# PHẦN III. LẬP TRÌNH

## III.1. Tổng quan về ngôn ngữ C

### III.1.1. Lịch sử phát triển

Đầu những năm 1970, việc lập trình hệ thống vẫn dựa trên hợp ngữ (*Assembly*) nên công việc nặng nề, phức tạp và khó chuyển đổi chương trình giữa các hệ thống máy tính khác nhau. Điều này dẫn đến nhu cầu cần có một ngôn ngữ lập trình hệ thống bậc cao đồng thời có khả năng chuyển đổi dễ dàng từ hệ thống máy tính này sang hệ thống máy tính khác (còn gọi là tính *khả chuyển – portability*) để thay thế hợp ngữ.

Cũng vào thời gian đó, người ta muốn viết lại hệ điều hành Unix để cài đặt trên các hệ máy tính khác nhau, vì vậy cần có một ngôn ngữ lập trình hệ thống có tính khả chuyển cao để viết lại hệ điều hành Unix.

Ngôn ngữ C ra đời tại phòng thí nghiệm BELL của tập đoàn AT&T (Hoa Kỳ) do Brian W. Kernighan và Dennis Ritchie phát triển vào đầu những năm 1970 và hoàn thành vào năm 1972.

C được phát triển dựa trên nền các ngôn ngữ BCPL (*Basic Combined Programming Language*) và ngôn ngữ B. Cũng vì được phát triển dựa trên nền ngôn ngữ B nên ngôn ngữ mới được Brian W. Kernighan và Dennis Ritchie đặt tên là ngôn ngữ C như là sự tiếp nối ngôn ngữ B.

C có các đặc điểm là một ngôn ngữ lập trình hệ thống mạnh, khả chuyển, có tính linh hoạt cao và có thế mạnh trong xử lí các dạng dữ liệu số, văn bản, cơ sở dữ liệu. Vì vậy C thường được dùng để viết các chương trình hệ thống như hệ điều hành (ví dụ hệ điều hành Unix có 90% mã nguồn được viết bằng C, 10% còn lại viết bằng hợp ngữ) và các chương trình ứng dụng chuyên nghiệp có can thiệp tới dữ liệu ở mức thấp như xử lí văn bản, cơ sở dữ liệu, xử lí ảnh…

W. Kernighan và Dennis Ritchie công bố ngôn ngữ C trong lần xuất bản đầu của cuốn sách "*The C programming language*" (1978). Sau đó người ta đã bổ sung thêm những yếu tố và khả năng mới vào trong ngôn ngữ C (ví dụ như đưa thêm kiểu liệt kê **enum**, cho phép kiểu dữ liệu trả về bởi hàm là kiểu **void**, **struct** hoặc **union**… và đặc biệt là sự bổ sung các thư viện cho ngôn ngữ. Lúc đó đồng thời tồn

tại nhiều phiên bản khác nhau của ngôn ngữ C nhưng không tương thích với nhau. Điều này gây khó khăn cho việc trao đổi mã nguồn chương trình C viết trên các phiên bản ngôn ngữ C khác nhau (bạn sẽ rất khó đọc và hiểu chương trình của người khác và khi bạn muốn sửa nó thành chương trình của mình và dịch trên bộ dịch của mình thì sẽ tốn rất nhiều công sức) điều này dẫn đến nhu cầu chuẩn hóa ngôn ngữ C.

Năm 1989, Viện tiêu chuẩn Quốc gia của Hoa Kỳ (*American National Standards Institute* - ANSI) đã công bố phiên bản chuẩn hóa của ngôn ngữ C trong lần tái bản thứ 2 cuốn sách "*The C programming language*" của các tác giả W. Kernighan và Dennis Ritchie. Từ đó đến nay phiên bản này vẫn thường được nhắc đến với tên gọi là *ANSI C, C chuẩn* hay C89 (vì được công bố năm 1989). ANSI C là sự kế thừa phiên bản đầu tiên của ngôn ngữ C (do K. Kernighan & D. Ritchie công bố năm 1978) có đưa thêm vào nhiều yếu tố mới và ANSI C đã quy định các thư viện chuẩn dùng cho ngôn ngữ C. Tất cả các phiên bản của ngôn ngữ C hiện đang sử dụng đều tuân theo các mô tả đã được nêu ra trong ANSI C, sự khác biệt nếu có thì chủ yếu nằm ở việc đưa thêm vào các thư viện bổ sung cho thư viện chuẩn của ngôn ngữ C.

Hiện nay cũng có nhiều phiên bản của ngôn ngữ C khác nhau và mỗi phiên bản này gắn liền với một bộ chương trình dịch cụ thể của ngôn ngữ C. Các bộ chương trình dịch phổ biến của ngôn ngữ C có thể kể tên như:

- Turbo C++ và Borland C++ của Borland Inc.
- MSC và VC của Microsoft Corp.
- GCC của GNU project.

...

Trong các trình biên dịch trên thì Turbo C++ là trình biên dịch rất quen thuộc và sẽ được chọn làm trình biên dịch cho các ví dụ sử dụng trong tài liệu này.

## III.1.2. Các phần tử cơ bản của ngôn ngữ C

### III.1.2.1. Tập kí tự

Chương trình nguồn của mọi ngôn ngữ lập trình đều được tạo nên từ các phần tử cơ bản là tập kí tự của ngôn ngữ đó. Các kí tự tổ hợp với nhau tạo thành các từ, các từ liên kết với nhau theo một quy tắc xác định (quy tắc đó gọi là cú pháp của ngôn ngữ) để tạo thành các câu lệnh. Từ các câu lệnh người ta sẽ tổ chức nên chương trình.

Tập kí tự sử dụng trong ngôn ngữ lập trình C gồm có:

| | |
|---|---|
| 26 chữ cái hoa: | A B C ... X Y Z |
| 26 chữ cái thường: | a b c ... x y z |
| 10 chữ số: | 0 1 2 3 4 5 6 7 8 9 |
| Các kí hiệu toán học: | + - * / = < > |
| Các dấu ngăn cách: | . ; , : space tab |
| Các dấu ngoặc: | ( ) [ ] { } |
| Các kí hiệu đặc biệt: | _ ? $ & # ^ \ ! ' " ~ .v.v. |

### III.1.2.2. Từ khóa

Từ khóa (*Keyword*) là những từ có sẵn của ngôn ngữ và được sử dụng dành riêng cho những mục đích xác định.

Một số từ khóa hay dùng trong Turbo C++:

| | | | | | |
|---|---|---|---|---|---|
| **break** | **case** | **char** | **const** | **continue** | **default** |
| **do** | **double** | **else** | **enum** | **float** | **for** |
| **goto** | **if** | **int** | **interrupt** | **long** | **return** |
| **short** | **signed** | **sizeof** | **static** | **struct** | **switch** |
| **typedef** | **union** | **unsigned** | **void** | **while** | |

*Chú ý: Tất cả các từ khóa trong C đều viết bằng chữ thường.*

Các từ khóa trong C được sử dụng để:

- Đặt tên cho các kiểu dữ liệu: **int**, **float**, **double**, **char**, **struct**, **union**...
- Mô tả các lệnh, các cấu trúc điều khiển: **for**, **do**, **while**, **switch**, **case**, **if**, **else**, **break**, **continue**...

### III.1.2.3. Định danh

Định danh (*Identifier* – hoặc còn gọi là *Tên*) là một dãy các kí tự dùng để gọi tên các đối tượng trong chương trình. Các đối tượng trong chương trình gồm có biến, hằng, hàm, kiểu dữ liệu..., ta sẽ làm quen ở những mục tiếp theo.

Định danh có thể được đặt tên sẵn bởi ngôn ngữ lập trình (đó chính là các từ khóa) hoặc do người lập trình đặt. Khi đặt tên cho định danh trong C, người lập trình cần tuân thủ các quy tắc sau :

1. Các kí tự được sử dụng trong các định danh của ngôn ngữ C chỉ được gồm có: Chữ cái, chữ số và dấu gạch dưới "_" (*underscore*).
2. Bắt đầu của định danh phải là chữ cái hoặc dấu gạch dưới, không được bắt đầu định danh bằng chữ số.
3. Định danh do người lập trình đặt không được trùng với từ khóa.
4. Turbo C++ không giới hạn độ dài của định danh, nhưng chỉ 32 kí tự đầu của định danh được chương trình biên dịch sử dụng (khi định danh có độ dài lớn hơn 32 kí tự thì Turbo C++ sẽ tự động cắt bỏ, không xem xét các kí tự cuối bắt đầu từ kí tự thứ 33).

Ngoài các quy tắc bắt buộc trên, lập trình viên có quyền tùy ý đặt tên định danh trong chương trình của mình.

Một số ví dụ về định danh:

i, x, y, a, b, _function, _MY_CONSTANT, PI, gia_tri_1...

Ví dụ về định danh không hợp lệ:

| | |
|---|---|
| 1_a, 3d, 55x | bắt đầu bằng chữ số |
| so luong, ti le | có kí tự không hợp lệ (dấu cách – *space*) trong tên |
| int, char | trùng với từ khóa của ngôn ngữ C |

Lưu ý:

- Đôi khi định danh do ta đặt gồm nhiều từ, khi đó để dễ đọc ta nên tách các từ bằng cách sử dụng dấu gạch dưới. Ví dụ, định danh danh_sach_sinh_vien dễ đọc và dễ hiểu hơn so với định danh danhsachsinhvien.
- Định danh nên có tính chất gợi nhớ, ví dụ nếu ta muốn lưu trữ các thông tin về các sinh viên vào một biến nào đó thì biến đó nên được đặt tên là danh_sach_sinh_vien hay ds_sv... Đồng thời định danh danh_sach_sinh_vien chỉ nên dùng để đặt tên cho những đối tượng liên quan đến sinh viên chứ không nên đặt tên cho các đối tượng chứa thông tin về cán bộ viên chức. Việc đặt tên có tính gợi nhớ giúp cho người lập trình và những người khác đọc chương trình viết ra được dễ dàng hơn.

- Ngôn ngữ C phân biệt chữ cái thường và chữ cái hoa trong các định danh, tức là dinh_danh khác với Dinh_danh.
- Một thói quen của những người lập trình là các hằng thường được đặt tên bằng chữ hoa, còn các biến, hàm hay cấu trúc thì đặt tên bằng chữ thường. Nếu tên gồm nhiều từ thì ta nên phân cách các từ bằng dấu gạch dưới.

*Ví dụ:*

| **Định danh** | **Loại đối tượng** |
|---|---|
| HANG_SO_1, _CONSTANT_2 | hằng |
| a, b, i, j, count | biến |
| nhap_du_lieu, tim_kiem, xu_li | hàm |
| sinh_vien, mat_hang | cấu trúc |

### III.1.2.4. Các kiểu dữ liệu

*Kiểu dữ liệu là gì?*

Dữ liệu là tài nguyên quan trọng nhất trong máy tính, song dữ liệu trong máy tính lại không phải tất cả đều giống nhau. Có dữ liệu là chữ viết, có dữ liệu là con số, lại có dữ liệu khác là hình ảnh, âm thanh... Ta nói rằng các dữ liệu đó thuộc các kiểu dữ liệu khác nhau.

Một cách hình thức, kiểu dữ liệu có thể được định nghĩa gồm 2 điểm như sau:

- Một kiểu dữ liệu là một tập hợp các giá trị mà một dữ liệu thuộc kiểu dữ liệu đó có thể nhận được.
- Trên một kiểu dữ liệu ta xác định một số phép toán đối với các dữ liệu thuộc kiểu dữ liệu đó.

*Ví dụ:*

Trong ngôn ngữ C có kiểu dữ liệu **int**. Một dữ liệu thuộc kiểu dữ liệu **int** sẽ là một số nguyên (*integer*) và nó có thể nhận giá trị từ - 32,768 ($-2^{15}$) đến 32,767 ($2^{15} - 1$). Trên kiểu dữ liệu **int,** ngôn ngữ C định nghĩa các phép toán số học đối với số nguyên như :

| Tên phép toán | Kí hiệu |
|---|---|
| Đảo dấu | - |
| Cộng | + |
| Trừ | - |
| Nhân | * |
| Chia lấy phần nguyên | / |
| Chia lấy phần dư | % |
| So sánh bằng | = = |
| So sánh lớn hơn | > |
| So sánh nhỏ hơn | < |
| ... | |

Trong máy tính, việc phân biệt kiểu dữ liệu là cần thiết vì qua kiểu dữ liệu máy tính biết được đối tượng mà nó đang xử lí thuộc dạng nào, có cấu trúc ra sao, có thể thực hiện các phép xử lí nào đối với đối tượng đó, hay là cần phải lưu trữ đối tượng đó như thế nào...

### III.1.2.5. Hằng

Định nghĩa: Hằng (*constant*) là đại lượng có giá trị không đổi trong chương trình. Để giúp chương trình dịch nhận biết hằng ta cần nắm được cách biểu diễn hằng trong một chương trình C.

*Biểu diễn hằng số nguyên*

Trong ngôn ngữ C, một hằng số nguyên có thể được biểu diễn dưới những dạng sau

- Dạng thập phân: Là cách viết giá trị số đó dưới hệ đếm cơ số 10 thông thường.
- Dạng thập lục phân: Ta viết giá trị số đó dưới dạng hệ đếm cơ số 16 và thêm tiền tố **0x** ở đầu.
- Dạng bát phân: Ta viết giá trị số đó dưới dạng hệ đếm cơ số 8 và thêm tiền tố **0** ở đầu.

Ví dụ:

| Giá trị thập phân | Giá trị hệ bát phân | Giá trị hệ thập lục phân |
|---|---|---|
| 2007 | 03727 | 0x7D7 |
| 396 | 0614 | 0x18C |

*Biểu diễn hằng số thực*

Có hai cách biểu diễn hằng số thực:

- Dưới dạng số thực dấu phẩy tĩnh.
- Dưới dạng số thực dấu phẩy động.

Ví dụ:

| Số thực dấu phẩy tĩnh | Số thực dấu phẩy động |
|---|---|
| 3.14159 | 31.4159 E-1 |
| 123.456 | 12.3456 E+1 hoặc 1.23456 E+2 |

*Biểu diễn hằng kí tự*

Có hai cách biểu diễn hằng kí tự:

- Bằng kí hiệu của kí tự đó đặt giữa hai dấu nháy đơn.
- Bằng số thứ tự của kí tự đó trong bảng mã ASCII (lưu ý số thứ tự của một kí tự trong bảng mã ASCII là một số nguyên nên có một số cách biểu diễn).

Ví dụ

| Kí tự cần biểu diễn | Cách 1 | Cách 2 | | |
|---|---|---|---|---|
| Chữ cái A | 'A' | 65 | hoặc 0101 | hoặc 0x41 |
| Dấu nháy đơn ' | '\'' | 39 | hoặc 047 | hoặc 0x27 |
| Dấu nháy kép " | '\"' | 34 | hoặc 042 | hoặc 0x22 |
| Dấu gạch chéo ngược \ | '\\' | 92 | hoặc 0134 | hoặc 0x5c |
| Kí tự xuống dòng | '\n' | | | |
| Kí tự NUL | '\0' | 0 hoặc 00 | hoặc 0x0 | |
| Kí tự Tab | '\t' | 9 hoặc 09 | hoặc 0x9 | |

*Biểu diễn hằng xâu kí tự*

Hằng xâu kí tự được biểu diễn bởi dãy các kí tự thành phần có trong xâu đó và được đặt trong cặp dấu nháy kép.

*Ví dụ:*

"ngon ngu lap trinh C", "tin hoc dai cuong"...

### III.1.2.6. Biến

Định nghĩa: Biến (*variable*) là đại lượng mà giá trị có thể thay đổi trong chương trình.

Trong chương trình, hằng và biến được sử dụng để lưu trữ dữ liệu. Dữ liệu lưu trữ trong biến, hằng phải thuộc một kiểu dữ liệu nào đó.

Biến và hằng đều phải được đặt tên để khi cần thì có thể gọi đến. Tên biến và hằng được đặt theo quy tắc đặt tên cho định danh.

### III.1.2.7. Hàm

Trong lập trình chúng ta rất hay phải tính toán giá trị của một số đại lượng thường gặp như sin(x), cos(x), căn bậc hai, lũy thừa, logarithm...

Ngôn ngữ C cung cấp cho người lập trình một công cụ dùng để tính toán giá trị các đại lượng đó mỗi khi cần trong chương trình, đó là các hàm.

Ví dụ: Để tính giá trị của đại lượng có tên là $y$ và có giá trị $y = \sin(x)$, với $x$ là một đại lượng nào đó, trong chương trình ta viết câu lệnh y = sin(x). Trong câu lệnh này sin() là một hàm, $x$ là đối số của nó (hay còn gọi là tham số). Khi gặp hàm trong chương trình, ngôn ngữ C sẽ tự động gọi một đoạn chương trình tương ứng với hàm để tính toán giá trị của hàm đó rồi trả lại kết quả tính được cho chương trình.

Ngoài hàm sin(x), ngôn ngữ C còn có nhiều hàm khác hỗ trợ người lập trình tính toán giá trị của các đại lượng thường gặp như cos(x), sqrt(x) (để tính căn bậc hai của x), exp(x) (để tính lũy thừa $e^x$), log(x) (để tính logarithm cơ số e của x)...

*Một số hàm toán học hay được sử dụng trong C*

| Hàm | Ý nghĩa | Kí hiệu toán học | Ví dụ |
|---|---|---|---|
| sqrt(x) | Căn bậc 2 của *x* | $\sqrt{x}$ | sqrt(16.0) bằng 4.0 |
| pow(x,y) | *x* mũ *y* | $x^y$ | pow(2,3) bằng 8 |
| exp(x) | *e* mũ *x* | $e^x$ | exp(1.0) bằng 2.718282 |
| log(x) | logarithm tự nhiên (cơ số e) của x | $ln\ x$ | log(2.718282) bằng 1.0 |
| log10(x) | logarithm cơ số 10 của x | $log\ x$ | log10(100) bằng 2 |
| sin(x) | sin của x | $sin\ x$ | sin(0.0) bằng 0.0 |
| cos(x) | cosin của x | $cos\ x$ | cos(0.0) bằng 1.0 |
| tan(x) | tang của x | $tg\ x$ | tan(0.0) bằng 0.0 |
| ceil(x) | phần nguyên già của x, tức là số nguyên nhỏ nhất không nhỏ hơn x | $\lceil x \rceil$ | ceil(2.5) bằng 3<br>ceil(-2.5) bằng –2 |
| floor(x) | phần nguyên non của x, tức là số nguyên lớn nhất không lớn hơn x | $\lfloor x \rfloor$ | floor(2.5) bằng 2<br>floor(-2.5) bằng –3 |

Khái niệm hàm sẽ được đề cập kĩ hơn trong mục III.6.

### III.1.2.8. Biểu thức

Biểu thức là sự ghép nối các toán tử (*operator*) và các toán hạng (*operand*) theo một quy tắc xác định.

Các toán hạng trong biểu thức có thể là biến, hằng, hàm hoặc một biểu thức khác. Bản thân một biến, hằng, hàm đứng độc lập cũng được coi là một biểu thức.

Các toán tử trong biểu thức rất đa dạng như cộng, trừ, nhân, chia, so sánh…

Biểu thức thường là sự thể hiện công thức tính toán giá trị một đại lượng nào đó.

Ví dụ về biểu thức:

chieu_dai * chieu_rong * chieu_cao

Trong biểu thức trên, chieu_dai, chieu_rong, chieu_cao là các biến hoặc hằng, * là kí hiệu của toán tử nhân. Nếu chieu_dai, chieu_rong, chieu_cao là các biến

(hoặc hằng) lưu trữ giá trị chiều dài, chiều rộng và chiều cao của một khối hộp chữ nhật thì biểu thức trên sẽ tính giá trị thể tích của khối hộp chữ nhật đó.

Vấn đề biểu thức sẽ tiếp tục được đề cập trong mục III.2.

### III.1.2.9. Câu lệnh

Câu lệnh (*statement*) diễn tả một hoặc một nhóm các thao tác trong giải thuật. Chương trình được tạo thành từ dãy các câu lệnh.

Cuối mỗi câu lệnh đều có dấu chấm phẩy ';' để đánh dấu kết thúc câu lệnh cũng như để phân tách các câu lệnh với nhau.

Câu lệnh được chia thành 2 nhóm chính:

- Nhóm các câu lệnh đơn: Là những câu lệnh không chứa câu lệnh khác. Ví dụ: phép gán, phép cộng, phép trừ...
- Nhóm các câu lệnh phức: Là những câu lệnh chứa câu lệnh khác trong nó. Ví dụ: lệnh khối, các cấu trúc lệnh rẽ nhánh, cấu trúc lệnh lặp...

Lệnh khối là một số các lệnh đơn được nhóm lại với nhau và đặt trong cặp dấu ngoặc nhọn { } để phân tách với các lệnh khác trong chương trình.

### III.1.2.10. Chú thích

Để giúp việc đọc và hiểu chương trình viết ra được dễ dàng hơn, chúng ta cần đưa vào các lời chú thích (*comment*). Lời chú thích là lời mô tả, giải thích vắn tắt cho một câu lệnh, một đoạn chương trình hoặc cả chương trình, nhờ đó người đọc có thể hiểu được ý đồ của người lập trình và công việc mà chương trình đang thực hiện.

Lời chú thích chỉ có tác dụng duy nhất là giúp chương trình viết ra dễ đọc và dễ hiểu hơn, nó không phải là câu lệnh và nó hoàn toàn không ảnh hưởng gì đến hoạt động của chương trình.

Khi gặp kí hiệu lời chú thích trong chương trình, trình biên dịch sẽ tự động bỏ qua không dịch phần nội dung nằm trong phạm vi của vùng chú thích đó.

Trong C, có 2 cách để viết lời chú thích:

- Dùng 2 dấu sổ chéo liên tiếp // để kí hiệu toàn bộ vùng bắt đầu từ 2 dấu sổ chéo liên tiếp đó đến cuối dòng là vùng chú thích. Ví dụ:

  // khai bao 2 bien nguyen

```
int a, b;
a = 5; b = 3;    // khoi tao gia tri cho cac bien nay
```

Cách này thường dùng nếu đoạn chú thích ngắn, có thể viết đủ trên một dòng.

- Dùng 2 cặp kí hiệu /* và */ để kí hiệu rằng toàn bộ vùng bắt đầu từ cặp kí hiệu /* kéo dài đến cặp kí hiệu */ là vùng chú thích. Ví dụ:

```
/* doan chuong trinh sau khai bao 2 bien nguyen va khoi tao gia tri cho 2
bien nguyen nay */
int a, b;
a = 5; b = 3;
```

Cách này thường dùng khi đoạn chú thích dài, phải viết trên nhiều dòng.

## III.1.3. Cấu trúc cơ bản của một chương trình C

Về cơ bản, mọi chương trình viết bằng ngôn ngữ C sẽ có cấu trúc gồm 6 phần có thứ tự như sau:

| |
|---|
| Khai báo tệp tiêu đề<br>#include |
| Định nghĩa kiểu dữ liệu<br>typedef ... |
| Khai báo các hàm nguyên mẫu |
| Khai báo các biến toàn cục |
| Định nghĩa hàm **main()**<br>main()<br>{<br>...<br>} |
| Định nghĩa các hàm đã khai báo nguyên mẫu |

***Hình III.1. Cấu trúc cơ bản của một chương trình C***

- Phần 1: Phần khai báo các tệp tiêu đề. Phần này có chức năng thông báo cho chương trình dịch biết là chương trình có sử dụng những thư viện nào (mỗi tệp tiêu đề tương ứng với một thư viện).
- Phần 2: Định nghĩa các kiểu dữ liệu mới dùng cho cả chương trình.
- Phần 3: Phần khai báo các hàm nguyên mẫu. Phần này giúp cho chương trình dịch biết được những thông tin cơ bản (gồm tên hàm, danh sách các tham số và kiểu dữ liệu trả về) của các hàm sử dụng trong chương trình.
- Phần 4: Phần khai báo các biến toàn cục.
- Phần 5: Phần định nghĩa hàm **main( )**. Hàm **main( )** là một hàm đặc biệt trong C. Khi thực hiện, chương trình sẽ gọi hàm **main( )**, hay nói cách khác chương trình sẽ bắt đầu bằng việc thực hiện các lệnh trong hàm **main( )**. Trong hàm **main( )** ta mới gọi tới các hàm khác.
- Phần 6: Phần định nghĩa các hàm đã khai báo nguyên mẫu. Ở phần 3 ta đã khai báo nguyên mẫu (*prototype*) của các hàm, trong đó chỉ giới thiệu các thông tin cơ bản về hàm như tên hàm, danh sách các tham số và kiểu dữ liệu trả về. Nguyên mẫu hàm không cho ta biết cách thức cài đặt và hoạt động của các hàm. Ta sẽ làm việc đó ở phần định nghĩa các hàm.

Trong 6 phần trên, thì phần 5 định nghĩa hàm **main( )** bắt buộc phải có trong mọi chương trình C. Các phần khác có thể có hoặc không.

Dưới đây là ví dụ một chương trình viết trên ngôn ngữ C:

```
1. // Chuong trinh sau se nhap vao tu ban phim 2 so nguyen
2. // va hien thi ra man hinh tong, hieu tich cua 2 so nguyen vua nhap vao
3. #include <stdio.h>
4. #include <conio.h>
5. void main()
6. {
7.     // khai bao cac bien trong chuong trinh
8.     int a, b
9.     int tong, hieu, tich;
10.    // Nhap vao tu ban phim 2 so nguyen
11.    printf("\n Nhap vao so nguyen thu nhat: ");
```

```
12.     scanf("%d",&a);
13.     printf("\n Nhap vao so nguyen thu hai: ");
14.     scanf("%d",&b);
15.     // Tinh tong, hieu, tich cua 2 so vua nhap
16.     tong = a+b;
17.     hieu = a – b;
18.     tich = a*b;
19.     // Hien thi cac gia tri ra man hinh
20.     printf("\n Tong cua 2 so vua nhap la %d", tong);
21.     printf("\n Hieu cua 2 so vua nhap la %d", hieu);
22.     printf("\n Tich cua 2 so vua nhap la %d", tich);
23.     // Cho nguoi su dung an phim bat ki de ket thuc
24.     getch();
25. }
```

Trong chương trình trên chỉ có 2 phần là khai báo các thư viện và định nghĩa hàm **main( )**. Các phần khai báo hàm nguyên mẫu, khai báo biến toàn cục và định nghĩa hàm nguyên mẫu không có trong chương trình này.

Các dòng 1, 2 là các dòng chú thích mô tả khái quát công việc chương trình sẽ thực hiện.

Dòng thứ 3 và thứ 4 là khai báo các tệp tiêu đề. Bởi vì trong chương trình ta sử dụng các hàm printf() (nằm trong thư viện **stdio** – *standard input/output*, thư viện chứa các hàm thực hiện các thao tác vào ra chuẩn) và getch() (nằm trong thư viện **conio** – *console input/output*, thư viện chứa các hàm thực hiện các thao tác vào ra qua bàn phím, màn hình…) nên ta phải khai báo với chương trình dịch gộp các thư viện đó vào chương trình. Nếu ta không gộp thư viện vào trong chương trình thì ta sẽ không thể sử dụng các hàm có trong thư viện đó.

