TRẦN THỊ KIM OANH
CAO THANH SƠN

Giáo trình

Ngôn ngữ lập trình C

NHÀ XUẤT BẢN ĐẠI HỌC VINH

MỤC LỤC

LỜI GIỚI THIỆU

Ngôn ngữ lập trình C là một ngôn ngữ lập trình phổ biến, được áp dụng trong nhiều lĩnh vực và chuyên ngành khác nhau. Chính vì vậy, ngôn ngữ lập trình C được đưa vào giảng dạy cho sinh viên ngành Công nghệ thông tin cũng như nhiều ngành khoa học kỹ thuật khác ở hầu hết các trường đại học, cao đẳng.

Với kinh nghiệm giảng dạy và tích lũy kiến thức qua nhiều năm với học phần ngôn ngữ lập trình C, đồng thời tham khảo và kế thừa các tài liệu hiện có, nhóm tác giả quyết định biên soạn giáo trình này nhằm phục vụ công tác học tập và giảng dạy được tốt hơn. Nội dung của giáo trình tập trung vào những kiến thức căn bản nhất của ngôn ngữ lập trình C, giúp người học bước đầu dễ dàng tiếp cận với ngôn ngữ lập trình này. Các kiến thức trong giáo trình được trình bày có mối liên hệ chặt chẽ, logic. Các nội dung đưa ra được minh họa cụ thể, trực quan, dễ hiểu, giúp người học có thể tự nghiên cứu. Ngoài ra, hệ thống các câu hỏi, bài tập thực hành và bài tập tự học ở cuối mỗi chương nhằm mục đích giúp người học hệ thống lại các kiến thức đã được học và có thể chủ động trong việc học của mình.

Giáo trình được phục vụ cho đối tượng là sinh viên ngành công nghệ thông tin và các ngành kỹ thuật khác của các trường đại học, cao đẳng; đồng thời giáo trình cũng là tài liệu tham khảo cho giảng

viên giảng dạy học phần ngôn ngữ lập trình C và các học phần có liên quan.

Nhóm tác giả xin được gửi lời cảm ơn chân thành đến PGS.TS. Nguyễn Tân Ân, PGS.TS. Lê Huy Thập, PGS.TS. Lê Khắc Thành, TS. Trần Minh Tước đã đọc bản thảo và có những ý kiến góp ý hết sức quý báu giúp nhóm tác giả hoàn thiện cuốn giáo trình. Nhóm tác giả cũng xin chân thành cảm ơn bộ môn Hệ thống Thông tin và Khoa Công nghệ Thông tin, Trường Đại học Vinh đã tạo điều kiện thuận lợi để chúng tôi tham gia giảng dạy học phần Ngôn ngữ lập trình C cho sinh viên trong nhiều năm, tích lũy được nhiều kinh nghiệm để biên soạn cuốn giáo trình này.

Mặc dù các tác giả đã cố gắng thể hiện nội dung của giáo trình một cách rõ ràng, dễ hiểu nhưng chắc chắn không thể tránh khỏi những thiếu sót, vì vậy nhóm tác giả mong nhận được ý kiến góp ý quý báu của bạn đọc để có thể bổ sung và sửa đổi cho giáo trình được hoàn thiện hơn.

Nghệ An, tháng 1 năm 2015

NHÓM TÁC GIẢ

TỔNG QUAN VỀ GIÁO TRÌNH

Nội dung giáo trình được chia làm 7 chương và 2 phụ lục, chứa đựng khá đầy đủ các vấn đề cơ bản của ngôn ngữ lập trình C. Đầu mỗi chương chỉ rõ các mục tiêu cụ thể cần đạt được của chương, giúp người học định hướng tốt hơn trong việc học cũng như giúp cho giáo viên có thể tham khảo trong quá trình giảng dạy của mình. Cuối mỗi chương có các bài tập thực hành và bài tập tự học giúp sinh viên củng cố lại kiến thức, trau dồi kỹ năng lập trình. Nội dung của giáo trình được trình bày với các nội dung sau:

Chương 1. Giới thiệu lịch sử ra đời và phát triển của ngôn ngữ lập trình C; lý do tại sao lại chọn ngôn ngữ lập trình C; các bước giải quyết vấn đề nhờ máy tính; thuật toán và cách biểu diễn thuật toán; chương trình; quy trình thực hiện chương trình; các hệ đếm và bài tập Chương 1.

Chương 2. Giới thiệu các yếu tố cơ bản của ngôn ngữ lập trình C, bao gồm: từ vựng; các kiểu dữ liệu cơ bản; biến; hằng; biểu thức; các hàm xuất, nhập dữ liệu; câu lệnh; cấu trúc chương trình viết bằng ngôn ngữ lập trình C và bài thực hành Chương 2.

Chương 3. Giới thiệu các cấu trúc điều khiển, bao gồm: cấu trúc tuần tự; cấu trúc rẽ nhánh; cấu trúc lặp; câu lệnh nhảy không điều kiện và bài thực hành Chương 3.

Chương 4. Giới thiệu về con trỏ và mảng, bao gồm: khai báo, sử dụng con trỏ; khai báo, sử dụng mảng một chiều, mảng hai chiều, xâu ký tự; một số thuật toán xử lý mảng, xử lý xâu ký tự và bài thực hành Chương 4.

Chương 5. Giới thiệu về hàm (*chương trình con*) trong ngôn ngữ lập trình C, bao gồm: xây dựng và sử dụng hàm; nguyên mẫu của hàm; truyền tham số cho hàm; hàm `main` có đối số; hàm đệ quy và bài thực hành Chương 5.

Chương 6. Giới thiệu các kiểu dữ liệu tự định nghĩa, bao gồm: kiểu cấu trúc; kiểu hợp; kiểu liệt kê; kiểu được định nghĩa bằng từ khóa `typedef` và bài thực hành Chương 6.

Chương 7. Giới thiệu dữ liệu kiểu tệp tin, bao gồm: phân loại tệp tin trong ngôn ngữ lập trình C; các thao tác đối với tệp tin; các thao tác với tệp tin văn bản; các thao tác với tệp tin nhị phân và bài thực hành Chương 7.

Phụ lục A. Giới thiệu các chỉ thị tiền xử lý, bao gồm: các chỉ thị thay thế; các chỉ thị chèn tệp và các chỉ thị lựa chọn.

Phụ lục B. Giới thiệu bảng mã ASCII.

Chương 1
MỞ ĐẦU

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Lịch sử ra đời và phát triển của ngôn ngữ lập trình C;*
- *Lý do lựa chọn ngôn ngữ lập trình C;*
- *Khái niệm thuật toán, biểu diễn thuật toán;*
- *Khái niệm chương trình;*
- *Các bước giải quyết bài toán nhờ máy tính;*
- *Phong cách lập trình;*
- *Các hệ đếm.*

Các tài liệu tham khảo chính của chương bao gồm [4, 7, 8].

1.1. Lịch sử phát triển ngôn ngữ lập trình C

Trước năm 1970, lập trình hệ thống chủ yếu dựa vào hợp ngữ (*Assembly*), công việc lập trình trong giai đoạn này còn nặng nề và phức tạp, khó chuyển đổi chương trình giữa các hệ máy tính khác nhau. Chính vì điều này, năm 1972 Dennis Ritchie giới thiệu ngôn ngữ lập trình C với mục đích dùng để viết lại hệ điều hành UNIX.

Tiền thân của ngôn ngữ lập trình C là ngôn ngữ BCPL (*Basic Combined Programming Language*) do Martin Richards giới thiệu năm 1976. Ảnh hưởng của ngôn ngữ BCPL đối với ngôn ngữ lập trình C gián tiếp thông qua ngôn ngữ B do Ken Thompson đề xuất năm 1970 cho hệ điều hành UNIX.

Ngôn ngữ lập trình C khác với ngôn ngữ BCPL và ngôn ngữ B ở chỗ: hai ngôn ngữ này có duy nhất một kiểu dữ liệu là *từ máy*, trong khi đó ngôn ngữ lập trình C đã có các kiểu dữ liệu cơ bản như *kiểu ký tự*, *các kiểu số nguyên*, *các kiểu số thực* và *con trỏ*.

Sau khi ra đời, đặc biệt là thành công với hệ điều hành UNIX, ngôn ngữ lập trình C được sử dụng phổ biến để lập trình hệ thống cùng với Assembly cho việc phát triển các ứng dụng (90% mã lệnh của UNIX được viết bằng ngôn ngữ lập trình C). Ngôn ngữ lập trình C còn được gọi là *ngôn ngữ lập trình hệ thống* vì nó hữu ích cho việc viết các trình biên dịch, trình điều khiển thiết bị, hệ điều hành và cũng được sử dụng để viết các chương trình phức tạp cho nhiều lĩnh vực khác nhau.

Đến năm 1978, Brian W. Kernighan và Dennis M. Ritchie xuất bản cuốn sách đầu tiên về ngôn ngữ lập trình C với tiêu đề *The C Programming Language*. Ngôn ngữ lập trình C cũng được chuẩn hóa bởi *Viện Tiêu chuẩn Quốc gia Hoa Kỳ* (*ANSI: American National Standards Institute*) với tên gọi *ANSI C* vào năm 1983. Tổ chức Tiêu chuẩn Quốc tế ISO (International Standard Organization) cũng xây dựng chuẩn cho ngôn ngữ lập trình C, hai chuẩn này giống nhau và biết đến với tên chung là *"ANSI C"*.

1.2. Lý do sử dụng ngôn ngữ lập trình C

Hiện nay, có rất nhiều ngôn ngữ lập trình ra đời, đặc biệt, các ngôn ngữ lập trình trực quan làm cho người lập trình giảm nhiều thời gian trong việc thiết kế cũng như lập trình. Nhưng theo một tổ chức xếp hạng có uy tín TIOBE thì Ngôn ngữ lập trình C là một trong những

ngôn ngữ được xếp hạng cao nhất (01/2015), hoặc luôn nằm trong 3 ngôn ngữ được dùng nhiều nhất. Các thông tin về xếp hạng của các ngôn ngữ lập trình có thể tham khảo tại website *www.tiobe.com*. Các lý do sau đây có thể kể ra tại sao dùng ngôn ngữ lập trình C:

- Ngôn ngữ lập trình C là một ngôn ngữ mềm dẻo, được dùng trong nhiều lĩnh vực khác nhau. Có thể sử dụng ngôn ngữ lập trình C để viết hệ điều hành, các trình điều khiển, soạn thảo văn bản, bảng tính, đồ họa và thậm chí là các chương trình dịch cho các ngôn ngữ khác.
- Ngôn ngữ lập trình C là ngôn ngữ khả chuyển hay còn gọi là dễ thích nghi. Tính dễ thích nghi được hiểu là một chương trình C có thể viết trên nhiều hệ thống máy tính khác nhau và cũng có thể viết trên nhiều nền hệ điều hành khác nhau. Chúng ta có thể lập trình C trên hệ điều hành Linux hay Windows.
- Ngôn ngữ lập trình C là một ngôn ngữ có ít từ khóa (là các từ dùng riêng cho ngôn ngữ khi viết chương trình). Thêm vào đó, ngôn ngữ lập trình C đi kèm với số thư viện khá phong phú.
- Ngôn ngữ lập trình C là một ngôn ngữ có cấu trúc mô đun. Đó chính là việc sử dụng các chương trình con (*hàm*). Các hàm này có thể sử dụng nhiều lần trong một hoặc nhiều chương trình. Điều này làm cho một chương trình C sáng sủa, ngắn gọn và dễ bảo trì.
- Vào những năm 80, do nhu cầu trong việc xử lý dữ liệu ngày một cao, các chương trình viết ra ngày càng phức tạp, việc bảo trì một chương trình ngày một khó khăn dẫn đến một phong cách lập trình mới - lập trình hướng đối tượng (*OOP – Object Oriented Programming*) xuất hiện và ngôn ngữ lập trình C được trang bị thêm khả năng lập trình hướng đối tượng, ngôn ngữ C++ ra đời từ đó và ngày được nhiều lập trình viên quan tâm.

1.3. Các môi trường phát triển tích hợp dùng cho lập trình C

Môi trường phát triển tích hợp (IDE: Integrated Development

Environment) là một loại phần mềm máy tính có công dụng hỗ trợ các lập trình viên trong việc phát triển phần mềm. Một IDE bao gồm:

- Một trình soạn thảo mã (source code editor);
- Trình biên dịch (compiler) và/hoặc trình thông dịch (interpreter);
- Công cụ xây dựng tự động;
- Trình gỡ lỗi (debugger);
- Ngoài ra, IDE còn có thể bao gồm hệ thống quản lí các công cụ nhằm đơn giản hóa công việc xây dựng giao diện người dùng đồ họa (GUI: Graphical User Interface). Nhiều IDE hiện đại còn tích hợp trình duyệt lớp (class browser), trình quản lí đối tượng (object inspector), lược đồ phân cấp lớp (class hierarchy diagram),... để sử dụng trong việc phát triển phần mềm theo hướng đối tượng.

Hiện nay, có rất nhiều IDE của nhiều hãng khác nhau dùng cho lập trình C như Turbo C++, Dev-C++, C-Free, Visual C++, Visual Studio,… Mỗi IDE đều có những ưu điểm và nhược điểm riêng, vì vậy, tùy vào mục đích khác nhau mà lập trình viên lựa chọn IDE cho phù hợp. Trong giáo trình này chúng tôi sử dụng Dev-C++ (phiên bản Dev-C++ 4.9.9.2) để minh họa cho các ví dụ, tuy nhiên các ví dụ này hoàn toàn có thể soạn thảo, biên dịch và chạy trên các IDE khác như Turbo C++, Visual C++,…

1.4. Các bước giải quyết bài toán nhờ máy tính

Việc sử dụng máy tính điện tử để giải quyết một bài toán nào đó là cả một quá trình phức tạp bao gồm nhiều giai đoạn phát triển mà lập trình chỉ là một trong các giai đoạn đó. Cho dù bài toán được đặt ra là lớn hay nhỏ, nếu giải quyết theo các bước cụ thể kết quả sẽ tốt hơn giải quyết bài toán được thực hiện theo một cách không rõ ràng. Các bước quan trọng của việc giải quyết một bài toán sử dụng máy tính điện tử như sau:

Bước 1. Xác định bài toán

Bước đầu tiên là phát biểu chính xác bài toán, làm rõ những yêu cầu đặt ra. Sau đó thiết lập, xác định mối phụ thuộc giữa các dữ kiện và kết quả cần đạt được. Trên cơ sở có được mô hình bài toán, tiếp tục đánh giá, nhận định tính khả thi của của bài toán.

Bước 2. Lựa chọn phương pháp giải

Mỗi bài toán có thể có nhiều cách khác nhau để giải quyết. Các phương pháp có thể khác nhau về thời gian thực hiện, chi phí lưu trữ dữ liệu, độ chính xác,... Nói chung không có phương pháp nào tối ưu về mọi phương diện. Chính vì vậy, tùy theo từng trường hợp cụ thể mà lựa chọn phương pháp thích hợp để giải quyết bài toán đó.

Bước 3. Xây dựng thuật toán hoặc thuật giải

Xây dựng mô hình chặt chẽ, chính xác hơn và chi tiết hơn phương pháp đã lựa chọn. Xác định rõ dữ liệu vào, ra cho các bước thực hiện cơ bản và thứ tự thực hiện các bước cơ bản đó. Nên áp dụng phương pháp thiết kế có cấu trúc, từ thiết kế tổng thể tiến hành làm mịn dần từng bước.

Bước 4. Cài đặt chương trình (Lập trình)

Mô tả thuật giải bằng chương trình. Dựa vào thuật toán đã được xây dựng, căn cứ quy tắc của một ngôn ngữ lập trình để soạn thảo ra chương trình thể hiện thuật giải đã thiết lập ở *Bước 3*.

Bước 5. Hiệu chỉnh chương trình

Trong khi cài đặt chương trình, thường người lập trình không thể kiểm soát hết được những sai sót có thể có trong quá trình thực hiện. Ở bước này, chúng ta cho chương trình chạy thử để phát hiện và sửa các lỗi nếu có.

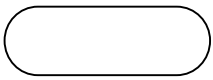
Bước 6. Thực hiện chương trình

Tiến hành chạy chương trình, phân tích kết quả thu được. Việc phân tích nhằm khẳng định kết quả có phù hợp hay không. Nếu không, cần kiểm tra lại toàn bộ các bước đã thực hiện ở trên.

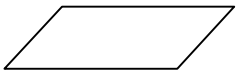
1.5. Biểu diễn thuật toán

Thuật toán được hiểu là một dãy hữu hạn các thao tác để giải quyết một lớp các bài toán sao cho khi có dữ liệu vào xác định thì cho kết quả ra xác định.

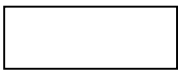
Có nhiều cách biểu diễn thuật toán nhưng cách trực quan nhất cho những người mới học lập trình là sử dụng lưu đồ thuật toán, nghĩa là sử dụng các khối ký hiệu để biểu diễn thuật toán.



Khối Bắt đầu/Kết thúc

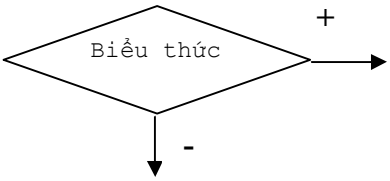


Khối Vào/Ra dữ liệu



Khối thao tác

Khối gọi chương trình con



Khối kiểm tra

Liên kết các khối

Trong đó: Dấu `+` có nghĩa là `Biểu thức` đúng

Dấu `–` có nghĩa là `Biểu thức` sai

Những ký hiệu trên là những ký hiệu cơ bản thường được sử dụng, ngoài ra đối với những thuật toán có lưu đồ lớn, phức tạp có thể dùng thêm các ký hiệu khác.

1.6. Chương trình

Ngôn ngữ dùng để diễn tả thuật toán sao cho máy tính có thể hiểu được gọi là ngôn ngữ lập trình.

Dùng ngôn ngữ lập trình để diễn tả thuật toán thì được chương trình. Chương trình này được gọi là chương trình nguồn. Chương trình dịch sẽ phải dịch chương trình nguồn ra chương trình mã máy. Việc thực hiện chương trình nghĩa là thực hiện chương trình mã máy (gọi là chạy chương trình).

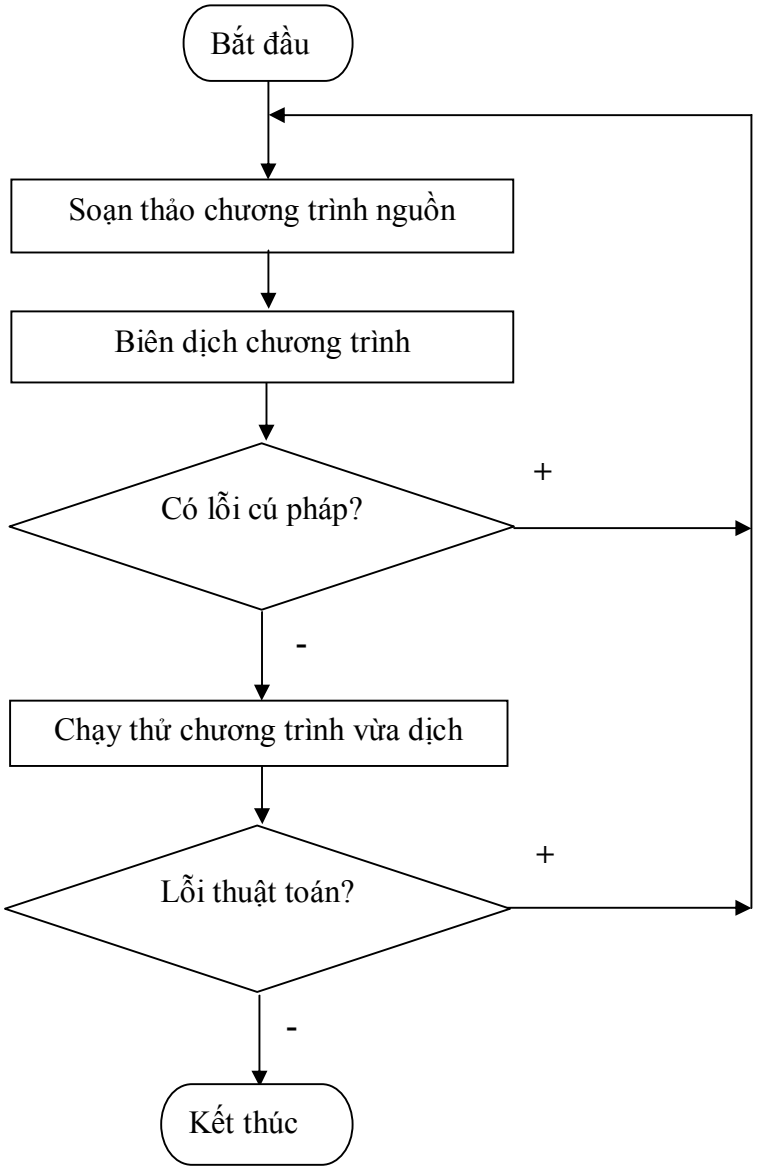
Chương trình nguồn ⇨ Chương trình dịch ⇨ Chương trình mã máy

Chương trình dịch có hai loại là *Trình biên dịch* (*Compiler*) và *Trình thông dịch* (*Interpreter*). Trình biên dịch sẽ dịch toàn bộ chương trình nguồn ra chương trình mã máy rồi mới thực thi. Trình thông dịch sẽ dịch và thực thi ngay lập tức từng lệnh một, muốn chạy lại chương trình thì phải thông dịch lại. Ngôn ngữ lập trình C là loại ngôn ngữ sử dụng trình biên dịch.

Khi dịch và chạy chương trình có thể xuất hiện hai loại lỗi là lỗi cú pháp và lỗi thuật toán.

Lỗi cú pháp (Syntax Error): Lỗi thuộc quy định về mặt cú pháp của ngôn ngữ lập trình như: Lỗi về từ khoá, cách đặt tên biến, cấu trúc lệnh, kiểu dữ liệu,…Tất cả những lỗi này sẽ được bộ biên dịch phát hiện khi biên dịch chương trình.

Lỗi thuật toán (Logic Error): Là những lỗi thuộc trách nhiệm của người lập trình mà bộ biên dịch không thể phát hiện được. Do vậy, yêu cầu khi chương trình chạy và cho kết quả, người lập trình phải biện luận, kiểm tra xem kết quả đã chính xác hay chưa?.



Hình 1.1. Lưu đồ thể hiện quá trình cài đặt (lập trình)

1.7. Phong cách lập trình

Khi học một ngôn ngữ lập trình, thường rất ít khi người học được nghe hay được hướng dẫn về phong cách lập trình. Mặc dù phong cách lập trình không hoàn toàn bắt buộc, tuy nhiên có thể xem phong cách lập trình có một sự liên hệ chặt chẽ và tất yếu đối với công việc lập trình. Có những tiêu chuẩn về phong cách lập trình riêng của từng ngôn ngữ, đôi khi có sự tương đồng vì không có sự khác biệt đáng kể. Phong cách lập trình ngoài mục đích giúp mã nguồn thêm trong sáng, rõ ràng còn để giúp phát huy tính chất làm việc tập thể dựa vào sự nhất quán theo một khuôn mẫu được quy ước. Vì thế, không cần phải gò bó trong một lối viết nào đó, mỗi tập thể có thể linh hoạt tự quy ước cho mình một phong cách lập trình với mức độ nào đó.

Ngoài những ưu điểm trên, phong cách lập trình còn thể hiện trình độ của lập trình viên, một mã nguồn có phong cách lập trình tốt luôn được đánh giá cao hơn một mã nguồn không tuân theo phong cách lập trình, cho dù trình độ giữa hai người viết là tương đương nhau. Để có một phong cách lập trình tốt, người lập trình phải tuân theo các chuẩn thông dụng và phải có chú giải đầy đủ mỗi khi không tuân theo chuẩn.

Phong cách lập trình thường không phân loại, tuy nhiên để trình bày tổng thể một chương trình thường có các quy ước chung như sau:

- Chương trình nên được chia thành nhiều mô đun, mỗi mô đun thực hiện một công việc và càng độc lập càng tốt. Điều này giúp người lập trình không phải nhớ nhiều các lệnh và dễ dàng cho việc bảo trì chương trình.
- Nên sử dụng các tham số khi truyền thông tin giữa các chương trình con. Tránh sử dụng biến toàn cục để truyền thông tin giữa các chương trình con (*được trình bày ở Chương 5*).
- Trình bày chương trình càng nhất quán sẽ càng dễ đọc và dễ hiểu. Mã nguồn nên giữ được tính đơn giản, rõ ràng và trong sáng trong hầu hết các tình huống để thuận tiện cho việc bảo trì chương trình sau này.

- Chương trình nên theo một cấu trúc nhất định, không nên có những thay đổi bất chợt (*do sử dụng* `goto` *hay* `continue` *được trình bày ở Chương 3*).
- Mỗi câu lệnh nên được viết trên một dòng để dễ đọc và dễ quan sát cách thực hiện trong quá trình tìm lỗi.
- Các dấu bắt đầu và kết thúc khối lệnh `{, }` phải được viết gióng theo hàng dọc với nhau. Các câu lệnh cùng cấp phải được viết gióng nhau và thụt vào một khoảng `Tab` so với dấu `{, }` của câu lệnh đó; các câu lệnh là thân của câu lệnh khác phải được viết thụt vào một khoảng `Tab`.
- Các phép toán và toán hạng trong một biểu thức nên cách nhau một khoảng trắng để biểu thức dễ đọc hơn, có khoảng trắng ngăn cách giữa dấu phẩy hay dấu chấm phẩy với các tham số.
- Nên sử dụng cặp ngoặc `( )` khi muốn tránh các lỗi về độ ưu tiên phép toán.
- Mỗi dòng lệnh không nên dài quá 80 ký tự, điều này giúp việc đọc chương trình dễ dàng hơn khi không phải thực hiện thao tác cuộn màn hình. Trong trường hợp dòng lệnh quá dài thì nên ngắt thành nhiều dòng.
- Đặt tên phải thể hiện đúng công dụng hay ý nghĩa của nó, nên thống nhất một cách đặt tên trong từng chương trình, nên có chú thích về ý nghĩa của các tên đã đặt, không nên đặt tên quá dài, không nên dùng một tên cho nhiều mục đích khác nhau.
- Nên viết chú thích ngắn gọn, đầy đủ, dễ hiểu qua việc trình bày chương trình rõ ràng, đơn giản và cách đặt tên hợp lý. Cũng không nên lạm dụng chú thích. Nên viết chú thích cho từng hàm, từng tệp tin.

1.8. Các hệ đếm

1.8.1. Biểu diễn số trong các hệ đếm

Hệ đếm được hiểu gồm tập các ký hiệu và quy tắc sử dụng tập ký hiệu đó để biểu diễn và xác định giá trị các số. Một số tự nhiên X bất kỳ lớn hơn 1 có thể làm cơ số cho hệ đếm cơ số X. Với hệ đếm cơ số X thì số các ký hiệu dùng để biểu diễn các số là X, từ 0 đến X - 1. Mỗi ngôn ngữ lập trình có quy định riêng về việc sử dụng các hệ đếm. Ngôn ngữ lập trình C cho phép sử dụng các hệ đếm sau:

- Hệ đếm cơ số 2 (*hệ nhị phân*): Sử dụng 2 ký hiệu 0, 1 để biểu diễn các số.
- Hệ đếm cơ số 8 (*hệ bát phân*): Sử dụng 8 ký hiệu 0, 1, 2, 3, 4, 5, 6, 7 để biểu diễn các số.
- Hệ đếm cơ số 10 (*hệ thập phân*): Sử dụng 10 ký hiệu 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 để biểu diễn các số.
- Hệ đếm cơ số 16 (*hệ thập lục phân hay hệ hexa*): Sử dụng 16 ký hiệu 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 và A, B, C, D, E, F (hoặc a, b, c, d, e, f) để biểu diễn các số.

Để biểu diễn một số trong hệ đếm cơ số X ta có thể sử dụng dạng biểu diễn tổng quát hoặc dạng biểu diễn khai triển.

- **Dạng tổng quát**

$a_n a_{n-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-m}$

***Trong đó*:** Các a_i ($i = 0, 1, \ldots, n$) và a_j ($j = -m, -m + 1, \ldots, -1$) là các ký hiệu trong hệ đếm tương ứng. Các chữ số 0, 1 trong hệ đếm cơ số 2, các chữ số 0, 1, 2, 3, 4, 5, 6, 7 trong hệ đếm cơ số 8, các chữ số 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 và A, B, C, D, E, F (hoặc a, b, c, d, e, f) trong hệ đếm cơ số 16.

Để biểu diễn một số theo dạng tổng quát ở cơ số nào ta ghi chỉ số của cơ số ở phía dưới bên phải của số đó, đối với hệ đếm cơ số 16 có thể

ghi chỉ số là chữ h thay cho số 16. Riêng các số ở hệ đếm cơ số 10 thì ngầm định, không cần ghi chỉ số.

Chẳng hạn:

10	biểu diễn số ở hệ đếm cơ số 10
1010_2	biểu diễn số ở hệ đếm cơ số 2
43_8	biểu diễn số ở hệ đếm cơ số 8
$5F_h$	biểu diễn số ở hệ đếm cơ số 16

- **Dạng khai triển**

$$a_n \times X^n + a_{n-1} \times X^{n-1} + \ldots + a_0 \times X^0 + a_{-1} \times X^{-1} + \ldots + a_{-m} \times X^{-m}$$

***Trong đó*:** X là cơ số của hệ đếm.

Thập phân	Nhị phân	Bát phân	Thập lục phân
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A (hoặc a)
11	1011	13	B (hoặc b)
12	1100	14	C (hoặc c)
13	1101	15	D (hoặc d)
14	1110	16	E (hoặc e)
15	1111	17	F (hoặc f)

Bảng 1.1. Biểu diễn các số từ 0 đến 15 trong các hệ đếm cơ số 10, 2, 8, 16 tương ứng

1.8.2. Cách chuyển đổi giữa các hệ đếm

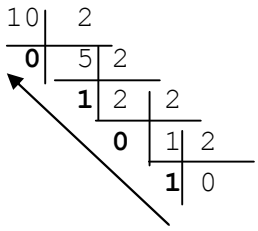
a. Đổi một số từ hệ đếm cơ số 10 sang hệ đếm cơ số 2, 8, 16

Muốn đổi một số nguyên từ hệ đếm cơ số 10 sang hệ đếm cơ số 2 (hay 8, 16) ta lấy số đó chia cho cơ số 2 (hay 8, 16) lấy phần dư, sau đó lại lấy thương tiếp tục chia cho cơ số lấy phần dư … và cứ tiếp tục như vậy cho đến khi thương bằng `0`. Dãy các số dư lấy theo thứ tự ngược lại chính là số tương ứng của số đó trong hệ đếm cơ số 2 (hay 8, 16).

Muốn đổi một số thực từ hệ đếm cơ số 10 sang hệ đếm cơ số 2 (hay 8, 16) trước hết ta đổi phần nguyên theo cách đổi số nguyên, còn phần thập phân ta làm như sau: Lấy phần thập phân nhân với cơ số được kết quả, lấy phần nguyên, sau đó lại lấy phần thập phân nhân với cơ số được kết quả, lấy phần nguyên… và cứ tiếp tục như vậy cho đến khi kết quả phép nhân bằng `0` hoặc có sự lặp lại của phần thập phân.

- **Đổi số nguyên từ hệ đếm cơ số 10 sang hệ đếm cơ số 2**

Ví dụ 1.1. Đổi số `10` ở hệ đếm cơ số 10 sang hệ đếm cơ số 2.



Kết quả: $10 = 1010_2$

- **Đổi số thực từ hệ đếm cơ số 10 sang hệ đếm cơ số 2**

Ví dụ 1.2. Đổi số `10.6875` từ hệ đếm cơ số 10 sang hệ đếm cơ số 2.

+ Trước hết ta đổi số nguyên `10` sang hệ đếm cơ số 2, ta có:

$10 = 1010_2$ (*Ví dụ 1.1*)

+ Tiếp theo, ta đổi `0.6875` sang hệ đếm cơ số 2.

Thực hiện phép nhân	Kết quả	Phần nguyên
0.6875 × 2	1.375	1
0.375 × 2	0.75	0
0.75 × 2	1.5	1
0.5 × 2	1.0	1

$0.6875 = 0.1011_2$

Kết quả: $10.6875 = 1010.1011_2$

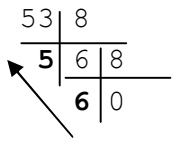
Ví dụ 1.3. Đổi số 0.4 từ hệ đếm cơ số 10 sang hệ đếm cơ số 2.

Thực hiện phép nhân	Kết quả	Phần nguyên
0.4 × 2	0.8	0
0.8 × 2	1.6	1
0.6 × 2	1.2	1
0.2 × 2	0.4	0
… (*quá trình lặp lại*)		

Kết quả: $0.4 = 0.(0110)_2$

- **Đổi số nguyên từ hệ đếm cơ số 10 sang hệ đếm cơ số 8**

Ví dụ 1.4. Đổi số 53 từ hệ đếm cơ số 10 sang hệ đếm cơ số 8.



Kết quả: $53 = 65_8$

- **Đổi số thực từ hệ đếm cơ số 10 sang hệ đếm cơ số 8**

Ví dụ 1.5. Đổi số 53.3125 từ hệ đếm cơ số 10 sang hệ đếm cơ số 8.

+ Trước hết ta đổi số nguyên 53 từ hệ đếm cơ số 10 sang hệ đếm cơ số 8.

$53 = 65_8$ (*Ví dụ 1.4*)

+ Tiếp đến, ta đổi phần thực 0.3125 từ hệ đếm cơ số 10 sang hệ đếm cơ số 8.

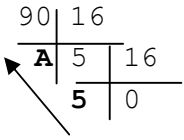
Thực hiện phép nhân	Kết quả	Phần nguyên
0.3125 × 8	2.5	2 ↓
0.5 × 8	4.0	4

$0.3125 = 0.24_8$

Kết quả: $53.3125 = 65.24_8$

- **Đổi số nguyên từ hệ đếm cơ số 10 sang hệ đếm cơ số 16**

Ví dụ 1.6. Đổi số 90 từ hệ đếm cơ số 10 sang hệ đếm cơ số 16.



Kết quả: $90 = 5A_h$

- **Đổi số thực từ hệ đếm cơ số 10 sang hệ đếm cơ số 16**

Ví dụ 1.7. Đổi số thực 90.142578 từ hệ đếm cơ số 10 sang hệ đếm cơ số 16.

+ Trước hết ta đổi số nguyên 90 từ hệ đếm cơ số 10 sang hệ đếm cơ số 16.

$90 = 5A_h$ (*Ví dụ 1.6*)

+ Tiếp theo, ta đổi phần thực 0.142578 từ hệ đếm cơ số 10 sang hệ đếm cơ số 16.

Thực hiện phép nhân	Kết quả	Phần nguyên
0.142578 × 16	2.28125	2 ↓
0.28125 × 16	4.5	4
0.5 × 16	8.0	8

$0.142578 = 0.248_h$

Kết quả: $90.142578 = 5A.248_h$

b. Đổi số từ hệ đếm cơ số 2, 8, 16 sang hệ đếm cơ số 10

Muốn đổi một số từ hệ đếm cơ số 2 (hay 8, 16) sang hệ đếm cơ số 10 ta biểu diễn số đó dưới dạng khai triển, sau đó, tính giá trị của biểu thức khai triển. Giá trị tính được chính là số tương ứng của số đó trong hệ đếm cơ số 10.

Ví dụ 1.8.

$a = 1010_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 10$

$b = 34_8 = 3 \times 8^1 + 4 \times 8^0 = 28$

$c = 5A_h = 5 \times 16^1 + 10 \times 16^0 = 90$

c. Đổi trực tiếp một số giữa hệ đếm cơ số 2 và hệ đếm cơ số 8, hệ đếm cơ số 2 và hệ đếm cơ số 16

Việc chuyển đổi một số qua lại giữa hệ đếm cơ số X và hệ đếm cơ số Y có thể được thực hiện thông qua trung gian là hệ đếm cơ số 10. Tuy nhiên, việc chuyển đổi một số qua lại giữa hệ đếm cơ số X và hệ đếm cơ số Y mà $Y = X^n$ hoặc $X = Y^n$ có thể được thực hiện trực tiếp. Như vậy, việc chuyển đổi một số qua lại giữa hệ đếm cơ số 2 và hệ đếm cơ số 8, hệ đếm cơ số 2 và hệ đếm cơ số 16 có thể được thực hiện trực tiếp vì $8 = 2^3$ và $16 = 2^4$.

- **Đổi trực tiếp một số từ hệ đếm cơ số 2 sang hệ đếm cơ số 8**

Muốn đổi trực tiếp một số từ hệ đếm cơ số 2 sang hệ đếm cơ số 8 ta nhóm từng nhóm ba chữ số từ phải sang trái và lần lượt đổi từng nhóm sang hệ đếm cơ số 8 (*Bảng 1.1*). Nếu số các chữ số không phải là bội của 3 ta chỉ cần thêm các chữ số 0 vào phía bên trái của số đó để số chữ số trở thành bội của 3.

Ví dụ 1.9.

$a = 10100010_2 = 10\ 100\ 010_2 = \mathbf{0}10\ 100\ 010 = 242_8$

- **Đổi trực tiếp một số từ hệ đếm cơ số 8 sang hệ đếm cơ số 2**

Muốn đổi một số từ hệ đếm cơ số 8 sang hệ đếm cơ số 2 ta đổi lần lượt từng chữ số của số đó sang số ở hệ đếm cơ số 2 tương ứng (*Bảng 1.1*).

Ví dụ 1.10. $a = 43_8 = 100\ 011_2 = 100011_2$

- **Đổi trực tiếp một số từ hệ đếm cơ số 2 sang hệ đếm cơ số 16**

Muốn đổi trực tiếp một số từ hệ đếm cơ số 2 sang hệ đếm cơ số 16 ta nhóm từng nhóm bốn chữ số từ phải sang trái và lần lượt đổi từng nhóm sang hệ đếm cơ số 16 (*Bảng 1.1*). Nếu số các chữ số không phải là bội của 4 ta chỉ cần thêm các chữ số 0 vào phía bên trái của số đó để số chữ số trở thành bội của 4.

Ví dụ 1.11. $a = 0101111101_2 = 01\ 0111\ 1101_2$

$= \mathbf{00}01\ 0111\ 1101_2 = 17D_h$

- **Đổi trực tiếp một số từ hệ đếm cơ số 16 sang hệ đếm cơ số 2**

Muốn đổi một số từ hệ đếm cơ số 16 sang hệ đếm cơ số 2 ta đổi lần lượt từng chữ số của số đó sang số ở hệ đếm cơ số 2 tương ứng (*Bảng 1.1*).

Ví dụ 1.12. $a = 5A_h = 0101\ 1010_2 = 01011010_2$

1.8.3. Các phép toán trên các hệ đếm

Các phép toán trên các hệ đếm cơ số 2, 8, 16 cũng được thực hiện theo quy tắc tương tự như đối với hệ đếm cơ số 10.

Ví dụ 1.13.

Phép cộng

111_2	128_8	$5A_h$
$+\ 10_2$	$+\ 52_8$	$+\ 2B_h$
1001_2	202_8	85_h

Phép trừ

101_2	143_8	$5A_h$
$-\ 10_2$	$-\ 75_8$	$-\ 2B_h$
11_2	66_8	$2F_h$

Phép nhân

$$
\begin{array}{r}
101_2 \\
*\ 11_2 \\
\hline
101_2 \\
101_2 \\
\hline
1111_2
\end{array}
$$

Phép chia

```
 1000001 |1101
-  1101  |101
 000110  |
 - 0000  |
  0001101|
   - 1101|
  0000000|
```

BÀI TẬP CHƯƠNG 1

- **Mục tiêu**

Qua việc giải quyết các bài tập người học cần đạt được:

- **Về kiến thức**
  - + Nắm được tổng quan các bước để có thể giải quyết bài toán nhờ máy tính;
  - + Nắm được khái niệm thuật toán, cách biểu diễn thuật toán bằng lưu đồ;
  - + Nắm được cách biểu diễn số trong hệ đếm cơ sơ 2, hệ đếm cơ số 8, hệ đếm cơ số 10 và hệ đếm cơ số 16;
  - + Chuyển đổi một số qua lại giữa các hệ đếm;
  - + Nguyên tắc thực hiện các phép toán cộng, trừ, nhân, chia trên các hệ đếm.
- **Về kỹ năng**
  - + Vận dụng được kiến thức trên để xây dựng được thuật toán của một số bài toán đơn giản;
  - + Có thể chuyển đổi số nguyên hoặc số thực qua lại giữa các hệ đếm;
  - + Thực hiện được các phép toán cộng, trừ, nhân, chia trên các hệ đếm.

