả giá trị lôgic bằng cách khác.
- Người lập trình cần tìm hiểu đặc trưng của các kiểu dữ liệu chuẩn được xác định bởi bộ dịch và sử dụng để khai báo biến.

## §5. KHAI BÁO BIẾN

Như ví dụ ở bài trước, ta thấy rằng, muốn sử dụng một biến trong C++, ta cần khai báo biến với kiểu dữ liệu mà ta mong muốn. Cấu trúc khai báo

**<Tên kiểu dữ liệu> <Tên biến>;**

**Ví dụ**

```
int a; //Khai báo biến a kiểu nguyên
float mynumber;  //Khai báo biến mynumber kiểu float
bool istrue;  //Khai báo biến istrue kiểu bool
long num1, num2, num3;  //Khai báo ba biến num1, num2, num3 cùng kiểu long
```

**Chú ý:**

- Nếu khi khai báo biến thuộc các kiểu nguyên mà ta không sử dụng khai báo có dấu (signed) hoặc không dấu (unsigned), thì chương trình dịch mặc định sẽ quy định là kiểu nguyên có dấu.

  ```
  int mynum; //tương đương signed int mynum;
  ```

- Đối với kiểu char thì có ngoại lệ. Chúng ta nên khai b|o tường minh là signed char hoặc unsigned char.
- Đối với signed int và unsigned int có thể viết đơn giản là signed hoặc unsigned.

***Một số chú ý khi khai báo biến:***

- Cần đặt tên biến sao cho gợi nhớ đến ý nghĩa của biến đó. Điều này rất có lợi cho việc đọc, hiểu và sửa đổi chương trình khi cần thiết.
  Ví dụ, không nên vì cho ngắn gọn mà đặt tên biến là d1, d2 mà nên đặt là dtoan, dtin gợi nhớ tới ngữ nghĩa của các biến đó là điểm toán, điểm tin của học sinh.
- Không nên đặt tên biến quá ngắn hay quá dài, dễ mắc lỗi khi viết nhiều lần tên biến. Ví dụ, không nên dùng d1, d2 hay diemmontoan, diemmontin cho điểm toán, điểm tin của học sinh.
- Khi khai báo biến cần đặc biệt lưu ý đến phạm vi giá trị của nó. Ví dụ, khi khai báo biến biểu diễn số học sinh của một lớp có thể sử dụng kiểu short, nhưng biến biểu diễn số học sinh của toàn trường thì phải là kiểu int

## §6. PHÉP TOÁN, BIỂU THỨC, CÂU LỆNH GÁN

Để mô tả các thao tác trong thuật toán, mỗi ngôn ngữ lập trình đều xác định và sử dụng một số khái niệm cơ bản: phép toán, biểu thức, gán giá trị cho biến.

Dưới đây sẽ xét các khái niệm đó trong Pascal.

### 1. Phép toán

Tương tự trong toán học, trong các ngôn ngữ lập trình đều có những phép toán số học như cộng, trừ, nhân, chia trên các đại lượng thực, các phép toán chia nguyên và lấy phần dư, các phép toán quan hệ,…

Bảng dưới đây là kí hiệu các phép toán đó trong toán và trong C++:

| Phép toán | Trong toán học | Trong Pascal |
|---|---|---|
| Các phép toán số học với số nguyên | + (cộng), - (trừ), . (nhân), mod (lấy phần dư) | +, -, *, % |
| Các phép toán số học với số thực | + (cộng), - (trừ), . (nhân),: (chia) | +, -, *, / |
| Các phép toán quan hệ | < (nhỏ hơn), ≤ (nhỏ hơn hoặc bằng), > (lớn hơn), ≥ (lớn hơn hoặc bằng), = (bằng), ≠(khác) | <, <=, >, >=, =, <> |
| Các phép toán lôgic | phủ định, và, hoặc | !, \|\|, && |

***Chú ý***

- Kết quả của các phép toán quan hệ cho giá trị lôgic.
- Một trong những ứng dụng của phép toán lôgic là để tạo ra các biểu thức phức tạp từ các quan hệ đơn giản.

### 2. Biểu thức số học

Trong lập trình, biểu thức số học là một biến kiểu số hoặc một hằng số hoặc các biến kiểu số và các hằng số liên kết với nhau bởi một số hữu hạn phép toán số học, các dấu ngoặc tròn ( và ) tạo thành một biểu thức có dạng tương tự như cách viết trong toán học với những quy tắc sau:

- Chỉ dùng cặp ngoặc tròn để xác định trình tự thực hiện phép toán trong trường hợp cần thiết;
- Viết lần lượt từ trái qua phải;
- Không được bỏ qua dấu nhân (*) trong tích.

Các phép toán được thực hiện theo thứ tự:

- Thực hiện các phép toán trong ngoặc trước;
- Trong dãy các phép toán không chứa ngoặc thì thực hiện từ trái sang phải, theo thứ tự các phép toán nhân (*), chia (/), chia nguyên (div), lấy phần dư (mod) thực hiện trước và các phép toán cộng (+), trừ (-) thực hiện sau.

***Ví dụ***

| Biểu thức trong Toán học | Trong C++ |
|---|---|
| 5a+6b | 5*a + 6*b |
| $\frac{xy}{z}$ | x*y/z |
| $Ax^2 + Bx + C$ | A*x*x + B*x + C |
| $\frac{x+y}{x-\frac{1}{2}} - \frac{x-z}{xy}$ | (x + y)/(x - 1/2) - (x - z)/(x*y) |

***Chú ý***

a) Nếu biểu thức chứa một hằng hay biến kiểu thực thì ta có biểu thức số học thực, giá trị của biểu thức cũng thuộc kiểu thực.

b) Trong một số trường hợp nên dùng biến trung gian để có thể tránh được việc tính một biểu thức nhiều lần.

## 3. Hàm số học chuẩn

Để lập trình được dễ dàng, thuận tiện hơn, các ngôn ngữ lập trình đều có thư viện chứa một số chương trình tính giá trị những hàm toán học thường dùng. Các chương trình như vậy được gọi là các *hàm số học chuẩn*. Mỗi hàm chuẩn có tên chuẩn riêng. Đối số của hàm là một hay nhiều biểu thức số học và được đặt trong cặp ngoặc tròn ( và ) sau tên hàm. Bản thân hàm chuẩn cũng được coi là một biểu thức số học và nó có thể tham gia vào biểu thức số học như một toán hạng (giống như biến và hằng). Kết quả của hàm có thể là nguyên hoặc thực hay phụ thuộc vào kiểu của đối số.

Bảng dưới đây cho biết một số hàm chuẩn thường dùng.

| Hàm | Biểu diễn Toán học | Biểu diễn trong C++ | Kiểu đối số | Kiểu kết quả |
|---|---|---|---|---|
| Lũy thừa | $x^y$ | **Pow(x,y)** | Thực hoặc nguyên | Theo kiểu của đối số |
| Căn bậc hai | $\sqrt{x}$ | **sqrt(**x**)** | Thực hoặc nguyên | Thực |
| Giá trị tuyệt đối | $\|x\|$ | **abs(**x**)** | Thực hoặc nguyên | Theo kiểu của đối số |
| Lôgarit tự nhiên | lnx | **ln(**x**)** | Thực | Thực |
| Luỹ thừa của số e | $e^x$ | **exp(**x**)** | Thực | Thực |
| Sin | sinx | **sin(**x**)** | Thực | Thực |
| Cos | cosx | **cos(**x**)** | Thực | Thực |

***Ví dụ***

Biểu thức toán học $\frac{-b+\sqrt{b^2-4ac}}{2a}$ trong Pascal có thể viết dưới dạng:

```
(-b+sqrt(b*b - 4*a*c))/(2*a)
```

hoặc

```
(-b+sqrt(sqr(b)- 4*a*c))/2/a
```

Ngoài những hàm số học chuẩn trên, còn có các hàm chuẩn khác được giới thiệu trong những phần sau.

## 4. Biểu thức quan hệ

Hai biểu thức cùng kiểu liên kết với nhau bởi phép toán quan hệ cho ta một biểu thức quan hệ.

Biểu thức quan hệ có dạng:

***<biểu thức 1> <phép toán quan hệ> <biểu thức 2>***

trong đó, *biểu thức 1* và *biểu thức 2* cùng là xâu hoặc cùng là biểu thức số học.

***Ví dụ***

```
x < 5
i+1 >= 2*j
```

Biểu thức quan hệ được thực hiện theo trình tự:

- Tính giá trị các biểu thức.
- Thực hiện phép toán quan hệ.

Kết quả của biểu thức quan hệ là giá trị lôgic: *true* (đúng) hoặc *false* (sai).

Trong ví dụ trên, nếu *x* có giá trị 3, thì biểu thức *x* < 5 có giá trị *true*. Nếu *i* có giá trị 2 và *j* có giá trị 3 thì biểu thức *i* + 1 >= 2**j* sẽ cho giá trị *false*.

***Ví dụ***

Điều kiện để điểm *M* có toạ độ (*x*; *y*) thuộc hình tròn tâm *I*(*a*; *b*), bán kính *R* là:

```
sqrt((x-a)*(x-a) + (y-b)*(y-b)) <= R
```

hoặc

```
sqr(x-a) + sqr(y-b) <= R*R
```

**5. Biểu thức lôgic**

Biểu thức lôgic đơn giản là biến lôgic hoặc hằng lôgic.

Biểu thức lôgic là các biểu thức lôgic đơn giản, các biểu thức quan hệ liên kết với nhau bởi phép toán lôgic. Giá trị biểu thức lôgic là *true* hoặc *false* (xem phụ lục 5. *Bảng giá trị phép toán lôgic*). Các biểu thức quan hệ thường được đặt trong cặp ngoặc ( và ).

Dấu phép toán ***!*** được viết trước biểu thức cần phủ định, ví dụ:

***!***($x < 1$) thể hiện phát biểu "$x$ không nhỏ hơn 1" và điều này tương đương với biểu thức quan hệ $x >= 1$.

Các phép toán && và || dùng để kết hợp nhiều biểu thức lôgic hoặc quan hệ, thành một biểu thức thường được dùng để diễn tả các điều kiện phức tạp.

***Ví dụ 1***

Để thể hiện điều kiện $5 \le x \le 11$, trong C++ cần phải tách thành phát biểu dưới dạng "$5 \le x$ và $x \le 11$" và được viết như sau:

```
(5 <= x) && (x <= 11)
```

***Ví dụ 2***

Giả thiết *M* và *N* là hai biến nguyên. Điều kiện xác định *M* và *N* đồng thời chia hết cho 3 hay đồng thời không chia hết cho 3 được thể hiện trong C++ như sau:

```
((M % 3 == 0) && (N % 3 == 0)) || ((M % 3 != 0) && (N % 3 != 0))
```

**6. Câu lệnh gán**

Lệnh gán là một trong những lệnh cơ bản nhất của các ngôn ngữ lập trình.

Trong Pascal câu lệnh gán có dạng:

***<tên biến> = <biểu thức>;***

Trong trường hợp đơn giản, *tên biến* là tên của biến đơn. Kiểu của giá trị biểu thức phải phù hợp với kiểu của biến.

Chức năng của lệnh gán là đặt cho biến có tên ở vế trái dấu "=" giá trị mới bằng giá trị của biểu thức ở vế phải.

***Ví dụ***

```
x1= (-b - sqrt(b*b - 4*a*c))/(2*a);
x2= -b/a - x1;
z= z - 1;
i= i + 1;
```

Trong ví dụ trên, ý nghĩa của lệnh gán thứ ba là giảm giá trị của biến *z* một đơn vị. Ý nghĩa của lệnh gán thứ tư là tăng giá trị của biến *i* lên một đơn vị.

**Toán tử gán hợp nhất**

Khi muốn thay đổi giá trị của một biến, chúng ta có thể sử dụng cách viết thông thường, nhưng trong C++ nó hỗ trợ các toán tử viết tắt.

| Toán tử | Ví dụ | Ý nghĩa | Phạm vi |
|---|---|---|---|
| += | a+=b | a=a+b | Phép toán số học |
| -= | a-=b | a=a-b | Phép toán số học |
| *= | a*=b | a=a*b | Phép toán số học |
| /= | a/=b | a=a/b | Phép toán số học |
| %= | a%=b | a=a%b | Phép toán số học |

| | | | |
|---|---|---|---|
| &= | a&=b | a=a&b | Phép toán bit |
| \|= | a\|=b | a=a\|b | Phép toán bit |
| ^= | a^=b | a=a^b | Phép toán bit |
| >>= | a>>=b | a=a>>b | Phép toán bit |
| <<= | a<<=b | a=a<<b | Phép toán bit |

**Toán tử tăng và giảm**

Một cách viết thu gọn hơn nữa, đó là sử dụng toán tử tăng và giảm. Nếu trong biểu thức a+=b, với b = 1 thì ta có thể viết thành a++. Tương tự, nếu a-=b, b = 1 thì ta có thể viết a--.

Chúng ta cũng lưu ý rằng, toán tử này có chút khác biệt. Nó có thể nằm trước hoặc nằm sau toán hạng. Có nghĩa là có thể có a++ hoặc ++a (tương ứng a-- hoặc --a).

| Phép toán | Ý nghĩa |
|---|---|
| a++; | Thực hiện phép toán trước, sau đó mới thực hiện toán tử. |
| ++a; | Thực hiện toán tử trước, sau đó mới thực hiện phép toán. |
| a--; | Tương tự a++; |
| --a; | Tương tự ++a; |

## §7. CÁC THỦ TỤC CHUẨN VÀO/RA ĐƠN GIẢN

Để khởi tạo giá trị ban đầu cho biến, ta có thể dùng lệnh gán để gán một giá trị cho biến. Như vậy, mỗi chương trình luôn làm việc với một bộ dữ liệu vào. Để chương trình có thể làm việc với nhiều bộ dữ liệu vào khác nhau, thư viện của các ngôn ngữ lập trình cung cấp một số chương trình dùng để đưa dữ liệu vào và đưa dữ liệu ra.

Những chương trình đưa dữ liệu vào cho phép đưa dữ liệu từ bàn phím hoặc từ đĩa vào gán cho các biến, làm cho chương trình trở nên linh hoạt, có thể tính toán với nhiều bộ dữ liệu đầu vào khác nhau. Kết quả tính toán được lưu trữ tạm thời trong bộ nhớ. Những chương trình đưa dữ liệu ra dùng để đưa các kết quả này ra màn hình, in ra giấy hoặc lưu trên đĩa.

Các chương trình đưa dữ liệu vào và ra đó được gọi chung là các *thủ tục chuẩn vào/ra đơn giản*.

Trong phần này, ta sẽ xét các *thủ tục chuẩn vào/ra đơn giản* của Pascal để nhập dữ liệu vào từ bàn phím và đưa thông tin ra màn hình.

### 1. Nhập dữ liệu vào từ bàn phím

Việc nhập dữ liệu từ bàn phím được thực hiện bằng thủ tục chuẩn:

***cin>>biến_1>>…>>biến_n;***

*Các biến số biến_1,..., biến_n cần được khai báo. Thông thường, những biến này chưa được khởi tạo giá trị. Sau đây là một vài ví dụ về việc sử dụng đối tượng cin:*

```
int age;
cin>>age;
float f;
cin>>f;
string s;
cin>>s;
```

*Chú ý rằng kiểu dữ liệu của biến được sử dụng trong đối tượng cin này. Nếu có một sự vi phạm nào về kiểu dữ liệu (ví dụ biến là int, nhưng khi nhập ta lại nhập vào một kí tự không phải là số) thì chương trình dịch sẽ bỏ qua việc khởi tạo giá trị cho biến đó. Chương trình hoàn toàn không phát sinh lỗi (process returned 0).*

### 2. Đưa dữ liệu ra màn hình

Để đưa dữ liệu ra màn hình, Pascal cung cấp thủ tục chuẩn:

**cout<<biến_1<<…<<biến_n;**

Trong đó, biến_1,…,biến_n: là các biến số. Chúng đã được khởi tạo giá trị. Nếu biến chưa khởi tạo giá trị, ta sẽ nhận được một lỗi khi thực thi chương trình. Chương trình dịch sẽ thông báo về việc sử dụng biến mà không khởi tạo giá trị cho nó. Các biến này có thể là biến thuộc kiểu dữ liệu nguyên thủy hoặc tham chiếu. Đối với các biến là các đối tượng thể hiện của các lớp, ta sẽ thảo luận ở mục "chồng chất toán tử nhập xuất" trong chương lập trình hướng đối tượng.

***Ví dụ:***

```
cout<<"Hello, world !";//In câu Hello, world ! ra màn hình
cout<<120;//In số 120 ra màn hình
cout<<x;//In giá trị của biến x ra màn hình
```

Đối tượng cout kết hợp với toán tử << có thể được ghép nhiều lần.

```
cout<<"Chao ban"<<" ban may tuoi";
cout<<"Chuc mung"<<endl;
cout<<x<<"+"<<y<<"="<<(x+y);
```

*Khi sử dụng đối tượng cout và cin, ta cần khai báo không gian sử dụng namespace là std. Hoặc, có thể viết ngắn gọn hơn std::*

| *Chương trình 1* | *Chương trình 2* |
|---|---|
| `#include <iostream>`<br>`using namespace std;`<br>`int main()`<br>`{`<br>`  cout<<"Hello";`<br>`}` | `#include <iostream>`<br>`int main()`<br>`{`<br>`  std::cout<<"Hello";`<br>`}` |

Đối tượng cin và xâu kí tự: trong ví dụ trên, tôi đã sử dụng đối tượng cin để tách một xâu kí tự và gán cho biến xâu kí tự s. Khi sử dụng đối tượng cin với xâu kí tự, cần lưu ý một điểm: đối tượng cin sẽ dừng việc trích tách nếu nó đọc thấy một kí tự trắng trong xâu kí tự đó (có nghĩa, nếu xâu nhập vào là "Toi di hoc" – thì nó chỉ tách được xâu "Toi" và gán cho biến s). Để khắc phục nhược điểm này của đối tượng cin, C++ cung cấp cho chúng ta một hàm khác là hàm getline, có chức năng tương tự.

***Cú pháp:***

***getline(chuẩn_nhập_dữ_liệu, tên_biến_xâu)***

Khi nhập xuất dữ liệu từ bàn phím và màn hình, tham số chuẩn_nhập_xuất_dữ_liệu luôn sử dụng là **cin**. Nếu làm việc với tập tin file, thì tham số này sẽ tương ứng với tên của file. Chúng ta sẽ tìm hiểu trường hợp này trong chương 6 của giáo trình.

