ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Phan Văn Tân

Ngôn ngữ lập trình Fortran 90

HÀ NỘI – 2005

MỤC LỤC

LỜI GIỚI THIỆU

Trong những năm gần đây, cùng với sự phát triển mạnh mẽ của Công nghệ Thông tin và Điện tử Viễn thông, nhiều chương trình, phần mềm máy tính đã ra đời và được ứng dụng rộng rãi, góp phần thúc đẩy sự phát triển kinh tế, xã hội. Trong số đó, các ngôn ngữ lập trình cũng ngày càng được phát triển và phổ biến. Ngôn ngữ lập trình Fortran cũng không phải là một ngoại lệ. Từ những phiên bản đầu tiên với nhiều hạn chế cho đến nay Fortran luôn là một trong những ngôn ngữ thông dụng rất được ưa chuộng trong lập trình giải các bài toán khoa học kỹ thuật. Với nhiều thế mạnh vượt trội so với các ngôn ngữ lập trình khác, Fortran thường được ứng dụng để giải các bài toán lớn, đòi hỏi phải xử lý tính toán nhiều, nhất là tính toán song song.

Trước những năm chín mươi của thế kỷ hai mươi, khi mà thế hệ máy PC hãy còn mới lạ ở Việt Nam, hầu như các bài toán ứng dụng đều được chạy trên các máy tính lớn (MINSK–32, EC–1022, EC–1035, IBM–360,…) với các chương trình thường được lập bằng ngôn ngữ Fortran. Song, khi các máy PC ngày càng phổ biến hơn, với nhiều phần mềm tiện dụng đi kèm, thêm vào đó là sự đòi hỏi về cấu hình máy tính của Fortran, ngôn ngữ Fortran hầu như đã bị lãng quên trong một thời gian khá dài. Nhiều người đã phải thay đổi thói quen sử dụng Fortran, tự thích ứng bằng cách chuyển sang tiếp cận với các ngôn ngữ lập trình khác hoặc chuyển hướng nghiên cứu. Sự thiếu thông tin cập nhật đã làm nhiều người tưởng rằng Fortran là một ngôn ngữ "cổ" rồi, không ai dùng nữa. Nhưng không phải như vậy. Trước sự đòi hỏi phải giải quyết những bài toán lớn (chúng tôi muốn nhấn mạnh lớp các bài toán khoa học kỹ thuật), chạy ở chế độ thời gian thực (Real–time), Fortran đã ngày càng được phát triển và hoàn thiện với nhiều đặc điểm mới. Điều đó đã cuốn hút nhiều người quay về với Fortran. Một lý do khác có tác động không nhỏ, khiến người ta tiếp tục lựa chọn ngôn ngữ lập trình Fortran là quá trình quan hệ hợp tác quốc tế. Khi làm việc với các đối tác nước ngoài, trong nhiều lĩnh vực hầu hết các chương trình được viết bằng ngôn ngữ Fortran, nếu không biết về nó, đồng nghĩa với việc đôi bên không cùng "tiếng nói"; và do đó có thể dẫn đến sự bất lợi, kém hiệu quả khi làm việc với nhau.

Nhận thức được tầm quan trọng của vấn đề này, những năm gần đây, ngôn ngữ lập trình Fortran đã được đưa vào chương trình đào tạo của một số khoa trong trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội. Mặt khác, đối với nhiều nhà khoa học, hiện nay ngôn ngữ Fortran đã trở thành một trong những công cụ làm việc không thể thiếu, và tất nhiên trong số đó có chúng tôi.

Bởi vậy, quyển sách này ra đời với kỳ vọng của chúng tôi là cung cấp cho bạn đọc những kiến thức cơ bản nhất về ngôn ngữ lập trình Fortran 90. Qua đó bạn đọc có thể ứng dụng nó một cách hiệu quả trong các lĩnh vực chuyên môn của mình. Quyển sách có thể được dùng làm giáo trình giảng dạy ở bậc đại học và sau đại học cho ngành Khí tượng Thủy văn và Hải dương học, trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà

Nội. Tuy nhiên chúng tôi cũng mong muốn nó sẽ giúp cho sinh viên các bậc đào tạo thuộc các ngành khoa học khác, như Vật lý học, Hóa học, Toán học trong trường Đại học Khoa học Tự nhiên có thêm một tài liệu tham khảo bổ ích trong quá trình học tập tại trường. Quyển sách cũng có thể làm tài liệu tham khảo cho các cán bộ, kỹ sư, các nhà nghiên cứu thuộc nhiều lĩnh vực khác nhau.

Trong quá trình biên soạn quyển sách, một số đồng nghiệp đã đề xuất chúng tôi đưa thêm vào phần đồ họa của Fortran. Một số khác lại đề nghị gắn phần giao diện giữa những kết quả tính toán kết xuất với một số phần mềm đồ họa khác, như GrADS, NCAR Graphics,... Chúng tôi xin chân thành cám ơn và ghi nhận những ý kiến đóng góp quí báu đó. Nhận thấy rằng phần đồ họa của Fortran chỉ được tích hợp trong một số phiên bản chạy trên môi trường Microsoft Windows; còn để gắn kết các file kết xuất của Fortran với các phần mềm đồ họa khác ít nhất cần phải có một số kiến thức cơ bản về các phần mềm này. Vì khuôn khổ quyển sách có hạn, chúng tôi sẽ cố gắng trình bày những nội dung trên trong một ấn phẩm khác trong tương lai.

Mặc dù đã cố gắng chuyển tải nội dung quyển sách sao cho có thể đáp ứng được nhiều đối tượng, từ những người mới làm quen cho đến những người đã từng có quá trình làm việc nhất định với ngôn ngữ Fortran, với bố cục từ dễ đến khó, từ đơn giản đến phức tạp, song do còn nhiều hạn chế về kinh nghiệm và kiến thức, quyển sách cũng không tránh khỏi những khiếm khuyết. Chúng tôi rất mong nhận được sự đóng góp ý kiến của tất cả các bạn đọc.