- **Nội dung**

Bài 1. Vẽ lưu đồ thuật toán của các bài toán sau:

a. Tìm số lớn nhất trong hai số `a` và `b`.

b. Tìm số lớn nhất trong ba số `a`, `b` và `c`.

c. Giải phương trình `ax + b = 0`.

Bài 2. Đổi các số sau đây từ hệ đếm cơ số 10 sang hệ đếm cơ số 2

a. 100

b. 255

c. 10.125

Bài 3. Đổi các số sau đây từ hệ đếm cơ số 10 sang hệ đếm cơ số 8

a. 256

b. 1568

c. 12.265625

Bài 4. Đổi các số sau đây từ hệ đếm cơ số 10 sang hệ đếm cơ số 16

a. 100

b. 255

c. 10.28125

Bài 5. Đổi các số sau đây từ hệ đếm cơ số 2 sang hệ đếm cơ số 10

a. 10010001_2

b. 1100111101_2

c. 11001010.101_2

Bài 6. Đổi các số sau đây từ hệ đếm cơ số 8 sang hệ đếm cơ số 10

a. 72_8

b. 437_8

c. 25.04_8

Bài 7. Đổi các số sau đây từ hệ đếm cơ số 16 sang hệ đếm cơ số 10

a. $17F_h$

b. FF_h

c. 10.6_h

Bài 8. Đổi trực tiếp các số sau đây từ hệ đếm cơ số 2 sang hệ đếm cơ số 16

a. 10011101_2

b. 110101110_2

c. 10101100001_2

Bài 9. Đổi trực tiếp các số sau đây từ hệ đếm cơ số 16 sang hệ đếm cơ số 2

a. $2F_h$

b. $3DA_h$

c. $A5B_h$

Bài 10. Thực hiện các phép toán sau:

a. $11011_2 + 1001_2$ $1101_2 - 110_2$

$1101_2 \times 111_2$ $101101_2 : 11_2$

b. $462_8 + 27_8$ $72_8 - 46_8$

$24_8 \times 17_8$ $144_8 : 24_8$

c. $5A_h + 2C_h$ $26A_h - 7B_h$

$FF_h \times 4C_h$ $FF_h : 5_h$

Chương 2

CÁC YẾU TỐ CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH C

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Bộ từ vựng của ngôn ngữ lập trình C;*
- *Khái niệm biến, hằng; phân biệt biến, hằng;*
- *Cách đặt tên trong ngôn ngữ lập trình C;*
- *Biểu thức và các phép toán;*
- *Các hàm nhập, xuất dữ liệu;*
- *Khái niệm câu lệnh;*
- *Cấu trúc chương trình viết bằng ngôn ngữ lập trình C.*

Các tài liệu tham khảo chính của chương bao gồm [1, 2, 4, 7, 8].

2.1. Từ vựng

2.1.1. Tập ký tự

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được kết hợp với nhau để tạo nên các từ. Các từ lại được kết hợp với nhau theo quy tắc nào đó để tạo nên các câu lệnh. Ngôn ngữ lập trình C cho phép sử dụng các ký tự sau đây:

- 26 chữ cái in hoa `A .. Z`
- 26 chữ cái in thường `a .. z`

- 10 chữ số từ 0 .. 9
- Các ký hiệu toán học + - * / = > < ( )
- Ký tự gạch nối _
- Các ký tự đặc biệt . , ; : [ ] { } | ! \ & % # $ ' " ? ~ ^
- Dấu cách (Space), dấu nhảy cách (Tab),…

Như vậy, các ký tự không nằm trong bộ ký tự trên sẽ không được sử dụng trong ngôn ngữ lập trình C.

Chẳng hạn: Khi giải phương trình bậc hai $ax^2 + bx + c = 0$ ta thường ký hiệu biểu thức $\triangle = b \times b - 4 \times a \times c$, nếu viết chương trình này bằng ngôn ngữ lập trình C thì kí tự $\triangle$ không được sử dụng mà phải thay bằng ký tự khác có trong bộ ký tự của ngôn ngữ lập trình C.

2.1.2. Tên

Tên là một định danh, dùng để đặt cho biến, hằng, nhãn, hàm, mảng, con trỏ, tệp, … trong chương trình. Trong Dev-C++, độ dài tối đa ngầm định của tên là 32, tuy nhiên người lập trình có thể thay đổi giá trị này trong phạm vi từ 1 đến 32.

Tên bao gồm 3 loại: từ khóa, tên chuẩn, tên do người lập trình định nghĩa.

- **Quy tắc của tên**

- Là một dãy các ký tự có thể bao gồm chữ cái, chữ số và dấu gạch nối;
- Ký tự đầu tiên của tên phải là chữ cái hoặc dấu gạch nối;
- Không được đặt tên trùng với từ khoá.

Ví dụ 2.1.

- Các tên đúng: x1, a_1, delta, _delta
- Các tên sai:

3x	sai vì tên bắt đầu bằng chữ số
r#3	sai vì tên có ký tự #
f(x)	sai vì tên có hai ký tự ( )
case	sai vì tên trùng với từ khóa
del ta	sai vì tên có dấu cách

2.1.3. Từ khoá

Từ khóa là các từ dùng riêng với ý nghĩa và tác dụng cụ thể được từng ngôn ngữ lập trình quy định. Từ khóa không thể định nghĩa lại. Ý nghĩa, tác dụng của các từ khóa trong ngôn ngữ lập trình C sẽ được giới thiệu trong các phần tiếp theo của giáo trình. Sau đây là một số từ khóa thường dùng:

asm	break	case	char	const
continue	default	define	do	double
else	enum	extern	far	float
for	goto	huge	if	include
int	interrupt	long	near	pascal
register	return	short	static	struct
signed	sizeof	switch	typedef	union
unsigned	void	volatile	while	

✍ **Lưu ý:**

- Trong chương trình C từ khoá luôn luôn phải viết thường;
- Ngôn ngữ lập trình C phân biệt chữ hoa và chữ thường. Như vậy, biến `a` và biến `A` là hai biến khác nhau.

2.1.4. Lời chú thích

Trong khi lập trình có thể thêm vào các chú thích để giải thích ý nghĩa của biến, hằng, hàm, câu lệnh,…làm cho chương trình rõ ràng, dễ hiểu, dễ nhớ và thuận lợi cho việc sau này bản thân người lập trình hay người khác đọc hiểu chương trình. Lời chú thích được chương trình dịch bỏ qua khi biên dịch chương trình. Lời chú thích trong ngôn ngữ lập trình C được quy định như sau:

```
/* Lời chú thích có thể bao gồm một
   hay nhiều dòng */
```

hoặc `// Lời chú thích cho đến hết dòng`

2.2. Các kiểu dữ liệu cơ bản

Ngôn ngữ lập trình C có ba kiểu dữ liệu cơ bản là: kiểu ký tự, kiểu số nguyên và kiểu số thực. Các bảng sau thể hiện tên kiểu, số `byte` cần thiết để biểu diễn mỗi giá trị và miền giá trị của kiểu.

2.2.1. Kiểu ký tự

TT	Tên kiểu	Số byte	Miền giá trị
1	char(signed char)	1	-128 .. 127
2	unsigned char	1	0 .. 255

Bảng 2.1. Các kiểu dữ liệu ký tự

2.2.2. Kiểu số nguyên

TT	Tên kiểu	Số byte	Miền giá trị
1	int	2	-32768 .. 32767
2	unsigned int	2	0 .. 65535
3	long (long int)	4	-2147483648 .. 2147483647
4	unsigned long	4	0.. 4294967295

Bảng 2.2. Các kiểu dữ liệu số nguyên

✍ **Lưu ý:** Trong ngôn ngữ lập trình C kiểu ký tự có thể được sử dụng như kiểu số nguyên. *Chẳng hạn:* Kết quả của biểu thức `'A' + 'a'` là `65 + 97 = 162`.

2.2.3. Kiểu số thực

TT	Tên kiểu	Số byte	Miền giá trị
1	float	4	3.4e-38 .. 3.4e+38
2	double	8	1.7e-308 .. 1.7e+308
3	long double	10	3.4e-4932 .. 1.1e+4932

Bảng 2.3. Các kiểu dữ liệu số thực

Trong đó: Cách viết số thực dùng ký hiệu `e` hoặc `E` là cách viết số thực theo dạng khoa học (*hay dạng dấu chấm động*).

Chẳng hạn: `3.4e-38` (hay `3.4E-38`) biểu diễn số thực có giá trị 3.4×10^{-38}.

Máy tính có thể lưu trữ được các số kiểu `float` có giá trị tuyệt đối từ `3.4e-38` đến `3.4e+38`, các số ngoài phạm vi trên được xem bằng `0`. Tương tự cho các kiểu `double` và `long double`.

2.3. Biến

Biến là một đại lượng do người lập trình định nghĩa và đặt tên theo quy tắc đặt tên của từng ngôn ngữ lập trình. Biến dùng để chứa dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể thay đổi trong quá trình này. Trong ngôn ngữ lập trình C, biến phải được khai báo trước khi sử dụng. Mỗi biến thuộc về một kiểu dữ liệu xác định và nhận giá trị thuộc kiểu đó.

Khai báo: `<Kiểu> <Danh sách biến>;`

Trong đó:

- `Kiểu` có thể là các kiểu dữ liệu cơ bản hoặc kiểu dữ liệu do người lập trình tự định nghĩa;
- `Danh sách biến` có thể là một biến hay nhiều biến. Nếu có nhiều hơn một biến, các biến phải cách nhau bởi dấu phẩy (`,`);
- Kết thúc khai báo bằng dấu chấm phẩy (`;`).

Ví dụ 2.2.

```
int n;              // Khai báo biến n có kiểu int
float a, b, c;      // Khai báo ba biến a, b, c có kiểu float
```

Có thể vừa khai báo vừa khởi tạo giá trị ban đầu cho biến

```
int n = 10;  /* Khai báo biến n có kiểu int và khởi tạo giá trị
                bằng 10 */
```

```
float a = 3.14, b = 123.45, c; /* Khai báo ba biến a, b,
                                  c và khởi tạo giá trị cho a bằng 3.14, cho b bằng
                                  123.45, còn c không khởi tạo giá trị */
```

2.4. Hằng

Hằng là đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán của chương trình. Trong ngôn ngữ lập trình C có các loại hằng sau: hằng số nguyên, hằng số thực, hằng ký tự, hằng xâu ký tự.

Khai báo: **`#define`** `<Tên hằng> <Giá trị>`

2.4.1. Hằng số nguyên

Các hằng loại này có thể khai báo dưới dạng số ở hệ đếm cơ số 10, hệ đếm cơ số 8 hay hệ đếm cơ số 16. Nếu khai báo hằng ở hệ đếm cơ số 10 thì số được viết bình thường, nếu khai báo ở hệ đếm cơ số 8 thì thêm ký hiệu `0` ở đầu, nếu khai báo ở hệ đếm cơ số 16 thì thêm ký hiệu `0x` hoặc `0X` ở đầu.

Ví dụ 2.3.

```
#define   n 100    /* Khai báo hằng n ở hệ đếm cơ số 10, có
                      giá trị 100 */
#define   m 0144   /* Khai báo hằng m ở hệ đếm cơ số 8, có giá
                      trị 100 ở hệ đếm cơ số 10 */
#define   t 0x64   /* Khai báo hằng t ở hệ đếm cơ số 16, có
                      giá trị 100 ở hệ đếm cơ số 10 */
```

Nếu muốn khai báo hằng có kiểu dữ liệu là `long` thì thêm ký hiệu `L` hoặc `l` vào sau số đó. Nếu khai báo một hằng `int` mà vượt quá phạm vi thì nó hiểu đó là hằng `long`.

Ví dụ 2.4.

```
#define a 33000L  // Khai báo hằng long giá trị 33000
#define b 33000   // Cũng được hiểu là hằng long giá trị 33000
#define c 0x18L   // Khai báo hằng long ở hệ cơ số 16
```

2.4.2. Hằng số thực

Hằng số thực có thể khai báo dạng thập phân (*dạng dấu chấm tĩnh*) hoặc dạng khoa học (*dạng dấu chấm động*).

Ví dụ 2.5.

```
#define    x    3.14        // Dạng thập phân
#define    y    314e-2      // Dạng khoa học
```

hoặc

```
#define    y    314E-2      // Dạng khoa học
```

2.4.3. Hằng ký tự

Là một ký tự được viết trong cặp dấu nháy đơn, chẳng hạn `'a'`. Giá trị của hằng ký tự chính là mã ASCII (*xem phụ lục B*) của ký tự đó. Vì vậy, hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác.

Ví dụ 2.6.

```
#define      ch    'a'   /* Hằng ký tự ch có giá trị 97 (mã
                          ASCII của ký tự a) */
```

Hằng	Ký tự / Chức năng
`'\''`	'
`'\"'`	"
`'\\'`	\
`'\n'`	Xuống dòng
`'\t'`	Tab
`'\b'`	BackSpace
`'\r'`	Về đầu dòng
`'\f'`	Sang trang
`'\0'`	NULL

Bảng 2.4. Một số hằng ký tự đặc biệt

✍ **Lưu ý:** Cần phân biệt hằng `'0'` và hằng `'\0'`. Hằng `'0'` có mã ASCII là 48, còn hằng `'\0'` có mã ASCII là 0.

2.4.4. Hằng xâu ký tự

Hằng xâu ký tự là dãy ký tự bất kỳ đặt trong cặp dấu nháy kép.

Ví dụ 2.7.

```
#define s   "Hello"    /* Khai báo hằng xâu ký tự s có giá trị
                          Hello */
#define t   ""         // Khai báo t là hằng xâu ký tự rỗng
```

Hằng xâu ký tự được lưu trữ trong máy bởi một dãy các `byte` liên tiếp nhau, mỗi `byte` lưu trữ một ký tự riêng biệt của xâu. Trình biên dịch tự động thêm ký tự `'\0'` (`NULL`) vào cuối xâu để báo hiệu kết thúc xâu.

✍ **Lưu ý:** Cần phân biệt giữa `'a'` và `"a"` bởi vì `'a'` là hằng ký tự được lưu trữ trong 1 `byte`, còn `"a"` là hằng xâu ký tự được lưu trữ trong 2 `byte`, trong đó 1 `byte` lưu ký tự a và 1 byte lưu hằng `NULL`.

2.5. Biểu thức

Biểu thức là sự kết hợp giữa phép toán và toán hạng nhằm biểu diễn một công thức nào đó. Trong đó, toán hạng có thể là biến, hằng, hàm, phần tử mảng, biểu thức. Phép toán bao gồm: phép gán, các phép toán số học, các phép toán quan hệ, các phép toán logic, các phép toán thao tác bit, các phép toán tăng, giảm, phép toán điều kiện, phép toán lấy địa chỉ, phép chuyển đổi kiểu, các phép gán mở rộng,…

2.5.1. Phép gán

Ngôn ngữ lập trình C sử dụng phép gán = để tạo biểu thức gán đơn giản.

Cú pháp: `<Biến> = <Biểu thức>;`

Chẳng hạn:

```
int i;
i = 5 + 3;     /* Gán giá trị biểu thức 5 + 3 cho
                  biến i */
```

2.5.2. Các phép toán số học

Ngôn ngữ lập trình C cung cấp các phép toán số học, bao gồm phép toán một ngôi `-` và các phép toán hai ngôi `+`, `-`, `*`, `/`, `%` dùng để tạo các biểu thức số học. Các phép toán này có thể áp dụng cho dữ liệu kiểu ký tự, kiểu số nguyên, kiểu số thực.

TT	Phép toán	Ý nghĩa	Cách sử dụng	Ghi chú
1	`+`	Cộng	`a + b`	
2	`-`	Trừ	`a - b`	
3	`*`	Nhân	`a * b`	
4	`/`	Chia	`a / b`	Nếu cả `a` và `b` đều là số nguyên thì `a / b` trở thành phép chia lấy phần nguyên.
5	`%`	Chia dư	`a % b`	Phép `%` chỉ áp dụng cho kiểu số nguyên.
6	`-`	Đảo dấu	`-a`	Đảo dấu giá trị của a.

Bảng 2.5. Ý nghĩa và cách sử dụng các phép toán số học

Ví dụ 2.8. Giá trị của biểu thức `10.0 / 3` là `3.3333`.

Giá trị của biểu thức `10 / 3` là `3`.

Giá trị của biểu thức `10 % 3` là `1`.

2.5.3. Các phép toán quan hệ

Ngôn ngữ lập trình C cung cấp các phép toán quan hệ, bao gồm các phép toán hai ngôi `>`, `>=`, `<`, `<=`, `==`, `!=` dùng để tạo biểu thức quan hệ. Nếu biểu thức đúng thì giá trị của biểu thức là `1`, ngược lại, giá trị của biểu thức là `0`.

TT	Phép toán	Ý nghĩa	Cách sử dụng
1	>	So sánh lớn hơn	`a > b`
2	>=	So sánh lớn hơn hoặc bằng	`a >= b`
3	<	So sánh bé hơn	`a < b`
4	<=	So sánh bé hơn hoặc bằng	`a <= b`
5	==	So sánh bằng	`a == b`
6	!=	So sánh khác	`a != b`

Bảng 2.6. Ý nghĩa và cách sử dụng các phép toán quan hệ

Ví dụ 2.9. Giá trị của biểu thức `5 > 10` là `0`.

Giá trị của biểu thức `4 > -1` là `1`.

✍ **Lưu ý:** Các biểu thức quan hệ có thể tham gia vào các phép toán đối với số nguyên.

Chẳng hạn: Giá trị của biểu thức `(3 > 2) + 10` là `11`.

2.5.4. Các phép toán logic

Ngôn ngữ lập trình C cung cấp các phép toán logic bao gồm hai phép toán hai ngôi `&&`, `||` và một phép toán một ngôi `!` để tạo các biểu thức logic.

TT	Phép toán	Ý nghĩa	Cách sử dụng
1	`&&`	Và logic	`a && b`
2	`||`	Hoặc logic	`a || b`
3	!	Phủ định	`!a`

Bảng 2.7. Ý nghĩa và cách sử dụng các phép toán logic

a	b	a && b	a || b	!a
Khác 0	Khác 0	1	1	0
Khác 0	Bằng 0	0	1	0
Bằng 0	Khác 0	0	1	1
Bằng 0	Bằng 0	0	0	1

Bảng 2.8. Giá trị chân lý của các phép toán logic

Ví dụ 2.10.

Giá trị của biểu thức `(5 > 4) && (9 > 4)` là `1`.

Giá trị của biểu thức `3 && 8` là `1`.

Giá trị của biểu thức `!5` là `0`.

Giá trị của biểu thức `!0` là `1`.

2.5.5. Các phép toán thao tác bit

Ngôn ngữ lập trình C cung cấp các phép toán thao tác với các bit bao gồm năm phép toán hai ngôi `&`, `|`, `^`, `<<`, `>>` và một phép toán một ngôi `~`. Các phép toán này chỉ áp dụng cho kiểu số nguyên.

TT	Phép toán	Ý nghĩa	Cách sử dụng	Ghi chú
1	`&`	Và theo bit	`a & b`	
2	`|`	Hoặc theo bit	`a | b`	
3	`^`	Hoặc loại trừ theo bit	`a ^ b`	
4	`~`	Lấy phần bù theo bit	`~a`	
5	`<<`	Dịch trái	`a << n`	`n` nguyên dương
6	`>>`	Dịch phải	`a >> n`	`n` nguyên dương

Bảng 2.9. Ý nghĩa và cách sử dụng các phép toán thao tác bit

1 & 1 = 1	1 | 1 = 1	1 ^ 1 = 0	~1 = 0	a << n = a * 2^n
1 & 0 = 0	1 | 0 = 1	1 ^ 0 = 1	~0 = 1	a >> n = a / 2^n
0 & 1 = 0	0 | 1 = 1	0 ^ 1 = 1		
0 & 0 = 0	0 | 0 = 0	0 ^ 0 = 0		

Bảng 2.10. Giá trị chân lý của các phép toán thao tác bit

Ví dụ 2.11.

```
unsigned char a = 10, b = 4;
```

Khi đó `a, b` được biểu diễn nhị phân 8 bit như sau:

a	0	0	0	0	1	0	1	0
b	0	0	0	0	0	1	0	0

Kết quả của các phép thao tác bit đối với `a, b`:

a & b	0	0	0	0	0	0	0	0
a | b	0	0	0	0	1	1	1	0
a ^ b	0	0	0	0	1	1	1	0
~a	1	1	1	1	0	1	0	1
a << 1	0	0	0	1	0	1	0	0
a >> 1	0	0	0	0	0	1	0	1

2.5.6. Biểu thức gán mở rộng

Ngôn ngữ lập trình C cung cấp các phép gán mở rộng `+=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=` để tạo các biểu thức gán mở rộng thay cho một số biểu thức gán đơn giản, nhằm giúp người lập trình tiết kiệm thời gian soạn thảo chương trình.

Ví dụ 2.12. `int x = 2;`

Biểu thức gán `x = x + 2;` có thể thay bằng `x += 2;`

Biểu thức gán `y = y - 5;` có thể thay bằng `y -= 5;`

Biểu thức gán `z = z * (t + 3);` có thể thay bằng `z *= t + 3;`

Biểu thức gán đơn giản	Biểu thức gán mở rộng
x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x % y	x %= y
x = x << y	x <<= y
x = x >> y	x >>= y
x = x & y	x &= y
x = x | y	x |= y
x = x ^ y	x ^= y

Bảng 2.11. Các phép gán mở rộng

2.5.7. Các phép toán tăng, giảm

Ngôn ngữ lập trình C cung cấp hai phép toán một ngôi ++ và -- để tăng hay giảm giá trị của biến lên hay xuống một đơn vị.

Để tăng giá trị của biến a lên một đơn vị ta viết: a++ hoặc ++a

Để giảm giá trị của biến a xuống một đơn vị ta viết: a-- hoặc --a

✍ **Lưu ý:** Nếu phép toán ++ hay -- sử dụng độc lập thì việc đặt phép toán trước hay sau biến đều có ý nghĩa như nhau. Nhưng khi kết hợp các phép toán ++, -- vào trong biểu thức thì việc đặt phép toán trước hay sau biến lại có ý nghĩa khác nhau. Cụ thể:

Nếu viết a++: Sử dụng a cho biểu thức sau đó mới tăng a lên một đơn vị.

Nếu viết ++a: Tăng a lên một đơn vị rồi mới sử dụng a cho biểu thức.

Tương tự đối với a-- và --a.

Ví dụ 2.13. `int a = 6, x;`

Nếu gán `x = a++;` // Kết quả: `x` có giá trị là `6`, `a` có giá trị là `7`

Nếu gán `x = ++a;` // Kết quả: `x` có giá trị là `7`, `a` có giá trị là `7`

2.5.8. Biểu thức điều kiện

Ngôn ngữ lập trình C cung cấp phép toán ba ngôi `?:` để tạo biểu thức điều kiện.

Cú pháp: `Điều kiện ? Biểu thức 1 : Biểu thức 2;`

Ý nghĩa: Nếu `Điều kiện` đúng thì giá trị của biểu thức điều kiện là giá trị của `Biểu thức 1`, ngược lại, giá trị của biểu thức điều kiện là giá trị của `Biểu thức 2`.

Ví dụ 2.14. Tìm số lớn nhất trong hai số `a` và `b`

```
max = a > b ? a : b;
```

/* Giá trị của biểu thức điều kiện `a > b ? a : b` chính là số lớn nhất trong hai số `a` và `b`. Giá trị này được gán cho biến `max`, do đó `max` là số lớn nhất trong hai số `a` và `b` */

2.5.9. Phép toán lấy địa chỉ của biến

Ngôn ngữ lập trình C cung cấp phép toán một ngôi `&` cho phép lấy địa chỉ của một biến.

Cú pháp: `&<Tên biến>`

Ví dụ 2.15. `int a;`

`&a` là địa chỉ của biến `a`.

2.5.10. Phép toán chuyển đổi kiểu

a. Phép chuyển đổi kiểu trong biểu thức

Hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu "*thấp hơn*" sẽ chuyển thành kiểu "*cao hơn*" và đó chính là kiểu kết quả.

Quy tắc chuyển đổi kiểu trong biểu thức

- Nếu một toán hạng có kiểu `long double`, các toán hạng khác cũng được chuyển sang `long double`;

- Trường hợp không có toán hạng nào có kiểu `long double`, nếu một toán hạng có kiểu `double`, các toán hạng khác được chuyển sang `double`;
- Trường hợp không có toán hạng nào có kiểu `long double`, `double`, nếu một toán hạng có kiểu `float`, các toán hạng khác được chuyển sang `float`;
- Nếu các toán hạng có cùng kiểu số nguyên có dấu, hoặc số nguyên không dấu, toán hạng có kiểu "*thấp hơn*" được chuyển đổi sang kiểu "*cao hơn*". Kiểu số nguyên có dấu được xếp theo thứ tự từ thấp lên cao: `char`, `short`, `int`, `long`. Tương ứng theo thứ tự này cho kiểu số nguyên không dấu;
- Nếu miền giá trị của kiểu số nguyên có dấu có thể chứa được kiểu số nguyên không dấu, toán hạng không dấu được chuyển sang kiểu nguyên có dấu.

Chẳng hạn: Giá trị của biểu thức `1.5 * (8 / 3)` là `3.0`.

b. Phép chuyển đổi kiểu thông qua phép gán

Trong một phép gán, giá trị vế phải được chuyển sang kiểu của biến ở vế trái, đó là kiểu kết quả.

Ví dụ 2.16.
```
int n = 3.14;          // n có giá trị là 3
float x = 8 / 3;       // x có giá trị là 2.0
```

c. Phép ép kiểu

Cú pháp: `(<Kiểu>)<Biểu thức>`

Ví dụ 2.17.
```
float a;
```
`(int)a` sẽ có giá trị nguyên, nhưng `a` vẫn có kiểu `float`

Giá trị của biểu thức `(float)8 / 3` là `2.666667`.

2.5.11. Phép toán gộp

Ngôn ngữ lập trình C cung cấp phép toán gộp (`,`), là phép toán hai ngôi dùng để tạo biểu thức gộp, trong đó các toán hạng là các biểu

thức, thứ tự của các toán hạng được tính từ trái qua phải. Kiểu và giá trị của biểu thức gộp chính là kiểu và giá trị của toán hạng cuối cùng bên phải.

Ví dụ 2.18.
```
int a, b, c;
c = (a = 2, b = a + 3); /* Kết quả: a bằng 2, b bằng 5, c bằng 5 */
```

2.5.12. Thứ tự ưu tiên các phép toán

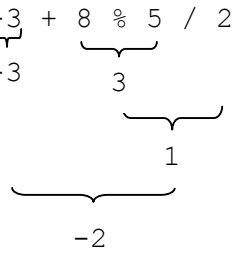
Trong ngôn ngữ lập trình C tất cả các phép toán đều được quy định thứ tự ưu tiên và thứ tự kết hợp. Vì vậy, trong một biểu thức có sử dụng nhiều phép toán hay một phép toán nhiều lần thì thứ tự thực hiện các phép toán phụ thuộc vào thứ tự ưu tiên và thứ tự kết hợp của chúng.

Thứ tự ưu tiên	Các phép toán	Thứ tự kết hợp
1	`() [] -> .`	trái qua phải
2	`! * & ~ ++ -- + - (type) sizeof`	phải qua trái
3	`* / %`	trái qua phải
4	`+ -`	trái qua phải
5	`<< >>`	trái qua phải
6	`< <= > >=`	trái qua phải
7	`== !=`	trái qua phải
8	`&`	trái qua phải
9	`^`	trái qua phải
10	`|`	trái qua phải
11	`&&`	trái qua phải
12	`||`	trái qua phải
13	`? :`	phải qua trái
14	`= *= /= %= += -= <<= >>= &= |= ^=`	phải qua trái
15	`,`	trái qua phải

Bảng 2.12. Thứ tự ưu tiên và thứ tự kết hợp của các phép toán

✍ **Lưu ý:** Các phép toán +, -, * một ngôi có thứ tự ưu tiên cao hơn các phép toán +, -, * hai ngôi.

Ví dụ 2.19. Giả sử với biểu thức `-3 + 8 % 5 / 2`, thứ tự thực hiện các phép toán được thể hiện như sau:



2.6. Các hàm xuất, nhập dữ liệu

Hàm là khái niệm có ý nghĩa quan trọng trong ngôn ngữ lập trình C. Hàm được xem là một đơn vị độc lập thực hiện một chức năng xác định trong chương trình. Hàm có thể do người lập trình tự xây dựng, cũng có thể được xây dựng sẵn trong các thư viện chuẩn của ngôn ngữ lập trình C. Trong chương trình, trước khi sử dụng hàm xây dựng sẵn nào ta phải khai báo thư viện chứa hàm đó (*thường khai báo ở đầu chương trình*). Toàn bộ kiến thức về hàm được trình bày trong Chương 5, phần này chỉ giới thiệu ý nghĩa và cách sử dụng một số hàm xây dựng sẵn dùng để xuất, nhập dữ liệu.

2.6.1. Hàm `printf`

Hàm `printf` dùng để xuất dữ liệu ra màn hình, được khai báo trong thư viện `stdio.h`.

Cú pháp:

```
printf("<Chuỗi định dạng>"[, <Mục 1>, <Mục 2>, ...]);
```

Trong đó:

- `Chuỗi định dạng` có thể bao gồm các mã định dạng và các đoạn văn bản. Mã định dạng dùng để xác định số mục và cách biểu thị giá trị các mục này lên màn hình (*Bảng 2.13*); đoạn văn bản dùng để làm sáng tỏ ý nghĩa kết quả hiện ra;
- Các `Mục i` (`i = 1, 2, ...`) có thể là biến, hằng, hàm, phần tử mảng, biểu thức;
- Có thể đưa vào phần chuỗi định dạng các ký tự đặc biệt để hiển thị lên màn hình hoặc thực hiện các chức năng.

▪ **Cách xuất dữ liệu có quy cách**

Ngôn ngữ lập trình C cung cấp các quy cách xuất dữ liệu nhằm làm cho kết quả chương trình được thể hiện khoa học, rõ ràng, dễ đọc, dễ theo dõi.

- **Đối với ký tự, số nguyên, xâu ký tự**

`%[m]mã`	Dành `m` vị trí để hiển thị giá trị, căn lề phải.
`%[-m]mã`	Dành `m` vị trí để hiển thị giá trị, căn lề trái.

- **Đối với kiểu số thực**

`%[m.n]mã`	Dành `m` vị trí để hiển thị giá trị, `n` chữ số sau phần thập phân, căn lề phải.
`%[-m.n]mã`	Dành `m` vị trí để hiển thị giá trị, `n` chữ số sau phần thập phân, căn lề trái.
`%[.n]mã`	Dành đủ vị trí để hiển thị giá trị, `n` chữ số sau phần thập phân.

Trong đó: `m, n` là các số nguyên dương.

TT	Mã định dạng	Ý nghĩa
1	%c	Xuất ký tự.
2	%d, %i	Xuất số nguyên int.
3	%u	Xuất số nguyên unsigned int.
4	%ld, %li	Xuất số nguyên long.
5	%lu	Xuất số nguyên unsigned long.
6	%o	Xuất số nguyên hệ đếm cơ số 8.
7	%x	Xuất số nguyên hệ đếm cơ số 16 với các ký hiệu a, b, c, d, e, f.
8	%X	Xuất số nguyên hệ đếm cơ số 16 với các ký hiệu A, B, C, D, E, F.
9	%lo	Xuất số nguyên long hệ đếm cơ số 8.
10	%lx	Xuất số nguyên long hệ đếm cơ số 16 với các ký hiệu a, b, c, d, e, f.
11	%lX	Xuất số nguyên long hệ đếm cơ số 16 với các ký hiệu A, B, C, D, E, F.
12	%f	Xuất số thực float dạng thập phân.
13	%e hoặc %E	Xuất số thực float dưới dạng khoa học.
14	%lf	Xuất số thực double, long double dạng thập phân.
15	%le hoặc %lE	Xuất số thực double, long double dạng khoa học.
16	%g hoặc %G	Xuất số thực dạng thập phân hay dạng khoa học tùy vào giá trị cần xuất.
17	%s	Xuất xâu ký tự.
18	%p	Xuất địa chỉ dạng XXXX:YYYY hoặc YYYY.

Bảng 2.13. Danh sách mã định dạng xuất

Ví dụ 2.20.

```
int n = 5;
float x = 3.14;
char ch = 'A';
```

Nếu xuất ra màn hình: `printf("\n%d\n%f\n%c", n, x, ch);`

Kết quả trên màn hình như sau:

```
5
3.140000
A
```

Nếu xuất ra màn hình: `printf("\n%5d\n%5.2f\n%5c", n, x, ch);`

Kết quả trên màn hình như sau:

```
    5
 3.14
    A
```

2.6.2. Hàm `scanf`

Hàm `scanf` dùng để nhập dữ liệu vào từ bàn phím, được khai báo trong thư viện `stdio.h`.

Cú pháp:

`scanf``("<Chuỗi định dạng>", &<Biến 1>[,&<Biến 2>,...]);`

Trong đó: `Chuỗi định dạng` gồm các mã định dạng quy định cách nhập giá trị cho các biến.

Ví dụ 2.21.

```
int n;
scanf("%d", &n);
```

```
/* Nhập số nguyên int vào cho biến n, kết thúc nhập bằng Enter. Nếu nhập 23ab thì n chỉ nhận giá trị là 23. Nếu nhập 12.34 thì n chỉ nhận giá trị là 12 */
float a, b, c;
scanf("%f%f%f", &a, &b, &c);
/* Nhập ba số thực float. Các số cách nhau bởi ít nhất một dấu cách trống hoặc ít nhất một dấu Enter. Kết thúc nhập bằng Enter */
int ngay, thang, nam;
scanf("%d/%d/%d", &ngay, &thang, &nam);
/* Nhập vào ngày, tháng, năm theo định dạng ngay/thang/nam. Chẳng hạn: 10/11/2012, kết thúc nhập bằng Enter */
int ngay, thang, nam;
scanf("%d%*c%d%*c%d", &ngay, &thang, &nam);
/* Nhập vào ngày, tháng, năm theo định dạng phân cách /, -, :,... (trừ ký tự số). Chẳng hạn: 10/11/2012 hoặc 10-11-2012, kết thúc nhập bằng Enter */
int ngay, thang, nam;
scanf("%2d%2d%4d", &ngay, &thang, &nam);
/* Nhập vào ngày, tháng, năm theo dạng dd/mm/yyyy. Chẳng hạn: 01/04/2012 */
```

2.6.3. Hàm `getchar, getch`

- **Hàm `getchar`**

Hàm `getchar` dùng để nhập một ký tự từ bàn phím. Hàm được khai báo trong thư viện `conio.h`. Giá trị của hàm là mã ASCII của ký tự nhập vào. Ký tự nhập vào sẽ được hiển thị lên màn hình.

TT	Mã định dạng	Ý nghĩa
1	%c	Nhập ký tự.
2	%d	Nhập số nguyên int.
3	%u	Nhập số nguyên unsigned int.
4	%ld, %D	Nhập số nguyên long.
5	%lu, %U	Nhập số nguyên unsigned long.
6	%o	Nhập số nguyên hệ đếm cơ số 8.
7	%x	Nhập số nguyên hệ đếm cơ số 16.
8	%lo, %O	Nhập số nguyên long hệ đếm cơ số 8.
9	%lx, %X	Nhập số nguyên long hệ đếm cơ số 16.
10	%f hoặc %e hoặc E	Nhập số thực float dạng thập phân hoặc dạng khoa học.
11	%lf hoặc %le hoặc %lE	Nhập số thực double, long double dạng thập phân hoặc dạng khoa học.

Bảng 2.14. Danh sách mã định dạng nhập

Ví dụ 2.22.

```
int c;
c = getchar();
/* Nếu ta nhập vào ký tự A thì A hiển thị lên màn hình,
   giá trị của c là 65 */
```

- **Hàm `getch`**

Hàm `getch` dùng để lấy một ký tự có sẵn trong bộ đệm bàn phím. Nếu bộ đệm rỗng hàm sẽ cho phép nhập một ký tự từ bàn phím và tự kết thúc nhập (*nghĩa là không cần bấm* `Enter` *để kết thúc nhập*). Hàm được khai báo trong thư viện `conio.h`. Giá trị của hàm bằng mã ASCII của ký tự nhập vào. Ký tự nhập vào không hiển thị lên màn

hình. Hàm `getch` thường được dùng trước khi kết thúc một chương trình C hoặc trong các chương trình tạo mật khẩu.

Ví dụ 2.23.

```
int c;
c = getch();
/* Nếu bộ đệm bàn phím khác rỗng thì hàm lấy một kí tự
trong bộ đệm, ngược lại  hàm cho phép nhập một ký tự
vào từ bàn phím. Chẳng hạn, nhập vào ký tự A thì A
không hiển thị lên màn hình, giá trị của c là 65 */
```

2.7. Khái niệm câu lệnh

Một câu lệnh xác định một công việc mà chương trình phải thực hiện. Mỗi câu lệnh được kết thúc bởi dấu chấm phẩy (`;`). Trong ngôn ngữ lập trình C có các loại câu lệnh sau:

- Câu lệnh đơn;
- Câu lệnh có cấu trúc;
- Câu lệnh ghép.

- ***Câu lệnh đơn***

Câu lệnh đơn là câu lệnh không chứa các câu lệnh khác. Các câu lệnh đơn bao gồm: câu lệnh gán; câu lệnh gọi chương trình con, chẳng hạn như gọi hàm xuất dữ liệu ra màn hình `printf`, gọi hàm nhập dữ liệu vào từ bàn phím `scanf`,...

- ***Câu lệnh có cấu trúc***

Câu lệnh có cấu trúc là câu lệnh trong đó có chứa các câu lệnh khác. Câu lệnh có cấu trúc được tạo bởi các cấu trúc lập trình bao gồm: cấu trúc rẽ nhánh, cấu trúc tuyển chọn, cấu trúc lặp.

- ***Câu lệnh ghép (khối lệnh)***

Câu lệnh ghép là nhóm các câu lệnh được gom vào một khối và đặt giữa cặp dấu ngoặc `{}`. Không có dấu chấm phẩy sau ngoặc đóng `}`.

2.8. Cấu trúc chương trình viết bằng ngôn ngữ lập trình C

Một chương trình viết bằng ngôn ngữ lập trình C có thể bao gồm các phần sau:

***Phần 1*:** Các chỉ thị tiền xử lý.

```
#include
#define
 ...
```

Phần này hầu như có và thường đặt ở đầu chương trình.

***Phần 2*:** Định nghĩa kiểu dữ liệu, biến toàn cục.

Phần này có thể có hoặc không.

***Phần 3*:** Khai báo nguyên mẫu của các hàm do người lập trình xây dựng.

Phần này có thể có hoặc không.

***Phần 4*:** Khai báo và định nghĩa hàm `main`.

Phần này bắt buộc phải có.

***Phần 5*:** Định nghĩa các hàm đã khai báo nguyên mẫu trong phần 3.

Phần này có thể có hoặc không.

Ví dụ 2.24. Viết chương trình hiển thị lên màn hình câu chào `Hello World`.

```
#include <stdio.h>   /* Thu vien chua cac ham
                     vao/ra du lieu */
#include <conio.h>   /* Thu vien chua cac ham xu ly
                     man hinh */
int main()
{
   printf("\nHello World");  /* Hien thi cau chao
                              Hello World */
   getch();        // Dung man hinh
   return 0;       // Tra lai gia tri cho ham main
}
```

- ***Soạn thảo chương trình nguồn*:** Vào màn hình soạn thảo của Dev-C++ và gõ vào chương trình trên. Việc soạn thảo chương trình cần tuân theo phong cách lập trình đã nêu trong Chương 1 (*Mục 1.7*).

- ***Biên dịch chương trình*:** Bấm tổ hợp phím Ctrl + F9 (hoặc sử dụng chức năng Execute\Compile).

- ***Chạy chương trình*:** Bấm tổ hợp phím Ctrl + F10 (hoặc sử dụng chức năng Execute\Run).

✍ **Lưu ý:** Sau khi soạn thảo một chương trình C, để có thể biên dịch ta cần lưu lại chương trình nguồn với phần mở rộng là *.C hoặc *.CPP.

BÀI THỰC HÀNH CHƯƠNG 2

- **Mục tiêu**

Qua bài thực hành người học cần đạt được:

- **Về kiến thức**

  + Nắm được cách xây dựng, biểu diễn thuật toán một số bài toán đơn giản;
  + Nắm được cấu trúc của một chương trình viết bằng ngôn ngữ lập trình C;
  + Thông qua lập trình hiểu được những khái niệm cơ bản của ngôn ngữ lập trình C;
  + Phân biệt được lỗi cú pháp, lỗi thuật toán.

