BÙI THẾ HỒNG

GIÁO TRÌNH
Ngôn ngữ lập trình C:
LÝ THUYẾT VÀ THỰC HÀNH

NHÀ XUẤT BẢN KHOA HỌC KỸ THUẬT

LỜI NÓI ĐẦU

Phần I: Các khái niệm cơ bản

Bài 1 Mở đầu

Tóm tắt lịch sử phát triển của ngôn ngữ C

Ngôn ngữ lập trình C do Dennis Ritchie tạo lập vào năm 1972 tại Bell Telephone Laboratories. Mục đích ban đầu của ngôn ngữ này là để thiết kế hệ điều hành UNIX.

C là một ngôn ngữ mạnh và mềm dẻo nên nó đã được rất nhiều người sử dụng. Khắp nơi người ta đã bắt đầu dùng nó để viết mọi loại chương trình. Tuy nhiên, các tổ chức khác nhau đã bắt đầu sử dụng các version khác nhau của C, và đã phát sinh nhiều sự khác nhau trong các thao tác làm cho người lập trình phải đau đầu. Để khắc phục vấn đề này, năm 1983, *Viện Tiêu chuẩn Quốc gia* của Mỹ (ANSI) đã thành lập một ủy ban để đưa ra một định nghĩa chuẩn cho ngôn ngữ C, được gọi là ***ANSI Standard C***.

Có lẽ phải nói thêm một chút về tên của ngôn ngữ. Ngôn ngữ này có tên là C vì trước nó đã có một ngôn ngữ được gọi là B. Ngôn ngữ B do Ken Thompson phát triển cũng tại Bell Labs. Có thể, B là để chỉ tên của phòng thí nghiệm, còn C chắc là do đứng sau B.

Tại sao lại dùng C?

Trong thế giới lập trình ngày nay, có rất nhiều ngôn ngữ lập trình bậc cao để lựa chọn, chẳng hạn như C, Pascal, BASIC, Modula, v.v. Tất cả đều là những ngôn ngữ tuyệt vời phù hợp cho hầu hết các công việc lập trình. Nhưng tuy vậy, vẫn có một vài lý do để nhiều nhà lập trình chuyên nghiệp cảm thấy C là ngôn ngữ đứng đầu, vì:

- *C là một ngôn ngữ mạnh và mềm dẻo*. Hạn chế duy nhất của C chính là sự hạn chế trong tư duy trừu tượng của chính người lập trình mà thôi. C được sử dụng cho nhiều mục đích khác nhau, như thiết kế các hệ điều hành, các bộ soạn thảo văn bản, đồ hoạ, trang tính, và thậm chí làm các chương trình dịch cho các ngôn ngữ khác.
- *C là một ngôn ngữ bình dân* được nhiều người lập trình chuyên nghiệp ưa dùng. Bằng chứng là đã có rất nhiều chương trình dịch khác nhau và nhiều tiện ích kèm theo.
- *C là một ngôn ngữ chuyển đổi được* tức là một chương trình C được viết cho một hệ máy vi tính (ví dụ IBM PC) có thể được dịch và chạy trên một hệ thống khác (ví dụ DEC VAX) mà chỉ cần thay đổi chút ít hoặc không cần thay đổi gì cả.
- *C là một ngôn ngữ ngắn gọn*, nó chỉ bao gồm một số các từ được gọi là từ khóa (keyword) làm cơ sở để tạo ra các câu lệnh của ngôn ngữ.

Với các đặc điểm trên, C là sự lựa chọn tuyệt vời đối với một ngôn ngữ lập trình. Nhưng có lẽ người ta còn nghe nói về C++ và một kỹ thuật lập trình mới được gọi là *lập trình hướng đối tượng*. Cũng không có gì đáng hoang mang cho lắm vì thực ra C++ chỉ là một ngôn ngữ siêu C với tất cả những gì mà C có cộng thêm với kỹ thuật lập trình hướng đối tượng. Nếu ai đó bắt đầu ngay bằng việc học C++, thì tất cả những gì đã được dạy khi học C vẫn còn được áp dụng cho C++. Trong khi học C, không những chỉ học một ngôn ngữ lập trình phổ thông và mạnh nhất hôm nay mà còn tự chuẩn bị cho mình một kỹ thuật lập trình hướng đối tượng cho ngày mai.

Chuẩn bị cho việc lập trình

Khi giải một bài toán cần phải tiến hành một số bước nhất định. Đầu tiên phải hình thành hay còn gọi là định nghĩa bài toán. Nếu không biết bài toán cần phải giải quyết là gì thì không thể nào tìm ra kết qủa được. Một khi bài toán đã được định rõ, thì có thể lập ra một kế hoạch để giải quyết nó. Sau khi đã có một kế hoạch thì việc thực hiện bài toán này sẽ trở nên dễ dàng. Cuối cùng, một khi kế hoạch đã được thực thi, thì các kết qủa phải được thử lại để kiểm chứng xem liệu bài toán đã được giải quyết đúng và trọn vẹn chưa. Một logic như trên có thể được áp dụng cho nhiều lĩnh vực khác nhau bao gồm cả việc lập trình trên máy tính.

Khi tạo lập một chương trình trong C, chúng ta nên tiến hành theo các bước sau đây:

1. Xác định các mục tiêu của chương trình.
2. Xác định các phương pháp mà ta muốn sử dụng trong khi viết chương trình.
3. Tạo lập chương trình để giải quyết bài toán.
4. Chạy chương trình để xem xét các kết qủa.

Chu trình phát triển chương trình

Chu trình phát triển chương trình có các bước riêng của nó. Trong bước thứ nhất, một bộ soạn thảo sẽ được sử dụng để tạo ra một tệp đĩa chứa các *mã gốc* (source code). Tại bước thứ hai, các mã gốc sẽ được dịch để tạo ra một *tệp đích* (object file). Tại bước thứ ba, các mã đã được dịch sẽ được kết nối lại để tạo ra một tệp *chương trình* có thể chạy được (executable file). Cuối cùng, bước bốn là để chạy chương trình và kiểm tra xem nó có làm việc như đã dự kiến không.