Để hoàn thành quyển sách này, chúng tôi nhận được sự hỗ trợ cả về tinh thần và vật chất từ phía trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội, đặc biệt từ các đồng nghiệp thuộc Khoa Khí tượng Thủy văn và Hải dương học của trường, nơi chúng tôi gắn bó trong công tác giảng dạy và hoạt động khoa học hàng chục năm nay. Nhân đây chúng tôi xin bày tỏ lòng biết ơn chân thành và lời cám ơn sâu sắc.

Hà Nội, 2–2005

Tác giả

MỞ ĐẦU

Tập hợp các qui tắc đặc biệt để mã hoá những kiến thức cho máy tính hiểu được gọi là ngôn ngữ lập trình. Có rất nhiều ngôn ngữ như vậy, ví dụ FORTRAN, BASIC, Pascal, C,... FORTRAN là tên cấu tạo từ FORmula TRANslation (diễn dịch công thức, hay còn gọi là công thức dịch), là một trong những ngôn ngữ lập trình bậc cao đầu tiên. Nó có thể sử dụng những tên tượng trưng để biểu diễn định lượng toán học và viết các công thức toán học dưới dạng thức hợp lý có thể hiểu được, như X = (–B+DELTA)/(2*A). Ý tưởng của FORTRAN được John Backus đề xuất vào khoảng cuối năm 1953 ở New York, và chương trình FORTRAN đầu tiên đã được chạy vào tháng 4 năm 1957.

Kể từ đó, việc sử dụng FORTRAN đã nhanh chóng được phổ biến rộng rãi. Điều đó đòi hỏi cần phải sớm tiêu chuẩn hoá nó sao cho chương trình viết ra phải bảo đảm chạy được ở mọi nơi. Vào năm 1966, lần đầu tiên phiên bản chuẩn của ngôn ngữ lập trình này được ấn hành. Phiên bản này, như đã biết, là Fortran 66 (chính xác hơn là FORTRAN 66, nhưng thực tế người ta cho cách viết hoa là không trang trọng). Phiên bản chuẩn mới sau đó, Fortran 77, được ấn hành vào năm 1978. Không bằng lòng với sự cạnh tranh của các ngôn ngữ mới khác, như Pascal và C, FORTRAN tiếp tục phát triển một cách mạnh mẽ. Và phiên bản chuẩn gần đây, FORTRAN 90 (hoặc Fortran 90), với nhiều đặc tính đột phá, đã ra đời vào tháng 8 năm 1991. Cho đến nay, FORTRAN đã phát triển đến những phiên bản mới hơn, như FORTRAN 95, FORTRAN 2003. Trong khuôn khổ quyển sách này chúng tôi chỉ hạn chế trình bày những kiến thức cơ bản của FORTRAN 90. Những phần bổ sung của các phiên bản sau so với FORTRAN 90 không nhiều và cũng chưa quá cần thiết phải đưa vào đây. Trong một số tình huống cụ thể, để giúp người đọc đã từng làm quen với FORTRAN 77 hoặc cần có thêm kiến thức để đọc những chương trình của người khác viết bằng FORTRAN 77, chúng tôi sẽ có thêm những ghi chú "mở rộng" thích hợp. Những người thành thạo Fortran muốn quan tâm đến lịch sử phát triển của ngôn ngữ lập trình này có thể tham khảo thêm cuốn ***Fortran 90 Explained***, Oxford University Press (Oxford, 1990) của Michael Metcalf và John ReidMetcalf và Reid.

Như đã nói ở trên, chính xác hơn nên viết ngôn ngữ FORTRAN, nhưng do "sở thích tuỳ tiện", ở đây chúng tôi cũng sẽ viết ***Fortran*** thay cho cách viết FORTRAN.

Quyển sách được bố cục trong 9 chương. Chương 1: Những yếu tố cơ bản của ngôn ngữ Fortran. Trong chương này trình bày cách chạy một chương trình Fortran, cấu trúc chung của một chương trình, cấu trúc câu lệnh, các kiểu dữ kiệu, biểu thức số, câu lệnh gán, các lệnh vào ra đơn giản và cách sử dụng hàm trong Fortran. Chương 2: Các câu lệnh cơ bản của Fortran. Ở đây trình bày các câu lệnh chu trình (DO Loops), lệnh rẽ nhánh với IF và SELECT CASE, cách sử dụng kết hợp DO và IF và một số thao tác với hằng và biến ký tự (CHARACTER). Chương 3: Các cấu trúc mở rộng. Chương này trình bày những kiến thức liên quan đến chu trình DO tổng quát và chu trình DO lồng nhau, cấu trúc IF tổng quát và cấu trúc IF lồng nhau, chu trình ngầm, định dạng dữ liệu bằng

lệnh FORMAT và chu trình lặp không xác định. Chương 4: Chương trình con và modul. Chương này đề cập đến những khái niệm về thư viện các hàm chuẩn của Fortran, các chương trình con trong, chương trình con ngoài và modul, và một số kiến thức khác. Chương 5 trình bày những kiến thức về mảng trong Fortran, như cách khai báo mảng, lưu trữ mảng trong bộ nhớ và truy cập đến các phần tử mảng. Chương 6 trình bày về biến ký tự và xử lý biến ký tự. Chương 7 cung cấp những kiến thức về file, như phân loại file, tổ chức dữ liệu trong file, các lệnh vào ra dữ liệu với file. Chương 8: Một số kiến thức mở rộng. Ở đây trình bày cách khai báo dùng chung bộ nhớ và ứng dụng, chương trình con BLOCK DATA, cấu trúc dữ liệu do người dùng định nghĩa và một số câu lệnh thường gặp khác. Chương 9 dẫn ra một số bài toán thông dụng, như lớp các bài toán thống kê, các bài toán về ma trận, tương quan và hồi qui tuyến tính, phương pháp số. Cuối mỗi chương là hệ thống các bài tập tự giải, nhằm củng cố những kiến thức có liên quan.