***Chương trình***

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string s;
   cout<<"Nhap ten: ";
   getline(cin, s);
   cout<<"Chao ban "<<s;
   return 0;
}
```

Trong hằng xâu kí tự, có thể chứa các kí tự đặc biệt như kí tự xuống dòng, đặt tab… Sau đây là một vài kí tự đặc biệt đó và ý nghĩa của chúng.

| **Kí hiệu** | **Ý nghĩa** |
|---|---|
| \n | Xuống dòng |
| \r | Di chuyển toàn bộ kí tự sau dấu \r đè lên các kí tự trước đó. Nếu số kí tự sau nhiều hơn số kí tự trước dấu \r, thì kết quả in ra sẽ là toàn bộ kí tự nằm sau. Ví dụ "abc\r1234" -> sẽ in ra 1234, nếu "abc\r12" -> sẽ in ra 12c. |
| \t | Đặt tab |
| \v | Đặt tab dọc |
| \b | Đặt backspace |
| \f | Đặt dấu form feed |
| \a | Tạo âm thanh beep |
| \', \", \?, \\ | Tạo các kí tự ', ", ?, \ |

# §8. SOẠN THẢO, DỊCH, THỰC HIỆN HIỆU CHỈNH CHƯƠNG TRÌNH

Để có thể thực hiện chương trình được viết bằng một ngôn ngữ lập trình, ta cần soạn thảo, sử dụng chương trình dịch để dịch chương trình đó sang ngôn ngữ máy. Các hệ thống lập trình cụ thể thường cung cấp phần mềm phục vụ cho việc soạn thảo, dịch và hiệu chỉnh chương trình.

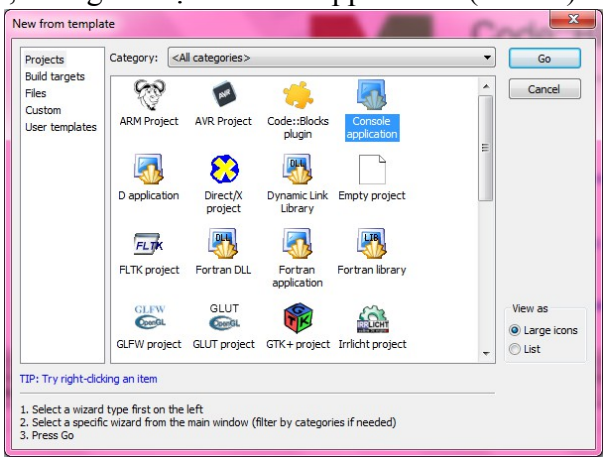
Với ngôn ngữ C, người ta thường dùng các phần mềm Turbo C hay Dev-C++, CodeBlocks. CodeBlocks cho phép tận dụng nhiều hơn khả năng của hệ thống.

Trước tiên, chúng ta sẽ tìm hiểu cách tạo dự án, biên dịch một tập tin C++ trên CodeBlocks. Độc giả cũng cần lưu ý rằng, CodeBlocks tổ chức công việc theo các dự án. Chúng ta có thể biên dịch từng tập tin cpp một c|ch đơn lẻ.

Tuy nhiên, làm việc theo dự án sẽ giúp ích cho chúng ta rất nhiều khi làm việc với những tác vụ lớn.

Đầu tiên chúng ta khởi động codeblocks, sau đó vào File → New → Project.

Trong hộp thoại hiện ra, chúng ta chọn console application (Hình 1).




Hình 1 – Tạo mới dự án trong CodeBlocks

Và nhấp Go, sau đó nhấp Next. Trong hộp thoại tiếp theo, ta chọn C++ và nhấp Next.

Hộp thoại yêu cầu điền thông tin về dự án sẽ xuất hiện. Hãy điền tên dự án, vị trí lưu trữ dự án. Sau đó nhấp Next. Cuối cùng nhấp Finish.

Trong cửa sổ quản lý dự án, ta nhấp đôi chuột vào tệp main.cpp. Nội dung soạn thảo sẽ được nhập vào trong tập tin này.

Nếu ta muốn bổ sung các tập tin khác hoặc các lớp đối tượng, ta có thể bổ sung chúng từ menu File > New.

**Biên dịch chương trình:**

+ Nhấp vào Build > Build and Run.

+ Hoặc nhấp phím F9.

# BÀI TẬP VÀ THỰC HÀNH 1

## 1. Mục đích, yêu cầu

- *Giới thiệu một chương trình Pascal hoàn chỉnh đơn giản;*
- *Làm quen với một số dịch vụ cơ bản của CodeBlocks trong việc soạn thảo, lưu trữ, dịch và thực hiện chương trình.*

## 2. Nội dung

***a)*** Gõ chương trình sau:

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    float a,b,c,x1,x2;
    cout << "Nhap a:";
    cin>>a;
    cout << "\n Nhap b:";
    cin>>b;
    cout << "\n Nhap c:";
    cin>>c;
    float D=b*b-4*a*c;
    x1= (-b-sqrt(D))/(2*a);
    x2=(-b+sqrt(D))/(2*a);
    cout<<"\n x1 = "<< x1<<"\n x2 = "<<x2;
    return 0;
}
```

***Chú ý***

- Dấu chấm phẩy (;) dùng để ngăn cách các khai báo và các câu lệnh.
- Thư viện math.h cho phép sử dụng các hàm toán học.

***b)*** Nhấn phím **F9** để dịch và sửa lỗi cú pháp (nếu có).

***c)*** Nhập các giá trị 1; -3 và 2. Quan sát kết quả hiển thị trên màn hình ($x1 = 1.00$ $x2 = 2.00$).

***d)*** Nhấn tổ hợp phím **F9** rồi nhập các giá trị 1 0 -2.
Quan sát kết quả hiển thị trên màn hình ($x1 = -1.41$ $x2 = 1.41$).

***e)*** Sửa lại chương trình trên sao cho không dùng biến trung gian *D*. Thực hiện chương trình đã sửa với các bộ dữ liệu trên.

***f)*** Sửa lại chương trình nhận được ở mục c bằng cách thay đổi công thức tính $x2$ (có hai cách để tính $x2$).

***g)*** Thực hiện chương trình đã sửa với bộ dữ liệu 1; -5; 6. Quan sát kết quả trên màn hình ($x1 = 2$ $x2 = 3$).

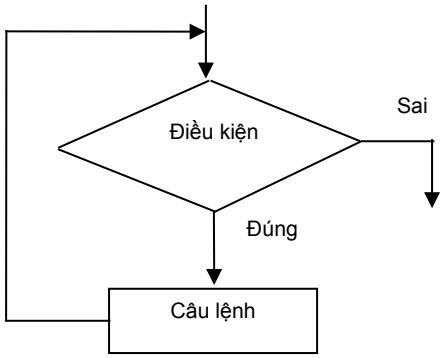
***h)*** Thực hiện chương trình với bộ dữ liệu 1; 1 ; 1 và quan sát kết quả trên màn hình.

### CÂU HỎI VÀ BÀI TẬP

1. *Nhập vào hai số a, b từ bàn phím. Tính tổng, hiệu, tích, thương của hai số đó.*
2. *Nhập vào ba số a, b, c. Trong đó $a \neq 0$ là giá trị của $ax^2+bx+c=0$. Tính nghiệm của phương trình bậc 2 trên.*
3. *Tính $s= (a+b)^2 + b+c$. Trong đó, a, b, c là các giá trị được nhập từ bàn phím.*

# Chương III
# Cấu trúc rẽ nhánh và lặp

- Cấu trúc rẽ nhánh và lặp trong lập trình;
- Các câu lệnh thực hiện rẽ nhánh và lặp của C++.



# §9. CẤU TRÚC RẼ NHÁNH

## 1. Rẽ nhánh

Thường ngày, có rất nhiều việc chỉ được thực hiện khi một điều kiện cụ thể nào đó được thoả mãn.

Ví dụ, Châu và Ngọc thường cùng nhau chuẩn bị các bài thực hành môn Tin học.

Một lần Châu hẹn với Ngọc: "*Chiều mai nếu trời không mưa thì Châu sẽ đến nhà Ngọc*".

Một lần khác, Ngọc nói với Châu: "*Chiều mai nếu trời không mưa thì Ngọc sẽ đến nhà Châu, nếu mưa thì sẽ gọi điện cho Châu để trao đổi*".

Câu nói của Châu cho ta biết một việc làm cụ thể (Châu đến nhà Ngọc) sẽ được thực hiện nếu một điều kiện cụ thể (trời không mưa) thoả mãn. Ngoài ra không đề cập đến việc gì sẽ xảy ra nếu điều kiện đó không thoả mãn (trời mưa).

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề thiếu:

*Nếu... thì…*

Câu nói của Ngọc khẳng định một trong hai việc cụ thể (Ngọc đến nhà Châu hay Ngọc gọi điện cho Châu) chắc chắn sẽ xảy ra. Tuy nhiên, việc nào trong hai việc sẽ được thực hiện thì tuỳ thuộc vào điều kiện cụ thể (trời không mưa) thoả mãn hay không.

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề đủ:

*Nếu… thì…, nếu không thì…*

Từ đó có thể thấy, trong nhiều thuật toán, các thao tác tiếp theo sẽ phụ thuộc vào kết quả nhận được từ các bước trước đó.

Cấu trúc dùng để mô tả các mệnh đề có dạng như trên được gọi là cấu trúc rẽ nhánh.

Ví dụ, để giải phương trình bậc hai:

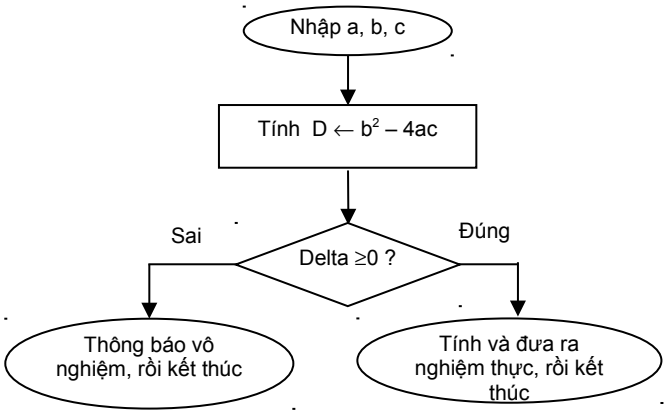
$$ax^2 + bx + c = 0, (a \neq 0)$$

trước tiên ta tính biệt số delta $D = b^2 - 4ac$.

Nếu $D$ không âm, ta sẽ đưa ra các nghiệm. Trong trường hợp ngược lại, ta phải thông báo là phương trình vô nghiệm.

Như vậy, sau khi tính $D$, tuỳ thuộc vào giá trị của $D$, một trong hai thao tác sẽ được thực hiện (h. 4).

Mọi ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc rẽ nhánh.




*Hình 4. Sơ đồ thể hiện cấu trúc rẽ nhánh*

## 2. Câu lệnh if

Để mô tả cấu trúc rẽ nhánh, CodeBlocks dùng câu lệnh ***if***. Tương ứng với hai dạng mệnh đề thiếu và đủ nói ở trên, Pascal có hai dạng câu lệnh *if*:

***a) Dạng thiếu***

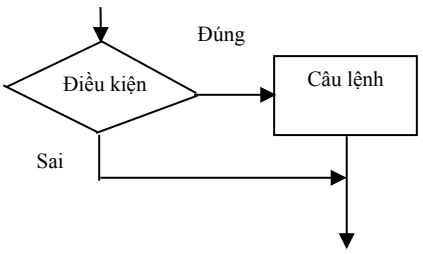
```
if (<Điều kiện>)
{
 Các_lệnh;
}
```

***b) Dạng đủ***

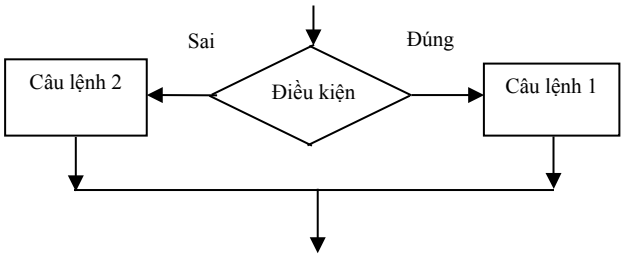
```
if (<Điều kiện>)
    { Câu lệnh 1;
    }
else
    { Câu lệnh 2;
    }
```

trong đó:

- *Điều kiện:* Biểu thức quan hệ hoặc lôgic.
- *Câu lệnh, câu lệnh 1, câu lệnh 2* là một câu lệnh của C++.




*Hình 5*




*Hình 6*

Ở dạng thiếu: *điều kiện* sẽ được tính và kiểm tra. Nếu *điều kiện* đúng (có giá trị *true*) thì *câu lệnh* sẽ được thực hiện, ngược lại thì *câu lệnh* sẽ bị bỏ qua (h. 5).

Ở dạng đủ: *điều kiện* cũng được tính và kiểm tra. Nếu *điều kiện* đúng thì *câu lệnh 1* sẽ được thực hiện, ngược lại thì *câu lệnh 2* sẽ được thực hiện (h. 6).

## 4. Một số ví dụ

***Ví dụ 1.*** Tìm nghiệm thực của phương trình bậc hai:

$$ax^2 + bx + c = 0 \text{ , với } a \neq 0.$$

***Input:*** Các hệ số *a*, *b*, *c* nhập từ bàn phím.

***Output:*** Đưa ra màn hình các nghiệm thực hoặc thông báo "*Phuong trinh vo nghiem*".

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
```

```
 float a,b,c,x1,x2;
 cout << "Nhap a:";
 cin>>a;
 cout << "\n Nhap b:";
 cin>>b;
 cout << "\n Nhap c:";
 cin>>c;
 float D=b*b-4*a*c;
 if (D<0)
 {
 cout << "\n Phuong trinh vo nghiem";
}
else
{
 x1= (-b-sqrt(D))/(2*a);
 x2=(-b+sqrt(D))/(2*a);
 cout<<"\n x1 = "<< x1<<"\n x2 = "<<x2;
}
 return 0;
}
```

# §10. CẤU TRÚC LẶP

## 1. Lặp

Với $a$ là số nguyên và $a > 2$, xét các bài toán sau đây:

***Bài toán 1.*** Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + ... + \frac{1}{a+100}.$$

***Bài toán 2.*** Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + ... + \frac{1}{a+N} + ...$$

cho đến khi $\frac{1}{a+N} < 0{,}0001$.

Với cả hai bài toán, dễ thấy cách để tính tổng $S$ có nhiều điểm tương tự:

- Xuất phát, $S$ được gán giá trị $\frac{1}{a}$;
- Tiếp theo, cộng vào tổng $S$ một giá trị $\frac{1}{a+N}$ với $N$ = 1, 2, 3, 4, 5,… Việc cộng này được lặp lại một số lần.

Đối với bài toán 1, số lần lặp là 100 và việc cộng vào tổng $S$ sẽ kết thúc khi đã thực hiện việc cộng 100 lần.

Đối với bài toán 2, số lần lặp chưa biết trước nhưng việc cộng vào tổng $S$ sẽ kết thúc khi điều kiện $\frac{1}{a+N} < 0{,}0001$ được thoả mãn.

Nói chung, trong một số thuật toán có những thao tác phải thực hiện lặp đi lặp lại một số lần. Một trong các đặc trưng của máy tính là có khả năng thực hiện hiệu quả các thao tác lặp. Cấu trúc lặp mô tả thao tác lặp và được phân biệt hai loại là *lặp với số lần biết trước* và *lặp với số lần chưa biết trước*.

Các ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc điều khiển lặp.

## 2. Lặp có số lần lặp biết trước và câu lệnh for

Có hai thuật toán *Tong_1a* và *Tong_1b* để giải bài toán 1 như sau:

***Thuật toán Tong_1a***

*Bước 1.* $S \leftarrow 1/a$; $N \leftarrow 0$; {Khởi tạo $S$ và $N$}

*Bước 2.* $N \leftarrow N + 1$;

*Bước 3.* Nếu $N > 100$ thì chuyển đến bước 5;

*Bước 4.* $S \leftarrow S + 1/(a + N)$ rồi quay lại bước 2;

*Bước 5.* Đưa $S$ ra màn hình, rồi kết thúc.

***Thuật toán Tong_1b***

*Bước 1.* $S \leftarrow 1/a$; $N \leftarrow 101$; {Khởi tạo $S$ và $N$}

*Bước 2.* $N \leftarrow N - 1$;

*Bước 3.* Nếu $N < 1$ thì chuyển đến bước 5;

*Bước 4.* $S \leftarrow S + 1/(a + N)$ rồi quay lại bước 2;

*Bước 5.* Đưa $S$ ra màn hình rồi kết thúc.

Lưu ý, số lần lặp của cả hai thuật toán trên là biết trước và như nhau (100 lần).

Trong thuật toán *Tong_1a*, giá trị $N$ khi bắt đầu tham gia vòng lặp là 1 và sau mỗi lần lặp $N$ tăng lên 1 cho đến khi $N > 100$ ($N = 101$) thì kết thúc lặp (thực hiện đủ 100 lần). Trong thuật toán *Tong_1b*, giá trị $N$ bắt đầu tham gia vòng lặp là 100 và sau mỗi lần lặp $N$ giảm đi 1 cho đến khi $N < 1$ ($N = 0$) thì kết thúc lặp (thực hiện đủ 100 lần). Ta nói cách lặp trong thuật toán *Tong_1a* là dạng tiến và trong thuật toán *Tong_1b* là dạng lùi.

Để mô tả cấu trúc lặp với số lần biết trước, C++ dùng câu lệnh ***for*** như sau:

```
for (<biểu thức khởi tạo>; <biểu thức giới hạn>; <biểu thức tăng giảm>)
{
    <Dãy lệnh>;
}
```

**Giải thích:** Thực hiện vòng lặp, với số vòng lặp từ biểu thức khởi tạo cho đến biểu thức giới hạn theo mức tăng là biểu thức tăng giảm.

**Lưu ý:** khi sử dụng vòng lặp for cần lưu ý các điểm sau đây:

- Các tham số trong vòng lặp for có thể khuyết một hoặc vài (thậm chí là tất cả) tham số. Tuy nhiên, dấu chấm phẩy là luôn bắt buộc. Số bước lặp của vòng lặp for sẽ được tính như sau:

**Biểu thức giới hạn – Biểu thức khởi tạo**

---

**Biểu thức tăng giảm**

- Nếu có nhiều lệnh chịu sự chi phối của for, thì chúng cần được đặt trong dấu khối lệnh.
- Ta có thể thực hiện việc khai báo biến trực tiếp bên trong dấu ngoặc đơn của vòng lặp for.

***Ví dụ 1.*** *Xuất ra các giá trị từ 1 đến n. Với n được nhập từ bàn phím.*

```
#include<iostream>
using namespace std;
int main()
{
  int n, i;
  cout<<"Nhap n:";
  cin>>n;
  for (i=0; i<=n;i++)
    {
      cout<<i<<endl;
    };
  return 0;
}
```

***Ví dụ 2.*** Chương trình sau thực hiện việc nhập từ bàn phím hai số nguyên dương *M* và *N* ( *M* < *N* ), tính và đưa ra màn hình tổng các số chia hết cho 3 hoặc 5 trong phạm vi từ *M* đến *N*.