Tạo lập mã gốc

Các mã gốc là một loạt các câu lệnh, hoặc lệnh được sử dụng để chỉ thị cho máy tính thực hiện các công việc mà người lập trình mong muốn. Ví dụ, dưới đây là một dòng của mã gốc:

```
printf ("Chào các bạn");
```

Lệnh này chỉ thị cho máy tính hiện trên màn hình thông điệp `Chào các bạn.`
Hầu hết các chương trình dịch đều có một bộ sạon thảo để soạn ra các tệp mã gốc. Tuy nhiên, ta cũng có thể sử dụng bất kỳ một bộ soạn thảo văn bản nào để viết các chương trình C. Chỉ cần lưu ý là khi cất giữ tệp thì phải chọn kiểu tệp là văn bản và có phần mở rộng là .C.

Dịch các mã gốc

Các máy tính chỉ hiểu được một ngôn ngữ, đó là *ngôn ngữ máy* (machine language). Bởi vậy, trước khi một chương trình C có thể chạy được trên một máy tính, nó phải được dịch từ các mã gốc sang mã máy bằng một chương trình được gọi là *chương trình dịch* (compiler). Chương trình dịch lấy các mã gốc làm input và cho ra một tệp đĩa chứa các câu lệnh mã máy tương ứng với các câu lệnh mã gốc. Các câu lệnh mã máy được tạo ra bởi chương trình dịch được gọi là các mã đích và tệp đĩa chứa chúng được gọi là tệp đích. Tệp đích có tên trùng với tên tệp gốc và luôn có đuôi là *OBJ*. Đuôi OBJ để chỉ rằng tệp này là tệp đích và sẽ được sử dụng bởi *chương trình kết nối* (linker).

Kết nối để tạo ra một tệp có thể chạy được

Cần phải thực hiện thêm một bước nữa thì chương trình mới có thể chạy được. Trong C có một bộ phận được gọi là *thư viện các hàm*, nó chứa các mã đích của các hàm đã được lập sẵn. Ví dụ, hàm `printf` được sử dụng trong ví dụ trước là một hàm thư viện. Các hàm thư viện thực hiện các công việc thường hay phải dùng đến, chẳng hạn như hiện các thông tin lên màn hình, đọc số liệu từ các tệp đĩa. Nếu một chương trình có sử dụng các hàm thư viện thì tệp đích của nó cần phải được kết nối với mã đích của các hàm này để tạo ra một chương trình có thể chạy được. Tệp này có tên trùng với tên tệp gốc và có đuôi là .EXE.

Hoàn thiện chu trình phát triển

Sau khi dịch và kết nối cần phải chạy thử chương trình. Nếu kết qủa nhận được không như mong muốn thì cần phải tìm ra nguyên nhân dẫn đến các sai sót và sửa lại chương trình cho đúng. Khi có bất kỳ một thay đổi nào trong mã gốc, chương trình phải được dịch và kết nối lại. Chu trình trên sẽ phải lặp đi lặp lại cho đến khi nào nhận được kết qủa như mong muốn.

Một điểm lưu ý cuối cùng về việc dịch và kết nối là, mặc dù việc dịch và kết nối như đã trình bầy trên đây là hai giai đoạn tách biệt nhưng nhiều chương trình dịch, chẳng hạn các chương trình dịch chạy dưới DOS lại gộp hai bước trên thành một. Cho dù như vậy nhưng vẫn cứ phải hiểu rằng đây là hai qúa trình riêng rẽ ngay cả khi chúng chỉ được thực hiện bằng một lệnh.

Chu trình phát triển một chương trình C

Bước 1: **Sử dụng một bộ soạn thảo để viết các mã gốc.**
Thông thường các tệp gốc của C có đuôi là .C

Bước 2: **Dùng một chương trình để dịch chương trình gốc.**
Nếu chương trình dịch không phát hiện một sai sót nào trong chương trình gốc, nó sẽ sinh ra một tệp đích. Tệp này có cùng tên với tệp chương trình gốc với đuôi là .OBJ. Nếu chương trình dịch tìm thấy một lỗi nào đó, nó sẽ thông báo lên màn hình và ta phải trở lại từ bước một để sửa các lỗi đã được phát hiện trong chương trình gốc.

Bước 3: **Kết nối chương trình bằng bộ kết nối.**
Nếu không có lỗi, bộ kết nối sẽ sinh ra một tệp đĩa có tên trùng với tên chương trình đích với đuôi .EXE, chứa một chương trình có thể chạy được.

Bước 4: **Chạy chương trình.**
Cần phải chạy thử chương trình sau khi kết thúc bước 3 để xem nó đã hoạt động đúng như mong muốn chưa. Nếu chưa thì phải quay lại từ bước 1 và tiến hành các thay đổi cần thiết trong mã gốc.

Kết luận

C là một công cụ lập trình mạnh, thông dụng và chuyển đổi được. Bài này giải thích các bước khác nhau trong qúa trình viết một chương trình C. Đó là chu trình **soạn thảo/dịch/kết nối/kiểm tra.**
Việc mắc lỗi trong khi phát triển chương trình là một điều không thể tránh khỏi. Chương trình dịch của C sẽ phát hiện các lỗi trong mã gốc và hiện một thông báo lỗi về kiểu lỗi và vị

trí của lỗi. Các thông tin này giúp người lập trình sửa lại mã gốc một cách dễ dàng. Tuy nhiên, chương trình dịch không phải lúc nào cũng thông báo được chính xác kiểu của lỗi cũng như vị trí của chúng. Đôi khi người lập trình phải vận dụng các kiến thức về C để phát hiện các sai sót đã gây ra thông báo lỗi.