Phần cuối của quyển sách là một số phụ lục, giúp bạn đọc có thể tra cứu nhanh ý nghĩa hệ thống các câu lệnh cũng như các hàm và thủ tục của Fortran trong quá trình lập trình.

CHƯƠNG 1. NHỮNG YẾU TỐ CƠ BẢN CỦA NGÔN NGỮ FORTRAN

1.1 CHẠY MỘT CHƯƠNG TRÌNH FORTRAN

Cũng như khi bắt đầu học một ngôn ngữ lập trình nào khác, nếu là người mới làm quen với Fortran, ta nên chạy các chương trình ví dụ trong phần này càng sớm càng tốt, không cần cố gắng hiểu một cách chi tiết chúng làm việc như thế nào. Việc giải thích chúng sẽ được giới thiệu dần dần ở các phần sau. Để chạy được các chương trình này trước hết ta cần phải có một bộ phần mềm biên dịch và đã được cài đặt trên hệ thống máy tính. Ngoài ra, ta cũng cần phải làm quen với bộ phần mềm này, phải biết cách soạn thảo các chương trình Fortran và biên dịch rồi chạy nó như thế nào. Việc làm quen này không mất nhiều thời gian và cũng khá đơn giản, nên không được trình bày ở đây. Hơn nữa, vì Fortran có thể làm việc trên nhiều hệ điều hành khác nhau, như các dòng UNIX, LINUX, WINDOWS, DOS,... và nó cũng có nhiều phiên bản khác nhau đối với từng hệ điều hành, nên sẽ không đầy đủ nếu chỉ trình bày ở đây một hoặc một vài trường hợp.

Chương trình sau đây sẽ đưa ra lời chào mừng, nếu ta đưa tên của mình vào khi được hỏi:

Ví dụ 1.1 Chương trình làm quen

```
! Vi du mo dau
! Loi Chao mung!
CHARACTER NAME*20
PRINT*, 'Ten ban la gi?'
READ*, NAME
PRINT*, 'Xin chao ban ', NAME
END
```

Kết quả nhận được trên màn hình khi chạy chương trình này như sau (câu trả lời là dòng chữ in nghiêng):

Ten ban la gi?

Nam

Xin chao ban Nam

Tuy nhiên, với chương trình trên, nếu ta gõ tên mình đầy đủ cả *Họ* và *tên*, và giữa các từ có dấu cách thì kết quả có thể hơi bất ngờ đấy. Nhưng không sao, chúng ta sẽ tìm hiểu vấn đề này sau.

Lưu ý rằng, trong đoạn chương trình trên các từ tiếng Việt được viết dưới dạng không dấu, vì không phải khi nào ta cũng có thể gõ tiếng Việt có dấu, và không phải khi nào kết quả hiển thị trên màn hình máy tính cũng bằng tiếng Việt có dấu. Bởi vậy, trong đa số trường hợp, những câu, từ tiếng Việt xuất hiện trong các chương trình ví dụ sẽ được dùng tiếng Việt không dấu. Có thể điều này sẽ gây khó chịu khi so sánh Fortran với một

số ngôn ngữ khác. Nhưng ta sẽ cảm thấy tự hài lòng với khiếm khuyết nhỏ này so với khả năng tuyệt vời của Fortran.

Chương trình sau đây cho phép tính giá trị của hàm $A(t) = 174.6(t-1981.2)^3$ khi nhập vào giá trị của biến *t*

Ví dụ 1.2: Tính giá trị của hàm

```
!
PROGRAM TinhHam
! Tinh gia tri ham A(t)=174.6*(t-1981.2)**3
INTEGER T          ! Biến nguyên lưu giá trị biến t
REAL  A            ! Biến thực lưu giá trị hàm A(t)
PRINT*,'Cho gia tri cua bien t:'
READ*, T
A = 174.6 * (T - 1981.2) ** 3
PRINT*,'Gia tri ham A(t) khi t= ', T, ' la : ', A
END PROGRAM TinhHam
```

Khi chạy chương trình này, trên màn hình sẽ xuất hiện dòng chữ (phía dưới dòng này là con trỏ màn hình (□) nhấp nháy):

```
Cho gia tri cua bien t:
□
```

Nếu đưa vào giá trị *2000* (cho biến *t*) ta sẽ nhận được kết quả:

```
Gia tri ham A(t) khi t = 2000 la :  1.1601688E+06
```

Giá trị kết quả của hàm được in ra dưới dạng *ký hiệu khoa học* **E+06**, có nghĩa là số trước đó nhân với 10 luỹ thừa 6, tức là trị số của *A(t)* vào khoảng 1,16 triệu. Bây giờ ta hãy chạy chương trình này nhiều lần, mỗi lần thay đổi giá trị của biến *t* và thử tìm xem khi nào thì giá trị của hàm *A(t)* sẽ đạt khoảng 10 triệu. Sau đó, hãy thử gõ nhầm giá trị của *t* (ví dụ gõ vào *2,000* thay vì gõ 2000) để xem Fortran phản ứng lại như thế nào.

Một ví dụ khác, giả sử ta có 1000 đôla gửi tiết kiệm trong ngân hàng với lãi suất 9% mỗi năm. Vậy, sau một năm số tiền sẽ có trong ngân hàng bằng bao nhiêu?