Để gộp một thư viện vào trong chương trình (nhờ đó ta có thể sử dụng các hàm của thư viện đó), ta khai báo tệp tiêu đề tương ứng với thư viện đó ở đầu chương trình bằng chỉ thị có mẫu sau:

```
#include <tên_tệp_tiêu_đề>
```

Ví dụ: Để gộp thư viện conio vào chương trình ta dùng chỉ thị:

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

Lưu ý: Các tệp tiêu đề có tên là tên của thư viện, có phần mở rộng là .h (viết tắt của từ *header*).

Các dòng tiếp theo (từ dòng thứ 5 đến dòng thứ 25) là phần định nghĩa hàm **main()**, trong đó các dòng 7, 10, 15, 19, 23 là các dòng chú thích mô tả công việc mà các câu lệnh sau đó sẽ thực hiện.

## III.1.4. Biên dịch chương trình C

### III.1.4.1. Trình biên dịch Turbo C++

Một chương trình sau khi viết ra phải được biên dịch thành mã máy (tức là chuyển các câu lệnh viết bằng ngôn ngữ lập trình thành các câu lệnh tương ứng của ngôn ngữ máy) thì mới có thể thực thi được. Công việc biên dịch được thực hiện bởi chương trình biên dịch (*compiler*).

Hiện có nhiều chương trình biên dịch ngôn ngữ C khác nhau như Turbo C++ của hãng Borland Inc, MSC của Microsoft Corp, GCC do GNU project phát triển, hay Lattice C của Lattice, Dev-C++ của Bloodshed Software... Tuy vậy đối với đa phần người bắt đầu học C ở Việt Nam thì Turbo C++ là trình biên dịch ngôn ngữ C quen thuộc.

Lưu ý: Turbo C++ có khả năng biên dịch chương trình viết bằng cả ngôn ngữ C và C++.

Turbo C++ cũng có nhiều phiên bản khác nhau, nhưng để bắt đầu học và thực hành C thì Turbo C++ 3.0 là thích hợp vì nó có đặc điểm là gọn nhẹ, đủ tính năng và dễ sử dụng.

### III.1.4.2. Cài đặt và sử dụng Turbo C++ 3.0

Để sử dụng Turbo C++ 3.0 ta cần phải cài đặt nó lên máy. Quá trình cài đặt thực hiện theo các bước sau:

B1: Chuẩn bị đĩa chứa bộ cài của Turbo C++ 3.0, kích thước của bộ cài khoảng 4 MB. Hãy copy bộ cài này vào máy của bạn, giả sử vào thư mục *C:\TC_Setup*.

B2: Tìm đến thư mục chứa bộ cài Turbo C++ 3.0 (như giả sử ở trên là *C:\TC_Setup*) và kích hoạt file *INSTALL.EXE* để chạy chương trình cài đặt Turbo C++ 3.0. Chương trình cài đặt Turbo C++ 3.0 ban đầu sẽ yêu cầu bạn chỉ ra ổ đĩa trên đó chứa bộ cài Turbo C++ 3.0.

Enter the SOURCE drive to use:

Hãy nhập vào tên ổ đĩa, chẳng hạn C (ta để bộ cài Turbo C++ 3.0 ở thư mục *C:\TC_Setup*).

B3: Sau đó chương trình yêu cầu nhập vào đường dẫn tới thư mục chứa các file của Turbo C++ 3.0.

Enter the SOURCE Path:

Thông thường chương trình sẽ tự tìm, chỉ cần ấn Enter để chuyển sang bước tiếp theo.

B4: Ở bước 4, cần xác định thư mục cài đặt. Thư mục này sẽ chứa các file của Turbo C++ 3.0 để sử dụng sau này.

Directories... [C:\TC]
Option... [IDE CMD LIB CLASS BGI HELP EXMPL]
Start Installation

Thư mục cài đặt mặc định sẽ là *\TC* nằm trên thư mục gốc của ổ đĩa chứa bộ cài. Nếu muốn thay đổi thư mục cài đặt thì hãy dùng các phím ↑ và ↓ để di chuyển hộp sáng đến phần Directories, gõ Enter và nhập vào đường dẫn mới, sau đó ấn phím Esc để trở về.

Dùng các phím ↑ và ↓ để di chuyển hộp sáng đến phần Start Installation và ấn Enter. Chương trình sẽ tự động thực hiện và hoàn tất quá trình cài đặt.

*Lưu ý*: Có thể copy toàn bộ thư mục đã cài đặt của Turbo C++ 3.0 về máy và sử dụng, nhưng phải cho Turbo C++ biết đường dẫn tới các tệp tiêu đề và các tệp thư viện bằng cách vào menu *Option*, chọn *Directories*.

### III.1.4.3. Sử dụng môi trường Turbo C++ 3.0

Sau khi cài đặt xong, tìm đến thư mục *BIN* trong thư mục cài đặt và chạy file *TC.EXE* để khởi động Turbo C++ 3.0.

Sau khi khởi động Turbo C++ 3.0, sẽ xuất hiện màn hình làm việc của Turbo C++ 3.0.

Bạn dùng chuột di chuyển đến menu File (hoặc ấn Alt-F), sau đó chọn mục New để mở cửa sổ soạn thảo mới.

Giờ hãy gõ vào toàn bộ chương trình viết bằng ngôn ngữ C của bạn lên cửa sổ soạn thảo này. Ấn phím F2 để lưu trữ tệp chương trình nguồn trên máy, một cửa sổ cất giữa tệp sẽ hiện ra yêu cầu bạn nhập vào tên mới cho tệp chương trình nguồn (tên mặc định sẽ là *NONAME.CPP*). Hãy đặt một tên cho tệp rồi chọn OK để lưu tệp chương trình nguồn lại.

Cuối cùng là ấn phím F9 để biên dịch chương trình viết ra. Nếu chương trình có lỗi thì Turbo C++ 3.0 sẽ báo lỗi và bạn phải sửa lại đến khi không còn lỗi. Nếu chương trình không có lỗi thì Turbo C++ 3.0 sẽ thông báo biên dịch thành công và bạn có thể ấn Ctrl-F9 để chạy chương trình đã biên dịch.

Với chương trình ví dụ ở phần trước, sau khi biên dịch và thực hiện bạn sẽ thấy trên màn hình kết quả như sau:

```
Nhap vao so nguyen thu nhat: 523
Nhap vao so nguyen thu hai: 257

Tong cua 2 so vua nhap la 780
Hieu cua 2 so vua nhap la 266
Tich cua 2 so vua nhap la 134411
```

## III.2. Kiểu dữ liệu và biểu thức trong C

### III.2.1. Các kiểu dữ liệu chuẩn trong C

| Kiểu dữ liệu | Ý nghĩa | Kích thước | Miền biểu diễn |
|---|---|---|---|
| **unsigned char** | Kí tự | 1 byte | 0 ÷ 255 |
| **char** | Kí tự | 1 byte | -128 ÷ 127 |
| **unsigned int** | Số nguyên không dấu | 2 byte | 0 ÷ 65,535 |
| **short int** | Số nguyên có dấu | 2 byte | -32,768 ÷ 32,767 |
| **int** | Số nguyên có dấu | 2 byte | - 32,768 ÷ 32,767 |
| **unsigned long** | Số nguyên không dấu | 4 byte | 0 ÷ 4,294,967,295 |
| **long** | Số nguyên có dấu | 4 byte | -2,147,483,648 ÷ 2,147,483,647 |
| **float** | Số thực dấu phẩy động, độ chính xác đơn | 4 byte | ± 3.4E-38 ÷ ± 3.4E+38 |
| **double** | Số thực dấu phẩy động độ chính xác kép | 8 byte | ± 1.7E-308 ÷ ± 1.7E+308 |
| **long double** | Số thực dấu phẩy động | 10 byte | ± 3.4E-4932 ÷ ± 1.1E+4932 |

**Khai báo và sử dụng biến, hằng**

*Khai báo và sử dụng biến*

Một biến trước khi sử dụng phải được khai báo. Cú pháp khai báo:

***kiểu_dữ_liệu  tên_biến;***

Ví dụ:

```
float x;     // biến kiểu thực
float y;     // biến kiểu thực
double z;    // biến kiểu thực
int i;       // biến kiểu nguyên
int j;       // biến kiểu nguyên
```

Nếu các biến thuộc cùng kiểu dữ liệu thì C cho phép khai báo chúng trên cùng một dòng, ngăn cách với nhau bởi dấu phẩy:

***kiểu_dữ_liệu  danh_sách_tên_biến;***

Ví dụ:

```
float x, y;
int i, j;
```

Sau khi khai báo, biến có thể nhận giá trị thuộc kiểu dữ liệu đã khai báo. Chúng ta có thể khởi tạo giá trị đầu cho biến nếu muốn với cú pháp:

***kiểu_dữ_liệu  tên_biến = giá_trị_đầu;***

Ví dụ:

```
int a = 3;    // sau lenh nay bien a se co gia tri bang 3
float x = 5.0, y = 2.6;// sau lenh nay x co gia tri 5.0, y co gia tri 2.6
```

Biến dùng để lưu giữ giá trị, dùng làm toán hạng trong biểu thức, làm tham số cho hàm, làm biến chỉ số cho các cấu trúc lặp (**for**, **while**, **do while**), làm biểu thức điều kiện trong các cấu trúc rẽ nhánh (**if**, **switch**)...

*Khai báo hằng*

Có 2 cách để khai báo hằng trong C là dùng chỉ thị **#define** hoặc khai báo với từ khóa **const**.

Dùng chỉ thị **#define**

Cú pháp khai báo:

**#define tên_hằng giá_trị**

Lưu ý: Không có dấu chấm phẩy ở cuối dòng chỉ thị.

Với cách khai báo này, mỗi khi chương trình dịch gặp tên_hằng trong chương trình, nó sẽ tự động thay thế bằng giá_trị. Ở đây kiểu dữ liệu của hằng tự động được chương trình dịch xác định dựa theo nội dung của giá_trị.

*Ví dụ*:

```
#define MAX_SINH_VIEN  50                // hằng kiểu số nguyên
#define CNTT "Cong nghe thong tin"       // hằng kiểu xâu kí tự (string)
#define DIEM_CHUAN  23.5                 // hằng kiểu số thực
```

Dùng từ khóa **const** để khai báo với cú pháp:

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

Khai báo này giống với khai báo biến có khởi tạo giá trị đầu, tuy nhiên cần lưu ý:

- Do có từ khóa **const** ở đầu nên giá trị của đối tượng tên_hằng sẽ không được phép thay đổi trong chương trình. Những lệnh nhằm làm thay đổi giá trị của tên_hằng trong chương trình sẽ dẫn tới lỗi biên dịch.
- Trong khai báo biến thông thường, người lập trình có thể khởi tạo giá trị cho biến ngay từ khi khai báo hoặc không khởi tạo cũng được. Nhưng trong khai báo hằng, giá trị của tất cả các hằng cần được xác định ngay trong lệnh khai báo.

Các khai báo hằng ở ví dụ trước giờ có thể viết lại theo cách khác như sau:

```
const int  MAX_SINH_VIEN  = 50;
const char  CNTT[20] = "Cong nghe thong tin";
const float  DIEM_CHUAN = 23.5;
```

## III.2.2. Các biểu thức

*Biểu thức số học*

Biểu thức số học là biểu thức mà giá trị của nó là các đại lượng số học (số nguyên, số thực).

Trong biểu thức số học, các toán tử là các phép toán số học (cộng, trừ, nhân, chia…), các toán hạng là các đại lượng số học. Ví dụ (giả sử a, b, c là các số thực):

3 * 3.7, 8 + 6/3, a + b – c…

*Biểu thức logic*

Biểu thức logic là biểu thức mà giá trị của nó là các giá trị logic, tức là một trong hai giá trị: Đúng (*TRUE*) hoặc Sai (*FALSE*).

Ngôn ngữ C coi các giá trị nguyên khác 0 (ví dụ 1, -2, -5) là giá trị logic Đúng (TRUE), giá trị 0 là giá trị logic Sai (FALSE).

*Các phép toán logic gồm có*

- AND (VÀ logic, trong ngôn ngữ C được kí hiệu là &&)
- OR (HOẶC logic, trong ngôn ngữ C được kí hiệu là ||)
- NOT (PHỦ ĐỊNH, trong ngôn ngữ C kí hiệu là !)

*Biểu thức quan hệ*

Biểu thức quan hệ là những biểu thức trong đó có sử dụng các toán tử quan hệ so sánh như lớn hơn, nhỏ hơn, bằng nhau, khác nhau... Biểu thức quan hệ cũng chỉ có thể nhận giá trị là một trong 2 giá trị Đúng (TRUE) hoặc Sai (FALSE). Ví dụ về biểu thức quan hệ:

| | |
|---|---|
| 5 > 7 | // có giá trị logic là sai, FALSE |
| 9 != 10 | // có giá trị logic là đúng, TRUE |
| 2 >= 2 | // có giá trị logic là đúng, TRUE |
| a > b | // giả sử a, b là 2 biến kiểu int |
| a+1 > a | // có giá trị đúng, TRUE |

Ví dụ về biểu thức logic:

| | |
|---|---|
| (5 > 7)&&(9!=10) | // có giá trị logic là sai, FALSE |
| 0 \|\| 1 | // có giá trị logic là đúng, TRUE |
| (5 > 7)\|\|(9!=10) | // có giá trị logic là đúng, TRUE |
| 0 | // có giá trị logic là sai, FALSE |
| !0 | // phủ định của 0, có giá trị logic là đúng, TRUE |
| 3 | // có giá trị logic là đúng, TRUE |
| !3 | // phủ định của 3, có giá trị logic là sai, FALSE |
| (a > b)&&( a < b) | // Có giá trị sai, FALSE. Giả sử a, b là 2 biến kiểu int |

*Sử dụng biểu thức*

Trong chương trình, biểu thức được sử dụng cho các mục đích sau:

- Làm vế phải của lệnh gán (sẽ đề cập ở mục sau).
- Làm toán hạng trong các biểu thức khác.
- Làm tham số thực trong lời gọi hàm.
- Làm chỉ số trong các cấu trúc lặp **for**, **while**, **do while**.
- Làm biểu thức kiểm tra trong các cấu trúc rẽ nhánh **if**, **switch**.

## III.2.3. Các phép toán

Các phép toán trong C được chia thành các nhóm phép toán cơ bản sau: nhóm các phép toán số học, nhóm các phép toán thao tác trên bit, nhóm các phép toán quan hệ, nhóm các phép toán logic. Ngoài ra C còn cung cấp một số phép toán khác nữa như phép gán, phép lấy địa chỉ...

### III.2.3.1. Phép toán số học

Các phép toán số học (*Arithmetic operators*) gồm có:

| Toán tử | Ý nghĩa | Kiểu dữ liệu của toán hạng | Ví dụ |
|---|---|---|---|
| - | Phép đổi dấu | Số thực hoặc số nguyên | int a, b;<br>-12; -a; -25.6; |
| + | Phép toán cộng | Số thực hoặc số nguyên | float x, y;<br>5 + 8; a + x;<br>3.6 + 2.9; |
| - | Phép toán trừ | Số thực hoặc số nguyên | 3 – 1.6; a – 5; |
| * | Phép toán nhân | Số thực hoặc số nguyên | a * b; b * y;<br>2.6 * 1.7; |
| / | Phép toán chia | Số thực hoặc số nguyên | 10.0/3.0; (bằng 3.33...)<br>10/3.0; (bằng 3.33...)<br>10.0/3; (bằng 3.33...) |

| | | | |
|---|---|---|---|
| / | Phép chia lấy phần nguyên | Giữa 2 số nguyên | 10/3; (bằng 3) |
| % | Phép chia lấy phần dư | Giữa 2 số nguyên | 10%3; (bằng 1) |

*Các phép toán trên bit*

| Toán tử | Ý nghĩa | Kiểu dữ liệu của toán hạng | Ví dụ |
|---|---|---|---|
| & | Phép VÀ nhị phân | 2 số nhị phân | 0 & 0 (có giá trị 0)<br>0 & 1 (có giá trị 0)<br>1 & 0 (có giá trị 0)<br>1 & 1 (có giá trị 1)<br>101 & 110 (có giá trị 100) |
| \| | Phép HOẶC nhị phân | 2 số nhị phân | 0 \| 0 (có giá trị 0)<br>0 \| 1 (có giá trị 0)<br>1 \| 0 (có giá trị 0)<br>1 \| 1 (có giá trị 1)<br>101 \| 110 (có giá trị 111) |
| ^ | Phép HOẶC CÓ LOẠI TRỪ nhị phân | 2 số nhị phân | 0 ^ 0 (có giá trị 0)<br>0 ^1 (có giá trị 1)<br>1 ^ 0 (có giá trị 1)<br>1 ^ 1 (có giá trị 0)<br>101 ^ 110 (có giá trị 011) |
| << | Phép DỊCH TRÁI nhị phân | Số nhị phân | a << n (có giá trị $a*2^n$)<br>101 << 2 (có giá trị 10100) |
| >> | Phép DỊCH PHẢI nhị phân | Số nhị phân | a >> n (có giá trị $a/2^n$)<br>101 >> 2 (có giá trị 1) |
| ~ | Phép ĐẢO BIT nhị phân (lấy Bù 1) | Số nhị phân | ~ 0 (có giá trị 1)<br>~ 1 (có giá trị 0)<br>~ 110 (có giá trị 001) |

### III.2.3.2. Phép toán quan hệ

Các phép toán quan hệ (*Comparison operators*) gồm có:

| Toán tử | Ý nghĩa | Ví dụ |
|---|---|---|
| > | So sánh lớn hơn giữa 2 số nguyên hoặc thực. | 2 > 3 (có giá trị 0)<br>6 > 4 (có giá trị 1)<br>a > b |
| >= | So sánh lớn hơn hoặc bằng giữa 2 số nguyên hoặc thực. | 6 >= 4 (có giá trị 1)<br>x >= a |
| < | So sánh nhỏ hơn giữa 2 số nguyên hoặc thực. | 5 < 3 (có giá trị 0), |
| <= | So sánh nhỏ hơn hoặc bằng giữa 2 số nguyên hoặc thực. | 5 <= 5 (có giá trị 1)<br>2 <= 9 (có giá trị 1) |
| == | So sánh bằng nhau giữa 2 số nguyên hoặc thực. | 3 == 4 (có giá trị 0)<br>a == b |
| != | So sánh không bằng (so sánh khác) giữa 2 số nguyên hoặc thực. | 5 != 6 (có giá trị 1)<br>6 != 6 (có giá trị 0) |

### III.2.3.3. Các phép toán logic

Các phép toán logic (*Logical operators*) gồm có:

| Toán tử | Ý nghĩa | Kiểu dữ liệu của toán hạng | Ví dụ |
|---|---|---|---|
| && | Phép VÀ LOGIC. Biểu thức VÀ LOGIC bằng 1 khi và chỉ khi cả 2 toán hạng đều bằng 1 | Hai biểu thức logic | 3<5 && 4<6 (có giá trị 1)<br>2<1 && 2<3 (có giá trị 0)<br>a > b && c < d |
| \|\| | Phép HOẶC LOGIC. Biểu thức HOẶC LOGIC bằng 0 khi và chỉ khi cả 2 toán hạng bằng 0. | Hai biểu thức logic | 6 \|\| 0 (có giá trị 1)<br>3<2 \|\| 3<3 (có giá trị 1)<br>x >= a \|\| x == 0 |
| ! | Phép PHỦ ĐỊNH LOGIC một ngôi. Biểu thức PHỦ ĐỊNH LOGIC có giá trị bằng 1 nếu toán hạng bằng 0 và có giá trị bằng 0 nếu toán hạng bằng 1 | Biểu thức logic | !3 (có giá trị 0)<br>!(2>5) (có giá trị 1) |

### III.2.3.4. Phép toán gán

Phép toán gán có dạng:

***tên_biến = biểu_thức;***

Phép toán gán có chức năng lấy giá trị của biểu_thức gán cho tên_biến. Dấu = là kí hiệu cho toán tử gán.

Ví dụ:

```
int a, b, c;
a = 3;
b = a + 5;
c = a * b;
```

Sau đoạn lệnh trên, biến a có giá trị là 3, b có giá trị là 8 và c có giá trị là 24.

Trong phép toán gán nếu ta bỏ dấu ; ở cuối đi thì ta sẽ thu được biểu thức gán. Biểu thức gán là biểu thức có dạng:

***tên_biến = biểu_thức***

Biểu thức gán là biểu thức nên cũng có giá trị. Giá trị của biểu thức gán bằng giá trị của biểu_thức, do đó ta có thể gán giá trị của biểu thức gán cho một biến khác hoặc sử dụng như một biểu thức bình thường. Ví dụ:

```
int a, b, c;
a = b = 2007;
c = (a = 20) * (b = 30);
```

Trong câu lệnh thứ 2, ta đã gán giá trị 2007 cho biến *b*, sau đó gán giá trị của biểu thức *b* = 2007 cho biến a. Giá trị của biểu thức *b* = 2007 là 2007, do đó kết thúc câu lệnh này ta có *a* bằng 2007, *b* bằng 2007.

Trong câu lệnh thứ 3, ta gán giá trị 20 cho a, gán giá trị 30 cho b. Sau đó ta tính giá trị của biểu thức tích (a = 20) * (b = 30) từ giá trị các biểu thức con *a* = 20 (có giá trị là 20) và *b* = 30 (có giá trị là 30). Cuối cùng ta gán giá trị của biểu thức tích thu được (600) cho biến *c*.

*Phép toán gán thu gọn*

Xét lệnh gán sau:

```
x = x + y;
```

Lệnh gán này sẽ tăng giá trị của biến *x* thêm một lượng có giá trị bằng giá trị của *y*. Trong C ta có thể viết lại lệnh này một cách gọn hơn mà thu được kết quả tương đương:

```
x += y;
```

Dạng lệnh gán thu gọn này còn áp dụng được với các phép toán khác nữa.

| **Lệnh gán thông thường** | **Lệnh gán thu gọn** |
|---|---|
| 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 |

## III.2.4. Thứ tự ưu tiên các phép toán

*Khái niệm thứ tự ưu tiên (operator precedence) của phép toán*

Thứ tự ưu tiên của các phép toán dùng để xác định trật tự kết hợp các toán hạng với các toán tử khi tính toán giá trị của biểu thức.

*Bảng thứ tự ưu tiên của các phép toán trong C*

| Mức | Các toán tử | Trật tự kết hợp |
|---|---|---|
| 1 | () [] . -> ++ (hậu tố) – (hậu tố) | ——> |
| 2 | ! ~ ++ (tiền tố) -- (tiền tố) - * & sizeof | <—— |
| 3 | * / % | ——> |
| 4 | + - | ——> |
| 5 | << >> | ——> |
| 6 | < <= > >= | ——> |
| 7 | == != | ——> |
| 8 | & | ——> |
| 9 | ^ | ——> |
| 10 | \| | ——> |
| 11 | && | ——> |
| 12 | \|\| | ——> |
| 13 | ?: | <—— |
| 14 | = += –= | <—— |

*Ghi chú*: ——> Trật tự kết hợp từ trái qua phải.

<—— Trật tự kết hợp từ phải qua trái.

*Nguyên tắc xác định trật tự thực hiện các phép toán*

i. Biểu thức con trong ngoặc được tính toán trước các phép toán khác.

ii. Phép toán một ngôi đứng bên trái toán hạng được kết hợp với toán hạng đi liền nó.

iii. Nếu toán hạng đứng cạnh hai toán tử thì có 2 khả năng là:

a. Nếu hai toán tử có độ ưu tiên khác nhau thì toán tử nào có độ ưu tiên cao hơn sẽ kết hợp với toán hạng.

b. Nếu hai toán tử cùng độ ưu tiên thì dựa vào trật tự kết hợp của các toán tử để xác định toán tử được kết hợp với toán hạng.

## III.2.5. Một số toán tử đặc trưng trong C

*Các phép toán tăng giảm một đơn vị*

Trong lập trình chúng ta thường xuyên gặp những câu lệnh tăng (hoặc giảm) giá trị của một biến thêm (đi) một đơn vị. Để làm điều đó chúng ta dùng lệnh sau:

***<tên biến> = <tên biến> + 1;***

***<tên biến> = <tên biến> - 1;***

Ta cũng có thể tăng (hoặc giảm) giá trị của một biến bằng cách sử dụng hai phép toán đặc biệt của C là phép toán ++ và phép toán --. Phép toán ++ sẽ tăng giá trị của biến thêm 1 đơn vị, phép toán -- sẽ giảm giá trị của biến đi 1 đơn vị. Ví dụ:

```
int a = 5;
float x = 10;
a ++;  // lệnh này tương đương với a = a + 1;
x --;  // tương đương với x = x – 1;
```

Phép toán tăng, giảm một đơn vị ở ví dụ trên là dạng hậu tố (vì phép toán đứng sau toán hạng). Ngoài ra còn có dạng tiền tố của phép toán tăng, giảm một đơn vị. Trong dạng tiền tố, ta thay đổi giá trị của biến trước khi sử dụng biến đó để tính toán giá trị của biểu thức. Trong dạng hậu tố, ta tính toán giá trị của biểu thức bằng giá trị ban đầu của biến, sau đó mới thay đổi giá trị của biến.

Ví dụ:

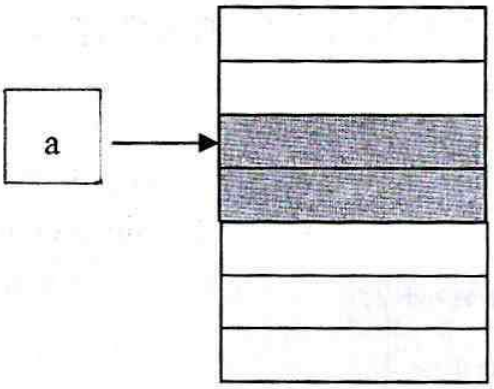
```
int a, b, c;
a = 3;        // a bằng 3;
b = a++;      // dạng hậu tố. b bằng 3; a bằng 4;
c = ++b;      // dạng tiền tố. b bằng 4, c bằng 4;
```

Sau khi thực hiện đoạn chương trình trên, ta có *a*, *b* và *c* đều có giá trị bằng 4.

*Phép toán lấy địa chỉ biến* (&)

Một biến thực chất là một vùng nhớ được đặt tên (là tên của biến) trên bộ nhớ của máy tính. Mọi ô nhớ trên bộ nhớ máy tính đều được đánh địa chỉ. Do đó mọi biến đều có địa chỉ.