Câu hỏi và trả lời

1. *Nếu muốn cho một người nào đó một chương trình C thì phải cho tệp nào?*
   C là một ngôn ngữ có chương trình dịch. Điều đó có nghĩa là, sau khi mã gốc đã được dịch sẽ phát sinh một chương trình có thể chạy được. Chương trình này có thể tự chạy một mình. Nếu muốn chuyển CHAO đến những người nào đó, điều duy nhất cần làm là hãy chuyển cho họ tệp CHAO.EXE. Họ không cần tệp gốc CHAO.C, hoặc tệp đích CHAO.OBJ và thậm chí ngay cả chương trình dịch C nữa.
2. *Sau khi đã tạo được tệp có thể chạy được, thì có cần phải giữ tệp gốc hoặc tệp đích nữa không?*
   Nếu tệp gốc bị mất thì sau này không còn cách nào khác để có thể thay đổi được chương trình đã dịch và kết nối. Do vậy nên giữ lại các tệp gốc. Nhưng đối với các tệp đích thì lại khác. Có một vài lý do cần phải giữ lại các tệp đích, tuy nhiên nếu nói ngay bây giờ thì còn hơi sớm. Do vậy, có thể xóa tệp đích một khi đã có tệp có thể chạy được. Nếu cần tệp đích, có thể dịch lại từ tệp gốc.
3. *Liệu có thể bỏ qua các thông báo nhắc nhở không?*
   Một số thông báo nhắc nhở không làm ảnh hưởng đến việc chạy của chương trình. Một số khác lại có. Nếu chương trình dịch cho ra một thông báo nhắc nhở, thì đó là một dấu hiệu báo rằng có một cái gì đó không hoàn toàn đúng. Hầu hết các chương trình dịch đều cho phép đặt mức nhắc nhở. Bằng cách đó thì chỉ nhận được các nhắc nhở từ mức này trở đi. Trong chương trình, cần phải xem từng nhắc nhở một và quyết định nên khắc phục nó hay là không. Tốt nhất là nên sửa lại mã gốc sao cho không còn một thông báo lỗi và nhắc nhở nào.

Luyện lập

Mục Luyện tập sẽ cho các câu hỏi để giúp người đọc củng cố lại kiến thức của mình và các bài tập nhằm cung cấp các kinh nghiệm khi sử dụng những điều đã học.

Câu hỏi

1. Hãy nêu ba lý do tại sao C là một lựa chọn tốt nhất về ngôn ngữ lập trình.
2. Chương trình dịch làm công việc gì?
3. Các bước trong chu trình phát triển chương trình là gì?
4. Cần phải gõ lệnh nào để dịch một chương trình có tên PROG1.C bằng chương trình dịch C ?
5. Chương trình dịch C kết nối và dịch bằng một lệnh hay bằng hai lệnh riêng rẽ?
6. Nên dùng đuôi nào cho các tệp gốc của C?
7. FILENAME.TXT có hợp lệ cho một tệp gốc của C không?
8. Nếu một chương trình đã được dịch và nó không làm việc như mong muốn, thì nên làm như thế nào ?
9. Mã máy là gì?

10. Bộ kết nối làm công việc gì?

Bài tập

1. Hãy nhập và dịch chương trình sau đây. Chương trình này làm việc gì? (không viết các số hiệu dòng khi soạn chương trình)

```
1:    include <stdio.h>
2:    int bankinh, dientich;
3:    main()
4:    {
5:       printf( "Nhập bán kính: " );
6:       scanf(  "%d", &bankinh );
7:       dientich = 3.14159 * bankinh * bankinh;
8:       printf( "\n\nDiện tích = %d", dientich );
9:       return 0;
10:   }
```

2. Nhập và dịch chương trình sau. Chương trình này làm việc gì?

```
1:  #include <stdio.h>
2:  int x,y;
3:  main()
4:  {
5:  for ( x = 0; x < 10; x++, printf( "\n" ) )
6:  for ( y = 0; y < 10; y++ )
7:  printf( "X" );
8:  return 0;
9:  }
```

3. Chương trình sau đây có một lỗi. Hãy nhập và dịch nó. Các dòng nào sinh ra thông báo lỗi?

```
1:  #include <stdio.h>
2:  main();
3:  {
4:  printf( "Hãy nhìn xem!" )
5:  printf( "Bạn sẽ tìm thấy nó!" );
6:  return 0;
7:  }
```

4. Chương trình sau đây cũng có lỗi. Hãy nhập và dịch nó. Các dòng nào sinh ra thông báo lỗi?

```
1:  #include <stdio.h>
2:  main()
3:  {
4:  printf( "Đây là một chương trình" )
5:  do_it( "có lỗi!" );
6:  return 0;
7:  }
```

5. Thay lệnh số 7 trong chương trình ở bài tập 2 như dưới đây. Dịch và chạy lại chương trình. Bây giờ chương trình này làm gì?

```
7:    printf( "%c", 1 );
```

Bài 2. Các thành phần của một chương trình C

Mọi chương trình C đều chứa một số các thành phần được liên kết với nhau theo một cách nhất định. Để có một bức tranh tổng thể, trước hết, ta bắt đầu xét một chương trình C ngắn nhưng hoàn chỉnh với tất cả các thành phần của nó.

Một chương trình C ngắn

Chương trình 2.1 là mã gốc của một chương trình có tên là NHAN.C. Đây là một chương trình rất đơn giản: nó chỉ bao gồm việc nhập hai số từ bàn phím và tính tích của chúng. Ở đây, chúng ta chưa cần phải hiểu tất cả các lệnh của chương trình, mà chủ yếu là làm quen với các thành phần của một chương trình C mà thôi.

Trước khi xét chi tiết chương trình này, cần phải hiểu rõ khái niệm về hàm (function), vì hàm là một yếu tố cơ bản trong việc lập trình C. Một hàm là một bộ phận độc lập của chương trình, được đặt tên và thực hiện một công việc nào đó. Bằng việc gọi tên một hàm, chương trình chính có thể thực hiện các lệnh trong hàm này. Chương trình chính có thể gửi các thông tin, được gọi là đối số (argument) cho hàm, và hàm có thể đưa trở lại thông tin cho chương trình. Có hai kiểu hàm: các hàm thư viện - một bộ phận của chương trình dịch của C; và các hàm do người sử dụng tự định nghĩa.