Để lập chương trình cho máy tính giải bài toán này trước hết cần phải làm rõ vấn đề về mặt nguyên tắc. Nhận thấy rằng, số tiền sẽ có sau một năm sẽ là tổng của số tiền gốc đã gửi và số tiền lãi sẽ có. Như vậy, lôgic các bước thực hiện bài toán sẽ là:

1) Nhập số liệu vào máy (số tiền gốc và lãi suất)

2) Tính tiền lãi (tức 9% của 1000, bằng 90)

3) Cộng tiền lãi vào số tiền gốc (90 + 1000, tức 1090)

4) In (hiển thị) số tiền sẽ có sau một năm.

Với lôgic đó, ta có thể viết chương trình như sau:

Ví dụ 1.3: Tính tiền gửi tiết kiệm

```
! Chuong trinh nay khong nhap du lieu tu ban phim
```

```
PROGRAM TinhTien
! Tinh tien gui tiet kiem
REAL SoTien, TienLai, LaiSuat
SoTien = 1000.0       ! Số tiền gốc ban đầu
LaiSuat = 0.09        ! Lãi suất
TienLai = LaiSuat * SoTien
SoTien = SoTien + TienLai
PRINT*, 'So tien se co sau mot nam:', SoTien
END PROGRAM TinhTien
```

Ta gõ chương trình này vào máy rồi chạy tính, và chú ý rằng ở đây máy không đòi hỏi phải nhập đầu vào (input) từ bàn phím như ví dụ trước đây (Tại sao?). Kết quả nhận được trên màn hình sẽ là:

```
So tien se co sau mot nam: 1.0900000E+03
```

Sẽ rất có ích nếu ta cố gắng thực hiện lặp lại nhiều lần các ví dụ trên đây, mỗi lần như vậy thử sửa đổi một ít trong chương trình và theo dõi xem kết quả thay đổi như thế nào. Điều đó sẽ sẽ giúp cho ta tự tin hơn khi tiếp cận với những nội dung sau này của Fortran.

Bây giờ ta tìm hiểu xem trong quá trình thực hiện, các chương trình Fortran sẽ làm những gì. Nói chung, sau khi gõ lời chương trình (*source code*) và tiến hành chạy (*run*) nó trong môi trường của hệ điều hành máy tính thích hợp (đã cài đặt phần mềm Fortran), sẽ có hai quá trình tách biệt xảy ra. Đầu tiên, chương trình được biên dịch (*compile*), tức là mỗi câu lệnh được dịch (*translated*) sang mã máy (*machine code)* sao cho máy tính có thể hiểu được. Quá trình này xảy ra như sau. Trước hết các câu lệnh của chương trình sẽ được kiểm tra về cú pháp (*Syntax*). Nếu không có lỗi, chúng sẽ được dịch sang mã máy và lưu trữ vào một file gọi là *đối tượng* (*Object*) hay *đích*. Sau đó chúng sẽ được *liên kết* (*Link*) với hệ thống thư viện chuẩn của Fortran để tạo thành file *có thể thực hiện* (*executable*) được. Nếu chương trình còn lỗi, các lỗi sẽ được chỉ ra và quá trình biên dịch kết thúc mà không tạo được file *đích*, và do đó không xảy ra quá trình thứ hai. Nếu quá trình thứ nhất thực hiện thành công thì chuyển sang quá trình thứ hai, trong đó chương trình đã dịch (tức file có thể thực hiện được) sẽ được thực hiện (*executed*). Ở bước này mỗi một chỉ thị đã dịch của chương trình sẽ lần lượt được thực hiện theo qui tắc đã lập.

Bộ chương trình thực hiện trọn vẹn quá trình thứ nhất (tức là cho đến khi tạo được file *có thể thực hiện – executable*) thường gọi là trình biên dịch (*compiler)*.

Trong khi biên dịch, không gian bộ nhớ RAM của máy tính định vị cho mọi dữ liệu sẽ được phát sinh bởi chương trình. Phần bộ nhớ này có thể hiểu như là những "vùng" bộ nhớ khu trú mà mỗi một trong chúng, tại một thời điểm, chỉ có thể xác định một giá trị dữ liệu. Các bộ nhớ khu trú này được tham chiếu đến bởi các tên ký hiệu (định danh) trong chương trình. Bởi vậy, câu lệnh:

```
SoTien = 1000.0
```

là cấp phát số 1000.0 đến vị trí bộ nhớ có tên **SoTien**. Vì nội dung của **SoTien** có thể thay đổi trong khi chương trình chạy nên nó được gọi là *biến* (*variable)*.

Về hình thức, chương trình tính tiền gửi tiết kiệm (ví dụ 1.3) trên đây được biên dịch như sau:

1) Đưa số 1000 vào vị trí bộ nhớ **SoTien**

2) Đưa số 0.09 vào vị trí bộ nhớ **LaiSuat**

3) Nhân nội dung của **LaiSuat** với nội dung của **SoTien** và đưa kết quả vào vị trí bộ nhớ **TienLai**

4) Cộng nội dung của **SoTien** với nội dung của **TienLai** và đưa kết quả vào **SoTien**

5) In (hiển thị) thông báo nội dung của **SoTien**

6) Kết thúc.

Khi chạy chương trình, các câu lệnh dịch này được thực hiện theo thứ tự từ trên xuống dưới. Quá trình thực hiện, các vị trí bộ nhớ được sử dụng sẽ có những giá trị sau:

```
SoTien : 1000
LaiSuat: 0.09
TienLai: 90
SoTien : 1090
```

Chú ý rằng nội dung ban đầu của **SoTien** đã bị thay thế bởi giá trị mới.