- **Về kỹ năng**

  + Biết cách xây dựng thuật toán của một số bài toán đơn giản, sử dụng ngôn ngữ lập trình C để viết thành chương trình;
  + Trình bày chương trình theo cấu trúc;
  + Biên dịch, chạy chương trình; sửa lỗi cú pháp, lỗi thuật toán của các chương trình này;
  + Sử dụng được một số chức năng cơ bản trong môi trường làm việc của Dev-C++ hoặc các IDE khác như Turbo C++, Visual C++, C-Free,…

- **Nội dung**

Bài 1. Viết chương trình in ra màn hình nội dung và hình thức như sau:

```
Truong Dai hoc Vinh
Khoa Cong nghe Thong tin
Nganh: Ky su tin hoc        Lop: 55K
```

Bài 2. Mở Dev-C++ và gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int a, b;
   printf("Nhap hai so a, b = ");
   scanf("%d%d", &a, &b);
   printf("\nTong cua %5d va %5d la %5d", a, b, a + b);
   printf("\nHieu cua %5d va %5d la %5d", a, b, a - b);
   printf("\nTich cua %5d va %5d la %5d", a, b, a * b);
   printf("\nThuong cua %5d va %5d la %5.2f", a, b,
   (float)a / b);
   getch();
   return 0;
}
```

Yêu cầu:

+ Cho biết: số biến, tên các biến, kiểu của các biến, từ khóa, câu lệnh, thư viện có trong chương trình.
+ Chương trình thực hiện công việc gì?
+ Biên dịch, chạy chương trình.

Bài 3. Mở Dev-C++ và gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    float a, b, max;
    printf("Nhap hai so a, b = ");
    scanf("%f%f", &a, &b);
```

```
    Max = a > b ? a : b;
    printf("\nSo lon nhat cua %5.2f va %5.2f la:
            %5.2f", a, b, max);
    getch();
    return 0;
}
```

Yêu cầu:

+ Biên dịch chương trình, phát hiện lỗi cú pháp và sửa chúng (nếu có);

+ Chương trình thực hiện công việc gì?

+ Biên dịch và chạy chương trình.

Bài 4. Mở Dev-C++ và gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    float r, cv, dt;
    printf("Nhap r  = ");
    scanf("%f", &r);
    cv = M_PI * r;
    dt = M_PI * r * r;
    printf("\nBan kinh = %5.2f", r);
    printf("\nChu vi = %5.2f", cv);
    printf("\nDien tich = %5.2f", dt);
    getch();
    return 0;
}
```

Yêu cầu:

+ Biên dịch, chạy chương trình. Kiểm tra kết quả;

+ Phát hiện lỗi, sửa lỗi;

+ Biên dịch và chạy lại chương trình.

Bài 5. Căn cứ vào mức độ ưu tiên của các phép toán, tính giá trị của các biểu thức sau. Viết chương trình kiểm tra lại kết quả.

a. `x = -10 + 2 * 3 + 4 % 5 / 2`

b.
```
y = 0
!y % 4
```

c.
```
z = 3
!++z % 4
```

d.
```
t = 3
!(++t % 4)
```

- **Câu hỏi củng cố**
  - Nêu quy tắc đặt tên trong ngôn ngữ lập trình C.
  - Vì sao phải nhớ các từ khóa của ngôn ngữ lập trình C?
  - Trong chương trình C, hai biến `Delta` và `DELTA` có phải là một không?
  - Trong biểu thức gán, vế trái có thể là một biểu thức được không?
  - Lỗi cú pháp xảy ra ở giai đoạn nào và lỗi thuật toán xảy ra ở giai đoạn nào khi lập trình? Bộ biên dịch phát hiện được loại lỗi nào?
- **Tự học**

Bài 6. Cho biết kết quả của đoạn chương trình sau và giải thích tại sao lại đạt kết quả đó? Viết chương trình hoàn chỉnh và kiểm tra lại kết quả.

```
unsigned char a, b, c;
a = 200;
b = 60;
c = a + b;
printf("\nKet qua c = %d", c);
```

Bài 7. Cho hai biến số nguyên `a = 4, b = 6`. Cho biết kết quả của `a, b, n` trong các phép toán tăng, giảm sau. Viết chương trình kiểm tra lại kết quả.

```
n = a + b
n = ++a + b
n = a++ + b
n = --a + b
n = a-- + b
n = a + b
```

Bài 8. Gán giá trị bất kỳ cho hai biến nguyên `i, j` và cho biết kết quả của các biểu thức sau. Viết chương trình để kiểm tra lại kết quả.

```
i > j       i < j
i >= j      i <= j
i == j      i != j
```

Bài 9. Gán giá trị bất kỳ cho hai biến nguyên `i, j` và cho biết kết quả của các biểu thức sau. Viết chương trình để kiểm tra lại kết quả.

```
!i          !j          i && j          i || j
~i          ~j          i & j           i | j
i ^ j       i << 1      i >>1           j << 1
j >> 1
```

Bài 10. Viết chương trình nhập vào từ bàn phím hai số thực `a, b`. Tính chu vi và diện tích hình chữ nhật có hai cạnh là `a, b` và thông báo lên màn hình:

```
Chu vi hinh chu nhat     =
Dien tich hinh chu nhat  =
```

Chương 3

CÁC CẤU TRÚC ĐIỀU KHIỂN

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

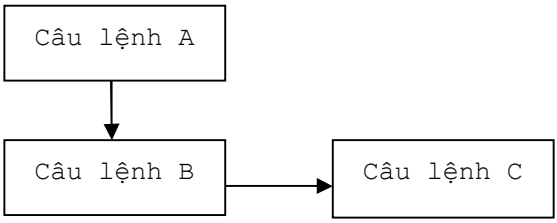
- *Cú pháp và cách thức hoạt động của các câu lệnh có cấu trúc* `if`, `switch`*;*
- *Chuyển đổi giữa* `if` *và* `switch`*;*
- *Cú pháp và cách thức hoạt động của các câu lệnh* `for`*,* `while`*,* `do while`*;*
- *Chuyển đổi giữa* `for`*,* `while` *và* `do while`*;*
- *Cách sử dụng các câu lệnh có cấu trúc lồng nhau;*
- *Câu lệnh nhảy không điều kiện.*

Các tài liệu tham khảo chính của chương bao gồm [1, 6, 7, 8].

Cấu trúc điều khiển của một ngôn ngữ lập trình chỉ ra thứ tự thực hiện của các câu lệnh trong chương trình viết bằng ngôn ngữ đó. Ngôn ngữ lập trình C cung cấp các cấu trúc điều khiển: cấu trúc tuần tự, cấu trúc rẽ nhánh, câu lệnh nhảy không điều kiện, cấu trúc tuyển chọn và cấu trúc lặp.

3.1. Cấu trúc tuần tự

Ngôn ngữ lập trình C quy định các câu lệnh trong chương trình được thực hiện *từ trên xuống dưới* và *từ trái sang phải*, trừ trường hợp gặp câu lệnh rẽ nhánh, câu lệnh lặp hay các câu lệnh `goto`, `return`, `break`, `continue`, `exit`.



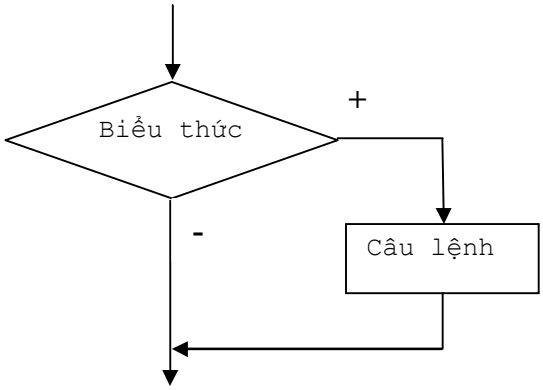
3.2. Cấu trúc rẽ nhánh

3.2.1. Cấu trúc rẽ nhánh dạng khuyết: câu lệnh `if`

Cú pháp:

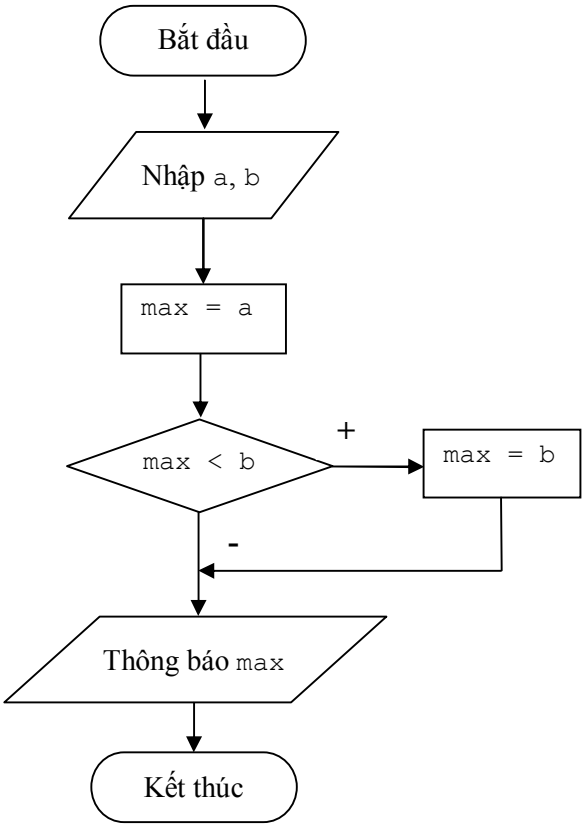
```
if (<Biểu thức>)
    <Câu lệnh>;
```

Sơ đồ hoạt động:



Ví dụ 3.1. Vẽ lưu đồ thuật toán và viết chương trình tìm số lớn hơn trong hai số `a, b` sử dụng câu lệnh rẽ nhánh dạng khuyết.

Lưu đồ thuật toán:



Chương trình:

```
#include <conio.h>
#include <stdio.h>
int main()
{
      int a, b, max;
      printf("\nNhap hai so a, b: ");
      scanf("%d%d", &a, &b);
      max = a;
```

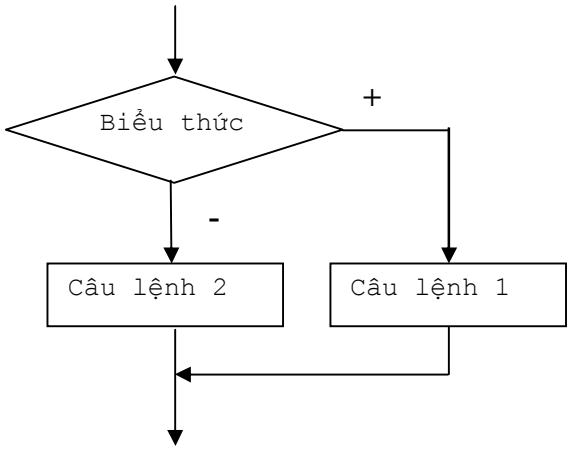
```
    if (max < b)
        max = b;
    printf("\nSo lon hon trong hai so %5d va %5d la:
            %5d", a, b, max);
    getch();
    return 0;
}
```

3.2.2. Cấu trúc rẽ nhánh dạng đủ: câu lệnh `if else`

Cú pháp:

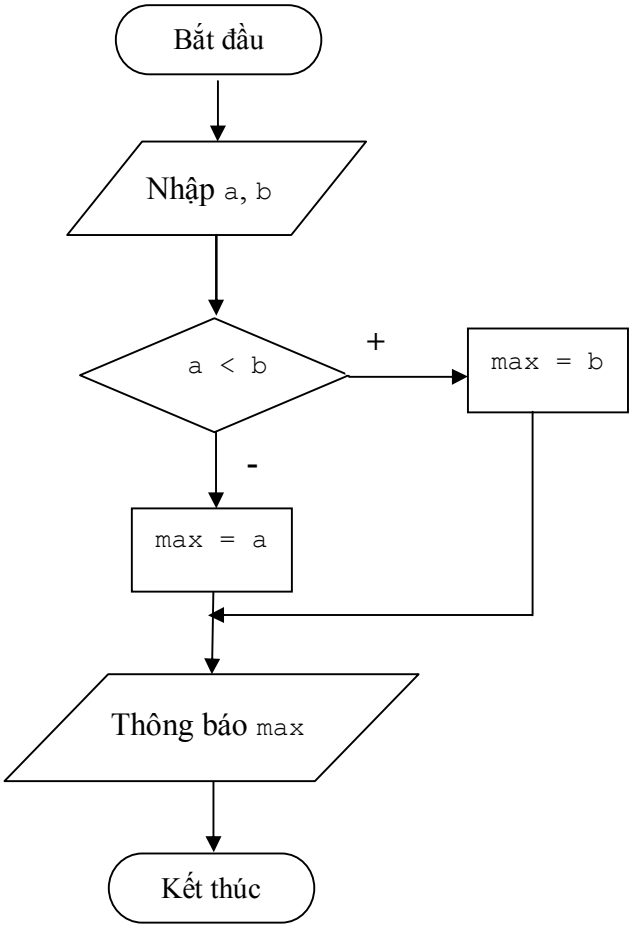
```
if (<Biểu thức>)
    <Câu lệnh 1>;
else
    <Câu lệnh 2>;
```

Sơ đồ hoạt động:



Ví dụ 3.2. Vẽ lưu đồ thuật toán và viết chương trình tìm số lớn hơn trong hai số `a, b` sử dụng câu lệnh rẽ nhánh dạng đủ.

Lưu đồ thuật toán:



Chương trình:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int a, b, max;
    printf("\nNhap hai so a, b: ");
    scanf("%d%d", &a, &b);
```

```
    if (a < b)
            max = b;
    else
            max = a;
    printf("\nSo lon hon trong hai so %5d va %5d la:
             %5d", a, b, max);
    getch();
    return 0;
}
```

Trong chương trình có thể sử dụng các câu lệnh rẽ nhánh lồng nhau dạng:

```
if (<Biểu thức>)
            if (<Biểu thức>)
                <Câu lệnh>;
            else
                <Câu lệnh>;
else
                <Câu lệnh>;
```

hoặc dạng:

```
            if (<Biểu thức>) <Câu lệnh>;
            else if (<Biểu thức>) <Câu lệnh>;
            else if (<Biểu thức>) <Câu lệnh>;
            else <Câu lệnh>;
```

Trong câu lệnh rẽ nhánh lồng nhau, mỗi `else` sẽ kết hợp với `if` gần nhất chưa kết hợp với `else` nào. Tuy nhiên, nếu câu lệnh rẽ nhánh dạng khuyết lồng trong câu lệnh rẽ nhánh dạng đủ thì ta phải sử dụng câu lệnh ghép để tránh sự nhập nhằng.

Ví dụ 3.3.

Trong câu lệnh sau, `else` kết hợp với `if` bên trong

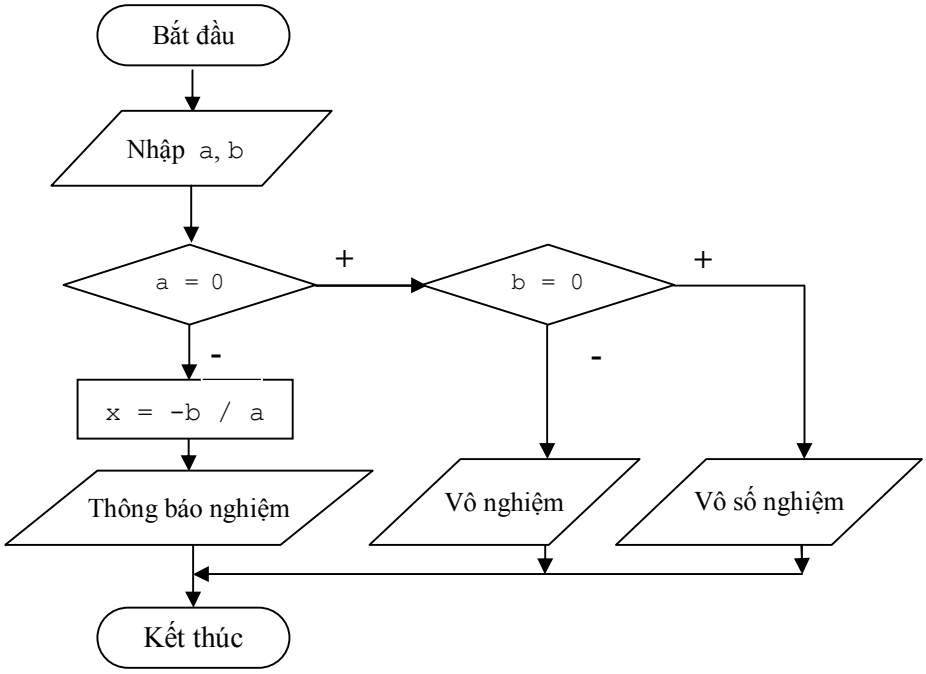
```
if (n > 0)
        if (a > b)
              z = a;
        else
              z = b;
```

Trong câu lệnh sau thì `else` kết hợp với `if` bên ngoài

```
if (n > 0)
{
     if (a > b)
         z = a;
}
else
     z = b;
```

Ví dụ 3.4. Vẽ lưu đồ thuật toán, sau đó viết chương trình giải phương trình dạng `ax + b = 0`.

Lưu đồ thuật toán:



Chương trình:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    float a, b;
    printf("\nNhap a, b = ");
    scanf("%f%f", &a, &b);
    if (a == 0)
        if (b == 0)
            printf("\nPhuong trinh vo so nghiem");
        else
            printf("\nPhuong trinh vo nghiem");
    else
        printf("\nPhuong trinh co nghiem x=%5.f",
                -b / a);
    getch();
    return 0;
}
```

3.3. Cấu trúc tuyển chọn: câu lệnh `switch`

Cú pháp:

```
switch (<Biểu thức>)
{
      case n1: <Công việc 1>; [break;]
      case n2: <Công việc 2>; [break;]
      ...
      case nk: <Công việc k>; [break;]
      [default: <Công việc k + 1>;]
}
```

Trong đó:

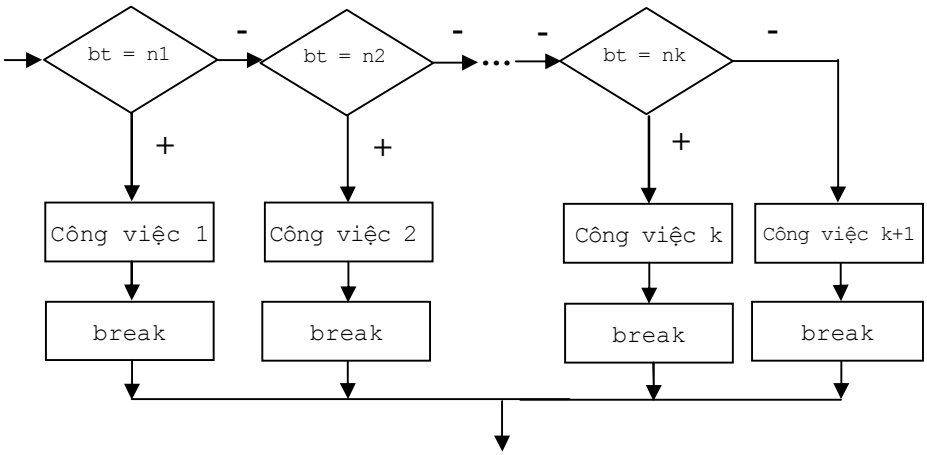
- Biểu thức (bt) có giá trị nguyên hoặc ký tự.

- Các ni (i = 1, 2, ..., k) là các biểu thức hằng có giá trị nguyên. Biểu thức hằng là biểu thức mà trong đó các toán hạng đều là các hằng.

- Công việc i (i = 1, 2, ..., k) là các câu lệnh. Câu lệnh break sau mỗi Công việc i có thể có hoặc không, tuy nhiên thường có câu lệnh break để sau khi thực hiện xong Công việc i sẽ kết thúc câu lệnh switch, nếu không chương trình vẫn tiếp tục kiểm tra các biểu thức tiếp theo cho đến khi gặp câu lệnh break hoặc kết thúc câu lệnh switch.

- Thành phần default có thể có hoặc không.

Sơ đồ hoạt động: (*trường hợp có* default *và có sử dụng câu lệnh* break *sau mỗi* Công việc i)



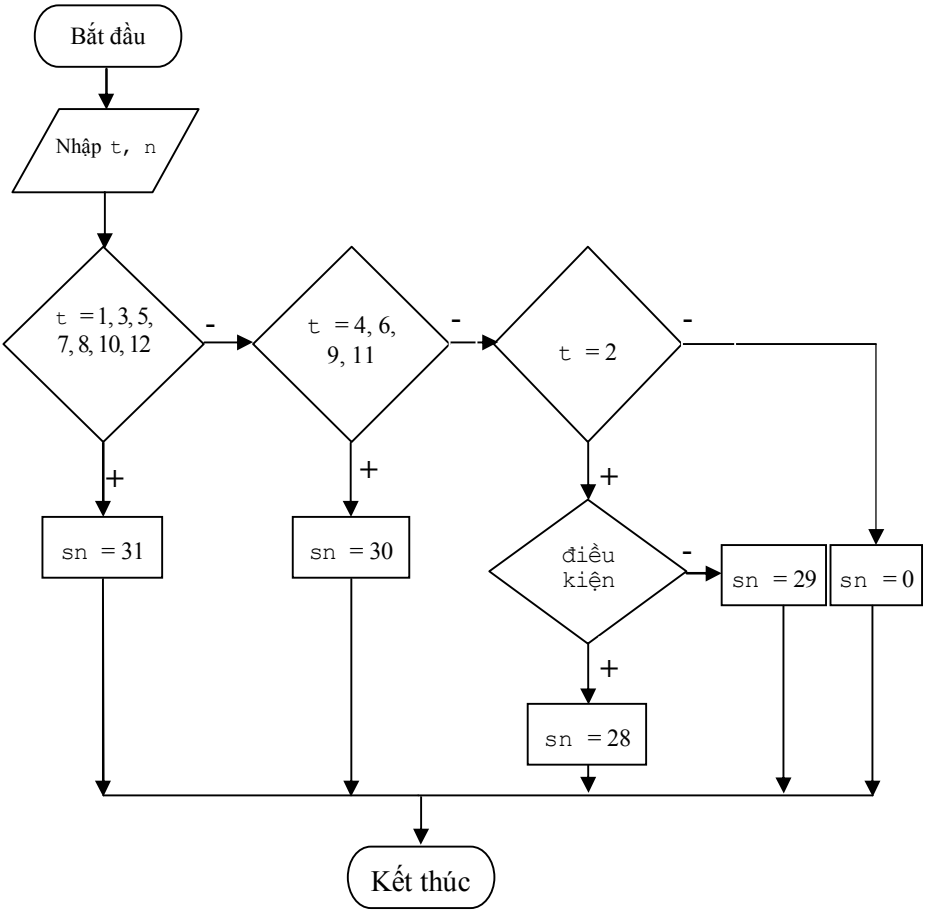
Ví dụ 3.5. Viết chương trình nhập vào số hiệu tháng và năm bất kỳ (dương lịch). Tính và thông báo ra màn hình số ngày của tháng và năm vừa nhập.

Quy luật tính số ngày của một tháng bất kỳ:

- Các tháng 1, 3, 5, 7, 8, 10, 12 có 31 ngày;

- Các tháng 4, 6, 9, 11 có 30 ngày;
- Tháng 2:
  + Nếu năm đó chia hết cho 4 thì có 29 ngày (trừ những năm chia hết cho 100 nhưng không chia hết cho 400).
  + Ngược lại có 28 ngày. Những năm chia hết 100 nhưng không chia hết cho 400, chẳng hạn như năm 1700, 1800, 1900, 2100 có 28 ngày.

Lưu đồ thuật toán:



Trong đó:

điều kiện = (n mod 4 <>0) hoặc ((n mod 100 = 0) và (n mod 400 <> 0))

Chương trình:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int n, t, sn;
    printf("\nNhap thang: ");
    scanf("%d", &t);
    printf("\nNhap nam: ");
    scanf("%d", &n);
    switch (t)
    {
       case 1:
       case 3:
       case 5:
       case 7:
       case 8:
       case 10:
       case 12: sn = 31; break;
       case 4:
       case 6:
       case 9:
       case 11: sn = 30; break;
       case 2: if ((n%4!=0)||(n%100==0)&&(n%400!=0))
                    sn = 28;
               else
                    sn = 29;
               break;
```

```
        default: sn = 0;
    }
    if (sn != 0)
        printf("\nThang %5d cua nam %5d co %5d ngay",
                t, n, sn);
    else
        printf("\nNhap sai thang!");
    getch();
}
```

3.4. Nhãn và câu lệnh nhảy đến nhãn goto

Cú pháp tạo nhãn:

```
<Tên nhãn>: <Các câu lệnh>;
```

Câu lệnh nhảy đến nhãn:

```
goto <Tên nhãn>;
```

Ví dụ 3.6. Cách tạo nhãn và sử dụng nhãn.

```
tiep: printf("Nhap so a = ");
      scanf("%d", &a);
if (a < 0)
      goto tiep;
```

✍ **Lưu ý:**

- Câu lệnh goto và nhãn tương ứng phải nằm trong cùng một hàm.
- Không cho phép dùng câu lệnh goto để nhảy từ ngoài vào trong một khối lệnh nhưng được phép nhảy từ trong một khối lệnh ra ngoài.
- Không nên sử dụng nhiều câu lệnh goto trong chương trình vì đây là câu lệnh nhảy không điều kiện nên khó kiểm soát và dễ gây rối chương trình.

Ví dụ 3.7. Viết chương trình nhập vào một số nguyên dương từ bàn phím. Nếu nhập số âm hoặc số `0` thì mời nhập lại.

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int n;
    nhap: printf("Nhap n = ");
    scanf("%d", &n);
    if (n <= 0)
       goto nhap;
    printf("\nSo nguyen duong vua nhap la: %5d", n);
    getch();
    return 0;
}
```

3.5. Cấu trúc lặp

3.5.1. Câu lệnh lặp `for`

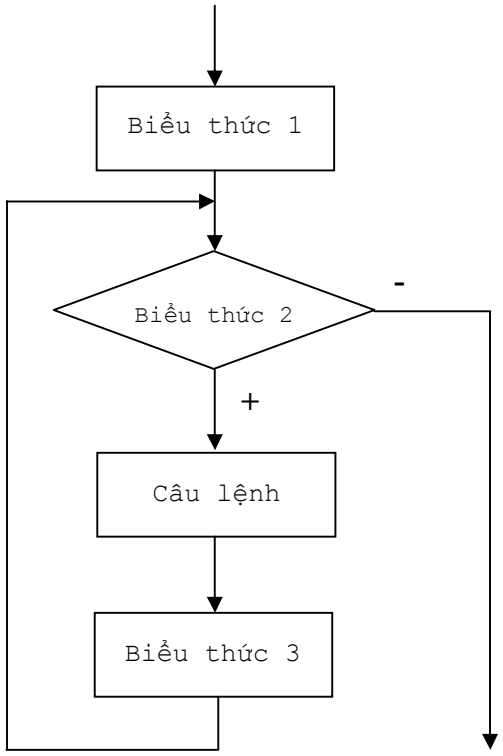
Cú pháp:

```
for(<Biểu thức 1>; <Biểu thức 2>; <Biểu thức 3>)
            <Câu lệnh>;    // Thân câu lệnh for
```

Trong đó:

- `Biểu thức 1`: Dùng để khởi tạo giá trị ban đầu cho các biến điều khiển (là biến mà giá trị của nó quyết định đến việc câu lệnh lặp tiếp tục lặp hay dừng lại).
- `Biểu thức 2`: Dùng để kiểm tra điều kiện kết thúc câu lệnh `for`.
- `Biểu thức 3`: Dùng để thay đổi giá trị cho các biến điều khiển.

Sơ đồ hoạt động:



✍ **Lưu ý:**

- Bất kỳ biểu thức nào trong ba biểu thức trên đều có thể vắng mặt nhưng phải giữ dấu chấm phẩy.

  Chẳng hạn: Để hiển thị lên màn hình 5 câu chào `"Hello"` ta sử dụng câu lệnh `for` đầy đủ sau:

```
for(i = 1; i <= 5; i++)
      printf("\nHello");
```

  Tuy nhiên, có thể thay câu lệnh `for` trên bằng câu lệnh `for` vắng mặt `Biểu thức 1` như sau:

```
int i = 1;
for(; i <= 5; i++)
      printf("\nHello");
```

- Khi `Biểu thức 2` vắng mặt thì nó luôn có giá trị bằng `1`. Trong trường hợp này việc kết thúc câu lệnh phải nhờ vào một trong các câu lệnh `break`, `goto`, `return` trong thân câu lệnh.

  Chẳng hạn:

```
int i = 1;
for(; ; i++)
{
      printf("\nHello");
      if (i >= 5) break;
}
```

- Biến điều khiển có thể có kiểu ký tự, số nguyên hoặc số thực.

  Chẳng hạn:

```
for(i = 1.5; i < 10.5; i++)
      printf("\nHello");
```

- Biến điều khiển có thể thay đổi (tăng, giảm) bởi một giá trị túy ý.

  Chẳng hạn:

```
for(i = 1; i < 10; i = i + 3)
      printf("\nHello");
```

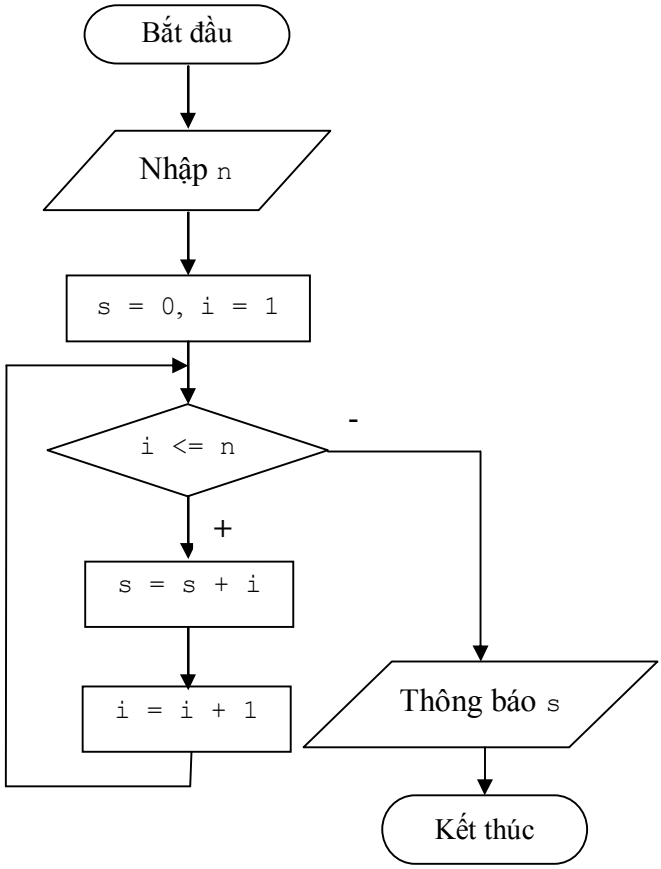
- Nếu câu lệnh `for` có thân đơn giản có thể kết hợp thân vào `Biểu thức 3` để tạo câu lệnh `for` rỗng. Lúc này phải có dấu chấm phẩy ngay sau câu lệnh `for`.

  Chẳng hạn:

```
for(i = 1; i < 5; s += i, i++);
```

Ví dụ 3.8. Sử dụng câu lệnh `for` tính tổng `s = 1 + 2 + … + n`, với `n` nguyên dương được nhập vào từ bàn phím.

Lưu đồ thuật toán:



Chương trình:

```
// Cach 1: Su dung cau lenh for day du
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
```

```
    for(i = 1; i <= n; i++)  // Cau lenh for day du
        s += i;              // Hay s = s + i;
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

```
// Cach 2: Su dung cau lenh for rong
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    for(i = 1; i <= n; s += i, i++); // Cau lenh for rong
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

```
// Cach 3: Su dung cau lenh for vang mat mot bieu thuc
#include <stdio.h>
#include <conio.h>
int main()
{
```

```
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    i = 1;
    for(; i <= n; i++) /* Cau lenh for vang mat
                          bieu thuc 1 */
        s += i;
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

```
// Cach 4: Su dung cau lenh for vang mat hai bieu thuc
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    i = 1;
    for(; i <= n;)  /* Cau lenh for vang mat bieu
                       thuc 1 va bieu thuc 3 */
        s += i++;
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

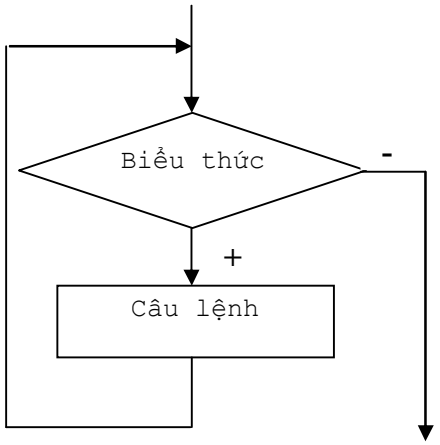
```
/* Cach 5: Su dung cau lenh for vang mat ca ba
bieu thuc */
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    i = 1;
    for(; ;)  // Cau lenh for vang mat ca ba bieu thuc
    {
        s += i++;
        if (i > n)
                break;
    }
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

3.5.2. Câu lệnh lặp `while`

Cú pháp:

```
while (<Biểu thức>)
        <Câu lệnh>;      // Thân câu lệnh while
```

Sơ đồ hoạt động:



Ví dụ 3.9. Viết chương trình tính tổng `s = 2 + 4 + … + 2n` sử dụng câu lệnh `while`.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    i = 1;
    while (i <= n)
        s += 2 * i++;
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

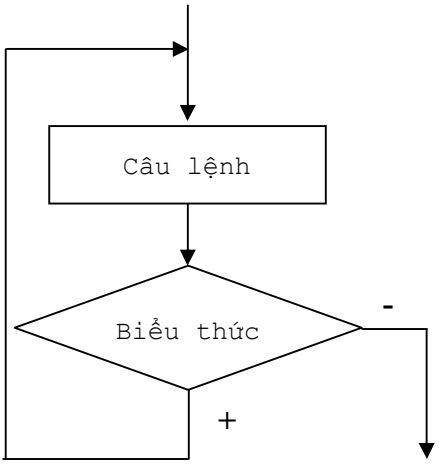
3.5.3. Câu lệnh lặp `do while`

Cú pháp:

```
do
{
    < Câu lệnh>;  // Thân câu lệnh do while
}
while (<Biểu thức>);
```

✍ **Lưu ý:** Thân của câu lệnh `do while` luôn luôn phải đặt giữa cặp ngoặc `{}` kể cả khi thân chỉ có một câu lệnh đơn.

Sơ đồ hoạt động:



Ví dụ 3.10. Viết chương trình tính tổng `s = 1 + 3 + … + (2n - 1)` sử dụng câu lệnh `do while`.

```
#include <stdio.h>
#include <conio.h>
int main()
```

```
{
    int i, n, s = 0;
    printf("\nNhap n = ");
    scanf("%d", &n);
    i = 1;
    do
    {
          s += 2 * i - 1;
          i++;
    }
    while (i <= n);
    printf("\nTong s = %5d", s);
    getch();
    return 0;
}
```

3.6. Câu lệnh `break` và `continue`

- **Câu lệnh `break`**

Câu lệnh `break` được sử dụng để kết thúc các câu lệnh có cấu trúc như `switch`, `for`, `while`, `do while` mà không quan tâm đến điều kiện kết thúc của chúng.

Ví dụ 3.11. Viết chương trình nhập vào từ bàn phím số nguyên dương `n` (`n` ≥ 2). Kiểm tra xem `n` có phải là số nguyên tố hay không?

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
int main()
```

```
{
    int i, kt = 1, n, s = 0;
    nhap: printf("Nhap so nguyen duong n = ");
    scanf("%d", &n);
    if (n <= 1)
        goto nhap;
    for(i = 2; i <= sqrt(n); i++)
    if (n % i == 0)
    {
        kt = 0;
        break;
    }
    if (kt == 1)
        printf("%5d la so nguyen to", n);
    else
       printf("%5d khong phai la so nguyen to", n);
    getch();
    return 0;
}
```

- **Câu lệnh `continue`**

Trong thân các câu lệnh lặp `for`, `while`, `do while` có thể dùng câu lệnh `continue` để chuyển đến đầu vòng lặp mà bỏ qua tất cả các câu lệnh trong thân câu lệnh lặp nhưng sau `continue`.

Ví dụ 3.12. Nhập vào từ bàn phím `n` số nguyên, tính và thông báo ra màn hình tổng của những số nguyên dương trong `n` số vừa nhập.