Chương trình 2.1. NHAN.C.

```
1:  /* Chương trình  tính tích của hai số. */
2:  #include<stdio.h>
3:  int a,b,c;
4:  int tich(int x, int y);
5:  main()
6:  {
7:    /* Nhập số thứ nhất */
8:    printf("Nhập một số nằm giữa 1 và 100: ");
9:    scanf("%d", &a);
10
11:   /* Nhập số thứ hai */
12:   printf("Nhập một số khác nằm giữa 1 và 100: ");
13:   scanf("%d", &b);
14:
15:   /* Tính và hiện kết qủa */
16:   c = tich(a,b);
17:   printf( "\n%d nhân %d = %d", a, b, c);
18: }
19:
20: /* Hàm tính tích của hai đối số */
21: int tich(int x, int y)
22: {
23:   return ( x * y );
24: }
```

Chương trình này sẽ hiện trên màn hình các dòng kết qủa sau:

```
 Nhập một số nằm giữa 1 và 100: 35
```

```
Nhập một số khác nằm giữa 1 và 100: 23
35 nhân 23 = 805
```

Các thành phần của chương trình

Sau đây là mô tả của các thành phần khác nhau của chương trình mẫu nói trên. Các số hiệu dòng không phải là thành phần của chương trình mà chỉ được đưa vào để tiện việc theo dõi và phân tích.

Hàm *main()* (Các dòng 5-18)

Bộ phận duy nhất cần phải có trong mọi chương trình C là hàm `main()`. Dạng đơn giản nhất của nó là: *main* theo sau là một cặp dấu ngoặc tròn ( ) và một cặp dấu ngoặc nhọn {}. Nằm giữa các dấu ngoặc nhọn là các lệnh tạo nên thân chính của chương trình. Dưới điều kiện bình thường, chương trình sẽ bắt đầu thực hiện từ lệnh thứ nhất trong *main()* và kết thúc bằng lệnh cuối cùng trong *main()*.

Chỉ thị *#include* (Dòng 2)

Chỉ thị #include báo cho chương trình dịch của C biết phải thêm vào chương trình một tệp trong khi dịch. Một tệp được thêm vào là một tệp đĩa riêng biệt chứa các thông tin cần thiết cho chương trình dịch. Một số tệp trong các tệp này (đôi khi còn gọi là các tệp tiêu đề - header files) được cung cấp cùng với chương trình dịch. Tất cả các tệp này đều có đuôi là .H.

Chỉ thị #include trong ví dụ này có nghĩa là "Cộng thêm nội dung của tệp stdio.h". Hầu hết các chương trình C đều yêu cầu một hoặc nhiều tệp cộng thêm.

Định nghĩa biến

Một biến là một tên gọi cho một đại lượng hoặc một đối tượng nào đó. Mỗi biến đều được phân bổ một địa chỉ trong bộ nhớ. Chương trình C sử dụng các biến để lưu giữ các loại dữ liệu khác nhau trong quá trình thực hiện chương trình. Trong C, một biến chỉ có thể được sử dụng sau khi đã được định nghĩa. Một định nghĩa biến cho chương trình dịch biết tên của biến và kiểu của dữ liệu mà nó lưu giữ. Trong ví dụ mẫu, định nghĩa ở dòng 3, `int a,b,c,` định nghĩa ba biến có tên là `a`, `b`, và `c` và mỗi biến sẽ chứa một giá trị nguyên (integer). Trong Bài 3 sẽ nói rõ hơn về các biến và các định nghĩa biến.

Khai báo hàm (Dòng 4)

Một khai báo hàm cho chương trình dịch biết tên và đối số của hàm sẽ được sử dụng trong chương trình. Khai báo này phải xuất hiện trước khi hàm được sử dụng. Khai báo hàm khác với định nghĩa hàm. Một định nghĩa hàm chứa các lệnh tạo ra hàm đó.

Các lệnh chương trình (Các dòng 8, 9, 12, 13, 16, 17, 23)

Công việc thực sự của một chương trình C được thực hiện bởi các câu lệnh của nó. Các câu lệnh của C là: hiện các thông tin trên màn hình, đọc các phím được gõ vào từ bàn phím, thực hiện các phép số học, gọi các hàm, đọc các tệp đĩa, và tất cả các thao tác khác mà một chương trình cần phải thực hiện. Từ bài này trở đi, mỗi một lệnh C được viết riêng trên

một dòng và luôn luôn được kết thúc bằng một dấu chấm phẩy (;). Sau đây là các giải thích ngắn gọn cho các câu lệnh của chương trình NHAN.C.

printf()

Lệnh `printf()` (các dòng 8, 12, và 17) là một hàm thư viện dùng để hiện thông tin lên màn hình. Lệnh này có thể dùng để hiện một thông báo văn bản đơn giản (như ở dòng 8 và 12) hoặc một thông báo và một hoặc nhiều biến chương trình (như ở dòng 17).

scanf()

Lệnh `scanf()` (các dòng 9 và 13) cũng là một hàm thư viện dùng để đọc dữ liệu từ bàn phím và gán dữ liệu này cho một hoặc nhiều biến chương trình.

c = tich (a, b);

Lệnh này gọi một hàm có tên là `tich()`. Nghĩa là, nó thực hiện các lệnh chương trình chứa trong hàm `tich()`. Nó còn gửi các đối số `a` và `b` cho hàm. Sau khi các lệnh trong hàm `tich()` được hoàn tất, `tich()` còn đưa ra một giá trị cho chương trình. Giá trị này được gán và lưu giữ trong biến có tên là `c`.

return (x * y);

Lệnh này là một bộ phận của hàm `tich()`. Nó tính tích của các biến `x` và `y` rồi đưa kết qủa trở lại cho chương trình đã gọi `tich()`.