Câu lệnh **PROGRAM** ở dòng thứ hai trong ví dụ 1.3 mở đầu cho chương trình. Nó là *câu lệnh tuỳ chọn*, và có thể kèm theo tên tuỳ ý. Dòng thứ nhất và dòng thứ ba, bắt đầu với dấu chấm than, là *lời giải thích*, có lợi cho người đọc chương trình, và không ảnh hưởng gì tới chương trình dịch. Các biến trong chương trình có thể có các kiểu (*type)* khác nhau; câu lệnh **REAL** trong ví dụ này là khai báo kiểu. Các dòng trống (nếu có) trong chương trình được xem như những câu lệnh không thực hiện (*non-executable)*, tức là không có tác động nào được thực hiện, có thể chèn thêm vào để cho chương trình được sáng sủa, không rối mắt.

Bây giờ ta hãy thử làm lại ví dụ này như sau.

1) Chạy chương trình và *ghi nhớ lại* kết quả

2) Thay đổi câu lệnh thực hiện **SoTien = 1000.0** bởi câu lệnh **SoTien = 2000.0** và chạy lại chương trình. Rõ ràng có thể hiểu được tại sao kết quả mới lại khác với kết quả trước đó.

3) Tiếp đến, loại bỏ dòng lệnh

```
SoTien = SoTien + TienLai
```

và chạy lại chương trình. Kết quả nhận được là số tiền không thay đổi! Như vậy, do loại bỏ dòng lệnh

```
SoTien = SoTien + TienLai
```

nên số tiền lãi sẽ không được cộng vào, tức nội dung bộ nhớ của biến **SoTien** không được cập nhật.

Tóm lại, để giải một bài toán bằng lập trình với ngôn ngữ Fortran ta cần thực hiện theo trình tự các bước sau:

1) Phân tích bài toán, xác định thuật giải, các bước thực hiện và trình tự thực hiện các bước. Đây là bước hết sức quan trọng, vì nó quyết định sự đúng đắn về mặt lôgic của việc giải bài toán. Do đó, nói chung ta nên lập một dàn bài cụ thể và biểu diễn nó qua các sơ đồ (thường gọi là sơ đồ khối)

2) Soạn thảo mã nguồn của chương trình (chương trình nguồn, hay lời chương trình), tức là ngôn ngữ hoá các thuật giải, theo đúng trình tự đã lập và lưu vào một (hoặc một số) file với phần mở rộng là *.f90 (hoặc *.f, *.for, ngầm định đối với Fortran 77).

3) Tiến hành biên dịch chương trình. Ở bước này nếu chương trình vẫn còn lỗi cú pháp ta sẽ quay lại bước 2) để chỉnh sửa rồi tiếp tục biên dịch lại chương trình. Quá trình cứ tiếp diễn cho đến khi trình biên dịch tạo được file *đích* (Ojective file) và thực hiện liên kết (link) để nhận được file *thực hiện* (executable file).

4) Chạy chương trình (tức chạy file thực hiện) để nhận được kết quả. Sau khi nhận được kết quả tính ta cần phân tích, xem xét tính hợp lý, đúng đắn của nó. Nếu kết quả không phù hợp cần phải xem xét lại bước 1) và bước 2).

1.2 CẤU TRÚC CHUNG CỦA MỘT CHƯƠNG TRÌNH FORTRAN

Cấu trúc chung của một chương trình Fortran đơn giản như sau (những phần đặt trong dấu ngoặc vuông là tuỳ chọn, có thể có, cũng có thể không):

```
[PROGRAM TenChuongTrinh]
     [Cac_cau_lenh_khai_bao]
     [Cac_cau_lenh_thuc_hien]
END [PROGRAM [TenChuongTrinh]]
```

Như đã thấy, chỉ có một câu lệnh bắt buộc trong chương trình Fortran là **END**. Câu lệnh này báo cho chương trình dịch rằng không còn câu lệnh nào hơn nữa để dịch.

Ký hiệu

```
END [PROGRAM [TenChuongTrinh]]
```

có nghĩa rằng có thể bỏ qua ***TenChuongTrinh*** trong câu lệnh `END`, nhưng nếu có ***TenChuongTrinh*** thì từ khoá `PROGRAM` là bắt buộc.

TenChuongTrinh là tên của chương trình, thường được đặt một cách tùy ý sao cho mang tính gợi nhớ, rằng chương trình sẽ giải quyết vấn đề gì. ***Cac_cau_lenh_khai_bao*** là những câu lệnh khai báo biến, hằng,... và kiểu dữ liệu tương ứng của chúng để trình

biên dịch cấp phát bộ nhớ, phân luồng xử lý. ***Cac_cau_lenh_thuc_hien*** là những câu lệnh xác định qui tắc và trình tự thực hiện tính toán, xử lý để đạt được kết quả.

Trong cấu trúc trên, các mục (nếu có) bắt buộc phải xuất hiện theo trình tự như đã mô tả. Có nghĩa là sau câu lệnh mô tả tên chương trình sẽ là các câu lệnh khai báo, tiếp theo là các câu lệnh thực hiện. Câu lệnh **END** phải đặt ở cuối chương trình.

1.3 CẤU TRÚC CÂU LỆNH

Dạng câu lệnh cơ bản của mọi chương trình Fortran 90 có thể gồm từ 0 đến 132 ký tự (câu lệnh có thể là trống rỗng; câu lệnh trống rỗng làm cho chương trình dễ đọc hơn bởi sự phân cách lôgic giữa các đoạn). Đối với phiên bản Fortran 77 và các phiên bản trước đó, nội dung các câu lệnh phải bắt đầu từ cột thứ 7 và kéo dài tối đa đến cột thứ 72. Nếu câu lệnh có nội dung dài hơn, nó sẽ được ngắt xuống dòng dưới, và ở dòng nối tiếp này phải có một ký tự bất kỳ (khác dấu cách) xuất hiện ở cột thứ 6. Bạn đọc cần lưu ý đặc điểm này khi sử dụng các chương trình của người khác, hoặc của chính mình, lập trình với các phiên bản Fortran 77 và trước đó. Fortran 90 không có sự hạn chế đó.