```
#include <iostream>
using namespace std;
int main()
{
  int m,n,s=0;
  cout << "Nhap m";
  cin>>m;
  cout << "\nNhap n";
  cin>>n;
  for(int i=m;i<=n;i++)
  {
    if ((i%3==0) || (i%5==0))
    {
      s+=i;
    }
  }
  cout<<"\n Tong S= "<<s;
  return 0;
}
```

## 3. Lặp với số lần chưa biết trước và câu lệnh while

Có thể xây dựng thuật toán *Tong_2* như sau để giải bài toán 2.

***Thuật toán Tong_2***

*Bước 1.* $S \leftarrow 1/a$; $N \leftarrow 0$; {Khởi tạo *S* và *N*}

*Bước 2.* Nếu $1/(a + N) < 0{,}0001$ thì chuyển đến bước 5;

*Bước 3.* $N \leftarrow N + 1$;

*Bước 4.* $S \leftarrow S + 1/(a + N)$ rồi quay lại bước 2.

*Bước 5.* Đưa S ra màn hình, rồi kết thúc

Như vậy, việc lặp với số lần chưa biết trước sẽ chỉ kết thúc khi một điều kiện cho trước được thoả mãn.

### 3.1 Câu lệnh While:

**Cú pháp:**

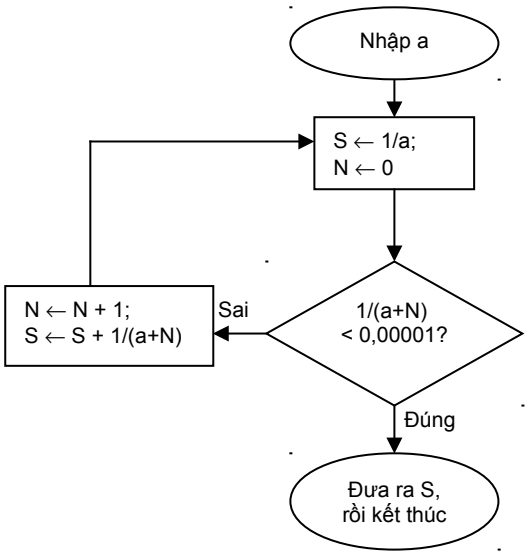
***while (<Điều kiện>)***
***{***
***<Dãy lệnh>;***
***}***

***Giải thích:*** *Nếu điều kiện đúng, các lệnh bên trong vòng lặp sẽ được thực hiện cho đến khi nó nhận giá trị sai.*

***Lưu ý:*** *khi sử dụng vòng lặp while cần lưu ý các điểm sau đây:*

- *Vòng lặp phải có tính dừng. Nghĩa là điều kiện phải có trường hợp sai. Trong một số tình huống, người ta vẫn sử dụng vòng lặp vô hạn, nhưng cần có cơ chế để thoát khỏi vòng lặp khi cần thiết.*
- *Nếu có nhiều lệnh chịu sự chi phối của while, thì chúng cần được đặt trong dấu khối lệnh.*

***Ví dụ 1.*** Sau đây là chương trình cài đặt thuật toán *Tong_2*.




*Hình 8. Sơ đồ khối của thuật toán Tong_2*

```
#include <iostream>
using namespace std;
int main()
{
  int n;
  float a,s;
  std::cout << "Nhap a";
  cin>>a;
  s=1/a; n=0;
  while (!(1/(a+n)<0.0001))
  {
    n++;
    s+=1/(a+n);
  }
  std::cout<<"\n Tong S= "<<s;
  return 0;
}
```

***Ví dụ 2.*** Tìm ước chung lớn nhất (UCLN) của hai số nguyên dương *M* và *N*.

Có nhiều thuật toán khác nhau tìm UCLN của *M* và *N*. Sau đây là một thuật toán tìm UCLN.

***Thuật toán***

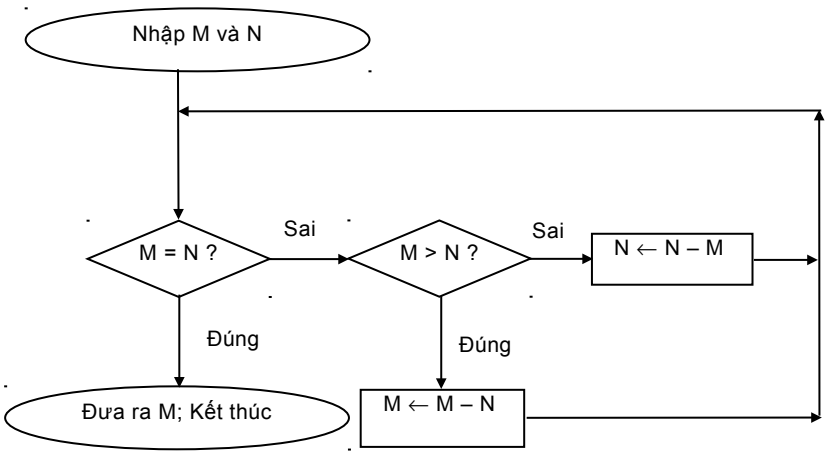
Bước 1. Nhập *M*, *N*;

Bước 2. Nếu *M* = *N* thì lấy giá trị chung này làm UCLN rồi chuyển đến bước 5;

Bước 3. Nếu *M* > *N* thì *M* ←*M* - *N* rồi quay lại bước 2;

*Bước 4. N* ← *N* - *M* rồi quay lại bước 2;

Bước 5. Đưa ra kết quả UCLN rồi kết thúc.




*Hình 9. Sơ đồ khối của thuật toán tìm ước chung lớn nhất*

Chương trình sau thể hiện thuật toán tìm ước chung lớn nhất.

```
#include <iostream>
using namespace std;
int main()
{
  int n,m;
  cout<<"nhap n, m: ";
  cin>>n>>m;
  while (n != m)
    {
      if (n>m) n=n-m; else m=m-n;
    }
  std::cout<<"\n UCLN= "<<n;
  return 0;
}
```

### *3.2. Vòng lặp do-while*

**Cú pháp:**

***do***

***{***

***<Dãy lệnh>;***

***}while (<điều kiện>);***

Ví dụ:

```
#include<iostream>
using namespace std;
int main(){
  int n;
  cout<<”Nhap n:”;
  cin>>n;
  do {
     cout<<n<<endl;
     n--;
  }while(n>0);
  return 0;
}
```

**Lưu ý:** khi sử dụng vòng lặp do-while cần lưu ý các điểm sau đây:

- Vòng lặp phải có tính dừng. Nghĩa là điều kiện phải có trường hợp sai. Trong một số tình huống, người ta vẫn sử dụng vòng lặp vô hạn, nhưng cần có cơ chế để thoát khỏi vòng lặp khi cần thiết.
- Nếu có nhiều lệnh chịu sự chi phối của do-while, thì chúng cần được đặt trong dấu khối lệnh.

***Chú ý***

Các câu lệnh trong vòng lặp thường được lặp lại nhiều lần, vì vậy để tăng hiệu quả của chương trình thì những thao tác không cần lặp lại nên đưa ra ngoài vòng lặp.

**TÓM TẮT**

- Các ngôn ngữ lập trình đều có câu lệnh thể hiện cấu trúc rẽ nhánh và cấu trúc lặp.
- Câu lệnh rẽ nhánh có hai dạng:

  a) Dạng thiếu;

  b) Dạng đủ.
- Có thể gộp dãy câu lệnh thành câu lệnh ghép.
- Các câu lệnh mô tả cấu trúc lặp:

  a) Lặp với số lần biết trước;

  b) Lặp với số lần không biết trước.

***Định lí Bohn Jacopini*** (Bon Ja-co-pi-ni): *Mọi quá trình tính toán đều có thể thực hiện dựa trên ba cấu trúc cơ bản là cấu trúc tuần tự, cấu trúc rẽ nhánh và cấu trúc lặp.*

# BÀI TẬP VÀ THỰC HÀNH 2

## 1. Mục đích, yêu cầu

- *Xây dựng chương trình có sử dụng cấu trúc rẽ nhánh;*
- *Làm quen với việc hiệu chỉnh chương trình.*

## 2. Nội dung

***Bài toán.*** *Bộ số Pi-ta-go*

Biết rằng bộ ba số nguyên dương *a*, *b*, *c* được gọi là bộ số Pi-ta-go nếu tổng các bình phương của hai số bằng bình phương của số còn lại. Viết chương trình nhập từ bàn phím ba số nguyên dương *a*, *b*, *c* và kiểm tra xem chúng có là bộ số Pi-ta-go hay không.

***Ý tưởng***: Kiểm tra xem có đẳng thức nào trong ba đẳng thức sau đây được thực hiện hay không:

$$a^2 = b^2 + c^2$$
$$b^2 = a^2 + c^2$$
$$c^2 = a^2 + b^2.$$

Những công việc cần thực hiện:

***a)*** Gõ chương trình sau:

```
#include <iostream>
using namespace std;
int main()
{
  int a,b,c;
  long long a2,b2,c2;
  cout<<"nhap a,b,c";
  cin>>a>>b>>c;
  a2 =a;
  b2=b;
  c2=c;
  a2*=a;
  b2*=b;
  c2*=c;
  if((a2==b2+c2) || (b2==a2+c2)||(c2==a2+b2))
    cout<<"\nBa so dang nhap la bo so Pi-ta-go";
  else
    cout<<"\nBa so dang nhap khong la bo so Pi-ta-go";
  return 0;
}
```

***b)*** Nhập các giá trị $a = 3$, $b = 4$, $c = 5$.

***c)*** Lặp lại các bước trên với bộ dữ liệu $a = 700$, $b = 1000$, $c = 800$.

## CÂU HỎI VÀ BÀI TẬP

**1.** Hãy cho biết sự giống và khác nhau của hai dạng câu lệnh **if**.

**2.** Có thể dùng câu lệnh **while-do** để thay cho câu lệnh **for-do** được không? Nếu được, hãy thực hiện điều đó với chương trình *Tong_*1*a*.

**3.** Viết câu lệnh **if** tính:

a) $z = \begin{cases} x^2 + y^2 & \text{nếu } x^2 + y^2 \le 1. \\ x + y & \text{nếu } x^2 + y^2 > 1 \text{ và } y \ge x. \\ 0,5 & \text{nếu } x^2 + y^2 > 1 \text{ và } y < x. \end{cases}$

b) $z = \begin{cases} |x| + |y| & \text{nếu điểm } (x;y) \text{ thuộc hình tròn bán kính } r\ (r > 0), \text{ tâm } (a; b). \\ x + y & \text{trong trường hợp còn lại.} \end{cases}$

**4.** Lập trình tính:

a) $Y = \sum_{n=1}^{50} \frac{n}{n+1}$;

b) $e(n) = 1 + \frac{1}{1!} + \frac{1}{2!} + ... + \frac{1}{n!} + ...$, với $n$ lần lượt bằng 3, 4,... cho đến khi $\frac{1}{n!} < 2 \times 10^{-6}$. Đưa các giá trị $e(n)$ ra màn hình.

**5.** Lập trình để giải bài toán cổ sau:

Vừa gà vừa chó.
Bó lại cho tròn.
Ba mươi sáu con,
Một trăm chân chẵn.
Hỏi bao nhiêu con mỗi loại?

**6.** Nhập từ bàn phím tuổi của cha và con (tuổi của cha hơn tuổi con ít nhất là 25). Đưa ra màn hình bao nhiêu năm nữa thì tuổi cha gấp đôi tuổi con.

**7.** Một người gửi tiết kiệm không kì hạn với số tiền A đồng với lãi suất 0,2% mỗi tháng. Hỏi sau $t$ tháng, người đó rút tiền thì sẽ nhận được số tiền là bao nhiêu. Biết rằng tiền gửi tiết kiệm không kì hạn không được tính lãi kép.

Bài đọc thêm 3

# CÂU LỆNH RẼ NHÁNH VÀ LẶP TRONG PASCAL

## 1. Câu lệnh case-of

Xét bài toán: Lập chương trình nhập từ bàn phím tháng và năm rồi tính và đưa ra màn hình số ngày của tháng.

Câu lệnh *if-then* giải quyết việc rẽ nhánh tuỳ theo một trong hai khả năng. Bài toán trên liên quan tới việc chọn lựa một trong nhiều khả năng (rẽ nhiều nhánh). Khi đó, nếu dùng câu lệnh *if-then* giải quyết sẽ làm chương trình rườm rà. Câu lệnh chọn *case-of* trong TP cho phép rẽ nhiều nhánh một cách thuận lợi.

Câu lệnh *case-of* có dạng:

**case** <*biểu thức nguyên hoặc kí tự* > **of**

<*danh sách 1*>**:** <*câu lệnh 1*>**;**

<*danh sách 2*>: <*câu lệnh 2*>**;**

. . . . . .

<*danh sách N*>**:** <*câu lệnh N*>

[ **else** <*câu lệnh N + 1*>]

**end;**

trong đó:

- *Danh sách i* (*i* = 1, 2,..., *N*) là một hoặc nhiều giá trị của biểu thức nêu sau ***case***, nếu có nhiều giá trị thì các giá trị viết cách nhau bởi dấu phẩy.
- Khi thực hiện, biểu thức sau từ khoá ***case*** sẽ được tính, giá trị nhận được sẽ lần lượt được kiểm tra xem nằm trong danh sách nào. Câu lệnh tương ứng với danh sách đầu tiên tìm thấy sẽ được thực hiện. Tiếp theo, thực hiện câu lệnh sau câu lệnh ***case-of***.
- Nếu giá trị tính được không xuất hiện ở bất kì danh sách nào thì *câu lệnh N + 1* sau ***else*** (nếu có) sẽ được thực hiện.

Chương trình để giải bài toán vừa nêu như sau:

```
program Vi_du_case;
uses crt;
var T, N, SN: integer;
begin
clrscr;
write('Cho biet thang va nam: ');
readln(T,N);
case T of
4,6,9,11: SN:= 30;
       2: if (N mod 400=0) or ((N mod 100<>0) and (N mod 4=0))
          then SN:= 29 else SN:= 28
else SN:= 31
 end;
 writeln('Thang ', T,' nam ', N, ' co ', SN,' ngay.');
 readln
end.
```

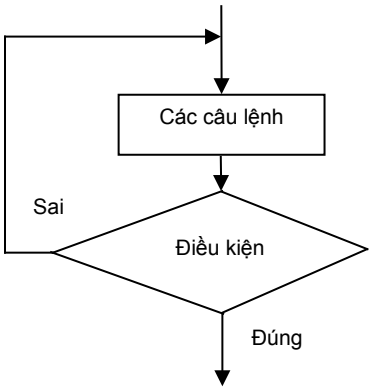
## 2. Câu lệnh lặp repeat-until

Câu lệnh này dùng để tổ chức lặp với số lần lặp không biết trước. Khác với câu lệnh *while-do*, điều kiện kết thúc lặp trong câu lệnh *repeat-until* được kiểm tra sau và có dạng:

```
repeat
          <dãy câu lệnh >;
until <điều kiện>;
```

trong đó *điều kiện* là biểu thức quan hệ hoặc lôgic.

Việc thực hiện lệnh *repeat-until* được thể hiện trên sơ đồ sau.




*Sơ đồ thực hiện lệnh* **repeat-until**

***Ví dụ 1***

Xét đoạn chương trình:

```
repeat
  write('HAY NHAP SO NGUYEN N: ');
  readln(N)
until (N>0) and (N <= 100);
```

Đoạn chương trình này dùng để nhập số nguyên N thoả mãn điều kiện $0 < N \le 100$. Nếu giá trị nguyên đưa vào nằm ngoài khoảng này thì chương trình sẽ yêu cầu nhập lại cho đến khi giá trị nhập vào thoả mãn điều kiện.

***Ví dụ 2***

Tính gần đúng $\sqrt{A}$ ($A > 0$) chính xác đến chữ số thập phân thứ năm theo phương pháp lặp Niu-tơn.

Theo phương pháp này, ta cần tính dãy $X_0$, $X_1$, $X_2$,..., theo công thức:

$X_0 = 1$

…

$$X_{n+1} = \frac{1}{2}(X_n + \frac{A}{X_n}),\ n = 0, 1, 2, ... \qquad (*)$$

Quá trình lặp kết thúc khi $|X_{n+1} - X_n| < 0{,}00001$. Giá trị của $X_{n+1}$ được lấy làm giá trị gần đúng của $\sqrt{A}$.

***Input:*** Giá trị thực *A*, nhập vào từ bàn phím.

***Output:*** Đưa ra màn hình giá trị gần đúng của $\sqrt{A}$.

Ta chỉ cần dùng hai biến thực trung gian *XNEW* và *XOLD* để chứa tương ứng các giá trị $X_{n+1}$ và $X_n$.

Thuật toán được mô tả như sau:

*Bước 1*. Nhập A;

*Bước 2*. *XNEW* ← *1* {giá trị $X_0$};

*Bước 3*. *XOLD* ← *XNEW* {chuyển giá trị mới thành giá trị cũ};

*Bước 4*. $XNEW \leftarrow \frac{1}{2}(XOLD + \frac{A}{XOLD})$ {Tính lại giá trị mới theo *};

*Bước 5*. Nếu $|XNEW - XOLD| > 0{,}00001$ thì quay về bước 3;

*Bước 6*. Đưa giá trị *XNEW* ra màn hình, rồi kết thúc.

Chương trình thể hiện thuật toán trên như sau:

```
program CAN_BAC_2;
uses crt;
var A, XNEW, XOLD: real;
begin
  clrscr;
  repeat
```

```
    write(' A = '); readln(A);{Buoc 1}
  until A > 0;
  XNEW:= 1;  {Buoc 2}
  repeat
    XOLD:= XNEW;               {Buoc 3}
    XNEW:= (XOLD + A/XOLD)/2;  {Buoc 4}
  until ABS(XNEW - XOLD) < 0.00001;  {Buoc 5}
  writeln('Can bac 2 cua ',A:10:5,' la ',XNEW:10:5); {Buoc 6}
  readln
end.
```

Bài đọc thêm 4

# CÁC CÂU LỆNH NHẢY

**1. Câu lệnh break**

Lệnh break dùng để thoát ra khỏi vòng lặp. Thông thường, khi ta sử dụng các vòng lặp không x|c định được số lần lặp, để tránh lặp vô hạn, người ta thường sử dụng câu lệnh break để thoát khỏi vòng lặp.

**Ví dụ:**