Định nghĩa hàm (Các dòng 21-24)

Một hàm là một đoạn mã gốc độc lập thực hiện một nhiệm vụ nhất định. Mỗi hàm đều có một tên, và các mã gốc của hàm này được thực hiện mỗi khi tên hàm được gọi trong một chương trình.

Hàm có tên `tich()` ở các dòng 21-24 là một hàm do người sử dụng tự định nghĩa và tự viết trong quá trình phát triển chương trình. Hàm này rất đơn giản: nó chỉ làm mỗi một việc là nhân hai giá trị với nhau và đưa trở lại kết qủa cho chương trình đã gọi nó.

C còn có các hàm thư viện được cung cấp cùng với chương trình dịch. Các hàm thư viện thực hiện hầu hết các công việc cơ bản nhất (như các thao tác trên màn hình, bàn phím, đọc/ghi đĩa) mà một chương trình cần đến. Trong ví dụ này, `printf()` và `scanf()` là các hàm thư viện.

Các chú thích trong chương trình (Các dòng 1, 7, 11, 15, 20)

Bất kỳ phần nào mà khởi đầu bằng `/*` và kết thúc bằng `*/` đều được gọi là một chú thích. Chương trình dịch bỏ qua tất cả các chú thích và như vậy thì chúng tuyệt đối không ảnh hưởng đến sự hoạt động của một chương trình. Có thể viết bất cứ thứ gì trong một chú thích và nó sẽ không ảnh hưởng đến hoạt động của chương trình. Một chú thích có thể chỉ chiếm một phần nào đó của một dòng, toàn bộ một dòng, hoặc cũng có thể nhiều dòng. Tuy nhiên, không nên lồng chú thích này trong một chú thích khác. Hầu hết các chương trình dịch đều không chấp nhận loại hình này.

Nhiều người mới bắt đầu học lập trình cho rằng không cần đến chú thích và làm như vậy là mất thì giờ. Thực ra thì không hẳn là như vậy. Các thao tác trong chương trình có thể rất rõ ràng và dễ theo dõi trong khi đang viết. Nhưng khi chương trình trở nên dài hơn và phức tạp dần lên, hoặc khi cần phải sửa đổi một chương trình đã viết từ nhiều tháng trước thì các chú thích thật là có giá trị. Rõ ràng là cần phải tạo ra một thói quen dùng các chú thích để làm sáng tỏ tất cả các cấu trúc và các hoạt động trong chương trình.

Các dấu ngoặc nhọn (Các dòng 6, 18, 22, 24)

Các dấu ngoặc nhọn ({}) dùng để nhóm các dòng lệnh thành một khối. Các lệnh của một hàm, kể cả hàm `main()`, phải được để trong các dấu ngoặc này.

Chạy chương trình

1. Chuyển vào thư mục chứa chương trình dịch C.
2. Khởi động bộ soạn thảo.
3. Nhập mã gốc của NHAN.C (lưu ý, không nhập các số hiệu dòng).
4. Cất giữ tệp chương trình.
5. Dịch và kết nối chương trình bằng cách ra các lệnh thích hợp với chương trình dịch hiện có. Nếu không có sai sót gì trong qúa trình dịch cũng như kết nối, chạy chương trình bằng lệnh run, hoặc bằng cách gõ `nhan` tại dấu nhắc của DOS.
6. Nếu xuất hiện một hoặc nhiều thông báo lỗi, quay lại từ bước hai và sửa các lỗi đã mắc phải.

Kết luận

Bài này tuy ngắn nhưng rất quan trọng. Nó cho biết cấu trúc và các thành phần của một chương trình C. Bộ phận duy nhất mà một chương trình C yêu cầu là hàm `main()`. Công việc thực sự của chương trình được thực hiện bởi các lệnh chương trình. Các lệnh này hướng dẫn cho máy tính thực hiện các hành động mà chương trình mong muốn. Bài này còn nói về các biến và các định nghĩa biến, và chỉ ra lợi ích của việc sử dụng các chú thích và cách dùng chúng.

Ngoài hàm `main()` ra, một chương trình C còn có thể sử dụng hai kiểu hàm khác: các hàm thư viện được cung cấp cùng với bộ chương trình dịch và các hàm do người sử dụng tự định nghĩa.

Câu hỏi và trả lời

1. *Các chú thích có tác động gì trên một chương trình?*
   Các chú thích là để cho người lập trình. Khi chương trình dịch chuyển các mã gốc thành các mã đích, nó sẽ vứt bỏ các chú thích và các vùng trắng. Điều đó có nghĩa là chúng không hề có một tác động nào lên chương trình có thể chạy được. Các chú thích có làm cho chương trình gốc dài hơn, nhưng điều đó cũng không đáng kể lắm. Về lâu dài, nên sử dụng các chú thích và các khoảng trống để chương trình gốc dễ đọc và dễ hiểu hơn.
2. *Sự khác nhau giữa một lệnh và một khối là gì?*
   Một khối là một nhóm các lệnh được đặt giữa các dấu ngoặc nhọn. Một khối có thể được sử dụng ở hầu hết các vị trí mà tại đó có thể sử dụng một lệnh.
3. *Làm thế nào để tìm thấy danh sách các hàm thư viện có thể sử dụng?*
   Nhiều chương trình dịch có kèm theo tài liệu mô tả các hàm thư viện. Chúng thường được xếp theo thứ tự từ điển.

Luyện tập

Câu hỏi

1. Thuật ngữ nào dùng để chỉ một nhóm của một hoặc nhiều lệnh C được để giữa các dấu ngoặc nhọn?
2. Thành phần nào phải luôn luôn có mặt trong một chương trình C?
3. Làm thế nào để thêm vào các chú thích chương trình và tại sao phải dùng chúng?
4. Một hàm là gì?
5. Trong C có hai kiểu hàm. Các hiểu đó là kiểu gì và chúng khác nhau như thế nào?
6. Chỉ thị `#include` dùng để làm gì?