Một câu lệnh cũng có thể có nhãn. Nhãn là một số nguyên dương trong khoảng 1–99999. Nhãn (nếu có) phải là duy nhất trong một chương trình và phải đặt ở đầu câu lệnh, phân cách với nội dung câu lệnh bởi ít nhất một dấu cách. Đối với Fortran 77 và các phiên bản trước, nhãn được ghi vào các cột 1–5.

Tất cả các câu lệnh, trừ *câu lệnh gán* (ví dụ **Sotien = 1000.0**), đều bắt đầu bằng các từ khoá (*keyword)*. Trên đây chúng ta đã gặp một số từ khoá như **END, PRINT, PROGRAM,** và **REAL**.

Nói chung trên mỗi dòng có một câu lệnh. Tuy nhiên, nhiều câu lệnh cũng có thể xuất hiện trên một dòng, nhưng chúng phải được phân cách nhau bởi các dấu chấm phẩy (;). Để cho rõ ràng, chỉ nên viết những câu lệnh gán rất ngắn, như:

```
A = 1; B = 1; C = 1
```

Những câu lệnh dài có thể được viết trên nhiều dòng và phải có ký hiệu nối dòng (sẽ được trình bày dưới đây).

1.3.1 Ý nghĩa của dấu cách (Blank)

Nói chung các dấu cách là không quan trọng, ta có thể sử dụng chúng để làm cho chương trình dễ đọc hơn bằng cách viết thụt câu lệnh vào (thêm dấu cách vào phía bên trái) hoặc chèn vào giữa các câu lệnh. Tuy nhiên, cũng có những chỗ không được phép chèn dấu cách vào, như các qui ước về cách viết từ khóa, tên biến,... mà ta gọi là các *ký hiệu qui ước*.

Ký hiệu qui ước trong Fortran 90 là một chuỗi liên tiếp các ký tự có ý nghĩa, chẳng hạn các *nhãn*, các *từ khóa*, *tên*, *hằng*,... Như vậy, các cách viết **INTE GER**, **So Tien** và **<**

= là không được phép (**<=** là một phép toán), vì giữa chúng có dấu cách không hợp lệ, trong khi **A * B** thì được phép và giống như **A*B**.

Tuy nhiên, tên, hằng hoặc nhãn cần phải được phân cách với các từ khoá, tên, hằng hoặc nhãn khác ít nhất một dấu cách. Như vậy **REALX** và **30CONTINUE** là không được phép (vì **X** là biến, còn **30** là nhãn).

1.3.2 Lời chú thích

Mọi ký tự theo sau dấu chấm than (!) (ngoại trừ trong xâu ký tự) là lời chú thích, và được chương trình dịch bỏ qua. Toàn bộ nội dung trên cùng một dòng có thể là lời chú thích. Dòng trắng cũng được dịch như dòng chú thích. Lời chú thích có thể được dùng một cách tuỳ ý để làm cho chương trình dễ đọc.

Đối với Fortran 77, nếu *cột đầu tiên* có ký tự "C" hoặc "c" thì nội dung chứa trên dòng đó sẽ được hiểu là lời chú thích. Qui tắc này không được Fortran 90 chấp nhận. Nhưng thay cho các ký tự "C" hoặc "c", nếu sử dụng ký tự dấu chấm than thì chúng lại tương đương nhau.

1.3.3 Dòng nối tiếp

Nếu câu lệnh quá dài nó có thể được chuyển một phần xuống dòng tiếp theo bằng cách thêm ký hiệu nối dòng (**&**) vào cuối cùng của dòng trước khi ngắt phần còn lại xuống dòng dưới. Ví dụ:

```
A = 174.6 *     &
       (T - 1981.2) ** 3
```

Như đã nói ở trên, Fortran 77 sử dụng cột thứ 6 làm cột nối dòng, do đó cách chuyển tiếp dòng của Fortran 90 sẽ không tương thích với Fortran 77.

Dấu **&** tại cuối của *dòng chú thích* sẽ không được hiểu là sự nối tiếp của dòng chú thích, vì khi đó **&** được xem như là một phần của chú thích.

1.4 KIỂU DỮ KIỆU

Như đã thấy trên đây, các chương trình Fortran thường được bắt đầu bằng các câu lệnh khai báo biến, hằng và kiểu dữ liệu của chúng. Khái niệm kiểu dữ liệu (*data type)* là khái niệm cơ bản trong Fortran 90. Kiểu dữ liệu bao gồm tập hợp các giá trị dữ liệu (chẳng hạn, toàn bộ các số), cách thức biểu thị chúng (ví dụ, –2, 0, 999), và tập hợp các phép toán (ví dụ, phép toán số học) cho phép xuất hiện trong chúng.

Fortran 90 định nghĩa 5 kiểu dữ liệu chuẩn, được chia thành hai lớp là lớp các kiểu số (*numeric)* gồm số nguyên (**integer**), số thực (**real**) và số phức (**complex**), và lớp các kiểu không phải số (*non-numeric)* gồm kiểu ký tự (**character**) và kiểu lôgic (**logical**).

Liên kết với mỗi kiểu dữ liệu là các loại (*kind*) dữ liệu. Về cơ bản điều đó liên quan đến khả năng lưu trữ và biểu diễn giá trị dữ liệu. Chẳng hạn, có thể có hai loại số nguyên (**integer**): số nguyên ngắn và số nguyên dài. Chúng ta sẽ đề cập đến vấn đề này sâu hơn ở các phần sau.

Ngoài các kiểu dữ liệu chuẩn trên đây, ta có thể định nghĩa cho riêng mình *các kiểu dữ liệu khác*, chúng có thể có các tập giá trị và các phép toán riêng.