```
#include<iostream>
using namespace std;
int main()
{
  int n = 2;
  for(;;){
   cout<<n<<endl;
   n--;
   if (n<0) break;
  }
  return 0;
}
```

**KẾT QUẢ**

2

1

0

**Giải thích:** Giá trị khởi tạo của n = 2. Vòng lặp for sẽ tiến hành in giá trị của n, sau đó, giảm giá trị của n đi 1. C}u lệnh điều kiện bên trong for sẽ kiểm tra điều kiện của n. Nếu n<0, thì lệnh break sẽ được thực hiện và vòng lặp bị hủy.

**2. Câu lệnh continue**

Câu lệnh continue thường được dùng trong các vòng lặp. Khi lệnh continue được gọi, bước lặp hiện tại sẽ được bỏ qua, và tiến hành bước lặp tiếp theo.

**Ví dụ:**

```
#include<iostream>
using namespace std;
int main()
{
 for(int i=0;i<10;i++){
  if (i%2!=0)
    continue;
  cout<<i<<endl;
 }
 return 0;
}
```

**Kết quả:**

0

2

4

6

8

**Giải thích:** Vòng lặp for sẽ thực thi các lệnh bên trong nó. Biến i chạy từ 0 đến 9, kiểm tra điều kiện i có phải là số chẵn hay không. Nếu i là số lẻ, thì câu lệnh continue sẽ được thực hiện và bước lặp hiện tại sẽ bị bỏ qua. Nếu i là số chẵn, thì lệnh continue sẽ không được gọi và bước lặp hiện tại vẫn được thực hiện. Do đó, lệnh cout chỉ được thực hiện trong trường hợp biến i là chẵn. Như vậy, mỗi khi giá trị i là chẵn, nó sẽ in kết quả. Nếu giá trị của i là lẻ, thì kết quả sẽ không được in ra.

### 3. Lệnh goto

Lệnh goto cho phép tạo ra một bước nhảy đến một nhãn được ấn định sẵn. Tên nhãn sẽ được đặt như sau tên_nhãn: và lệnh goto sẽ nhảy đến tên nhãn. Một lời khuyên cho chúng ta là nên hạn chế tối đa việc sử dụng lệnh goto. Bởi vì lệnh goto thường làm phá vỡ cấu trúc của lập trình hiện đại. Nhiều ngôn ngữ lập trình họ nhà C ra đời sau C++ như Java đã tuyệt giao hoàn toàn với câu lệnh goto này.

**Ví dụ:**

```
#include <iostream>
using namespace std;
int main()
{
   int n = 5;
   loop://Tên nhãn
   cout<<n<<endl;
   n--;
   if(n>0) goto loop;
   return 0;
}
```

**Kết quả:**

5

4

3

2

1

Giải thích: Giá trị khởi tạo của biến n là 5. Nhãn được đặt tên là loop. Nhãn có thể hiểu như một vị trí được đ|nh dấu (bookmark). Chương trình tiến hành in giá trị của biến n. Sau đó, giảm giá trị của n đi. Câu lệnh điều kiện, kiểm tra giá trị của biểu thức n>0, nếu đúng thi lệnh goto được gọi và nó sẽ được chuyển đến vị trí đ~ được đ|nh dấu là loop. Chương trình lại thực thi thêm lần nữa kể từ vị trí loop đó. Nếu n<=0, lệnh goto không được gọi. Chương trình kết thúc. Việc sử dụng các câu lệnh lặp, hoàn toàn có thể thay thế cho lệnh goto. Hãy luôn ghi nhớ: CHỈ NÊN sử dụng goto khi thực sự cần thiết.

### 4. Lệnh exit

Lệnh exit dùng để thoát khỏi chương trình và trả về một mã được chỉ định. Mã chỉ định này tùy thuộc vào hệ điều hành, nó có thể sử dụng trong chương trình theo quy ước như sau: nếu chương trình kết thúc bình thường, thì mã chương trình là 0; nếu có một sự cố không mong muốn xảy ra, thì mã chương trình là một giá trị khác 0.

**void exit(int mã_chỉ_định);**

Nếu tham số mã_chỉ_định không được cấp vào, tức là exit (không có dấu ngoặc đơn), thì nó sẽ tiến hành theo mặc định – tức giá trị 0. Hàm exit nằm trong thư viện stdlib.h. Đây là một hàm

tương đối cũ nằm trong thư viện .h. Ta chỉ có thể sử dụng lệnh exit nếu khai báo thư viện stdlib.h mà không có thư viện tương ứng là stdlib.

# BÀI ĐỌC THÊM 5
# CẤU TRÚC SWITCH

**Cú pháp:**

```
switch(biểu_thức)
{
  case hằng_1:
    nhóm_các_lệnh;
    break;
  case hằng_2:
    nhóm_các_lệnh;
    break;
  …
  default:
    nhóm_các_lệnh;
}
```

Giải thích: kiểm tra giá trị của biểu thức, nếu giá trị của biểu thức rơi v{o danh sách hằng, thì nó sẽ thực hiện các lệnh trong từng trường hợp case tương ứng (nếu là hằng_1 – các lệnh trong trường hợp case hằng_1, ….). Nếu biểu thức không thuộc vào danh sách hằng, thì nó sẽ thực hiện lệnh trong trường hợp default.

Chương trình

```
#include <iostream>
using namespace std;
int main()
{
   char n;
   cout<<”Ban la nam hay nu b/g:”<<endl;
   cin>>n;
   switch(n)
   {
      case ‘b’:
         cout<<”Nam”;
         break;
      case ‘g’:
         cout<<”Nu”;
         break;
      default:
         cout<<”Khong xac dinh”;
   }
   return 0;
}
```

**KẾT QUẢ:**

Ban la nam hay nu b/g:b

Nam

**Giải thích:** chương trình buộc người dùng nhập vào một kí tự (b – boy) hay (g – girl). Nếu người dùng nhập vào một kí tự khác, chương trình vẫn tính đến trường hợp này. Kí tự mà người dùng nhập vào được lưu trong biến n.

Câu lệnh switch sẽ kiểm tra biến nhập vào đó có nằm trong danh sách hằng hay không (danh sách hằng ở đây là 'b' và 'g'). Nếu có, thì tương ứng với 'b' nó sẽ thực hiện trường hợp case 'b', nếu là 'g' nó sẽ thực hiện trường hợp case 'g'. Lệnh break trong mỗi trường hợp có tác dụng là thoát ra khỏi câu lệnh lựa chọn (cũng mang tính lặp), nếu không có lệnh break, tất cả các trường hợp đều được xét duyệt và nếu rơi vào trường hợp nào thì các lệnh tương ứng sẽ được thực thi, đồng thời lệnh ở trường hợp bên dưới nó cũng được thực thi. Trong trường hợp, kí tự nhập vào không tương ứng trong danh sách hằng, nó sẽ thực thi trường hợp default. Vì default là trường hợp cuối cùng, nên nó không cần lệnh break.

**Chú ý:**

Lệnh switch chỉ được lựa chọn để sử dụng khi cần kiểm tra giá trị của một biểu thức có tương ứng với một tập các hằng số nào đó hay không (sự tương ứng ở đây có thể là thuộc hoặc không thuộc tương ứng với khái niệm trong tập hợp). Các hằng_1, hằng_2,… có thể là một vùng liên tục, hoặc gián đoạn (như các số từ 0..1, 'a'..'d',…). Nhưng nhất thiết các giá trị tương ứng với các trường hợp case phải là hằng số (hoặc khoảng hằng).

**Ví dụ:**

```
int a = 1;
switch(a>0)
{
case true:
  cout<<"Duong";
  break;
case false:
  cout<<"Am";
  break;
default:
  cout<<"Khong";
}
```

**Kết quả**

Liên tục

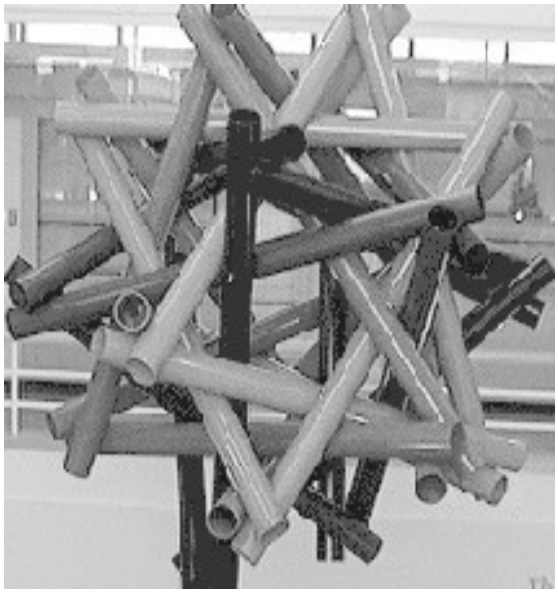
```
int a = 1;
switch(a)
{
case 1: case 2: case 3:
  cout<<"Xuan";
  break;
case 4: case 5: case 6:
  cout<<"Ha";
  break;
case 7: case 8: case 9:
  cout<<"Thu";
  break;
case 10: case 11: case 12:
  cout<<"Dong";
  break;
default:
  cout<<"Khong phai thang cua nam";
}
```

**Kết quả:** Rời rạc

# Chương IV. KIỂU DỮ LIỆU CÓ CẤU TRÚC

Ngoài các kiểu dữ liệu chuẩn, các ngôn ngữ lập trình còn có những kiểu dữ liệu được xây dựng từ những kiểu đã có.

- Kiểu mảng;
- Kiểu xâu;
- Kiểu bản ghi.



Dữ liệu là thông tin xử lí bằng máy tính gồm dữ liệu vào, kết quả trung gian và kết quả ra. Dữ liệu có thể thuộc kiểu chuẩn như đã biết ở chương II. Các kiểu dữ liệu chuẩn nói chung không đủ để biểu diễn dữ liệu của các bài toán trong thế giới thực, thường có cấu trúc phức tạp, gồm nhiều thành phần liên kết với nhau theo một cách thức nào đó.

Ví dụ, cho hai véctơ thực $a = (a_1, a_2,..., a_n)$ và $b = (b_1, b_2,..., b_n)$, cần tính véctơ $c = (c_1, c_2,..., c_n)$ là tổng của $a$ và $b$ ($c_i = a_i + b_i$, $1 \le i \le n$). Dữ liệu vào của bài toán là hai dãy, mỗi dãy gồm $n$ số thực, tương ứng giá trị các thành phần của hai véctơ $a$ và $b$, dữ liệu ra là dãy $n$ giá trị thực $c_i = a_i + b_i$ ($1 \le i \le n$). Như vậy, các dữ liệu của bài toán trên có cấu trúc là dãy hữu hạn các phần tử có cùng kiểu dữ liệu chuẩn là kiểu số thực. Để mô tả dữ liệu có cấu trúc phức tạp như vậy, các ngôn ngữ lập trình đều có các quy tắc, cách thức cho phép người lập trình có thể xây dựng những kiểu dữ liệu phức tạp từ những kiểu đã có. Kiểu dữ liệu được xây dựng bằng cách như vậy gọi là kiểu dữ liệu có cấu trúc. Chẳng hạn, kiểu mảng được giới thiệu dưới đây là một kiểu dữ liệu có cấu trúc thông dụng, có thể dùng để mô tả dữ liệu bài toán tính tổng hai véctơ nói trên.

Mỗi ngôn ngữ lập trình luôn có tập kiểu dữ liệu chuẩn, các quy tắc, cách thức để xây dựng các kiểu dữ liệu cấu trúc từ kiểu dữ liệu chuẩn. Theo các quy tắc và cách thức này người lập trình xác định tên kiểu, cấu trúc, khuôn dạng của kiểu dữ liệu cần xây dựng từ các thành phần, mỗi thành phần có kiểu dữ liệu chuẩn hoặc kiểu dữ liệu đã được xác định trước đó, cách khai báo biến, tập giá trị và các phép toán có thể thực hiện trên tập giá trị đó.

Dưới đây xét ba kiểu dữ liệu có cấu trúc thông dụng nhất trong nhiều ngôn ngữ lập trình là kiểu mảng, kiểu bản ghi và kiểu xâu.

# §11. KIỂU MẢNG VÀ BIẾN CÓ CHỈ SỐ

**Chúng ta chỉ xét hai kiểu mảng thông dụng với nhiều ngôn ngữ lập trình là kiểu mảng một chiều và kiểu mảng hai chiều.**

## 1. Kiểu mảng một chiều

*Mảng một chiều là dãy hữu hạn các phần tử cùng kiểu. Mảng được đặt tên và mỗi phần tử của nó có một chỉ số. Để mô tả mảng một chiều cần xác định kiểu của các phần tử và cách đánh số các phần tử của nó.*

Để người lập trình có thể xây dựng và sử dụng kiểu mảng một chiều, các ngôn ngữ lập trình có quy tắc cách thức cho phép xác định:

- Tên kiểu mảng một chiều;
- Số lượng phần tử;
- Kiểu dữ liệu của phần tử;
- Cách khai báo biến;
- Cách tham chiếu đến phần tử.

Ví dụ, xét bài toán nhập vào nhiệt độ (trung bình) của mỗi ngày trong tuần. Tính và đưa ra màn hình nhiệt độ trung bình của tuần và số lượng ngày trong tuần có nhiệt độ cao hơn nhiệt độ trung bình của tuần.

Ta có thể dùng bảy biến thực để lưu trữ nhiệt độ của các ngày trong tuần. Chương trình giải bài toán có thể được viết bằng C++ như sau:

```
#include <iostream>

using namespace std;

int main()
{
  float t1,t2,t3,t4,t5,t6,t7;
  int dem;
  cout<<"nhap nhiet do cua 7 ngay: ";
  cin>>t1>>t2>>t3>>t4>>t5>>t6>>t7;
   float tb=(t1+t2+t3+t4+t5+t6+t7)/7;
   dem=0;
   if (t1>tb) dem++;
   if (t2>tb) dem++;
   if (t3>tb) dem++;
   if (t4>tb) dem++;
   if (t5>tb) dem++;
   if (t6>tb) dem++;
   if (t7>tb) dem++;
   cout<<"\nNhiet do trung binh tuan "<<tb;
   cout<<"\nSo ngay nhiet do cao hon trung binh: "<<dem;
   return 0;
}
```

Khi cần giải bài toán trên với *N* ngày (*N* khá lớn) thì cách làm tương tự không những đòi hỏi một khối lượng khai báo khá lớn, mà đoạn chương trình tính toán cũng khá dài.

Để giải quyết vấn đề đó, ta sử dụng kiểu dữ liệu mảng một chiều để mô tả dữ liệu. Chương trình giải bài toán tổng quát với *N* ngày trong C++ có thể như sau:

```
#include <iostream>
```

```
using namespace std;

int main()
{
  float nhietdo[365];
  int dem,i,n;
  float tong,tb;
  cout<<"nhap so ngay: ";
  cin>>n;
  for(i=1;i<=n;i++)
  {
    cout<<"nhap nhiet do ngay thu "<<i<<'\t';
    cin>>nhietdo[i];
    tong+=nhietdo[i];
  }
   dem=0;
   tb=tong/n;
   for(i=1;i<=n;i++)
  {
    if(nhietdo[i]>tb) dem++;
  }

   cout<<"\nNhiet do trung binh "<<n<<"ngay"<<tb;
   cout<<"\nSo ngay nhiet do cao hon trung binh: "<<dem;
   return 0;
}
```

**Trong chương trình này đã khai báo biến mảng một chiều (thông qua khai báo kiểu mảng) như sau:**

***a) Khai báo***

Tổng quát, khai báo biến mảng một chiều có dạng:

***<kiểu dữ liệu> <tên biến mảng>[kích thước tối đa];***

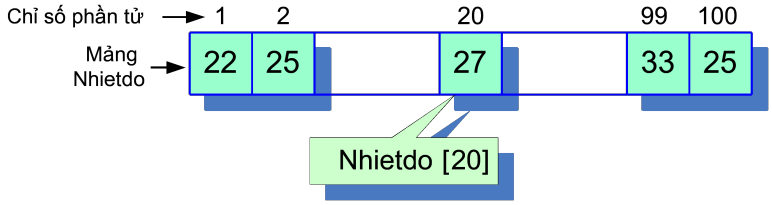
***Ví dụ .*** Các khai báo kiểu mảng một chiều sau đây là hợp lệ:

float nhietdo[365];

int a[100];

Tham chiếu tới phần tử của mảng một chiều được xác định bởi tên mảng cùng với chỉ số, được viết trong cặp ngoặc **[** và **]**.

Ví dụ, tham chiếu tới nhiệt độ của ngày thứ 20, trong chương trình trên, được viết là *Nhietdo*[20].




*Hình 10. Minh hoạ mảng một chiều*

***b) Một số ví dụ***

Ta xét chương trình có sử dụng mảng một chiều cài đặt một số thuật toán giải những bài toán tìm kiếm và sắp xếp.

***Ví dụ 1.*** *Tìm phần tử lớn nhất của dãy số nguyên*

***Input:*** Số nguyên dương $N$ ($N \le 250$) và dãy $N$ số nguyên dương $a_1$, $a_2$ ,..., $a_N$, mỗi số đều không vượt quá 500.

***Output:*** Chỉ số và giá trị của phần tử lớn nhất trong dãy số đã cho (nếu có nhiều phần tử lớn nhất chỉ cần đưa ra một trong số chúng).

*Bước 1.* Nhập $N$ và dãy $a_1,..., a_N$;

*Bước 2.* Max ơ $a_1$, $i$ ơ 2;

*Bước 3.* Nếu $i > N$ thì đưa ra giá trị Max rồi kết thúc;

*Bước 4.*

*Bước 4.1.* Nếu $a_i >$ Max thì Max ← $a_i$;

*Bước 4.2.* $i \leftarrow i + 1$ rồi quay lại bước 3;

*Hình 11. Thuật toán tìm phần tử lớn nhất của dãy số*

Chương trình dưới đây thực hiện việc duyệt tuần tự các phần tử để tìm ra số lớn nhất của dãy số nguyên.