Bài tập

I. Hãy viết một chương trình C ngắn nhất.

II. Dùng chương trình sau đây để trả lời cho các câu hỏi sau:

A. Các dòng nào chứa các lệnh?
B. Các dòng nào chứa các định nghĩa biến?
C. Các dòng nào chứa các khai báo hàm?
D. Các dòng nào chứa các định nghĩa hàm?
E. Các dòng nào chứa các chú thích?

```
1: /* BT2-2.C */
2: #include <stdio.h>
3: void in_dong(void);
4: main()
5: {
6:    in_dong();
7:    printf("\n Tự dạy chương trình C trong 21 ngày!\n";
8:    in_dong();
9:    return 0}
10:/* In một dòng dấu sao */
11:void in_dong(void)
12:{
13:   int dem;
14:   for(dem = 0; dem = 21; dem ++)
15:         printf("*");
16:}
17:/* kết thúc chương trình */
```

III. Chương trình sau đây làm việc gì?

```
1: /*BT2-3.C */
2: #include<stdio.h>
3: main()
4: {
5:    int ctr;
6:    for ( ctr=65; ctr,91; ctr++ );
7:         printf( "%c", ctr );
8:    return 0;
9: }
10:/* kết thúc chương trình */
```

IV. Chương trình sau đây làm việc gì?

```
1: /* BT2-4.C */
2: #include<stdio.h>
3: main()
4: {
5:   char buffer(256);
6:   printf( "Hãy viết tên của bạn và ấn <Enter>:\n";
7:   gets(buffer);
8:   printf( "\nTên của bạn có %d chữ cái và dấu cấch!",
     strlen( bufferr);
9:   return 0;
10:}
```

Bài 3. Các biến và các hằng

Các chương trình máy tính thường làm việc với các kiểu dữ liệu khác nhau và cần một cách nào đó để lưu giữ các giá trị sẽ phải sử dụng. Các giá trị này có thể là các số hoặc các ký tự. C có hai cách để lưu giữ các giá trị số - *biến (variable) và hằng (contstant)* - với nhiều tuỳ chọn khác nhau cho mỗi cách. Một biến là một tên gọi và được phân bổ một địa chỉ lưu trữ. Địa chỉ này được dùng để lưu giữ giá trị của biến. Giá trị này có thể thay đổi trong khi chương trình thực hiện. Ngược lại, một hằng có một giá trị cố định không được thay đổi.

Trước khi đi vào các biến, cần biết qua một chút về sự hoạt động của bộ nhớ máy tính.

Bộ nhớ của máy tính

Máy tính dùng một bộ nhớ truy nhập ngẫu nhiên (random access memory - RAM) để lưu giữ thông tin trong khi đang hoạt động. RAM được đặt trong các mạch tích hợp, hay còn gọi là chip, nằm bên trong máy tính. Các thông tin lưu trong RAM thường xuyên có thể bị xóa và được thay thế bằng các thông tin mới. Ngoài ra RAM chỉ nhớ thông tin khi máy đang chạy và các thông tin sẽ bị mất khi tắt máy.

Một byte là đơn vị cơ bản của bộ nhớ dữ liệu của máy tính. Các kiểu dữ liệu khác nhau được lưu giữ bằng các số lượng byte khác nhau. Bảng 3.1. cho thấy sự khác nhau này.

Bảng 3.1. Số lượng bytes cần thiết để lưu giữ dữ liệu

Dữ liệu	*Số bytes cần thiết*
Chữ cái x	1
Số 100	2
Số 120.145	4
Cụm từ *Tự học ngôn ngữ C*	17
Một trang đánh máy	3000 (gần đúng)

RAM trong máy tính được tổ chức theo kiểu tuần tự, byte này tiếp byte kia. Mỗi byte có một địa chỉ duy nhất để định danh cho nó và phân biệt nó với tất cả các byte khác trong bộ nhớ. Các địa chỉ được gán cho các vị trí bộ nhớ một cách tuần tự, khởi đầu từ 0 cho đến hết bộ nhớ của máy.

Các biến

Một biến là một tên gọi và được phân bổ một địa chỉ lưu trữ. Địa chỉ này được dùng để lưu giữ giá trị của biến. Giá trị này có thể thay đổi trong khi chương trình thực hiện.

Các tên biến

Trong C, các tên biến phải tuân theo các qui tắc sau:

- Tên biến có thể chứa các chữ, các chữ số và dấu gạch dưới (_).
- Ký tự đầu tiên của tên phải là một chữ. Có thể dùng dấu gạch dưới như là ký tự đầu tiên, nhưng không nên làm như vậy vì xem ra không được hợp lý cho lắm.
- Chữ hoa và chữ thường là khác nhau. Tức là, `Count` và `count` là hai biến khác nhau.

- Các từ giành riêng (keyword) của C không được sử dụng làm tên biến. Một keyword là một từ mà C sử dụng như là một thành phần của nó.

Sau đây là một số ví dụ về các tên biến hợp lệ và không hợp lệ trong C:

```
phantram          /* hợp lệ */
y2x5_fg7h         /* hợp lệ */
loinhuan_hangnam /* hợp lệ */
_1995_thue        /* hợp lệ, nhưng không nên dùng */
Taikhoan#tk       /* không hợp lệ, vì ký tự # không hợp lệ */
double            /* không hợp lệ, vì là keyword của C */
9thang            /* không hợp lệ, vì ký tự đầu tiên là số */
```

Vì C phân biệt chữ hoa với chữ thường, nên `phantram`, `Phantram`, và `PHANTRAM` là tên của ba biến khác nhau. Dù rằng không bắt buộc, nhưng các tên biến trong C thường được viết bằng chữ thường, còn tên của các hằng thì được viết bằng chữ hoa.

Đối với nhiều chương trình dịch, một tên biến của C có thể dài tới 31 ký tự. Với độ dài như vậy, có thể tạo ra các tên biến nói lên được ý nghĩa của dữ liệu được lưu giữ trong chúng.

Các tên có bao hàm ý nghĩa thường gồm nhiều từ. Có thể dùng dấu gạch dưới để phân cách các từ hoặc các cụm từ với nhau cho dễ hiểu. Còn một kiểu khác gọi là bướu lạc đà. Thay cho dấu cách, người ta viết hoa chữ đầu tiên của mỗi từ. Ví dụ, `TyLeLaiSuat`, hoặc `TyleLaisuat`. Kiểu này có vẻ thuận tiện hơn vì không phải gõ thêm dấu gạch dưới.

Các kiểu biến số

C cung cấp một số kiểu biến số khác nhau. Tại sao lại phải cần một số kiểu biến khác nhau? Đó là vì, các giá trị số khác nhau yêu cầu số bytes để lưu giữ khác nhau và các phép toán số học có thể thực hiện trên chúng cũng khác nhau. Các số nguyên nhỏ (ví dụ, 1, 19, -8) yêu cầu ít byte để lưu giữ, và các phép toán số học (cộng, nhân, v.v.) trên các số này có thể được thực hiện rất nhanh bằng máy tính. Ngược lại, những số nguyên lớn và các số dấu chấm động (ví dụ, 123000000 hoặc 0.000000345213) yêu cầu nhiều bytes hơn và thời gian tính toán với chúng cũng lâu hơn. Bằng cách dùng các kiểu biến số thích hợp, chương trình sẽ chạy một cách hiệu qủa hơn.

Các biến số của C rơi vào hai nhóm chính sau đây:

- Các biến nguyên (interger) dùng để lưu giữ các giá trị không có phần thập phân. Các biến nguyên lại phân thành hai lớp: biến nguyên có dấu để ghi các giá trị nguyên dương và nguyên âm; biến nguyên không dấu chỉ để ghi các giá trị không âm (bao gồm cả số 0).
- Các biến dấu chấm động (Floating-point) giữ các giá trị có phần thập phân (số thực). Lưu ý một điểm là, trước đây ta vẫn quen dùng dấu phẩy để ngăn cách phần nguyên và phần phân của một số, nên mới có tên gọi *dấu phẩy động*, nay theo chuẩn của ANSI thì người ta dùng dấu chấm thập phân, bởi vậy mới có tên gọi *dấu chấm động*. Từ nay trở đi sẽ dùng dấu chấm thập phân thay cho dấu phẩy thập phân. Còn dấu phẩy được dùng làm dấu ngăn cách hàng nghìn, hàng triệu, hàng tỷ, v.v.

Bảng 3.2. Các kiểu dữ liệu số của C

Kiểu biến	*Keyword*	*Số bytes*	*Miền giá trị*
ký tự	`char`	1	-128 đến 127
nguyên	`int`	2	-32768 đến 32767
nguyên ngắn	`short`	2	-32768 đến 32767
nguyên dài	`long`	4	-2147483648 đến 2147483647
ký tự không dấu	`unsigned` char	1	0 đến 255
nguyên không dấu	`unsigned` int	2	0 đến 65535
nguyên ngắn kh. dấu	`unsigned` short	2	0 đến 65535
nguyên dài kh. dấu	`unsigned` long	4	0 đến 4294967295
dấu chấm động độ chính xác đơn	`float`	4	1.2E-38 đến 3.4E38*
dấu chấm động độ chính xác gấp đôi	`float`	8	2.2E-308 đến 1.8E308**

* Miền gần đúng; độ chính xác= 7 chữ số.
** Miền gần đúng; độ chính xác= 19 chữ số.

Miền gần đúng trong bảng này có nghĩa là các giá trị cao nhất và thấp nhất mà một biến đã cho có thể nhận. Độ chính xác nghĩa là đúng đến bao nhiêu chữ số của giá trị được ghi trong biến đã cho. (Ví dụ, nếu tính 1/3, thì kết qủa đúng là 0.333333.... với các số 3 cứ tiếp tục đến vô hạn. Một biến với độ chính xác đơn chỉ ghi được 7 con 3.)
Trong bảng này, các kiểu `int` và `short int` là giống nhau. Nhưng tại sao lại phải có hai kiểu khác nhau như vậy? Các kiểu biến `int` và `short int` là giống hệt nhau trên các hệ thống tương thích với các máy tính IBM-PC 16-bit, nhưng chúng có thể sẽ khác nhau trên các phần cứng kiểu khác. Trên một hệ VAX, một biến `short` và một biến `int` không cùng cỡ với nhau. Một biến `short` chiếm 2 bytes, nhưng một biến `int` lại chiếm những 4 bytes. Do vậy, đối với các máy PC, hai kiểu này là như nhau.

Các khai báo biến

Mọi biến cần phải được khai báo trước khi được đem ra sử dụng. Một khai báo biến cho chương trình dịch biết *tên* và *kiểu* của một biến và có thể khởi tạo cho biến một giá trị nào đó. Một khai báo biến có dạng sau:

tênkiểu tênbiến

Tênkiểu xác định kiểu của biến và phải là một trong các keyword đã cho trong Bảng 3.2. *Tênbiến* là tên của biến được đặt theo các qui tắc đã nói ở phần trước. Có thể khai báo nhiều biến cùng một kiểu trên một dòng, mỗi biến cách nhau bằng một dấu phẩy. Ví dụ:

```
int tuoi, luong, con; /* ba biến nguyên */
float phantram, tong;  /* hai biên động */
```

Keyword *typedef*

Từ khóa *typedef* được sử dụng để tạo lập một tên mới cho một kiểu dữ liệu đã có. Nói một cách chính xác, *typedef* tạo ra một kiểu đồng nghĩa. Ví dụ, lệnh

```
typedef int integer;
```

tạo ra kiểu `integer` từ kiểu `int`. Sau đó, có thể dùng `integer` để định nghĩa các biến kiểu `int`.