Địa chỉ của một biến được định nghĩa là địa chỉ của ô nhớ đầu tiên trong vùng nhớ dành cho biến đó. Hình III.2 minh họa một biến tên là *a*, kiểu dữ liệu **int** được lưu trữ trong bộ nhớ tại 2 ô nhớ. Khi đó địa chỉ của biến *a* sẽ là địa chỉ ô nhớ thứ nhất trong hai ô nhớ này.




***Hình III.2. Địa chỉ của biến***

Trong C để xác định địa chỉ của một biến ta sử dụng toán tử một ngôi & đặt trước tên biến, cú pháp là:

***& <tên biến>;***

Ví dụ: &a;

*Phép toán chuyển đổi kiểu bắt buộc*

Chuyển đổi kiểu là chuyển kiểu dữ liệu của một biến từ kiểu dữ liệu này sang kiểu dữ liệu khác. Cú pháp của lệnh chuyển kiểu dữ liệu là như sau:

***(<kiểu dữ liệu mới>) <biểu thức>;***

Có những sự chuyển đổi được thực hiện hết sức tự nhiên, không có khó khăn gì, thậm chí đôi khi chương trình dịch sẽ tự động chuyển đổi kiểu hộ cho ta, ví dụ chuyển một dữ liệu kiểu số nguyên **int** sang một số nguyên kiểu **long int**, hay từ một số **long int** sang một số thực **float**... Đó là vì một số nguyên kiểu **int** thực ra cũng là một số nguyên kiểu **long int**, một số nguyên kiểu **long int** cũng chính là một số thực kiểu **float**, một số thực kiểu **float** cũng là một số thực kiểu **double**.

Tuy nhiên điều ngược lại thì chưa chắc, ví dụ số nguyên **long int** 50,000 không phải là một số nguyên kiểu **int** vì phạm vi biểu diễn của kiểu **int** là từ (-32,768 đến 32,767). Khi đó nếu phải chuyển kiểu dữ liệu thì ta phải cẩn thận nếu không sẽ bị mất dữ liệu. Ví dụ, một số thực **float** khi chuyển sang kiểu số nguyên **int** sẽ bị loại bỏ phần thập phân, còn một số nguyên kiểu **long int** khi chuyển sang kiểu **int** sẽ nhiều khả năng thu được một giá trị xa lạ. Ví dụ: 1,193,046 (0x123456) là một số kiểu **long int**, khi chuyển sang kiểu **int** sẽ thu được 13,398 (0x3456). Đoạn chương trình sau minh họa điều đó:

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

```
void main()
{
    long int li;
    int i;
    float f;
    clrscr();
    li = 0x123456;
    f = 123.456;
    i = (int) li;
    printf("\n li = %ld",li);
    printf("\n i  = %d",i);
    i = (int) f;
    printf("\n f  = %f",f);
    printf("\n i  = %d",i);
    getch();
}
```

Kết quả:

```
li = 1193046
i = 13398
f = 123.456001
i = 123
```

C hỗ trợ chuyển kiểu tự động trong những trường hợp sau:

**char → int → long int → float → double → long double**

Khả năng chuyển kiểu dữ liệu là một điểm mạnh của C, tạo sự linh hoạt cho biến trong chương trình. Tuy nhiên nếu các bạn mới bắt đầu lập trình thì không nên chuyển kiểu dữ liệu trong chương trình vì có thể sinh ra những kết quả không như ý nếu bạn chưa có kinh nghiệm. Ngoài ra hình dung trước kiểu dữ liệu cho một biến và duy trì kiểu dữ liệu đó trong suốt chương trình cũng là một thói quen lập trình tốt.

*Biểu thức điều kiện*

Là biểu thức có dạng:

***biểu_thức_1 ? biểu_thức_2 : biểu_thức_3***

Giá trị của biểu thức điều kiện sẽ là giá trị của biểu_thức_2 nếu biểu_thức_1 có giá trị khác 0 (tương ứng với giá trị logic TRUE), trái lại giá trị của biểu thức điều kiện sẽ là giá trị của biểu_thức_3 nếu biểu_thức_1 có giá trị bằng 0 (tương ứng với giá trị logic FALSE).

Ví dụ sau sẽ cho ta xác định được giá trị nhỏ nhất của 2 số nhờ sử dụng biểu thức điều kiện:

```
float x, y, z;          // khai báo biến
x = 3.8; y = 2.6;       // gán giá trị cho các biến x, y
z = (x<y) ? x : y;      // z sẽ có giá trị bằng giá trị
                        // nhỏ nhất trong 2 số x và y
```

*Lệnh dãy*

Lệnh dãy là lệnh gồm một dãy các biểu thức phân cách nhau bằng dấu phẩy và kết thúc lệnh là dấu chấm phẩy. Nó có dạng:

***biểu_thức_1, biểu_thức_2, ..., biểu_thức_n;***

Trong lệnh dãy các biểu thức được tính toán độc lập với nhau.

# III.3. Cấu trúc lập trình trong C

## III.3.1. Vào/ra

### III.3.1.1. Các lệnh vào ra dữ liệu với các biến (printf, scanf)

Để vào ra dữ liệu, C cung cấp 2 hàm vào ra cơ bản là printf() và scanf(). Muốn sử dụng 2 hàm printf() và scanf() ta cần khai báo tệp tiêu đề stdio.h.

*Hàm printf()*

Cú pháp sử dụng hàm printf ( ):

***printf(xâu_định_dạng [,danh_sách_tham_số]);***

Hàm printf() được dùng để hiển thị ra màn hình các loại dữ liệu cơ bản như số, kí tự và xâu kí tự cùng một số hiệu ứng hiển thị đặc biệt.

xâu_định_dạng là xâu điều khiển cách thức hiển thị dữ liệu trên thiết bị ra chuẩn (màn hình máy tính). Trong xâu_định_dạng có chứa:

- Các kí tự thông thường, chúng sẽ được hiển thị ra màn hình bình thường.
- Các nhóm kí tự định dạng dùng để xác định quy cách hiển thị các tham số trong phần danh_sách_tham_số.
- Các kí tự điều khiển dùng để tạo các hiệu ứng hiển thị đặc biệt như xuống dòng ('\n') hay sang trang ('\f')...

Phần danh_sách_tham_số là các giá trị biến, hằng, biểu thức mà ta muốn hiển thị ra màn hình và được ngăn cách với nhau bởi dấu phẩy. Nhóm kí tự định dạng thứ *k* trong xâu_định_dạng dùng để xác định quy cách hiển thị tham số thứ *k* trong danh_sách_tham_số. Do đó danh_sách_tham_số phải phù hợp về số lượng, thứ tự và kiểu với các nhóm kí tự định dạng trong xâu_định_dạng.

Ví dụ:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    int a = 5;
    float x = 1.234;
    printf("Hien thi mot so nguyen %d và mot so thuc %f",a,x);
    getch();
}
```

Kết quả:

```
Hien thi mot so nguyen 5 va mot so thuc 1.234000
```

Trong ví dụ trên "Hien thi mot so nguyen %d và mot so thuc %f" là xâu định dạng, còn a và x là các tham số của hàm printf(). Trong xâu định dạng trên có 2 nhóm kí tự định dạng là %d và %f, với %d dùng để báo cho máy biết rằng cần phải hiển thị tham số tương ứng (biến a) theo định dạng số nguyên và %f dùng để báo cho máy cần hiển thị tham số tương ứng (biến x) theo định dạng số thực.

Lưu ý là mỗi nhóm kí tự định dạng chỉ dùng cho một kiểu dữ liệu, còn một kiểu dữ liệu có thể hiển thị theo nhiều cách khác nhau nên có nhiều nhóm kí tự định dạng khác nhau. Nếu giữa nhóm kí tự định dạng và tham số tương ứng không phù hợp với nhau thì sẽ hiển thị ra kết quả không như ý. Phần sau đây giới thiệu một số nhóm kí tự định dạng hay dùng trong C và ý nghĩa của chúng.

| Nhóm kí tự định dạng | Áp dụng cho kiểu dữ liệu | Ghi chú |
|---|---|---|
| %d | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên có dấu hệ đếm thập phân. |
| %i | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên có dấu hệ đếm thập phân. |
| %o | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên không dấu trong hệ đếm cơ số 8. |
| %u | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên không dấu. |
| %x | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên hệ đếm 16 (không có 0x đứng trước), sử dụng các chữ cái a b c d e f. |
| %X | **int, long, char** | Hiển thị tham số tương ứng dưới dạng số nguyên hệ đếm 16 (không có 0x đứng trước), sử dụng các chữ cái A B C D E F. |
| %e | **float, double** | Hiển thị tham số tương ứng dưới dạng số thực dấu phẩy động. |
| %f | **float, double** | Hiển thị tham số tương ứng dưới dạng số thực dấu phẩy tĩnh. |
| %g | **float, double** | Hiển thị tham số tương ứng số thực dưới dạng ngắn gọn hơn trong 2 dạng dấu phẩy tĩnh và dấu phẩy động. |
| %c | **int, long, char** | Hiển thị tham số tương ứng dưới dạng kí tự. |
| %s | **char *** (xâu kí tự) | Hiển thị tham số tương ứng dưới dạng xâu kí tự. |

Để trình bày dữ liệu được đẹp hơn, C cho phép đưa thêm một số thuộc tính định dạng dữ liệu khác vào trong xâu định dạng như độ rộng tối thiểu, căn lề trái, căn lề phải.

**Độ rộng tối thiểu**

Thông thường khi hiển thị dữ liệu, C tự động xác định số chỗ cần thiết sao cho hiển thị vừa đủ nội dung dữ liệu.

Nếu ta muốn C hiển thị dữ liệu trên một số lượng vị trí xác định bất kể nội dung dữ liệu đó có điền đầy số chỗ được cung cấp hay không, ta có thể chèn một số nguyên vào trong nhóm kí tự định dạng, ngay sau dấu %.

Ví dụ khi hiển thị số nguyên:

```
a = 1234;
printf("\n%5d",a); // dành 5 chỗ để hiển thị số nguyên a
printf("\n%5d",34);// dành 5 chỗ để hiển thị số nguyên 34
```

Kết quả:

```
□1234
□□□34
```

*Ở đây □ kí hiệu thay cho dấu cách (space).*

Như vậy với nhóm kí tự định dạng **%md**, m dùng để báo số chỗ cần dành để hiển thị dữ liệu, còn d báo rằng hãy hiển thị dữ liệu đó dưới dạng một số nguyên. Tương tự với các nhóm kí tự định dạng **%mc** khi hiển thị kí tự và **%ms** khi hiển thị xâu kí tự.

Ví dụ:

```
printf("\n%3d %15s %3c", 1, "nguyen van a", 'g');
printf("\n%3d %15s %3c", 2, "tran van b", 'k');
```

Lưu ý: trong lệnh printf viết ở trên, giữa các tham số định dạng có ngăn cách nhau bởi một dấu cách (space).

Kết quả:

```
□□1□□□□□nguyen van a□□g
□□2□□□□□□tran van b□□k
```

Nếu nội dung dữ liệu không điền đầy số chỗ được cấp thì những chỗ không dùng đến sẽ được điền bởi dấu cách.

Khi số chỗ cần thiết để hiển thị nội dung dữ liệu lớn hơn m thì C tự động cung cấp thêm chỗ mới để hiển thị chứ không cắt bớt nội dung của dữ liệu để cho vừa m vị trí.

Với dữ liệu là số thực ta sử dụng mẫu nhóm kí tự định dạng **%m.nf** để báo rằng cần dành m vị trí để hiển thị số thực và trong m vị trí đó dành n vị trí để hiển thị phần thập phân.

Ví dụ:

```
printf("\n%f",12.345);
printf("\n%.2f",12.345);
printf("\n%8.2f",12.345);
```

Kết quả:

```
12.345000
12.35
□□□12.35
```

**Căn lề trái**

Khi hiển thị dữ liệu, mặc định C căn lề phải. Nếu muốn căn lề trái khi hiển thị dữ liệu ta chỉ cần thêm dấu trừ - vào ngay sau dấu %.

Ví dụ:

```
printf("\n%-3d %-15s %-4.2f %-3c", 1, "nguyen van a", 8.5, 'g');
printf("\n%-3d %-15s %-4.2f %-3c", 2, "tran van b", 6.75, 'k');
```

Kết quả:

```
1  nguyen van a   8.50 g
2  tran van b     6.75 k
```

Dễ thấy các thuộc tính định dạng độ rộng tối thiểu, căn lề... giúp cho việc hiển thị dữ liệu được thẳng, đều và đẹp hơn.

| Thuộc tính | Quy cách kí tự định dạng (m, n là các số nguyên) | Ví dụ |
|---|---|---|
| Độ rộng tối thiểu | **%md, %ms, %mc** | printf("%3d",10);<br>printf("%4s","CNTT");<br>printf("%2c",'A'); |
| Độ rộng dành cho phần thập phân | **%m.nf** | printf("%5.1f",1234.5); |
| Căn lề trái | **%-md, %-ms, %-mc** | printf("%-3d",10);<br>printf("%-4s","CNTT");<br>printf("%-2c",'A'); |

*Hàm scanf()*

Cú pháp:

***scanf(xâu_định_dạng [, danh_sách_địa_chỉ]);***

Hàm scanf() dùng để nhập dữ liệu từ bàn phím. Cụ thể nó sẽ đọc các kí tự được nhập từ bàn phím, sau đó căn cứ theo xâu_định_dạng sẽ chuyển những thông tin đã nhập được sang kiểu dữ liệu phù hợp. Cuối cùng sẽ gán những giá trị vừa nhập được vào các biến tương ứng trong danh_sách_địa_chỉ.

xâu_định_dạng trong hàm scanf() xác định khuôn dạng của các dữ liệu được nhập vào. Trong xâu_định_dạng có chứa các nhóm kí tự định dạng xác định khuôn dạng dữ liệu nhập vào.

Địa chỉ của một biến được viết bằng cách đặt dấu & trước tên biến. Ví dụ, ta có các biến có tên là a, x, ten_bien thì địa chỉ của chúng lần lượt sẽ là &a, &x, &ten_bien.

danh_sách_địa_chỉ phải phù hợp với các nhóm kí tự định dạng trong xâu_định_dạng về số lượng, kiểu dữ liệu và thứ tự. Các địa chỉ cũng được ngăn cách với nhau bởi dấu phẩy. Dưới đây là một số nhóm kí tự định dạng hay dùng và ý nghĩa.

*Ghi chú: unsigned long long

| Nhóm kí tự định dạng | Ghi chú |
|---|---|
| %d | Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int** |
| %o | Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int** hệ cơ số 8 |
| %x | Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **int** hệ cơ số 16 |
| %c | Định khuôn dạng dữ liệu nhập vào dưới dạng kí tự kiểu **char** |
| %s | Định khuôn dạng dữ liệu nhập vào dưới dạng xâu kí tự |
| %f | Định khuôn dạng dữ liệu nhập vào dưới dạng số thực kiểu **float** |
| %ld | Định khuôn dạng dữ liệu nhập vào dưới dạng số nguyên kiểu **long** |
| %lf | Định khuôn dạng dữ liệu nhập vào dưới dạng số thực kiểu **double** |

Ví dụ:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    // khai bao bien
    int a;
    float x;
    char ch;
    char  str[30];
    // Nhap du lieu
    printf("Nhap vao mot so nguyen");
    scanf("%d",&a);
    printf("\n  Nhap vao mot so thuc");
    scanf("%f",&x);
    printf("\n Nhap vao mot ki tu");
    fflush(stdin); scanf("%c",&ch);
    printf("\n Nhap vao mot xau ki tu");
```

```
    fflush(stdin); scanf("%s",str);
    // Hien thi du lieu vua nhap vao
    printf("\n Nhung du lieu vua nhap vao");
    printf("\n So nguyen: %d",a);
    printf("\n So thuc : %.2f",x);
    printf("\n Ki tu: %c:,ch);
    printf("\n Xau ki tu: %s",str);
}
```

Kết quả:

```
Nhap vao mot so nguyen: 2007
Nhap vao mot so thuc: 18.1625
Nhap vao mot ki tu: b
Nhap vao mot xau ki tu: ngon ngu lap trinh C
Nhung du lieu vua nhap vao
So nguyen: 2007
So thuc: 18.16
Ki tu: b
Xau ki tu: ngon
```

Một số quy tắc cần lưu ý khi sử dụng hàm scanf():

- Quy tắc 1: Khi đọc số, hàm scanf() quan niệm rằng mọi kí tự số, dấu chấm ('.') đều là kí tự hợp lệ. Khi gặp các dấu phân cách như tab, xuống dòng hay dấu cách (space bar) thì scanf() sẽ hiểu là kết thúc nhập dữ liệu cho một số.
- Quy tắc 2: Khi đọc kí tự, hàm scanf() cho rằng mọi kí tự có trong bộ đệm của thiết bị vào chuẩn đều là hợp lệ, kể cả các kí tự tab, xuống dòng hay dấu cách.
- Quy tắc 3: Khi đọc xâu kí tự, hàm scanf() nếu gặp các kí tự dấu cách, dấu tab hay dấu xuống dòng thì nó sẽ hiểu là kết thúc nhập dữ liệu cho một xâu kí tự. Vì vậy trước khi nhập dữ liệu kí tự hay xâu kí tự ta nên dùng lệnh fflush(stdin).

### III.3.1.2. Các lệnh nhập xuất khác

Hàm gets(), có cú pháp:

***gets(xâu_kí_tự);***

Hàm gets() dùng để nhập vào từ bàn phím một xâu kí tự bao gồm cả dấu cách, điều mà hàm scanf() không làm được.

Hàm puts(), có cú pháp:

***puts(xâu_kí_tự);***

Hàm puts() sẽ hiển thị ra màn hình nội dung xâu_kí_tự và sau đó đưa con trỏ xuống dòng mới. Vì vậy nó tương đương với lệnh printf("%s\n",xâu_kí_tự).

Hàm getch(), có cú pháp:

***getch();***

Hàm getch() là hàm không có tham số. Nó đọc một kí tự bất kì nhập vào từ bàn phím nhưng không hiển thị kí tự đó lên màn hình. Lệnh getch() thường dùng để chờ người sử dụng ấn một phím bất kì rồi sẽ kết thúc chương trình.

Để sử dụng các hàm gets(), puts(), getch() ta cần khai báo tệp tiêu đề conio.h.

Ví dụ:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    // khai bao bien
    char str[30];
    // Nhap du lieu
    puts("Nhap vao mot xau ki tu:");
    fflush(stdin); gets(str);
    // Hien thi du lieu vua nhap vao
    puts("Xau vua nhap vao: ");
    puts(str);
    puts("An phim bat ki de ket thuc ...");
    getch();
}
```

Kết quả:

```
Nhap vao mot xau ki tu:
ngon ngu lap trinh C
Xau vua nhap vao:
ngon ngu lap trinh C
An phim bat ki de ket thuc ...
```

## III.3.2. Cấu trúc lệnh khối

Một cách hình thức ta có thể định nghĩa một lệnh khối là dãy các câu lệnh được đặt trong cặp dấu ngoặc nhọn { }.

```
{
    lệnh_1;
    lệnh_2;
    ....
    lệnh_n;
}
```

Trong lệnh khối có thể chứa lệnh khối khác, ta gọi đó là các lệnh khối lồng nhau. Sự lồng nhau của các lệnh khối là không hạn chế. Các lệnh trong lệnh khối được thực hiện tuần tự theo trật tự xuất hiện.

```
{
lệnh;
    {
        lệnh;
        ...

    }
    ...
}
```

C cho phép khai báo biến trong lệnh khối. Ràng buộc duy nhất là phần khai báo phải nằm trước phần câu lệnh.

Ví dụ:

```
#include <conio.h>
#include <stdio.h>
void main()
// Noi dung cua ham main() cung la mot khoi lenh
{
        // khai bao bien
        int c;
        c = 10;
        printf(" Gia tri cua c = %d day la c ngoai",c);
        // bat dau mot khoi lenh khac
        {
            int c;
            c = 10;
            printf("\n Gia tri cua c = %d day la c trong",c);
            printf("\n Tang gia tri cua c them 10 don vi");
            c = c + 10;
            printf("\n Gia tri cua c = %d day la c trong",c);
        }
        printf("\n Gia tri cua c = %d day la c ngoai",c);
        getch();
}
```

Kết quả:

```
Gia tri cua c = 10 day la c ngoai
Gia tri cua c = 10 day la c trong
Tang gia tri cua c them 10 don vi
Gia tri cua c = 20 day la c trong
Gia tri cua c = 10 day la c ngoai
```

## III.3.3. Cấu trúc if

Cú pháp cấu trúc **if**

***if (biểu_thức_điều_kiện)***

***lệnh;***

Cú pháp cấu trúc **if … else**

***if (biểu_thức_điều_kiện)***

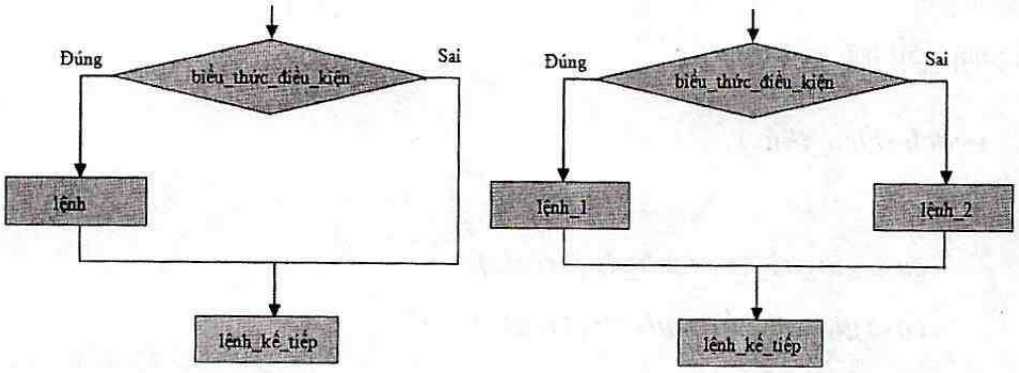
***lệnh_1;***

***else***

***lệnh_2;***

Lưu ý là lệnh, lệnh_1 và lệnh_2 có thể là lệnh khối.

Dưới đây là sơ đồ khối của hai cú pháp lệnh trên.




***Hình III.3. Sơ đồ khối cấu trúc if***

Ví dụ: Bài toán tìm số lớn nhất trong 2 số thực *a* và *b*. Cách làm là so sánh *a* với *b*, nếu *a* nhỏ hơn *b* thì *b* là số lớn nhất, còn nếu không, tức là $a \geq b$, thì *a* là số lớn nhất.

```
#include <conio.h>
#include <stdio.h>
void main()
{
        // khai bao bien
        float a, b;
        float max;
        printf(" Nhap gia tri a va b: ");
        scanf("%f %f",&a,&b);
        if(a<b)
            max = b;
        else
            max = a;
        printf("\n So lon nhat trong 2 so %.0f va %.0f la %.0f ",a,b,max);
        getch();
```

}

Kết quả:

```
Nhap vao 2 gia tri a va b: 23 247
So lon nhat trong hai so 23 va 247 la 247
```

## III.3.4. Cấu trúc lựa chọn switch

Cú pháp cấu trúc **switch**

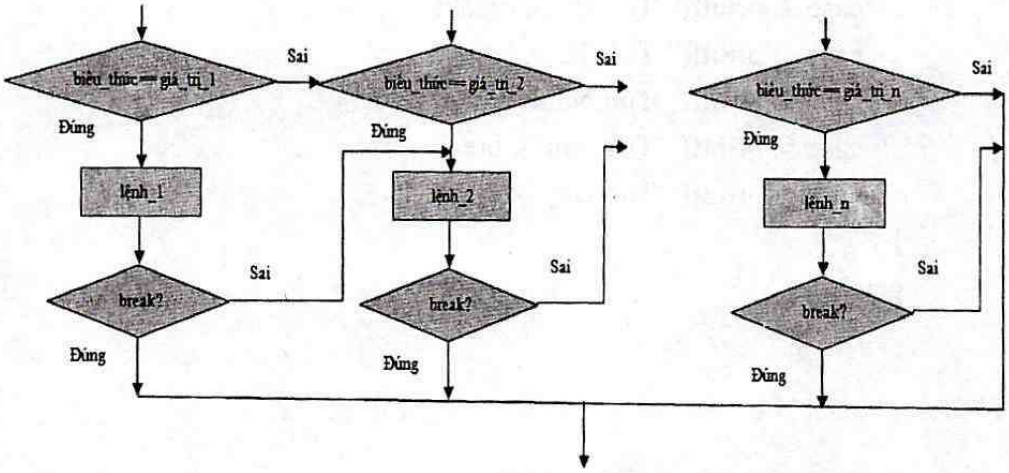
```
switch (biểu_thức)
{
    case gia_tri_1:   lenh_1; [break;]
    case gia_tri_2:   lenh_2; [break;]
    ...
    case gia_tri_n:   lenh_n; [break;]
    [default:         lenh_n+1; [break;]]
}
```

*Cơ chế hoạt động*: câu lệnh **switch** ban đầu sẽ tính giá trị của biểu_thức, sau đó so sánh với các giá_trị_k với k = 1, 2, …, n đứng sau **case**. Xảy ra 2 trường hợp:

- Nếu trong dãy các giá trị giá_trị_1, giá_trị_2, ... tồn tại giá trị bằng biểu_thức: Gọi i là chỉ số của giá trị đầu tiên trong dãy thỏa mãn giá_trị_i bằng biểu_thức, khi đó lệnh_i sẽ được thực hiện. Sau khi thực hiện xong lệnh_i, nếu có lệnh **break** thì chương trình sẽ chuyển sang thực hiện lệnh tiếp sau cấu trúc **switch**. Nếu không có lệnh **break** thì chương trình sẽ chuyển sang thực hiện các lệnh sau lệnh_i nằm trong **switch** (tức là lệnh_i+1, lệnh_i+2…) cho đến khi gặp lệnh **break** đầu tiên hoặc sau khi thực hiện xong lệnh n. Sau đó chương trình sẽ chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.
- Nếu không tồn tại giá_trị_k (với k = 1, 2, …n) nào bằng giá trị của biểu_thức thì sẽ có 2 khả năng:

- Nếu có nhãn **default**: Chương trình sẽ thực hiện lệnh_n+1 rồi chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.
- Nếu không có nhãn **default**: Chương trình chuyển sang thực hiện lệnh tiếp theo sau cấu trúc **switch**.

*Sơ đồ*:




***Hình III.4. Sơ đồ khối cấu trúc switch***

Ví dụ:

```
/* Ví dụ sau yêu cầu người dùng nhập vào một số nguyên không âm và đưa ra
ngày trong tuần tương ứng với số nguyên đó. Ở đây ta quy ước những số chia
hết cho 7 ứng với Chủ nhật, chia 7 dư 1 ứng với thứ Hai, ..., chia 7 dư 6 ứng
với thứ Bảy.*/
#include <conio.h>
#include <stdio.h>
void main()
{
        // khai bao bien
        int a;
        do
        {
            printf("\n Nhap mot gia tri so nguyen khong am: ");
            scanf("%d",&a);
            if(a<0)
               printf("\n so vua nhap la so am");
```

```
        }while(a<0);
        printf("\n Thu trong tuan tuong ung voi so do la: ");
        switch(a % 7)
        {
            case 0: printf(" Chu nhat"); break;
            case 1: printf(" Thu Hai"); break;
            case 2: printf(" Thu Ba"); break;
            case 3: printf(" Thu Tu"); break;
            case 4: printf(" Thu Nam"); break;
            case 5: printf(" Thu Sau"); break;
            case 6: printf(" Thu Bay"); break;
        }
        getch();
    }
```

Kết quả:

```
Nhap vao mot gia tri so nguyen: 2356
Thu tuong ung voi so do la Thu Nam
```

Người ta thường dựa trên tính chất tự động chuyển xuống các câu lệnh sau khi không có lệnh break để viết chung mã lệnh cho các trường hợp khác nhau nhưng cùng được xử lý giống nhau. Ví dụ khi viết chương trình hỗ trợ menu dòng lệnh không phân biệt chữ hoa chữ thường hay bài toán in ra số ngày trong các tháng trong năm dưới đây. Trong một năm các tháng có 30 ngày là 4, 6, 9, 11 còn các tháng có 31 ngày là 1, 3, 5, 7, 8, 10, 12. Riêng tháng hai có thể có 28 hoặc 29 ngày.