```
#include <iostream>
#define max 100
using namespace std;
int main()
{
  int a[max];
  int ptln,n,i,csmax;
  cout<<"nhap so phan tu: ";
  cin>>n;
  for(i=1;i<=n;i++)
  {
    cout<<"nhap gia tri phan tu thu "<<i<<": ";
    cin>>a[i];
  }
   ptln=a[1];
   csmax=1;
   for(i=2;i<=n;i++)
    if(a[i]>ptln)
    {
      ptln=a[i];
      csmax=i;
    }

   cout<<"\nPhan tu lon nhat la "<<ptln;
   cout<<"\nChi so cua phan tu lon nhat la "<<csmax;
   return 0;
}
```

***Ví dụ 2.*** *Sắp xếp dãy số nguyên bằng thuật toán tráo đổi*

***Input:*** Số nguyên dương $N$ ($N \le 250$) và dãy $A$ gồm $N$ số nguyên dương $a_1$, $a_2$,..., $a_N$, mỗi số đều không vượt quá 500.

***Output:*** Dãy số *A* đã được sắp xếp thành dãy không giảm.

Để giải bài toán này, ta sẽ sử dụng thuật toán tráo đổi như mô tả trong sách giáo khoa Tin học 10.

```
(* Chuong trinh giai bai toan sap xep day so *)
#include <iostream>
#define max 250
using namespace std;
int main()
{
  int a[max];
  int n,i,temp;
  cout<<"nhap so phan tu: ";
  cin>>n;
  for(i=1;i<=n;i++)
  {
    cout<<"\nnhap gia tri phan tu thu "<<i<<": ";
    cin>>a[i];
  }
  cout<<"\n Mang vua nhap la: \n";
  for(i=1;i<=n;i++)  cout<<a[i]<<'\t';
  for(int j=n;j>=2;j--)
    for (i=1;i<=j-1;i++)
      if(a[i]>a[i+1])
      {
        temp=a[i];
        a[i]=a[i+1];
        a[i+1]=temp;
      }
  cout<<"\n Mang sau khi sap xep la: \n";
  for(i=1;i<=n;i++)  cout<<a[i]<<'\t';
   return 0;
}
```

***Ví dụ 3.*** *Tìm kiếm nhị phân*

***Input:*** Dãy *A* là dãy tăng gồm *N* ($N \le 250$) số nguyên dương $a_1, a_2,..., a_N$ và số nguyên *k*.

***Output:*** Chỉ số *i* mà $a_i = k$ hoặc thông báo "*Khong tim thay*" nếu không có số hạng nào của dãy *A* có giá trị bằng *k*.

Để giải bài toán này, chúng ta sẽ sử dụng thuật toán tìm kiếm nhị phân được trình bày trong sách giáo khoa Tin học 10.

*Bước 1.* Nhập *N*, các số hạng $a_1, a_2,..., a_N$ và khoá *k*;

*Bước 2.* $Dau \leftarrow 1, Cuoi \leftarrow N$;

*Bước 3.* $Giua \leftarrow \left[\frac{Dau + Cuoi}{2}\right]$;

*Bước 4.* Nếu $a_{Giua} = k$ thì thông báo chỉ số *Giua*, rồi kết thúc;

*Bước 5.* Nếu $a_{Giua} > k$ thì đặt $Cuoi = Giua - 1$ rồi chuyển đến bước 7;

*Bước 6.* $Dau \leftarrow Giua + 1$;

*Bước 7.* Nếu *Dau* > *Cuoi* thì thông báo dãy A không có số hạng có giá trị bằng *k*, rồi kết thúc;

*Bước 8.* Quay lại bước 3.

*Hình 12. Thuật toán tìm kiếm nhị phân*

(* Chuong trinh cai dat thuat toan tim kiem nhi phan*)

```
#include <iostream>
#define max 250
using namespace std;
int main()
{
  int a[max];
  int n,i,dau,cuoi,giua;
  bool timthay;
  cout<<"nhap so phan tu: ";
  cin>>n;
  for(i=1;i<=n;i++)
  {
    cout<<"\nnhap gia tri phan tu thu "<<i<<": ";
    cin>>a[i];
  }
  cout<<"\n Mang vua nhap la: \n";
  for(i=1;i<=n;i++) cout<<a[i]<<'\t';
   int k;
   cout<<"\nNhap gia tri k: ";
  cin>>k;
  dau = 1; cuoi = n;
  timthay=false;
  do
  {
    giua=(dau+cuoi)/2;
    if (a[giua]==k) timthay=true;
    else if (a[giua]>k) cuoi=giua-1;
      else dau=giua+1;
  } while ((dau<=cuoi)&& (timthay==false));
  if (timthay==true)
    cout<<"chi so tim duoc la: "<<giua;
  else cout<<"Khong tim thay";
   return 0;
}
```

## 2. Kiểu mảng hai chiều

Xét bài toán tính và đưa ra màn hình bảng cửu chương.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |

*Bảng cửu chương*

Ta thấy bảng cửu chương có dạng bảng gồm các giá trị cùng kiểu. Ta có thể biểu diễn bảng cửu chương bằng kiểu dữ liệu mảng hai chiều.

*Mảng hai chiều là bảng các phần tử cùng kiểu.*

Nhận xét rằng mỗi hàng của mảng hai chiều có cấu trúc như một mảng một chiều cùng kích thước. Nếu ta coi mỗi hàng của mảng hai chiều là một phần tử thì ta có thể nói mảng hai chiều là mảng một chiều mà mỗi phần tử là mảng một chiều.

Như vậy, ta cũng có thể mô tả dữ liệu của bảng cửu chương là kiểu mảng một chiều gồm 10 phần tử, mỗi phần tử lại là mảng một chiều có 9 phần tử, mỗi phần tử là một số nguyên.

Tương tự như với kiểu mảng một chiều, với kiểu mảng hai chiều, các ngôn ngữ lập trình cũng có các quy tắc, cách thức cho phép xác định:

- Tên kiểu mảng hai chiều;
- Số lượng phần tử của mỗi chiều;
- Kiểu dữ liệu của phần tử;
- Cách khai báo biến;
- Cách tham chiếu đến phần tử.

***a) Khai báo***

Tổng quát, khai báo biến mảng hai chiều trong C++ như sau:

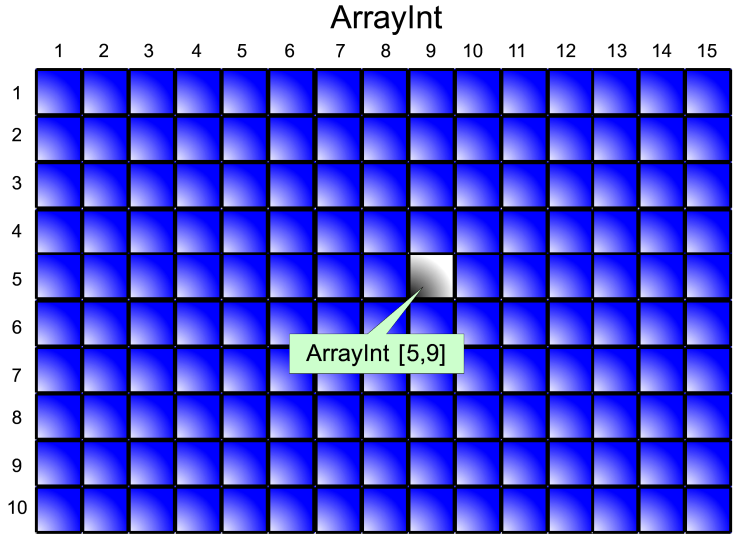
***<kiểu dữ liệu> <tên biến mảng>[chỉ số hàng][chỉ số cột];***

***Ví dụ.*** Các khai báo sau đây là hợp lệ:

```
int a[100][100];
float b[100][100];
```

Tham chiếu tới phần tử của mảng hai chiều được xác định bởi tên mảng cùng với hai chỉ số được cách nhau bởi dấu phẩy và viết trong cặp ngoặc **[** và **]**.

***Ví dụ.*** Tham chiếu tới phần tử ở dòng thứ 5, cột thứ 9 của biến mảng *ArrayInt* khai báo trong ví dụ 1 được viết: *ArrayInt* [5][ 9]**.**




*Hình 13. Minh hoạ mảng hai chiều*

***b) Một số ví dụ***

***Ví dụ 1.*** Chương trình sau tính và đưa ra màn hình bảng cửu chương.

```
#include <iostream>
using namespace std;
int main()
{
  int a[9][10];
```

```
  int i,j;
  for(i=1;i<=9;i++)
    for(j=1;j<=10;j++)
      a[i][j]=i*j;
  for(i=1;i<=9;i++)
  {
    for(j=1;j<=10;j++)
      std::cout<<a[i][j]<<'\t';
    std::cout<<endl;
  }
   return 0;
}
```

***Ví dụ 2.*** Chương trình sau nhập vào từ bàn phím các phần tử của mảng hai chiều *B* gồm 5 dòng, 7 cột với các phần tử là các số nguyên và số nguyên *k*. Sau đó, đưa ra màn hình các phần tử của mảng có giá trị nhỏ hơn *k*.

```
#include <iostream>

using namespace std;

int main()
{
  int a[10][10];
  int i,j,k;
  cout<<"nhap k";
  cin>>k;
  for(i=1;i<=5;i++)
    for(j=1;j<=7;j++)
    {
      cout<<"a["<<i<<"]["<<j<<"]:";
      cin>>a[i][j];
    }
  for(i=1;i<=5;i++)
  {
    for(j=1;j<=7;j++)
      std::cout<<a[i][j]<<'\t';
    std::cout<<endl;
  }
  for(i=1;i<=5;i++)
  {
    for(j=1;j<=7;j++)
      if(a[i][j]<k)
        std::cout<<a[i][j]<<'\t';
  }
   return 0;
}
```

***Chú ý***

- Các biến mảng thường gồm số lượng lớn các phần tử nên cần lưu ý phạm vi sử dụng chúng để khai báo kích thước và kiểu dữ liệu để tiết kiệm bộ nhớ.
- Ngoài hai kiểu mảng một chiều và hai chiều, còn có kiểu mảng nhiều chiều.

## BÀI TẬP VÀ THỰC HÀNH 3

### 1. Mục đích, yêu cầu

- *Nâng cao kĩ năng sử dụng một số câu lệnh và một số kiểu dữ liệu thông qua việc tìm hiểu, chạy thử các chương trình có sẵn;*
- *Biết giải một số bài toán tính toán, tìm kiếm đơn giản trên máy tính.*

### 2. Nội dung

***Bài 1.*** Tạo mảng $A$ gồm $n$ ($n \leq 100$) số nguyên, mỗi số có trị tuyệt đối không vượt quá 300. Tính tổng các phần tử của mảng là bội số của một số nguyên dương $k$ cho trước.

***Bài 2.*** Viết chương trình tìm phần tử có giá trị lớn nhất của mảng và đưa ra màn hình chỉ số và giá trị của phần tử tìm được. Nếu có nhiều phần tử có cùng giá trị lớn nhất thì đưa ra phần tử có chỉ số nhỏ nhất.

## BÀI TẬP VÀ THỰC HÀNH 4

### 1. Mục đích, yêu cầu

- *Biết nhận xét, phân tích đề xuất thuật toán giải bài toán sao cho chương trình chạy nhanh hơn;*
- *Làm quen với dữ liệu có cấu trúc và bài toán sắp xếp.*

### 2. Nội dung

***Bài 1.***

**a)** Hãy tìm hiểu và chạy thử chương trình thực hiện thuật toán sắp xếp dãy số nguyên bằng tráo đổi với các giá trị khác nhau của *n* dưới đây. Qua đó, nhận xét về thời gian chạy của chương trình.

***Bài 2***

**a)** Hãy đọc và tìm hiểu những phân tích để viết chương trình giải bài toán:

*Cho mảng A gồm n phần tử. Hãy viết chương trình tạo mảng B[1..n], trong đó B[i] là tổng của i phần tử đầu tiên của A.*

Tuy tốc độ tính toán của máy tính nhanh nhưng có giới hạn. Do đó, trong khi viết chương trình, ta nên tìm cách viết sao cho chương trình thực hiện càng ít phép toán càng tốt.

# §12. KIỂU DỮ LIỆU XÂU

Dữ liệu trong các bài toán không chỉ thuộc kiểu số mà cả kiểu phi số - dạng kí tự. Dữ liệu kiểu xâu là dãy các kí tự.

***Ví dụ.*** Các xâu kí tự đơn giản:

'Bach khoa'  'KI SU'  '2007 la nam Dinh Hoi'

*Xâu là dãy các kí tự trong bảng mã ASCII, mỗi kí tự được gọi là một phần tử của xâu. Số lượng kí tự trong một xâu được gọi là độ dài của xâu. Xâu có độ dài bằng 0 gọi là xâu rỗng.*

Các ngôn ngữ lập trình đều có quy tắc, cách thức cho phép xác định:

- Tên kiểu xâu;
- Cách khai báo biến kiểu xâu;
- Số lượng kí tự của xâu;
- Các phép toán thao tác với xâu;
- Cách tham chiếu tới phần tử xâu.

Biểu thức gồm các toán hạng là biến xâu, biến kí tự hoặc hằng xâu được gọi là biểu thức xâu. Với dữ liệu kiểu xâu có thể thực hiện phép toán ghép xâu và các phép toán quan hệ.

Có thể xem xâu là mảng một chiều mà mỗi phần tử là một kí tự. Các kí tự của xâu được đánh số thứ tự, thường bắt đầu từ 0 đối với C++ và từ 1 đối với Pascal.

Tương tự như với mảng, tham chiếu tới phần tử của xâu được xác định bởi tên biến xâu và chỉ số đặt trong cặp ngoặc [ và ].

Ví dụ, trong C++: giả sử có biến *Hoten* = '*Nguyen Le Huyen*' thì *Hoten*[5] cho ta kí tự '*n*' là kí tự thứ sáu của biến xâu *Hoten*.

Dưới đây trình bày cách khai báo kiểu dữ liệu xâu, các thao tác xử lí xâu và một số ví dụ sử dụng kiểu xâu trong Pascal.

## 1. Khai báo

Khi khai báo biến xâu cần sử dụng thư viện **cstring**

Để khai báo kiểu dữ liệu xâu sử dụng tên dành riêng ***string*** tiếp theo là độ dài lớn nhất của xâu (không vượt quá 255) được ghi trong cặp ngoặc **[** và **]**.

Biến kiểu xâu có thể khai báo như sau:

**string <tên biến xâu>[độ đài lớn nhất]**

***Ví dụ***

**string hoten[50];**

Trong mô tả xâu có thể bỏ qua phần khai báo độ dài, chẳng hạn:

**string diachi**;

Khi đó độ dài lớn nhất của xâu sẽ nhận giá trị ngầm định là 255.

## 2. Các thao tác xử lí xâu

***a)*** Phép ghép xâu, kí hiệu là dấu cộng (+), được sử dụng để ghép nhiều xâu thành một. Có thể thực hiện phép ghép xâu đối với các hằng và biến xâu.

***Ví dụ***

Phép ghép xâu:

```
#include <iostream>
```

```
#include <string>

using std::string;
using namespace std;

int main()
{
  string s,t;
  getline(cin,s);
  getline(cin,t);
  s=s+t;
  cout <<s;
  return 0;
}
```

Chúng ta vẫn sử dụng thủ tục cin để ghép xâu được, điểm khác nhau giữa cin và getline được thể hiện trong ví dụ sau:

Giả sự bạn muốn nhập xâu "Truong Dong Da" cho xâu s.

```
cin>>s;          chỉ nhận được xâu "Truong"
getline(cin,s);     nhận toàn bộ xâu "Truong Dong Da"
```

***b)*** Các phép so sánh bằng (=), khác (<>), nhỏ hơn (<), lớn hơn (>), nhỏ hơn hoặc bằng (<=), lớn hơn hoặc bằng (>=) có thứ tự ưu tiên thực hiện thấp hơn phép ghép xâu và thực hiện việc so sánh hai xâu theo các quy tắc sau:

- Xâu *A* là lớn hơn xâu *B* nếu như kí tự đầu tiên khác nhau giữa chúng kể từ trái sang trong xâu *A* có mã ASCII lớn hơn.
- Nếu *A* và *B* là các xâu có độ dài khác nhau và *A* là đoạn đầu của *B* thì *A* là nhỏ hơn *B*.

***Ví dụ***

"May tinh" < "May tinh cua toi"

- Hai xâu được coi là bằng nhau nếu như chúng giống nhau hoàn toàn.

***Ví dụ***

"TIN HOC" = "TIN HOC"

Để xử lí các xâu có thể sử dụng các thủ tục và hàm chuẩn dưới đây:

***b)*** Hàm **<tên biến xâu>.erase(vt, n)**

xóa n của chuỗi <tên biến xâu> kể từ vt; nếu không quy định giá trị n thì tất cả các ký tự của <tên biến xâu> từ vt trở đi sẽ bị xóa

***Ví dụ***

```
#include <iostream>
#include <string>
using std::string;
using namespace std;
int main()
{
  string s="truong da";
  s=s.erase(6,3);
  cout <<s;
  return 0;
}
```

*Kết quả màn hình cho là xâu "truong"*

***c)*** Hàm ***<tên biến xâu>.insert(vt, <xâu chèn>)*** chèn <xâu chèn> vào xâu <*tên biến xâu*>, bắt đầu ở vị trí *vt*.

***Ví dụ***

```
#include <iostream>
#include <string>

using std::string;
using namespace std;

int main()
{
  string s="truong da",t=" dong";

  s=s.insert(6,t);
  cout <<s;
  return 0;
}
```

*Kết quả màn hình xuất ra xâu "truong dong da"*

***d)*** Hàm ***<tên biến xâu>.substr(vt, n)*** tạo xâu gồm *N* kí tự liên tiếp bắt đầu từ vị trí *vt* của xâu <*tên biến xâu*>.

***Ví dụ***

```
#include <iostream>
#include <string>
using std::string;
using namespace std;
int main()
{
  string s,t;
  getline(cin,s);
  t=s.substr(2,4);
  cout <<t;
  return 0;
}
```

*Chú ý: Trong C++ được đếm bắt đầu bởi số 0.*

*Ví dụ: s= "truong dong da";*

*t=s.substr(7,4); sẽ cho chúng ta kết quả t= "dong"*

***e)*** Hàm *len*(*s*) cho giá trị là độ dài xâu *s*.

**Ví dụ:**

St="nguyen";

thì len(nguyen)=6

***f)*** **So sánh**

Bạn có thể đơn giản là sử dụng những toán tử quan hệ ( ==, !=, <, <=, >=) được định nghĩa sẵn. Tuy nhiên, nếu muốn so sánh một phần của một chuỗi thì sẽ cần sử dụng phương thức compare():

```
int compare ( const string& str ) const;
int compare ( const char* s ) const;
int compare ( size_t pos1, size_t n1, const string& str ) const;
int compare ( size_t pos1, size_t n1, const char* s) const;
```

```
int compare ( size_t pos1, size_t n1, const string& str, size_t pos2,
size_t n2 ) const;
int compare ( size_t pos1, size_t n1, const char* s, size_t n2) const;
```

Hàm trả về 0 khi hai chuỗi bằng nhau và lớn hơn hoặc nhỏ hơn 0 cho trường hợp khác

Ví dụ:

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
string str1 ("green apple");
string str2 ("red apple");
if (str1.compare(str2) != 0)
cout << str1 << " is not " << str2 << "\n";
if (str1.compare(6,5,"apple") == 0)
cout << "still, " << str1 << " is an apple\n";
if (str2.compare(str2.size()-5,5,"apple") == 0)
cout << "and " << str2 << " is also an apple\n";
if (str1.compare(6,5,str2,4,5) == 0)
cout << "therefore, both are apples\n";
return 0;
}
```

***g)*** **Hàm find( )** tìm kiếm xem một ký tự hay một chuỗi nào đó **có** xuất hiện trong một chuỗi str cho trước hay không. Có nhiều cách dùng phương thức này:

str.find(int ch, int pos = 0); tìm ký tự ch kể từ vị trí pos đến cuối chuỗi str

str.find(char *s, int pos = 0); tìm s (mảng ký tự kết thúc '\0') kể từ vị trí pos đến cuối str.find(string& s, int pos = 0); tìm chuỗi s kể từ vị trí pos đến cuối chuỗi.