Khởi tạo các biến số

Khi một biến được khai báo, chương trình dịch sẽ dành ra một vùng nhớ cho biến này. Tuy nhiên, giá trị được nhớ tại vùng này chưa được xác định. Nó có thể là 0, hoặc có thể là một giá trị ngẫu nhiên nào đó còn lại sau khi vùng này được giải phóng. Bởi vậy, trước khi một biến được sử dụng, nó phải được khởi tạo bằng một giá trị xác định nào đó. Việc này có thể thực hiện hoàn toàn độc lập với việc khai báo biến bằng cách dùng một lệnh gán. Ví dụ:

```
int count; /* Dành một vùng nhớ cho biến count */
count = 0; /* Ghi 0 vào count */
```

Có thể khởi tạo một biến ngay khi khai báo bằng cách viết dấu bằng và giá trị khởi tạo sau tên biến. Ví dụ

```
int count = 0;
```

```
double phantram = 0.01, tyle_thue = 20.5;
```

Các hằng

Giống như một biến, một hằng cũng là một địa chỉ trong bộ nhớ được sử dụng bởi một chương trình. Nhưng không giống như một biến, giá trị được ghi trong một hằng không được thay đổi trong quá trình thực hiện của chương trình. C có hai kiểu hằng, mỗi kiểu lại có các cách sử dụng khác nhau.

Các hằng thực sự (literal constants)

Một *hằng literal* là một giá trị được gõ trực tiếp vào chương trình gốc bất cứ khi nào nó được cần đến. Đây là hai thí dụ:

```
int count = 20;
float tyle_thue = 0.20;
```

Các giá trị `20` và `.20` là các hằng literal. Các lệnh trên lưu giữ các giá trị này vào các biến `count` và `tyle_thue`.

Một hằng literal với dấu chấm thập phân là một hằng dấu chấm động và được chương trình dịch C xem như một số có độ chính xác gấp đôi.

Một hằng không có dấu chấm thập phân được chương trình dịch biểu diễn như một số nguyên. Các hằng nguyên có thể được viết theo ba dạng dưới đây:

- Một hằng khởi đầu bằng một chữ số khác 0 được xem là số nguyên cơ số 10 (*decimal*). Một hằng nguyên hệ 10 có thể chứa các số 0-9 và một dấu trừ hoặc + ở đằng trước.
- Một hằng khởi đầu bằng chữ số 0 được xem là một số nguyên cơ số 8 (*octal*). Các hằng hệ 8 có thể chứa các chữ số 0-7 và phía trước có thể có thêm dấu trừ hoặc cộng.
- Một hằng khởi đầu bằng 0x hoặc 0X được xem là một hằng hệ cơ số 16 (*hexadecimal*). Các hằng hệ 16 có thể chứa các chữ số 0-9 và các chữ A, B, C, D và F, và một dấu trừ hoặc cộng ở đằng trước.

Các hằng ký hiệu (symbolic constants)

Một hằng ký hiệu là một hằng được ký hiệu bởi một tên gọi (symbolic) nào đó. Giống như một hằng *literal*, giá trị của một hằng *symbolic* không thể thay đổi. Giá trị thực sự của hằng symbolic chỉ được nhập vào khi nó được định nghĩa. Việc sử dụng một hằng kiểu này y hệt như sử dụng một biến.

Hằng ký hiệu có hai ưu điểm nổi bật so với hằng thực sự. Ví dụ, để tính chu vi và diện tích của một hình tròn khi biết bán kính, có thể viết như sau:

```
chu_vi = 3.14 * (2 * ban_kinh);
dien_tich = 3.14 * ban_kinh * ban_kinh;
```

Tuy nhiên, nếu định nghĩa một hằng ký hiệu với tên là `PI` và giá trị là `3.14`, thì đoạn chương trình trên có thể thay bằng:

```
chu_vi = PI * (2 * ban_kinh);
dien_tich = PI * ban_kinh * ban_kinh;
```

Hai lệnh này trở nên sáng sủa và dễ hiểu hơn. Thay vì phải băn khoăn về giá trị `3.14` có nghĩa là gì, ở đây có thể hiểu ngay được PI là gì.

Ưu điểm thứ hai của hằng ký hiệu là ở chỗ dễ thay đổi. Tiếp tục ví dụ trên, để cho kết qủa của chương trình chính xác hơn cần phải thay giá trị của PI là 3.14 bằng một giá trị khác với độ chính xác cao hơn, chẳng hạn 3.14159. Nếu đã chót dùng một hằng thực sự cho số PI, thì bây giờ phải duyệt lại tất cả chương trình để thay các giá trị 3.14 bằng 3.14159. Với một hằng ký hiệu, chỉ cần thay ở đúng một chỗ, nơi hằng này được định nghĩa.

C có hai phương pháp dùng để định nghĩa một hằng ký hiệu: chỉ thị `#define` và từ khóa `const`. Chỉ thị `#define` là một trong những chỉ thị tiền xử lý sẽ được trình bầy trong các bài sau. Chỉ thị này được sử dụng như sau:

```
#define TENHANG giatri
```

Dòng này tạo ra một hằng có tên là `TENHANG` với giá trị là `giatri`. Trong đó `giatri` chính là một hằng số, còn `TENHANG` tuân theo các qui luật giống như của tên biến. Để cho dễ phân biệt giữa một hằng và một biến, ta nên viết các tên hằng bằng chữ hoa, còn tên biến viết bằng chữ thường. Trong ví dụ trước, để định nghĩa hằng ký hiệu PI, có thể viết như sau:

```
#define PI 3.14159
```

Chú ý là, chỉ thị `#define` không kết thúc bằng dấu chấm phẩy (;).`#define` có thể được đặt ở bất cứ chỗ nào trong chương trình gốc, nhưng chỉ có hiệu lực cho phần mã gốc đứng sau nó mà thôi. Nói chung, người ta hay gom tất cả các `#define` lại với nhau và đặt chúng ở đầu tệp ngay trước hàm `main()`.

Cách thứ hai để định nghĩa một hằng ký hiệu là dùng từ khóa `const`. Một biến được khai báo là `const` thì không thể thay đổi giá trị trong khi chương trình thực hiện. Đây là một vài ví dụ:

```
const int count = 100;
const float pi= 3.14159;
const long no= 12000000, float tyle_thue = 0.21;
```