```
#include <conio.h>
#include <stdio.h>
#define n 10
int main()
{
    int a, i, s = 0;
    for(i = 1; i <= n; i++)
    {
          printf("\nNhap mot so nguyen duong: ");
          scanf("%d", &a);
          if (a <= 0)
              continue;
          s += a;
    }
    printf("\nTong =  %5d", s);
    getch();
}
```

BÀI THỰC HÀNH CHƯƠNG 3

- **Mục tiêu**

Qua bài thực hành người học cần đạt được:

- **Về kiến thức**
  - + Nắm được ý nghĩa, cách thức hoạt động của các câu lệnh có cấu trúc `if`, `switch`, `for`, `while`, `do while`, `goto`;
  - + Cách sử dụng các câu lệnh có cấu trúc lồng nhau.
- **Về kỹ năng**
  - + Xác định các câu lệnh có cấu trúc sử dụng phù hợp cho từng bài toán;
  - + Chuyển đổi cách dùng câu lệnh `if` và câu lệnh `switch`;
  - + Chuyển đổi cách dùng các câu lệnh `for`, `while`, `do while`.

- **Nội dung**

Bài 1. Viết chương trình nhập vào số nguyên `a`. Nếu `a` ≥ 0 thì thông báo lên màn hình "`a la so khong am`".

Yêu cầu:

+ Xác định câu lệnh có cấu trúc cần sử dụng;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

Bài 2. Viết chương trình nhập vào ba số nguyên `a`, `b` và `c` từ bàn phím. Tìm và thông báo lên màn hình số lớn nhất trong ba số vừa nhập.

Yêu cầu:

a. Sử dụng câu lệnh `if` *dạng khuyết.*

+ Số biến cần sử dụng, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

b. Sử dụng câu lệnh `if` *dạng đủ.*

+ Số biến cần sử dụng, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

Bài 3. Viết chương trình tính tổng $s = 1^2 + 2^2 + \ldots + n^2$, với `n` nguyên dương được nhập vào từ bàn phím.

Yêu cầu:

a. Sử dụng câu lệnh `for`

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

b. Sử dụng câu lệnh `while`

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

c. Sử dụng câu lệnh `do while`

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

Bài 4. Viết chương trình nhập vào một số nguyên `n`, nếu $n \leq 1$ hoặc $n \geq 9$ thì mời nhập lại. Thông báo lên màn hình số đó chẵn hay lẻ (sử dụng câu lệnh `switch`).

Yêu cầu:

+ Xác định biểu thức sau `switch`, các hằng n_i, công việc tương ứng với các hằng n_i;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

Bài 5. Viết chương trình tính tổng `s = 1! + 2! + 3! + … + n!`, với $0 \leq n \leq 10$ được nhập vào từ bàn phím (sử dụng câu lệnh `while`).

Yêu cầu:

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán;

+ Viết chương trình.

- **Câu hỏi củng cố**

- Biểu thức ngay sau từ khóa `if`, `switch` có nhất thiết phải đặt trong cặp ngoặc `()` không?

- Thân của câu lệnh `for` có thể là một câu lệnh `while` không?

- Thân của câu lệnh `do while` nếu chỉ có một câu lệnh có cần đặt giữa cặp ngoặc `{}` không?

- Mọi câu lệnh `for` đều có thể chuyển thành câu lệnh `while` hay `do while` và ngược lại đúng hay sai?

- Có xảy ra trường hợp các câu lệnh trong thân của câu lệnh `while`, `do while` không được thực hiện lần nào không?

- **Tự học**

Bài 6. Viết chương trình nhập vào từ bàn phím số nguyên `n`, nếu `n < 1` hoặc `n > 100` thì mời nhập lại, ngược lại sử dụng vòng lặp `for` tính tổng `s = n + (n + 1) + … + 100`.

Bài 7. Viết chương trình tính tổng `s = 1/100 + 1/99 + ... + 1/n`, với $1 \leq$ `n` ≤ 100 được nhập vào từ bàn phím (sử dụng câu lệnh `do while`).

Bài 8. Viết chương trình giải phương trình $ax^2 + bx + c = 0$, với các hệ số `a, b, c` nhập vào từ bàn phím.

Bài 9. Một người gửi tiết kiệm với số tiền ban đầu là `a`, lãi suất `b%` trên một năm. Tính tổng tiền gốc và lãi mà người đó có được sau `n` năm gửi tiết kiệm (`n` nguyên, `n` ≥ 1).

Bài 10. Viết chương trình nhập vào từ bàn phím số nguyên `n`, nếu `n < 1` hoặc `n > 100` thì mời nhập lại, ngược lại sử dụng vòng lặp tính tổng đan dấu `s = 1 - 1/2 + 1/3 - ... + (-1)`$^{n+1}$`/n`.

Bài 11. Viết chương trình nhập vào số thực `a` ≥ 1.5, tìm số nguyên dương `n` sao cho biểu thức `a - (1 + 1/2 + ... + 1/n)` có giá trị dương nhỏ nhất.

Gợi ý: **-** Khởi tạo `n = 1`.

- Sử dụng biến `s` để lưu tổng `1 + 1/2 + ... + 1/n`, kiểm tra giá trị biểu thức `a - s`, nếu giá trị này dương thì tiếp tục tăng `n` lên một đơn vị và quay lại kiểm tra giá trị `a - s`.

Chương 4

CON TRỎ VÀ MẢNG

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Ý nghĩa, cách khai báo và sử dụng biến con trỏ;*
- *Ý nghĩa, cách khai báo và sử dụng dữ liệu kiểu mảng một chiều, mảng hai chiều, xâu ký tự;*
- *Cấp phát động cho mảng một chiều, mảng hai chiều;*
- *Một số kỹ thuật xử lý mảng, xử lý xâu ký tự.*

Các tài liệu tham khảo chính của chương bao gồm [1, 3, 4, 7, 8].

4.1. Dữ liệu kiểu con trỏ

4.1.1. Khái niệm

Các biến đã được giới thiệu và sử dụng ở các chương trước đều là các biến có kích thước và kiểu dữ liệu xác định. Khi khai báo, máy sẽ cấp phát vùng nhớ cho biến mà không quan tâm đến việc vùng nhớ này có sử dụng hết hay không. Hơn nữa, các biến dạng này sẽ chiếm giữ bộ nhớ trong suốt thời gian thực thi đoạn chương trình mà chúng có hiệu lực. Do đó, khi sử dụng biến dạng này ta có thể gặp các khó khăn sau:

- Có thể gây lãng phí bộ nhớ khi cấp phát nhiều mà sử dụng không hết bộ nhớ đã cấp phát;
- Khi cấp phát thiếu, chương trình thực thi sẽ bị lỗi.

Để tránh những hạn chế trên, ngôn ngữ lập trình C cung cấp một loại biến đặc biệt là biến con trỏ dùng để chứa địa chỉ. Đặc điểm của biến con trỏ:

- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu mà luôn có kích thước cố định là hai `byte`;
- Khi cần sử dụng thì mới cấp phát bộ nhớ cho biến con trỏ quản lý;
- Cấp phát bộ nhớ vừa đủ để sử dụng;
- Khi không cần sử dụng bộ nhớ đã được cấp cho con trỏ quản lý thì có thể giải phóng để tiết kiệm bộ nhớ.

4.1.2. Cách khai báo biến con trỏ

Cú pháp: `<Kiểu> *<Tên biến trỏ>;`

Trong đó: `Kiểu` có thể là các kiểu dữ liệu cơ bản, cũng có thể là các kiểu dữ liệu do người lập trình định nghĩa (*được trình bày ở Chương 6*). Như vậy, có bao nhiêu kiểu dữ liệu thì có bấy nhiêu kiểu con trỏ.

Ví dụ 4.1.
```
int *x;     // Khai báo con trỏ x có kiểu int
float *y;   // Khai báo con trỏ y có kiểu float
```

4.1.3. Phép toán *

Phép toán `*` là phép toán một ngôi cho phép lấy nội dung của vùng nhớ mà biến con trỏ đang chứa địa chỉ.

4.1.4. Các phép toán đối với biến con trỏ

Các phép toán liên quan đến con trỏ và địa chỉ bao gồm:

Phép gán;

Phép tăng giảm địa chỉ;

Phép truy cập bộ nhớ;

Phép so sánh;

Phép trừ hai con trỏ.

a. Phép gán

- **Gán địa chỉ của một biến cho biến con trỏ**

Có thể gán địa chỉ của một biến cho một biến con trỏ cùng kiểu.

Cú pháp: `<Tên biến trỏ> = &<Tên biến>;`

Ví dụ 4.2.
```
int *x, a;
float *y, b;
```

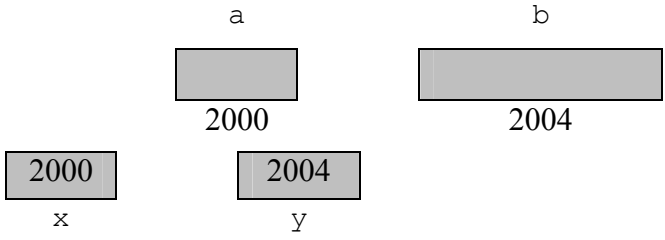
Khi đó, các phép gán sau là đúng:

```
x = &a; y = &b;     /* Ta có thể nói con trỏ x trỏ tới
                       vùng nhớ a, con trỏ y trỏ tới vùng
                       nhớ b */
```

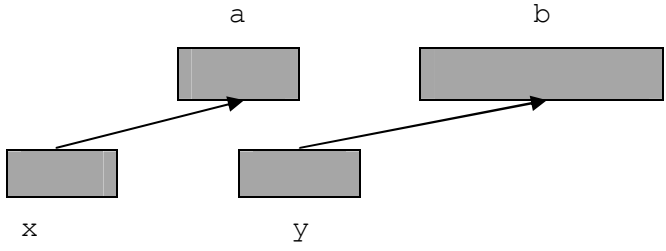
Các phép gán sau là sai:

```
x = a; y = b; x = &b; y = &a;
```

Giả sử địa chỉ vùng nhớ `a` là 2000, địa chỉ vùng nhớ `b` là 2004, khi đó hình ảnh con trỏ `x` trỏ tới vùng nhớ `a` và con trỏ `y` trỏ tới vùng nhớ `b` thể hiện trong bộ nhớ như sau:



Về mặt hình thức, có thể biểu diễn hình ảnh con trỏ `x` trỏ tới vùng nhớ `a` và con trỏ `y` trỏ tới vùng nhớ `b` như sau:



Muốn gán cho biến con trỏ địa chỉ khác kiểu thì phải sử dụng phép ép kiểu.

Cú pháp: `(<Kiểu>*)(<Địa chỉ>)`

Trong *Ví dụ 4.2*, các phép gán sau là hợp lệ:

```
x = (int*)(&b); y = (float*)(&a);
```

- **Gán một biến con trỏ cho một biến con trỏ**

Có thể gán một biến con trỏ cho một biến con trỏ cùng kiểu.

Cú pháp: `<Biến con trỏ 1> = <Biến con trỏ 2>;`

Ví dụ 4.3.
```
int *x, *y, a;
x = &a;     // Gán địa chỉ biến a cho con trỏ x
y = x;      /* Gán địa chỉ đang chứa trong con trỏ x
               cho con trỏ y */
```

✍ **Lưu ý:** Có thể gán giá trị `NULL` cho một biến con trỏ.

Chẳng hạn:
```
int *x;
x = NULL;
```

Muốn gán cho biến con trỏ một biến con trỏ khác kiểu thì phải sử dụng phép ép kiểu.

Cú pháp: `(<Kiểu>*)(<Tên biến trỏ>)`

Chẳng hạn:
```
int *x;
float *y;
```
Phép gán sau là hợp lệ: `y = (float*)x;`

- **Gán giá trị cho vùng nhớ mà biến con trỏ đang trỏ**

Cú pháp: `*<Tên biến trỏ> = <Giá trị>;`

Ví dụ 4.4.
```
int *x, a;
x = &a;
*x = 10;
```

Như vậy, biến con trỏ `x` đang chứa địa chỉ biến `a`, khi gán `*x = 10` nghĩa là `a = 10`.

Ví dụ 4.5. Chương trình sau hiển thị giá trị của một biến kiểu số nguyên, địa chỉ của nó được lưu trong một biến con trỏ và địa chỉ của biến con trỏ.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int *ptr, var = 500;
    ptr = &var; //Cho con tro ptr chua dia chi bien var
    printf("\nGia tri bien var duoc luu tai dia
              chi:%u", &var);
    printf("\nGia tri %d duoc luu tai dia chi: %u",
              *ptr, ptr);
    printf("\nDia chi cua bien con tro ptr: %u", &ptr);
    getch();
    return 0;
}
```

✍ **Lưu ý:**

- Khi hiển thị địa chỉ của một biến ta sử dụng mã định dạng `%u` vì nói chung tất cả các biến con trỏ đều được lưu trữ trong 2 `byte (unsigned int)` hoặc dùng `%p` để hiển thị địa chỉ `offset`;
- `ptr` là biến nên nó cũng có thể sử dụng phép toán `&` để lấy địa chỉ chứa nó;
- Kết quả hai câu lệnh sau là tương đương:

```
printf("\nGia tri cua bien var la: %d", var);
printf("\nGia tri cua bien var la: %d", *(&var));
```

b. Phép truy nhập bộ nhớ

Khi con trỏ trỏ tới một vùng nhớ nào đó thì vùng nhớ do con trỏ chiếm giữ có số byte bằng số byte của kiểu của con trỏ.

Chẳng hạn: Con trỏ float chiếm giữ 4 byte, con trỏ int chiếm giữ 2 byte, con trỏ char chiếm giữ 1 byte,…

Ví dụ 4.6.

Giả sử ta có các khai báo:

```
float *pf;
int *pi;
char *pc;
```

Khi đó:

- Nếu pf trỏ tới byte thứ 100 thì *pf chiếm giữ vùng nhớ 4 byte liên tiếp từ byte 100 đến 103.
- Nếu pi trỏ tới byte thứ 100 thì *pi chiếm giữ vùng nhớ 2 byte liên tiếp từ byte 100 đến 101.
- Nếu pc trỏ tới byte thứ 100 thì *pc chiếm giữ vùng nhớ 1 byte chính là byte 100.

c. Phép tăng giảm địa chỉ

Ta có thể cộng, trừ một con trỏ với một số nguyên n, kết quả trả về là địa chỉ vùng nhớ cách vùng nhớ hiện tại n phần tử (mỗi phần tử chiếm vùng nhớ bằng số byte của kiểu của con trỏ). Các phép toán ++, -- cũng có thể sử dụng cho biến con trỏ.

Ví dụ 4.7.

```
int *x, a;
x = &a;          // Gán địa chỉ biến a cho con trỏ x
x = x + 1;       // Hay x++;
```

Khi này con trỏ x sẽ trỏ tới vùng nhớ ngay sau vùng nhớ a. Vùng nhớ này có địa chỉ hơn địa chỉ vùng nhớ a là 2 (vì số byte của kiểu int là 2).

Ví dụ 4.8. Chương trình sau minh họa các phép toán tăng, giảm địa chỉ.

```
#include <conio.h>
#include <stdio.h>
int main()
{
      int *m, n = 5;
      m = &n;
      printf("\nDia chi cua bien n = %u", &n);
      printf("\nDia chi cua bien n thong qua con
      tro m la: %u", m);
      m = m + 2;
      printf("\nDia chi vung nho con tro m tro toi
      sau phep gan m = m + 2 la: %u", m);
      m--;
      printf("\nDia chi vung nho con tro m tro toi
      sau khi thuc hien m-- la: %u", m);
      getch();
      return 0;
}
```

Kết quả chạy chương trình:

```
Dia chi cua bien n = 65524
Dia chi cua bien n thong qua con tro m la: 65524
```

```
Dia chi vung nho do con tro m tro toi sau phép gán m =
m + 2 la: 65528

Dia chi vung nho do con tro m tro toi sau khi thuc
hien m-- la: 65526
```

Nhìn vào kết quả chạy chương trình ta thấy: Lúc đầu `m` trỏ tới vùng nhớ có địa chỉ `65524`, sau khi thực hiện `m = m + 2` con trỏ `m` trỏ tới vùng nhớ có địa chỉ `65528`, vùng nhớ này cách vùng nhớ ban đầu 4 `byte` (2 phần tử `int`). Sau khi thực hiện `m--` con trỏ `m` trỏ tới vùng nhớ có địa chỉ `65526`, vùng nhớ này cách vùng nhớ trước khi thực hiện `m--` là 2 `byte` (1 phần tử `int`).

d. Phép so sánh

Ta có thể sử dụng các phép toán so sánh `>, >=, <, <=, ==, !=` đối với hai con trỏ cùng kiểu.

Giả sử ta có hai con trỏ cùng kiểu `p` và `q`, khi đó:

`p > q` nếu địa chỉ `p` trỏ tới cao hơn địa chỉ `q` trỏ tới.
`p >= q` nếu địa chỉ `p` trỏ tới cao hơn hoặc bằng địa chỉ `q` trỏ tới.
`p < q` nếu địa chỉ `p` trỏ tới thấp hơn địa chỉ `q` trỏ tới.
`p <= q` nếu địa chỉ `p` trỏ tới thấp hơn hoặc bằng địa chỉ `q` trỏ tới.
`p == q` nếu địa chỉ `p` trỏ tới cũng là địa chỉ `q` trỏ tới.
`p != q` nếu địa chỉ `p` trỏ tới khác địa chỉ `q` trỏ tới.

Ví dụ 4.9.

```
int *p, *q, x;
p = &x;
q = p + 1;
```

Ta có:
Giá trị của biểu thức `p > q` là `0`.
Giá trị của biểu thức `p >= q` là `0`.
Giá trị của biểu thức `p < q` là `1`.
Giá trị của biểu thức `p <= q` là `1`.
Giá trị của biểu thức `p == q` là `0`.
Giá trị của biểu thức `p != q` là `1`.

e. Phép trừ hai con trỏ

Ta có thể thực hiện phép trừ hai con trỏ cùng kiểu. Kết quả của phép trừ là một số nguyên, trị tuyệt đối của số nguyên này chỉ số phần tử giữa hai vùng nhớ mà hai con trỏ đang trỏ.

Ví dụ 4.10.

```
int *p,*q, x;
p = &x;
q = p + 4;
```

Kết quả `q - p` bằng `4`, chính là số phần tử tương ứng giữa `p` và `q`. Mỗi phần tử chiếm vùng nhớ 2 `byte` (*bằng số* `byte` *của kiểu* `int`).

4.1.5. Con trỏ kiểu `void`

Khai báo: **`void`** `*<Tên biến trỏ>;`

Con trỏ **`void`** là con trỏ đặc biệt, không kiểu, có thể gán cho nó bất cứ địa chỉ thuộc kiểu nào.

Ví dụ 4.11.

```
void *p;
int x;
float y;
p = &x;    // Câu lệnh này hợp lệ
p = &y;    // Câu lệnh này cũng hợp lệ
```

✍ **Lưu ý:**

- Các phép toán tăng giảm địa chỉ, so sánh và truy cập bộ nhớ không dùng được trên con trỏ `void`;

- Khi gán con trỏ `void` cho một con trỏ kiểu khác `void` ta phải sử dụng phép ép kiểu để ép con trỏ `void` thành kiểu con trỏ cần gán.

Chẳng hạn:

```
void *p;
int *x;
x = (int*)p;
```

4.1.6. Cấp phát vùng nhớ cho con trỏ quản lý

Để cấp phát vùng nhớ cho con trỏ quản lý ta sử dụng các hàm `malloc` hoặc `calloc`. Các hàm này được khai báo trong thư viện `stdlib.h`.

Cú pháp:

```
void* malloc(size)          /* Cấp phát vùng nhớ có kích thước là size */
void* calloc(nitems, size)  /* Cấp phát vùng nhớ có kích thước nitems*size */
```

✍ **Lưu ý:** Khi sử dụng các hàm này ta phải ép kiểu vì nguyên mẫu của các hàm trả về con trỏ `void`.

Ví dụ 4.12.

```
int *p, q;
```

// Cấp phát vùng nhớ 2 `byte` có thể lưu 1 số nguyên `int`

```
p = (int*)malloc(sizeof(int));
```

// Cấp phát vùng nhớ 10 `byte` (= 5 * 2) có thể lưu 5 số nguyên `int`

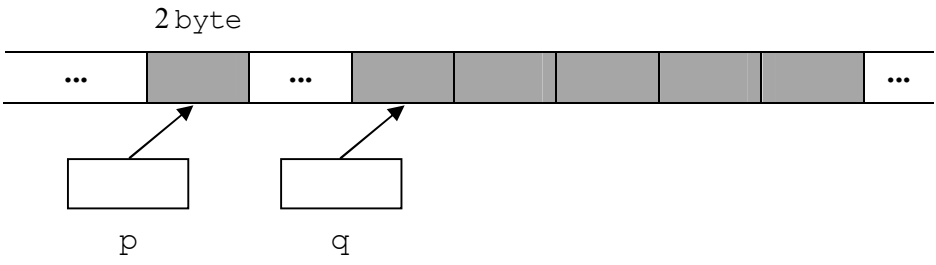
```
q = (int*)malloc(5 * sizeof(int))
```

Tuy nhiên, ta cũng có thể sử dụng hàm `malloc` cấp phát vùng nhớ 10 `byte` để có thể lưu 5 số nguyên `int` như sau:

```
q = (int*)calloc(5, sizeof(int));
```

// **`sizeof`**`(<Kiểu>)` cho số `byte` của `Kiểu`

Hình ảnh vùng nhớ sau khi cấp phát cho con trỏ `p` và `q`



4.1.7. Cấp phát lại vùng nhớ đã cấp cho con trỏ quản lý

Trong quá trình sử dụng biến con trỏ, nếu ta cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm `realloc` thuộc thư viện `stdlib.h`.

Cú pháp: `void *realloc(*block, size)`

Ý nghĩa :

- Cấp phát lại một vùng nhớ cho con trỏ `block` quản lý, vùng nhớ này có kích thước mới là `size`, khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại. Nội dung của vùng nhớ trước đó được sao chép sang vùng nhớ mới.
- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ khi cấp phát ban đầu.

4.1.8. Giải phóng vùng nhớ đã cấp cho con trỏ quản lý

Khi không còn sử dụng vùng nhớ đã cấp phát cho con trỏ quản lý ta nên giải phóng vùng nhớ này để có thể sử dụng vào việc khác bằng cách sử dụng hàm `free` thuộc thư viện `stdlib.h`.

Cú pháp: `void *free(void *block)`

Ví dụ 4.13.

```
int *p;
p = (int*)calloc(5, sizeof(int));
// Giải phóng vùng nhớ đã cấp cho con trỏ p
free(p);
```

Ví dụ 4.14. Chương trình sau cấp phát vùng nhớ cho con trỏ quản lý gồm `n` phần tử, nhập giá trị cho các phần tử. Hiển thị lên màn hình giá trị các phần tử theo thứ tự xuôi, ngược. Sau đó, giải phóng vùng nhớ đã cấp cho con trỏ.

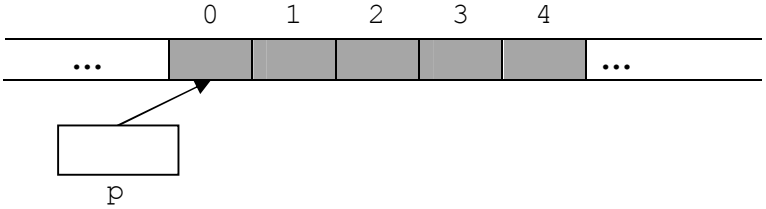
```
#include <conio.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#define n 5
//=======================================
int main()
{
    int i, *p;
    p = (int*)calloc(n, sizeof(int)); /* Cap phat
                                         vung nho
                                         cho con
                                         tro p */
    for(i = 0; i < n; i++)
    {
          printf("Nhap mot so nguyen : ");
          scanf("%d", p);
          p++;
    }
    printf("\nDanh sach in xuoi: ");
    p = p - n;
    for(i = 0; i < n; i++)
    {
          printf("%5d", *p);
          p++;
    }
    printf("\nDanh sach in nguoc: ");
    p--;
    for(i = 0; i < n; i++)
    {
          printf("%5d", *p);
          p--;
    }
    p++;
```

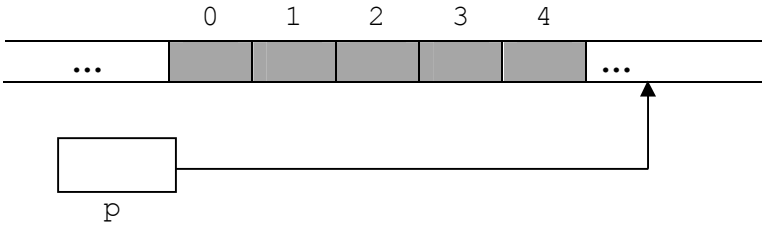
```
    free(p);   /* Giai phong vung nho da cap cho
                  con tro p */
    getch();
    return 0;
}
```

Chương trình được thể hiện như sau:

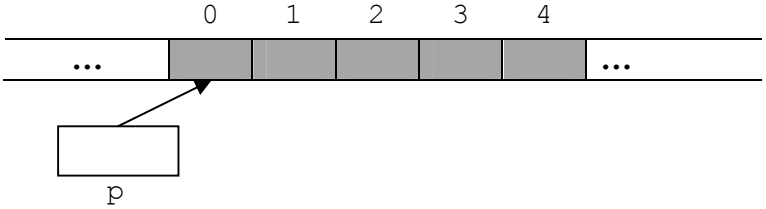
Kết quả câu lệnh `p = (int*)calloc(n, sizeof(int));`



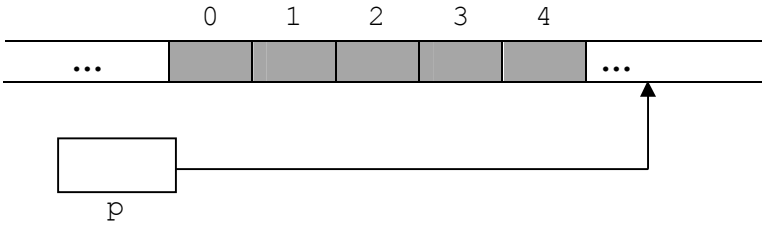
Sau khi nhập dữ liệu



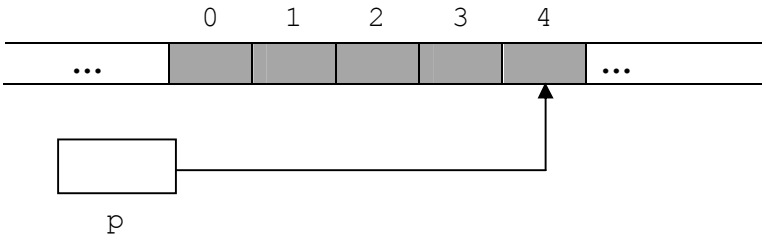
Cho con trỏ `p` *trỏ tới phần tử đầu tiên bằng lệnh* `p = p - n;`



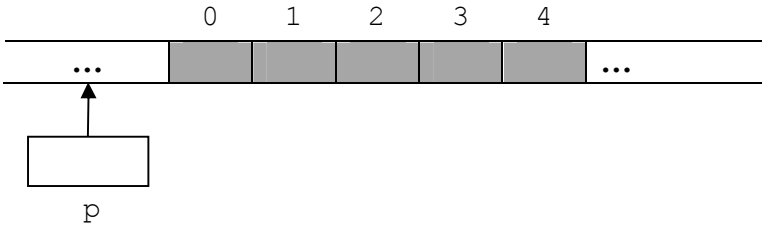
Sau khi in xuôi



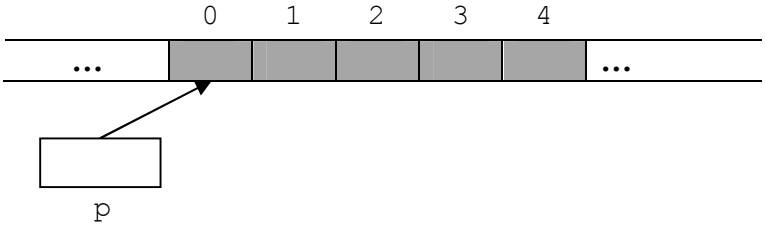
Cho con trỏ `p` *trỏ tới phần tử cuối cùng*



Sau khi in ngược danh sách



Cho con trỏ `p` *trỏ tới phần tử đầu tiên*



4.2. Con trỏ đa cấp

Như ta đã biết con trỏ là một loại biến mà giá trị của nó là một địa chỉ. Nhưng vì bản thân con trỏ cũng là một biến nhớ nên con trỏ cũng có địa chỉ của mình và ta hoàn toàn có thể lấy địa chỉ đó bằng phép toán `&`.

Ví dụ 4.15.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int *x, y = 10;
    x = &y;
```

```
    printf("\nDia chi cap 1 %p", x);
    printf("\nDia chi cap 2 %p", &x);
    getch();
    return 0;
}
```

✍ **Lưu ý:** Trong ví dụ trên, địa chỉ đang lưu giữ trong `x` được gọi là địa chỉ cấp 1, còn `&x` được gọi là địa chỉ cấp 2. Vì địa chỉ cấp 2 cũng là một địa chỉ nên chúng ta hoàn toàn có thể khai báo một biến để chứa địa chỉ cấp 2 này, biến đó được gọi là con trỏ cấp 2.

Cú pháp: `<Kiểu>  **<Biến trỏ>;`

Phép toán `**` được dùng để khai báo con trỏ cấp hai. Phép toán này có thể áp dụng cho tất cả các kiểu dữ liệu cơ bản hay kiểu cấu trúc. Như vậy, về nguyên tắc ta hoàn toàn có thể có các biến con trỏ cấp `n` ($n > 2$) nhưng những con trỏ như vậy rất ít sử dụng nên không được giới thiệu trong giáo trình này.

4.3. Dữ liệu kiểu mảng

4.3.1. Khái niệm mảng

Mảng là một tập hợp các phần tử liên tiếp nhau có cùng kiểu được gọi là kiểu phần tử. Kiểu phần tử có thể là các kiểu dữ liệu cơ bản, có thể là kiểu tự định nghĩa như kiểu cấu trúc, thậm chí kiểu phần tử có thể là kiểu mảng. Dữ liệu kiểu mảng sẽ hỗ trợ cho việc lập trình thuận lợi hơn đối với nhiều bài toán.

Chẳng hạn, muốn viết chương trình để tính tổng của `n` số nguyên ta có thể khai báo `n` biến kiểu số nguyên để lưu các giá trị đó. Giả sử, với `n = 10` ta có thể khai báo 10 biến kiểu số nguyên để lưu giá trị, tuy nhiên với `n` lớn thì việc khai báo số biến lớn như vậy là khó chấp nhận được. Trong trường hợp này việc sử dụng dữ liệu kiểu mảng sẽ rất thuận lợi.

4.3.2. Mảng một chiều

Là một cấu trúc dữ liệu thuận tiện cho việc lưu trữ dữ liệu dạng vectơ số trong toán học.

- **Khai báo mảng với số phần tử xác định**

Cú pháp: `<Kiểu> <Tên mảng>[<Số phần tử>];`

Trong đó:

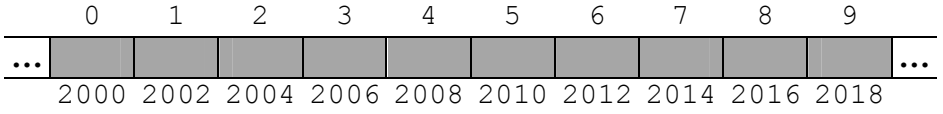
- `Kiểu`: Là kiểu của các phần tử của mảng;
- `Tên mảng`: Được đặt theo quy tắc đặt tên của ngôn ngữ lập trình C;
- `Số phần tử`: Là một số nguyên dương cho biết số phần tử của mảng.

Ví dụ 4.16.

```
int   a[10]; /* Khai báo biến mảng a gồm 10 phần tử kiểu
                số nguyên */
```

Máy sẽ cấp phát cho mảng `a 10` phần tử liên tiếp nhau, mỗi phần tử chiếm `2 byte` (bằng số `byte` của kiểu `int`). Chỉ số các phần tử của mảng được tính từ `0` đến `9`. Giả sử địa chỉ của phần tử đầu tiên là `2000`, ta có hình ảnh mảng `a` trong bộ nhớ như sau:

Hình ảnh mảng a trong bộ nhớ



Vừa khai báo mảng với số phần tử xác định vừa gán giá trị khởi tạo cho mảng

Cú pháp:

`<Kiểu> <Tên mảng>[<Số phần tử>] = {<Các giá trị>};`

Trong đó: `Các giá trị` nếu có nhiều hơn một giá trị thì được viết cách nhau bởi dấu phẩy.

Nếu số phần tử gán sẵn nhiều hơn số phần tử của mảng thì ngôn ngữ lập trình C không báo lỗi nhưng sẽ bỏ qua các phần tử thừa. Ngược lại, nếu số phần tử khởi tạo ít hơn số phần tử của mảng thì các phần tử còn lại sẽ được ngầm định gán giá trị bằng `0`.

Ví dụ 4.17.

```
int[10] = {10, 20, 30, 40, 50};
```

Khi này các giá trị khởi tạo được đặt vào các phần tử, bắt đầu từ phần tử thứ 0.

Hình ảnh mảng `a` trong bộ nhớ

	0	1	2	3	4	5	6	7	8	9	
…	10	20	30	40	50	0	0	0	0	0	…

- **Khai báo mảng với số phần tử không xác định**

Cú pháp: `<Kiểu> <Tên mảng>[];`

✍ **Lưu ý:** Cách khai báo này thường áp dụng trong các trường hợp: vừa khai báo mảng vừa khởi tạo giá trị hoặc khai báo mảng làm tham số hình thức cho hàm (*được trình bày trong Chương 5*).

Vừa khai báo mảng với số phần tử không xác định vừa khởi tạo giá trị cho mảng

Cú pháp: `<Kiểu> <Tên mảng>[] = {<Các giá trị>};`

Trong trường hợp này, ngôn ngữ lập trình C sẽ hiểu số phần tử của mảng là số giá trị mà ta đã gán trực tiếp trong cặp ngoặc `{}`. Chúng ta có thể lấy số phần tử của mảng như sau:

```
Số phần tử = sizeof(<Tên mảng>) / sizeof(<Kiểu>)
```

Chẳng hạn:

```
int a[] = {1, -2, 0, 4, -5};
```

Số phần tử của mảng = `sizeof(a) / sizeof(int) = 10 / 2 = 5`

- **Truy nhập đến phần tử mảng**

Muốn truy nhập đến phần tử mảng một chiều ta có thể sử dụng một trong hai cách sau:

- **Sử dụng phép toán** `[]`

Để truy nhập đến phần tử thứ `i` của mảng ta viết `<Tên mảng>[i]`.

Để truy nhập đến địa chỉ của phần tử thứ `i` của mảng ta viết `&<Tên mảng>[i]`.

Ví dụ 4.18. Nếu ta khai báo `int a[5];`

`a[0]` là phần tử thứ 0; `&a[0]` là địa chỉ phần tử thứ `0`.

`a[1]` là phần tử thứ `1`; `&a[1]` là địa chỉ phần tử thứ `1`.

`a[2]` là phần tử thứ `2`; `&a[2]` là địa chỉ phần tử thứ `2`.

`a[3]` là phần tử thứ `3`; `&a[3]` là địa chỉ phần tử thứ `3`.

`a[4]` là phần tử thứ `4`; `&a[4]` là địa chỉ phần tử thứ `4`.

- **Sử dụng con trỏ**

Trong ngôn ngữ lập trình C, tên mảng được hiểu là một hằng địa chỉ và nó luôn luôn là địa chỉ của phần tử thứ `0`.

Ví dụ 4.19. Nếu khai báo `int a[5];`

Thì

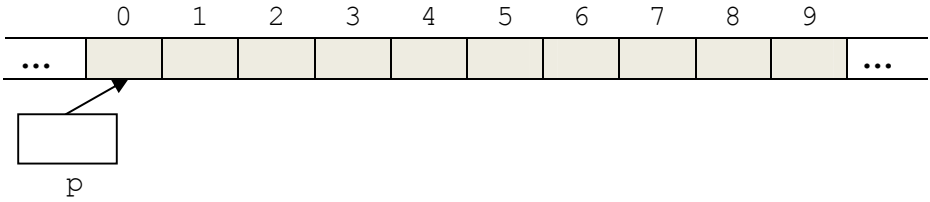
`a`	tương đương với `&a[0]`
`a + 1`	tương đương với `&a[1]`
`a + 2`	tương đương với `&a[2]`
`a + 3`	tương đương với `&a[3]`
`a + 4`	tương đương với `&a[4]`
`*a`	tương đương với `a[0]`
`*(a + 1)`	tương đương với `a[1]`
`*(a + 2)`	tương đương với `a[2]`

`*(a + 3)` tương đương với `a[3]`

`*(a + 4)` tương đương với `a[4]`

Như vậy, để truy nhập đến phần tử mảng nhờ con trỏ ta khai báo một con trỏ cùng kiểu với phần tử mảng, sau đó gán con trỏ bằng tên mảng. Khi này, con trỏ và mảng gần như tương đương nhau.

Ví dụ 4.20.
```
int a[5], *p;
p = a; // Hoặc có thể gán p = &a[0];
```



`p` trỏ tới phần tử `a[0]`; `*p` tương đương `a[0]`

`p + 1` trỏ tới phần tử `a[1]`; `*(p + 1)` tương đương `a[1]`

`p + 2` trỏ tới phần tử `a[2]`; `*(p + 2)` tương đương `a[2]`

`p + 3` trỏ tới phần tử `a[3]`; `*(p + 3)` tương đương `a[3]`

`p + 4` trỏ tới phần tử `a[4]`; `*(p + 4)` tương đương `a[4]`

Như vậy, ta có các cách viết sau là tương đương

`a[i]` `*(a + i)` `p[i]` `*(p + i)` đều chỉ giá trị của `a[i]`

`&a[i]` `(a + i)` `p + i` đều chỉ địa chỉ của `a[i]`

Ví dụ 4.21. Viết chương trình nhập và hiển thị lên màn hình giá trị các phần tử của mảng một chiều bằng cách sử dụng chỉ số.

Để nhập giá trị cho `n` phần tử của mảng ta có thể sử dụng `n` lần hàm `scanf`. Tương tự, để hiển thị giá trị của `n` phần tử của mảng ta dùng `n` lần hàm `printf`. Trong trường hợp này, để thuận tiện ta sử dụng câu lệnh lặp cho việc nhập và hiển thị giá trị các phần tử của mảng.

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int a[10], i, n;
    printf("\nNhap so phan tu cua mang n = ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        printf("\nNhap phan tu thu %d = ", i);
        scanf("%d", &a[i]);
    }
    printf("\nHien thi bang chi so cach 1: \n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\nHien thi bang chi so cach 2: \n");
    for(i = 0; i < n; i++)
        printf("%5d", *(a + i));
    getch();
    return 0;
}
```

Ví dụ 4.22. Viết chương trình nhập và hiển thị lên màn hình giá trị các phần tử của mảng một chiều bằng cách sử dụng con trỏ.

```
#include <conio.h>
#include <stdio.h>
int main()
{
```

```
    int a[10], i, n, *p;
    p = a;
    printf("\nNhap so phan tu cua mang n = ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        printf("\nNhap phan tu thu %d = ", i);
        scanf("%d", p);
        p++;
    }
    p = p - n; //Dua con tro p ve dau mang
    printf("\nHien thi bang con tro cach 1: ");
    for(i = 0; i < n; i++)
        printf("%5d", p[i]);
    printf("\nHien thi bang con tro cach 2: ");
    for(i = 0; i < n; i++)
        printf("%5d", *(p + i));
    printf("\nHien thi bang con tro cach 3: ");
    for(i = 0; i < n; i++)
    {
        printf("%5d", *p);
        p++;
    }
    p = p - n;
    getch();
    return 0;
}
```

Ví dụ 4.23. Viết chương trình khai báo mảng một chiều các số thực với các phần tử khởi tạo sẵn. Hiển thị lên màn hình giá trị các phần tử của mảng.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, n;
    int dayso[] = {66, 65, 69, 68, 67, 70};
    n = sizeof(dayso) / sizeof(int); /* Lay so phan tu
                                        cua mang */
    printf("\nMang da cho: ");
    for (i = 0; i < n; i++)
        printf("%5d ", dayso[i]);
    getch();
    return 0;
}
```

Ví dụ 4.24. Viết chương trình nhập mảng một chiều các số nguyên. Hiển thị lên màn hình giá trị các phần tử và giá trị lớn nhất của mảng.

Thuật toán tìm giá trị lớn nhất của mảng: Giả sử ta có mảng `a` gồm `n` phần tử, `max` là biến dùng để chứa giá trị lớn nhất.

- Giả sử `a[0]` là phần tử có giá trị lớn nhất, gán `max = a[0]`.
- So sánh `max` với giá trị các phần tử từ `a[1]` đến `a[n - 1]`, nếu `max` bé hơn giá trị của phần tử nào thì gán `max` bằng giá trị của phần tử đó. Giá trị `max` cuối cùng nhận được chính là giá trị lớn nhất của mảng.

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int a[10], i, max, n;
    printf("\nNhap so phan tu cua mang n = ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        printf("\nNhap phan tu thu %d = ", i);
        scanf("%d", &a[i]);
    }
    printf("\nMang vua nhap la: \n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    max = a[0];
    for(i = 1; i < n; i++)
        if (max < a[i])
            max = a[i];
    printf("\nGia tri lon nhat cua mang la: %5d", max);
    getch();
    return 0;
}
```

Ví dụ 4.25. Viết chương trình nhập vào hai vectơ số cùng chiều gồm các số nguyên. Tính tổng của hai vectơ. Hiển thị lên màn hình hai vectơ và vectơ tổng.

Cách tốt nhất để lưu trữ các vectơ số là sử dụng mảng một chiều. Chương trình sau khai báo ba mảng một chiều `a`, `b`, `c` có cùng số phần tử, trong đó hai mảng `a`, `b` được nhập giá trị vào từ bàn phím, mảng `c` lưu tổng của `a` và `b`.

Có thể khai báo một hằng số nguyên dương để làm số phần tử của mảng.

```
#include <conio.h>
#include <stdio.h>
#define n 5
int main()
{
    int a[n], b[n], c[n], i;
    printf("\nNhap vecto a: \n");
    for(i = 0; i < n; i++)
    {
          printf("\nNhap phan tu thu %d = ", i);
          scanf("%d", &a[i]);
    }
    printf("\nNhap vecto b: \n");
    for(i = 0; i < n; i++)
    {
          printf("\nNhap phan tu thu %d = ", i);
          scanf("%d", &b[i]);
    }
    printf("\nVecto a: \n");
    for(i = 0; i < n; i++)
          printf("%5d", a[i]);
    printf("\nVecto b: \n");
```

```
    for(i = 0; i < n; i++)
          printf("%5d", b[i]);
    //Tinh vecto c la tong cua hai vecto a va b
    for(i = 0; i < n; i++)
          c[i] = a[i] + b[i];
    printf("\nVecto tong c: \n");
    for(i = 0; i < n; i++)
          printf("%5d", c[i]);
    getch();
    return 0;
}
```

Ví dụ 4.26. Viết chương trình nhập mảng một chiều `n` số nguyên. Sắp xếp lại mảng theo thứ tự tăng dần. Hiển thị lên màn hình giá trị các phần tử của mảng trước và sau khi sắp xếp.

Giả sử mảng ban đầu là:

3	2	1	5	4

Mảng sau khi sắp xếp tăng là:

1	2	3	4	5

Thuật toán sắp xếp mảng: Có nhiều thuật toán để sắp xếp (tăng hoặc giảm) một mảng một chiều, song thuật toán trực quan nhất đối với những người mới học lập trình là thuật toán **"Sắp xếp nổi bọt" (Bubble Sort)**. Giả sử ta cần sắp xếp mảng `a` gồm `n` phần tử, thuật toán được mô tả như sau:

Bước 1: Lấy phần tử `a[0]` đem so sánh với các phần tử từ `a[1]` đến `a[n - 1]`, nếu giá trị `a[0]` lớn hơn giá trị phần tử nào thì đổi chỗ giá trị hai phần tử đó cho nhau. Như vậy, sau bước 1 giá trị phần tử `a[0]` là nhỏ nhất trong `n` phần tử của mảng (ta có thể hiểu sau bước 1 giá trị phần tử `a[0]` là nhẹ nhất nên được nổi lên).

Bước 2: Lấy phần tử `a[1]` đem so sánh với các phần từ từ `a[2]` đến `a[n - 1]`, nếu giá trị `a[1]` lớn hơn giá trị phần tử nào thì đổi chỗ giá trị hai phần tử đó cho nhau.

...

Bước n - 1: Lấy phần tử `a[n - 2]` so với `a[n - 1]`, nếu giá trị `a[n - 2]` lớn hơn giá trị `a[n - 1]` thì đổi chỗ giá trị hai phần tử cho nhau. Kết thúc bước `n - 1` mảng ban đầu trở thành mảng được sắp xếp tăng.