```
#include <stdio.h>
#include<conio.h>
int main ()
{
        int thang;
        clrscr();
        printf("\n Nhap vao thang trong nam ");
        scanf("%d",&thang);
        switch(thang)
        {
```

```
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            printf("\n Thang %d co 31 ngay ",thang);
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            printf("\n Thang %d co 30 ngay ",thang);
            break;
        case 2:
            printf ("\ Thang 2 co 28 hoac 29 ngay");
            break;
        default :
            printf("\n Khong co thang %d", thang);
            break;
    }
    getch();
    return 0;
}
```

*Lưu ý*: Giá trị của biểu thức kiểm tra phải là số nguyên tức là phải có kiểu dữ liệu là **char**, **int**, **long**. Một cách tương ứng các giá trị sau **case** cũng phải nguyên. Đây là một trong những điểm phân biệt giữa cấu trúc **switch** và **if...else**.

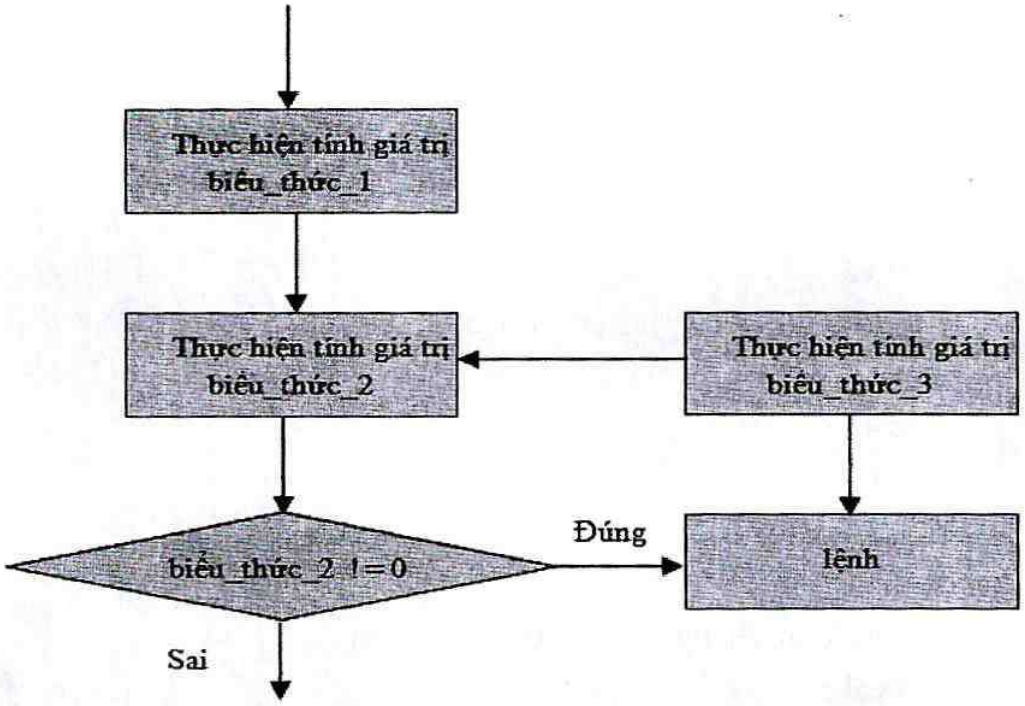
## III.3.5. Vòng lặp for

Dạng thường gặp của vòng lặp **for** là:

***for([biểu_thức_1];[biểu_thức_2];[biểu_thức_3])***

***lệnh;***

*Sơ đồ:*




***Hình III.5. Sơ đồ khối cấu trúc for***

Câu lệnh **for** thường dùng để thực hiện lặp đi lặp lại một công việc nào đó với số lần lặp xác định.

Ví dụ 1: Hãy đưa ra màn hình các số nguyên dương nhỏ hơn 10.

Cách làm: Có 9 số nguyên dương nhỏ hơn 10 là 1, 2, 3, 4, 5, 6, 7, 8 và 9. Để in 9 số nguyên dương này ta cần sử dụng một biến nguyên đặt tên là *i*.

Bước 1: Gán cho i giá trị bằng 1;

Bước 2: Đưa ra màn hình giá trị của i;

Bước 3: Tăng giá trị của i thêm 1 đơn vị;

Bước 4: Kiểm tra nếu giá trị của i ≤ 9 thì quay về bước 2, nếu giá trị của i > 9 thì chuyển sang bước 5;

Bước 5: Kết thúc.

Chương trình in ra màn hình các số nguyên dương nhỏ hơn 10 sử dụng vòng lặp for:

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

```
void main()
{
    int i;
    for(i = 1;i <= 9;i++)
        printf("%5d",i);// ta danh 5 vi tri de in moi so
    getch();
}
```

Kết quả thực hiện:

```
  0  1  2  3  4  5  6  7  8  9
```

Ta có thể so sánh cách làm này với việc phải viết một lệnh printf() trong đó liệt kê toàn bộ các số nguyên dương lẻ nhỏ hơn 10.

Ví dụ 2: Tính và hiển thị ra màn hình tổng của 100 số tự nhiên lẻ đầu tiên.

Cách làm: 100 số lẻ đầu tiên là 1, 3, 5, ..., 199. Ta cần sử dụng một biến nguyên S để chứa giá trị của tổng và một biến nguyên i.

Bước 1: Ban đầu gán cho S giá trị bằng 0, gán cho i giá trị bằng 1;

Bước 2: S = S + i;

Bước 3: Tăng giá trị của i thêm 2 đơn vị;

Bước 4: Kiểm tra nếu giá trị của i ≤ 199 thì quay về bước 2, ngược lại nếu i > 199 thì chuyển sang bước 5;

Bước 5: Kết thúc.

Chương trình:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i;
        int S;
        S = 0;
        for(i = 1; i <= 199; i = i+2)
            S = S+i;
```

```
        printf("Tong cua 100 so nguyen duong le dau tien la %d",S);
        getch();
}
```

Kết quả thực hiện:

```
Tong cua 100 so nguyen duong le dau tien la 10000
```

## III.3.6. Vòng lặp while và do – while

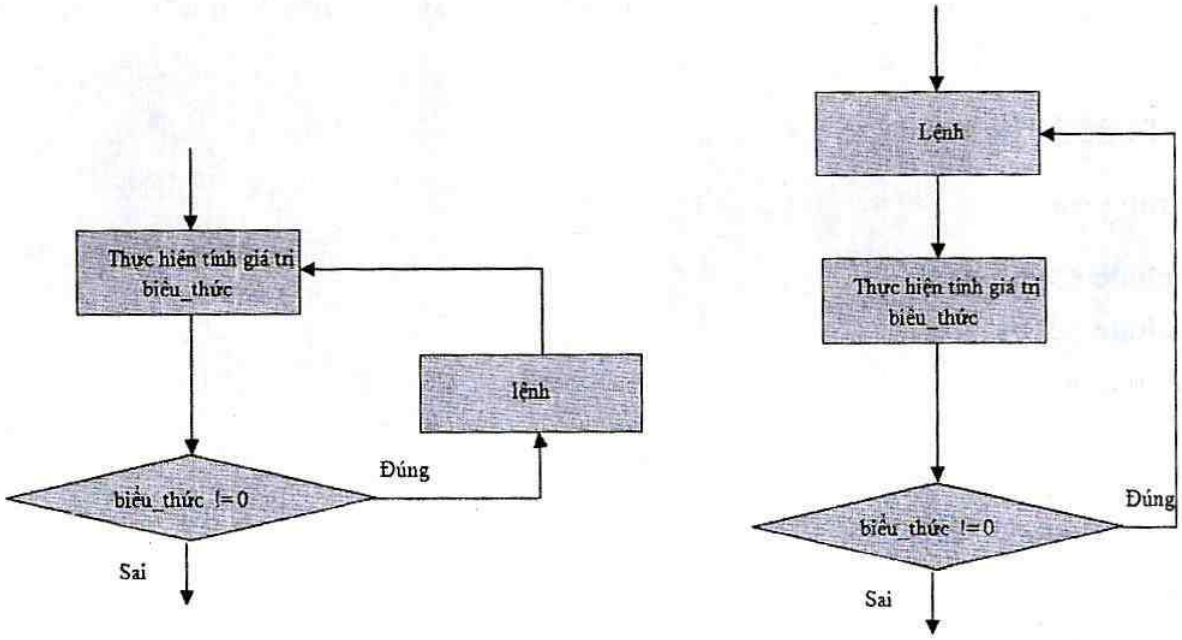
*Cú pháp vòng lặp* **while**

```
while (biểu_thức)
    lệnh;
```

*Cú pháp vòng lặp* ***do {...} while***

```
do
{
    lệnh;
}while (biểu_thức);
```

Sơ đồ vòng lặp **while** và **do{...}while:**




***Hình III.6. Sơ đồ khối cấu trúc while và do while***

Sự khác nhau giữa **while** và **do{...}while:**

- Lệnh **while** kiểm tra điều kiện vòng lặp (tức là giá trị của biểu thức) trước rồi mới thực hiện lệnh, do vậy sẽ có khả năng lệnh không được thực hiện lần nào.
- Lệnh **do{...}while** thực hiện lệnh trước rồi mới kiểm tra điều kiện của vòng lặp, do vậy lệnh sẽ luôn được thực hiện ít nhất một lần.

Cấu trúc **while** và **do{...}while** được dùng để thực hiện lặp đi lặp lại một công việc nào đó với số lần lặp không xác định.

Ví dụ: Sau đây là 2 đoạn chương trình có chức năng tương đương nhau nhưng một đoạn sử dụng cấu trúc **while**, một đoạn sử dụng cấu trúc **do{...}while**.

Chức năng của chương trình là yêu cầu người dùng nhập vào giá trị 1 số nguyên, đưa ra thông báo số đó có phải là số hoàn thiện hay không, sau đó hỏi người dùng có muốn nhập lại số nguyên khác và kiểm tra có phải số hoàn thiện hay không.

*Đoạn chương trình sử dụng cấu trúc **do{...}while***

```
#include <stdio.h>
#include <conio.h>
void main()
{
    long int n;
    long int tong;
    long int i;
    char ch;

    clrscr();
    do
    {
        tong = 0;
        printf("\n Nhap vao mot so nguyen: ");
        scanf("%ld",&n);
        printf("\n Cac uoc so cua %ld la: ",n);
        for(i = 1;i<n;i++)
            if(n % i == 0)
                        {
```

```
                    printf("%5d",i);
                    tong = tong + i;
            }
        printf("\n Tong cac uoc so cua %ld bang %ld",n,tong);
        if(tong == n)
            printf("\n %5ld LA so hoan thien", n);
        else
            printf("\n %5ld KHONG LA so hoan thien", n);
        printf("\n Ban co muon thuc hien lai(c/k)? ");
        fflush(stdin);
        scanf("%c",&ch);
    }while((ch!='k')&&(ch!='K'));
    printf("\n An phim bat ki de ket thuc ...");
    getch();
}
```

*Đoạn chương trình viết bằng cấu trúc **while***

```
#include <stdio.h>
#include <conio.h>
void main()
{
    long int n;
    long int tong;
    long int i;
    char ch;

    clrscr();
    ch = 'c';
    while((ch != 'k')&&(ch != 'K'))
    {
        tong = 0;
        printf("\n Nhap vao mot so nguyen: ");
        scanf("%ld",&n);
        printf("\n Cac uoc so cua %ld la: ", n);
        for(i = 1;i<n;i++)
            if(n % i == 0)
                {
```

```
                    printf("%5d",i);
                    tong = tong + i;
            }
        printf("\n Tong cac uoc so cua %ld bang %ld",n,tong);
        if(tong == n)
            printf("\n %5ld LA so hoan thien", n);
        else
            printf("\n %5ld KHONG LA so hoan thien", n);
        printf("\n Ban co muon thuc hien lai(c/k)? ");
        fflush(stdin);
        scanf("%c",&ch);
    }
    printf("\n An phim bat ki de ket thuc ...");
    getch();
}
```

Kết quả thực hiện chương trình:

```
Nhap vao mot so nguyen: 10
Cac uoc so cua 10 la: 1   2   5
Tong cac uoc so cua 10 bang 8
10 KHONG LA so hoan thien
Ban co muon thuc hien lai(c/k)? c
Nhap vao mot so nguyen: 12
Cac uoc so cua 12 la: 1   2   3   4   6
Tong cac uoc so cua 12 bang 16
12 KHONG LA so hoan thien
Ban co muon thuc hien lai(c/k)? c
Nhap vao mot so nguyen: 28
Cac uoc so cua 28 la: 1   2   4   7   14
Tong cac uoc so cua 28 bang 28
28 LA so hoan thien
Ban co muon thuc hien lai(c/k)? k
An phim bat ki de ket thuc ...
```

## III.3.7. Các lệnh thay đổi cấu trúc lập trình

Các vòng lặp **while**, **do{...}while**, hay **for** sẽ kết thúc quá trình lặp khi biểu thức điều kiện của vòng lặp không còn được thỏa mãn. Tuy nhiên trong lập trình đôi khi ta cũng cần thoát khỏi vòng lặp ngay cả khi biểu thức điều kiện của vòng lặp vẫn còn được thỏa mãn. Để hỗ trợ người lập trình làm việc đó, ngôn ngữ C cung cấp 2 câu lệnh là **continue** và **break**.

### III.3.7.1. Continue

Khi gặp lệnh **continue** trong thân vòng lặp, chương trình sẽ chuyển sang thực hiện một vòng lặp mới và bỏ qua việc thực hiện các câu lệnh nằm sau lệnh **continue** trong thân vòng lặp.

Ví dụ sau đây sẽ in ra màn hình các số tự nhiên lẻ và nhỏ hơn 100:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    for(i = 1;i<100;i++)
    {
        if(i%2 == 0) continue;
                printf("%5d",i);
        if((i+1)%20 ==0) printf("\n");
    }
    getch();
}
```

Kết quả thực hiện:

```
  1   3   5   7   9  11  13  15  17  19
 21  23  25  27  29  31  33  35  37  39
 41  43  45  47  49  51  53  55  57  59
 61  63  65  67  69  71  73  75  77  79
 81  83  85  87  89  91  93  95  97  99
```

### III.3.7.2. Break

Khi gặp lệnh **break**, chương trình sẽ thoát khỏi vòng lặp (đối với trường hợp lệnh **break** nằm trong các cấu trúc lặp **while**, **do{...}while**, **for**) hoặc thoát khỏi cấu trúc **switch** (với trường hợp lệnh **break** nằm trong cấu trúc **switch**).

Ví dụ sau sẽ thực hiện việc yêu cầu người dùng nhập vào một kí tự và màn hình thông báo về kí tự vừa nhập. Việc này được lặp đi lặp lại cho đến khi kí tự nhập vào là kí tự 't' hoặc 'T' (viết tắt của từ thoát).

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char ch;
    clrscr();
    do
    {
        printf("\n Nhap vao mot ki tu: ");
        fflush(stdin);
        scanf("%c",&ch);
        printf("\n Ki tu vua nhap vao la %c",ch);
        if((ch =='T')||(ch =='t')) break;
    }while(1);
    printf("\n An phim bat ki de ket thuc chuong trinh...");
    getch();
}
```

Kết quả thực hiện chương trình:

```
Nhap vao mot ki tu: a
Ki tu vua nhap vao la a
Nhap vao mot ki tu: 5
Ki tu vua nhap vao la 5
Nhap vao mot ki tu: t
Ki tu vua nhap vao la t
An phim bat ki de ket thuc chuong trinh...
```

# III.4. Mảng, con trỏ và xâu ký tự

## III.4.1. Mảng

### III.4.1.1. Khái niệm mảng

*Khái niệm mảng*

Mảng là một tập hợp hữu hạn các phần tử có cùng kiểu dữ liệu được lưu trữ kế tiếp nhau trong bộ nhớ. Các phần tử trong mảng có cùng tên (và cũng là tên mảng) nhưng phân biệt với nhau ở chỉ số cho biết vị trí của chúng trong mảng.

### III.4.1.2. Khai báo và sử dụng mảng

*Cú pháp khai báo*

Trong C để khai báo một mảng ta sử dụng cú pháp khai báo sau:

***kiểu_dữ_liệu tên_mảng [kích_thước_mảng];***

Trong đó kiểu_dữ_liệu là kiểu dữ liệu của các phần tử trong mảng, tên_mảng là tên của mảng, kích_thước_mảng cho biết số phần tử trong mảng.

*Ví dụ*:

```
int mang_nguyen[10]; // khai bao mang 10 phan tu co kieu du lieu int
float mang_thuc[4];  // khai bao mang 4 phan tu co kieu du lieu float
char mang_ki_tu[6];  // khai bao mang 6 phan tu co kieu du lieu char
```

Trong ví dụ trên, mảng mang_nguyen được lưu trữ trên 20 ô nhớ (mỗi ô nhớ có kích thước 1 byte, 2 ô nhớ kích thước là 2 byte lưu trữ được một số nguyên kiểu **int**) liên tiếp nhau. Do C đánh số các phần tử của mảng bắt đầu từ 0 nên phần tử thứ i của mảng sẽ có chỉ số là i-1 và do vậy sẽ có tên là mang_nguyen[i-1]. Ví dụ, phần tử thứ nhất của mảng là mang_nguyen[0], phần tử thứ hai là mang_nguyen[1], phần tử thứ năm là mang_nguyen[4]...

| | | | | |
|---|---|---|---|---|
| mang_nguyen[0] | mang_nguyen[1] | ... | ... | mang_nguyen[9] |

Kích thước của mảng bằng kích thước một phần tử nhân với số phần tử.

*Mảng một chiều và nhiều chiều*

Các mảng trong ví dụ trên là các mảng một chiều. Mảng là tập hợp các phần tử cùng kiểu dữ liệu, nếu mỗi phần tử của mảng cũng là một mảng khác thì khi đó ta có mảng nhiều chiều. Khái niệm mảng rất giống với khái niệm vector trong toán học.

Ví dụ sau khai báo một mảng gồm 6 phần tử, trong đó mỗi phần tử lại là một mảng gồm 5 số nguyên kiểu **int**. Mảng này là mảng 2 chiều:

```
int a[6][5];
```

Còn khai báo

```
int b[3][4][5];
```

thì lại khai báo một mảng gồm 3 phần tử, mỗi phần tử lại là một mảng 2 chiều gồm 4 phần tử. Mỗi phần tử của mảng 2 chiều lại là một mảng (1 chiều) gồm 5 số nguyên kiểu **int**. Mảng b ở trên được gọi là một mảng 3 chiều.

*Sử dụng mảng*

Để truy nhập vào một phần tử của mảng ta phải sử dụng tên của nó. Tên một phần tử của mảng được tạo thành từ tên mảng và theo sau là chỉ số của phần tử đó trong mảng được đặt trong cặp dấu ngoặc vuông.

***tên_mảng[chỉ_số_của_phần_tử]***

Ví dụ: Với khai báo

```
int mang_nguyen[3];
```

Thì mang_nguyen[0] sẽ là phần tử thứ nhất của mảng.

mang_nguyen[1] sẽ là phần tử thứ 2 của mảng.

mang_nguyen[2] sẽ là phần tử thứ 3 của mảng.

Với mảng nhiều chiều như:

```
int a[6][5];
```

thì a[0] là phần tử đầu tiên của mảng, phần tử này bản thân nó lại là một mảng một chiều. Phần tử đầu tiên của mảng một chiều a[0] sẽ là a[0][0]. Phần tử tiếp theo của a[0] sẽ là a[0][1]... và dễ dàng tính được a[2][3] sẽ là phần tử thứ 4 của phần tử thứ 3 của a.

Một cách tổng quát a[i][j] sẽ là phần tử thứ $j$+1 của a[i], mà phần tử a[i] lại là phần tử thứ $i$+1 của a.

### III.4.1.3. Các thao tác cơ bản trên mảng

#### *a. Nhập dữ liệu cho mảng*

Sau khi khai báo mảng ta phải nhập dữ liệu cho mảng. Nhập dữ liệu cho mảng là nhập dữ liệu cho từng phần tử của mảng. Mỗi một phần tử của mảng thực chất là một biến có kiểu dữ liệu là kiểu dữ liệu chung của mảng. Để nhập dữ liệu cho các phần tử của mảng ta có thể dùng hàm scanf() hoặc lệnh gán tương tự như biến thông thường.

Ví dụ:

```
float a[10];    // khai bao mot mang so thuc co 10 phan tu.
int i;
// Nhap tu ban phim mot so thuc va gan gia tri so thuc do
// cho phan tu thu 2 cua mang, tuc la a[1].
scanf("%f",&a[1]);
// Gan gia tri cho phan tu a[2].
a[2] = a[1] + 5;
```

Nếu ta muốn gán giá trị cho các phần tử của mảng một cách hàng loạt, ta có thể dùng lệnh **for**. Ví dụ:

```
int b[10];
int i;
// Nhap gia tri tu ban phim cho tat ca cac phan tu cua mang b.
for(i = 0; i < 10; i++)
{
   printf("\n Nhap gia tri cho b[%d]", i);
   scanf("%d",&b[i]);
}
```

Trường hợp ta không biết trước mảng sẽ có bao nhiêu phần tử mà chỉ biết số phần tử tối đa có thể có của mảng, còn số phần tử thực sự của mảng thì chỉ biết khi chạy chương trình. Khi đó cần khai báo mảng với số phần tử bằng số phần tử tối đa, ngoài ra cần một biến để lưu giữ số phần tử thực sự của mảng.

```
int a[100]; // khai bao mang, gia su so phan tu toi da cua a la 100.
int n;      // bien luu giu so phan tu thuc su cua mang.
int i;
printf("\n Cho biet so phan tu cua mang: ");
scanf("%d",&n);
```

```
for(i = 0; i < n; i++)
{
    printf("\n a[%d] = ", i);
    scanf("%d",&a[i]);
}
```

Mảng có thể được khởi tạo giá trị ngay khi khai báo, ví dụ:

```
int a[4] = {4, 9, 22, 16};
float b[3] = {40.5, 20.1, 100};
char c[5] = {'h', 'e', 'l', 'l', 'o'};
```

Câu lệnh thứ nhất có tác dụng tương đương với 4 lệnh gán:

```
a[0] = 4; a[1] = 9; a[2] = 22; a[3] = 16;
```

### b. Xuất dữ liệu chứa trong mảng

Để hiển thị giá trị của các phần tử trong mảng ta dùng hàm printf(). Ví dụ sau minh họa việc nhập giá trị cho các phần tử của mảng, sau đó hiển thị giá trị của các phần tử đó theo các cách khác nhau.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a[5];
    int i, k;
    // Nhap gia tri cho cac phan tu cua mang a tu ban phim
    for(i = 0; i < 5; i++)
    {
        printf("\n a[%d] = ", i);
        scanf("%d", &a[i]);
    }
    // Hien thi gia tri cua phan tu bat ki, gia su a[3] len man hinh
    printf("\n a[3] = %d", a[3]);
    // Hien thi gia tri cua tat ca cac phan tu, moi phan tu tren mot dong
    for(i = 0; i < 5; i++)
        printf("\n%d", a[i]);
    // Hien thi gia tri cua tat ca cac phan tu tren cung mot dong, cac gia tri cach
    // nhau 2 vi tri.
```

```
    printf("\n");     // Xuong dong moi
    for(i = 0; i < 5; i++)
        printf("%d  ", a[i]);
    // Hien thi gia tri cua tat ca cac phan tu, trong do k phan tu tren mot dong.

    // Cac phan tu tren cung dong cach nhau 2 vi tri

    printf("\n Cho biet gia tri cua k = ");
    scanf("%d",&k);
    for(i = 0; i < 5; i++)
    {
        printf("%d  ",a[i]);
        if((i+1)%k == 0)// da hien thi du k phan tu tren mot dong thi phai xuong dong.
            printf("\n");
    }
    getch();
}
```

Kết quả:

```
a[0] = 6
a[1] = 14
a[2] = 23
a[3] = 37
a[4] = 9
a[3] = 37
6
14
23
37
9
6  14  23  37  9
 Cho biet gia tri cua k = 2
6  14
23  37
9
```

### c. Tìm phần tử có giá trị lớn nhất, phần tử có giá trị nhỏ nhất

Để tìm phần tử có giá trị lớn nhất trong mảng, ban đầu ta giả sử phần tử đó là phần tử đầu tiên của mảng. Sau đó lần lượt so sánh với các phần tử còn lại trong mảng. Nếu gặp phần tử nhỏ hơn thì chuyển sang so sánh với phần tử tiếp theo. Nếu gặp phần tử lớn hơn thì ta sẽ coi phần tử này là phần tử lớn nhất rồi chuyển sang so sánh với phần tử tiếp theo. Sau khi so sánh với phần tử cuối cùng ta sẽ tìm được phần tử lớn nhất trong mảng. Đoạn chương trình sau minh họa giải thuật tìm phần tử lớn nhất:

```
int a[100];
int i, n;
int max;
printf("\n Cho biet so phan tu cua mang: ");
scanf("%d",&n);
// Nhap du lieu cho mang
for(i = 0; i < n; i++)
{
    printf("\n a[%d] = ",i);
    scanf("%d",&a[i]);
}
// Tim phan tu lon nhat
max = a[0];    // Ban dau gia su phan tu lon nhat la a[0]
// Lan luot so sanh voi cac phan tu con lai trong mang
for(i = 1; i < n; i++)
    if(max < a[i])        // Gap phan tu co gia tri lon hon
        max = a[i]; // coi phan tu nay la phan tu lon nhat
printf("\n Phan tu lon nhat trong mang la: %d", max);
```

Ta cũng làm tương tự với trường hợp tìm phần tử nhỏ nhất trong mảng.

## III.4.1.4. Tìm kiếm trên mảng

Yêu cầu của thao tác tìm kiếm trên mảng: Cho một mảng dữ liệu và một dữ liệu bên ngoài mảng. Hãy tìm (các) phần tử của mảng có giá trị bằng giá trị của dữ liệu bên ngoài trên. Nếu có (các) phần tử như vậy thì hãy chỉ ra vị trí của chúng trong mảng. Nếu không tồn tại (các) phần tử như vậy thì đưa ra thông báo không tìm thấy.

Cách làm là lần lượt duyệt qua từng phần tử của mảng, so sánh giá trị của phần tử đang được duyệt với giá trị của dữ liệu bên ngoài. Nếu phần tử đang xét bằng dữ liệu bên ngoài thì ta ghi nhận vị trí của phần tử đó. Sau đó chuyển qua duyệt phần tử kế tiếp trong mảng. Quá trình được lặp đi lặp lại cho đến khi duyệt xong phần tử cuối cùng của mảng. Để có thể trả lời cho cả tình huống không tồn tại phần tử như vậy trong mảng ta nên sử dụng một biến kiểm tra và khi gặp phần tử bằng giá trị dữ liệu bên ngoài thì ta bật biến đó lên, nếu biến đó không được bật lần nào thì ta trả lời là không có phần tử như vậy trong mảng. Phương pháp trên được gọi là phương pháp tìm kiếm tuần tự (*sequential search*).

Dưới đây là cài đặt của thuật toán tìm kiếm tuần tự cho trường hợp mảng dữ liệu là mảng các số nguyên kiểu **int**.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int m[100], chi_so[100];
    int n;   // n la so phan tu trong mang
    int i, k, kiem_tra;
    clrscr();     // xoa man hinh de tien theo doi
    // Nhap gia tri du lieu cho mang m
    // Truoc tien phai biet so phan tu cua mang
    printf(" Cho biet so phan tu co trong mang: ");
    scanf("%d",&n);
    // Roi lan luot nhap gia tri cho cac phan tu trong mang
        for(i  = 0;i<n;i++)
    {
        int temp;
        printf("\n Cho biet gia tri cua m[%d] = ",i);
        scanf("%d",&temp);
        m[i] = temp;
    }
    // Yeu cau nguoi su dung nhap vao gia tri cho du lieu k
    printf("\n Cho biet gia tri cua du lieu k: ");
    scanf("%d",&k);
    // Bat đau qua trinh tim kiem
    kiem_tra = 0;
```

```
    // Duyet qua tat ca cac phan tu
    for(i = 0;i<n;i++)
        if(m[i] = = k)//So sanh phan tu dang xet voi du lieu k
        {
            // Ghi nhan chi so cua phan tu dang xet
            chi_so[kiem_tra] = i;
                kiem_tra ++; //Tang bien kiem_tra them 1 don vi
        }
            // Ket luan
    if(kiem_tra > 0)
    {
        printf("\n Trong mang co %d phan tu co gia tri bang %d",kiem_tra,k);
        printf("\n Chi so cua cac phan tu la: ");
        for(i = 0;i < kiem_tra;i++)
            printf("%3d",chi_so[i]);
    }
    else
        printf("\n Trong mang khong co phan tu nao co gia tri bang %d",k);
    getch();     // Cho nguoi su dung an phim bat ky de ket thuc.
}
```

### III.4.1.5. Sắp xếp mảng

Yêu cầu của bài toán: Cho một mảng dữ liệu m[n] với n là số phần tử trong mảng. Hãy sắp xếp các phần tử trong mảng theo một trật tự nào đó, giả sử là theo chiều tăng dần (với chiều giảm dần ta hoàn toàn có thể suy luận từ cách làm với chiều tăng dần).

Sắp xếp kiểu lựa chọn (*Selection sort*): Ý tưởng của phương pháp là ta cần thực hiện n – 1 lượt sắp xếp, trong đó:

- Ở lượt sắp xếp đầu tiên ta so sánh phần tử đầu tiên của mảng m[0] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử m[1], m[2]…m[n-1]). Nếu có giá trị m[i] nào đó (i = 1, 2,…n–1) nhỏ hơn m[0] thì ta hoán đổi giá trị giữa m[0] và m[i] cho nhau. Rõ ràng sau lượt sắp xếp thứ nhất m[0] sẽ là phần tử có giá trị nhỏ nhất trong mảng.

- Ở lượt sắp xếp thứ 2 ta so sánh phần tử thứ 2 của mảng m[1] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử m[2], m[3]…m[n-1]). Nếu có giá trị m[i] nào đó (i = 2, 3,…n–1) nhỏ hơn m[1] thì ta hoán đổi giá

trị giữa m[1] và m[i] cho nhau. Sau lượt sắp xếp thứ 2 thì m[1] sẽ là phần tử có giá trị nhỏ thứ 2 trong mảng.

...

- Ở lượt sắp xếp thứ k ta so sánh phần tử thứ k của mảng là m[k-1] với tất cả các phần tử đứng sau nó trong mảng (tức là các phần tử m[k], m[k+1]...m[n-1]). Nếu có giá trị m[i] nào đó (i = k, k+1,...n–1) nhỏ hơn m[k] thì ta hoán đổi giá trị giữa m[k] và m[i] cho nhau. Sau lượt sắp xếp thứ k thì m[k-1] sẽ là phần tử có giá trị nhỏ thứ k trong mảng.

...

- Ở lượt sắp xếp thứ n-1 ta so sánh phần tử thứ n-1 của mảng m[n-2] với tất cả các phần tử đứng sau nó trong mảng (tức là phần tử m[n-1]). Nếu m[n-1] nhỏ hơn m[n-2] thì ta hoán đổi giá trị giữa m[n-2] và m[n-1] cho nhau. Sau lượt sắp xếp thứ n-1 thì m[n-2] sẽ là phần tử có giá trị nhỏ thứ n-2 trong mảng. Và dĩ nhiên phần tử còn lại là m[n-1] sẽ là phần tử nhỏ thứ n trong mảng (tức là phần tử lớn nhất trong mảng). Kết thúc n-1 lượt sắp xếp ta có các phần tử trong mảng đã được sắp xếp theo thứ tự tăng dần.

Cài đặt giải thuật:

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int m[100];
    int n;   // n la so phan tu trong mang
    int i, j, k;
    clrscr();     // Xoa man hinh de tien theo doi
    // Nhap gia tri du lieu cho mang m
    // Truoc tien phai biet so phan tu cua mang
    printf(" Cho biet so phan tu co trong mang: ");
    scanf("%d",&n);
    // Roi lan luot nhap gia tri cho cac phan tu trong mang
        for(i  = 0;i<n;i++)
    {
        int temp;
            printf("\n Cho biet gia tri cua m[%d] = ",i);
        scanf("%d",&temp);
```

```
        m[i] = temp;
    }
    // Hien thi ra man hinh mang vua nhap vao
    printf("\n Mang truoc khi sap xep:    ");
    for(i=0;i<n;i++)
        printf("%3d",m[i]);
    // Bat dau sap xep
    for(i = 0; i<n-1;i++)
    {
        // O luot sap xep thu i+1
            for(j = i+1;j<n;j++)
        {
            // So sanh m[i] voi cac phan tu con lai
           // va doi cho khi tim thay phan tu < m[i].
            if(m[j]<m[i])
            {
                int temp;
                temp = m[j]; m[j] = m[i]; m[i] = temp;
            }
        }
        // Hien thi mang sau luot sap xep thu i+1
                printf("\n Mang o luot sap xep thu %d",i+1);
        for(k = 0;k < n ;k++)
                        printf("%3d",m[k]);
    }
    getch();     // Cho nguoi su dung an phim bat ki de ket thuc.
}
```

Kết quả thực hiện:

```
Cho biet so phan tu co trong mang: 5
Cho biet gia tri cua m[0]: 34
Cho biet gia tri cua m[1]: 20
Cho biet gia tri cua m[2]: 17
Cho biet gia tri cua m[3]: 65
Cho biet gia tri cua m[4]: 21
```

```
Mang truoc khi sap xep:    34 20 17 65 21
Mang o luot sap xep thu 1: 17 34 20 65 21
Mang o luot sap xep thu 2: 17 20 34 65 21
Mang o luot sap xep thu 3: 17 20 21 65 34
Mang o luot sap xep thu 4: 17 20 21 34 65
```

## III.4.2. Con trỏ

Từ khi bắt đầu môn học cho đến giờ, chúng ta chỉ truy nhập, tác động hay thay đổi nội dung của các biến một cách trực tiếp – qua tên của chúng. Tuy nhiên, ngôn ngữ C cung cấp cho người lập trình một phương tiện gián tiếp khác đôi khi rất tiện dụng để truy xuất đến biến, đó là các con trỏ.

### III.4.2.1. Khái niệm và cách khai báo con trỏ

*Địa chỉ và giá trị của một biến*

Bộ nhớ có thể hiểu như một dãy các byte nhớ được xác định một cách duy nhất qua một *địa chỉ*. Tất cả các biến trong chương trình được lưu ở một vùng nào đó trong bộ nhớ.

Khi chúng ta khai báo một biến, chương trình dịch sẽ cấp phát cho biến đó một số ô nhớ liên tiếp đủ để chứa nội dung của biến, ví dụ một biến ký tự được cấp phát 1 byte, một biến nguyên được cấp phát 2 byte, một biến thực được cấp phát 4 byte .v.v. Địa chỉ của một biến chính là địa chỉ của byte đầu tiên trong số đó.

Một biến luôn có hai đặc tính:

- Địa chỉ của biến.
- Giá trị của biến.

Xét ví dụ sau:

```
int i, j;
i = 3;
j = i;
```

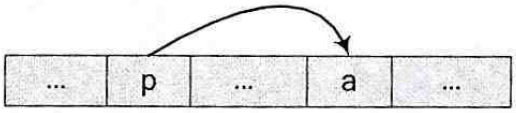
Giả sử chương trình dịch đặt biến i tại địa chỉ FFEC trong bộ nhớ, và biến j ở địa chỉ FFEE, chúng ta có

| biến | địa chỉ | giá trị |
|---|---|---|
| i | FFEC | 3 |
| j | FFEE | 3 |

Hai biến khác nhau có địa chỉ khác nhau. Phép gán i = j; chỉ có tác dụng trên giá trị của các biến, nó chỉ có ý nghĩa nội dung của vùng nhớ lưu trữ tương ứng biến j được sao chép sang nội dung vùng nhớ dành cho i. Hai biến i, j có kiểu nguyên, chúng được lưu trong hai byte.

*Khái niệm con trỏ*

Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ.Vùng nhớ này có thể chứa các biến thuộc các kiểu dữ liệu cơ sở như int, char, hay double hoặc dữ liệu có cấu trúc như mảng.




*Cú pháp khai báo một con trỏ*

*Kiểu_dữ_liệu *tên_contrỏ;*

Câu lệnh này khai báo một định danh tên_contrỏ gắn với một biến con trỏ có giá trị là địa chỉ của một biến có kiểu chỉ định. Ký tự * xác định rằng một biến con trỏ đang được khai báo. Giá trị của biến con trỏ hoàn toàn có thể thay đổi được.
Kiểu của một con trỏ phụ thuộc vào kiểu biến mà nó trỏ đến. Trong ví dụ sau, ta định nghĩa con trỏ p trỏ đến biến nguyên i:

```
int i = 3;
int *p;
p = &i;
```

Khi gán địa chỉ của i cho p, ta nói rằng p trỏ đến i. Giá trị của các biến này trong bộ nhớ sẽ là:

| biến | địa chỉ | giá trị |
|---|---|---|
| i | FFEC | 3 |
| p | FFEE | FFEC |

Chú ý: Một con trỏ chỉ có thể trỏ tới một đối tượng cùng kiểu, không thể dùng con trỏ float để trỏ vào một biến kiểu int hay ngược lại. Tuy vậy, trong một số trường hợp ép kiểu đặc biệt, quy tắc này có thể bị phá vỡ.

### III.4.2.2. Toán tử & và *

Có hai toán tử đặc biệt được dùng với con trỏ: & và *. Toán tử & là một toán tử một ngôi và nó trả về địa chỉ của biến. Toán tử thứ hai, toán tử * là một toán tử một ngôi và trả về giá trị chứa trong vùng nhớ được trỏ bởi giá trị của biến con trỏ. Trong ví dụ sau, chương trình:

```
#include<stdio.h>
void main()
{
    int i = 3;
    int *p;
    p = &i;
    printf("*p = %d \n",*p);
}
```

hiển thị ra màn hình *p = 3. Trong chương trình này i và *p là tương đương. Mọi thay đổi trên *p sẽ thay đổi luôn i. Ví dụ, nếu ta gán *p = 0 thì i sẽ có giá trị là 0.

Cả hai toán tử * và & có độ ưu tiên cao hơn tất cả các toán tử số học ngoại trừ toán tử đảo dấu. Chúng có cùng độ ưu tiên với toán tử lấy giá trị âm.

*Gán giá trị cho con trỏ*

Một cách tổng quát, một biến con trỏ có thể được gán bởi địa chỉ của một biến khác:

```
pointer_variable = &variable;
```

Hay bởi giá trị của một con trỏ khác (tốt nhất là cùng kiểu):

```
pointer_variable2 = pointer_variable;
```

Giá trị NULL cũng có thể được gán đến một biến con trỏ bằng số 0 như sau:

```
pointer_variable = 0;
```

Và cuối cùng, các biến cũng có thể được gán giá trị thông qua con trỏ của chúng.

```
*pointer_variable = 10;
```

Nói chung, các biểu thức có chứa con trỏ cũng theo cùng quy luật như các biểu thức khác trong C. Điều quan trọng cần chú ý là phải gán giá trị cho biến con trỏ trước khi sử dụng chúng, nếu không chúng có thể trỏ đến một giá trị không xác

định nào đó. Trong chương trình chúng ta có thể thao tác đồng thời trên con trỏ p và *p, nhưng ý nghĩa của chúng là rất khác nhau. Quan sát hai chương trình sau:

```
void main()
{
        int i = 3, j = 6;
        int *p1, *p2;
        p1 = &i; // gán địa chỉ của biến i vào con trỏ
        p2 = &j;
        *p1 = *p2; // gán giá trị của biến
}
```

và:

```
void main()
{
        int i = 3, j = 6;
        int *p1, *p2;
        p1 = &i;
        p2 = &j;
        p1 = p2;   // gán địa chỉ của biến con trỏ
}
```

Trước lệnh gán cuối cùng, giả sử giá trị các biến trong bộ nhớ như sau:

| biến | địa chỉ | giá trị |
|---|---|---|
| i | FFEC | 3 |
| j | FFEE | 6 |
| p1 | FFDA | FFEC |
| p2 | FFDC | FFEE |

Sau lệnh gán *p1 = *p2; ở chương trình thứ nhất:

| biến | địa chỉ | giá trị |
|---|---|---|
| i | FFEC | 6 |
| j | FFEE | 6 |
| p1 | FFDA | FFEC |
| p2 | FFDC | FFEE |

Trong khi đó lệnh gán p1 = p2 trong chương trình thứ hai sẽ dẫn tới kết quả sau:

| biến | địa chỉ | giá trị |
|---|---|---|
| i | FFEC | 3 |
| j | FFEE | 6 |
| p1 | FFDA | FFEE |
| p2 | FFDC | FFEE |

### III.4.2.3. Các phép toán trên con trỏ

Một điểm mạnh của ngôn ngữ C là khả năng thực hiện tính toán trên các con trỏ. Các phép toán số học có thể thực hiện trên con trỏ là:

- Cộng con trỏ với một số nguyên (int, long) và kết quả là một con trỏ cùng kiểu.
- Trừ con trỏ với một số nguyên và kết quả là một con trỏ cùng kiểu.
- Trừ hai con trỏ cùng kiểu cho nhau, kết quả là một số nguyên. Kết quả này nói lên khoảng cách (số phần tử thuộc kiểu dữ liệu của con trỏ) ở giữa hai con trỏ.

Chú ý là phép toán cộng hai con trỏ và nhân chia, lấy phần dư trên con trỏ là không hợp lệ. Mỗi khi con trỏ được tăng giá trị, nó sẽ trỏ đến ô nhớ của phần tử kế tiếp. Mỗi khi nó được giảm giá trị, nó sẽ trỏ đến vị trí của phần tử đứng trước nó.

Xét ví dụ sau:

```
int x, *p1, *p2;
p1= &x;
p2= p1+1;
```

Khi đó p2 sẽ trỏ tới số nguyên nằm ngay kề sau x trong bộ nhớ. Cũng phải chú ý rằng mặc dù 1 chỉ là khoảng cách tương đối tính theo số phần tử kiểu int giữa p1 và p2, còn thực tế giá trị địa chỉ theo byte của p2 hơn p1 là 2. Nếu p2, p1 là con trỏ kiểu float thì khoảng cách sẽ là 4, chính là kích thước của kiểu dữ liệu trỏ bởi con trỏ.

### III.4.2.4. Con trỏ void

Được khai báo như sau:

```
void  *con_trỏ;
```

Đây là con trỏ đặc biệt, con trỏ không có kiểu, nó có thể nhận giá trị là địa chỉ của một biến thuộc bất kỳ kiểu dữ liệu nào. Con trỏ void được dùng làm đối để nhận bất kỳ địa chỉ nào từ tham số của các lời gọi hàm. Các lệnh sau đây là hợp lệ:

```
void *p, *q;
int x = 21;
float y = 34.34;
p = &x; q = &y;
```

### III.4.2.5. Mối quan hệ giữa con trỏ và mảng một chiều

Với một mảng một chiều có tên là a thì a là một địa chỉ và a có giá trị bằng &a[0].

Như vậy nếu ta khởi tạo một con trỏ p với giá trị bằng địa chỉ của một mảng, ta có thể dùng con trỏ này để duyệt qua các phần tử trong mảng.

Xét khai báo mảng tab gồm 10 số nguyên sau:

```
int a[10], *p;
```

nếu p = a; khi đó:

p+1 sẽ trỏ tới phần tử cùng kiểu ngay sau phần tử đầu tiên của mảng, chính là a[1], nghĩa là *(p+1) chính là a[1].

p+2 sẽ trỏ tới a[2], hay *(p+2) là a[2]

....

p+i  sẽ trỏ tới a[i]

Ví dụ sau đây minh họa việc sử dụng con trỏ để duyệt mảng một chiều:

```
#include<stdio.h>
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
void main()
{
```

```
    int i;
    int *p;
    p = tab;
    for (i = 0; i < N; i++)
     {
          printf(" %d \n",*p);   //*p là tab[0], trỏ đến phần tử đầu tiên của mảng -> lấy giá trị của phần tử
          p++;                   // p++ = p+1 -> *(p+1) trỏ đến a[1] -> tăng chỉ số của của phần tử và trỏ vào phần tử thứ 2.
     }
}
```

Ngoài ký pháp *(p+i) để chỉ nội dung phần tử thứ i tính từ đầu mảng (trỏ bởi p), ngôn ngữ C còn cho phép chúng ta sử dụng chỉ số i trên con trỏ. Cụ thể là p[i] có vai trò như *(p+i) và cũng là phần tử thứ i của mảng được trỏ bởi p. Ví dụ trước có thể được viết lại như sau:

```
#include<stdio.h>
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
main()
{
    int i;
    int *p;
    p = tab;
    for (i = 0; i < N; i++)
        printf(" %d \n", p[i]);
}
```

Lưu ý:

- Con trỏ luôn cần phải được khởi tạo, hoặc bằng cách gán cho nó một địa chỉ nào đó, hoặc qua thao tác cấp phát động bộ nhớ.
- Một (tên) mảng là một hằng địa chỉ, nó không bao giờ có thể nằm ở vế trái của một phép gán như con trỏ, cũng như chấp nhận các phép toán số học trên nó, ví dụ như tab++;

## III.4.3. Xâu ký tự

### III.4.3.1. Khái niệm xâu ký tự

Xâu kí tự (*string*): Là một dãy các kí tự viết liên tiếp nhau.

Xâu rỗng: Là xâu không gồm kí tự nào cả.

Độ dài xâu: Là số kí tự có trong xâu.

Biểu diễn xâu kí tự: Xâu kí tự được biểu diễn bởi dãy các kí tự đặt trong cặp dấu ngoặc kép. Các kí tự nằm trong cặp dấu ngoặc kép là nội dung của xâu.

Ví dụ:

- "String" là một xâu kí tự gồm 6 kí tự: 'S', 't', 'r', 'i', 'n', 'g' được viết liên tiếp nhau.
- "Tin hoc" là một xâu kí tự gồm 7 kí tự: 'T', 'i', 'n', dấu cách (' '), 'h', 'o', và 'c'.

Lưu trữ dữ liệu kiểu xâu kí tự: Các kí tự của xâu được lưu trữ kế tiếp nhau và kết thúc bằng kí tự kết thúc xâu (kí tự '\0' hay kí tự NUL, có số thứ tự 0 trong bảng mã ASCII). Nhờ có kí tự NUL mà người ta xác định được độ dài của xâu kí tự bằng cách đếm các kí tự có trong xâu đến khi gặp kí tự NUL (kí tự NUL không được tính vào độ dài xâu).

Ví dụ: Xâu kí tự "Tin hoc" sẽ được lưu trữ như sau:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 'T' | 'i' | 'n' | ' ' | 'h' | 'o' | 'c' | '\0' |

*Lưu ý*:

- Xâu kí tự khác mảng kí tự ở chỗ xâu kí tự có kí tự kết thúc xâu (kí tự NUL hay '\0') trong khi mảng kí tự không có kí tự kết thúc.
- Phân biệt giữa một kí tự và xâu kí tự có một kí tự: Ví dụ 'A' là một kí tự, nó được lưu trữ trong 1 byte, còn "A" là xâu kí tự, nó được lưu trữ trong 2 byte, trong đó byte đầu tiên lưu trữ kí tự 'A', byte thứ 2 lưu trữ kí tự kết thúc xâu (NUL).

### III.4.3.2. Khai báo và sử dụng xâu

#### *a. Khai báo xâu kí tự*

Trong C, một xâu kí tự được khai báo với cú pháp như sau:

***char tên_xâu [số_kí_tự_tối_đa];***

Trong đó số_kí_tự_tối_đa cho biết số lượng kí tự nhiều nhất có thể có trong xâu.

Sau khi khai báo, biến xâu kí tự tên_xâu có thể được dùng để lưu trữ một xâu kí tự bất kì, miễn là độ dài xâu kí tự (số kí tự có trong xâu) đó không vượt quá giá trị số_kí_tự_tối_đa.

Ví dụ:

```
char ho_va_ten[20];
```

Đây là khai báo của một biến xâu kí tự tên là ho_va_ten, biến này có thể có tối đa 20 kí tự.

Lưu ý: Đôi khi ta vẫn có thể nhập một xâu có nhiều hơn 20 kí tự cho xâu ho_va_ten mà trình biên dịch C vẫn không báo lỗi, tuy nhiên cần tránh điều này vì khi chạy chương trình thì chương trình quản lí bộ nhớ của hệ điều hành sẽ bắt lỗi và buộc chương trình kết thúc.

#### *b. Truy nhập vào một phần tử của xâu*

Có thể truy nhập đến từng phần tử của xâu tương tự như truy nhập đến từng phần tử của mảng. Cú pháp sử dụng để truy nhập là

**tên_xâu[chỉ_số_của_kí_tự_cần_truy_nhập]**

Ví dụ: Ta đã có khai báo char que_quan[10], và giả sử xâu que_quan có nội dung là "Ha Noi". Khi đó ta có thể hình dung xâu kí tự que_quan như sau :

| Phần tử thứ | Chỉ số của phần tử | Tên của phần tử | Nội dung lưu trữ |
|---|---|---|---|
| 1 | 0 | que_quan[0] | 'H' |
| 2 | 1 | que_quan[1] | 'a' |
| 3 | 2 | que_quan[2] | ' ' |
| 4 | 3 | que_quan[3] | 'N' |
| 5 | 4 | que_quan[4] | 'o' |
| 6 | 5 | que_quan[5] | 'i' |

| | | | |
|---|---|---|---|
| 7 | 6 | que_quan[6] | '\0' |
| 8 | 7 | que_quan[7] | |
| 9 | 8 | que_quan[8] | |
| 10 | 9 | que_quan[9] | |

### III.4.3.3. Các hàm xử lý ký tự

Lưu ý: Để sử dụng các hàm này ta khai báo tệp tiêu đề ctype.h.

Hàm toupper():

***int toupper(int ch)***

Hàm toupper() dùng để chuyển một kí tự chữ cái thường (các kí tự 'a', 'b', ..., 'z') thành kí tự chữ cái hoa tương ứng ('A', 'B', ..., 'Z').

Hàm tolower():

***int tolower(int ch)***

Hàm tolower() dùng để chuyển một kí tự chữ cái hoa ('A', 'B', ..., 'Z') thành kí tự chữ cái thường tương ứng ('a', 'b', ...'z').

Hàm isalpha():

***int isalpha(int ch)***

Hàm isalpha() dùng để kiểm tra một kí tự có phải là chữ cái hay không ('a', 'b', ..., 'z', 'A', 'B', ..., 'Z'). Hàm trả về giá trị khác không nếu đúng là chữ cái, trả về giá trị 0 nếu ngược lại.

Hàm isdigit():

***int isdigit(int ch)***

Hàm isdigit() dùng để kiểm tra một kí tự có phải là chữ số hay không ('0', '1', ...'9'). Hàm trả về giá trị khác không nếu đúng, trả về giá trị 0 nếu ngược lại.

Hàm islower():

***int islower(int ch)***

Hàm islower() dùng để kiểm tra một kí tự có phải là chữ cái thường hay không ('a', 'b', ...'z'). Hàm trả về giá trị khác không nếu đúng, trả về giá trị 0 nếu ngược lại.

Hàm isupper():

***int isupper(int ch)***

Hàm isupper() dùng để kiểm tra một kí tự có phải là chữ cái hoa hay không ('A', 'B', ...'Z'). Hàm trả về giá trị khác không nếu đúng, trả về giá trị 0 nếu ngược lại.

Hàm iscntrl():

***int iscntrl(int ch)***

Hàm iscntrl() dùng để kiểm tra một kí tự có phải là kí tự điều khiển hay không (là các kí tự không hiển thị được và có mã ASCII từ 0 đến 31). Hàm trả về giá trị khác không nếu đúng, trả về giá trị 0 nếu ngược lại.

Hàm isspace():

***int isspace(int ch)***

Hàm isspace() dùng để kiểm tra một kí tự có phải là dấu cách (space, mã ASCII là 32), kí tự xuống dòng ('\n', mã ASCII là 10), kí tự về đầu dòng ('\r', mã ASCII là 13), dấu tab ngang ('\t', mã ASCII là 9) hay dấu tab dọc ('\v', mã ASCII là 11) hay không. Hàm trả về giá trị khác không nếu đúng, trả về giá trị 0 nếu ngược lại.

## III.4.3.4. Các hàm xử lý xâu

### *Vào ra dữ liệu*

Vào ra dữ liệu trên xâu kí tự tức là nhập dữ liệu cho xâu và hiển thị dữ liệu chứa trong xâu.

Để nhập dữ liệu cho xâu ta có thể sử dụng 2 hàm scanf() hoặc gets()

***scanf("%s",tên_xâu);***

***gets(tên_xâu);***

Để hiển thị nội dung xâu ta có thể dùng 2 hàm printf() hoặc puts()

***printf("%s",tên_xâu);***

***puts(tên_xâu);***

Các hàm scanf(), gets, printf(), puts() được khai báo trong tệp tiêu đề stdio.h.

Ví dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
char Ten[12];
printf("Nhap chuoi: ");gets(Ten);
printf("Chuoi vua nhap: ");puts(Ten);
getch();
return 0;
}
```

### *Một số hàm xử lí xâu kí tự khác*

Hàm strlen():

***size_t strlen(char* tên_xâu);***

Hàm trả về độ dài (số kí tự có trong xâu) của xâu kí tự tên_xâu.