Gắn liền với các kiểu dữ liệu còn có các *thuộc tính* dữ liệu. Fortran định nghĩa khá nhiều *thuộc tính*, sau đây là một số thuộc tính thông dụng:

- **PARAMETER:** thuộc tính *hằng*,
- **DIMENSION:** thuộc tính *mảng*,
- **ALLOCATABLE:** thuộc tính *cấp phát động*,
- **POINTER:** thuộc tính *con trỏ*,
- ...

Thuộc tính có thể được dùng đi kèm với câu lệnh khai báo kiểu dữ liệu để mô tả kiểu dữ liệu của biến, hằng. Trong nhiều trường hợp thuộc tính cũng có thể được dùng độc lập như những câu lệnh khai báo.

1.4.1 Lớp các kiểu số (Integer, Real, Complex)

a. Kiểu số nguyên

Dữ liệu có kiểu số nguyên là những dữ liệu nhận các giá trị thuộc tập số nguyên, ví dụ 0, 1, 2, 3,..., –5, –10,... Đó là tập hợp các số có thể "đếm được" hay tập có thứ tự, tức là một số nguyên bất kỳ luôn có một số liền trước và một số liền sau. Để khai báo biến hoặc hằng có kiểu số nguyên ta sử dụng câu lệnh:

```
INTEGER [([KIND=]kind)][,attrs] ::] vname
```

Trong đó:

kind là loại, nhận một trong các giá trị 1, 2, 4 hoặc 8 (đối với UNIX hoặc LINUX).

attrs là thuộc tính, nhận một, hoặc nhiều hơn, trong các giá trị **PARAMETER, DIMENSION, ALLOCATABLE, POINTER**,...

vname là danh sách biến hoặc hằng, được viết cách nhau bởi các dấu phẩy.

Tùy theo *loại* mà một biến/hằng nguyên sẽ chiếm dung lượng bộ nhớ và phạm vi giá trị là lớn hay nhỏ. Trong bảng 1.1 dẫn ra miền giá trị hợp lệ đối với các *loại* số nguyên được khai báo, trong đó cột 1 biểu thị những cách có thể khai báo, cột 2 là dung lượng bộ nhớ bị chiếm giữ ứng với các *loại* số nguyên, và cột 3 là phạm vi giá trị của các loại số nguyên tương ứng đã khai báo.

Bảng 1.1 Miền giá trị và dung lượng bộ nhớ của kiểu số nguyên

Cách khai báo	Số byte chiếm giữ	Phạm vi giá trị
INTEGER	4	–2 147 483 648 đến

		2 147 483 647
INTEGER*1 hoặc INTEGER (1) hoặc INTEGER (KIND=1)	1	–128 đến 127
INTEGER*2 hoặc INTEGER (2) hoặc INTEGER (KIND=2)	2	–32 768 đến 32 767
INTEGER*4 hoặc INTEGER (4) hoặc INTEGER (KIND=4)	4	–2 147 483 648 đến 2 147 483 647

Các ví dụ sau đây cho thấy có thể sử dụng các cách khác nhau để khai báo kiểu số nguyên cho các biến, hằng.

```
INTEGER, DIMENSION(:), POINTER :: days, hours
INTEGER(2), POINTER :: k, limit
INTEGER(1), DIMENSION(10) :: min
```

Tất cả các biến được khai báo trên đây đều có kiểu số nguyên. Dòng thứ nhất khai báo các biến **days, hours** là những biến mảng một chiều có thuộc tính con trỏ, với kích thước chưa xác định, mỗi phần tử mảng là một số nguyên 4 byte; dòng thứ hai khai báo hai biến đơn (biến vô hướng) **k, limit** có thuộc tính con trỏ kiểu số nguyên loại 2 byte; dòng thứ ba khai báo một biến mảng **min** gồm 10 phần tử, mỗi phần tử là một số nguyên loại 1 byte. Những khai báo trên tương đương với cách khai báo dưới đây:

```
INTEGER days, hours
INTEGER(2) k, limit
INTEGER(1) min
DIMENSION days(:), hours(:), min (10)
POINTER days, hours, k, limit
```

Các biến trên cũng có thể được khởi tạo giá trị ban đầu thông qua các lệnh khai báo, chẳng hạn:

```
INTEGER (2) :: k=4
INTEGER (2), PARAMETER :: limit=12
```

Trong khai báo trên, biến **limit** có thuộc tính là **PARAMETER** nên giá trị của nó sẽ *không bị biến đổi* trong quá trình thực hiện chương trình. Bởi vậy nó được gọi là *hằng*, khác với **k** là *biến*. Cũng có thể khai báo biến và hằng dưới dạng sau đây:

```
INTEGER days, hours
INTEGER (2):: k=4, limit
DIMENSION days(:), hours(:)
POINTER days, hours
PARAMETER (limit=12)
```

Với cách khai báo này, các từ khóa **DIMENSION, POINTER, PARAMETER** (ở ba dòng cuối) được gọi là các lệnh khai báo, dùng để định nghĩa biến, hằng và thuộc tính của chúng.

b. Kiểu số thực

Kiểu số thực nói chung gần giống với tập số thực trong toán học. Khác với kiểu số nguyên, kiểu số thực là tập hợp "không đếm được", hay tập không có thứ tự. Để biểu diễn số thực Fortran 90 sử dụng hai phương pháp gần đúng là độ chính xác đơn và độ chính xác kép. Có thể khai báo kiểu số thực bằng câu lệnh:

```
REAL [([KIND=]kind)][[,attrs] ::] vname
```

Đối với số thực độ chính xác kép (hay độ chính xác gấp đôi) ta còn có thể sử dụng câu lệnh khai báo:

```
DOUBLE PRECISION [[,attrs] ::] vname
```

Trong đó:

kind là loại, nhận giá trị 4, 8 hoặc 16 (đối với UNIX hoặc LINUX).

attrs là thuộc tính, nhận một, hoặc nhiều hơn, trong các giá trị **PARAMETER, DIMENSION, ALLOCATABLE, POINTER,**...

vname là danh sách biến hoặc hằng, viết cách nhau bởi các dấu phẩy.