✍ **Lưu ý:** Việc đổi chỗ giá trị hai biến `a` và `b` cùng kiểu được thực hiện như sau: khai báo một biến trung gian `(tg)` cùng kiểu với `a` và `b` sau đó thực hiện ba câu lệnh:

```
tg = a; a = b; b = tg;
```

```
#include <conio.h>
#include <stdio.h>
int main()
{
    int a[10], i, j, n, tg;
    printf("\nNhap so phan tu cua mang n = ");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        printf("\nNhap phan tu thu %d = ", i);
        scanf("%d", &a[i]);
    }
    printf("Mang vua nhap la: \n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    for(i = 0; i < n - 1; i++)
        for(j = i + 1; j < n; j++)
```

```
            if (a[i] > a[j])
            {
                tg = a[i];
                a[i] = a[j];
                a[j] = tg;
            }
    printf("\nMang sau khi sap xep tang la: \n");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    getch();
    return 0;
}
```

4.3.3. Mảng hai chiều

Là một cấu trúc dữ liệu thuận tiện cho việc lưu trữ dữ liệu dạng ma trận trong toán học. Thực chất, trong ngôn ngữ lập trình C mảng hai chiều được hiểu là mảng một chiều mà mỗi phần tử là một mảng một chiều.

- **Khai báo mảng hai chiều với số phần tử xác định**

Cú pháp:

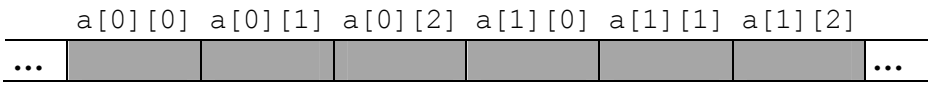
`<Kiểu> <Tên mảng>[<Số phần tử chiều 1>][<Số phần tử chiều 2>];`

Ví dụ 4.27. Để lưu trữ các giá trị của một ma trận các số thực 2 hàng, 3 cột ta sử dụng một mảng hai chiều như sau:

`float a[2][3];` /* Khai báo mảng hai chiều `a` có 6 (= 2 x 3) phần tử là các biến thực, ta có thể nói: khai báo mảng hai chiều a_{2x3} */

Sau khai báo này máy sẽ cấp phát cho mảng `a` 6 phần tử liên tiếp nhau, mỗi phần tử chiếm 4 `byte` (số `byte` của kiểu `float`).

Hình ảnh mảng a trong bộ nhớ



Như vậy, mảng hai chiều cũng được lưu trữ trong máy bởi một dãy các phần tử liên tiếp nhau.

- **Khai báo mảng hai chiều với số phần tử không xác định**

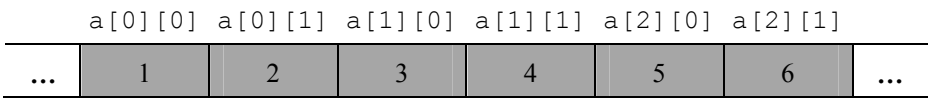
Để khai báo mảng hai chiều với số phần tử không xác định ta vẫn phải chỉ ra `Số phần tử chiều 2`.

Cú pháp: `<Kiểu> <Tên mảng>[][<Số phần tử chiều 2>];`

Cách khai báo này cũng được áp dụng trong trường hợp vừa khai báo vừa gán giá trị cho mảng hay dùng mảng hai chiều là tham số hình thức của hàm.

Ví dụ 4.28. `int a[][2] = {{1, 2}, {3, 4}, {5, 6}};`

Hình ảnh mảng a trong bộ nhớ



- **Truy nhập phần tử của mảng hai chiều**

- **Sử dụng phép toán** `[]`

Để truy nhập đến phần tử có chỉ số chiều 1 là `i`, chiều 2 là `j` ta viết: `a[i][j]`

Để truy nhập đến địa chỉ phần tử có chỉ số chiều 1 là `i`, chiều 2 là `j` ta viết: `&a[i][j]`

✍ **Lưu ý:** Vì ngôn ngữ lập trình C xem mảng hai chiều là mảng một chiều mà mỗi phần tử là một mảng một chiều, do đó mảng hai chiều tuân theo mọi tính chất của mảng một chiều.

Ví dụ 4.29. Nếu khai báo mảng `int a[2][3];`

Ta có `a[0], a[1]` là các mảng một chiều.

Do đó:

`a[0]` là hằng địa chỉ và là địa chỉ của phần tử `a[0][0]`.

`a[1]` là hằng địa chỉ và là địa chỉ của phần tử `a[1][0]`.

Ta cũng có

`a` là hằng địa chỉ và là địa chỉ của phần tử `a[0][0]` (tức là địa chỉ của phần tử đầu tiên của mảng một chiều `a[0]`).

`a + 1` là hằng địa chỉ và là địa chỉ của phần tử `a[1][0]` (tức là địa chỉ của phần tử đầu tiên của mảng một chiều `a[1]`).

- **Sử dụng con trỏ**

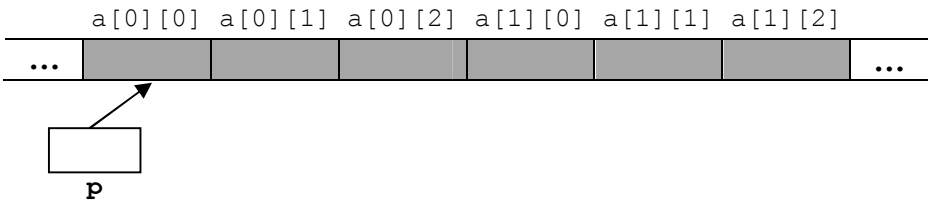
Để truy nhập đến các phần tử của mảng hai chiều `a` bằng con trỏ ta khai báo một con trỏ cùng với kiểu phần tử dữ liệu của mảng, sau đó ép kiểu địa chỉ mảng thành kiểu con trỏ và gán cho con trỏ. Lúc này, ta có thể sử dụng con trỏ để truy nhập đến các phần tử của mảng. Tuy nhiên, đối với mảng hai chiều việc truy nhập các phần tử mảng bằng con trỏ ít được sử dụng.

Ví dụ 4.30. Nếu ta khai báo:

```
float a[2][3], *p;
p = (float*)a;
```

Lúc này `p` trỏ tới phần tử `a[0][0]` và ta có thể sử dụng con trỏ `p` để truy nhập các phần tử của mảng hai chiều `a`.

Hình ảnh mảng `a` trong bộ nhớ sau phép gán `p = (float*)a`



Ví dụ 4.31. Viết chương trình nhập một ma trận các số thực gồm `m` hàng và `n` cột. Hiển thị lên màn hình ma trận vừa nhập.

Để lưu trữ một ma trận số cách tốt nhất là sử dụng dữ liệu kiểu mảng hai chiều.

Cách nhập mảng hai chiều: Sử dụng hai vòng lặp lồng nhau để có thể truy nhập đến tất cả các phần tử của mảng hai chiều.

```
#include <conio.h>
#include <stdio.h>
int main()
{
    float a[10][20], tg;
    int i, j, m, n;
    printf("\nNhap so hang m = ");
    scanf("%d", &m);
    printf("\nNhap so cot n = ");
    scanf("%d", &n);
    //Nhap cac phan tu cua mang
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
        {
            printf("\nNhap a[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
    // Hien thi mang duoi dang ma tran
    printf("\nMang vua nhap la: \n");
    for(i = 0; i < m; i++)
```

```
    {
        for(j = 0; j < n; j++)
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
    getch();
    return 0;
}
```

✍ **Lưu ý:** Trong ví dụ trên có thể khai báo hai hằng `m` và `n`, sau đó khai báo mảng hai chiều có số phần tử chiều 1 là `m` và số phần tử chiều 2 là `n` để tránh lãng phí bộ nhớ khi khai báo mảng nhiều phần tử nhưng lại sử dụng ít.

Ví dụ 4.32. Viết chương trình nhập ma trận thực `a` gồm `m` hàng, `n` cột. Hiển thị lên màn hình ma trận `a` và dạng chuyển vị của nó.

```
#include <conio.h>
#include <stdio.h>
#define m 2
#define n 3
int main()
{
    float a[m][n];
    int i, j;
    printf("Nhap ma tran a: \n");
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
        {
```

```
            printf("\nNhap a[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
    printf("\nMa tran a: \n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
    printf("\nMa tran a dang chuyen vi: \n");
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < m; j++)
            printf("%5.2f", a[j][i]);
        printf("\n");
    }
    getch();
    return 0;
}
```

Ví dụ 4.33. Viết chương trình nhập hai ma trận các số thực a_{mxn}, b_{mxn}. Tính tổng của hai ma trận. Hiển thị lên màn hình hai ma trận a, b và tổng của chúng.

```
#include <conio.h>
#include <stdio.h>
#define m 2
#define n 3
int main()
```

```
{
    float a[m][n], b[m][n], c[m][n];
    int i, j;
    printf("Nhap ma tran a: \n");
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
        {
            printf("\nNhap a[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
    printf("\nNhap ma tran b: \n");
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
        {
            printf("\nNhap b[%d][%d] = ", i, j);
            scanf("%f", &b[i][j]);
        }
    printf("\nMa tran a: \n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
    printf("\nMa tran b: \n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%5.2f", b[i][j]);
        printf("\n");
```

```
    }
    //Tinh ma tran c la tong hai ma tran a va b
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            c[i][j] = a[i][j] + b[i][j];
    printf("\nMa tran tong c: \n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%5.2f", c[i][j]);
        printf("\n");
    }
    getch();
    return 0;
}
```

Ví dụ 4.34. Viết chương trình nhập ma trận vuông các số thực a_{mxm}. Xây dựng vectơ b gồm các phần tử nằm trên đường chéo chính của ma trận a. Hiển thị lên màn hình ma trận a và vectơ b.

```
#include <conio.h>
#include <stdio.h>
#define m  3
int main()
{
    float a[m][m], b[m];
    int i, j;
    printf("\nNhap ma tran a: \n");
    for(i = 0; i < m; i++)
        for(j = 0; j < m; j++)
```

```
        {
            printf("\nNhap a[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
    printf("\nMa tran a: \n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < m; j++)
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
    //Xay dung vecto b
    for(i = 0; i < m; i++)
        b[i] = a[i][i];
    printf("\nVecto b: \n");
    for(i = 0; i < m; i++)
        printf("%5.2f", b[i]);
    getch();
    return 0;
}
```

4.3.4. Cấp phát động cho mảng

Khi sử dụng dữ liệu kiểu mảng ta có thể khai báo mảng theo cách tường minh hoặc không tường minh. Tuy nhiên, các mảng này vẫn không linh hoạt khi sử dụng bởi hai lý do sau:

- Mảng chiếm giữ bộ nhớ cho đến khi kết thúc hàm nếu là mảng được khai báo cục bộ hoặc cho đến khi kết thúc chương trình nếu là mảng được khai báo toàn cục.

- Nếu ta khai báo mảng ít phần tử mà sử dụng nhiều thì không đủ bộ nhớ, ngược lại nếu ta khai báo nhiều phần tử mà sử dụng ít thì gây ra lãng phí bộ nhớ.

Vì hai lý do trên, khi sử dụng mảng ta có thể cấp phát động cho mảng. Mảng cấp phát động có tính linh hoạt, khi cần thì cấp phát vùng nhớ để dùng, khi không dùng nữa thì có thể giải phóng vùng nhớ đó.

Cấp phát động cho mảng một chiều `n` phần tử

```
<Biến mảng> = (<Kiểu>*)malloc(n * sizeof(<Kiểu>));
```

Giải phóng vùng nhớ đã cấp cho mảng một chiều

```
free(<Biến mảng>);
```

Cấp phát động cho mảng hai chiều `m x n` phần tử

```
<Biến mảng> = (<Kiểu>**)malloc(m * sizeof(<Kiểu>*));
   for(i = 0; i < m; i++)
<Biến mảng>[i] = (<Kiểu>*)malloc(n * sizeof(<Kiểu>));
```

Giải phóng vùng nhớ đã cấp cho mảng hai chiều

```
for(i = 0; i < m; i++)
   free(<Biến mảng>[i]);
free(<Biến mảng>);
```

Ví dụ 4.35. Viết chương trình cấp phát động và nhập giá trị cho mảng một chiều `n` phần tử. Hiển thị lên màn hình giá trị các phần tử của mảng.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *a, i, n;
    printf("Nhap so phan tu cua mang n = ");
```

```
    scanf("%d", &n);
    a = (int*)malloc(n * sizeof(int));
    printf("\nNhap cac phan tu mang: ");
    for(i = 0; i < n; i++)
    {
        printf("\nNhap a[%d]=", i);
        scanf("%d", &a[i]);
    }
    printf("\nMang vua nhap: ");
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
    //Giai phong vung nho da cap cho mang
    free(a);
    getch();
    return 0;
}
```

Ví dụ 4.36. Viết chương trình cấp phát động và nhập giá trị cho mảng hai chiều a_{mxn}. Hiển thị lên màn hình giá trị các phần tử của mảng dưới dạng ma trận.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int **a, i, j, m, n;
    printf("Nhap so hang m = ");
    scanf("%d", &m);
    printf("Nhap so cot n = ");
    scanf("%d", &n);
```

```
    a = (int**)malloc(m * sizeof(int*));
    for(i = 0; i < m; i++)
        a[i] = (int*)malloc(n * sizeof(int));
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
        {
            printf("\nNhap a[%d][%d] = ", i, j);
            scanf("%d", &a[i][j]);
        }
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%5d", a[i][j]);
        printf("\n");
    }
    //Giai phong vung nho da cap cho mang
    for(i = 0; i < m; i++)
        free(a[i]);
    free(a);
    getch();
    return 0;
}
```

4.4. Xâu ký tự

4.4.1. Khái niệm xâu ký tự

Xâu ký tự là một mảng một chiều các ký tự nhưng được kết thúc bằng hằng ký tự `'\0'` (NULL). Hằng xâu ký tự được đặt trong cặp dấu nháy kép " ".

Chẳng hạn: `"Hello world"`

`""` // Xâu rỗng

4.4.2. Khai báo xâu ký tự

a. Khai báo xâu ký tự theo kiểu mảng một chiều

Cú pháp: `char <Biến xâu> [<Độ dài tối đa>];`

✍ **Lưu ý:** Độ dài tối đa của `Biến xâu` là một số nguyên nằm trong khoảng từ 1 đến 255.

Chẳng hạn: `char s[10];`

Với khai báo này, chương trình sẽ cấp 11 `byte` sẵn sàng để lưu trữ nội dung của xâu `s`. `Byte` ngay sau `byte` cuối cùng chứa nội dung xâu `s` sẽ lưu hằng ký tự `'\0'`.

- **Vừa khai báo vừa khởi tạo giá trị**

Cú pháp: `char <Biến xâu>[]= "<Hằng xâu ký tự>";`

Chẳng hạn: `char hoten[] = "Nguyen Van An";`

✍ **Lưu ý:** Xâu ký tự tuân theo các tính chất của mảng một chiều.

b. Khai báo xâu ký tự theo kiểu con trỏ

Cú pháp: `char *<Biến xâu> ;`

Chẳng hạn: `char *s;`

Với khai báo này, máy sẽ dành 2 `byte` để lưu trữ địa chỉ của biến con trỏ nhưng chưa cấp vùng nhớ để lưu trữ dữ liệu. Muốn có vùng nhớ để lưu trữ dữ liệu ta phải sử dụng các hàm `malloc` hoặc `calloc` để cấp phát.

Chẳng hạn: `s = (char*)calloc(10, sizeof(char));`

Lúc này `s` là xâu có độ dài tối đa là `10`.

- **Vừa khai báo vừa khởi tạo giá trị**

Cú pháp: `char *<Biến xâu> = <Giá trị>;`

Chẳng hạn: `char *s = "Hello";`

4.4.3. Các hàm xử lý ký tự và xâu ký tự

Ngôn ngữ lập trình C cung cấp nhiều hàm khai báo sẵn trong các thư viện chuẩn để xử lý ký tự và xâu ký tự. Sau đây là một số hàm thường dùng:

- **Hàm đổi chữ cái hoa thành chữ cái thường**

Muốn đổi một chữ cái hoa thành chữ cái thường ta sử dụng hàm `tolower` được khai báo trong thư viện `ctype.h`.

Cú pháp: `int tolower(int ch);`

- **Hàm đổi chữ cái thường thành chữ cái hoa**

Muốn đổi một chữ cái thường thành chữ cái hoa ta sử dụng hàm `toupper` được khai báo trong thư viện `ctype.h`.

Cú pháp: `int toupper(int ch);`

- **Hàm nhập xâu**

Để nhập một xâu ký tự vào từ bàn phím ta sử dụng hàm `gets` được khai báo trong thư viện `string.h`. Kết thúc nhập xâu bằng phím `Enter`.

Cú pháp: `char *gets(<Biến xâu>);`

Chẳng hạn:
```
char hoten[30];
gets(hoten);
```

✍ **Lưu ý:**

- Ta cũng có thể sử dụng hàm `scanf` để nhập dữ liệu cho biến xâu ký tự, tuy nhiên lúc này ta chỉ có thể nhập được những xâu không có dấu cách trống vì nếu nhập dấu cách trống máy sẽ hiểu là kết thúc nhập xâu;
- Trước khi nhập xâu ký tự ta phải làm sạch vùng đệm bằng cách sử dụng `fflush(stdin)` được khai báo trong thư viện `stdio.h`.

- **Hàm hiển thị xâu**

Để hiển thị một xâu ký tự lên màn hình ta sử dụng hàm `puts` được khai báo trong thư viện `string.h`.

Cú pháp: `int puts(s);` // Với `s` là hằng xâu hoặc biến xâu.

Chẳng hạn:
```
char hoten[] = "Nguyen Van An";
printf("\nXau ho ten la: ");
puts(hoten);
```

✍ **Lưu ý:** Ta cũng có thể sử dụng hàm `printf` để hiển thị xâu ký tự lên màn hình với mã định dạng là `%s`.

Chẳng hạn: `printf("\nXau ho ten la: %s", hoten);`

- **Hàm xác định độ dài xâu**

Để lấy độ dài thực tế của một xâu ta sử dụng hàm `strlen` được khai báo trong thư viện `string.h`.

Cú pháp: `size_t strlen(s);` // Với `s` là hằng xâu hoặc biến xâu.

Trong đó: `size_t`: Là kiểu số nguyên không dấu được định nghĩa trong thư viện `stdio.h`.

Chẳng hạn:
```
char hoten[30] = "Nguyen Van An";
int n = strlen(hoten);      //n có giá trị là 13
```

- **Hàm ghép xâu**

Để ghép hai xâu thành một xâu ta sử dụng hàm `strcat` được khai báo trong thư viện `string.h`.

Cú pháp:
```
char *strcat(char  *<Xâu đích>, const char *<Xâu nguồn>);
```

Ví dụ 4.37. Viết chương trình nhập vào hai xâu họ lót và tên của một người. Ghép xâu tên vào xâu họ lót để được xâu họ và tên đầy đủ.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main()
{
    char holot[20], ten[10];
    printf("Nhap Ho lot: ");
    fflush(stdin);
    gets(holot);
    printf("Nhap Ten: ");
    fflush(stdin);
    gets(ten);
    strcat(holot, " "); // Ghep dau cach sau ho lot
    strcat(holot, ten); // Ghep ten vao holot
    printf("Ho ten la: ");
    puts(holot);
    getch();
    return 0;
}
```

- **Hàm đổi xâu chữ cái thường thành xâu chữ cái hoa**

Muốn đổi một xâu chữ cái thường thành xâu chữ cái hoa ta sử dụng hàm `strupr` được khai báo trong thư viện `string.h`.

Cú pháp: `char* strupr(char *s);`

- **Hàm đổi xâu chữ cái hoa thành xâu chữ cái thường**

Muốn đổi một xâu chữ cái hoa thành xâu chữ cái thường ta sử dụng hàm `strlwr` được khai báo trong thư viện `string.h`.

Cú pháp: `char* strlwr(char *s);`

- **Hàm sao chép xâu**

Muốn sao chép nội dung của xâu nguồn sang xâu đích ta sử dụng hàm *strcpy* được khai báo trong thư viện `string.h`.

Cú pháp: `char *strcpy(char *<Xâu đích>, const char *<Xâu nguồn>);`

Chẳng hạn:

```
char s[] = "Hello", t[10];
strcpy(t, s);
puts(t);
```

✍ **Lưu ý:** Trong ngôn ngữ lập trình C, việc gán trực tiếp hai xâu khai báo dạng mảng là không hợp lệ.

Chẳng hạn:

```
char s[10];
s = "Hello"; // Không hợp lệ
char t[10];
t = s;       // Không hợp lệ
```

Sở dĩ việc gán hai xâu khai báo dạng mảng là không hợp lệ vì bản chất tên mảng là hằng địa chỉ. Nhưng nếu khai báo xâu dạng con trỏ thì có thể gán.

Chẳng hạn:

```
char *t;
t = "Hello";  // Hợp lệ
```

- **Hàm trích một phần xâu**

Muốn trích một xâu con từ một xâu bắt đầu từ một ký tự cụ thể cho đến hết xâu ta sử dụng hàm `strchr` được khai báo trong thư viện `string.h`.

Cú pháp: `char *strchr(const char *str, int c)`

Nếu thành công hàm trả về con trỏ trỏ vào ký tự `c` đầu tiên trong xâu `str`, ngược lại (ký tự `c` không có trong xâu `str`) hàm trả về `NULL`.

- **Hàm tìm kiếm xâu**

Muốn tìm kiếm sự có mặt của xâu `s2` trong xâu `s1` ta sử dụng hàm `strstr` được khai báo trong thư viện `string.h`.

Cú pháp: `char *strstr(const char *s1, const char *s2);`

Nếu thành công hàm trả về con trỏ trỏ tới vị trí đầu tiên mà xâu `s2` có mặt trong xâu `s1`, ngược lại (xâu `s2` không có mặt trong xâu `s1`) hàm trả về `NULL`.

- **Hàm so sánh hai xâu**

Để so sánh hai xâu ký tự theo bảng mã ASCII ta sử dụng hàm `strcmp` được khai báo trong thư viện `string.h`.

Cú pháp: `int strcmp(const char *s1, const char *s2);`

Kết quả của hàm là hiệu của hai ký tự bắt đầu khác nhau trong xâu `s1` và `s2` tính từ trái sang phải (*có phân biệt chữ hoa và chữ thường*).

Nếu kết quả > `0` nghĩa là `s1` lớn hơn `s2`.

Nếu kết quả = `0` nghĩa là `s1` bằng `s2`.

Nếu kết quả < `0` nghĩa là `s1` bé hơn `s2`.

- **Hàm chuyển đổi xâu số thành số**

Để chuyển đổi xâu số ra số ta sử dụng các hàm `atoi, atof, atol` được khai báo trong thư viện `stdlib.h`.

Cú pháp:

```
int atoi(const char *s); // Đổi xâu số s thành số nguyên
long atol(const char *s); // Đổi xâu số s thành số nguyên long
float atof(const char *s); // Đổi xâu số s thành số thực
```

Nếu chuyển đổi không thành công hàm trả về giá trị `0`.

4.5. Mảng các con trỏ

Bản thân các con trỏ là các biến, vì vậy chúng có thể được lưu trữ vào một mảng như các biến bình thường. Để khai báo một mảng một chiều các con trỏ ta sử dụng cú pháp sau:

Cú pháp: `<Kiểu> *<Biến mảng>[<Số phần tử>];`

Chẳng hạn: `int *a[5];`

Lúc này ta có một mảng 5 phần tử `a[0]`, `a[1]`, `a[2]`, `a[3]`, `a[4]`, mỗi phần tử là một con trỏ kiểu `int`.

Ví dụ 4.38. Viết chương trình in ra màn hình tên của tháng trong năm khi nhập vào số hiệu của tháng.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main()
{
   int n;
   char *name[13] =  {
              "Khong hop le", "Thang mot",
              "Thang hai", "Thang ba", "Thang tu",
              "Thang nam", "Thang sau",
              "Thang bay", "Thang tam",
              "Thang chin", "Thang muoi",
              "Thang muoi mot", "Thang muoi hai"
                     };
   char  ten[50];
   printf("\nNhap so hieu thang n = ");
   scanf("%d", &n);
   strcpy(ten, (n < 1) || ( n > 12) ? name[0] : name[n]);
```

```
    printf("\n%s", ten);
    getch();
    return 0;
}
```

BÀI THỰC HÀNH CHƯƠNG 4

- **Mục tiêu**

  Qua bài thực hành người học cần đạt được:

  - Về kiến thức

  + Nắm ý nghĩa, cách khai báo, các phép toán đối với con trỏ;
  + Nắm được ý nghĩa, cách khai báo mảng một chiều, xâu ký tự, mảng hai chiều;
  + Cách truy nhập đến các phần tử của mảng;
  + Cách nhập mảng, hiển thị mảng;
  + Nắm được một số thuật toán xử lý mảng, xử lý xâu.

  - Về kỹ năng

  + Vận dụng kiến thức về con trỏ và mảng để giải quyết một số bài toán có dữ liệu phù hợp với kiểu mảng, chẳng hạn các bài toán về dãy số, vectơ, ma trận,…

- **Nội dung**

Bài 1. Gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
int main()
{
      int a[5] = {10, 20, 30, 40, 50};
      printf("\nCac phan tu cua mang:\n");
      printf("%5d", a[0]);
      printf("%5d", a[1]);
      printf("%5d", a[2]);
```

```
      printf("%5d", a[3]);
      printf("%5d", a[4]);
      getch();
      return 0;
}
```

Yêu cầu:

+ Chỉ ra biến mảng, cách khai báo biến mảng trong chương trình.
+ Cách hiển thị giá trị các phần tử của mảng như chương trình trên có thuận tiện, đặc biệt có phù hợp với các mảng có số phần tử lớn không?
+ Viết lại đoạn chương trình hiển thị giá trị các phần tử của mảng theo cách thuận tiện hơn.

Bài 2. Gõ vào chương trình sau:

```
include <conio.h>
#include <stdio.h>
int main()
{
      int a[] = {1, -2, 0, 4, -5, 3, 7};
      int i, n;
      n = sizeof(a) / sizeof(int);
      for(i = 0; i < n; i++)
            printf("%5d", a[i]);
      for(i = 0; i < n; i++)
            printf("\n%u", &a[i]);
      getch();
      return 0;
}
```

Yêu cầu: Cho biết chương trình trên thực hiện những công việc gì? Tương ứng với những câu lệnh nào?

Bài 3. Gõ vào chương trình giải quyết bài toán: nhập mảng một chiều gồm `n` số thực, với `n` nguyên dương được nhập vào từ bàn phím. Hiển thị lên màn hình giá trị các phần tử của mảng và tổng của chúng.

Yêu cầu:

+ Chỉ ra câu lệnh làm nhiệm vụ tính tổng giá trị các phần tử của mảng;

+ Chỉ ra câu lệnh thông báo tổng giá trị các phần tử của mảng;

+ Biên dịch và chạy chương trình để kiểm tra kết quả.

```
#include <conio.h>
#include <stdio.h>
int main()
{
      float a[10], s = 0;
      int i, n;
      printf("\nNhap so phan tu cua mang n = ");
      scanf("%d", &n);
      printf("\nNhap mang:\n");
      for(i = 0; i < n; i++)
      {
            printf("\nNhap a[%d] = ", i);
            scanf("%f", &a[i]);
      }
      printf("\nMang vua nhap:\n");
      for(i = 0; i < n; i++)
            printf("%5.2f", a[i]);
```

```
      for(i = 0; i < n; i++)
            s += a[i];
      printf("\n%5.2f", s);
      getch();
      return 0;
}
```

Bài 4. Viết chương trình nhập mảng một chiều gồm `n` số thực, với `n` nguyên dương nhập vào từ bàn phím. Hiển thị lên màn hình giá trị lớn nhất và giá trị bé nhất của mảng.

Yêu cầu:

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán tìm giá trị lớn nhất và giá trị bé nhất của mảng;

+ Viết chương trình.

Bài 5. Viết chương trình nhập mảng một chiều gồm `n` số nguyên, với `n` nguyên dương nhập vào từ bàn phím. Hiển thị lên màn hình giá trị các phần tử của mảng và trung bình cộng của chúng.

Yêu cầu:

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;

+ Vẽ lưu đồ thuật toán tính trung bình cộng của các phần tử của mảng;

+ Viết chương trình.

Bài 6. Gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
int main()
{
```

```
      char x[100];
      x[0] = 'a';
      x[1] = 'n';
      x[2] = 'h';
      printf("%s", x);
      getch();
      return 0;
}
```

Yêu cầu:

+ Biên dịch, chạy chương trình;

+ Kiểm tra kết quả để xác định lỗi thuật toán;

+ Sửa lỗi, biên dịch và chạy lại chương trình, kiểm tra kết quả.

Bài 7. Gõ vào chương trình sau:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main()
{
      int dem = 0, i = 0, n;
      char s[20] = "AbcdEFGH";
      n = strlen(s);
      for(i = 0; i < n; i++)
            if ((s[i] >= 'A') && (s[i] <= 'Z'))
                  dem++;
      printf("%5d", dem);
      getch();
      return 0;
}
```

Yêu cầu: Cho biết đoạn chương trình làm những công việc gì? Mỗi công việc đó tương ứng với những câu lệnh nào?

Bài 8. Gõ vào chương trình giải quyết bài toán: viết chương trình khai báo một mảng hai chiều a_{2x3} gồm các số nguyên với các giá trị cho sẵn. Hiển thị lên màn hình giá trị các phần tử của mảng dưới dạng ma trận.

Yêu cầu:

+ Biên dịch, chạy chương trình;

+ Bỏ cặp ngoặc **{ }** (`// 1` và `// 2`), biên dịch, chạy chương trình và cho nhận xét.

```
#include <conio.h>
#include <stdio.h>
int main()
{
      int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
      int i, j;
      printf("\nMang a:\n");
      for(i = 0; i < 2; i++)
      {     // 1
            for(j = 0; j < 3; j++)
                  printf("%5d", a[i][j]);
            printf("\n");
      }     // 2
      getch();
      return 0;
}
```

Bài 9. Viết chương trình nhập mảng hai chiều gồm `m x n` số thực, với hai số nguyên `m` và `n` được nhập từ bàn phím. Hiển thị lên màn hình giá trị các phần tử của mảng dưới dạng ma trận và tổng các giá trị dương chẵn của mảng.

Yêu cầu:

+ Xác định số biến, ý nghĩa các biến, kiểu các biến;
+ Vẽ lưu đồ thuật toán tính tổng các phần tử dương chẵn của mảng hai chiều;
+ Viết chương trình.

- **Câu hỏi củng cố**

  - Có thể dùng con trỏ kiểu `int` để lưu địa chỉ của một biến kiểu `float` không?
  - Lợi ích của việc sử dụng con trỏ?
  - Khi khai báo một mảng gồm 5 phần tử thì chỉ số các phần tử được xác định như thế nào?
  - Khai báo `int a[10]` thì `a` là địa chỉ của phần tử `a[0]`, như vậy `a++` là địa chỉ của phần tử nào?
  - Mảng hai chiều có phải là mảng một chiều không? Nó có tuân theo các quy tắc của mảng một chiều không?

- **Tự học**

Bài 10. Viết chương trình nhập mảng một chiều `a` gồm các số nguyên. Xây dựng mảng một chiều `b` gồm các giá trị đại diện lấy từ mảng một chiều a.

Chẳng hạn:

Mảng `a`: `1 2 3 2 3 5 1`

Mảng `b`: `1 2 3 5`

Gợi ý: Lấy giá trị phần tử `a[i]` (`i = 0, 1, …, n - 2`) so sánh với giá trị các phần tử còn lại của mảng, nếu giá trị `a[i]` bằng giá trị một phần tử nào đó thì bỏ qua giá trị `a[i]`, ngược lại giá trị `a[i]` được đưa vào mảng `b`. Cuối cùng đưa giá trị phần từ `a[n - 1]` vào mảng `b`.

Bài 11. Viết chương trình nhập mảng một chiều n số nguyên, sắp xếp mảng theo thứ tự tăng dần. Nhập một số nguyên x vào từ bàn phím, chèn giá trị x vào mảng sao cho không làm thay đổi thứ tự sắp xếp của mảng.

Gợi ý: Khi chèn một phần tử vào mảng có thể xảy ra hai khả năng:

+ Phần tử được thêm vào sau cùng của mảng;

+ Phần tử được chèn vào đầu hoặc giữa mảng.

Với mỗi khả năng tìm cách chèn hợp lý.

Bài 12. Viết chương trình nhập vào hai dãy số nguyên $a_1, a_2, ..., a_n$ và $b_1, b_2, ..., b_n$. Kiểm tra xem có phải hai dãy trên chỉ khác nhau về trật tự sắp xếp các phần tử không?

Gợi ý: Có thể sắp xếp hai mảng theo cùng trật tự (tăng hoặc giảm) sau đó kiểm tra từng cặp phần tử của hai mảng.

Bài 13. Viết chương trình nhập vào từ bàn phím một xâu ký tự s. Đếm xem mỗi loại chữ cái từ 'A' đến 'Z' xuất hiện bao nhiêu lần trong xâu s (*không phân biệt chữ hoa chữ thường*).

Chẳng hạn: Với xâu "AbcDeaB" thì có 2 ký tự 'A'; 2 ký tự 'B'; 1 ký tự 'C'; 1 ký tự 'D'; 1 ký tự 'E'.

Gợi ý: Có thể sử dụng vòng lặp for để duyệt từng ký tự của xâu kết hợp câu lệnh switch để đếm từng loại ký tự.

Bài 14. Viết chương trình nhập mảng hai chiều gồm m x n số nguyên. Hiển thị lên màn hình giá trị các phần tử của mảng và tổng của các phần tử trên mỗi hàng của mảng.

Bài 15. Viết chương trình nhập hai ma trận $a_{m \times n}$, $b_{n \times p}$ gồm các số nguyên. Hãy tính ma trận $c_{m \times p}$ là tích của hai ma trận trên. Hiển thị lên màn hình các ma trận a, b và c.

Chương 5

HÀM

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Khái niệm hàm;*
- *Lợi ích của việc tổ chức chương trình thành các hàm;*
- *Cách xây dựng và sử dụng hàm;*
- *Nguyên tắc hoạt động của hàm;*
- *Cách truyền tham số trong hàm;*
- *Hàm đệ quy.*

Các tài liệu tham khảo chính của chương bao gồm [1, 4, 6, 7, 8].

5.1. Khái niệm hàm trong ngôn ngữ lập trình C

Trong lập trình, đối với những chương trình phức tạp ta có thể chia nhỏ thành các mô đun, mỗi mô đun thực hiện một chức năng nhất định. Các mô đun này được gọi là các chương trình con, chúng càng độc lập với nhau càng tốt. Ngôn ngữ lập trình C gọi các chương trình con là các hàm. Một chương trình C có thể chỉ có một hàm cũng có thể có nhiều hàm nhưng bắt buộc phải có một hàm chính với tên quy định là `main`, nội dung của hàm `main` được gọi là chương trình chính. Mỗi chương

trình C có duy nhất một hàm có tên là `main`, chương trình luôn bắt đầu thực thi từ hàm này.

Ngôn ngữ lập trình C phân hàm thành hai loại chính là hàm được xây dựng sẵn và hàm do người lập trình xây dựng. Chương này sẽ chú trọng giới thiệu loại hàm do người lập trình xây dựng, bao gồm cách khai báo, định nghĩa và sử dụng hàm.

Ưu điểm của việc tổ chức chương trình thành các chương trình con:

- Làm cho chương trình được rõ ràng hơn, đặc biệt là những chương trình phức tạp và có nhiều dòng lệnh;
- Những đoạn chương trình được lặp đi lặp lại nhiều lần ta có thể tổ chức đoạn chương trình đó thành một chương trình con và sẽ sử dụng nó khi cần thiết;
- Người lập trình có thể dễ kiểm tra tính đúng đắn của các chương trình con trước khi sử dụng nó;
- Việc tổ chức thành các chương trình con làm cho chương trình có tính mô đun hóa cao và thuận lợi trong việc triển khai viết chương trình theo nhóm.

5.2. Hàm xây dựng sẵn

Trong ngôn ngữ lập trình C có rất nhiều hàm được xây dựng sẵn đặt trong các thư viện chuẩn, người lập trình chỉ cần khai báo thư viện chứa hàm cần dùng ở đầu chương trình thì có thể sử dụng chúng. Tuy nhiên, để sử dụng các hàm này một cách chính xác phải biết cú pháp của từng hàm và sử dụng chúng theo đúng cú pháp.

Ý nghĩa của một số thư viện thường dùng

- **Thư viện stdio.h (standard input output)** chứa các hàm vào/ra chuẩn: `printf, scanf, getc, putc, gets, puts, fflush, fopen, fclose, fwrite, getchar, putchar, getw, putw,…`

alloc.h	dirent.h	iomanip.h	share.h	strstrea.h
assert.h	dos.h	iostream.h	signal.h	sys\stat.h
bcd.h	errno.h	limits.h	stdarg.h	sys\timeb.h
bios.h	fcntl.h	locale.h	stddef.h	sys\types.h
complex.h	float.h	malloc.h	stdio.h	time.h
conio.h	fstream.h	math.h	stdiostr.h	values.h
ctype.h	generic.h	mem.h	stdlib.h	
dir.h	graphics.h	process.h	stream.h	
Direct	io.h	setjmp.h	string.h	

Bảng 5.1. Một số thư viện chuẩn của ngôn ngữ lập trình C

- **Thư viện conio.h (console input output)** chứa các hàm vào/ra trong chế độ DOS: `clrscr`, `getch`, `getche`, `getpass`, `cgets`, `cputs`, `putch`.
- **Thư viện math.h (mathematics)** chứa các hàm toán học: `abs`, `sqrt`, `log`, `sin`, `cos`, `tan`, `acos`, `asin`, `atan`, `pow`, `exp`,...
- **Thư viện alloc.h (allocation*)*** hoặc **stdlib.h (standard library)** chứa các hàm liên quan đến việc quản lý bộ nhớ: `alloc`, `realloc`, `malloc`, `free`, `farmalloc`, `farcalloc`, `farfree`,...
- **Thư viện graphic.h** chứa các hàm liên quan đến đồ họa: `circle`, `putpixel`, `getpixel`, `setcolor`,...

5.3. Hàm do người lập trình xây dựng

5.3.1. Khai báo và định nghĩa hàm

Cú pháp:

```
<Kiểu hàm>  <Tên hàm> ([<Kiểu 1> < Đối số 1 >] [,<Kiểu 2 > <Đối số 2 >] [,…])
```

```
{ [Khai báo biến cục bộ]
        <Thân hàm>
  [return <Biểu thức>;]
}
```

Trong đó:

- `Kiểu hàm`: Có thể là các kiểu dữ liệu cơ bản hoặc là các kiểu do người lập trình tự định nghĩa như kiểu con trỏ, kiểu mảng, kiểu cấu trúc,…;
- `Tên hàm`: Được đặt theo quy tắc đặt tên của ngôn ngữ lập trình C;
- Một hàm không có kết quả trả về ta dùng từ khóa `void`. Lúc này trong hàm **không có câu lệnh** `return <Biểu thức>;`
- Các `Đối số 1`, `Đối số 2`, … còn được gọi là các tham số hình thức (*gọi tắt là tham số*) của hàm, `Kiểu 1`, `Kiểu 2`,... là kiểu của các tham số tương ứng;
- Hàm có thể không có tham số nào, trong trường hợp này, khi khai báo và định nghĩa hàm ta nên sử dụng từ khóa `void` trong phần khai báo tham số để tránh nhầm lẫn;
- Câu lệnh `return <Biểu thức>` sẽ trả giá trị của biểu thức về cho hàm (*nếu hàm có kiểu hàm khác* `void`), giá trị trả về phải cùng kiểu với `Kiểu hàm`.

✍ **Lưu ý:** Trong Dev-C++ quy định hàm `main` phải có kiểu `int`. Vì vậy, trước khi kết thúc hàm `main` ta cần có câu lệnh `return` để trả giá trị về cho hàm. Giá trị trả về là một số nguyên thuộc phạm vi của `int`, thường ta `return 0`. Đối với Visual C++ hoặc Turbo C++ thì hàm `main` có kiểu `int` hoặc `void`, nếu kiểu hàm là `void` thì không có câu lệnh `return` trước khi kết thúc hàm.

Ví dụ 5.1. Viết hàm tìm số lớn nhất của hai số nguyên `a` và `b`.