Nếu không quy định giá trị pos thì hiểu mặc nhiên là 0; nếu tìm có thì phương thức trả về vị trí xuất hiện đầu tiên, ngược lại trả về giá trị -1.

```
//find substring
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
string str="ConCho chay qua rao";
cout << str.find("chay") << endl;
cout << (int)str.find("Chay") << endl; // -1
getchar();
return 0;
}
```

***h)*** **Hàm tìm kiếm ngược (rfind)**

```
//find from back
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
string str="ConCho chay qua chay qua rao";
```

```
cout << str.find("chay") << endl; // 7
cout << (int)str.rfind("chay") << endl; // 16
getchar();
return 0;
}
```

***i)*** **Hàm replace( )** thay thế một đoạn con trong chuỗi str **cho** trước (đoạn con kể từ một vị trí pos và đếm tới nchar ký tự ký tự về phía cuối chuỗi) bởi một chuỗi s nào đó, hoặc bởi n ký tự ch nào đó. Có nhiều cách dùng, thứ tự tham số như sau:

```
str.replace(int pos, int nchar, char *s);
str.replace(int pos, int nchar, string s);
str.replace(int pos, int nchar, int n, int ch);
// replace from a string
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
string str="con cho la con cho con. Con meo ko phai la con cho";
str.replace(4, 3, "CHO"); // "con CHO la con cho con. Con meo ko phai la
con cho";
cout << str << endl;
getchar();
return 0;
}
```

***j)*** **Tách xâu**

Trong việc xử lý xâu ký tự, không thể thiếu được các thao tác tách xâu ký tự thành nhiều xâu ký tự con thông qua các ký tự ngăn cách. Các hàm này có sẵn trong các ngôn ngữ khác như Visual Basic,Java, hay thậm chí là trong <string.h> (không phải <string>) Với STL, các bạn có thể dễ dàng tự xây dựng một hàm với chức năng tương tự:

```
#include <iostream>
#include <vector>
#include <stdio.h>
#include <string>
using namespace std;
int main()
{
string S = "Xin chao tat ca cac ban"; // Khởi tạo giá trị của xâu
string::iterator t, t2; // Các biến lặp
vector<string> split; // Mảng các xâu (lưu kết quả tách)
for (t=S.begin(); t<S.end();)
{ // Lặp từ vị trí bắt đầu
t2=find(t, S.end(), ' '); // TÌm ký tự space ' ' đầu tiên
// kể từ vị trí t
if (t!=t2) split.push_back(string(t, t2)); // Lấy xâu ký tự giữa 2 vị trí
t = t2+1; // Chuyển sang vị trí sau
}
for (int i=0; i<splitìsize(); i++)
cout << split[i] << endl; // In mảng các xâu ký tự
getchar();
return 0;
}
```

**Output:**

Xin
chao
tat
ca
cac
ban

***k)*** **Hàm <tên biến xâu>.size()** lấy chiều dài của xâu <tên biến xâu>

```
#include <iostream>
#include <string>

using std::string;
using namespace std;

int main()
{
  string s="truong da";
  int l=s.size();
  cout <<l;
  return 0;
}
```

Kết quả màn hình là 9.

***l)*** **Hàm đổi ký tự hoa toupper(<ký tự>):**

**Ví dụ:**

```
#include <iostream>
#include <string>

using std::string;
using namespace std;

int main()
{
  string s="truong dong da";
  int l=s.size();
  for(int i=0;i<=l;i++)
    s[i]=toupper(s[i]);
  cout <<s;
  return 0;
}
```

***m)*** **Hàm đổi ký tự sang thường tolower(<ký tự>):**

**Ví dụ:**

```
#include <iostream>
#include <string>

using std::string;
using namespace std;

int main()
{
  string s="TRUONG DONG DA";
  int l=s.size();
  for(int i=0;i<=l;i++)
```

```
    s[i]=tolower(s[i]);

  cout <<s;
  return 0;
}
```

## §13. KIỂU BẢN GHI

Dữ liệu kiểu bản ghi (record) dùng để mô tả các đối tượng có cùng một số thuộc tính mà các thuộc tính có thể có các kiểu dữ liệu khác nhau.

Ví dụ, bảng kết quả thi tốt nghiệp gồm thông tin về các thí sinh như họ và tên, ngày sinh, giới tính, điểm các môn thi,... mà những thông tin này thuộc các kiểu dữ liệu khác nhau.

**Bảng kết quả thi tốt nghiệp**

| Họ và tên | Ngày sinh | Giới tính | Điểm Tin | Điểm Toán | Điểm Lí | Điểm Hoá | Điểm Văn | Điểm Sử | Điểm Địa |
|---|---|---|---|---|---|---|---|---|---|
| Nguyễn Thị Minh Huệ | 12/12/1990 | Nữ | 9 | 10 | 7 | 8 | 8 | 7 | 8 |
| Dương Trúc Lâm | 2/1/1990 | Nam | 9 | 10 | 8 | 8 | 9 | 6 | 7 |
| Đào Văn Bình | 5/12/1990 | Nam | 8 | 8 | 9 | 8 | 7 | 7 | 6 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Một ví dụ khác, khi xem xét doanh số của một cửa hàng, ta quan tâm đến tập hoá đơn bán hàng, mỗi hoá đơn đều có các thuộc tính như tên hàng, đơn giá, chủng loại, số lượng bán, giá thành, người bán, người mua, ngày bán,...

Để mô tả các đối tượng như vậy, ngôn ngữ lập trình cho phép xác định kiểu dữ liệu bản ghi (trong C++ gọi là kiểu cấu trúc (struct)). Mỗi đối tượng được mô tả bằng một bản ghi. Mỗi thuộc tính của đối tượng tương ứng với một trường của bản ghi. *Các trường khác nhau có thể có các kiểu dữ liệu khác nhau*.

Ngôn ngữ lập trình đưa ra quy tắc, cách thức xác định:

- Tên kiểu bản ghi;
- Tên các thuộc tính (trường);
- Kiểu dữ liệu của mỗi trường;
- Cách khai báo biến;
- Cách tham chiếu đến trường.

Dưới đây giới thiệu cách khai báo kiểu, biến, tham chiếu đến trường và phép gán giá trị bản ghi trong C++

### 1. Khai báo

Các thông tin cần khai báo bao gồm tên kiểu bản ghi, tên các thuộc tính, kiểu dữ liệu của mỗi thuộc tính.

Do dữ liệu kiểu bản ghi thường dùng để mô tả nhiều đối tượng nên ta thường định nghĩa một kiểu bản ghi và sau đó dùng nó để khai báo các biến liên quan.

Kiểu bản ghi thường được định nghĩa như sau:

```
struct tên_struct{
  type  thành_viên_1;
  type  thành_viên_2;
  …
} [tên_đối_tượng_struct];
```

Trong đó:

**struct** là một từ khóa.

**tên_struct** là tên kiểu dữ liệu struct mà ta định nghĩa.

**thành_viên_1, …** là các phần tử thành viên của struct.

**tên_đối_tượng_struct:** là tên biến thuộc kiểu tên_struct.

Hai cách khai báo sau đây là tương đương.

**Cách 1:**

```
struct product
{
  int weight;
  float price;
}
product apple;
product banana, melon;
```

**Cách 2:**

```
struct product
{
  int weight;
  float price;
}apple, banana, melon;
```

**Giải thích:** trong hai ví dụ trên product là kiểu dữ liệu struct mà ta tạo ra. Nó gồm có hai thành viên là weight và price. Tương ứng với kiểu dữ liệu này, ta có các biến apple, banana, melon. Để khai báo nó, ta có thể sử dụng một trong hai cách theo như hai ví dụ này. Cần lưu ý rằng, các biến apple, banana, melon là các biến thuộc kiểu dữ liệu product.

Để truy cập đến các thành viên của biến struct, ta sử dụng toán tử chấm (.).

```
apple.weight = 200;
apple.price = 2;
banana.weight = 150;
banana.price = 1;
…
```

Chúng ta có thể hiểu apple.weight là khối lượng của táo, apple.price là giá tiền của táo... Có thể xem mỗi thành viên của một biến struct như là một biến độc lập mà ta có thể truy cập bằng cách tên_biến.phần_tử_thành_viên. Ta hoàn toàn có thể thực hiện các phép toán tương ứng với mỗi dữ liệu thành viên này (nghĩa là các phép toán đó tương ứng với các phép toán trên kiểu dữ liệu của các phần tử thành viên đó: nếu phần tử thành viên là kiểu nguyên thì ta có thể thực hiện các phép toán số nguyên với thành viên này, nếu phần tử thành viên là kiểu xâu thì có thể thực thi các phép toán với xâu cho biến thành viên này,…).

**Ví dụ** tôi mua 400gam táo, và 300g chuối. Giá của mỗi 100g táo là 1$, và 100g chuối là 0.7$. Cần tính toán số tiền mà tôi cần trả.

```
apple.weight = 400;
apple.price = (float)apple.weight*1/100;
banana.weight = 300;
banana.price = banana.weight*0.7/100;
float money = apple.price+banana.price;
```

Như trong ví dụ trên, tôi có thể thực hiện các phép toán số học với các phần tử thành viên là kiểu số như c|c biến số bình thường.

Struct là một kiểu dữ liệu do người dùng định nghĩa (nhờ vào từ khóa struct – từ khóa struct viết thường). Ta cũng có thể khai báo một mảng các phần tử thuộc kiểu struct. Ví dụ sau đ}y minh họa cho việc mua bán khi đi siêu thị.

**Bài toán:** Trong siêu thị, giá táo và chuối được ấn định trên từng sản phẩm

tùy thuộc chất lượng của sản phẩm, không phụ thuộc vào khối lượng của nó.

a. Tính khối lượng hàng hóa (kg) mà người tiêu dùng mua.

b. Nếu khách hàng mua hàng trị giá trên 10$, thì người tiêu dùng sẽ được

khuyến mãi. Hãy kiểm tra xem người tiêu dùng có được khuyến mãi

hay không.

Biết rằng số lượng táo và chuối do người tiêu dùng lựa chọn.

Chương trình

```
#include<iostream>
using namespace std;
#define MAX 10
struct product{
  int weight;
  float price;
}apples[MAX], bananas[MAX];
/*
Khai báo hàm
*/
void buyProducts(product pd, string
name, int &weight, float &price)
{
  int i = 0;
  cout<<”Buy “<<name<<endl;
  while (true){
   cout<<”weight and price: <<endl;
   cin>>pd[i].weight;
   cin>>pd[i].price;
   weight += pd[i].weight;
   price += pd[i].price;
   cout<<”Continue to buy “<<name<<”
(y/n) ?”;
   char yn;
   cin>>yn;
   if((yn==’n’)||(yn==’N’)) break;
   else i++;
  }
}
int main()
{
int weight = 0;
  float price = 0;
  buyProducts(apples, “apples”, weight,
price);
  buyProducts(bananas, “bananas”,
weight, price);
  cout<<”Total weight:
“<<(float)weight/1000<<endl;
  if(price>10)
   cout<<”Promotion”;
  else
   cout<<”No Promotion”;
  return 0;
}
```

**Kết quả**

Buy apples

weight and price

100

2

Continue to buy apples (y/n) ?y

weight and price
200
3
Continue to buy apples (y/n) ?n
Buy bananas
weight and price
150
1
Continue to buy bananas (y/n)
?y
weight and price
250
3
Continue to buy bananas (y/n)
?n
Total weight: 0.7
No Promotion

**Giải thích:**

Mảng apples và bananas thuộc kiểu dữ liệu product. Hàm buyProducts dùng để tính toán khối lượng và tổng giá hàng hóa mỗi loại mua được. Nếu tham số products trong hàm là apples, thì nó sẽ tính tương ứng với khối lượng tổng cộng và tổng đơn gi| của apples. Tương tự cho bananas. Ta sử dụng vòng lặp while vì ta không biết chính xác số lượng hàng hóa mà người dùng lựa chọn. Tuy nhiên, con số tổng không thể vượt hằng số MAX mà ta đã định nghĩa. Biến weight và price được truyền theo tham biến, do đó, trước phép cộng dồn (toán tử cộng đồng nhất +=), ta không cần khởi tạo giá trị 0 cho chúng, ta có thể khởi tạo cho chúng ngay đầu hàm main. Nếu khởi tạo đầu hàm buyProducts, thì ta chỉ có thể tính được khối lượng và giá của từng loại một mà thôi (khi đó, nếu muốn tính toán cho cả hai sản phẩm, ta bổ sung thêm biến để lưu lại các giá trị này).

Sau khi gọi hàm này hai lần, tương ứng với apples và bananas, thì biến weight lưu lại tổng khối lượng của cả hai sản phẩm, biến price lưu lại giá của cả hai sản phẩm. Khối lượng được quy đổi sang kg, nên ta sẽ chia cho 1000. Nếu quy định đơn vị là kg, thì điều này là không cần thiết.

Trong phần thực thi chương trình, người tiêu dùng đ~ mua hai quả táo và hai quả chuối. Quả táo thứ nhất có khối lượng 100g và giá 2$, quả táo thứ hai – 200g và 3$. Quả chuối thứ nhất có khối lượng 150g và giá 1$, quả chuối thứ hai – 250g và 3$.

**TÓM TẮT**

- Kiểu dữ liệu có cấu trúc được xây dựng từ những kiểu dữ liệu đã có theo quy tắc, khuôn dạng do ngôn ngữ lập trình cung cấp.
- Mảng một chiều
  - Mảng một chiều là dãy hữu hạn các phần tử cùng kiểu.
  - Khai báo: tên mảng, kiểu chỉ số, kiểu phần tử
  - Tham chiếu phần tử mảng: *tên biến mảng*[*chỉ số phần tử*]
- Mảng hai chiều

- Mảng hai chiều là bảng các phần tử cùng kiểu.
- Khai báo: tên mảng, kiểu chỉ số dòng, kiểu chỉ số cột, kiểu phần tử.
- Tham chiếu phần tử mảng: *tên biến mảng***[***chỉ số dòng***][***chỉ số cột***]**

➢ Kiểu dữ liệu xâu

- Xâu là dãy các kí tự trong bảng mã ASCII.
- Các thao tác xử lí thường sử dụng

➢ Kiểu bản ghi

- Khai báo kiểu bản ghi: tên bản ghi, tên và kiểu các trường.
- Tham chiếu trường của bản ghi: *tên biến bản ghi***.***tên trường*

# BÀI TẬP VÀ THỰC HÀNH 5

## 1. Mục đích, yêu cầu

- *Làm quen với việc tìm kiếm, thay thế và biến đổi xâu.*

## 2. Nội dung

1. Đếm có bao nhiêu khoảng trắng trong chuỗi.

2. Tên của học sinh chỉ được phép có một khoảng trắng. Nhưng khi nhập, giáo viên nhập có một số tên có nhiều hơn một khoảng trắng. Em hãy giúp giáo viên của mình loại bỏ các khoảng trắng thừa trong tên của học sinh trong lớp.

3. Khi nhập tên của học sinh. Nhân viên văn phòng đã tách tên ra thành họ lót và tên. Em hãy nhập hai xâu đó thành họ và tên của học sinh.

4. Viết chương trình đổi những kí tự đầu tiên của tên thành phố thành chữ in hoa.

5. Viết chương trình đảo ngược các kí tự trong chuỗi.

   Ví dụ: nhập ABCDE, xuất ra màn hình là:EDCBA

6. Viết chương trình tìm kiếm 1 kí tự xem có trong chuỗi không, nếu có xuất ra vị trí của từ chứ kí tự đó. (Vd: xâu a là "ho chi minh": nhập 'm' => kết quả là 3)

7. Viết 1 chương trình đếm một kư tự xuất hiện bao nhiêu lần trong chuỗi. (vd:xâu a nhập là "ho chi minh", nhập 'i' =>kq: 2)

8. Nhập vào chuỗi s1 và s2, cho biết vị trí xuất hiện của chuỗi s2 trong s1.

9. Viết chương trình tìm kiếm tên trong chuỗi họ tên. Nếu có thì xuất ra là tên này đã nhập đúng, ngược lại thông báo là đã nhập sai.

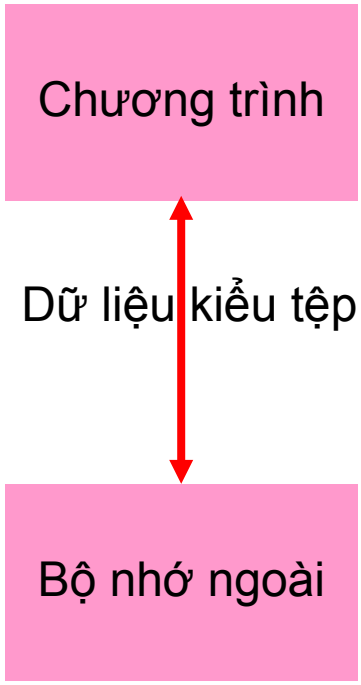
10. Viết chương đảo vị trí của từ đầu và từ cuối.

    Ví dụ: nhập "bo an co" xuat ra "co an bo"

11. Viết hàm cắt chuỗi họ tên thành chuỗi họ lót và chuỗi tên.

# Chương V. Tệp và thao tác với tệp

- Vai trò và các đặc điểm của kiểu dữ liệu tệp;
- Thao tác với tệp văn bản.



## §14. KIỂU DỮ LIỆU TỆP

### 1. Vai trò kiểu tệp

Tất cả các dữ liệu có các kiểu dữ liệu đã xét đều được lưu trữ ở bộ nhớ trong (RAM) và do đó dữ liệu sẽ bị mất khi tắt máy. Với một số bài toán, dữ liệu cần được lưu trữ để xử lí nhiều lần và với khối lượng lớn cần có kiểu dữ liệu tệp (file).

Kiểu dữ liệu tệp có những đặc điểm sau:

- Được lưu trữ lâu dài ở bộ nhớ ngoài (đĩa từ, CD,…) và không bị mất khi tắt nguồn điện;
- Lượng thông tin lưu trữ trên tệp có thể rất lớn và chỉ phụ thuộc vào dung lượng đĩa.

### 2. Phân loại tệp và thao tác với tệp

Xét theo cách tổ chức dữ liệu, có thể phân tệp thành hai loại:

- *Tệp văn bản* là tệp mà dữ liệu được ghi dưới dạng các kí tự theo mã ASCII. Trong tệp văn bản, dãy kí tự kết thúc bởi nhóm kí tự xuống dòng hay kí tự kết thúc tệp tạo thành một dòng.

Các dữ liệu dạng văn bản như sách, tài liệu, bài học, giáo án, các chương trình nguồn viết bằng ngôn ngữ bậc cao,... thường được lưu trữ dưới dạng tệp văn bản.

- *Tệp có cấu trúc* là tệp chứa dữ liệu được tổ chức theo một cách thức nhất định. Dữ liệu ảnh, âm thanh,... thường được lưu trữ dưới dạng tệp có cấu trúc.

Xét theo cách thức truy cập, có thể phân tệp thành hai loại:

- *Tệp truy cập tuần tự* cho phép truy cập đến một dữ liệu nào đó trong tệp chỉ bằng cách bắt đầu từ đầu tệp và đi qua lần lượt tất cả các dữ liệu trước nó.
- *Tệp truy cập trực tiếp* cho phép tham chiếu đến dữ liệu cần truy cập bằng cách xác định *trực tiếp* vị trí (thường là số hiệu) của dữ liệu đó.

*Khác với mảng, số lượng phần tử của tệp không xác định trước.*

*Hai thao tác cơ bản đối với tệp là ghi dữ liệu vào tệp và đọc dữ liệu từ tệp.*

Thao tác đọc/ghi với tệp được thực hiện với từng phần tử của tệp.

Để có thể thao tác với kiểu dữ liệu tệp, người lập trình cần tìm hiểu cách thức mà ngôn ngữ lập trình cung cấp cách:

- Khai báo biến tệp;
    - Mở tệp;
    - Đọc/ghi dữ liệu;
    - Đóng tệp.

## §15. THAO TÁC VỚI TỆP

Trong mục này ta chỉ xét cách khai báo, thao tác với tệp văn bản trong C++.

Để sử dụng được tệp ta phải khai báo thư viện **#include <fstream>**

### 1. Thao tác với tệp

***a) Mở tệp***

Tệp có thể dùng để chứa kết quả ra hoặc dữ liệu vào.

**Thủ tục mở tệp để ghi dữ liệu có dạng:**

**ofstream <tên biến tệp> (<tên tệp>);**

Tên tệp là một hằng xâu

***Ví dụ***

ofstream fo ("xuat.out");

**Thủ tục mở tệp để đọc dữ liệu có dạng:**

**ifstream <tên biến tệp> (<tên tệp>);**

Tên tệp là một hằng xâu

***Ví dụ***

```
ifstream fi ("nhap.inp");
```

***c) Đọc/ghi tệp văn bản***

**Ghi tệp văn bản:**

<tên biến tệp> **<<** <Danh sách kết quả>;

**Ví dụ:**

```
for (int i=1;i<=n;i++)
    {
        fo<<a[i]<<'\t';
    }
```

**Đọc tệp văn bản:**

<tên biến tệp> **>>** <Danh sách biến>;

**Ví dụ:**

```
fi>>n>>k;
for(int i=1;i<=n;i++)
{
   fo>>a[i];
}
```

***d) Đóng tệp***

Sau khi làm việc xong với tệp cần phải đóng tệp. Việc đóng tệp là đặc biệt quan trọng sau khi ghi dữ liệu, khi đó hệ thống mới thực sự hoàn tất việc ghi dữ liệu ra tệp.

Thủ tục đóng tệp có dạng:

*<tên biến tệp>*.**close();**

***Ví dụ***

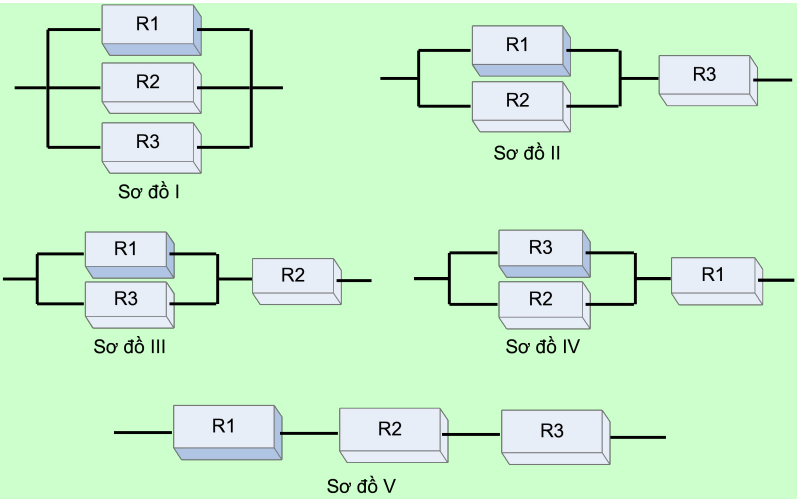
```
fi.close();
fo.close();
```

## §16. VÍ DỤ LÀM VIỆC VỚI TỆP

***Ví dụ 1***

Tính điện trở tương đương.

Cho ba điện trở R1, R2, R3. Sử dụng cả ba điện trở ta có thể tạo ra năm điện trở tương đương bằng cách mắc các sơ đồ nêu ở hình 17.




*Hình 17. Sơ đồ mắc điện trở.*

Mỗi cách mắc sẽ cho một điện trở tương đương khác nhau. Ví dụ, nếu mắc theo sơ đồ I thì điện trở tương đương sẽ là:

$$R = \frac{R1 * R2 * R3}{R1 * R2 + R1 * R3 + R2 * R3}$$

Còn nếu mắc theo sơ đồ V thì R = R1 + R2 +R3.

Cho tệp văn bản RESIST.DAT gồm nhiều dòng, mỗi dòng chứa ba số thực R1, R2 và R3, các số cách nhau một dấu cách, $0 < R1, R2, R3 \le 10^5$.

Chương trình sau đọc dữ liệu từ tệp RESIST.DAT, tính các điện trở tương đương và ghi kết quả ra tệp văn bản RESIST.EQU, mỗi dòng ghi năm điện trở tương đương của ba điện trở ở dòng dữ liệu vào tương ứng.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    float a[5];
    float r1,r2,r3;
    ifstream fi ("RESIST.DAT");
    ofstream fo ("RESIST.EQU");
    fi>>r1>>r2>>r3;
    a[1] =(r1*r2*r3)/(r1*r2+r1*r3+r2*r3);
    a[2] = (r1*r2)/(r1+r2)+r3;
    a[3] = (r1*r3)/(r1+r3)+r2;
    a[4] = (r2*r3)/(r2+r3)+r1;
    a[5] = r1+r2+r3;
```

```
    for(int i=1;i<=5;i++)
        fo<<a[i]<<'\t';
        fo<<"\n";
    fi.close();
    fo.close();

    return 0;
}
```

***Ví dụ 2.***

Thầy hiệu trưởng tổ chức cho giáo viên và học sinh của trường đi cắm trại, sinh hoạt ngoài trời ở vườn quốc gia Cúc Phương. Để lên lịch đến thăm khu trại các lớp, thầy Hiệu trưởng cần biết khoảng cách từ trại của mình (ở vị trí có toạ độ (0;0))đến trại các giáo viên chủ nhiệm lớp. Mỗi lớp có một khu trại, vị trí trại của mỗi thầy chủ nhiệm đều có tọa độ nguyên (x;y), được ghi vào tệp văn bản TRAI.TXT (như vậy, tệp TRAI.TEX chứa các cặp số nguyên liên tiếp, các số cách nhau bởi dấu cách và không kết thúc bằng dấu xuống dòng).

Chương trình sau sẽ đọc từ tệp TRAI.TXT các cặp toạ độ, tính và đưa ra màn hình khoảng cách (với độ chính xác hai chữ số sau dấu chấm thập phân) từ trại mỗi giáo viên chủ nhiệm lớp đến trại của thầy hiệu trưởng.

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

int main()
{
    float d;
    int x,y;
    ifstream fi ("TRAI.txt");
    fi>>x>>y;
    d=sqrt(x*x+y*y);
    cout<<"khoang cach: "<<d;
    fi.close();

    return 0;
}
```

**TÓM TẮT**

- Việc trao đổi dữ liệu với bộ nhớ ngoài được thực hiện thông qua kiểu dữ liệu tệp;
- Để có thể làm việc với tệp cần phải khai báo biến tệp;
- Mỗi ngôn ngữ lập trình đều có các chương trình chuẩn để làm việc với tệp;
- Các thao tác với tệp văn bản

## CÂU HỎI VÀ BÀI TẬP

**1.** Nêu một số trường hợp cần phải dùng tệp.

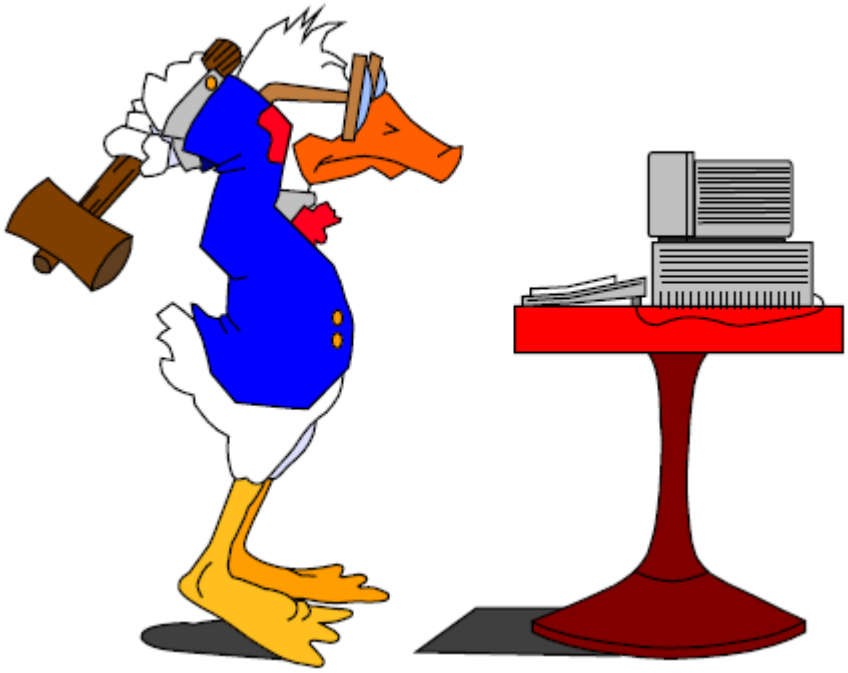
**2.** Trong sơ đồ thao tác với tệp, khi cần nhập dữ liệu từ tệp phải dùng những thao tác nào?

**3.** Tại sao cần phải có câu lệnh mở tệp trước khi đọc/ghi tệp?

**4.** Tại sao phải dùng câu lệnh đóng tệp sau khi đã kết thúc ghi dữ liệu vào tệp?

# Chương VI. Chương trình con và lập trình có cấu trúc

- Khái niệm cơ bản của chương trình con;
- Chương trình con: cấu trúc và phân loại;
- Tham số hình thức và tham số thực sự;
- Biến cục bộ, biến toàn cục;
- Thư viện chương trình.



## §17. CHƯƠNG TRÌNH CON VÀ PHÂN LOẠI

### 1. Khái niệm chương trình con

Các chương trình giải các bài toán phức tạp thường rất dài, có thể gồm hàng trăm, hàng nghìn lệnh. Khi đọc những chương trình dài, rất khó nhận biết được chương trình thực hiện các công việc gì và việc hiệu chỉnh chương trình cũng khó khăn. Vì vậy, vấn đề đặt ra là phải cấu tạo chương trình như thế nào để cho chương trình dễ đọc, dễ hiệu chỉnh, nâng cấp.

Mặt khác, việc giải quyết một bài toán phức tạp thường đòi hỏi và nói chung có thể phân thành các bài toán con.

Xét bài toán tính tổng bốn luỹ thừa:

$TLuythua = a^n + b^m + c^p + d^q$

Bài toán đó bao gồm bốn bài toán con tính $a^n$, $b^m$ , $c^p$, $d^q$, có thể giao cho bốn người, mỗi người thực hiện một bài. Giá trị *TLuythua* là tổng kết quả của bốn bài toán con đó. Với những

bài toán phức tạp hơn, mỗi bài toán con lại có thể được phân chia thành các bài toán con nhỏ hơn. Quá trình phân rã làm "mịn" dần bài toán như vậy được gọi là cách thiết kế từ trên xuống.

Tương tự, khi lập trình để giải bài toán trên máy tính có thể phân chia chương trình (gọi là chương trình chính) thành các khối (môđun), mỗi khối bao gồm các lệnh giải một bài toán con nào đó. Mỗi khối lệnh sẽ được xây dựng thành một *chương trình con*. Sau đó, chương trình chính sẽ được xây dựng từ các chương trình con này. Có thể xem chương trình con cũng là một chương trình và nó cũng có thể được xây dựng từ các chương trình con khác.

Cách lập trình như vậy dựa trên phương pháp lập trình có cấu trúc và chương trình được xây dựng gọi là chương trình có cấu trúc.

*Chương trình con là một dãy lệnh mô tả một số thao tác nhất định và có thể được thực hiện (được gọi) từ nhiều vị trí trong chương trình.*

Ví dụ, chương trình nhập dữ liệu từ bàn phím, tính và đưa ra màn hình giá trị *TLuythua* được mô tả như trên với *a*, *b*, *c*, *d* có kiểu thực và *m*, *n*, *p*, *q* có kiểu nguyên thì chương trình trong C++ có thể như sau:

```
#include <iostream>

using namespace std;

int main()
{
    float Tluythua,luythua1,luythua2,luythua3,luythua4;
    float a,b,c,d;
    int i,n,m,p,q;
    cout<<"Hay nhap du lieu theo thu tu a,b,c,d,m,n,p,q";
    cin>>a>>b>>c>>d>>m>>n>>p>>q;
    luythua1 =1;
    for(i=1;i<=n;i++) luythua1*=a;
   luythua2 =1;
    for(i=1;i<=m;i++) luythua2*=b;
    luythua3 =1;
    for(i=1;i<=p;i++) luythua3*=c;
    luythua4 =1;
    for(i=1;i<=q;i++) luythua4*=d;
    Tluythua = luythua1+luythua2+luythua3+luythua4;
    cout<<"Tong luy thua "<<Tluythua;

    return 0;
}
```

Trong chương trình trên có bốn đoạn lệnh tương tự nhau, việc lặp lại những đoạn lệnh tương tự nhau làm cho chương trình vừa dài vừa khó theo dõi. Để nâng cao hiệu quả lập trình các ngôn ngữ lập trình bậc cao đều cung cấp khả năng xây dựng chương trình con dạng tổng quát "đại diện" cho nhiều đoạn lệnh tương tự nhau, chẳng hạn tính luỹ thừa *Luythua* = $x^k$, trong đó *Luythua* và *x* là giá trị kiểu thực còn *k* thuộc kiểu nguyên:

```
Int j;
Luythua=1;
for (j=1;j<=k;j++)
    luythua*=x;
```

Ta có thể đặt tên cho chương trình con này là *Luythua* và tên các biến chứa dữ liệu vào của nó là *x* và *k*. Khi cần tính luỹ thừa của những giá trị cụ thể ta chỉ cần viết tên gọi chương trình

con và thay thế ($x$, $k$) bằng giá trị cụ thể tương ứng. Chẳng hạn để tính $a^n$, $b^m$, $c^p$, $d^q$ ta viết *Luythua*($a$, $n$), *Luythua*($b$, $m$), *Luythua*($c$, $p$), *Luythua*($d$, $q$).

*Lợi ích của việc sử dụng chương trình con*

- Tránh được việc phải viết lặp đi lặp lại cùng một dãy lệnh nào đó tương tự như trong ví dụ tính *TLuythua* ở trên. Ngôn ngữ lập trình cho phép tổ chức dãy lệnh đó thành một chương trình con. Sau đó, mỗi khi chương trình chính cần đến dãy lệnh này chỉ cần gọi thực hiện chương trình con đó.
- Khi phải viết chương trình lớn hàng nghìn, hàng vạn lệnh, cần huy động nhiều người tham gia, có thể giao cho mỗi người (hoặc mỗi nhóm) viết một chương trình con, rồi sau đó lắp ghép chúng lại thành chương trình chính. Ví dụ, với các bài toán mà việc tổ chức dữ liệu vào và ra không đơn giản thường người ta chia bài toán thành ba bài toán con như nhập, xử lí và xuất dữ liệu, rồi viết các chương trình con tương ứng.
- Phục vụ cho quá trình trừu tượng hoá: Người lập trình có thể sử dụng các kết quả được thực hiện bởi chương trình con mà không phải quan tâm đến việc các thao tác này được cài đặt như thế nào. Trừu tượng hoá là tư tưởng chủ đạo để xây dựng chương trình nói chung và chương trình có cấu trúc nói riêng.
- Các ngôn ngữ lập trình thường cung cấp phương thức đóng gói các chương trình con nhằm cung cấp như một câu lệnh mới (tương tự như các lệnh gọi thực hiện các hàm và thủ tục chuẩn) cho người lập trình sử dụng mà không cần biết mã nguồn của nó như thế nào.
- Thuận tiện cho phát triển, nâng cấp chương trình. Do chương trình được tạo thành từ các chương trình con nên chương trình dễ đọc, dễ hiểu, dễ kiểm tra và hiệu chỉnh. Việc nâng cấp, phát triển chương trình con nào đó, thậm chí bổ sung thêm các chương trình con mới nói chung không gây ảnh hưởng đến các chương trình con khác.
- Hiện nay, ngày càng có nhiều thiết bị kĩ thuật số tiện ích như máy quay phim, máy ảnh, máy ghi âm, các thiết bị âm thanh, màn hình màu độ phân giải cao,... có thể được kết nối với máy tính. Việc thiết kế những chương trình con thực hiện các giao tiếp cơ bản với các thiết bị như vậy là rất cần thiết và giúp mở rộng khả năng ứng dụng của ngôn ngữ.

## 2. Phân loại và cấu trúc của chương trình con

### *a) Phân loại*

Trong nhiều ngôn ngữ lập trình, chương trình con thường gồm hai loại:

- *Hàm* là chương trình con thực hiện một số thao tác nào đó và trả về một giá trị qua tên của nó. Ví dụ hàm toán học hay hàm xử lí xâu:

  **sin, cos, sqr, sqrt, length, pos,...**

  **sin(x)** nhận vào giá trị thực $x$ và trả về giá trị $sinx$,
  **sqrt(x)** nhận vào giá trị $x$ trả về giá trị căn bậc hai của $x$,
  **length(x)** nhận vào xâu $x$ và trả về độ dài của xâu $x$,...

- *Thủ tục* (void) là chương trình con thực hiện các thao tác nhất định nhưng không trả về giá trị nào qua tên của nó. Ví dụ các thủ tục vào/ra chuẩn.

### *b) Cấu trúc chương trình con*

Chương trình con có cấu trúc tương tự chương trình, nhưng nhất thiết phải có tên và phần đầu dùng để khai báo tên, nếu là hàm phải khai báo kiểu dữ liệu cho giá trị trả về của hàm:

<*phần đầu*
[<*phần khai báo*>]

*<phần thân>*

**Phần khai báo**

Phần khai báo có thể có khai báo biến cho dữ liệu vào và ra, các hằng và biến dùng trong chương trình con.

**Phần thân**

Phần thân của chương trình con là dãy câu lệnh thực hiện để từ những dữ liệu vào ta nhận được dữ liệu ra hay kết quả mong muốn.

- *Tham số hình thức*

Các biến được khai báo cho dữ liệu vào/ra được gọi là *tham số hình thức* của chương trình con. Các biến được khai báo để dùng riêng trong chương trình con được gọi là *biến cục bộ*.

Ví dụ, trong chương trình con *Luythua(x, k)* ở trên thì *x*, *k* là các tham số hình thức và *j* là các biến cục bộ.

Nói chung, chương trình chính và các chương trình con khác không thể sử dụng được các biến cục bộ của một chương trình con nhưng mọi chương trình con đều sử dụng được các biến của chương trình chính. Do vậy, các biến của chương trình chính được gọi là *biến toàn cục*. Ví dụ, biến *TLuythua* khai báo trong chương trình chính ở ví dụ trên là biến toàn cục.

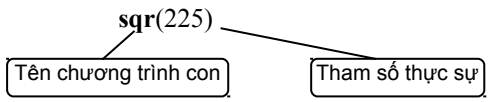
Một chương trình con thường có thể có hoặc không có tham số hình thức, có thể có hoặc không có biến cục bộ.

***c) Thực hiện chương trình con***

- *Tham số thực sự*

Để thực hiện (gọi) một chương trình con, ta cần phải có lệnh gọi nó tương tự lệnh gọi hàm hay thủ tục chuẩn, bao gồm tên chương trình con với tham số (nếu có) là các hằng và biến chứa dữ liệu vào và ra tương ứng với các tham số hình thức đặt trong cặp ngoặc ( và ). Các hằng và biến này được gọi là các *tham số thực sự*.

***Ví dụ***




Khi thực hiện chương trình con, các tham số hình thức để nhập dữ liệu vào sẽ nhận giá trị của tham số thực sự tương ứng, còn các tham số hình thức để lưu trữ dữ liệu ra sẽ trả giá trị đó cho tham số thực sự tương ứng.

Ví dụ, khi thực hiện tính *TLuythua* cần bốn lần gọi chương trình con *Luythua(x, k)* với các tham số (*a*, *n*), (*b*, *m*), (*c*, *p*), (*d*, *q*) và các tham số này là tham số thực sự tương ứng với tham số hình thức (*x*, *k*).

Sau khi chương trình con kết thúc, lệnh tiếp theo lệnh gọi chương trình con sẽ được thực hiện.

## §18. VÍ DỤ VỀ CÁCH VIẾT VÀ SỬ DỤNG CHƯƠNG TRÌNH CON

Các ngôn ngữ lập trình đều có các quy tắc viết và sử dụng chương trình con. Trong mục này sẽ xét cách viết và sử dụng chương trình con trong Pascal.

### 1. Cách viết và sử dụng thủ tục

Xét ví dụ vẽ hình chữ nhật có dạng sau:

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

Ta có thể vẽ hình chữ nhật trên với ba câu lệnh:

```
cout<<("* * * * * * *");
cout<<("*           *");
cout<<("* * * * * * *");
```

Như vậy, trong một chương trình, mỗi khi cần vẽ một hình chữ nhật như trên cần phải đưa vào ba câu lệnh này.

Trong chương trình sau, ta đưa ba câu lệnh trên vào một thủ tục có tên là *Ve_Hcn* (vẽ hình chữ nhật). Mỗi khi cần vẽ một hình chữ nhật ta cần đưa vào một *câu lệnh gọi* thủ tục đó. Chương trình gọi thủ tục *Ve_Hcn* ba lần để vẽ ba hình chữ nhật.

```
#include <iostream>
using namespace std;
void veHCN();
int main()
{
    cout << "Ve hinh chu nhat" << endl;
    veHCN();
    cout<<endl;cout<<endl;
    veHCN();
    cout<<endl;cout<<endl;
    veHCN();
    return 0;
}
void veHCN()
{
    cout<<"**********"<<endl;
    cout<<"*        *"<<endl;
    cout<<"**********"<<endl;
}
```

#### *a) Cấu trúc*

Thủ tục có cấu trúc như sau:

**void** <*tên thủ tục*>([<*danh sách tham số*>]);

**{**

**[**<*dãy các lệnh*>**]**

**}**

*Đầu thủ tục* gồm tên dành riêng ***void***, tiếp theo là tên thủ tục. Danh sách tham số có thể có hoặc không có.

Phần khai báo dùng để xác định các hằng, kiểu, biến và cũng có thể xác định các chương trình con khác được sử dụng trong thủ tục. Thủ tục *Ve_Hcn* ở trên không khai báo hằng, biến hay chương trình con nào.

Dãy câu lệnh được viết giữa cặp tên dành riêng ***{*** và ***}*** tạo thành thân của thủ tục.

Các thủ tục, nếu có, phải được khai báo và mô tả trong phần khai báo của chương trình chính, ngay sau phần khai báo các biến.

Khi cần thực hiện, ta phải viết lệnh gọi thủ tục tương tự như các thủ tục chuẩn.

***b) Ví dụ về thủ tục***

Thủ tục *Ve_Hcn* trong ví dụ trên chỉ vẽ được hình chữ nhật với kích thước cố định là 7×3. Giả sử chương trình cần vẽ nhiều hình chữ nhật với kích thước khác nhau. Để thủ tục *Ve_Hcn* có thể thực hiện được điều đó, cần có hai *tham số* cho dữ liệu vào là chiều dài, chiều rộng và đầu của thủ tục được viết như sau:

**void** Ve_Hcn(int chdai,int chrong);

Khai báo này có nghĩa thủ tục *Ve_Hcn* sẽ được thực hiện để vẽ hình chữ nhật có kích thước tuỳ theo giá trị của các tham số *chdai*, *chrong* và giá trị của các tham số *chdai* và *chrong* là nguyên (i*nt*).

Trong chương trình sau đây mô tả đầy đủ thủ tục *Ve_Hcn* với các tham số *chdai*, *chrong* và sử dụng thủ tục này để vẽ các hình chữ nhật có kích thước khác nhau.

```
#include <iostream>

using namespace std;

void veHCN(int chdai,int chrong);

int main()
{
    cout << "Ve hinh chu nhat" << endl;
    veHCN(3,5);
    cout<<endl;cout<<endl;
    veHCN(4,3);
    cout<<endl;cout<<endl;
    veHCN(5,5);

    return 0;
}
void veHCN(int chdai,int chrong)
{
    for(int i=1;i<=chdai;i++)
    cout<<"*";
    cout<<endl;
    for(int j=1;j<=chrong-2;j++)
    {
        cout<<"*";
        for (int k=1;k<=chdai-2;k++)
            cout<<" ";
        cout<<"*";
        cout<<endl;
    }
```

```
    for(int i=1;i<=chdai;i++)
    cout<<"*";
    cout<<endl;
}
```

*Trong lệnh gọi thủ tục, các tham số hình thức được thay bằng các tham số thực sự tương ứng là các giá trị cụ thể được gọi là các* ***tham số giá trị*** *(gọi tắt là tham trị).*

Các tham số *chdai*, *chrong* của thủ tục *Ve_Hcn* là tham trị. Trong lệnh gọi thủ tục *Ve_Hcn*(5, 3) (vẽ hình chữ nhật kích thước 5×3) tham số *chdai* được thay bởi số nguyên 5, tham số *chrong* được thay bởi số nguyên 3.

Còn trong lời gọi thủ tục *Ve_Hcn(*a, b) vẽ hình chữ nhật kích thước a× b, tham số *chdai* được thay bởi giá trị hiện thời của biến a, tham số *chrong* được thay bởi giá trị hiện thời của biến b.

*Trong lệnh gọi thủ tục, các tham số hình thức được thay bằng các tham số thực sự tương ứng là tên các biến chứa dữ liệu ra được gọi là các* ***tham số biến*** *(gọi tắt là tham biến).*

Để phân biệt tham biến và tham trị, Pascal sử dụng từ khoá ***&*** để khai báo những tham số biến.

Ví dụ, thủ tục *Hoan_doi* trong chương trình sau đổi giá trị của hai biến. Vì cả hai biến đều chứa dữ liệu ra nên cần sử dụng từ khoá ***var*** để khai báo cho cả hai biến.

```
#include <iostream>

using namespace std;
void swap(int &a,int &b);
int main()
{
    int t=7,k=8;
    cout << t<<" "<<k<< endl;
    swap(t,k);
    cout << t<<" "<<k<< endl;
    return 0;
}
void swap(int &a,int &b)
{
    int temp=a;
    a=b;
    b=temp;
}
```

Khai báo sau xác định *t* và *k* là hai tham biến kiểu nguyên:

```
int t=7,k=8;
```

Trong lệnh gọi *swap*(*a*, *b*) các tham biến *t* và *k* được thay thế bởi các biến nguyên tương ứng *a* và *b*.

Giá trị của các biến này bị thay đổi khi thực hiện các lệnh:

```
int temp=a;
a=b;
b=temp;
```

Do vậy, sau khi thực hiện thủ tục *swap*, biến *a* sẽ nhận giá trị của biến *b* và biến *b* nhận giá trị của biến *a*. Nếu giá trị của *a* là 5 còn giá trị của *b* là 10 thì trên màn hình có hai dòng:

```
5 10
10 5
```

Chương trình sau cho thấy sự khác nhau khi tham số chương trình con dùng tham số giá trị và tham số biến (có và không khai báo với từ khoá ***&***):

```
#include <iostream>

using namespace std;
void swap(int a,int &b);
int main()
{
    int t=7,k=8;
    cout << t<<" "<<k<< endl;
    swap(t,k);
    cout << t<<" "<<k<< endl;
    return 0;
}
void swap(int a,int &b)
{
    int temp=a;
    a=b;
    b=temp;
}
```

Khi chương trình thực hiện, sẽ nhận được kết quả:

**7 8**
**7 7**

## 2. Cách viết và sử dụng hàm

Điểm khác nhau cơ bản giữa thủ tục và hàm là việc thực hiện hàm luôn trả về giá trị kết quả thuộc kiểu xác định và giá trị đó được gán cho tên hàm.

Hàm có cấu trúc tương tự như thủ tục, tuy nhiên có khác nhau phần đầu. Khai báo đầu hàm:

*<kiểu dữ liệu> <tên hàm>* ([*<danh sách tham số>*])**;**

*Kiểu dữ liệu* là kiểu dữ liệu của giá trị mà hàm trả về và chỉ có thể là các kiểu *int*, *float*, *char*, *bool*, *string*.

Cũng giống như thủ tục, nếu hàm không có tham số hình thức thì không cần *danh sách tham số*. Trong thân hàm cần có lệnh gán giá trị cho tên hàm:

*return <biểu thức>*;

Xét chương trình thực hiện việc rút gọn một phân số, trong đó có sử dụng hàm tính ước chung lớn nhất (UCLN) của hai số nguyên.

```
#include <iostream>

using namespace std;
int UCLN(int x,int y);

int tuso,mauso,a;

int main()
{
    cout<<"nhap tu so va mau so";
    cin>>tuso>>mauso;
    a=UCLN(tuso,mauso);
    if(a>1)
    {
        tuso/=a;
```

```
        mauso/=a;
    }
    cout<<tuso<<'\t'<<mauso;

    return 0;
}
int UCLN(int x,int y)
{
    int sodu;
    while(y!=0)
    {
        sodu=x%y;
        x=y;
        y=sodu;
    }
    return x;
}
```

***Ghi chú***

Trong chương trình này, các biến *TuSo*, *MauSo* và *a* là các biến toàn cục, còn biến *sodu* là biến cục bộ.

***Sử dụng hàm***

Việc sử dụng hàm *hoàn toàn tương tự* với việc sử dụng các hàm chuẩn, khi viết lệnh gọi gồm tên hàm và tham số thực sự tương ứng với các tham số hình thức.

Lệnh gọi hàm có thể tham gia vào biểu thức như một toán hạng và thậm chí là tham số của lời gọi hàm, thủ tục khác, ví dụ:

```
A= 6*Ucln(Tu,Mau)+1;
```

***Ví dụ***

Chương trình sau cho biết giá trị nhỏ nhất trong ba số nhập từ bàn phím, trong đó có sử dụng hàm tìm số nhỏ nhất trong hai số.

```
#include <iostream>

using namespace std;
float min(float a,float b);
int main()
{
    float a,b,c;
    cout << "Nhap vao 3 so a, b, c" <<endl;
    cin>>a>>b>>c;
    cout<<"gia tri nho nhat la"<<min(min(a,b),c);
    return 0;
}
float min(float a,float b)
{
    if (a<b) return a;
    else return b;
}
```

# BÀI TẬP VÀ THỰC HÀNH 6

## 1. Mục đích, yêu cầu

- *Rèn luyện các thao tác xử lí xâu, kĩ năng tạo hiệu ứng chữ chạy trên màn hình;*
- *Nâng cao kĩ năng viết, sử dụng chương trình con.*

## 2. Nội dung

### Hãy tìm hiểu và xây dựng hai thủ tục sau đây:

- Thủ tục *CatDan*(*S*1, *S*2) nhận đầu vào là xâu *S*1 gồm không quá 79 kí tự, tạo xâu *S*2 thu được từ xâu *S*1 bằng việc chuyển kí tự đầu tiên của nó xuống vị trí cuối cùng. Ví dụ nếu *S*1 = '*abcd*' thì *S*2 = '*bcda*'.
- Thủ tục *CanGiua*(*s*) nhận đầu vào là xâu *s* gồm không quá 79 kí tự, bổ sung vào đầu *s* một số dấu cách để khi đưa ra màn hình xâu kí tự trong *s* ban đầu được căn giữa dòng gồm 80 kí tự.

## BÀI TẬP VÀ THỰC HÀNH 7

### 1. Mục đích, yêu cầu

- *Nâng cao kĩ năng viết, sử dụng chương trình con;*
- *Biết cách viết một chương trình có cấu trúc để giải một bài toán trên máy tính.*

### 2. Nội dung

***a)*** Tìm hiểu việc xây dựng các hàm và thủ tục thực hiện tính độ dài các cạnh, chu vi, diện tích, kiểm tra các tính chất đều, cân, vuông của tam giác được trình bày dưới đây.

Giả thiết tam giác được xác định bởi toạ độ của ba đỉnh. Ta sử dụng kiểu dữ liệu bản ghi để mô tả một tam giác:

***b)*** Tìm hiểu chương trình nhập vào toạ độ ba đỉnh một tam giác và sử dụng các hàm, thủ tục được xây dựng dưới đây để khảo sát các tính chất của tam giác.

***c)*** Viết chương trình sử dụng các hàm và thủ tục xây dựng ở trên để giải bài toán:

Cho tệp dữ liệu TAMGIAC.DAT có cấu trúc như sau:

- Dòng đầu tiên chứa số $N$;
- $N$ dòng tiếp theo, mỗi dòng chứa sáu số thực $x_A$, $y_A$, $x_B$, $y_B$, $x_C$, $y_C$ là toạ độ ba đỉnh $A(x_A, y_A)$, $B(x_B, y_B)$, $C(x_C, y_C)$ của tam giác $ABC$.

Hãy nhập dữ liệu từ tệp đã cho và trong số $N$ tam giác đó, đưa ra tệp TAMGIAC.OUT gồm ba dòng:

- Dòng đầu tiên là số lượng tam giác đều;
- Dòng thứ hai là số lượng tam giác cân (nhưng không là đều);
- Dòng thứ ba là số lượng tam giác vuông.

**TÓM TẮT**

- Chương trình con đóng vai trò quan trọng trong lập trình, đặc biệt trong lập trình có cấu trúc.
- Dùng chương trình con sẽ thuận lợi cho việc tổ chức, viết, kiểm tra chương trình và sử dụng lại.
- Chương trình con có phần đầu, phần khai báo và phần thân.
- Chương trình con có thể có *tham số hình thức* khi khai báo và được thay bằng *tham số thực sự* khi gọi. Các tham số hình thức và thực sự phải tương ứng về thứ tự và kiểu dữ liệu.
- *Gọi chương trình con* bằng tên của nó.
- Biến được khai báo trong chương trình con là *biến cục bộ*.
- Thư viện cung cấp những chương trình con chuẩn mở rộng khả năng ứng dụng.

# CÂU HỎI VÀ BÀI TẬP

**1.** Hãy nêu sự giống nhau và khác nhau giữa thủ tục và hàm.

**2.** Chương trình con có thể không có tham số được không? Cho ví dụ.

**3.** Hãy cho ví dụ chương trình con có nhiều hơn một kết quả ra.

**4.** Viết chương trình con (hàm, thủ tục) tính bội số chung nhỏ nhất của hai số nguyên dương *a*, *b*. Hãy cho biết trong trường hợp này viết chương trình con dưới dạng hàm hay thủ tục là thuận tiện hơn. Vì sao?