Hàm strcpy():

***char* strcpy(char* xâu_đích, char* xâu_nguồn);***

Hàm này sẽ sao chép nội dung xâu_nguồn và ghi lên xâu_đích.

Hàm strcmp():

***int strcmp(char* xâu_thứ_nhất, char* xâu_thứ_hai);***

Hàm strcmp trả về

- giá trị < 0 nếu xâu_thứ_nhất nhỏ hơn xâu_thứ_hai
- giá trị 0 nếu xâu_thứ_nhất bằng xâu_thứ_hai
- giá trị > 0 nếu xâu_thứ _nhất lớn hơn xâu_thứ_hai

Hàm strcat():

***char* strcat(char* xâu_đích, char* xâu_nguồn);***

Hàm strcat sẽ ghép nối xâu_nguồn vào ngay sau xâu_đích. Kết quả trả về của hàm strcat() là xâu mới ghép nối từ 2 xâu xâu_nguồn và xâu_đích.

Hàm strchr():

***char* strchr(char* str, int ch);***

Hàm strchr() dùng để tìm kiếm vị trí của kí tự ch trong xâu str. Nếu có kí tự ch trong str thì hàm strchr() trả về con trỏ trỏ tới kí tự ch đầu tiên trong str, ngược lại nó sẽ trả về con trỏ NULL.

Hàm strstr():

***char* strstr(char* str1, char* str2);***

Hàm strstr() dùng để tìm kiếm vị trí của xâu con str2 trong xâu str1. Nếu str2 là xâu con của str1 thì hàm strstr() trả về con trỏ trỏ tới kí tự đầu tiên của xâu con str2 đầu tiên trong str1, ngược lại nó sẽ trả về con trỏ NULL.

Hàm atoi():

***int atoi(char* str);***

Hàm atoi() dùng để chuyển một xâu kí tự là biểu diễn của một số nguyên thành số nguyên tương ứng. Nếu chuyển đổi thành công, hàm atoi() trả về giá trị số nguyên chuyển đổi được, ngược lại trả về giá trị 0.

Hàm atol():

***long int atol(char* str);***

Hàm atol() dùng để chuyển một xâu kí tự là biểu diễn của một số nguyên dài (tương ứng với kiểu dữ liệu **long int**) thành số nguyên dài tương ứng. Nếu chuyển đổi thành công, hàm atol() trả về giá trị số nguyên chuyển đổi được, ngược lại trả về giá trị 0.

Hàm atof():

***double atof(char* str);***

Hàm atof() dùng để chuyển một xâu kí tự là biểu diễn của một số thực (ở cả dạng số dấu phẩy tĩnh và động đều được) thành số thực tương ứng. Nếu chuyển đổi thành công, hàm atof() trả về giá trị số thực chuyển đổi được, ngược lại trả về giá trị 0.

Các hàm strcpy(), strlen(),strcmp(), strcat(), strchr(), strstr() khai báo trong tệp tiêu đề string.h.

Các hàm atoi(), atol(), atof() khai báo trong tệp tiêu đề stdlib.h.

Ví dụ minh họa:

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

```
#include <conio.h>
#include <string.h>     // Phai co thu vien string.h thi moi
                        // su dung duoc cac ham strcpy, strcmp...
void main()
{
    char str1[10] = "abc";
    char str2[10] = "def";
    clrscr();
    printf(" str1: %s",str1);
    printf("\n str2: %s",str2);
    printf("\n strcmp(str1,str2) = %d",strcmp(str1,str2));
    printf("\n strcat(str1,str2) = %s",strcat(str1,str2));
    printf("\n str1: %s",str1);
    printf("\n str2: %s",str2);
    printf("\n strcpy(str1,str2) = %s",strcpy(str1,str2));
    printf("\n str1: %s",str1);
    printf("\n str2: %s",str2);
    strcpy(str1,"ab");
    strcpy(str2,"abc");
    printf("\n strcmp(str1,str2) = %d",strcmp(str1,str2));
    getch();
}
```

Kết quả:

```
str1: abc
str2: def
strcmp(str1,str2) = -3
strcat(str1,str2) = abcdef
str1: abcdef
str2: def
strcpy(str1,str2) = def
str1: def
str2: def
strcmp(str1,str2) = -3
```

Cần lưu ý khi sử dụng các hàm strcat(), strcpy(),.. là kích thước bộ nhớ lưu trữ dành cho xâu đích phải đủ để chứa kết quả thu được sau lời gọi các hàm trên.

# III.5. Cấu trúc

## III.5.1. Khái niệm cấu trúc

Kiểu dữ liệu cấu trúc (**struct**) là kiểu dữ liệu phức hợp bao gồm nhiều thành phần, mỗi thành phần có thể thuộc những kiểu dữ liệu khác nhau.

Các thành phần dữ liệu trong cấu trúc được gọi là các trường dữ liệu (*field*).

Ví dụ: Khi cần lưu giữ thông tin về một dạng đối tượng nào đó như đối tượng sinh viên chẳng hạn, ta lưu giữ các thông tin liên quan đến sinh viên như họ tên, tuổi, kết quả học tập... Mỗi thông tin thành phần lại có kiểu dữ liệu khác nhau như họ tên có kiểu dữ liệu là xâu kí tự, tuổi có kiểu dữ liệu là số nguyên, kết quả học tập có kiểu dữ liệu là số thực.

## III.5.2. Khai báo và sử dụng cấu trúc

### III.5.2.1. Khai báo kiểu dữ liệu cấu trúc

Để khai báo một kiểu dữ liệu cấu trúc ta dùng cú pháp khai báo sau:

***struct tên_cấu_trúc***

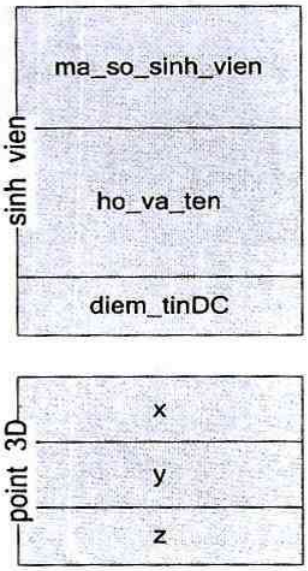
***{***

***<khai báo các trường dữ liệu>;***

***};***

Ví dụ:

```
struct sinh_vien
{      //3 trường dữ liệu
char ma_so_sinh_vien[10];
char ho_va_ten[30];
float diem_TinDC;
}
struct point_3D
{
float x;
float y;
float z;
}
```



Khai báo thứ nhất là khai báo của một kiểu dữ liệu cấu trúc có tên là sinh_vien gồm có 3 trường dữ liệu là ma_so_sinh_vien kiểu xâu kí tự, ho_va_ten kiểu xâu kí tự và diem_TinDC kiểu số thực **float**.

Ở khai báo thứ 2 ta đã khai báo một kiểu dữ liệu cấu trúc có tên là point_3D gồm có 3 trường và cả 3 trường này đều có cùng kiểu dữ liệu số thực **float**. Vì 3 trường này cùng kiểu dữ liệu nên ta có thể sử dụng mảng để thay thế cấu trúc, tuy nhiên việc sử dụng cấu trúc để mô tả dữ liệu điểm 3 chiều sẽ giúp sự mô tả được tự nhiên hơn, "thật" hơn (nghĩa là trong cấu trúc các tọa độ vẫn độc lập với nhau, nhưng nhìn từ bên ngoài thì các tọa độ này lại là một thể thống nhất cung cấp thông tin về vị trí của một điểm. Còn nếu sử dụng mảng thì các tọa độ của một điểm độc lập và rời rạc với nhau, không thấy mối liên hệ. Khi đó ta sẽ phải tự mình ngầm quy ước rằng các phần tử của mảng có chỉ số là $3*k$, $3*k+1$ và $3*k+2$ với $k = 0, 1, 2, \ldots$ là tọa độ của một điểm).

### III.5.2.2. Khai báo biến cấu trúc

Để khai báo biến cấu trúc ta dùng cú pháp khai báo sau:

***struct tên_cấu_trúc tên_biến_cấu_trúc;***

Ví dụ:

```
struct sinh_vien a, b, c;
```

Câu lệnh trên khai báo 3 biến lần lượt tên là a, b, c có kiểu dữ liệu là cấu trúc sinh_vien.

Tuy nhiên ta cũng có thể kết hợp đồng thời vừa khai báo kiểu dữ liệu cấu trúc vừa khai báo biến cấu trúc bằng cách sử dụng cú pháp khai báo sau:

```
struct [tên_cấu_trúc]
{
    <khai báo các trường>;
} tên_biến_cấu_trúc;
```

Theo cú pháp khai báo trên thì ta thấy phần tên_cấu_trúc có thể có hoặc không. Nếu có tên_cấu_trúc thì sau này ta có thể khai báo bổ sung biến có kiểu dữ liệu là tên_cấu_trúc đó, còn nếu không có tên_cấu_trúc thì cấu trúc khai báo tương ứng không được sử dụng về sau.

Ví dụ:

```
struct diem_thi
{
    float diem_Toan;
    float diem_Ly;
    float diem_Hoa;
}
struct thi_sinh
{
    char SBD[10];       // so bao danh
    char ho_va_ten[30];
    struct diem_thi ket_qua;
} thi_sinh_1, thi_sinh_2;
```

Qua ví dụ trên ta cũng thấy rằng các cấu trúc có thể lồng nhau, nghĩa là cấu trúc này có thể là trường dữ liệu của cấu trúc khác và mức độ lồng là không hạn chế. Để tăng thêm sự tiện lợi, ngôn ngữ C còn cho phép khai báo trực tiếp trường dữ liệu là cấu trúc bên trong cấu trúc chứa nó, vì thế ta có thể viết lại ví dụ ở trên như sau:

```
struct thi_sinh
{
    char SBD[10];
    char ho_va_ten[30];
    struct diem_thi
      {
          float diem_Toan;
          float diem_Ly;
          float diem_Hoa;
    }ket_qua;
} thi_sinh_1, thi_sinh_2;
```

### III.5.2.3. Định nghĩa kiểu dữ liệu cấu trúc với typedef

Trong các ví dụ trước ta thấy một khai báo biến cấu trúc bắt đầu bằng từ khóa **struct**, sau đó đến tên cấu trúc rồi mới đến tên biến. Cách khai báo này có phần "rắc rối" hơn so với khai báo biến thông thường và có không ít trường hợp người lập trình quên đặt từ khóa **struct** ở đầu. Để tránh điều đó, ngôn ngữ C cho phép ta đặt tên mới cho kiểu dữ liệu cấu trúc bằng câu lệnh **typedef** có cú pháp khai báo như sau:

***typedef struct tên_cũ tên_mới;***

Hoặc ta có thể đặt lại tên cho cấu trúc ngay khi khai báo bằng cú pháp:

***typedef struct [tên_cũ]***

***{***

***<khai báo các trường>;***

***}danh_sách_các_tên_mới;***

Sau câu lệnh này ta có thể sử dụng tên_mới thay cho tổ hợp struct tên_cũ khi khai báo biến.

*Lưu ý*: Được phép đặt tên_mới trùng với tên_cũ.

Ví dụ:

```
struct point_3D
{
    float x, y, z;
} P;
struct point_3D M;
typedef struct point_3D point_3D;
point_3D N;
```

Trong ví dụ trên ta đã đặt lại tên cho cấu trúc struct point_3D thành point_3D và dùng tên mới này làm kiểu dữ liệu cho khai báo của biến N. Các biến P, M được khai báo theo cách chúng ta đã biết.

Ví dụ:

```
typedef struct point_2D
{
    float x, y;
}point_2D, diem_2_chieu, ten_bat_ki;
point_2D X;
diem_2_chieu Y;
ten_bat_ki Z;
```

Với ví dụ này ta cần chú ý là point_2D, diem_2_chieu và ten_bat_ki không phải là tên biến mà là tên mới của cấu trúc struct point_2D. Các biến X, Y, Z được khai báo với kiểu dữ liệu là các tên mới này.

## III.5.3. Xử lý dữ liệu cấu trúc

### III.5.3.1. Truy nhập các trường dữ liệu của cấu trúc

Dữ liệu của một biến cấu trúc bao gồm nhiều trường dữ liệu, và các trường dữ liệu này độc lập với nhau. Do đó muốn thay đổi nội dung dữ liệu bên trong một biến cấu trúc ta cần truy nhập tới từng trường và thực hiện thao tác cần thiết trên từng trường đó. Để truy nhập tới một trường trong cấu trúc ta dùng cú pháp sau:

***tên_biến_cấu_trúc.tên_trường***

Dấu chấm "." sử dụng trong cú pháp trên là toán tử truy nhập thành phần cấu trúc, Nếu trường được truy nhập lại là một cấu trúc, thì ta có thể tiếp tục áp dụng toán tử này để truy nhập tới các trường thành phần nằm ở mức sâu hơn.

Giờ đây ta có thể "đối xử" tên_biến_cấu_trúc.tên_trường giống như một biến thông thường có kiểu dữ liệu là kiểu dữ liệu của tên_trường, tức là ta có thể nhập giá trị, hiển thị giá trị của biến cấu trúc, sử dụng giá trị đó trong các biểu thức…

Ví dụ:

```
// duoi day la mot cau truc mo ta mot diem trong khong gian 2 chieu.
// cac truong du lieu gom: ten cua diem va toa do cua diem do.
// toa do la mot cau truc gom 2 truong: hoanh do va tung do
#include <stdio.h>
#include <conio.h>
void main()
{
    struct point_2D
    {
        char ten_diem;
        struct
          {
            float x, y;
          } toa_do;
    } p;
    float temp_float;
    char temp_char;
    printf("\n Hay nhap thong tin ve mot diem");
    printf("\n Ten cua diem: ");
        fflush(stdin);
    scanf("%c",&temp_char);
```

```
        p.ten_diem = temp_char;
    printf("\n nhap vao hoanh do cua diem: ");
        scanf("%f",&temp_float);
    p.toa_do.x = temp_float;
        // gia su diem dang xet nam tren duong thang y = 3x + 2;
    p.toa_do.y = 3*p.toa_do.x + 2;
    printf("\n %c = (%5.2f,%5.2f)",p.ten_diem, p.toa_do.x, p.toa_do.y);
    getch();
}
```

Kết quả khi chạy chương trình:

```
Hay nhap thong tin ve mot diem
Ten cua diem: A
nhap vao hoanh do cua diem: 5
A = ( 5.00,17.00)
```

*Lưu ý*: Cũng như việc nhập giá trị cho các phần tử của mảng, việc nhập giá trị cho các trường của cấu trúc (đặc biệt là các trường có kiểu dữ liệu **float**) nên thực hiện qua biến trung gian để tránh những tình huống có thể làm treo máy hoặc kết quả nhập được không như ý.

### III.5.3.2. Phép gán giữa các biến cấu trúc

Giả sử ta có 2 biến cấu trúc là *a* và *b* có cùng kiểu dữ liệu là một cấu trúc nào đó và giả sử các trường dữ liệu của *a* đều đã được khởi tạo (tức là giá trị của các trường dữ liệu đó đều đã được xác định). Giờ đây ta cũng muốn giá trị các trường dữ liệu của *b* có giá trị giống với các trường dữ liệu tương ứng của *a*. Ta có thể thực hiện điều đó bằng cách gán giá trị từng trường của *a* cho các trường tương ứng của *b*. Cách này có vẻ rất "thủ công" và rất bất tiện nếu như trong cấu trúc có nhiều trường dữ liệu. Do vậy C cung cấp cho ta một phương tiện để thực hiện việc này theo cách thứ hai, đó là sử dụng phép gán các biến cấu trúc. Phép gán cấu trúc có cú pháp tương tự như phép gán thông thường.

***biến_cấu_trúc_1 = biến_cấu_trúc_2;***

Câu lệnh trên sẽ gán giá trị của các trường trong biến_cấu_trúc_2 cho các trường tương ứng trong biến_cấu_trúc_1.

Ví dụ:

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

```
#include <conio.h>
#include <string.h>
void main()
{
    struct s
    {
        char ho_ten[20];
        float diem;
    }a, b, c;
    float temp_f;
    printf("\na.ho_ten: ");fflush(stdin);
    gets(a.ho_ten);
    printf("\na.diem = ");scanf("%f",&temp_f);
    a.diem = temp_f;
    strcpy(c.ho_ten, a.ho_ten);
    c.diem = a.diem;
    b = a;
    printf("\na: %20s %5.2f", a.ho_ten, a.diem);
    printf("\nb: %20s %5.2f", b.ho_ten, b.diem);
    printf("\nc: %20s %5.2f\n", c.ho_ten, c.diem);
}
```

Kết quả thực hiện

```
a.ho_ten: nguyen van minh
a.diem = 7.5
a:     nguyen van minh  7.50
b:     nguyen van minh  7.50
c:     nguyen van minh  7.50
```

Trong chương trình trên ta đã nhập giá trị cho các trường của biến cấu trúc a từ bàn phím, sau đó chép dữ liệu từ biến a sang biến c bằng cách sao chép từng trường và chép dữ liệu từ biến a sang biến b bằng cách dùng lệnh gán. Kết quả là như nhau và rõ ràng cách thứ hai ngắn gọn hơn.

*Lưu ý* : Để chép dữ liệu là xâu kí tự ta phải dùng lệnh strcpy(), không được dùng lệnh gán thông thường để chép nội dung xâu kí tự.

# III.6. Hàm

## III.6.1. Khái niệm hàm

### III.6.1.1. Khái niệm chương trình con

Trong khi lập trình chúng ta thường gặp những đoạn chương trình lặp đi lặp lại nhiều lần ở những chỗ khác nhau. Để tránh rườm rà và tiết kiệm thời gian, chương trình đó được thay thế bằng các chương trình con tương ứng và khi cần ta chỉ việc gọi những chương trình con đó ra mà không phải viết lại cả đoạn chương trình đó.

Lấy ví dụ khi giải các bài toán lượng giác ta thường xuyên cần phải tính giá trị sin của biến lượng giác $x$ nào đó. Như vậy ta nên lập một chương trình con tên là sin và tham số là $x$ để tính giá trị sin($x$). Mỗi khi cần tính toán giá trị sin của một biến $y$ nào đó, ta chỉ cần gọi chương trình con sin đã lập sẵn và truyền giá trị biến $y$ làm tham số. Khi đó ta vẫn thu được kết quả mong muốn mà không phải viết lại cả đoạn chương trình tính giá trị sin($y$).

Bên cạnh chương trình con sin còn có rất nhiều chương trình con khác được tạo sẵn như cos, exp (dùng để tính lũy thừa cơ số e), pow (tính lũy thừa), sqrt (tính căn bậc 2), ... giúp người lập trình tính toán giá trị của các đại lượng thông dụng. Những chương trình con này nằm trong *thư viện các chương trình con mẫu* và được trình biên dịch C quản lý, vì vậy chúng còn được gọi là các chương trình con chuẩn. Trình biên dịch Turbo C++ phân loại và đặt khai báo các chương trình con chuẩn này trong các đơn vị chương trình khác nhau dưới dạng các tệp tiêu đề như stdio.h, conio.h, math.h, string.h...

Ngoài ra còn có một lý do khác cần đến chương trình con. Khi ta giải quyết một bài toán lớn thì chương trình của ta có thể rất lớn và dài, điều này làm cho việc sửa chữa, gỡ rối, hiệu chỉnh chương trình gặp nhiều khó khăn. Nhưng nếu ta chia bài toán lớn, phức tạp ban đầu thành các bài toán con nhỏ hơn và tương đối độc lập với nhau, rồi lập các chương trình con giải quyết từng bài toán con, cuối cùng ghép các chương trình con đó lại thành một chương trình giải quyết bài toán ban đầu thì sẽ rất tiện lợi cho việc phát triển, kiểm tra và sửa chữa cả chương trình.

Việc này tương tự như trong dây chuyền sản xuất công nghiệp, khi ta lắp ráp sản phẩm hoàn thiện từ các bán thành phẩm, các mô đun được chế tạo ở những nơi khác nhau. Vì các bán thành phẩm này được chế tạo độc lập nên khi phát hiện lỗi ở mô đun nào ta chỉ việc tìm đến nơi sản xuất ra nó để sửa chữa.

Việc chia nhỏ một chương trình thành các chương trình con đảm nhận những công việc nhỏ khác nhau chính là tư tưởng chính cho phương pháp lập trình có cấu trúc (*structured programming*).

Cần lưu ý là có khi một chương trình con chỉ sử dụng đúng một lần nhưng nó vẫn làm cho chương trình trở nên sáng sủa và dễ đọc, dễ hiểu hơn.

### III.6.1.2. Phân loại chương trình con

Có 2 loại chương trình con là hàm (*function*) và thủ tục (*procedure*). Sự khác nhau giữa hàm và thủ tục là ở chỗ hàm sau khi thực hiện xong thì sẽ trả về giá trị, còn thủ tục không trả về giá trị.

Mặc dù vậy hàm và thủ tục là tương đương nhau, tức là có thể xây dựng được thủ tục có chức năng tương đương với một hàm bất kì và có thể xây dựng được hàm có chức năng tương đương với một thủ tục bất kì. Vì thế có những ngôn ngữ lập trình cho phép chương trình con có thể là hàm và thủ tục (Pascal) và có những ngôn ngữ chỉ cho phép chương trình con là hàm mà thôi (như C, Java).

Lưu ý là nếu chương trình con là hàm thì nó luôn có giá trị trả về. Nếu thực sự không có giá trị gì để trả về (nghĩa là nó hoạt động giống thủ tục) thì ta phải khai báo hàm đó có kiểu giá trị trả về là "*không là kiểu giá trị nào cả*" (kiểu **void** trong C).

## III.6.2. Khai báo và sử dụng hàm

### III.6.2.1. Khai báo hàm

Cú pháp khai báo một hàm trong C là như sau:

***[<kiểu giá trị trả về>] <tên hàm>([<danh sách tham số>,...])***

***Thân hàm***

Khai báo của một hàm được chia làm 2 phần:

- Dòng đầu hàm:

  ***[<kiểu giá trị trả về>] <tên hàm>([<danh sách tham số>,...])***

- Thân hàm: là tập hợp các khai báo và câu lệnh đặt trong cặp dấu ngoặc nhọn.

```
{
        <Các khai báo>
        ...
        <Các câu lệnh>
}
```

Trong thân hàm có ít nhất một lệnh return.

Ví dụ sau khai báo và định nghĩa hàm tính giai thừa của một số nguyên dương. Ta quy ước rằng giai thừa của một số âm thì bằng –1, của 0 bằng 1, của một số nguyên dương $a$ là $a! = a \times (a-1) \times \ldots \times 1$.

```
int giai_thua(int a)
{
    int ket_qua;
    int i;

    ket_qua = 1;
    for(i = 1;i<= a;i++)
      ket_qua = ket_qua * i;
    if(a < 0) ket_qua = -1;
    return ket_qua;
}
```

Dòng đầu hàm

Các khai báo

Các câu lệnh

## Các thành phần của dòng đầu hàm

Dòng đầu hàm là các thông tin được trao đổi giữa bên trong và bên ngoài hàm. Khi nói tới dòng đầu hàm là nói tới tên của hàm, hàm đó cần những thông tin gì từ môi trường để hoạt động (các tham số đầu vào), hàm đó cung cấp những thông tin gì cho môi trường (những tham số đầu ra và giá trị trả về).

Dòng đầu hàm phân biệt các hàm với nhau, hay nói cách khác không được có hai hàm có dòng đầu hàm giống nhau.

*Kiểu dữ liệu trả về của hàm*

Thông thường hàm sau khi được thực hiện sẽ trả về một giá trị kết quả tính toán nào đó. Để sử dụng được giá trị đó ta cần phải biết nó thuộc kiểu dữ liệu gì. Kiểu dữ liệu của đối tượng tính toán được hàm trả về được gọi là kiểu dữ liệu trả về của hàm.

Trong C, kiểu dữ liệu trả về của hàm có thể là kiểu dữ liệu bất kỳ (kiểu dữ liệu có sẵn hoặc kiểu dữ liệu do người dùng tự định nghĩa) nhưng không được là kiểu dữ liệu mảng.

Nếu kiểu dữ liệu trả về là kiểu **void** thì hàm không trả về giá trị nào cả.

Trường hợp ta không khai báo kiểu dữ liệu trả về thì chương trình dịch của C sẽ ngầm hiểu rằng kiểu dữ liệu trả về của hàm là kiểu **int**.

*Tên hàm*

Tên hàm là có thể là bất kì một định danh hợp lệ nào, tuy nhiên tên hàm nên mang nghĩa gợi ý chức năng công việc mà hàm thực hiện. Ví dụ, một hàm có chức năng tính và trả về bình phương của một số thực *x* thì nên có tên là *binh_phuong*. Trong C, các hàm không được đặt tên trùng nhau.

*Tham số của hàm*

Tham số của hàm là các thông tin cần cho hoạt động của hàm và các thông tin, kết quả tính toán được hàm trả lại, tức là có những tham số chứa dữ liệu vào cung cấp cho hàm, có những tham số chứa dữ liệu ra mà hàm tính toán được.

Các tham số sử dụng trong lời khai báo hàm được gọi là tham số hình thức, nó là tham số giả định của hàm. Khi khai báo tham số hình thức, phải chỉ ra tên của tham số và kiểu dữ liệu của tham số.

Các tham số được cung cấp trong quá trình thực hiện hàm được gọi là tham số thực. Kiểu dữ liệu của tham số thực cung cấp cho hàm trong chương trình phải giống kiểu dữ liệu của tham số hình thức tương ứng với tham số thực đó và có cùng số lượng, nếu không sẽ có báo lỗi biên dịch.

Một hàm có thể có một, nhiều hoặc không có tham số nào cả, nếu có nhiều tham số thì chúng phải được phân cách với nhau bằng dấu phẩy. Lưu ý là nếu hàm không có tham số nào cả thì vẫn phải có cặp dấu ngoặc đơn sau tên hàm, ví dụ **main()**.

*Lệnh **return***

Một hàm được thực hiện khi ta gặp lời gọi hàm của hàm đó trong chương trình. Một lời gọi hàm là tên hàm theo sau bởi các tham số thực trong chương trình. Sau khi hàm thực hiện xong, nó sẽ trở về chương trình đã gọi. Có hai cách để từ hàm trở về chương trình đã gọi hàm:

- Sau khi thực hiện tất cả các câu lệnh có trong thân hàm.
- Khi gặp lệnh **return**.

Cú pháp chung của lệnh **return** là:

*return **biểu_thức**;*

Khi gặp lệnh này, chương trình sẽ tính toán giá trị của ***biểu_thức***, lấy kết quả tính toán được làm giá trị trả về cho lời gọi hàm rồi kết thúc việc thực hiện hàm, trở về chương trình đã gọi nó. Lưu ý: Kiểu của biểu_thức phải trùng với kiểu giá trị trả về trong phần khai báo hàm.

Trong lệnh **return** cũng có thể không có phần ***biểu_thức***, khi đó ta sẽ kết thúc thực hiện hàm mà không trả về giá trị nào cả.

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
int max(int x, int y, int z)
{
    int max;
    max = x>y?x:y;
    max = max>z?max:z;
    return max;
}
void main()
{
    int a,b,c;
    clrscr();
    printf("\n Nhap gia tri cho 3 so nguyen a, b, c: ");
    scanf("%d %d %d",&a,&b,&c);
    printf("\n Gia tri cac so vua nhap: ");
    printf(" a = %-5d b = %-5d c = %-5d",a,b,c);
    printf("\n Gia tri lon nhat trong 3 so la %d",max(a,b,c));
    getch();
}
```

Kết quả khi thực hiện:

```
Nhap gia tri cho 3 so nguyen a, b, c: 3 12 8
Gia tri cac so vua nhap: a = 3    b = 12    c = 8
Gia tri lon nhat trong 3 so la 12
```

### III.6.2.2. Sử dụng hàm

Có thể sử dụng hàm sau khi đã khai báo. Để sử dụng một hàm (hay còn nói là *gọi hàm*) ta sử dụng cú pháp sau:

***<tên hàm> ([danh sách các tham số])***

Ví dụ: Chương trình dưới đây sẽ khai báo và định nghĩa một hàm có tên là Uscln với 2 tham số đều có kiểu **unsigned int**. Hàm Uscln tìm ước số chung lớn nhất của 2 tham số này theo thuật toán *Euclid* và trả về ước số chung tìm được. Sau đó ta sẽ gọi hàm Uscln trong hàm **main** để tìm ước số chung lớn nhất của hai số nguyên được nhập từ bàn phím.

```
#include <stdio.h>
#include <conio.h>
unsigned int Uscln(unsigned int a, unsigned int b)
{
    unsigned int u;
    if (a<b)
    {
        u = a; a = b; b = u;
    }
    do
    {
        u = a%b;
        a = b;
        b = u;
    }while (u!=0);
    return a;
}

int main()
{
    unsigned int a, b;
    do
    {
        printf("\n Nhap vao 2 so nguyen duong a va b ");
        printf("\n a = "); scanf("%d",&a);
        printf("\n b = "); scanf("%d",&b);
        if(a*b == 0)
        {
          printf("\n Khong hop le");
```

```
            continue;
        }
        printf("\n Uoc chung lon nhat cua %d va %d la: %d", a, b, Uscln(a, b));
    }while ((a != 0)||(b != 0));
    printf("\n An phim bat ki de ket thuc chuong trinh...");
    getch();
    return 0;
}
```

Kết quả khi thực hiện:

```
Nhap vao 2 so nguyen duong a va b
a = 6
b = 9
Uoc chung lon nhat cua 6 va 9 la: 3
Nhap vao 2 so nguyen duong a va b
a = 15
b = 26
Uoc chung lon nhat cua 15 va 26 la: 1
Nhap vao 2 so nguyen duong a va b
a = 3
b = 0
Khong hop le
Nhap vao 2 so nguyen duong a va b
a = 0
b = 0
Khong hop le
An phim bat ki de ket thuc chuong trinh...
```

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi, quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tương ứng*.
- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.
- Khi gặp lệnh **return** hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

## III.6.3. Truyền tham số trong lời gọi hàm

Thông thường trong các ngôn ngữ lập trình có hai cách để truyền tham số trong lời gọi hàm, đó là truyền theo tham trị và truyền theo tham biến.

*Truyền theo tham trị*

Khi tham số được truyền theo tham trị, một bản sao giá trị của tham số thực được tạo ra và gán cho tham số hình thức của hàm. Vì vậy, mọi thay đổi trong hàm sẽ không ảnh hưởng đến giá trị ban đầu của biến (nếu có) nằm trong lời gọi hàm.

*Truyền theo tham biến*

Khi tham số được truyền theo tham biến, hàm sẽ truyền trực tiếp tham số đó (bắt buộc phải là biến) cho hàm được gọi. Trong trường hợp này, tham số hình thức và tham số thực là một. Vì vậy, mọi thay đổi trong hàm sẽ ảnh hưởng đến giá trị ban đầu của biến được truyền trong lời gọi hàm.

Truyền theo tham trị được dùng khi không cần thay đổi giá trị của biến được truyền vào trong lời gọi hàm. Trong khi đó, truyền theo tham biến chỉ nên dùng khi thật sự cần thiết thay đổi giá trị của biến được truyền vào.

Trong ngôn ngữ C, cách truyênd tham số vào các hàm là truyền theo tham trị.

Dưới đây là một ví dụ về truyền tham số theo tham trị:

```
#include <stdio.h>
#include<conio.h>
void swap (int a, int b){
    int temp = a;
    a = b;
    b = temp;
}
void main(){
    int x = 3, y = 4;
    clrscr();
    printf("Truoc khi goi ham swap\n");
    printf("%3d %3d\n",x,y);
    swap(x,y);
    printf("Sau khi goi ham swap\n");
    printf("%3d %3d\n",x,y);
```

```
    getch();
}
```

Kết quả khi thực hiện:

```
Truoc khi goi ham swap
    3  4
Sau khi goi ham swap
    3  4
```

Để có thể thay đổi giá trị của tham số truyền vào, chúng ta sẽ khai báo tham số hình thức của hàm là con trỏ, khi đó tham số thực tương ứng sẽ là một địa chỉ. Khi truyền địa chỉ của biến cho hàm thì người lập trình có thể thay đổi giá trị của biến được truyền vào. Khi truyền tên của mảng cho hàm, chúng ta còn tiết kiệm được thời gian sao chép dữ liệu.

Ta chỉnh sửa lại chương trình trên:

```
#include <stdio.h>
#include<conio.h>
void swap (int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void main(){
    int x = 3, y = 4;
    clrscr();
    printf("Truoc khi goi ham swap\n");
    printf("%3d %3d\n",x,y);
    swap(&x,&y);
    printf("Sau khi goi ham swap\n");
    printf("%3d %3d\n",x,y);
    getch();
```

```
}
```

Kết quả khi thực hiện:

```
Truoc khi goi ham swap
    3  4
Sau khi goi ham swap
    4  3
```

## III.6.4. Phạm vi của biến

*Phạm vi của các biến*

Một biến sau khi khai báo thì có thể được sử dụng trong chương trình. Tuy nhiên tùy vào vị trí khai báo biến mà phạm vi sử dụng các biến sẽ khác nhau. Nguyên tắc sử dụng biến là biến khai báo trong phạm vi nào thì được sử dụng trong phạm vi đó.

Một biến có thể được khai báo trong chương trình chính hay trong các chương trình con hoặc thậm chí trong một lệnh khối. Nếu biến được khai báo trong một lệnh khối thì nó chỉ có thể được gọi trong lệnh khối đó, không thể gọi từ bên ngoài lệnh khối được. Một biến được khai báo trong một chương trình con chỉ có thể được sử dụng trong phạm vi chương trình con đó. Một biến được khai báo trong chương trình chính thì có thể được sử dụng trong toàn bộ chương trình, trong tất cả các chương trình con cũng như trong các lệnh khối của chương trình.

Lưu ý:

Một số ngôn ngữ lập trình như Pascal cho phép khai báo một chương trình con nằm trong một chương trình con khác, nhưng ngôn ngữ C không cho phép điều này.

Bên trong một lệnh khối thì có thể có chứa lệnh khối khác, khi đó biến được khai báo ở lệnh khối bên ngoài có thể được sử dụng ở lệnh khối bên trong.

Việc trùng tên của các biến: Trong cùng một phạm vi ta không được phép khai báo hai biến có cùng tên nhưng ta có thể khai báo hai biến trùng tên thuộc hai phạm vi khác nhau. Nếu có hai biến trùng tên khai báo ở hai phạm vi khác nhau thì xảy ra hai trường hợp:

– Hai phạm vi này tách rời nhau: Khi đó các biến sẽ có tác dụng ở phạm vi riêng của nó, không ảnh hưởng đến nhau.

– Phạm vi này nằm trong phạm vi kia: Khi đó nếu chương trình đang ở phạm vi ngoài (tức là đang thực hiện câu lệnh nằm ở phạm vi ngoài) thì biến khai báo ở

phạm vi ngoài có tác dụng, còn nếu chương trình đang ở phạm vi trong (đang thực hiện câu lệnh nằm ở phạm vi trong) thì biến khai báo ở phạm vi trong sẽ có tác dụng và nó che lấp biến trùng tên ở bên ngoài.

Ví dụ:

```
#include <stdio.h>
void main()
{
    {
        int a = 1;
        printf("\n a = %d",a);
        {
            int a = 2;
            printf("\n a = %d",a);
        }
        printf("\n a = %d",a);
    }
    {
        int a = 3;
        printf("\n a = %d",a);
    }
}
```

Kết quả thực hiện chương trình:

```
a = 1
a = 2
a = 1
a = 3
```

*Phân loại biến:*

Theo phạm vi sử dụng, biến chia làm hai loại: biến cục bộ (biến địa phương – local variable) và biến toàn cục (global variable).

**Biến địa phương**

Là các biến được khai báo trong lệnh khối hoặc trong thân chương trình con. Việc khai báo các biến cục bộ phải được đặt trước phần câu lệnh trong lệnh khối hoặc trong chương trình con.

**Biến toàn cục**

Là biến được khai báo trong chương trình chính. Vị trí khai báo của biến toàn cục là sau phần khai báo tệp tiêu đề và khai báo hàm nguyên mẫu.

Lưu ý:

– Hàm **main()** cũng chỉ là một chương trình con, nhưng nó là chương trình con đặc biệt ở chỗ chương trình được bắt đầu thực hiện từ hàm **main()**.

– Biến khai báo trong hàm **main()** không phải là biến toàn cục mà là biến cục bộ của hàm **main()**.

Một số lệnh đặc trưng của C: **register, static**

Chúng ta biết rằng các thanh ghi có tốc độ truy nhập nhanh hơn so với các loại bộ nhớ khác (RAM, bộ nhớ ngoài), do vậy nếu một biến thường xuyên sử dụng trong chương trình được lưu vào trong thanh ghi thì tốc độ thực hiện của chương trình sẽ được tăng lên. Để làm điều này ta đặt từ khóa **register** trước khai báo của biến đó.

Ví dụ:

```
register int a;
```

Tuy nhiên có một lưu ý khi khai báo biến **register** là vì số lượng các thanh ghi có hạn và kích thước của các thanh ghi cũng rất hạn chế (ví dụ trên dòng máy 80x86, các thanh ghi có kích thước 16 bit = 2 byte) cho nên số lượng biến khai báo **register** sẽ không nhiều và thường chỉ áp dụng với những biến có kích thước nhỏ như kiểu **char**, **int**.

Như ta đã biết, một biến cục bộ khi ra khỏi phạm vi của biến đó thì bộ nhớ dành để lưu trữ biến đó sẽ được giải phóng. Tuy nhiên trong một số trường hợp ta cần lưu giá trị của các biến cục bộ này để phục vụ cho những tính toán sau này, khi đó ta hãy khai báo biến với từ khóa **static** ở đầu.

Ví dụ:

```
static int a;
```

Từ khóa **static** giúp chương trình dịch biết được đây là một biến tĩnh, nghĩa là nó sẽ được cấp phát một vùng nhớ thường xuyên từ lúc khai báo và chỉ giải phóng khi chương trình chính kết thúc. Như vậy về thời gian tồn tại biến **static** rất giống với biến toàn cục, chỉ có một sự khác biệt nhỏ là biến toàn cục thì có thể truy cập ở mọi

nơi trong chương trình (miễn là ở đó không có biến địa phương nào cùng tên che lấp nó), còn biến **static** thì chỉ có thể truy nhập trong phạm vi mà nó được khai báo mà thôi.

Hãy xét ví dụ sau:

```
# include <stdio.h>
# include <conio.h>
void fct()
{
    static int count = 1;
    printf("\n Day la lan goi ham fct lan thu %2d", count++);
}
void main()
{
    int i;
    for(i = 0;  i < 10; i++)
        fct();
    getch();
}
```

Kết quả khi thực hiện:

```
Day la lan goi ham fct lan thu  1
Day la lan goi ham fct lan thu  2
Day la lan goi ham fct lan thu  3
Day la lan goi ham fct lan thu  4
Day la lan goi ham fct lan thu  5
Day la lan goi ham fct lan thu  6
Day la lan goi ham fct lan thu  7
Day la lan goi ham fct lan thu  8
Day la lan goi ham fct lan thu  9
Day la lan goi ham fct lan thu 10
```

# III.7. Tệp dữ liệu

## III.7.1. Khái niệm và phân loại tệp

*Khái niệm tệp*

Tệp dữ liệu (*File*) là một tập hợp các dữ liệu có liên quan với nhau và có cùng kiểu dữ liệu.

Tệp được lưu trữ trên các thiết bị nhớ ngoài (đĩa mềm, đĩa cứng, CD-ROM...) với một tên nào đó để phân biệt với nhau.

*Phân loại tệp*

Dựa theo bản chất dữ liệu của tệp, người ta chia tệp thành hai loại là tệp văn bản và tệp nhị phân.

Tệp văn bản (*text file*) là tệp mà các phần tử của nó là các kí tự như chữ cái, chữ số, các dấu câu, các dấu cách và một số kí tự điều khiển (như CR – *Carriage Return* – có mã ASCII là 13, để về đầu dòng, LF – *Line Feed* – có mã ASCII là 10, để xuống dòng mới).

Tệp nhị phân (*binary file*) là tệp mà các phần tử của nó là các số nhị phân 0 và 1 mã hóa thông tin. Thông tin được mã hóa bởi các bit nhị phân có thể là số nguyên, số thực, các cấu trúc dữ liệu... Nếu thông tin được mã hóa là kí tự thì khi đó tệp nhị phân trở thành tệp văn bản. Vì vậy tệp văn bản là một trường hợp riêng của tệp nhị phân.

*Vai trò của tệp*

Dữ liệu phục vụ cho chương trình được lưu trữ trong các biến, tức là nằm ở bộ nhớ trong. Do đó khi chương trình kết thúc hoặc khi tắt máy thì các thông tin dữ liệu đó cũng không còn. Muốn lưu trữ dữ liệu lâu dài để sử dụng cho những lần sau ta phải lưu dữ liệu lên tệp, tức là để dữ liệu nằm ở bộ nhớ ngoài. Dữ liệu trên bộ nhớ ngoài không bị mất đi khi chương trình kết thúc hay khi ta tắt máy. Vì vậy tệp là phương tiện dùng để cất giữ dữ liệu lâu dài.

*Phân biệt tệp và mảng*

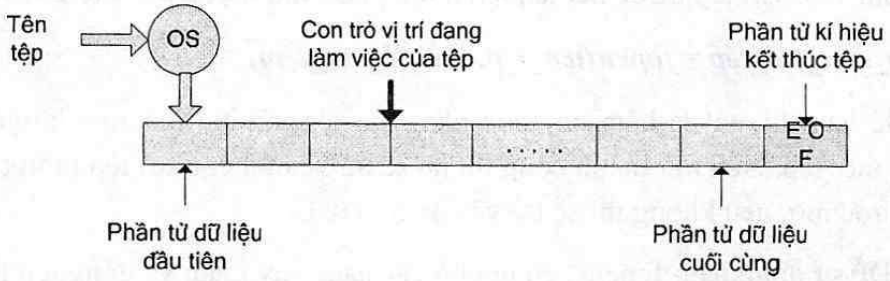
Về mặt tổ chức tệp và mảng hai rất giống nhau, đó là tập hợp các phần tử cùng kiểu, nhưng tệp khác với mảng ở hai điểm quan trọng sau đây:

- Tệp được lưu trữ trên bộ nhớ ngoài, do vậy dữ liệu trên tệp tồn tại lâu dài. Còn mảng thì được lưu trữ ở bộ nhớ trong, nên dữ liệu của mảng sẽ không còn khi chương trình kết thúc hoặc khi tắt máy.

- Kích thước của một tệp thường lớn hơn kích thước của một mảng rất nhiều vì tệp được lưu trữ trên bộ nhớ ngoài có dung lượng lớn hơn nhiều so với dung lượng của bộ nhớ trong là nơi lưu trữ mảng.

*Tổ chức của tệp*

Như đã nói, kiểu dữ liệu tệp khá giống với kiểu dữ liệu mảng ở chỗ là một tập hợp các phần tử cùng kiểu dữ liệu. Điểm khác nhau là số lượng phần tử trong mảng bị giới hạn trong khi số lượng phần tử của tệp không bị giới hạn. Vì số lượng phần tử trong tệp không bị giới hạn nên để biết được khi nào tới phần tử cuối cùng của tệp, tất cả các tệp đều sử dụng một phần tử đặc biệt gọi là phần tử kí hiệu kết thúc tệp (*End Of File indicator* – **EOF**). Còn làm thế nào để biết được phần tử nằm ở vị trí đầu của tệp? Từ đường dẫn và tên tệp hợp lệ của tệp, hệ điều hành sẽ tự động giúp ta tìm ra được vị trí lưu trữ phần tử đầu tiên của tệp trên đĩa.




***Hình III. 7. Cấu trúc của tệp***

*Con trỏ tệp*

Các phần tử của một tệp tạo thành một dãy và tại một thời điểm ta chỉ có thể truy cập được vào một phần tử của tệp mà thôi. Việc truy cập vào một phần tử được thực hiện thông qua một biến đệm, biến đệm này dùng để đánh dấu vị trí truy cập vào tệp tại thời điểm xác định. Biến đệm này được gọi là con trỏ tệp (*File positon locator* – tức là con trỏ chỉ vị trí đang làm việc của tệp, gọi tắt là con trỏ tệp).

Khi mở tệp con trỏ tệp sẽ luôn trỏ vào vị trí đầu tiên của tệp. Sau mỗi thao tác đọc ghi trên tệp, con trỏ tệp sẽ tự động dịch chuyển về phía cuối tệp. Khoảng cách dịch chuyển (tính theo byte) sẽ bằng số byte đã được đọc từ tệp hoặc ghi lên tệp.

*Quy trình thao tác với tệp*

Các thao tác với tệp phải tuân thủ theo trình tự sau:

- Khai báo tệp
- Mở tệp để làm việc
- Truy nhập tệp
- Đóng tệp

## III.7.2. Các thao tác với tệp

### III.7.2.1. Khai báo

Trong C truy nhập tệp phải thông qua con trỏ tệp. Một con trỏ tệp (*file pointer*) được khai báo như sau:

**FILE *tên_con_trỏ_tệp;**

Ví dụ:

**FILE *f1, *f2;**

### III.7.2.2. Mở tệp

Muốn làm việc với tệp trước hết ta phải mở tệp. Để mở một tệp ta dùng lệnh sau:

***tên_con_trỏ_tệp = fopen(tên_tệp,chế_độ_mở_tệp);***

Trong đó hàm fopen()là hàm có chức năng mở tệp với hai tham số là tên_tệp và chế_độ_mở_tệp. Nếu mở thành công thì nó sẽ trả về một con trỏ tệp tương ứng với tệp đã được mở, nếu không thì sẽ trả về con trỏ NULL.

Lưu ý: Để sử dụng hàm fopen() cũng như các hàm truy nhập và đóng tệp trình bày ở phần dưới ta phải khai báo tệp tiêu đề stdio.h.

*Chế độ mở tệp*

Phụ thuộc vào mục đích sử dụng tệp và bản chất dữ liệu trên tệp, người ta có các chế độ mở tệp sau:

**"rb"**, **"wb"**, **"ab"**, **"r+b"**, **"w+b"**, **"a+b"** và **"rt"**, **"wt"**, **"at"**, **"r+t"**, **"w+t"**, **"a+t"**. Trong xâu kí hiệu cho chế độ mở tệp, kí tự cuối cùng dùng để kí hiệu cho bản chất dữ liệu của tệp, các kí tự còn lại ở đầu dùng để kí hiệu cho mục đích sử dụng tệp.

| Kí hiệu | Mục đích sử dụng tệp |
|---|---|
| "r" | Mở tệp đã có để đọc, không được ghi. Nếu tệp không tồn tại, hàm fopen() sẽ trả lại trạng thái lỗi. |
| "w" | Mở tệp mới để ghi. Nếu tệp đã tồn tại nội dung của nó sẽ bị xóa hết. |
| "a" | Mở tệp để ghi thêm dữ liệu vào cuối tệp. Nếu tệp chưa tồn tại, nó sẽ được tạo mới. |

| | |
|---|---|
| "r+" | Mở tệp để vừa đọc vừa ghi. Nếu tệp chưa tồn tại thì sẽ báo lỗi. |
| "w+" | Mở tệp để vừa đọc vừa ghi. Nếu tệp đã tồn tại, nội dung của nó sẽ bị xóa hết. |
| "a+" | Mở tệp để ghi thêm dữ liệu vào cuối tệp. Tệp mới sẽ được tạo nếu nó chưa tồn tại. |

| Kí hiệu | Bản chất dữ liệu của tệp |
|---|---|
| "b" | Tệp nhị phân |
| "t" | Tệp văn bản |

Ví dụ với khai báo:

```
FILE *f1, *f2, *f3;
```

Để mở tệp *c:\abc.txt* để đọc ta dùng lệnh

```
f1 = fopen("c:\\abc.txt", "r");
```

Để mở tệp *c:\ho_so.dat* để ghi ta dùng lệnh

```
f2 = fopen("c:\\ho_so.dat", "w");
```

Để mở tệp *c:\abc.txt* để vừa đọc và ghi ta dùng lệnh

```
f3 = fopen("c:\\abc.txt", "r+");
```

Lưu ý: Khi mở tệp, nếu không chỉ rõ bản chất dữ liệu của tệp thì C sẽ ngầm hiểu đó là tệp văn bản. Nếu muốn chỉ rõ bản chất dữ liệu của tệp là tệp nhị phân thì ta sử dụng thêm kí hiệu b đi kèm với kí hiệu xác định mục đích mở tệp.

Ví dụ: Để mở tệp *c:\ho_so.dat* theo chế độ nhị phân và để ghi, ta dùng lệnh:

```
f2 = fopen("c:\\ho_so.dat", "wb");
```

Đôi khi do những lý do khách quan hoặc chủ quan mà việc mở tệp không thành công và hàm fopen() sẽ trả về giá trị NULL để báo rằng việc mở tệp không thành công. Khi đó ta nên "bắt" lấy kết quả trả về này để có những xử lí thích hợp, nếu không chương trình sẽ báo lỗi và tự động thoát ra ngoài.

Để bắt lỗi phát sinh khi mở tệp ta có thể sử dụng mẫu sau:

```
// Trường hợp mở tệp có lỗi
if(con_trỏ_tệp = fopen(tên_tệp, chế_độ_mở_tệp)) == NULL)
{
    <Xử lí cho trường hợp mở tệp không thành công>
}else // Trường hợp mở tệp thành công
{
    <Xử lí khi mở tệp thành công>
}
```

### III.7.2.3. Truy nhập tệp văn bản

*Đọc dữ liệu từ tệp*

Để đọc tệp ta dùng các hàm sau: fscanf(), fgets(), getc().

Hàm fscanf() có cú pháp khai báo là:

***int fscanf(FILE* con trỏ tệp, xâu định dạng, [danh_sách_địa_chỉ]);***

Lệnh fscanf() có chức năng đọc từ tệp văn bản tương ứng với con_trỏ_tệp dãy các dữ liệu, định dạng các dữ liệu đó theo khuôn dạng có trong xâu_định_dạng, sau đó lưu các giá trị đã được định dạng vào các vùng nhớ xác định trong danh_sách_địa_chỉ.

Kết quả trả về: Nếu thực hiện thành công, hàm fscanf() trả về một giá trị nguyên là số byte đọc được từ tệp. Nếu thực hiện không thành công thì hàm trả về giá trị **EOF**.

Ví dụ:

```
fscanf(fptr, "%d %c",&a, &c);
```

Lệnh trên nhập giá trị cho 2 biến a, c từ tệp.

Hàm fflush(): tương tự như hàm scanf(), trước khi dùng hàm fscanf() để nhập dữ liệu là kí tự, xâu kí tự từ tệp ta nên dùng lệnh fflush(). Hàm fflush() có cú pháp khai báo như sau:

***int fflush(FILE* con trỏ tệp);***

Hàm fflush() ghi toàn bộ dữ liệu chứa trong vùng đệm của tệp (nằm ở bộ nhớ trong) tương ứng với con_trỏ_tệp ra vùng nhớ của tệp trên bộ nhớ ngoài.

Giá trị trả về: Nếu hàm fflush() thực hiện thành công thì sẽ trả về giá trị 0, ngược lại giá trị trả về là **EOF**.

Hàm fgets() có cú pháp khai báo là:

***char* fgets(char* xâu_kí_tự, int n, FILE* con_trỏ_tệp);***

Hàm fgets() đọc từ tệp một xâu kí tự (cho phép chứa dấu cách) và gán xâu đọc được cho biến xâu_kí_tự. Việc đọc từ tệp sẽ dừng lại khi fgets() đọc đủ n-1 kí tự hoặc khi nó gặp kí tự xuống dòng, tùy gặp cái nào trước. Hàm fgets() sẽ tự động thêm kí tự xuống dòng ("\n") và kí tự kết thúc xâu ("\0", kí tự NUL) vào cuối xâu_kí_tự.

Giá trị trả về: Nếu đọc thành công, hàm fgets() sẽ trả về xâu kí tự trỏ bởi xâu_kí_tự, nếu có lỗi nó sẽ trả về con trỏ NULL.

Hàm getc() có cú pháp khai báo là:

***int getc(FILE* con trỏ tệp);***

Hàm getc() đọc từ tệp một kí tự (tức là một byte dữ liệu), sau đó chuyển đổi kí tự đó sang dạng số nguyên **int** (bằng cách thêm byte cao 0x00) rồi lấy giá trị số nguyên thu được làm giá trị trả về của hàm.

Giá trị trả về: Nếu thành công hàm getc() trả về kí tự đọc được sau khi đã chuyển sang dạng **int**. Nếu không thành công hàm trả về giá trị **EOF**.

*Ghi dữ liệu lên tệp*

Để ghi dữ liệu lên tệp ta dùng các hàm sau: fprintf(), fputs(), putc().

Hàm fprintf() có cú pháp khai báo là:

***int fprintf(FILE* con_trỏ_tệp, xâu_định_dạng, [danh_sách_tham_số]);***

Hàm fprintf() có chức năng hoàn toàn tương tự như hàm printf(), chỉ có một chỗ khác là hàm printf() ghi dữ liệu lên thiết bị ra chuẩn (stdin, thông thường là màn hình) còn fprintf() ghi dữ liệu lên một tệp được chỉ định trong tham số con_trỏ_tệp.

Kết quả trả về: Nếu thực hiện thành công, hàm fprintf() trả về một giá trị nguyên là số byte dữ liệu đã ghi lên tệp. Nếu thực hiện không thành công thì hàm fprintf() trả về giá trị **EOF**.

Hàm fputs() có cú pháp khai báo là:

***int fputs(char* xâu_kí_tự, FILE* con_trỏ_tệp);***

Hàm fputs() sẽ ghi nội dung của xâu_kí_tự lên tệp tương ứng với con_trỏ_tệp, tuy nhiên nó khác với hàm puts() ở chỗ không tự động ghi thêm kí tự xuống dòng lên tệp.

Giá trị trả về: Nếu thực hiện thành công, hàm fputs() trả về kí tự cuối cùng mà nó ghi được lên tệp, còn nếu không thành công nó trả về giá trị **EOF**.

Hàm putc() có cú pháp khai báo là:

***int putc(int ch, FILE* con_trỏ_tệp);***

Hàm putc() ghi nội dung của kí tự chứa trong biến **int** ch (kí tự được chứa trong byte thấp của biến ch) lên tệp tương ứng với con_trỏ_tệp.

Giá trị trả về: Nếu thành công, hàm putc() sẽ trả về số nguyên (kiểu **int**) là số thứ tự trong bảng mã ASCII của kí tự đã ghi lên tệp. Nếu không thành công nó trả về giá trị **EOF**.

Ví dụ:

```
int m;
m = putc(0x2345,stdout); // m = 0x0045 = 69, la so thu tu cua ki tu E
```

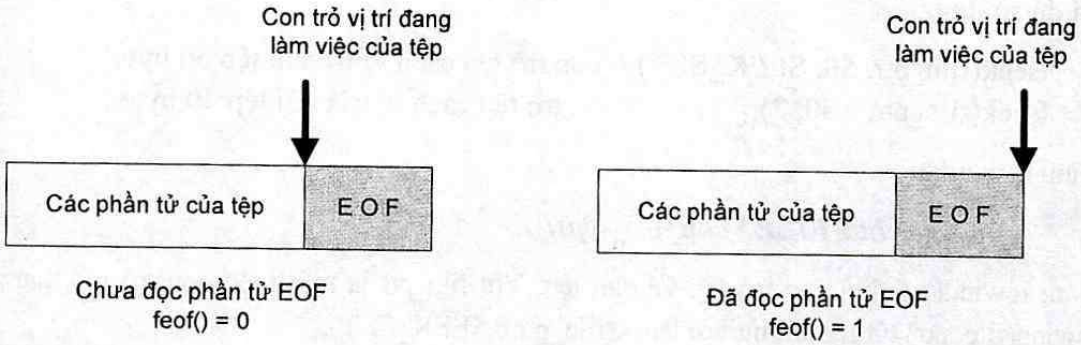
*Một số thao tác khác*

Hàm feof()

***int feof(FILE* con_trỏ_tệp);***

Hàm feof() dùng để kiểm tra xem đã duyệt đến vị trí cuối tệp hay chưa. Nó thực hiện việc này bằng cách kiểm tra xem trong khối dữ liệu được đọc vào ở lần thực hiện gần nhất có phần tử **EOF** hay không (tức là phần tử **EOF** có được đọc trong lần đọc gần đây nhất hay không), nếu có thì hàm feof() trả về một giá trị khác 0, còn nếu chưa thì trả về giá trị 0.

Cần lưu ý điều này vì ta thường lầm tưởng rằng hàm feof() kiểm tra việc đã duyệt đến phần tử cuối cùng của tệp hay chưa bằng cách kiểm tra xem con trỏ tệp đã trỏ đến phần tử cuối cùng hay chưa. Tuy nhiên, khi con trỏ vừa đọc xong phần tử cuối cùng, thì phần tử **EOF** vẫn chưa được đọc.




Hàm fseek()

***int fseek(FILE* con_trỏ_tệp, long int n, int vị_trí_ban_đầu);***

Hàm fseek() dùng để dịch chuyển con trỏ tệp từ vị_trí_ban_đầu đi một khoảng cách có độ dài n byte.

Giá trị trả về: Hàm fseek() sẽ trả về giá trị 0 nếu như việc dịch chuyển thành công và trả về giá trị khác 0 nếu việc dịch chuyển không thành công.

*Lưu ý*:

- Giá trị của biến n có thể lớn hơn, nhỏ hơn hoặc bằng 0. Nếu lớn hơn 0 thì hướng của sự dịch chuyển là về phía cuối tệp, nhỏ hơn 0 nghĩa là dịch chuyển về phía đầu tệp, bằng 0 nghĩa là thực tế không dịch chuyển gì cả.
- Cẩn thận trọng khi dùng hàm fseek() với tệp mở theo kiểu văn bản vì khi mở tệp theo kiểu văn bản có thể có sự tự động chuyển đổi kí tự (ví dụ chuyển đổi cặp kí tự '\r' và '\n' thành kí tự '\n'), điều đó khiến cho việc tính toán số bước dịch chuyển của con trỏ tệp rất dễ bị sai.
- Tham số vị_trí_ban_đầu có thể nhận 1 trong 3 giá trị hằng số sau:

| Tên hằng | Giá trị | Ý nghĩa |
|---|---|---|
| **SEEK_SET** | 0 | Vị trí ban đầu là đầu tệp |
| **SEEK_CUR** | 1 | Vị trí ban đầu là vị trí hiện thời của con trỏ vị trí làm việc của tệp |
| **SEEK_END** | 2 | Vị trí ban đầu là cuối tệp |

Ví dụ sử dụng:

```
fseek(file_ptr, 50, SEEK_SET);// con trỏ tệp cách vị trí đầu tệp 50 byte
fseek(file_ptr, - 40, 2);          // con trỏ tệp cách vị trí cuối tệp 40 byte
```

Hàm rewind():

***void rewind(FILE* con_trỏ_tệp);***

Hàm rewind()sẽ đưa con trỏ tệp về đầu tệp. Với file_ptr là một biến con trỏ tệp, hàm rewind(file_ptr) tương đương với fseek(file_ptr,0,SEEK_SET);

Hàm rewind() không có giá trị trả về.

*Lưu ý*: Để sử dụng các hàm fscanf(), fgets(), getc(), fflush(), fprintf(), fputs(), putc(), feof(), fseek() và rewind() ta cần khai báo tệp tiêu đề stdio.h.

### III.7.2.4. Truy nhập tệp nhị phân

*Đọc dữ liệu trên tệp*

Để đọc dữ liệu từ tệp nhị phân ta dùng hàm fread() có cú pháp khai báo như sau:

**int fread (void *địa_chỉ_biến, int kích_thước_phần_tử, int số_phần_tử, FILE* con_trỏ_tệp);**

Hàm fread() đọc từ tệp một khối dữ liệu kích thước số_phần_tử × kích_thước_phần_tử byte, sau đó ghi khối dữ liệu đó lên vùng nhớ có địa chỉ là địa_chỉ_biến.

Kết quả trả về: Nếu việc đọc dữ liệu từ tệp thực hiện thành công, hàm fread() trả về một giá trị nguyên là số mục (không phải số byte) đọc được từ tệp. Nếu thực hiện không thành công thì hàm fread() trả về giá trị 0.

*Ghi dữ liệu trên tệp*

Để ghi dữ liệu lên tệp nhị phân ta dùng hàm fwrite() có cú pháp khai báo là:

**int fwrite (void *địa_chỉ_biến, int kích_thước_phần_tử, int số_phần_tử, FILE* con trỏ tệp);**

Hàm fwrite() sẽ đọc từ bộ nhớ một khối dữ liệu có địa chỉ bắt đầu là địa_chỉ_biến và có kích thước là số_phần_tử × kích_thước_phần_tử byte, sau đó nó ghi khối dữ liệu này lên tệp.

Kết quả trả về: Nếu việc ghi dữ liệu lên tệp thực hiện thành công, hàm fwrite() sẽ trả về một giá trị nguyên là số mục (không phải số byte) đã ghi lên tệp. Nếu thực hiện không thành công thì hàm fwrite()trả về giá trị 0.

*Dịch chuyển con trỏ tệp*

Tương tự như tệp văn bản, ta có thể dùng các hàm fseek()và rewind()để dịch chuyển con trỏ tệp trên tệp nhị phân. Hàm fseek()khi dùng với tệp nhị phân thì không phải lưu ý như khi dùng với tệp văn bản.

Nhận xét: Các hàm trong các cặp hàm fread() – fwrite(), fscanf() – fprintf(), fputs() – fgets(), và getc() – putc() có chức năng đối ngẫu nhau.

### III.7.2.5. Đóng tệp

Đóng tệp là đảm bảo những thay đổi dữ liệu được lưu lại trên tệp.

Để đóng tệp, ta dùng hàm fclose() có cú pháp khai báo:

**int fclose(FILE* <tên con trỏ tệp>);**

Hàm fclose() trả lại giá trị 0 nếu đóng thành công, trả về giá trị **EOF** nếu không đóng tệp thành công.

Ví dụ tổng hợp:

Bài toán: Tạo tệp nhị phân float.dat đặt tại ổ đĩa C để ghi 100 số thực. Sau đó mở lại tệp này và ghi nội dung sang một tệp khác, có tên được nhập từ bàn phím. Tiếp theo đó, hiển thị giá trị số thực đầu tiên trong tệp float.dat; cho phép người dùng nhập một số thứ tự từ bàn phím và hiển thị số thực ở vị trí này trong tệp float.dat.

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
void main(){
    FILE* fptr1, *fptr2;// Khai bao bien con tro tep
    int i;// Bien dem
    float f, a[100]; // Mot bien so thuc va mot mang so thuc
    char file_name_2[20];// xau ki tu chua ten tep thu 2
    clrscr();
    // Mở tệp c:\float.dat de ghi len do 100 so thuc
    if((fptr1 = fopen("c:\\float.dat","wb"))==NULL)
    {
```

```
    printf("\n Khong mo duoc file c:\\float.dat");
    printf("\n An phim bat ki de ket thuc chuong trinh");
    exit(1);
}
// Tao 100 so thuc theo mot cong thuc nao do va ghi vao mang a[100]
for(i=0;i<100;i++)
    a[i] = (i*i+1.0)/(i+1);
// Ghi cac so thuc len tep c:\float.dat
for(i=0;i<100;i++)
    fwrite(&a[i],sizeof(float),1,fptr1);
// Đóng tệp c:\float.dat lai de luu du lieu vua ghi
fclose(fptr1);
// Mở tệp c:\float.dat vua tao de doc du lieu
if((fptr1 = fopen("c:\\float.dat","rb"))==NULL)
{
    printf("\n Khong mo duoc file c:\\float.dat");
    printf("\n An phim bat ki de ket thuc chuong trinh");
    exit(1);
}
// Mo tep thu 2 co ten nhap tu ban phim
// ta se copy du lieu tu tep c:\float.dat sang tep nay
printf("\n Nhap vao ten tep thu 2: ");
fflush(stdin);
gets(file_name_2);
if((fptr2 = fopen(file_name_2,"wb"))==NULL)
{
    printf("\n Khong mo duoc file %s",file_name_2);
    printf("\n An phim bat ki de ket thuc chuong trinh");
```

```
    exit(1);
 }
 // Sao chep du lieu tu tep c:\float.dat sang tep thu 2
 fread(&f,4,1,fptr1);
 while(!feof(fptr1))
 {
    fwrite(&f,4,1,fptr2);
    fread(&f,4,1,fptr1);
 }
 // Dich chuyen con tro vi tri lam viec hien tai cua tep
 // c:\float.dat ve dau tep
 // Doc va hien thi so thuc dau tien trong tep c:\float.dat
rewind(fptr1);
 fread(&f,sizeof(float),1,fptr1);
 printf("\n So thuc dau tien tren c:\\float.dat la %f",f);
 // Doc va hien thi so thuc trong tep c:\float.dat
// co so thu tu nhap tu ban phim
 printf("\n Cho biet thu tu cua so thuc trong c:\\float.dat: ");
 scanf("%d",&i);
 // Dich chuyen con tro cua tep c:\float.dat
 // den vi tri tuong ung voi so thuc muon hien thi
 fseek(fptr1,(i-1)*sizeof(float),SEEK_SET);
 fread(&f,sizeof(float),1,fptr1);
 printf(" So thuc thu %d trong c:\\float.dat la %f",i,f);
 fclose(fptr1);
 fclose(fptr2);
 // Cho an phim bat ki de ket thuc chuong trinh
 getch();
```

}

Kết quả thực hiện:

Nhap vao ten tep thu 2: c:\f_copy.dat

So thuc dau tien tren c:\float.dat la 1.000000

Cho biet thu tu cua so thuc trong tep c:\float.dat: 10

So thuc thu 10 trong c:\float.dat la 8.200000

Và trên thư mục gốc của ổ đĩa C ta sẽ thấy có 2 tệp là *float.dat* và *f_copy.dat* cùng có kích thước 400 byte.

# III.8. Câu hỏi và bài tập

**1.** Trong các định danh sau, đâu là định danh hợp lệ, đâu là định danh không hợp lệ?

1_a, 3d, so luong, int, char, string, _constant_2, a, sinh_vien

**2.** Hãy cho biết các hàm toán học sau thực hiện chức năng gì?

sqrt(x), pow(x,y), exp(x), log(x), cos(x), ceil(x), floor(x)

**3.** Để viết chú thích trong ngôn ngữ C, có mấy cách? Đó là gì?

**4.** Viết các khai báo cho các yêu cầu sau:

a. x là biến nguyên, y là biến thực

b. biến k sẽ nhận các giá trị trong đoạn [0, 255]

c. m, n là các biến nguyên trong đoạn [-32768, +32767]

d. i, j là các biến có thể nhận giá trị trong đoạn [0, 65535]

e. ch là biến ký tự

**5.** Chọn cách khai báo đúng tệp tiêu đề trong ngôn ngữ C:

a. #include<tên_tệp_ tiêu_ đề>

b. #include tên_tệp_ tiêu_ đề

c. include < tên_tệp_ tiêu_ đề >

d. #include< tên_tệp_ tiêu_ đề >;

**6.** Tìm các ký hiệu ghi chú thích trong ngôn ngữ C:

a. (*dòng chú thích*)

b. {dòng chú thích}

c. /* dòng chú thích */

d. // dòng chú thích

e. <-- dòng chú thích-->

**7.** Để sử dụng hàm clrscr(), phải thêm tệp tiêu đề nào vào chương trình?

a. string.h

b. conio.h

c. stdio.h

d. math.h

**8.** Xét khai báo:

int a, b;

Chỉ ra biểu thức không hợp lệ:

a. a-=b

b. a==b

c. a=b

d. a-b=0

**9.** Chọn khai báo hợp lệ:

a. float: a-9;

b. a=9.0: float;

c. a:float=9;

d. float a=9.0;

**10.** Nhập vào một số nguyên từ bàn phím và hiển thị số đó ra màn hình.

**11.** Nhập vào một số thực từ bàn phím và hiển thị số đó ra màn hình theo quy cách có 3 vị trí sau dấu chấm.

**12.** Nhập vào hai số nguyên từ bàn phím, sau đó tính và hiển thị ra màn hình giá trị tổng, hiệu, tích, thương của hai số vừa nhập.

**13.** Nhập vào một xâu ký tự và hiển thị xâu vừa nhập ra màn hình.

**14.** Viết chương trình tính và hiển thị ra màn hình chu vi và diện tích hình tròn theo giá trị bán kính r nhập vào từ bàn phím.

**15.** Nhập vào ba số nguyên a, b, c từ bàn phím. Hiển thị giá trị của các biểu thức sau ra màn hình:

a. a++ + ++a

b. --a - b-- * ++c

**16.** Dùng biểu thức điều kiện ?: để tìm và hiển thị ra số lớn nhất trong ba số thực nhập vào từ bàn phím

**17.** Nhập vào 2 số thực từ bàn phím. Viết chương trình dùng lệnh if else tìm và hiển thị giá trị nhỏ nhất ra màn hình.

**18.** Viết chương trình nhập vào số nguyên từ bàn phím. Nếu đó là số chẵn thì hiển thị "Ban vua nhap so chan", nếu là số lẻ thì hiển thị "Ban vua nhap so le".

**19.** Nhập vào 3 số thực từ bàn phím, kiểm tra xem nó có tạo thành 3 cạnh của một tam giác không. Nếu có thì hiện thông báo ra màn hình "La 3 canh cua tam giac", ngược lại hiển thị "Không phai la 3 canh cua tam giac".

**20.** Trong một năm các tháng có 30 ngày là 4, 6, 9, 11 còn các tháng có 31 ngày là 1, 3, 5, 7, 8, 10, 12. Riêng tháng hai có thể có 28 hoặc 29 ngày. Hãy viết chương trình nhập vào 1 số thể hiện tháng, sau đó đưa ra kết luận tháng đó có bao nhiêu ngày (dùng cấu trúc switch).

**21.** Đưa ra màn hình các số nguyên lẻ nhỏ trong khoảng từ 1 đến 100 (dùng vòng lặp for).

**22.** Nhập vào một số N nguyên, N>1 từ bàn phím. Tính tổng các số nguyên từ 1 đến N (dùng vòng lặp for, while, do while).

**23.** Viết chương trình nhập vào số nguyên dương N từ bàn phím, sau đó tính giai thừa và hiển thị ra màn hình (dùng for, while và do while).

**24.** Hãy tìm giá trị của biến sum sau đoạn chương trình sau:

```
for(int i =0;i+1<=10;i+=2)
if(i==0) sum = i;
else sum+=i;
```

a. 12

b. 20

c. 30

d. Các phương án trên đều sai

**25.** Lập chương trình giải phương trình bậc hai: $ax^2 + bx + c = 0$ với a, b, c nhập vào từ bàn phím.

**26.** Viết chương trình kiểm tra một số nguyên nhập từ bàn phím có phải là số nguyên tố hay không?

**27.** Lập trình tính và hiển thị tổng S = 1 + 1/2 + 1/3 + … + 1/n với n nhập vào từ bàn phím.

**28.** Viết chương trình nhập vào 2 số nguyên dương từ bàn phím và đưa ra ước số chung lớn nhất của chúng.

**29.** Viết chương trình nhập vào 2 số nguyên từ bàn phím và đưa ra bội số chung nhỏ nhất của chúng.

**30.** Nhập vào một mảng số thực gồm 10 phần tử. Tìm và hiển thị giá trị và chỉ số của phần tử lớn nhất.

**31.** Nhập vào một mảng n số nguyên (tối đa 20 phần tử) với n nhập vào từ bàn phím. Sắp xếp lại mảng theo sự tăng dần của các giá trị phần tử.

**32.** Chỉ ra kết quả in trên màn hình:

```
int i = 0;
while(i++<=2) printf("Hello! ");
```

a. Hello!

b. Hello! Hello! Hello! Hello!

c. Hello! Hello!

d. Hello! Hello! Hello!

**33.** Nhập một xâu kí tự từ bàn phím gồm các từ, ví dụ "Thu  do Ha Noi". Lập chương trình để bỏ bớt các dấu trống giữa các từ sao cho các từ chỉ cách nhau ít nhất một dấu trống.

**34.** Viết chương trình nhập vào từ bàn phím họ và tên của một người, sau đó in phần tên ra màn hình. Ví dụ: "Tran Hung Dao" thì in ra "Dao".

**35.** Nhập vào một câu, kết thúc bằng dấu chấm. In ra câu đó có bao nhiêu từ.

**36.** Cho khai báo:

```
struct quoc_gia
{  char ten[31];
```

```
        char thu_do[31];
        int dan_so;
    }dsqg[10];
```

Mảng dsqg chiếm số byte trong bộ nhớ là:

a. 440

b. 460

c. 640

d. 660

**37.** Viết một chương trình thực hiện những công việc sau:

a. Yêu cầu người dùng nhập vào một số nguyên dương n với $5 \le n \le 20$ (có kiểm tra tính hợp lệ của giá trị được nhập vào, nếu giá trị n nhập vào không thỏa mãn điều kiện thì yêu cầu nhập lại).

b. Yêu cầu người dùng nhập vào thông tin của n sinh viên gồm những mục sau:

- Họ và tên: Có kiểu dữ liệu là xâu kí tự gồm không quá 30 kí tự
- Lớp: Xâu kí tự có độ dài không quá 5 kí tự
- Điểm thi Tin đại cương: Là một số nguyên có giá trị từ 0 đến 10

i. Đưa ra màn hình danh sách các sinh viên cùng thông tin của họ mà người dùng vừa nhập vào.

ii. Yêu cầu người dùng nhập vào từ bàn phím một số thực. Đưa ra màn hình danh sách các sinh viên có điểm thi Tin đại cương nhỏ hơn giá trị số thực vừa nhập vào.

iii. Đưa ra màn hình danh sách sinh viên được sắp xếp theo chiều giảm dần của điểm thi Tin đại cương.

iv. Đưa ra màn hình danh sách sinh viên với họ và tên được sắp xếp theo chiều của bảng chữ cái.

**38.** Thông tin về nguồn vốn viện trợ ODA mà các quốc gia viện trợ cho Việt Nam **từ năm 2000 đến 2007** được lưu trong **cấu trúc kiểu ODA** gồm có:

**tên quốc gia** *(xâu không quá 30 kí tự thường)*; **số tiền** *(mà quốc gia đó tài trợ, đơn vị: triệu USD)*, **năm** *(năm tài trợ)*.

Giả sử thông tin trên đã được lưu trữ trong tệp **TAITRO.DAT** tại thư mục gốc ổ **D**. Lập chương trình thực hiện những công việc sau:

a. Nhập thêm vào cuối file trên thông tin về sự tài trợ của các quốc gia trong năm 2008. Điều kiện kết thúc là khi nhập tên quốc gia là xâu "$$$".

b. Nhập vào một số N ( $2000 \le N \le 2008$). Đọc file trên, đưa ra tên, số tiền tài trợ của các quốc gia cho Việt Nam trong năm N theo quy cách: 30 vị trí cho phần tên quốc gia, 6 vị trí cho phần số tiền tài trợ, mỗi quốc gia trên một dòng.

c. Nhập vào một xâu kí tự S thể hiện tên một quốc gia. Đọc lại file trên và đưa ra tổng số tiền từ năm 2003 đến nay mà quốc gia đó đã tài trợ cho Việt Nam. Nếu không tìm thấy thì thông báo "KHÔNG TÌM THẤY".

**39.** Viết hàm tìm ước số chung lớn nhất uscln(int x, int y) và bội số chung nhỏ nhất bscnn(int x, int y) của hai số.

**40.** Lập hàm giải phương trình bậc 2 ptb2(float a, float b, float c, float *x1, float *x2), trong đó a,b,c là các hệ số của phương trình. Hàm nhận giá trị 0 nếu phương trình vô nghiệm, giá trị 1 nếu phương trình có nghiệm kép (chứa trong *x1), giá trị 2 nếu phương trình có 2 nghiệm (chứa trong *x1 và *x2), giá trị 3 nếu phương trình có vô số nghiệm.

**41.** Lập hàm tính giai thừa và dùng hàm này tính tổng: 1! + 2! +...+N! (N nhập từ bàn phím và <=10).

**42.** Đa thức cấp n được tính theo công thức.

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0 x^0$$

Lập hàm đa thức f(float a[], int n, float x) để tính giá trị của đa thức

**43.** Viết chương trình nhập từ bàn phím N số thực lưu vào một mảng (N < 100 và N được nhập từ bàn phím). Sau đó ghi ra một file văn bản có tên là "float.dat" theo quy cách: dòng đầu tiên lưu số lượng các số thực, các dòng tiếp theo lưu các số thực, mỗi số lưu trên một dòng. Đọc lại tệp văn bản đó và lưu các số thực đọc được vào một mảng. Sắp xếp các số thực trong mảng theo thứ tự tăng dần và ghi ra một tệp văn bản khác có tên là "float_sx.dat" theo quy cách giống như tệp "float.dat".

**44.** Viết chương trình sao chép nội dung tệp mã nguồn chương trình C có tên là file_1.C sang tệp có tên là file_2.C.

**45.** Viết chương trình copy file:

a. Nhập vào từ bàn phím hai xâu kí tự là đường dẫn của file nguồn và file đích.

b. Copy nội dung của file nguồn sang file đích.

**46.** Viết chương trình ghép nối nội dung hai file:

a. Nhập vào từ bàn phím 2 xâu kí tự là đường dẫn của file nguồn và file đích.

b. Ghép nội dung của file nguồn vào cuối file đích.

**47.** Viết chương trình so sánh nội dung hai file:

a. Nhập từ bàn phím hai xâu kí tự là đường dẫn tới hai file cần so sánh.

b. Hiển thị ra màn hình dòng thông báo:

Hai file ten_1 và ten_2 giống nhau

nếu hai file có nội dung giống nhau (hai file có cùng kích thước và các bít cùng vị trí thì có giá trị giống nhau).

Hai file ten_1 và ten_2 không giống nhau

nếu hai file có nội dung khác nhau. Ở đây ten_1 và ten_2 là đường dẫn của hai file cần so sánh.

**48.** Mở một tệp văn bản, đếm xem trong văn bản đó có bao nhiêu kí tự, bao nhiêu từ, bao nhiêu câu.

**49.** Một tệp văn bản tên là "thisinh.dat" lưu dữ liệu về các thí sinh và có tổ chức như sau:

- Dòng đầu tên lưu số lượng thí sinh.
- Các dòng tiếp theo mỗi dòng lưu thông tin về một thí sinh gồm có: số báo danh (10 kí tự), họ và tên (30 kí tự), điểm thi (4 kí tự với 1 kí tự dành cho phần thập phân, một kí tự cho dấu "." dùng để ngăn cách và 2 kí tự cho phần nguyên).

Hãy viết chương trình:

a. Đọc dữ liệu từ tệp "thisinh.dat" và hiển thị ra màn hình danh sách các thí sinh theo quy cách:

Số thứ tự  Số báo danh  Họ tên  Điểm thi

Trong đó số thứ tự chiếm 3 vị trí, số báo danh chiếm 10 vị trí, họ và tên chiếm 30 vị trí, điểm thi chiếm 5 vị trí với 2 vị trí dành cho phần thập phân.

b. Sắp xếp các thí sinh theo kết quả điểm thi tăng dần và lưu vào tệp "thisinh2.dat" với quy cách giống như quy cách của tệp "thisinh.dat".