```
/* Cách 1: Kiểu hàm là int; Tên hàm là max; Hai đối số a, b có
   kiểu int */
```

```
int max(int a, int b)
{
    return (a > b) ? a : b;
}
```

/* Cách 2: `Kiểu hàm` là `void`; `Tên hàm` là `max`; Hai đối số `a, b` có kiểu `int` */

```
void max(int a, int b)
{
    printf("\nGia tri lon nhat la: %5d",
                            (a > b) ? a : b);
}
```

✍ **Lưu ý:** Khi khai báo đối số của hàm thì đối số nào đi kèm với kiểu của nó, ngay cả khi các đối số cùng kiểu cũng **không được khai báo gộp**.

Chẳng hạn, khai báo sau không hợp lệ: `int max(int a, b);`

5.3.2. Cách gọi hàm

Cách gọi hàm không tham số: `<Tên hàm>();`

Chẳng hạn:
```
getch();
getchar();
```

Cách gọi hàm có tham số:

```
<Tên hàm>(Danh sách tham số thực sự);
```

Chẳng hạn:

```
sqrt(4);  // Gọi hàm chuẩn của ngôn ngữ lập trình C
max(3, 5); // Gọi hàm max trong Ví dụ 5.1
```

Tham số thực sự là các tham số có giá trị thực tế sẽ truyền vào cho các đối số của hàm khi có lời gọi hàm.

Chẳng hạn:
```
int a = 5, b = 7;
max(a, b);     // a, b là các tham số thực sự
```

5.3.3. Phạm vi của biến

- **Biến toàn cục (biến tổng thể):** Là biến được khai báo ngoài tất cả các hàm (kể cả hàm `main`). Các biến này chiếm giữ bộ nhớ từ khi được cấp phát cho đến khi kết thúc chương trình.
- **Biến cục bộ (biến địa phương):** Là biến được khai báo trong một hàm hoặc một khối lệnh nào đó. Các biến này chỉ có hiệu lực trong hàm hoặc khối lệnh khai báo nó. Nó chiếm giữ bộ nhớ từ khi được cấp phát trong hàm hoặc khối lệnh và được giải phóng khi kết thúc thực hiện hàm hoặc khối lệnh.
- **Tham số hình thức của hàm:** Là những biến được khai báo trong cặp ngoặc ngay sau tên hàm. Khi có lời gọi hàm chúng được thực hiện việc truyền tham số. Tham số hình thức chiếm giữ bộ nhớ từ khi bắt đầu thực hiện hàm và được giải phóng khi kết thúc thực hiện hàm.

5.3.4. Nguyên tắc hoạt động của hàm

Khi gặp một lời gọi hàm trong chương trình thì hàm sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau:

- Lưu các trạng thái đang thực hiện;
- Lưu địa chỉ trả về;
- Cấp phát bộ nhớ cho các biến cục bộ;
- Gán giá trị của các tham số thực cho các đối số tương ứng;
- Thực hiện các câu lệnh trong thân hàm;
- Khi gặp câu lệnh `return` hoặc dấu `}` cuối cùng của thân hàm thì máy sẽ giải phóng vùng nhớ của các đối số, biến cục bộ và thoát khỏi hàm;
- Nếu trả về từ một câu lệnh `return` có chứa biểu thức thì giá trị của biểu thức được gán cho hàm;
- Khôi phục các trạng thái đã lưu giữ và thực hiện tiếp chương trình.

✍ **Lưu ý:**

- Do đối số và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối số và biến cục bộ cần có tên khác nhau;
- Đối số và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được gọi và được giải phóng khi ra khỏi hàm nên không thể lấy trực tiếp giá trị của đối số ra khỏi hàm;
- Tên đối số và biến cục bộ có thể trùng với các đại lượng ngoài hàm mà không gây nhầm lẫn;
- Khi khai báo hàm không có `Kiểu hàm` thì nó được mặc định là kiểu `int`.

5.4. Nguyên mẫu của hàm

Hàm được gọi phải được khai báo và định nghĩa trước lời gọi hàm. Tuy nhiên, nếu muốn hàm được gọi được định nghĩa ở vị trí bất kỳ (*có thể sau lời gọi hàm*) thì chúng ta phải khai báo nguyên mẫu của hàm và đặt trước lời gọi hàm. Nguyên mẫu của hàm thực chất là dòng đầu tiên của phần định nghĩa hàm và thêm vào dấu chấm phẩy (`;`). Khi khai báo nguyên mẫu của hàm có thể bỏ tên các đối số.

Ví dụ 5.2. Cách sử dụng nguyên mẫu của hàm.

```
#include <conio.h>
#include <stdio.h>
//==============================
float max_3so(float a, float b, float c);/* Nguyen
mau ham max_3so */
//==============================
int main()
{
    float  x, y, z;
```

```
    printf("\nNhap x, y, z = ");
    scanf("%f%f%f", &x, &y, &z);
    printf("\nMax la: %5.2f", max_3so(x, y, z));
                /* x, y, z la cac tham so thuc su */
    getch();
    return 0;
}
//==============================
// Dinh nghia ham max_3so
float max_3so(float a, float b, float c)
{
    float max, max1;
    max = (max1 = a > b ? a : b) > c ? max1 : c;
    return max;
}
```

5.5. Truyền tham số

Khi có lời gọi hàm, tham số thực sự sẽ truyền cho các đối số của hàm theo một trong hai cách sau:

- Truyền tham trị
- Truyền địa chỉ

5.5.1. Truyền tham trị

Khi có lời gọi hàm, giá trị của các tham số thực sự sẽ được truyền cho các đối số của hàm. Vì vậy, mọi sự thay đổi giá trị của đối số trong hàm không làm ảnh hưởng đến giá trị của tham số thực sự. Để thực hiện việc truyền tham trị, các đối số của hàm được khai báo như khai báo biến bình thường.

Ví dụ 5.3. Cách truyền tham số theo kiểu truyền tham trị.

```
#include <conio.h>
#include <stdio.h>
//============================
void cong_them(float a, float b)
{
    a = a + 10;
    b = b + 10;
}
//============================
int main()
{
    float x = 1, y = 2;
    printf("\nTruoc khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    cong_them(x, y);
    printf("\nSau khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    getch();
    return 0;
}
```

Kết quả chạy chương trình:

```
Truoc khi goi ham cong_them x = 1.00  y = 2.00
Sau khi goi ham cong_them x = 1.00 y = 2.00
```

5.5.2. Truyền địa chỉ

Khi có lời gọi hàm, địa chỉ của các tham số thực sự sẽ được truyền cho các đối số của hàm. Vì vậy, mọi sự thay đổi giá trị của đối số trong hàm dẫn tới sự thay đổi giá trị của tham số thực sự đã truyền cho hàm.

Để thực hiện việc truyền tham số bằng địa chỉ các đối số của hàm được khai báo là các con trỏ.

Ví dụ 5.4. Cách truyền tham số theo kiểu truyền địa chỉ.

```
#include <conio.h>
#include <stdio.h>
//================================
void cong_them(float *a, float *b)
{
    *a = *a + 10;
    *b = *b + 10;
}
//================================
int main()
{
    float x = 1, y = 2;
    printf("\nTruoc khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    cong_them(&x, &y);
    printf("\nSau khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    getch();
    return 0;
}
```

Kết quả chạy chương trình:

```
Truoc khi goi ham cong_them x = 1.00  y = 2.00
Sau khi goi ham cong_them x = 11.00 y = 12.00
```

✍ **Lưu ý:** Khi khai báo một hàm nhiều đối số chúng ta cần xác định xem đối số nào cần truyền tham trị và đối số nào cần truyền địa chỉ để khai báo cho phù hợp.

Ví dụ 5.5. Kết hợp giữa truyền tham trị và truyền địa chỉ.

```
#include <conio.h>
#include <stdio.h>
//==============================
void cong_them(float *a, float b)
{
    *a = *a + 10;
    b = b + 10;
}
//==============================
int main()
{
    float x = 1, y = 2;
    printf("\nTruoc khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    cong_them(&x, y);
    printf("\nSau khi goi ham cong_them x = %5.2f
              y = %5.2f", x, y);
    getch();
    return 0;
}
```

Kết quả sau khi chạy chương trình:

```
Truoc khi goi ham cong_them x = 1.00  y = 2.00
Sau khi goi ham cong_them x = 11.00 y = 2.00
```

5.6. Mảng tham gia làm đối số của hàm

Khi mảng một chiều hoặc mảng hai chiều tham gia làm đối số của hàm thì chúng phải được khai báo dưới dạng con trỏ hoặc dưới dạng không tường minh.

5.6.1. Mảng một chiều tham gia làm đối số của hàm

Nếu mảng một chiều tham gia làm đối số của hàm thì phải được khai báo theo một trong hai cách sau:

Cách 1: `<Tên mảng>[]`

Cách 2: `*<Tên mảng>`

Ví dụ 5.6. Viết lại chương trình của *Ví dụ 4.25: nhập vào hai vectơ cùng chiều gồm các số nguyên. Tính tổng của hai vectơ. Hiển thị lên màn hình hai vectơ và vectơ tổng,* bằng cách khai báo, định nghĩa và sử dụng các hàm nhằm làm cho chương trình rõ ràng hơn, tránh việc phải viết lặp đi lặp lại các đoạn chương trình giống hệt nhau, hoặc các đoạn chương trình có thuật toán tương tự nhau.

Ta chia bài toán thành các hàm thực hiện các chức năng độc lập bao gồm: hàm nhập vectơ (tương ứng hàm nhập mảng một chiều, hàm hiển thị vectơ (hàm hiển thị mảng một chiều), hàm cộng hai vectơ (hàm cộng hai mảng một chiều) và hàm `main`.

```
#include <conio.h>
#include <stdio.h>
#define m 5
//=======================================
void nhap_mang(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        printf("Nhap phan tu thu %d = ", i);
        scanf("%d", &a[i]);
    }
}
//=======================================
```

```
void hien_thi_mang(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);

}
//=======================================
void tong(int a[], int b[], int c[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}
//=======================================
int main()
{
    int x[m], y[m], z[m];
    printf("\nNhap mang x: \n");
    nhap_mang(x, m);
    printf("\nNhap mang y: \n");
    nhap_mang(y, m);
    printf("\nMang x: \n");
    hien_thi_mang(x, m);
    printf("\nMang y: \n");
    hien_thi_mang(y, m);
    tong(x, y, z, m);
    printf("\nMang tong z: \n");
    hien_thi_mang(z, m);
    getch();
    return 0;
}
```

✍ **Lưu ý:** Khi chương trình có xây dựng nhiều hàm thì nên sử dụng cách khai báo nguyên mẫu của hàm nhằm làm cho chương trình rõ ràng hơn, dễ theo dõi hơn.

Ví dụ 5.7. Ta có thể viết lại chương trình của *Ví dụ 5.6* bằng cách sử dụng nguyên mẫu của hàm.

```
#include <conio.h>
#include <stdio.h>
#define m 5
//=========================================
void nhap_mang(int a[], int n);
     // Khai bao nguyen mau cua ham nhap_mang
void hien_thi_mang(int a[], int n);
     // Khai bao nguyen mau cua ham hien_thi_mang
void tong_mang(int a[], int b[], int c[], int n);
     // Khai bao nguyen mau cua ham tong_mang
//=========================================
int main()
{
    int x[m], y[m], z[m];
    printf("\nNhap mang x: \n");
    nhap_mang(x, m);
    printf("\nNhap mang y: \n");
    nhap_mang(y, m);
    printf("\nMang x: \n");
    hien_thi_mang(x, m);
    printf("\nMang y: \n");
    hien_thi_mang(y, m);
    tong_mang(x, y, z, m);
```

```
    printf("\nMang tong z: \n");
    hien_thi_mang(z, m);
    getch();
    return 0;
}
//=============================================
void nhap_mang(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        printf("\nNhap phan tu thu %d = ", i);
        scanf("%d", &a[i]);
    }
}
//=============================================
void hien_thi_mang(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%5d", a[i]);
}
//=============================================
void tong_mang(int a[], int b[], int c[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}
```

5.6.2. Mảng hai chiều tham gia làm đối số của hàm

Nếu mảng hai chiều tham gia làm đối số của hàm thì chúng phải được khai báo theo một trong hai cách sau:

Cách 1: `<Tên mảng>[][<Số phần tử chiều 2>]`

Cách 2: `(*<Tên mảng>)[<Số phần tử chiều 2>]`

Như vậy, khi có lời gọi hàm các tham số là mảng một chiều hoặc mảng hai chiều luôn được truyền địa chỉ.

Ví dụ 5.8. Viết lại chương trình của *Ví dụ 4.33*: *nhập hai ma trận các số thực* a_{mxn}, b_{mxn}.*Tính tổng của hai ma trận. Hiển thị lên màn hình hai ma trận* `a`, `b` *và tổng của chúng*, có khai báo, định nghĩa và sử dụng các hàm.

Ta sẽ chia bài toán thành các hàm thực hiện các chức năng độc lập bao gồm: hàm nhập ma trận (tương ứng hàm nhập mảng hai chiều), hàm hiển thị ma trận (hàm hiển thị mảng hai chiều), hàm cộng hai ma trận (hàm cộng hai mảng hai chiều) và hàm `main`.

```
#include <conio.h>
#include <stdio.h>
#define m 2
#define n 3
void nhap_mang(float a[][n], int p, int q);
void hien_thi_mang(float a[][n], int p, int q);
void tong_mang(float a[][n], float b[][n], float
c[][n], int p, int q);
//=========================================
int main()
{
    float a[m][n], b[m][n], c[m][n];
    printf("\nNhap mang a: \n");
    nhap_mang(a, m, n);
```

```
    printf("\nNhap mang b: \n");
    nhap_mang(b, m, n);
    printf("\nMang a: \n");
    hien_thi_mang(a, m, n);
    printf("\nMang b: \n");
    hien_thi_mang(b, m, n);
    tong_mang(a, b, c, m, n);
    printf("\nMang tong c: \n");
    hien_thi_mang(c, m, n);
    getch();
    return 0;
}
//=============================================
void nhap_mang(float a[][n], int p, int q)
{
    int i, j;
    for(i = 0; i < p; i++)
        for(j = 0; j < q; j++)
        {
            printf("Nhap phan tu [%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
}
//=============================================
void hien_thi_mang(float a[][n], int p, int q)
{
    int i, j;
    for(i = 0; i < p; i++)
    {
         for(j = 0; j < q; j++)
```

```
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
}
void tong_mang(float a[][n], float b[][n], float
c[][n], int p, int q)
{
    int i, j;
    for(i = 0; i < p; i++)
        for(j = 0; j < q; j++)
            c[i][j] = a[i][j] + b[i][j];
}
```

Vi dụ 5.9. Viết lại chương trình của *Ví dụ 4.34: nhập ma trận vuông các số thực* a_{nxn}. *Xây dựng vectơ* `b` *gồm các phần tử nằm trên đường chéo chính của ma trận* `a`. *Hiển thị lên màn hình ma trận* `a` *và vectơ* `b`, có khai báo, định nghĩa và sử dụng các hàm.

```
#include <conio.h>
#include <stdio.h>
#define n 3
void nhap_mang(float a[][n], int p);
void hien_thi_mang_1(float b[], int p);
void hien_thi_mang_2(float a[][n], int p);
void xay_dung_mang(float a[][n], float b[], int p);
//=========================================
int main()
{
    float a[n][n], b[n];
```

```
    printf("\nNhap mang a: \n");
    nhap_mang(a, n);
    printf("\nMang a: \n");
    hien_thi_mang_2(a, n);
    xay_dung_mang(a, b, n);
    printf("\nMang b: \n");
    hien_thi_mang_1(b, n);
    getch();
    return 0;
}
//=============================================
void nhap_mang(float a[][n], int p)
{
    for(int i = 0; i < p; i++)
       for(int j = 0; j < p; j++)
       {
          printf("\nNhap phan tu [%d][%d] = ", i, j);
           scanf("%f", &a[i][j]);
       }
}
//=============================================
void hien_thi_mang_1(float b[], int p)
{
    for(int i = 0; i < p; i++)
        printf("%5.2f", b[i]);
}
//=============================================
```

```
void hien_thi_mang_2(float a[][n], int p)
{
    for(int i = 0; i < p; i++)
    {
        for(int j = 0; j < p; j++)
            printf("%5.2f", a[i][j]);
        printf("\n");
    }
}
//=============================================
void xay_dung_mang(float a[][n], float b[], int p)
{
    for(int i = 0; i < p; i++)
        b[i] = a[i][i];
}
```

5.7. Đệ quy

Định nghĩa

Đệ quy là phương pháp định nghĩa một đối tượng thông qua chính nó. Hàm đệ quy là hàm được định nghĩa dựa trên chính nó nhưng với tham số thay đổi. Mỗi lần gọi đệ quy đến chính nó máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với tập các biến cục bộ đã được tạo ra trong các lần gọi trước.

Bài toán có những đặc điểm sau thì có thể dùng phương pháp đệ quy:

- Bài toán dễ dàng giải quyết trong một số trường hợp đặc biệt của tham số, gọi là trường hợp **suy biến** hay trường hợp **neo**, chính là trường hợp để kết thúc đệ quy;
- Trong trường hợp tổng quát, bài toán có thể quy về một bài toán cùng dạng nhưng với tham số thay đổi;

- Sau một số hữu hạn bước đệ quy dẫn đến trường hợp suy biến.

Ví dụ 5.10. Viết hàm tính `n!`

Nhận xét: Bài toán tính `n!` có những đặc điểm có thể sử dụng phương pháp đệ quy:

- Trường hợp suy biến: `n = 0` thì `n! = 1`
- Trường hợp tổng quát: `n! = n * (n - 1)!`

  `(n - 1)! = (n - 1) * (n - 2)!`

  …
- Đối số `n` giảm mỗi lần một đơn vị, vì vậy sau một số hữu hạn bước sẽ dẫn đến trường hợp suy biến (*tức trường hợp* `n = 0`).

Từ các đặc điểm trên ta có thể xây dựng hàm đệ quy tính `n!` như sau:

```
long giaithua(int n)
{
    if (n == 0)
       return 1;
    return n * giaithua(n - 1);
}
```

Ví dụ 5.11. Viết hàm tính ước chung lớn nhất hai số nguyên dương `a` và `b`.

Nhận xét: Bài toán tính ước chung lớn nhất của hai số có những đặc điểm có thể sử dụng phương pháp đệ quy:

- Trường hợp suy biến:

  `a = b` thì `ucln(a, b) = a`
- Trường hợp tổng quát:

  `a > b` thì `ucln(a, b) = ucln(a - b, b)`

  `a < b` thì `ucln(a, b) = ucln(a, b - a)`

- Ta thấy rằng cứ nếu `a > b` thì gán `a = a - b`, nếu `a < b` thì gán `b = b - a` và sau một số hữu hạn bước sẽ dẫn tới trường hợp suy biến (*trường hợp* `a = b`).

Ta có thể xây dựng hàm đệ quy tính ước chung lớn nhất của hai số nguyên dương như sau:

```
int ucln(int a, int b)
      {
            if (a == b)
                  return a;
            if (a > b)
                  return ucln(a - b, b);
            return  ucln(a, b - a);
      }
```

✍ **Lưu ý:** Phương pháp đệ quy không phải bao giờ cũng là phương pháp hữu hiệu vì hàm đệ quy sử dụng nhiều bộ nhớ. Mỗi lần gọi đệ quy máy phải dùng một số vùng nhớ mới để lưu giữ giá trị các tham số và biến cục bộ. Vì vậy, khi gặp một bài toán mà có thể có cách giải không dùng đệ qui (*cách giải lặp*) thì ta nên dùng cách này. Song, vẫn tồn tại những bài toán cho đến nay chỉ có thể giải bằng phương pháp đệ quy.

Ví dụ 5.12. So sánh số `byte` cấp phát cho hàm tính `n!` theo phương pháp đệ quy và hàm tính `n!` theo phương pháp lặp.

Hàm tính `n!` theo phương pháp đệ quy

```
long giai_thua_dq(int n)
{
      if (n == 0)
            return 1;
      return n * giaithua(n - 1);
}
```

Nếu gọi hàm **giai_thua_dq(4)** thì quá trình thực hiện như sau:

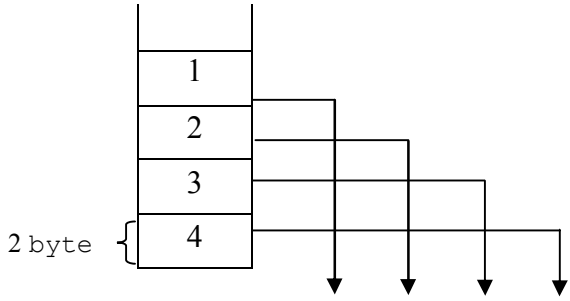
giai_thua_dq(4) = 4 * giai_thua_dq(3) /* Dùng 2 byte để lưu giữ giá trị 4 */

giai_thua_dq(3) = 3 * giai_thua_dq(2) /* Dùng 2 byte để lưu giữ giá trị 3 */

giai_thua_dq(2) = 2 * giai_thua_dq(1) /* Dùng 2 byte để lưu giữ giá trị 2 */

giai_thua_dq(1) = 1 * giai_thua_dq(0) /* Dùng 2 byte để lưu giữ giá trị 1 */

giai_thưa_dq(0) = 1 * 1 = 1

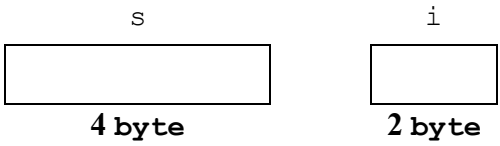


Kết quả **giai_thua_dq(4) = 1 * 1 * 2 * 3 * 4**

Như vậy, với lời gọi **giai_thua_dq(4)** hàm phải sử dụng **8 byte (= 2 * n)** để lưu giữ giá trị các tham số, với n càng lớn thì số byte sử dụng càng nhiều.

Hàm tính n! theo phương pháp lặp

```
long giai_thua_lap(int n)
{
      long s = 1;
      for(int i = 2; i <= n; i++)
            s *= i;
      return s;
}
```



Nếu gọi hàm **giai_thua_lap(4)** hàm chỉ cần sử dụng 4 byte cho biến s và 2 byte cho biến i, với n lớn hơn thì số byte cần dùng cho hàm vẫn không thay đổi.

5.8. Hàm main có đối số

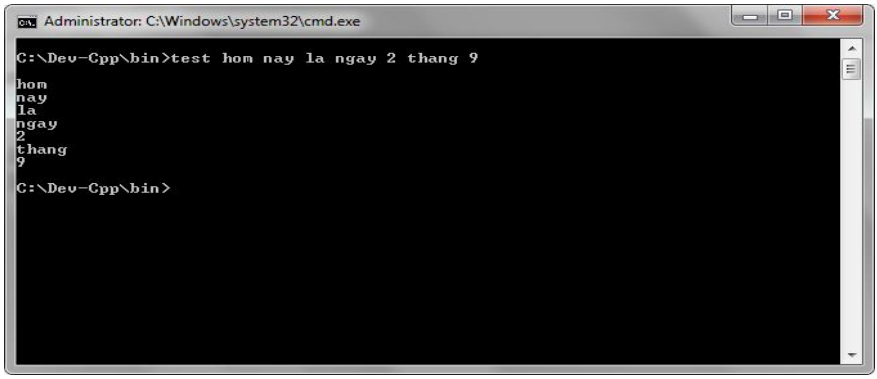
Ta đã rất quen với việc định nghĩa hàm main không có đối số dạng main(), trong phần này ta sẽ được làm quen với hàm main có đối số. Ngôn ngữ lập trình C cho phép đưa vào hàm main hai đối số dạng **main(int n, char *a[])**, giá trị của **n** và **a** nhận được khi gọi chương trình từ màn hình **DOS**.

Ví dụ 5.13.

```
#include <stdio.h>
int main(int n, char *a[])
{
    int i;
    for (i = 1; i < n; i++)
        printf("\n%s", a[i]);
    printf("\n");
    return 0;
}
```

Cách chạy chương trình: Ghi lại với tên chương trình là test.cpp, biên dịch chương trình. Ở dấu nhắc hệ thống ta nhập vào test hom nay la ngay 2 thang 9. Ta đã đưa vào 7 tham số, cùng với tên chương trình là 8 tham số, như vậy:

```
n = 8
a[0] = "C:\Dev-Cpp\bin\test.exe"
a[1] = "hom"
a[2] = "nay"
a[3] = "la"
a[4] = "ngay"
a[5] = "2"
a[6] = "thang"
a[7] = "9"
```



Ví dụ 5.14. Chương trình sau cho phép chúng ta nhập giá trị vào cho các đối số là các xâu số, chuyển đổi các xâu số đó ra số và thực hiện tính toán với các số đó.

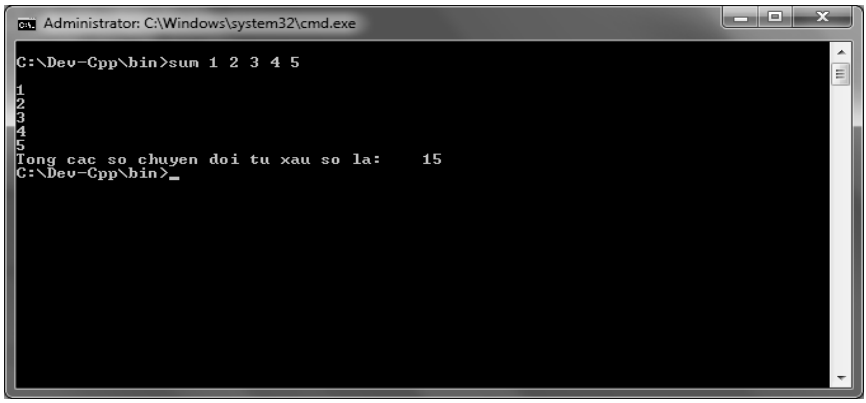
```
#include <stdio.h>
#include <stdlib.h>
int main(int n, char *a[])
{
    int i,  s = 0;
    for (i = 1; i < n; i++)
    {
        printf("\n%s", a[i]);
```

```
            s += atoi(a[i]);
    }
    printf("\nTong cac so chuyen doi tu xau so la:
            %5d", s);
    return 0;
}
```

Ghi lại với tên chương trình là `sum.cpp`, biên dịch chương trình. Ở dấu nhắc hệ thống ta nhập vào `sum 1 2 3 4 5`. Ta đã đưa vào 5 tham số, cùng với tên chương trình là 6 tham số, như vậy:

```
n = 6
a[0] = "C:\Dev-Cpp\bin\sum.exe"
a[1] = "1"
a[2] = "2"
a[3] = "3"
a[4] = "4"
a[5] = "5"
```

Dùng hàm `atoi` để chuyển các `a[i]` (`i = 1, 2`, ...,5) sang số và tính tổng các số nhận được bằng câu lệnh `s += atoi(a[i])`.



BÀI THỰC HÀNH CHƯƠNG 5

- **Mục tiêu**

Sau bài thực hành người học cần đạt được:

- Về kiến thức

+ Nắm được khái niệm hàm, ý nghĩa của việc sử dụng hàm;

+ Cách xây dựng và sử dụng hàm;

+ Phân biệt hàm có kiểu hàm là `void` và kiểu hàm khác `void`;

+ Nắm được cách truyền tham số trong hàm;

+ Nắm được cách viết hàm đệ quy.

- Về kỹ năng

+ Khi có một bài toán đặt ra, xác định được các mô đun công việc để có thể xây dựng hàm;

+ Xác định số đối số, kiểu các đối số, cách truyền tham số cho từng đối số.

- **Nội dung**

Bài 1. Chương trình sau đây xây dựng bao nhiêu hàm. Cho biết kiểu hàm, tên hàm, kiểu của các đối số, ý nghĩa của các hàm, biến tổng thể (*nếu có*), biến địa phương (*nếu có*) trong chương trình.

```
#include <conio.h>
#include <stdio.h>
int tong(int n)
{
    int i, s = 0;
    for(i = 1; i <= n; i++)
        s += i;
```

```
    return s;
}
int main()
{
    int n;
    printf("\nNhap n = ");
    scanf("%d", &n);
    printf("\n%5d", tong(n));
    getch();
    return 0;
}
```

Bài 2. Cho biết hàm sau thực hiện nhiệm vụ gì? Viết chương trình hoàn chỉnh sử dụng hàm đó.

```
int tong(int n)
{
      int i, s = 0;
      for(i = 1; i <= n; i++)
            s += 2 * i;
      return s;
}
```

Bài 3. Cho biết hàm sau thực hiện nhiệm vụ gì? Viết chương trình hoàn chỉnh sử dụng hàm đó.

```
void tong(int n)
{
      int i;
      float s = 0;
      for(i = 1; i <= n; i++)
            s += 1.0 / i;
}
```

Bài 4. Viết chương trình tính giai thừa của `n`, với `n` nguyên dương được nhập vào từ bàn phím.

Yêu cầu:

+ Xác định số lượng hàm cần xây dựng;
+ Xác định kiểu hàm, tên hàm, số đối số, kiểu của mỗi đối số cho từng hàm;
+ Xây dựng hàm;
+ Sử dụng hàm.

Bài 5. Viết chương trình giải phương trình bậc nhất `ax + b = 0` có xây dựng và sử dụng các hàm.

Yêu cầu:

+ Xác định số lượng hàm cần xây dựng;
+ Xác định kiểu hàm, tên hàm, số đối số, kiểu của mỗi đối số cho từng hàm;
+ Xây dựng hàm;
+ Sử dụng hàm.

Bài 6. Viết chương trình giải phương trình $ax^2 + bx + c = 0$ có xây dựng và sử dụng các hàm.

Yêu cầu:

+ Xác định số lượng hàm cần xây dựng;
+ Xác định kiểu hàm, tên hàm, số đối số, kiểu của mỗi đối số cho từng hàm;
+ Xây dựng hàm;
+ Sử dụng hàm.

Bài 7. Viết chương trình nhập một mảng một chiều gồm `n` số nguyên dương, tính và thông báo ra màn hình tổng các phần tử có giá trị lẻ trong mảng bằng cách xây dựng và sử dụng các hàm.

Yêu cầu:

+ Xác định số lượng hàm cần xây dựng;

+ Xác định kiểu hàm, tên hàm, số đối số, kiểu của mỗi đối số cho từng hàm;

+ Xây dựng hàm;

+ Sử dụng hàm.

Bài 8. Viết chương trình nhập và hiển thị mảng hai chiều $a_{m \times n}$, tìm và in ra màn hình phần tử lớn nhất, bé nhất của mảng.

Yêu cầu:

+ Xác định số lượng hàm cần xây dựng;

+ Xác định kiểu hàm, tên hàm, số đối số, kiểu của mỗi đối số cho từng hàm;

+ Xây dựng hàm;

+ Sử dụng hàm.

- **Tự học**

Bài 9. Viết lại các chương trình trong bài thực hành của các Chương 2, 3, 4 bằng cách xây dựng và sử dụng các hàm.

Bài 10. Viết hàm đổi một số nguyên không âm thành xâu nhị phân. Viết chương trình nhập vào từ bàn phím số nguyên không âm `n`, áp dụng hàm trên in ra màn hình xâu nhị phân tương ứng, theo dạng:

```
Số n        Xâu nhị phân
```

Chẳng hạn:

```
2           10
10          1010
```

Bài 11. Viết hàm kiểm tra tính nguyên tố của một số nguyên dương. Viết chương trình nhập vào từ bàn phím số nguyên dương `n`, áp dụng hàm trên hãy cho biết `n` có phải là số nguyên tố không?

Bài 12. Viết hàm kiểm tra tính hoàn thiện của một số nguyên dương. Viết chương trình nhập vào từ bàn phím số nguyên dương `n`, áp dụng hàm trên thông báo ra màn hình số `n` có phải là số hoàn thiện không?

Số nguyên dương `n` *là số hoàn thiện nếu tổng tất cả các ước của* `n` *gấp đôi* `n`.

Chẳng hạn: `6` *là số hoàn thiện vì* `6` *có các ước là* `1, 2, 3, 6` *mà* `1 + 2 + 3 + 6 = 2 * 6`.

Bài 13. Viết hàm kiểm tra xem ba số thực có thể là số đo ba cạnh của một tam giác không? Viết chương trình nhập vào từ bàn phím ba số thực `a, b, c`, áp dụng hàm trên kiểm tra nếu ba số `a, b, c` có thể là số đo ba cạnh của một tam giác thì tính và hiển thị lên màn hình chu vi, diện tích của tam giác đó, ngược lại thông báo ba số đó không thể là số đo ba cạnh của một tam giác.

Chẳng hạn: Ba số `3, 4, 5` *có thể là số đo ba cạnh của một tam giác, tam giác này có chu vi là* `12`*, diện tích là* `6`.

Bài 14. Viết hàm tính số `Fibonaxi` thứ `n`. Viết chương trình nhập vào từ bàn phím số nguyên dương `n`. Áp dụng hàm trên tìm và in ra màn hình số `Fibonaxi` tương ứng.

(*Dãy số* `Fibonaxi` *là dãy số được quy định:*

`Fib1 = 1; Fib2 = 1;`

`Fibn = (Fibn – 1) + (Fibn – 2)` với `n > 2`).

Bài 15. Viết hàm đổi giá trị hai biến thực cho nhau. Viết chương trình nhập vào từ bàn phím hai số thực `x, y`, áp dụng hàm trên để đổi giá trị hai biến `x, y` cho nhau. Hiển thị lên màn hình giá trị của `x` và `y` trước và sau khi đổi.

Chương 6

KIỂU DỮ LIỆU DO NGƯỜI LẬP TRÌNH ĐỊNH NGHĨA

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Khái niệm kiểu cấu trúc;*
- *Cách khai báo và sử dụng kiểu cấu trúc;*
- *Con trỏ cấu trúc;*
- *Mảng các cấu trúc;*
- *Cấu trúc tự trỏ và danh sách liên kết;*
- *Kiểu hợp, kiểu liệt kê.*

Các tài liệu tham khảo chính của chương bao gồm [1, 4, 5, 6, 7, 8].

6.1. Kiểu cấu trúc `struct`

6.1.1. Khái niệm

Kiểu cấu trúc là kiểu dữ liệu do người lập trình tự định nghĩa, bao gồm nhiều thành phần có thể có kiểu khác nhau, mỗi thành phần được gọi là một trường. Các trường của cấu trúc được khai báo như khai báo các biến. Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử

của mảng là cùng kiểu dữ liệu còn các phần tử của cấu trúc có thể có các kiểu dữ liệu khác nhau. Cấu trúc thường giúp tổ chức các dữ liệu phức tạp, đặc biệt là trong các chương trình lớn. Bởi vì cấu trúc cho phép một nhóm các biến có liên quan được xử lý như một đơn vị thay vì xử lý các biến riêng biệt. *Chẳng hạn,* thông tin về một sinh viên bao gồm họ và tên, ngày sinh, nơi sinh, giới tính, hộ khẩu,… có thể được quản lý dễ dàng, thuận tiện nhờ sử dụng dữ liệu kiểu cấu trúc.

6.1.2. Định nghĩa kiểu cấu trúc

Cách 1: **Sử dụng từ khoá** `struct`

```
struct <Tên kiểu cấu trúc>
  {
     <Kiểu 1> <Các trường 1> ;
     <Kiểu 2> <Các trường 2> ;
     …
     <Kiểu n> <Các trường n> ;
  };
```

Cách 2: **Sử dụng từ khóa** `typedef`

```
typedef struct
  {
     <Kiểu 1> <Các trường 1> ;
     <Kiểu 2> <Các trường 2> ;
     …
     <Kiểu n> <Các trường n> ;
  } <Tên kiểu cấu trúc>;
```

Trong đó:

- `Tên kiểu cấu trúc`: Được đặt theo quy tắc đặt tên của ngôn ngữ lập trình C;
- `Kiểu i` (`i = 1, 2, …, n`): Có thể là kiểu dữ liệu cơ bản, kiểu mảng, kiểu con trỏ hoặc cũng có thể là một kiểu cấu trúc đã được định nghĩa trước kiểu cấu trúc này;

- Các trường i (i = 1, 2, …, n): Có thể là một hay nhiều trường có kiểu dữ liệu là Kiểu i. Nếu có nhiều hơn một trường thì các trường được viết cách nhau bởi dấu phẩy. Tên của mỗi trường được đặt theo quy tắc đặt tên của ngôn ngữ lập trình C. Các trường được khai báo như khai báo biến.

Ví dụ 6.1. Để quản lý ngày, tháng, năm của một ngày trong năm ta có thể khai báo kiểu cấu trúc gồm ba thông tin: Ngay, Thang, Nam.

Cách 1:

```
struct NgayThang
  {
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
  };
```

Cách 2:

```
typedef struct
  {
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
  } NgayThang;
```

hoặc khai báo gộp các trường cùng kiểu:

Cách 1:

```
struct NgayThang
  {
    unsigned char Ngay,
                  Thang;
    unsigned int Nam;
  };
```

Cách 2:

```
typedef struct
  {
    unsigned char
              Ngay,Thang;
    unsigned int Nam;
  } NgayThang;
```

Ví dụ 6.2. Để quản lý sinh viên, mỗi sinh viên gồm các thông tin: Mã số sinh viên, họ tên, ngày, tháng, năm sinh, giới tính, địa chỉ thường trú, ta có thể khai báo một cấu trúc gồm các trường như sau:

Cách 1:	***Cách 2:***

`struct SinhVien`
`{`
`char MSSV[10];`
`char HoTen[40];`
`struct NgayThang NgaySinh;`
`int GioiTinh;`
`char DiaChi[40];`
`};`
	`typedef struct`
	`{`
	`char MSSV[10];`
	`char HoTen[40];`
	`NgayThang NgaySinh;`
	`int GioiTinh;`
	`char DiaChi[40];`
	`} SinhVien;`

Trong đó: Cấu trúc `NgayThang` đã được định nghĩa trong *Ví dụ 6.1.*

6.1.3. Khai báo biến cấu trúc

Cú pháp:

- **Đối với kiểu cấu trúc được định nghĩa theo cách 1**

`struct <Tên kiểu cấu trúc> <Danh sách biến cấu trúc>;`

- **Đối với kiểu cấu trúc được định nghĩa theo cách 2**

`<Tên kiểu cấu trúc> <Danh sách biến cấu trúc>;`

Ví dụ 6.3. Khai báo biến `NgaySinh` có kiểu cấu trúc `NgayThang`; biến `SV` có kiểu cấu trúc `SinhVien`.

Cách 1:	***Cách 2:***
`struct NgayThang NgaySinh;` `struct SinhVien SV;`	`NgayThang NgaySinh;` `SinhVien SV;`

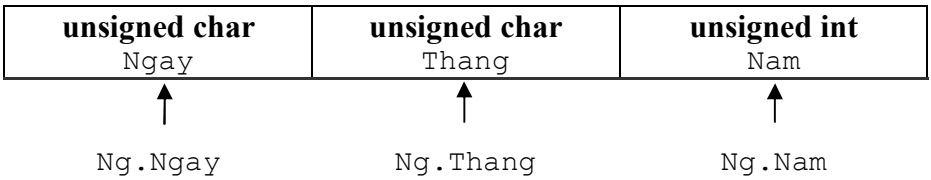
6.2. Các thao tác trên biến kiểu cấu trúc

6.2.1. Truy nhập đến từng trường của biến cấu trúc

Cú pháp: `<Biến cấu trúc>.<Tên trường>`

Chẳng hạn: `struct NgayThang Ng;`

Hình ảnh biến `Ng` trong bộ nhớ



✍ **Lưu ý:**

- Các biến cấu trúc có thể gán cho nhau. Thực chất đây là thao tác trên toàn bộ cấu trúc không phải trên một trường riêng lẻ nào;
- Với các biến kiểu cấu trúc **không thể thực hiện** được các thao tác sau đây:
  + Sử dụng các hàm xuất nhập trên biến cấu trúc (chỉ cho phép sử dụng hàm nhập xuất đối với các trường của cấu trúc);
  + Các phép toán quan hệ, các phép toán số học và logic.

Ví dụ 6.4. Viết chương trình quản lý đối tượng sinh viên, mỗi sinh viên gồm các thông tin sau: `họ tên`, `tuổi`, `điểm trung bình`, `xếp loại`, trong đó `xếp loại` được tính như sau:

xếp loại "`Giỏi`" nếu `điểm trung bình` ≥ 8.0,

xếp loại "`Khá`" nếu $7.0 \leq$ `điểm trung bình` < 8.0,

xếp loại "`Trung bình`" nếu $5.0 \leq$ `điểm trung bình` < 7.0,

còn lại xếp loại "`Yếu`". Nhập vào từ bàn phím thông tin cho một sinh viên. Hiển thị thông tin của sinh viên vừa nhập.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
typedef struct
{
   char hoten[30];
```

```
    int tuoi;
    float dtb;
    char xeploai[10];
}SinhVien;
//=======================================
int main()
{
    SinhVien s;
    printf("\nNhap thong tin cua sinh vien: ");
    printf("\nNhap ho va ten: ");
    fflush(stdin);
    gets(s.hoten);
    printf("\nNhap tuoi: ");
    scanf("%d", &s.tuoi);
    printf("\nNhap diem trung binh: ");
    scanf("%f", &s.dtb);
    if (s.dtb >= 8)
    strcpy(s.xeploai, "Gioi");
    else if (s.dtb >= 7)
    strcpy(s.xeploai, "Kha");
    else if (s.dtb >= 5)
    strcpy(s.xeploai, "Trung binh");
    else strcpy(s.xeploai, "Yeu");
    printf("\nThong tin cua sinh vien s: \n");
    printf("\n%-30s  %-5d  %-5.2f  %-10s",  s.hoten,
            s.tuoi, s.dtb, s.xeploai);
    getch();
    return 0;
}
```

Ví dụ 6.5. Cũng với bài toán quản lý đối tượng sinh viên trong *Ví dụ 6.4*, viết chương trình quản lý `n` sinh viên bao gồm các công việc: nhập thông tin cho `n` sinh viên, hiển thị lên màn hình thông tin của `n` sinh viên. Sắp xếp `n` sinh viên theo thứ tự giảm dần của điểm trung bình, hiển thị lên màn hình thông tin của `n` sinh viên sau khi sắp xếp.

Để quản lý `n` sinh viên ta sẽ sử dụng một mảng một chiều mà kiểu phần tử của mảng là kiểu cấu trúc. Với chương trình này ta sẽ định nghĩa các hàm, bao gồm: hàm nhập thông tin sinh viên, hàm hiển thị thông tin sinh viên, hàm sắp xếp sinh viên theo thứ tự giảm dần của `điểm trung bình` và hàm `main`.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
//=============================================
typedef struct
{
    char hoten[30];
    int tuoi;
    float dtb;
    char xeploai[10];
}SinhVien;
//=============================================
void nhap(SinhVien a[], int n)
{
    float tg;
    int i;
    for(i = 0; i < n; i++)
    {
        printf("Nhap thong tin sinh vien thu %d:\n", i);
```

```
        printf("Nhap ho va ten sinh vien : ");
        fflush(stdin);
        gets(a[i].hoten);
        printf("Nhap tuoi: ");
        scanf("%d", &a[i].tuoi);
        printf("Nhap diem trung binh: ");
        scanf("%f", &tg);
        a[i].dtb = tg;
        if (a[i].dtb >= 8.0)
            strcpy(a[i].xeploai, "Gioi");
        else if (a[i].dtb >= 7.0)
            strcpy(a[i].xeploai, "Kha");
        else if (a[i].dtb >= 5.0)
            strcpy(a[i].xeploai, "Trung binh");
        else strcpy(a[i].xeploai, "Yeu");
    }
}
//=============================================
void hien_thi(SinhVien a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
    printf("\n\n%-30s %-5d %-5.2f %-10s",
        a[i].hoten, a[i].tuoi, a[i].dtb, a[i].xeploai);
}
//=============================================
void sap_giam(SinhVien a[], int n)
{
    int i, j;
    SinhVien tg;
```

```
    for(i = 0; i < n - 1; i++)
        for(j = i + 1; j < n; j++)
            if (a[i].dtb < a[j].dtb)
            {
                tg = a[i];
                a[i] = a[j];
                a[j] = tg;
            }
}
//=============================================
int main()
{
    SinhVien x[10];
    int n;
    printf("Nhap so sinh vien: ");
    scanf("%d", &n);
    nhap(x, n);
    printf("\nDanh sach truoc khi sap:\n");
    hien_thi(x, n);
    sap_giam(x, n);
    printf("\nDanh sach sau khi sap:\n");
    hien_thi(x, n);
    getch();
    return 0;
}
```

6.2.2. Khởi tạo cấu trúc

Việc khởi tạo cấu trúc có thể được thực hiện trong khi khai báo biến cấu trúc. Các trường của cấu trúc cần khởi tạo được đặt giữa cặp ngoặc `{}`, chúng được phân cách nhau bởi dấu phẩy (`,`).

Chẳng hạn: Khởi tạo một biến cấu trúc `NgaySinh` có kiểu cấu trúc `NgayThang`:

```
struct NgayThang NgaySinh = {29, 8, 1986};
```

6.3. Con trỏ cấu trúc

6.3.1. Khai báo con trỏ cấu trúc

Việc khai báo biến con trỏ kiểu cấu trúc cũng tương tự như khai báo một biến con trỏ khác.

Cú pháp:

```
struct <Tên kiểu cấu trúc> * <Tên biến trỏ>;
```

Chẳng hạn: Khai báo một con trỏ cấu trúc kiểu `NgayThang` như sau:

```
struct NgayThang *p;
```

6.3.2. Sử dụng con trỏ cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa có địa chỉ cụ thể. Lúc này nó chỉ mới được cấp phát 2 `byte` để lưu giữ địa chỉ nhưng chưa trỏ tới một đối tượng cụ thể. Muốn thao tác trên con trỏ cấu trúc hợp lệ, cũng tương tự như các con trỏ khác, ta phải:

- Hoặc cấp phát một vùng nhớ cho nó (*sử dụng hàm* `malloc` *hay* `calloc`);
- Hoặc cho nó quản lý địa chỉ của một biến cấu trúc nào đó.

Chẳng hạn: Sau khi khởi tạo giá trị của cấu trúc:

```
struct NgayThang NgaySinh = {29, 8, 1986};
struct NgayThang *p;
p = &NgaySinh;
```

(Lúc này biến con trỏ `p` đã chứa địa chỉ của biến `NgaySinh`)

6.3.3. Truy nhập các thành phần của cấu trúc đang được quản lý bởi con trỏ

Để truy cập đến từng trường của một cấu trúc thông qua con trỏ, ta sử dụng phép toán **->** (*dấu trừ và dấu lớn hơn viết sát nhau*).

✍ **Lưu ý:** Ngoài ra, ta vẫn có thể sử dụng đến phép toán * để truy cập vùng dữ liệu đang được quản lý bởi con trỏ cấu trúc để lấy thông tin cần thiết.

Ví dụ 6.6. Sử dụng con trỏ cấu trúc.

```
#include<conio.h>
#include<stdio.h>
typedef struct
{
    unsigned char Ngay, Thang;
    unsigned int Nam;
}NgayThang;
int main()
{
    NgayThang Ng = {15, 1, 1990};
    NgayThang *p;
    p = &Ng;
    printf("Truy nhap binh thuong %d-%d-%d\n",
                Ng.Ngay, Ng.Thang, Ng.Nam);
    printf("Truy nhap thong qua con tro %d-%d-%d\n",
                p -> Ngay, p -> Thang, p -> Nam);
    printf("Truy nhap qua vung nho con tro %d-%d-%d\n",
                (*p).Ngay, (*p).Thang, (*p).Nam);
    getch();
    return 0;
}
```

6.4. Cấu trúc tự trỏ và danh sách liên kết

Cấu trúc tự trỏ là loại cấu trúc mà trong các thành phần của nó có ít nhất một thành phần là con trỏ có kiểu cấu trúc đang định nghĩa. Cấu trúc tự trỏ thường được dùng để tạo danh sách liên kết như: *danh sách liên kết đơn, danh sách liên kết đôi, danh sách liên kết vòng*... Cũng tương tự như mảng, ý nghĩa của danh sách liên kết là để lưu trữ các phần tử cùng kiểu. Tuy nhiên, việc lưu trữ bằng danh sách liên kết ưu điểm hơn lưu trữ bằng mảng ở chỗ:

- Tiết kiệm bộ nhớ vì cần bổ sung phần tử nào vào danh sách thì mới cấp phát bộ nhớ cho phần tử đó và kết nối vào danh sách;
- Việc bổ sung hay loại bỏ các phần tử khỏi danh sách được thực hiện đơn giản hơn mảng.

6.4.1. Khai báo cấu trúc tự trỏ

Cách 1:

```
struct <Tên kiểu cấu trúc tự trỏ>
{
   <Kiểu 1> <Các trường 1> ;
   <Kiểu 2> <Các trường 2> ;
   …
   <Kiểu n> <Các trường n> ;
   struct <Tên kiểu cấu trúc tự trỏ> <Các biến
   con trỏ>;
};
```

Cách 2:

```
typedef struct  <Tên kiểu cấu trúc>
{
   <Kiểu 1> <Các trường 1> ;
   <Kiểu 2> <Các trường 2> ;
```

```
    …
    <Kiểu n> <Các trường n> ;
    <Tên kiểu cấu trúc> <Các biến con trỏ>;
} <Tên kiểu cấu trúc tự trỏ>;
```

Cách 3:

```
typedef struct  <Tên kiểu cấu trúc> <Tên kiểu cấu
trúc tự trỏ>;
struct <Tên kiểu cấu trúc>
{
    <Kiểu 1> <Các trường 1> ;
    <Kiểu 2> <Các trường 2> ;
    …
    <Kiểu n> <Các trường n> ;
    <Tên kiểu cấu trúc> <Các biến con trỏ>;
};
```

Cách 4:

```
struct <Tên kiểu cấu trúc>
{
    <Kiểu 1> <Các trường 1> ;
    <Kiểu 2> <Các trường 2> ;
    …
    <Kiểu n> <Các trường n> ;
    <Tên kiểu cấu trúc> <Các biến con trỏ>;
};
typedef struct  <Tên kiểu cấu trúc> <Tên kiểu cấu
trúc tự trỏ>;
```

6.4.2. Danh sách liên kết

Danh sách liên kết là một cấu trúc dữ liệu cho phép quản lý các cấu trúc cùng kiểu bằng việc liên kết với nhau thông qua các con trỏ trong cấu trúc. Có nhiều dạng danh sách liên kết phụ thuộc vào cách kết nối.

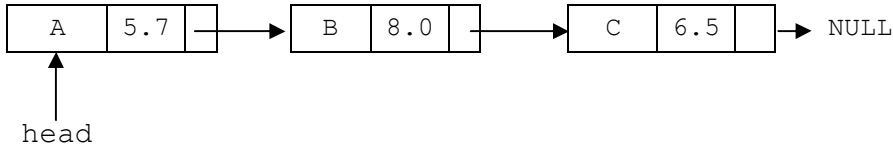
- **Danh sách liên kết đơn:** Mỗi cấu trúc chứa một con trỏ trỏ tới cấu trúc tiếp theo (*thường gọi là con trỏ* `next`) hoặc trước đó (*thường gọi là con trỏ* `previous`). Đối với danh sách con trỏ trỏ về trước, cấu trúc đầu tiên của danh sách sẽ chứa hằng `NULL`, cấu trúc cuối cùng được đánh dấu bởi một con trỏ (*thường gọi là con trỏ* `last`). Đối với danh sách con trỏ trỏ về cấu trúc tiếp theo, cấu trúc đầu sẽ được đánh dấu bởi một con trỏ (*thường gọi là con trỏ* `head`) và cấu trúc cuối cùng chứa hằng `NULL`.
- **Danh sách liên kết đôi:** Gồm hai con trỏ, một trỏ tới cấu trúc trước và một trỏ tới cấu trúc sau, hai đầu của danh sách được đánh dấu bởi các con trỏ `head`, `last`.
- **Danh sách liên kết vòng:** Gồm một con trỏ trỏ về sau (hoặc trước), hai đầu của danh sách được nối với nhau tạo thành một vòng. Chỉ cần một con trỏ `head` để đánh dấu đầu danh sách.

Do trong cấu trúc có chứa các con trỏ trỏ tới cấu trúc tiếp theo và/hoặc cấu trúc đứng trước nên từ một cấu trúc này chúng ta có thể truy cập đến một cấu trúc khác (trước và/hoặc sau nó). Kết hợp với các con trỏ đánh dấu hai đầu danh sách (`head`, `last`) chúng ta sẽ dễ dàng làm việc với bất kỳ phần tử nào của danh sách. Một số công việc thường thực hiện trên một danh sách liên kết như: *bổ sung phần tử vào danh sách; chèn thêm một phần tử mới; xoá một phần tử của danh sách; tìm kiếm; sắp xếp danh sách; in danh sác*h;…

Ví dụ 6.7. Giả sử ta có kiểu cấu trúc `SinhVien`

```
typedef struct SV SinhVien;
struct SV
{
    char  hoten[30];
    float dtb;
    SV *next;
};
```

Khi đó, một danh sách liên kết đơn gồm ba phần tử kiểu cấu trúc `SinhVien` sẽ được thể hiện trong bộ nhớ như sau:



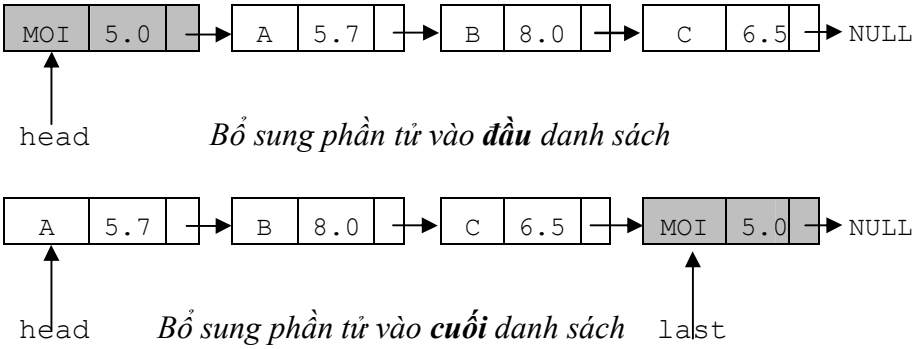
6.4.3. Các phép toán trên danh sách liên kết

a. Tạo phần tử mới

Để tạo phần tử mới thông thường chúng ta thực hiện theo các bước sau đây:

- Cấp phát một vùng nhớ đủ chứa một phần tử của danh sách bằng hàm `malloc` hoặc `calloc`;
- Nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ `next` của phần tử mới được gán bằng `NULL`;
- Gắn phần tử vừa tạo được vào danh sách theo một trong hai trường hợp:
  + Gắn vào đầu danh sách bằng cách điều chỉnh con trỏ `head` để trỏ vào phần tử mới. Nếu mọi phần tử mới của danh sách đều kết nối vào đầu ta được danh sách `LIFO` (*Last In First Out - Vào Sau Ra Trước*), vì như vậy sau này khi duyệt danh sách phần tử nào vào sau sẽ được duyệt trước, phần tử nào vào trước sẽ duyệt sau.
  + Gắn vào cuối danh sách bằng cách cho con trỏ `next` của phần tử cuối danh sách (*đang chứa* `NULL`) trỏ vào phần tử mới. Con trỏ `last` được điều chỉnh để trỏ vào phần tử mới. Nếu danh sách không có con trỏ `last` thì để tìm được phần tử cuối chương trình phải duyệt từ đầu, bắt đầu từ con trỏ `head` cho đến khi gặp phần tử chứa `NULL`, đó là phần tử cuối của danh sách. Nếu tất cả các phần tử mới đều được bổ

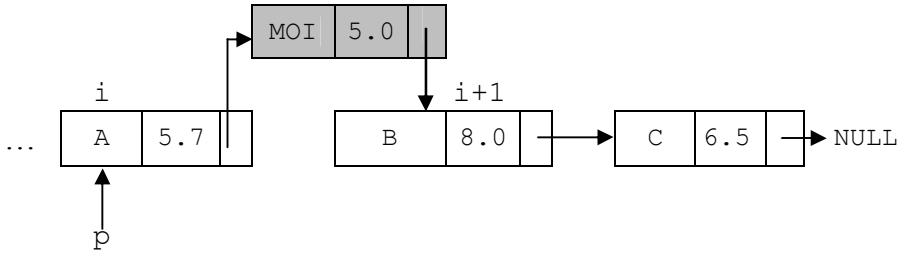
sung vào cuối thì ta được danh sách FIFO (*First In First Out – Vào Trước Ra Trước*), vì sau này khi duyệt danh sách phần tử nào vào trước sẽ được duyệt trước, phần tử nào vào sau sẽ duyệt sau.



b. Chèn phần tử mới vào giữa

Giả sử phần tử mới được chèn vào giữa phần tử thứ `i` và `i + 1`. Để chèn ta nối phần tử mới vào phần tử thứ `i + 1` và nối phần tử thứ `i` vào phần tử mới. Các bước thực hiện như sau:

- Cho con trỏ `p` trỏ tới phần tử thứ `i`;
- Cho con trỏ `next` của phần tử mới trỏ vào phần tử thứ `i + 1`;
- Cho con trỏ `next` của phần tử thứ `i` (*hiện được trỏ bởi* `p`) thay vì trỏ vào phần tử thứ `i + 1` sẽ trỏ vào phần tử mới.

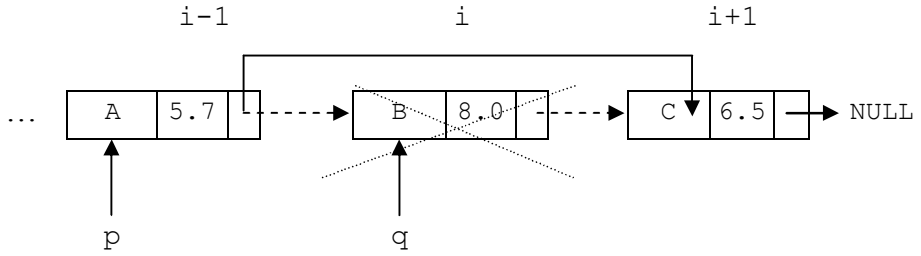


c. Xoá phần tử

Việc xoá một phần tử ra khỏi danh sách liên kết rất đơn giản bởi chỉ việc thay đổi các con trỏ. Cụ thể, giả sử cần xoá phần tử thứ `i` ta

chỉ cần cho con trỏ `next` của phần tử thứ `i - 1` trỏ vào phần tử thứ `i + 1`. Các bước thực hiện như sau:

- Cho con trỏ `p` trỏ tới phần tử thứ `i - 1`;
- Cho con trỏ `q` trỏ vào phần tử thứ `i`;
- Cho con trỏ `next` của phần tử thứ `i - 1` trỏ vào phần tử thứ `i + 1`;
- Giải phóng bộ nhớ được trỏ bởi con trỏ `q`.



✍ **Lưu ý:** Nếu xóa phần tử đầu tiên của danh sách ta cho con trỏ `q` trỏ vào đầu danh sách, cho con trỏ `head` trỏ tới phần tử tiếp theo (`head = head -> next`), giải phóng bộ nhớ được trỏ bởi con trỏ `q`.

d. Duyệt danh sách

Duyệt là thao tác đi qua từng phần tử của danh sách, tại mỗi phần tử chương trình thực hiện một công việc nào đó mà ta gọi là *thăm phần tử đó*. Một phép thăm có thể đơn giản là hiển thị nội dung của phần tử đó ra màn hình. Để duyệt danh sách, ta chỉ cần cho một con trỏ (*chẳng hạn là con trỏ* `p`) chạy từ phần tử đầu đến phần tử cuối của danh sách. Câu lệnh cho con trỏ `p` trỏ tới phần tử tiếp theo của danh sách là:

```
p = p -> next;
```

Ngoài các thao tác trên, đối với danh sách còn có nhiều thao tác khác như: *tìm kiếm phần tử trong danh sách, sắp xếp danh sách theo một tiêu chí nào đó,...*

Ví dụ 6.8. Cho cấu trúc `SinhVien` gồm hai trường `họ tên` và `điểm trung bình`. Tạo danh sách `LIFO` gồm `n` sinh viên. Hiển thị lên màn hình danh sách vừa tạo.

Trong chương trình này ta sử dụng một con trỏ `head` trỏ vào đầu danh sách là biến tổng thể, vì vậy ta định nghĩa các hàm tạo danh sách và hiển thị danh sách là các hàm không tham số.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define n 5
//=============================================
typedef struct SV SinhVien;
struct SV
{
    char hoten[30];
    float dtb;
    struct SV *next;
};
SinhVien *head = NULL;
//=============================================
void tao_ds()
{
    int i;
    float tg;
    SinhVien *p;
    for(i = 0; i < n; i++)
    {
        printf("\nNhap thong tin sinh vien thu
        %d:\n", i);
        p = (SinhVien*)malloc(sizeof(SinhVien));
```

```
        printf("Nhap ho va ten: ");
        fflush(stdin);
        gets(p -> hoten);
        printf("Nhap diem: ");
        scanf("%f", &tg);
        p -> dtb = tg;
        p -> next = head;
        head = p; //Gan p vao danh sach LIFO
    }
}
//=============================================
void hien_thi_ds()
{
    SinhVien *p;
    p = head;
    while (p != NULL)
    {
        printf("\n%-30s %5.2f", p -> hoten, p -> dtb);
        p = p -> next;
    }
}
//=============================================
int main()
{
    printf("\nTao danh sach:\n");
    tao_ds();
    printf("\nDanh sach vua tao:\n");
```

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

Ví dụ 6.9. Tạo danh sách LIFO gồm các số nguyên dương được nhập từ bàn phím, kết thúc nhập là một số âm hoặc số không. Sắp xếp danh sách theo thứ tự tăng dần và hiển thị lên màn hình danh sách sau khi sắp.

Với chương trình này ta có thể định nghĩa các hàm có tham số hình thức là con trỏ trỏ vào đầu danh sách. Danh sách này có thể tạo cho đến khi ta nhập vào một số âm hoặc số không.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
//=============================================
struct BanGhi
{
    int so;
    struct BanGhi *next;
};
//=============================================
struct BanGhi *tao_ds(struct BanGhi *h)
{
    int a;
    struct BanGhi *p;
    do
    {
        printf("Nhap so nguyen duong (So am hoac so
```

```
        0 de thoat): ");
        scanf("%d",&a);
        if (a > 0)
        {
            p = (struct BanGhi*)malloc(sizeof(struct BanGhi));
            p -> so = a;
            p -> next = h;
            h = p;
        }
    }
    while (a > 0);
    return h;
}
//=============================================
void hien_thi_ds(struct BanGhi *h)
{
    struct BanGhi *p;
    p = h;
    while (p != NULL)
    {
        printf("%5d", p -> so);
        p = p -> next;
    }
}
//=============================================
void sap_xep(struct BanGhi *h)
{
    struct BanGhi *p = h, *q;
```

```
    int tg;
    while (p -> next != NULL)
    {
       q = p -> next;
       while (q != NULL)
       {
            if (p -> so > q -> so)
            {
                tg = p -> so;
                p -> so = q -> so;
                q -> so = tg;
            }
            q = q -> next;
       }
       p = p -> next;
    }
}
//=============================================
int main()
{
    struct BanGhi *head = NULL;
    printf("\nTao danh sach:\n");
    head = tao_ds(head);
    printf("\nDanh sach vua tao:\n");
    hien_thi_ds(head);
    sap_xep(head);
    printf("\nSap xep danh sach\n");
```

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

Ví dụ 6.10. Cũng bài toán trong *Ví dụ 6.8* nhưng danh sách cần tạo là danh sách FIFO.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define n 5
//=============================================
typedef struct SV SinhVien;
struct SV
{
    char hoten[30];
    float dtb;
    struct SV *next;
};
SinhVien *head = NULL, *last = NULL;
//=============================================
void tao_ds()
{
    int i;
    float tg;
    SinhVien *p;
    for(i = 0; i < n; i++)
    {
```

```
        printf("\nNhap thong tin sinh vien thu %d:\n", i);
        p = (SinhVien*)malloc(sizeof(SinhVien));
        printf("Nhap ho va ten: ");
        fflush(stdin);
        gets(p -> hoten);
        printf("Nhap diem: ");
        scanf("%f", &tg);
        p -> dtb = tg;
        p -> next = NULL;
        if (head != NULL)
          last -> next = p; // Gan p vao danh sach FIFO
        else
          head = p;
        last = p;
    }
}
//=============================================
void hien_thi_ds()
{
    SinhVien *p;
    p = head;
    while (p != NULL)
    {
        printf("\n\n%-30s %5.2f", p -> hoten, p -> dtb);
        p = p -> next;
    }
}
//=============================================
```

```
int main()
{
    printf("\nTao danh sach:\n");
    tao_ds();
    printf("\nDanh sach vua tao:\n");
    hien_thi_ds();
    getch();
    return 0;
}
```

6.5. Kiểu hợp `union`

Việc định nghĩa một kiểu hợp cũng như kiểu cấu trúc nhưng chúng khác nhau ở chỗ: Các thành phần của cấu trúc có những vùng nhớ khác nhau, còn các thành phần của kiểu hợp được cấp phát cùng một vùng nhớ chung cho các biến khác nhau nhằm tiết kiệm bộ nhớ. Độ lớn của kiểu hợp bằng độ lớn của trường có kiểu dữ liệu lớn hơn.

Chẳng hạn:

```
union u_type
{
    int i;
    char ch;
};
u_type a;
```

Khi đó, vùng nhớ dành cho biến `a` chính là vùng nhớ dành cho biến `i` và biến `ch`. Tuy nhiên, đối với kiểu hợp cả `i` và `ch` cùng chung một vùng nhớ. Như vậy, tại một thời điểm ta chỉ sử dụng một biến hoặc `i` hoặc `ch`.

Khi khai báo biến kiểu hợp trình biên dịch sẽ cấp phát vùng nhớ có kích thước lớn nhất trong tất cả các phần tử của kiểu hợp. Trong ví dụ

trên, trình biên dịch sẽ cấp phát 2 byte cho kiểu u_type vì trong hai thành phần int và char thì int có kích thước lớn nhất (2 byte).

Việc định nghĩa một kiểu hợp, khai báo biến kiểu hợp, mảng kiểu hợp, con trỏ kiểu hợp cũng như cách truy nhập đến các trường của biến kiểu hợp hoàn toàn tương tự như đối với cấu trúc. Một cấu trúc có thể có trường kiểu hợp và ngược lại một trường của kiểu hợp có thể có kiểu cấu trúc.

Ví dụ 6.11. Cách sử dụng dữ liệu kiểu hợp.

```
#include <conio.h>
#include <stdio.h>
typedef union
{
    int a;
    int b;
}Hop;
int main()
{
    Hop h;
    h.a = 10;
    printf("%5d %5d", h.a, h.b);
    h.b = 20;
    printf("\n%5d %5d", h.a, h.b);
    getch();
    return 0;
}
```

Ví dụ 6.12. Tạo một kiểu hợp vợ chồng gồm hai thành phần tên vợ và tên chồng. Sau đó, tạo một cấu trúc người gồm: họ tên, tuổi, giới tính (*quy định* 0: Nữ, 1: Nam), tình trạng hôn nhân (*quy định* 0: Độc thân, 1: Có gia đình). Nếu là Nữ và đã có gia

đình thì nhập họ và tên chồng, nếu là Nam và đã có gia đình thì nhập họ và tên vợ. Hiển thị lên màn hình thông tin của mỗi người.

```
#include <conio.h>
#include <stdio.h>
union Vo_Chong
{
    char tenvo[30];
    char tenchong[30];
};
typedef struct Nguoi
{
    char hoten[30];
    int tuoi;
    unsigned char gioitinh;
    unsigned char honnhan;
    union Vo_Chong ten;
};
int main()
{
    Nguoi s;
    printf("Nhap thong tin nguoi s: ");
    printf("Nhap ho va ten: ");
    fflush(stdin);
    gets(s.hoten);
    printf("Nhap tuoi: ");
    scanf("%d", &s.tuoi);
    printf("Nhap gioi tinh (0: Nu, 1: Nam): ");
    scanf("%d", &s.gioitinh);
```

```
    printf("Tinh trang hon nhan (0: Doc than, 1: Co
    gia dinh): ");
    scanf("%d", &s.honnhan);
    if ((s.gioitinh == 1) && (s.honnhan == 1))
    {
       printf("Nhap ten vo: ");
       fflush(stdin);
       gets(s.ten.tenvo);
    }
    if ((s.gioitinh == 0) && (s.honnhan == 1))
    {
       printf("Nhap ten chong: ");
       fflush(stdin);
       gets(s.ten.tenchong);
    }
    printf("\nThong tin nguoi s: \n");
    if (s.honnhan == 0)
       printf("%s  %5d  %5d  %5d", s.hoten, s.tuoi,
       s.gioitinh, s.honnhan);
    if ((s.gioitinh == 1) && (s.honnhan == 1))
       printf("%s %5d %5d %5d %s", s.hoten, s.tuoi,
       s.gioitinh, s.honnhan, s.ten.tenvo);
    if ((s.gioitinh == 0) && (s.honnhan == 1))
       printf("%s %5d %5d %5d %s", s.hoten, s.tuoi,
       s.gioitinh, s.honnhan, s.ten.tenchong);
    getch();
    return 0;
}
```

6.6. Kiểu liệt kê `enum`

Là kiểu dữ liệu được định nghĩa bằng cách liệt kê tên của các giá trị. Về bản chất dãy tên được liệt kê có giá trị là các số nguyên với giá trị ngầm định bắt đầu từ 0, giá trị tiếp theo tăng lên 1. Tuy nhiên, ta có thể gán giá trị cho các tên bằng các giá trị nguyên bất kỳ. Đối với kiểu liệt kê chúng ta không thể đọc và xuất trực tiếp các tên của nó, do đó khi nhập ta cũng nhập số nguyên và khi xuất cũng xuất số nguyên tương ứng với tên được liệt kê.

Khai báo kiểu liệt kê

```
enum <Kiểu> {<Tên liệt kê> [= <Giá trị>], ...};
```

Khai báo biến có kiểu liệt kê

```
enum  <Kiểu> <Danh sách biến kiểu hợp>;
```

hoặc

```
<Kiểu> <Danh sách biến kiểu hợp>;
```

Có thể kết hợp khai báo kiểu liệt kê với khai báo biến

```
enum <Kiểu> {<Tên liệt kê> [= <Giá trị>], ...}
<Danh sách biến kiểu hợp>;
```

Ví dụ 6.13.

```
enum Mycolor1  {Blue, Red, Purple, Yellow, Pink, White};
```

Trong đó: `enum` là từ khóa, **`Mycolor1`** là tên kiểu liệt kê, `Blue, Red, Purple, Yellow, Pink, White` là các tên liệt kê. `Blue` có giá trị 0, `Red` có giá trị 1, `Purple` có giá trị 2, `Yellow` có giá trị 3, `Pink` có giá trị 4, `White` có giá trị 5.

Khai báo biến có kiểu **`Mycolor1`**

```
enum Mycolor1 Flag_color;
```

hoặc

```
Mycolor1 Flag_color;
printf("%d %d", Blue, Yellow); // Kết quả là 0 3
```

```
enum Mycolor2 {Blue = 2, Red, Purple,
Yellow, Pink, White};
```

Trong kiểu **Mycolor2:** Blue có giá trị 2, Red có giá trị 3, Purple có giá trị 4, Yellow có giá trị 5, Pink có giá trị 6, White có giá trị 7.

```
enum Mycolor3 {Blue = 2, Red, Purple,
Yellow = 10, Pink, White};
```

Trong kiểu **Mycolor3**: Blue có giá trị 2, Red có giá trị 3, Purple có giá trị 4, Yellow có giá trị 10, Pink có giá trị 11, White có giá trị 12.

```
enum Mycolor4 {Blue = 2, Red = 3, Purple = 5,
Yellow = 7, Pink = 8, White = 9};
```

Trong kiểu **Mycolor4**: Blue có giá trị 2, Red có giá trị 3, Purple có giá trị 5, Yellow có giá trị 7, Pink có giá trị 8, White có giá trị 9.

Ví dụ 6.14. Cách sử dụng dữ liệu kiểu liệt kê.

```
#include <conio.h>
#include <stdio.h>
enum Days {Mon = 2, Tue, Wed, Thu, Fri, Sat, Sun};
int main()
{
    Days d;
    printf("\nNhap so nguyen thuoc doan [2..8]: ");
    scanf("%d", &d);
    switch(d)
    {
      case Mon: printf("\nDay la: Thu Hai"); break;
      case Tue: printf("\nDay la: Thu Ba"); break;
```

```
        case Wed: printf("\nDay la: Thu Tu"); break;
        case Thu: printf("\nDay la: Thu Nam"); break;
        case Fri: printf("\nDay la: Thu Sau"); break;
        case Sat: printf("\nDay la: Thu Bay"); break;
        case Sun: printf("\nDay la: Chu Nhat"); break;
    }
    getch();
    return 0;
}
```

6.7. Định nghĩa kiểu bằng từ khóa `typedef`

Từ khoá `typedef` dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ được sử dụng để khai báo dữ liệu sau này.

Cú pháp: `typedef <Kiểu dữ liệu> <Tên kiểu>;`

Khi đó, ta có thể sử dụng `Tên kiểu` để khai báo các biến, mảng, cấu trúc,...

Ví dụ 6.15.

```
typedef int Nguyen;
typedef float MT10[10];
```

Như vậy, ta đã định nghĩa một kiểu dữ liệu có tên là **`Nguyen`**, một kiểu dữ liệu có tên là **`MT10`**. Bây giờ, ta có thể khai báo các biến có kiểu **`Nguyen`** và kiểu **`MT10`** như sau:

```
Nguyen x, y, a[10], b[20][30];
MT10 u, v;
```

BÀI THỰC HÀNH CHƯƠNG 6

- **Mục tiêu**

Qua bài thực hành người học cần đạt được:

- **Về kiến thức**
  - + Nắm được cách khai báo, sử dụng dữ liệu kiểu cấu trúc;
  - + Nắm được cách sử dụng con trỏ kiểu cấu trúc;
  - + Nắm được cách khai báo cấu trúc tự trỏ và sử dụng để tạo danh sách liên kết;
  - + Nắm được cách khai báo và sử dụng dữ liệu kiểu hợp, kiểu liệt kê.
- **Về kỹ năng**
  - + Vận dụng được các kiến thức trên để giải quyết các bài toán có dữ liệu phù hợp với kiểu cấu trúc, kiểu hợp, kiểu liệt kê;
  - + Xác định được trường hợp nào nên dùng dữ liệu kiểu cấu trúc, trường hợp nào nên dùng dữ liệu kiểu hợp.

- **Nội dung**

Bài 1. Viết chương trình quản lý điểm thi đại học của `n` thí sinh, thông tin mỗi thí sinh gồm: `số báo danh`, `họ tên`, `tuổi`, `điểm môn 1`, `điểm môn 2`, `điểm môn 3`, `tổng điểm`. Trong đó, `tổng điểm = điểm môn 1 + điểm môn 2 + điểm môn 3`. Hiển thị lên màn hình thông tin của những thí sinh có `tổng điểm` ≥ 15.

Yêu cầu:

+ Định nghĩa kiểu cấu trúc cần sử dụng;

+ Chỉ ra số trường, tên các trường, kiểu các trường của kiểu cấu trúc đó;
+ Xác định các hàm cần xây dựng;
+ Viết chương trình.

Bài 2. Viết lại chương trình của *Bài 1*, trong đó thông tin tuổi của thí sinh được thay bằng ngày sinh và bổ sung thêm thông tin giới tính.

Yêu cầu:

+ Định nghĩa các kiểu cấu trúc, kiểu hợp cần sử dụng;
+ Xác định các hàm cần xây dựng;
+ Viết chương trình.

Bài 3. Viết chương trình nhập vào từ bàn phím một mảng `n` phân số, mỗi phân số gồm `tử số` và `mẫu số`. Hiển thị lên màn hình các phân số dạng `tử số / mẫu số`.

Yêu cầu:

+ Xác định kiểu cấu trúc cần sử dụng;
+ Xác định các hàm cần xây dựng;
+ Viết chương trình.

Bài 4. Viết chương trình tạo một danh sách liên kết gồm các số phức nhập vào từ bàn phím, mỗi số phức gồm `phần thực` và `phần ảo`. Hiển thị lên màn hình các số phức dạng:

```
phần thực + i * phần ảo
```

Yêu cầu:

+ Xác định kiểu cấu trúc, kiểu danh sách liên kết cần sử dụng;
+ Xác định các hàm cần xây dựng;
+ Viết chương trình.

Bài 5. Viết chương trình nhập vào một mảng gồm `n` phần tử thời gian, mỗi phần tử gồm `giờ`, `phút`, `giây`. Hiển thị lên màn hình các phần tử thời gian theo dạng `giờ : phút : giây`. Sắp xếp mảng theo thứ tự tăng dần. Hiển thị giá trị các phần tử của mảng sau khi sắp xếp.

Yêu cầu:

+ Xác định kiểu cấu trúc cần sử dụng;

+ Xác định các hàm cần xây dựng;

+ Viết chương trình.

- **Câu hỏi củng cố**

  - Số `byte` của một kiểu cấu trúc được tính như thế nào?

  - Có thể thực hiện các phép toán `+, -, *, /` cho hai biến có kiểu cấu trúc không?

  - Nếu kiểu cấu trúc `B` có một trường có kiểu cấu trúc `A`, thì kiểu cấu trúc `A` có cần phải định nghĩa trước kiểu cấu trúc `B` không?

  - Tham số của hàm có thể có kiểu cấu trúc không?

  - Kiểu hàm có thể là kiểu cấu trúc không?

- **Tự học**

Bài 6. Viết lại chương trình *Bài 3*, thêm yêu cầu tính và hiển thị lên màn hình tổng của `n` phân số.

Bài 7. Viết lại chương trình *Bài 4*, thêm yêu cầu tính và hiển thị lên màn hình tổng của `n` số phức.

Bài 8. Viết chương trình tạo mảng gồm `n` cán bộ, thông tin mỗi cán bộ gồm: `họ tên`, `hệ số lương`, `lương`, trong đó: `lương = hệ số lương * 1050000`. Sắp xếp lại mảng theo thứ tự tăng dần của lương. Nhập thông tin cho một người `x`, chèn người `x` vào mảng sao cho vẫn đảm bảo tính sắp tăng của mảng theo lương.

Bài 9. Viết chương trình nhập thông tin của `n` học sinh, mỗi học sinh gồm: `họ tên`, `ngày sinh`, `giới tính`, `lớp`. Sắp xếp `n` học sinh theo thứ tự từ điển của `tên`.

Gợi ý: Cần xây dựng hàm để tách tên từ một xâu họ và tên, dùng hàm này để sắp xếp mảng học sinh theo thứ tự từ điển của `tên`.

Bài 10. Viết chương trình tạo danh sách liên kết gồm các phần tử ngày tháng, mỗi phần tử gồm `ngày`, `tháng`, `năm`. Hiển thị lên màn hình các phần tử ngày tháng theo dạng `ngày – tháng - năm`. Cho biết phần tử gần ngày hiện tại nhất.

Chương 7

DỮ LIỆU KIỂU TỆP TIN

Mục tiêu: *Sau khi học xong chương này người học cần nắm được:*

- *Khái niệm tệp tin;*
- *Các bước thao tác với tệp tin;*
- *Các hàm truy xuất tệp tin văn bản;*
- *Các hàm truy xuất tệp tin nhị phân.*

Các tài liệu tham khảo chính của chương bao gồm [1, 4, 5, 7, 8].

7.1. Một số khái niệm về tệp tin

Trong các chương trước, ta đã làm quen với các kiểu dữ liệu cơ bản, kiểu mảng, kiểu cấu trúc, các kiểu dữ liệu này đều được tổ chức trong bộ nhớ trong (RAM) của máy tính nên khi kết thúc việc thực hiện chương trình thì dữ liệu cũng bị mất, khi cần thực hiện lại bài toán chúng ta bắt buộc phải nhập lại toàn bộ dữ liệu từ bàn phím. Điều này dẫn tới việc khi giải quyết các bài toán có dữ liệu vào lớn sẽ mất rất nhiều thời gian để nhập dữ liệu. Để giải quyết vấn đề này, người ta đưa ra kiểu tệp tin cho phép lưu trữ dữ liệu ở bộ nhớ ngoài. Khi kết thúc chương trình thì dữ liệu vẫn còn, khi cần dữ liệu đó ta chỉ cần truy nhập vào tệp để đọc, do đó chúng ta có thể sử dụng nhiều lần.

Một đặc điểm nổi bật khác của kiểu tệp tin là kích thước lớn với số lượng các phần tử không hạn chế (*chỉ bị hạn chế bởi dung lượng của bộ nhớ ngoài*) rất phù hợp với các bài toán có số lượng dữ liệu lớn.

Ngôn ngữ lập trình C định nghĩa ba loại tệp tin:

- **Tệp tin văn bản:** Là loại tệp tin dùng để ghi các ký tự lên đĩa, các ký tự này được lưu trữ dưới dạng mã ASCII. Điểm đặc biệt là dữ liệu của tệp tin được lưu trữ thành các dòng, mỗi dòng được kết thúc bằng ký tự xuống dòng (*new line*), ký hiệu '\n'; ký tự này là sự kết hợp của hai ký tự `CR` (*Carriage Return - Về đầu dòng, mã ASCII là* 13) và `LF` (*Line Feed - Xuống dòng, mã ASCII là* 10). Mỗi tệp tin được kết thúc bởi ký tự `EOF` (*End Of File*) có mã ASCII là 26 (*xác định bởi tổ hợp phím Ctrl + Z*). Tệp tin văn bản chỉ có thể truy xuất theo kiểu tuần tự.
- **Tệp tin định kiểu:** Là loại tệp tin bao gồm nhiều phần tử có cùng kiểu: `char`, `int`, `long` hay kiểu cấu trúc, … Dữ liệu đối với tệp tin định kiểu được lưu trữ trên đĩa dưới dạng một chuỗi các `byte` liên tục.
- **Tệp tin không định kiểu:** Là loại tệp tin mà dữ liệu của chúng gồm các cấu trúc dữ liệu mà người ta không quan tâm đến nội dung hoặc kiểu của nó, chỉ lưu ý đến các yếu tố vật lý của tệp tin như độ lớn và các yếu tố tác động lên tệp tin.

▪ **Biến tệp tin**

Là một biến có kiểu dữ liệu tệp tin dùng để đại diện cho một tệp tin. Dữ liệu chứa trong một tệp tin được truy xuất qua các thao tác với tham số là biến tệp tin đại diện cho tệp tin đó.

▪ **Con trỏ cửa sổ tệp tin**

Khi một tệp tin được mở ra để làm việc, tại mỗi thời điểm sẽ có một vị trí của tệp tin mà tại đó việc đọc/ghi thông tin sẽ xảy ra. Ta hình dung, có một con trỏ đang trỏ tới vị trí đó và nó được gọi là con trỏ cửa sổ tệp tin.

Sau một thao tác đọc/ghi xong dữ liệu, con trỏ cửa sổ tệp tin sẽ chuyển dịch thêm một phần tử về phía cuối tệp tin. Sau phần tử dữ liệu cuối cùng của tệp tin là dấu kết thúc tệp tin `EOF` (*End Of File*).

7.2. Các thao tác trên tệp tin

Trong ngôn ngữ lập trình C, để có thể thực hiện được các thao tác trên tệp tin ta tiến hành thông qua các hàm thư viện. Các hàm này đã được định nghĩa trong thư viện `stdio.h`. Muốn thao tác trên tệp tin, ta phải lần lượt thực hiện các bước sau:

Bước 1: Khai báo biến tệp tin.

Bước 2: Mở tệp tin bằng hàm `fopen`.

Bước 3: Thực hiện các thao tác xử lý dữ liệu của tệp tin bằng các hàm đọc/ghi dữ liệu.

Bước 4: Đóng tệp tin bằng hàm `fclose`.

7.2.1. Khai báo biến con trỏ tệp tin

Cú pháp: `FILE <Danh sách các biến con trỏ tệp tin>;`

Các biến trong danh sách phải là các biến con trỏ tệp tin và được phân cách bởi dấu phẩy (`,`).

Chẳng hạn:

```
FILE *f1,*f2; // Khai báo hai biến con trỏ tệp tin f1 và f2
```

7.2.2. Mở tệp tin

Cú pháp:

```
FILE *fopen(char *<Tên tệp>, const char <"Kiểu mở">);
```

Trong đó:

- `Tên tệp`: Là tên tệp tin cần mở.
- `Kiểu mở`: Chuỗi xác định cách thức mà tệp tin sẽ mở (*Bảng 7.1*).

Kiểu mở	Ý nghĩa
r hoặc rt	Mở tệp tin văn bản để đọc.
w hoặc wt	Tạo ra tệp tin văn bản mới để ghi.
a hoặc at	Nối vào hay tạo mới tệp tin văn bản.
rb	Mở tệp tin nhị phân để đọc.
wb	Tạo ra tệp tin nhị phân mới để ghi.
ab	Nối vào hay tạo mới tệp tin nhị phân.
r+	Mở một tệp tin văn bản để đọc/ghi.
w+	Tạo ra tệp tin văn bản để đọc/ghi.
a+	Nối vào hay tạo mới tệp tin văn bản để đọc/ghi.
r+b	Mở ra tệp tin nhị phân để đọc/ghi.
w+b	Tạo ra tệp tin nhị phân để đọc/ghi.
a+b	Nối vào hay tạo mới tệp tin nhị phân để đọc ghi.

Bảng 7.1. Các kiểu mở tệp

Hàm `fopen` trả về một con trỏ tệp tin, vì vậy chúng ta gán kết quả của hàm cho một biến con trỏ tệp tin. Nếu khi mở tệp có lỗi, kết quả của hàm là `NULL`.

Ví dụ 7.1. Mở một tệp tin với tên "`TEXT.TXT`" để đọc và ghi dữ liệu.

Đọc tệp

```
FILE *f;
f = fopen("TEXT.TXT", "r");
// Các câu lệnh để đọc tệp tin
```

Ghi tệp

```
FILE *f;
f = fopen("TEXT.TXT", "w");
//Các câu lệnh ghi vào tệp tin
```

✍ **Lưu ý:**

- Khi mở tệp tin để ghi, nếu tệp tin đã tồn tại thì tệp tin đó sẽ bị xóa và một tệp tin mới được tạo ra. Tuy nhiên, nếu muốn ghi nối dữ liệu ta phải sử dụng `Kiểu mở` là "`a`";

- Khi mở với chế độ đọc, tệp tin phải tồn tại, nếu không sẽ xuất hiện lỗi;
- Một tệp tin sau khi đã được mở mới có thể thực hiện các thao tác đọc/ghi lên tệp tin đó. Do vậy, khi mở tệp tin ta phải kiểm tra xem liệu thao tác mở tệp tin có thành công hay không? Nếu thành công mới tiến hành các thao tác tiếp theo, ngược lại phải thông báo cho người sử dụng biết thông qua đoạn chương trình sau:

```
FILE *f;
f = fopen(char *<Tên tệp>, const char <"Kiểu mở">);
if (f == NULL)
{
    printf("\nKhong mo duoc tep!");
    getch();
    exit(-1); /* Hàm dừng chương trình và giải
                 phóng toàn bộ vùng nhớ đã cấp
                 cho các biến, hàm  thuộc thư
                 viện stdlib.h */
}
```

7.2.3. Đóng tệp tin

Cú pháp: `int fclose(FILE *f);`

Trong đó:

- `f`: Là biến con trỏ quản lý tệp tin được mở bởi hàm `fopen`;
- Giá trị trả về của hàm là `0` báo rằng việc đóng tệp tin thành công. Hàm trả về `EOF` nếu có xuất hiện lỗi.

Hàm `fclose` được dùng để đóng tệp tin mà đã được mở bởi hàm `fopen`. Hàm này sẽ ghi dữ liệu còn lại trong vùng đệm vào tệp tin và đóng tệp tin. Ngoài ra, ta còn có thể sử dụng hàm `fcloseall` để đóng tất cả các tệp tin đang được mở.

***Cú pháp*:** `int fcloseall(void);`

Trong đó:

- Kết quả trả về của hàm là tổng số các tệp tin được đóng;
- Nếu không thành công, kết quả trả về là `EOF`.

Ví dụ 7.2.

Đọc tệp

```
FILE *f;
f = fopen("TEXT.TXT", "r");
/* Các câu lệnh đọc tệp tin*/
fclose(f);
```

Ghi tệp

```
FILE *f;
f = fopen("TEXT.TXT", "w");
/* Các câu lệnh ghi vào tệp tin*/
fclose(f);
```

7.2.4. Kiểm tra kết thúc tệp tin

Cú pháp: `int feof(FILE *f);`

Hàm `feof` kiểm tra xem con trỏ tệp đã ở cuối tệp tin hay chưa? Giá trị trả về của hàm:

- Bằng `0` nếu chưa kết thúc tệp;
- Khác `0` nếu đã kết thúc tệp;

Trong đó: `f` là biến con trỏ quản lý tệp tin cần kiểm tra.

Ví dụ 7.3. Thao tác với tệp tin `"TEXT.TXT"` trong khi chưa kết thúc tệp, dùng hàm `feof` để kiểm tra con trỏ tệp đã ở cuối tệp hay chưa.

```
FILE *f;
f = fopen("TEXT.TXT", "r");
while !feof(f)
{
    /* Các câu lệnh thao tác với tệp tin */
}
fclose(f);
```

7.2.5. Di chuyển con trỏ về đầu tệp tin

Khi đang thao tác một tệp tin đang mở, con trỏ tệp tin luôn di chuyển về phía cuối tệp tin. Muốn cho con trỏ quay về đầu tệp tin như khi mở nó, ta sử dụng hàm `rewind`.

Cú pháp: `void rewind(FILE *f);`

Trong đó: `f` là biến con trỏ quản lý tệp tin cần thao tác.

Ví dụ 7.4.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     FILE *f;
     char first;
     f = fopen("TEXT.TXT","w+");
     fprintf(f, "abcdefghijklmnopqrstuvwxyz");
     rewind(f);
     fscanf(f, "%c", &first);
     printf("Ky tu dau tien la: %c\n", first);
     fclose(f);
     getch();
     return 0;
}
```

7.3. Các thao tác với tệp tin văn bản

7.3.1. Ghi dữ liệu lên tệp tin văn bản

- **Hàm `putc`**

Hàm `putc` được dùng để ghi một ký tự lên tệp tin văn bản đang được mở.

Cú pháp: `int putc(int ch, FILE *f);`

***Trong đó*:** Tham số `ch` chứa mã ASCII của một ký tự nào đó. Mã này được ghi lên tệp tin quản lý bởi con trỏ `f`. Hàm trả về `EOF` nếu gặp lỗi.

Ví dụ 7.5. Ghi từng ký tự trong xâu "`Hello world`" vào tệp "`TEXT.TXT`".

```
#include <stdio.h>
#include <conio.h>
int main()
{
      char msg[] = "Hello world\n";
      FILE *f;
      int i = 0;
      f = fopen("TEXT.TXT","w");
      while (msg[i])
      putc(msg[i++], f);
      fclose(f);
      return 0;
}
```

- **Hàm `fputs`**

Hàm `fputs` dùng để ghi một xâu ký tự chứa trong vùng đệm lên tệp tin văn bản.

Cú pháp: `int puts(const char *s, FILE *f);`

Trong đó: `s` là xâu ký tự cần ghi vào tệp tin quản lý bởi con trỏ `f`. Hàm trả về giá trị `0` nếu `s` chứa xâu rỗng và trả về `EOF` nếu gặp lỗi.

Ví dụ 7.6. Ghi xâu ký tự vào tệp tin "TEXT.TXT".

```
#include <stdio.h>
#include <conio.h>
int main()
{
      FILE *f;
      f = fopen("TEXT.TXT", "w");
      fputs("Hello ", f);
      fputs("World\n", f);
      fclose(f);
      return 0;
}
```

- **Hàm `fprintf`**

Hàm `fprintf` dùng để ghi dữ liệu có định dạng lên tệp tin văn bản.

Cú pháp:

`fprintf(FILE *f, "<Chuỗi định dạng>"[, <Mục 1>, <Mục 2>, ...]);`

Trong đó:

- `Chuỗi định dạng`, `Mục 1`, `Mục 2`,... giống với các định dạng của hàm `printf`;
- Dữ liệu được ghi vào tệp quản lý bởi con trỏ `f`.

Ví dụ 7.7. Ghi dữ liệu có định dạng lên tệp văn bản "DATA.TXT".

```
#include <stdio.h>
#include <conio.h>
int main()
{
      FILE *f;
```

```
    int i = 100;
    char c = 'C';
    float j = 1.234;
    f = fopen("DATA.TXT", "w");
    fprintf(f, "%d %c %5.3f", i, c, j);
    fclose(f);
    return 0;
}
```

7.3.2. Đọc dữ liệu từ tệp tin văn bản

- **Hàm `getc`**

Hàm `getc` dùng để đọc một ký tự từ tệp tin văn bản đang được mở, kết quả trả về mã ASCII của ký tự đọc được (*kể cả* `EOF`).

Cú pháp: `int getc(FILE *f);`

Trong đó: `f` là con trỏ quản lý tệp tin cần đọc.

Ví dụ 7.8. Đọc từng ký tự từ tệp "`TEXT.TXT`" và hiển thị lên màn hình.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    FILE *f;
    char ch;
    f = fopen("TEXT.TXT","r");
    while (!feof(f))
    {
        ch = getc(f);
        printf("%c", ch);
    }
    fclose(f);
    return 0;
}
```

- **Hàm `fgets`**

Hàm `fgets` được dùng để đọc một xâu ký tự từ tệp tin văn bản đang mở được quản lý bởi con trỏ `f`.

Cú pháp: `char *fgets(char *s, int n, FILE *f);`

Trong đó:

- `s`: Con trỏ có kiểu `char` chỉ đến vùng nhớ đủ lớn chứa các ký tự nhận được;
- `n`: Giá trị nguyên chỉ độ dài lớn nhất của xâu ký tự nhận được;
- `f`: Con trỏ quản lý tệp tin cần đọc dữ liệu;
- Ký tự `NULL ('\0')` tự động được thêm vào cuối xâu kết quả;
- Hàm trả về địa chỉ đầu tiên của vùng nhớ khi không gặp lỗi và chưa gặp ký tự kết thúc `EOF`. Ngược lại, hàm trả về giá trị `NULL`.

Hàm sẽ đọc một xâu cho đến khi đọc đủ `n` ký tự hoặc gặp ký tự xuống dòng `'\n'` (*ký tự này cũng được đưa vào chuỗi kết quả*) hay gặp ký tự kết thúc `EOF` (*ký tự này không được đưa vào chuỗi kết quả*).

Ví dụ 7.9. Sử dụng hàm `fgets` để đọc dữ liệu từ tệp văn bản "`TEXT.TXT`", độ dài mỗi dòng là 20 ký tự.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     FILE *f;
     char st[20];
     f = fopen("TEXT.TXT", "r");
     while (!feof(f))
     {
      fgets(st, 20, f);
```

```
      printf("%s", st);
      }
      fclose(f);
      return 0;
}
```

7.4. Các thao tác với tệp tin nhị phân

7.4.1. Ghi dữ liệu lên tệp tin nhị phân

Cú pháp: `size_t fwrite(const void *ptr, size_t size, size_t n, FILE *f);`

Trong đó:

- `ptr`: Con trỏ trỏ tới vùng nhớ chứa thông tin cần ghi lên tệp tin;
- `size`: Kích thước của mỗi phần tử;
- `n`: Số phần tử sẽ ghi lên tệp tin;
- `f`: Con trỏ tệp tin đã được mở;
- Giá trị trả về của hàm này là số phần tử thực sự được ghi lên tệp tin. Giá trị này bằng `n` trừ khi xuất hiện lỗi.

Ví dụ 7.10. Lưu thông tin của cấu trúc `TestStruct` vào tệp văn bản "`STRUCT.DAT`".

```
#include <stdio.h>
#include <conio.h>
typedef struct TestStruct
{
      int i;
      char ch;
};
int main()
```

```
{
      FILE *f;
      TestStruct s;
      f = fopen("STRUCT.DAT", "wb");
      s.i = 65;
      s.ch = 'A';
      fwrite(&s, sizeof(s), 1, f);
      fclose(f);
      return 0;
}
```

7.4.2. Đọc dữ liệu từ tệp tin nhị phân

Cú pháp: `size_t fread(const void *ptr, size_t size, size_t n, FILE *f);`

Trong đó:

- `ptr`: Con trỏ trỏ tới vùng nhớ sẽ nhận dữ liệu từ tệp tin.
- `n`: Số phần tử được đọc từ tệp tin.
- `size`: Kích thước của mỗi phần tử.
- `f`: Con trỏ quản lý tệp tin đã được mở.
- Giá trị trả về của hàm này là số phần tử thực sự đã đọc được từ tệp tin. Giá trị này bằng `n` hay nhỏ hơn `n` nếu con trỏ đã đến cuối tệp tin hoặc có lỗi xuất hiện.

Ví dụ 7.11. Đọc thông tin từ tệp "`STRUCT.DAT`" vào biến cấu trúc `TestStruct`.

```
#include <conio.h>
#include <stdio.h>
typedef struct TestStruct
```

```
{
      int i;
      char ch;
};
void hien_thi(TestStruct s)
{
      printf("%5d %5c\n", s.i, s.ch);
}
int main()
{
      FILE *f;
      TestStruct s;
      f = fopen("STRUCT.DAT", "rb");
      while (!feof(f))
      {
            fread(&s, sizeof(s), 1, f);
            hien_thi(s);
      }
      fclose(f);
      return 0;
}
```

7.5. Di chuyển con trỏ tệp tin

Việc ghi hay đọc dữ liệu từ tệp tin sẽ làm cho con trỏ tệp tin dịch chuyển một số `byte`, đây chính là kích thước của kiểu dữ liệu của mỗi phần tử của tệp tin. Khi đóng tệp tin rồi mở lại nó, con trỏ luôn ở vị trí ngay đầu tệp tin. Nhưng nếu ta sử dụng kiểu mở tệp tin là `"a"` để ghi nối dữ liệu, con trỏ tệp tin sẽ di chuyển đến vị trí cuối cùng của tệp tin

này. Ta cũng có thể điều khiển việc di chuyển con trỏ tệp tin đến vị trí chỉ định bằng hàm `fseek`.

Cú pháp: `int fseek(FILE *f, long offset, int whence);`

Trong đó:

- `f`: Con trỏ tệp tin đang thao tác;
- `offset`: Số `byte` cần dịch chuyển con trỏ tệp tin kể từ vị trí trước đó. Phần tử đầu tiên là vị trí `0`;
- `whence`: Vị trí bắt đầu để tính `offset`, ta có thể chọn điểm xuất phát là:

0	`SEEK_SET`	Vị trí đầu tệp tin
1	`SEEK_CUR`	Vị trí hiện tại của con trỏ tệp tin
2	`SEEK_END`	Vị trí cuối tệp tin

- Kết quả trả về của hàm là `0` nếu việc di chuyển thành công, ngược lại hàm trả về một giá trị khác `0` (*là một mã lỗi*).

Chẳng hạn:

Đưa con trỏ về đầu tệp đang mở: `fseek(f, 0, SEEK_SET);`

Đưa con trỏ về cuối tệp đang mở: `fseek(f, 0, SEEK_END);`

BÀI THỰC HÀNH CHƯƠNG 7

- **Mục tiêu**

Qua bài thực hành người học cần đạt được:

- **Về kiến thức**
  - + Nắm được ý nghĩa của việc sử dụng dữ liệu kiểu tệp;
  - + Nắm được ý nghĩa và cách sử dụng các hàm xử lý tệp.
- **Về kỹ năng**
  - + Vận dụng dữ liệu kiểu tệp để có thể hạn chế việc nhập dữ liệu trực tiếp từ bàn phím cho các bài toán;
  - + Vận dụng dữ liệu kiểu tệp để lưu dữ liệu ra của các bài toán.

- **Nội dung**

Bài 1. Viết chương trình nhập vào từ bàn phím `n` số nguyên. Ghi các số đó và một tệp văn bản.

Yêu cầu:

+ Bổ sung vào chương trình trên đoạn chương trình đọc tệp;

+ Hiển thị nội dung của tệp.

Bài 2. Cho biết đoạn chương trình sau thực hiện nhiệm vụ gì? Hoàn thành chương trình để có thể biên dịch, chạy chương trình. Kiểm tra kết quả.

```
while (!feof(f))
{
     fscanf(f, "%d", &a);
     if ((a > 0) && (a % 2 == 0))
          s += a;
}
printf("%5d", s);
```

Bài 3. Cho một tệp văn bản `input.txt` gồm các số nguyên cách nhau ít nhất một ký tự cách trống hoặc ít nhất một ký tự xuống dòng. Viết chương trình đọc các giá trị trong tệp vào một mảng một chiều. Hiển thị lên màn hình mảng đã tạo.

Yêu cầu:

+ Thay đổi các giá trị trong tệp `input.txt`;

+ Hiển thị mảng kết quả.

Bài 4. Cho một tệp văn bản gồm các xâu ký tự. Viết chương trình đọc tệp, đếm và thông báo ra màn hình số chữ cái in hoa có trong tệp.

Yêu cầu:

+ Bổ sung thêm đoạn chương trình đếm số chữ cái thường, số chữ số có trong tệp;

+ Biên dịch và chạy lại chương trình. Kiểm tra kết quả.

Bài 5. Viết chương trình tạo một tệp kiểu số nguyên với `n` số nguyên bất kỳ. Đổi các số nguyên trong tệp trên thành xâu nhị phân, ghi kết quả vào một tệp văn bản khác, mỗi dòng gồm:

```
Số tự nhiên      Xâu nhị phân
```

Yêu cầu:

+ Bổ sung thêm đoạn chương trình (hoặc hàm) đổi số nguyên ra xâu Hexa, kết quả ghi vào tệp:

```
Số tự nhiên      Xâu nhị phân     Xâu nhị Hexa
```

+ Biên dịch và chạy lại chương trình.

- **Câu hỏi củng cố**

  - Có thể mở một tệp vừa ghi vừa đọc được không?
  - Có thể ghi bổ sung dữ liệu vào một tệp đã có sẵn không?
  - Có thể hiển thị nội dung của tệp nhị phân trực tiếp trên màn hình soạn thảo không không?

- Dữ liệu vào của một chương trình được lưu trong tệp `input.txt`, dữ liệu ra được lưu trong tệp `output.txt`. Khi có sự thay đổi dữ liệu trong tệp `input.txt` thì dữ liệu trong tệp `output.txt` có tự động thay đổi theo không?
- Ghi giá trị hai biến nguyên `a = 1, b = 2` vào một tệp văn bản bằng câu lệnh `fprintf(f,"%d%d", a, b);` sau đó, mở tệp để đọc nội dung của tệp vào cho hai biến nguyên `x, y` bằng câu lệnh `fscanf(f, "%d%d", &x, &y);` thì giá trị của `x = 1` và `y = 2` đúng hay sai?

▪ **Tự học**

Bài 6. Viết chương trình nhập từ bàn phím một mảng hai chiều $a_{m \times n}$ gồm các số nguyên. Ghi mảng hai chiều vào tệp văn bản theo quy tắc: dòng đầu ghi hai số `m, n`, các dòng sau ghi các giá trị phần tử của mảng dưới dạng ma trận gồm `m` hàng và `n` cột.

Bài 7. Cho một tệp văn bản có dòng đầu tiên là hai số nguyên dương `m, n`, các dòng tiếp theo gồm `m × n` số thực, các số cách nhau ít nhất một dấu cách trống hoặc ít nhất một ký tự xuống dòng. Viết chương trình đọc hai giá trị `m, n` làm số phần tử chiều 1 và số phần tử chiều 2 của một mảng hai chiều, sau đó đọc các giá trị thực vào cho các phần tử của mảng. Hiển thị lên màn hình mảng đã tạo được dưới dạng ma trận.

Bài 8. Cho một tệp văn bản gồm các số nguyên cách nhau ít nhất một ký tự cách trống hoặc ít nhất một ký tự xuống dòng. Viết chương trình hiển thị lên màn hình các số nguyên tố có trong tệp trên, mỗi dòng 10 số.

Bài 9. Cho một tệp văn bản, mỗi dòng gồm hai số nguyên dương cách nhau ít nhất một ký tự trống. Viết chương trình tìm ước chung lớn nhất và bội chung nhỏ nhất của các cặp số đọc ra từ tệp trên, kết quả ghi vào một tệp văn bản khác theo quy tắc:

```
a     b     ucln        bcnn
```

Chẳng hạn:
```
4    6    2    12
3    7    1    21
```

Gợi ý: Có thể xây dựng các hàm: Hàm tính ước chung lớn nhất của hai số, áp dụng hàm ước chung lớn nhất để viết hàm tính bội chung nhỏ nhất của hai số,...

Bài 10. Viết chương trình tạo tệp lưu trữ các thông tin sinh viên, mỗi sinh viên gồm: `họ tên`, `tuổi`, `điểm kỳ 1`, `điểm kỳ 2`, `điểm trung bình cả năm`, trong đó:

`điểm trung bình cả năm = (điểm kỳ 1 + điểm kỳ 2 * 2 ) / 3`

Cho biết sinh viên có `điểm trung bình cả năm` lớn nhất.

Phụ lục A

CÁC CHỈ THỊ TIỀN XỬ LÝ

Các chỉ thị tiền xử lý giúp cho việc soạn thảo chương trình nguồn được thuận lợi hơn. Các chỉ thị này không phải là câu lệnh nên không kết thúc bằng dấu chấm phẩy (`;`). Khi biên dịch chương trình C, không phải chính chương trình nguồn ta soạn thảo được dịch mà trước khi biên dịch các tiền xử lý sẽ chỉnh lý bản gốc sau đó bản chỉnh lý này sẽ được dịch. Các tài liệu tham khảo chính của Phụ lục A bao gồm [1, 2, 4, 5, 7, 8]. Các thao tác chỉnh lý bao gồm:

- Thao tác thay thế;
- Thao tác chèn tệp;
- Thao tác lựa chọn.

A.1. Chỉ thị `#define` đơn giản

Cú pháp: **`#define`** `<Tên> <Nội dung cần thay thế>`

Thay `Tên` bằng `Nội dung cần thay thế`, trong đó `Nội dung cần thay thế` có thể là hàm, biểu thức, hằng hay đoạn chương trình. Nếu `Nội dung cần thay thế` dài hơn một dòng thì có thể sử dụng ký tự '\' vào cuối dòng trước.

Chẳng hạn:

```
#define hien_thi printf
#define Max 1000
#define N (2 + 3)
#define nhap {printf("Nhap so nguyen: ");
scanf("%d", &n);}
```

A.2. Chỉ thị `#undef`

Cú pháp: **`#undef`** `Tên`

Giải phóng tên đã định nghĩa bằng `#define`

Ví dụ A1. Sử dụng chỉ thị `#define và #undef`.

```
#include <conio.h>
#include <stdio.h>
#define hien_thi printf
#define N 100
int main()
{
     int M = 200;
     hien_thi("\nM = %5d N = %5d", M, N);
     #define M 300
     hien_thi("\nM = %5d N = %5d", M, N);
     #undef M
     #define M 400
     hien_thi("\nM = %5d N = %5d", M, N);
     getch();
     return 0;
}
```

Trong chương trình trên: `printf` được thay bằng `hien_thi`, `N` (*ở ngoài hàm* `main`) được thay thế bằng `100`, `M` ở câu lệnh `hien_thi` thứ nhất có giá trị là `200`, ở câu lệnh `hien_thi` thứ hai có giá trị là `300`, ở câu lệnh `hien_thi` thứ ba có giá trị bằng `400`.

A.3. Chỉ thị `#define` có đối số

Ta có thể định nghĩa các macro tương tự như các hàm.

Cú pháp:

`#define` <Tên macro>(<Danh sách đối số >) <Nội dung macro>

Chẳng hạn:

```
#define max(a, b) (a)>(b)?(a):(b)
```

Đối với những hàm đơn giản ta có thể xây dựng `macro`, đối số của các `macro` không ứng với kiểu dữ liệu cụ thể nào. Như vậy, `macro max` có thể dùng cho dữ liệu kiểu dữ số nguyên, số thực hoặc ký tự,…

✍ Lưu ý:

- Giữa `Tên macro` và dấu ngoặc mở không được có dấu cách. Chẳng hạn, trong ví dụ trên nếu viết `max (<Danh sách đối số>)` là không hợp lệ.
- Khi viết biểu thức thay thế, các đối số hình thức (*trong ví dụ trên là* `a` *và* `b`) cần được đặt giữa cặp ngoặc để đảm bảo tính chính xác khi thay thế.

Chẳng hạn:

Nếu định nghĩa

```
#define tich(a, b) (a)*(b)
```

Khi gọi **`tich(x + y, z)`** thì kết quả thay thế là **`(x + y) * z`**

Nếu định nghĩa

```
#define tich(a, b) a * b
```

Khi gọi **`tich(x + y, z)`** thì kết quả thay thế là **`x + y * z`**

A.4. Chỉ thị bao hàm tệp #include

Chỉ thị bao hàm #include có tác dụng trước khi biên dịch chương trình nguồn, chương trình dịch sẽ tìm tệp theo tên và đường dẫn ghi trong chỉ thị #include và thay thế vào đúng vị trí của #include.

Có thể khai báo chỉ thị bao hàm tệp theo một trong hai dạng sau:

Dạng 1: **#include** <[<Đường dẫn>]<\Tên tệp>>

Dạng 2: **#include** "[<Đường dẫn>]<\Tên tệp>"

Chẳng hạn:

```
#include <C:\TV\thuvien.h>
#include "C:\TV\thuvien.h"
#include <thuvien.cpp>
#include "thuvien.cpp"
```

Hai cách trên chỉ có ý nghĩa khác nhau khi không có thông tin về đường dẫn. Nếu khai báo theo *Dạng 1* mà không có đường dẫn, trình biên dịch sẽ tìm tệp thuvien.h trong thư mục INCLUDE, còn nếu khai báo theo *Dạng 2* thì trình biên dịch sẽ tìm trong thư mục gốc trước rồi mới đến thư mục INCLUDE.

- **Lợi ích của việc sử dụng chỉ thị #include**

- **Tổ chức chương trình trên nhiều tệp**

Điều này thuận lợi cho việc khi một nhóm người phát triển chương trình. Một chương trình phức tạp có thể gồm nhiều hàm, các hàm có thể chứa ở các tệp khác nhau. Sử dụng chỉ thị #include có thể ghép các hàm trên các tệp khác nhau vào tệp gốc để tạo chương trình hoàn chỉnh.

- **Tổ chức thư viện**

Những hàm có tần suất sử dụng nhiều trong các chương trình ta có thể tổ chức chúng vào một thư viện. Khi xây dựng chương trình cần đến những hàm đó thì ta có thể dùng chỉ thị #include để dùng chúng.

- **Sử dụng các tệp tiêu đề**

Là những tệp mà ngôn ngữ lập trình C đã cung cấp sẵn, trong đó đã khai báo, định nghĩa sẵn các hàm chuẩn, định nghĩa các hằng, các kiểu dữ liệu. Khi muốn sử dụng các hàm, hằng, kiểu dữ liệu chuẩn của ngôn ngữ lập trình C ta phải khai báo tệp tiêu đề chứa nó.

Chẳng hạn:

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

A.5. Chỉ thị biên dịch có điều kiện `#if`

***Dạng 1*:**

```
#if <Biểu thức hằng>
    <Đoạn chương trình>
#endif
```

Ý nghĩa: Nếu `Biểu thức hằng` khác `0` thì trình biên dịch sẽ biên dịch `Đoạn chương trình` giữa `#if` và `#endif`, ngược lại thì bỏ qua.

Ví dụ A2. Sử dụng chỉ thị biên dịch có điều kiện `#if`.

```
#include <conio.h>
#include <stdio.h>
#define N 200
int main()
{
     #if N>100
          printf("Bien dich voi N > 100");
     #endif
     getch();
     return 0;
}
```

Dạng 2:

```
#if <Biểu thức hằng>
    <Đoạn chương trình 1>
#else
    <Đoạn chương trình 2>
#endif
```

Ý nghĩa: Nếu `Biểu thức hằng` khác `0` thì trình biên dịch sẽ biên dịch `Đoạn chương trình 1`, ngược lại biên dịch `Đoạn chương trình 2`.

Ví dụ A3. Sử dụng chỉ thị biên dịch có điều kiện `#if #else`.

```
#include <conio.h>
#include <stdio.h>
#define N 100
int main()
{
     #if N > 100
          printf("Bien dich voi N > 100");
     #else
          printf("Bien dich voi N <= 100");
     #endif
     getch();
     return 0;
}
```

Dạng 3:

```
#if   <Biểu thức hằng 1>
      <Đoạn chương trình 1>
#elif <Biểu thức hằng 2>
```

```
        <Đoạn chương trình 2>
        …
#elif <Biểu thức hằng n>
        <Đoạn chương trình n>
[#else
        <Đoạn chương trình n + 1>]
#endif
```

***Ý nghĩa**:* Nếu `Biểu thức hằng 1` khác `0` thì biên dịch `Đoạn chương trình 1`, ngược lại nếu `Biểu thức hằng 2` khác `0` thì biên dịch `Đoạn chương trình 2`, …, nếu các `Biểu thức hằng i` (`i = 1, 2, ...,n`) đều bằng `0` mà có `#else` thì biên dịch `Đoạn chương trình n + 1`.

✍ **Lưu ý:** Có thể sử dụng các chỉ thị `#if` lồng nhau dạng

```
#if
  #if …
  #else…
  #endif
#else…
#endif
```

A.5. Chỉ thị biên dịch có điều kiện `#ifdef` và `#ifndef`

Có thể sử dụng các chỉ thị biên dịch có điều kiện `#ifdef` và `#ifndef` theo các dạng sau:

Dạng 1:

```
#ifdef <Tên macro>
        <Đoạn chương trình>
#endif
```

Ý nghĩa: Nếu đã định nghĩa `Tên macro` (*bởi* `#define`) thì trình biên dịch sẽ biên dịch `Đoạn chương trình` nằm giữa `#ifdef` và `#endif`, ngược lại bỏ qua.

Dạng 2:

```
#ifdef <Tên macro>
        <Đoạn chương trình 1>
#else
        <Đoạn chương trình 2>
#endif
```

Ý nghĩa: Nếu đã định nghĩa `Tên macro` (*bởi* `#define`) thì trình biên dịch sẽ biên dịch `Đoạn chương trình 1`, ngược lại biên dịch `Đoạn chương trình 2`.

Dạng 3:

```
#ifndef <Tên macro>
         <Đoạn chương trình>
#endif
```

Ý nghĩa: Nếu chưa định nghĩa `Tên macro` thì trình biên dịch sẽ biên dịch `Đoạn chương trình` nằm giữa `#ifndef` và `#endif`, ngược lại bỏ qua.

Dạng 4:

```
#ifndef <Tên macro>
         <Đoạn chương trình 1>
#else
         <Đoạn chương trình 2>
#endif
```

Ý nghĩa: Nếu chưa định nghĩa `Tên macro` thì trình biên dịch sẽ biên dịch `Đoạn chương trình 1`, ngược lại biên dịch `Đoạn chương trình 2`.

✍ **Lưu ý:** Có thể tổ chức các chỉ thị tiền xử lý `#ifdef` hoặc `#ifndef` lồng nhau như `#if`.

Ví dụ A4. Sử dụng chỉ thị biên dịch có điều kiện `#ifdef` và `#ifndef`.

```
#include <conio.h>
#include <stdio.h>
#define N 200
int main()
{
      #ifdef N
            printf("Da dinh nghia N");
      #else
            printf("Chua dinh nghia N");
      #endif
      #ifndef M
            printf("Chua dinh nghia M");
      #endif
      getch();
      return 0;
}
```

✍ **Lưu ý:** Nếu muốn kết nối nhiều lần một tệp thư viện vào một chương trình nguồn mà không gây lỗi ta sử dụng các chỉ thị tiền xử lý `#if, #ifdef, #ifndef`.

Ví dụ A5. Tạo tệp tiêu đề `thuvien.h`.

```
#ifndef M
#define M 100
/* Noi dung tep thuvien.h */
#endif
```

Lúc này nếu kết nối nhiều lần tệp `thuvien.h` vào một chương trình nguồn nào đó thì `M` sẽ được định nghĩa một lần khi gặp lần đầu tiên, còn các lần sau sẽ bỏ qua.

A.6. Chỉ thị `#error`

Cú pháp:

`#error` <Thông báo lỗi>

Chỉ thị này cho phép dừng biên dịch chương trình và thông báo lỗi.

Ví dụ A6. Sử dụng chỉ thị thông báo lỗi `#error`.

```
#include <conio.h>
#include <stdio.h>
#ifndef N
#error Chua dinh nghia hang N
#else
int main()
{
     int M;
     M = 10 * N;
     printf("%5d", M);
     getch();
     return 0;
}
#endif
```

Phụ lục B

BẢNG MÃ ASCII

Tất cả các máy tính muốn trao đổi thông tin qua các thiết bị vào/ra phải sử dụng một tập hữu hạn các ký tự có sắp thứ tự. Có nhiều cách sắp xếp thứ tự cho tập các ký tự này và không có cách sắp nào là chuẩn cho mọi loại máy tính. Tuy nhiên, một bộ mã ký tự rất phổ biến được dùng để trao đổi thông tin trên máy tính đó là bộ mã ký tự ASCII (*American Standard Code for Information Interchange*). Với bộ mã này các ký tự được mã hóa bởi 1 `byte`, vì vậy bộ mã có thể mã hóa 256 ($256 = 2^8$) ký tự. Tuy nhiên, số ký tự cơ bản nhất là 128 ký tự đầu (*từ 0 đến 127*) trong bộ mã, còn 128 ký tự tiếp theo (*từ 128 đến 255*) được gọi là phần mã mở rộng được dùng để mã hóa một số ký tự toán học, ký tự đồ họa. Ở đây chúng ta quan tâm đến 128 ký tự đầu của bảng mã ASCII là những ký tự chuẩn để xây dựng bộ ký tự cho ngôn ngữ lập trình.

Các tài liệu tham khảo chính của Phục lục B bao gồm [1, 6].

Có thể chia 256 ký tự trong bảng mã ASCII làm ba nhóm:

- **Nhóm 1:** Nhóm các ký tự điều khiển có mã từ 0 đến 31. Các ký tự này dùng để điều khiển các thiết bị ngoại vi, điều khiển các thủ tục trao đổi thông tin. Chẳng hạn, ký tự mã 13 dùng để chuyển con trỏ màn hình về đầu dòng, ký tự 10 chuyển con trỏ màn hình xuống dòng dưới (trên cùng một cột). Các ký tự nhóm này không hiển thị lên màn hình (*Bảng B1*).

- **Nhóm 2:** Nhóm các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể đưa ra màn hình hoặc máy in (*Bảng B1*).
- **Nhóm 3:** Nhóm các ký tự mở rộng có mã từ 128 đến 255. Các ký tự này có thể đưa ra màn hình nhưng không in ra được (*Bảng B2*).

Hex	0	1	2	3	4	5	6	7
0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	STX 2	DC2 18	“ 34	2 50	B 66	R 82	b 98	r 114
3	♥ 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	♦ 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	♣ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
6	♠ 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
7	BEL 7	ETB 23	‘ 39	7 55	G 71	W 87	g 103	w 119
8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	EM 25	) 41	9 57	I 73	Y 89	I 105	y 121
A	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
E	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	DEL 127

Bảng B1. Mã ASCII của 128 ký tự đầu tiên (từ ký tự thứ 0 đến 127)

Hex	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	░ 176	└ 192	╙ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	▒ 177	┴ 193	╤ 209	ß 225	± 241
2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
4	ä 132	ö 148	ñ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
5	à 133	ò 149	Ñ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
6	å 134	û 150	ª 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
7	ç 135	ù 151	º 167	╖ 183	╟ 199	╫ 215	τ 231	≈ 247
8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
9	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	Θ 233	∙ 249
A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Ω 234	· 250
B	ï 139	¢ 155	½ 171	╗ 187	╦ 203	█ 219	δ 235	√ 251
C	î 140	£ 156	¼ 172	╝ 188	╠ 204	■ 220	∞ 236	ⁿ 252
D	ì 141	¥ 157	¡ 173	╜ 189	═ 205	▌ 221	φ 237	² 253
E	Ä 142	₧ 158	« 174	╛ 190	╬ 206	▐ 222	ε 238	■ 254
F	Å 143	ƒ 159	» 175	┐ 191	╧ 207	■ 223	∩ 239	255

Bảng B2. Mã ASCII của 128 ký tự mở rộng (từ ký tự thứ 128 đến 255)

TÀI LIỆU THAM KHẢO

[1]. Phạm Văn Ất, *Kỹ thuật lập trình C cơ sở và nâng cao*, NXB Giao thông Vận tải, 2006.

[2]. Dương Tử Cường, *Ngôn ngữ lập trình C - Học và sử dụng*, NXB Khoa học và Kỹ thuật, 1997.

[3]. Lê Văn Doanh, Trần Khắc Tuấn, Lê Đình Anh, *101 thuật toán và chương trình bằng ngôn ngữ C*, NXB Khoa học và Kỹ thuật, 1996.

[4]. Quách Tuấn Ngọc, *Ngôn ngữ lập trình C*, NXB Thống kê, 2003.

[5]. Nguyễn Đình Tê, Hoàng Đức Hải, *Giáo trình lý thuyết và bài tập ngôn ngữ C*, NXB Giáo dục, 1998.

[6]. Ngô Trung Việt, *Ngôn ngữ lập trình C và C++*, NXB Thống kê, 2003.

[7]. Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language*, Prentice - Hall, 1988.

[8]. Bryon Gottfried, *Programming with C*, Mc Graw Hill, 1996.

NHÀ XUẤT BẢN ĐẠI HỌC VINH
182 Lê Duẩn, Vinh, Nghệ An
Điện thoại: 038. 3551 345 - Fax: 038. 3855 269
Email: nxbdhv@gmail.com

GIÁO TRÌNH
NGÔN NGỮ LẬP TRÌNH C

Chịu trách nhiệm xuất bản:
Giám đốc kiêm Tổng Biên tập
PGS.TS. ĐINH TRÍ DŨNG

Chịu trách nhiệm nội dung:
HỘI ĐỒNG NGHIỆM THU TRƯỜNG ĐẠI HỌC VINH

Người nhận xét:
PGS.TS. LÊ HUY THẬP
ThS. LÊ VĂN TẤN

Biên tập:
ĐINH ĐỨC TÀI
PHAN ANH PHONG

Bìa, trình bày:
PHAN QUỐC TRƯỜNG

Sửa bản in:
TÁC GIẢ

ISBN 978-604-923-159-9

In 300 bản, khổ 16 x 24 cm
Tại Công ty CP In Hà Tĩnh, số 53 Hà Huy Tập, TP. Hà Tĩnh
Đăng ký kế hoạch xuất bản số: 1272-2015/CXBIPH/01-23/ĐHV
Quyết định xuất bản số: 97/QĐXB-ĐHV ngày 5 tháng 10 năm 2015
In xong và nộp lưu chiểu tháng 10 năm 2015