Cách khai báo, phạm vi giá trị, độ chính xác và dung lượng bộ nhớ bị chiếm giữ ứng với từng *loại* số thực được cho trong bảng 1.2, trong đó các cột 1, 2, 4 được mô tả tương tự như các cột 1, 2, 3 trong bảng 1.1. Riêng cột thứ 3 ở đây, do số thực chỉ được biểu diễn gần đúng nên giá trị của chúng chỉ đạt được độ chính xác nhất định tùy theo dung lượng ô nhớ dùng để mô tả chúng. Độ chính xác trong trường hợp này được hiểu là *số chữ số* có thể *biểu diễn chính xác* giá trị của biến/hằng thực. Ví dụ, nếu chạy chương trình sau đây

```
REAL X
X = 123456789.0
PRINT '(F30.2)', X
end
```

ta sẽ nhận được kết quả trên màn hình là:

```
X=        123456800.00
```

Có lẽ bạn đọc sẽ ngạc nhiên, vì biến **x** chỉ được gán giá trị rồi in ra mà giá trị in ra lại *khác* với giá trị gán vào? Nguyên nhân của sự khác nhau này là ở chỗ, ta đã khai báo biến **x** là loại số thực 4 byte, do đó chỉ có 6 chữ số đầu tiên biểu diễn chính xác giá trị của biến **x**.

Bảng 1.2 Miền giá trị và dung lượng bộ nhớ của kiểu số thực

Cách khai báo	Số byte chiếm giữ	Độ chính xác (số chữ số)	Phạm vi giá trị
REAL REAL*4 REAL (KIND=4)	4	6	–3.4028235E+38 đến –1.1754944E–38; 0; +1.1754944E–38 đến +3.4028235E+38
REAL*8 REAL (KIND=8) DOUBLE PRECISION	8	15	–1.797693134862316D+308 đến –2.225073858507201D–308; 0; +2.225073858507201D–308 đến +1.797693134862316D+308

Sau đây là một số ví dụ khai báo các biến, hằng có kiểu số thực.

```
! Khai bao cac bien co kieu du lieu so thuc
REAL X, Y(10)
REAL*4 A,B
REAL (KIND=8), DIMENSION (5) :: U,V
DOUBLE PRECISION, DIMENSION (:), ALLOCATABLE :: T
```

```
REAL, PARAMETER :: R_TDat = 6370.0
```

Dòng thứ nhất khai báo một biến đơn **X** và một biến mảng **Y** gồm 10 phần tử, chúng đều là những số thực *loại* 4 byte; dòng thứ hai khai báo hai biến đơn **A** và **B** là những biến thực *loại* 4 byte; dòng thứ ba khai báo hai biến mảng **U, V**, mỗi biến gồm 5 phần tử là những số thực *loại* 8 byte; dòng thứ tư khai báo biến mảng thuộc tính động **T** có độ chính xác gấp đôi, tức mỗi phần tử mảng chiếm 8 byte; dòng cuối cùng khai báo hằng đơn **R_TDat**, có giá trị khởi tạo bằng **6370.0**.

c. Kiểu số phức

Số phức được định nghĩa như một cặp *có thứ tự* của hai số thực được gọi là phần thực và phần ảo. Dữ liệu kiểu số phức được khai báo bằng câu lệnh:

```
COMPLEX [([KIND =]kind)] [[,attrs] :: ] vname
```

Trong đó tham số ***kind*** nhận giá trị 4 hoặc 8; tham số ***attrs*** là một hoặc nhiều thuộc tính, nhận các giá trị **PARAMETER, DIMENSION, ALLOCATABLE, POINTER**,...; ***vname*** là danh sách biến hoặc hằng, viết cách nhau bởi các dấu phẩy.

Độ chính xác và phạm vi giá trị của kiểu số phức là độ chính xác và phạm vi giá trị của các phần thực và phần ảo. Dung lượng bộ nhớ chiếm giữ của một số phức là dung lượng của hai số thực. Bảng 1.3 liệt kê các cách khai báo và số byte chiếm giữ của các biến, hằng có kiểu số phức.

Ví dụ, câu lệnh:

```
COMPLEX (4), DIMENSION (8) :: cz, cq
```

khai báo hai biến phức **cz** và **cq**, mỗi biến là một mảng gồm 8 phần tử phức, tức là 8 cặp số thực, mỗi số thực chiếm 4 byte. Câu lệnh này tương đương với hai câu lệnh sau:

```
COMPLEX(4) cz, cq
DIMENSION(8) cz, cq
```

Bảng 1.3 Miền giá trị và dung lượng bộ nhớ của kiểu số phức

Cách khai báo	Số byte chiếm giữ
COMPLEX COMPLEX *4 COMPLEX (4) COMPLEX (KIND=4)	8
COMPLEX *8 COMPLEX (8) COMPLEX (KIND=8) DOUBLE CPMPLEX	16

1.4.2 Kiểu ký tự (Character) và kiểu lôgic (Logical)

a. Kiểu ký tự

Kiểu ký tự có tập giá trị là các ký tự lập thành xâu (chuỗi) ký tự. Độ dài của xâu là số ký tự trong xâu đã được khai báo. Mỗi ký tự trong xâu ký tự chiếm 1 byte bộ nhớ. Do đó, số byte chiếm giữ bộ nhớ của biến, hằng kiểu ký tự tùy thuộc độ dài của xâu. Câu lệnh tổng quát khai báo biến, hằng kiểu ký tự có thể là một trong các cách sau.

Cách 1: