NGÔN NGỮ LẬP TRÌNH FORTRAN 90

Phan Văn Tân

NXB Đại học Quốc gia Hà Nội 2007

Từ khoá: File, hàm, lệnh , chương trình con, thủ tục, ký tự, xâu, mảng, biến, tương quan, cấu trúc, phổ, thuật toán

---

PHAN VĂN TÂN

NGÔN NGỮ LẬP TRÌNH FORTRAN 90

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

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**	**–2 147 483 648 đến**

(4) hoặc INTEGER (KIND=4)		**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:

CHARACTER (*length*) *vname*

Trong đó ***length*** là một số nguyên dương chỉ độ dài cực đại của ***vname*; *vname*** là danh sách tên biến, hằng có kiểu xâu ký tự, viết cách nhau bởi dấu phẩy.

Cách 2:

CHARACTER (*type*[,*type*…])[*attrib*[,*attrib*]…] :: *vname*

Với ***type*** là tham số độ dài và loại, nhận một trong các dạng:

(LEN = type-value)
(KIND = expr)
(KIND = expr, LEN = type-value)
([LEN =] type-value, KIND = expr)

trong đó ***type–value*** có thể là dấu sao (*), hằng nguyên không dấu, hoặc biểu thức nguyên; ***expr*** là biểu thức xác định giá trị hằng nguyên tương ứng với phương pháp biểu diễn ký tự (chẳng hạn, chữ cái Latinh, chữ cái Hylạp,…).

attrib là một hoặc nhiều thuộc tính, viết cách nhau bởi dấu phẩy. Nếu chỉ ra thuộc tính thì sau đó phải sử dụng dấu (::). Các thuộc tính có thể là: **ALLOCATABLE, DIMENSION, PARAMETER, POINTER,...**

Cách 3:

CHARACTER [*chrs] *vname* [*lengths][(*dim*)] &
[/*values*/][,*vname* [*lengths][(*dim*)]] [/*values*/]

Trong đó: ***chrs*** là độ dài (cực đại) của các xâu, có thể là một số nguyên không dấu, biểu thức nguyên nằm trong ngoặc đơn, hoặc dấu sao nằm trong ngoặc đơn (*); ***lengths*** là độ dài (cực đại) của xâu, có thể là số nguyên không dấu, biểu thức nguyên nằm trong ngoặc đơn, hoặc dấu sao nằm trong ngoặc đơn (*); ***dim***: khai báo mảng, tức ***vname*** như là mảng; ***/values/*** là liệt kê các hằng ký tự, tức giá trị của các biến, hằng ***vname***.

Ví dụ:

```
CHARACTER (20) St1, St2*30
CHARACTER  wt*10, city*80, ch
CHARACTER (LEN = 10), PRIVATE :: vs
CHARACTER*(*) arg
CHARACTER  name(10)*20
CHARACTER(len=20), dimension(10):: plume
CHARACTER(2) susan,patty,alice*12,dotty, jane(79)
CHARACTER*5 word /'start'/
```

Các khai báo trên đây có ý nghĩa như sau: biến **St1** có độ dài cực đại bằng 20 ký tự, biến **St2** có độ dài cực đại bằng 30 ký tự; các biến **wt, city, ch** tương ứng có độ dài cực đại là 10, 80 và 1 ký tự; biến **vs** có độ dài cực đại bằng 10 ký tự và có thuộc tính **PRIVATE**; biến **arg** có độ dài không xác định; các biến mảng một chiều **name**, **plume** mỗi mảng gồm 10 phần tử, mỗi phần tử là một xâu có độ dài cực đại 20 ký tự; các biến **susan, patty, dotty** có độ dài cực đại 2 ký tự, biến **alice** có độ dài cực đại 12 ký tự và biến mảng **jane** gồm 79 phần tử, mỗi phần tử là một xâu dài 2 ký tự; biến **word** dài tối đa 5 ký tự và được khởi tạo giá trị đầu bằng **'start'**.

b. Kiểu lôgic

Dữ liệu kiểu lôgic chỉ nhận các giá trị .**TRUE.** hoặc .**FALSE**. Câu lệnh khai báo kiểu dữ liệu lôgic có dạng:

LOGICAL [([KIND=]*kind*)] [, *attrs* ::] *vname*

Trong đó:

kind: là độ dài tính bằng byte, nhận các giá trị 1, 2, hoặc 4.

attrs: là các thuộc tính, có thể nhận một hoặc nhiều giá trị, phân cách nhau bởi dấu phẩy.

vname: Danh sách các biến, hằng, phân cách nhau bởi dấu phẩy.

Số byte chiếm giữ bộ nhớ của kiểu dữ liệu logic phụ thuộc vào loại dữ liệu như mô tả trong bảng 1.4.

Bảng 1.4 Miền giá trị và dung lượng bộ nhớ của kiểu lôgic

Cách khai báo	Loại (KIND)	Số byte chiếm giữ
LOGICAL	**4**	**4**
LOGICAL*1 hoặc LOGICAL (1) hoặc LOGICAL (KIND=1)	**1**	**1**
LOGICAL*2 hoặc LOGICAL (2) hoặc LOGICAL (KIND=2)	**2**	**4**
LOGICAL*4 hoặc LOGICAL (4) hoặc LOGICAL (KIND=4)	**4**	**4**

Ví dụ, các câu lệnh sau đây khai báo các biến có kiểu lôgic dưới các dạng khác nhau:

```
LOGICAL, ALLOCATABLE :: flag1, flag2
LOGICAL (2), SAVE :: doit, dont = .FALSE.
LOGICAL  switch
```

Cách khai báo đó hoàn toàn tương đương với các câu lệnh khai báo sau đây:

```
LOGICAL flag1, flag2
LOGICAL (2) doit, dont = .FALSE.
ALLOCATABLE  flag1, flag2
SAVE doit, dont
```

1.4.3 Phép toán trên các kiểu dữ liệu

Trong các ví dụ trước đây ta đã thấy một số biểu thức viết bằng ngôn ngữ Fortran trong đó có sử dụng một số phép toán, như phép nhân hai số, phép cộng hai số,... Tuy nhiên, với các kiểu dữ liệu khác nhau, các phép toán trên chúng cũng có thể khác nhau. Sau đây sẽ trình bày chi tiết hơn về vấn đề này.

Nói chung, Fortran định nghĩa bốn lớp phép toán tương ứng với các kiểu dữ liệu đã được mô tả:

– Phép toán số học: Sử dụng với các kiểu số nguyên, số thực và số phức.

– Phép toán quan hệ, hay phép toán so sánh: Sử dụng với các kiểu số nguyên, số thực, kiểu ký tự, và cũng có thể đối với cả số phức trong trường hợp so sánh *bằng* hoặc *không bằng*.

– Phép toán lôgic: Sử dụng với kiểu lôgic, và có thể với cả số nguyên.

– Phép toán gộp xâu ký tự: Sử dụng với kiểu ký tự.

Bảng 1.5 liệt kê ký hiệu các phép toán, thứ tự ưu tiên, thứ tự thực hiện trong biểu thức và ý nghĩa của chúng, trong đó thứ tự ưu tiên được xếp sao cho mức ưu tiên cao nhất là 1.

Mặc dù vậy, trong lúc viết chương trình ta cần chú ý một số điểm sau đây khi thực hiện các phép toán đối với các kiểu dữ liệu khác nhau:

– Trong một biểu thức số học, nếu các toán hạng có cùng kiểu dữ liệu thì kiểu dữ liệu kết quả là kiểu dữ liệu của các toán hạng. Nếu các toán hạng khác kiểu dữ liệu thì kết quả nhận được sẽ có kiểu dữ liệu của toán hạng có kiểu "mạnh nhất". Chẳng hạn, biểu thức gồm hỗn hợp cả số nguyên và số thực thì kết quả sẽ có kiểu số thực, vì kiểu số thực "mạnh hơn" kiểu số nguyên. Tuy nhiên, khi gán kết quả đó cho một biến thì kiểu của kết quả sẽ được chuyển thành kiểu dữ liệu của biến. Ví dụ, nếu **a, b, x** là các biến thực, còn **n** là biến nguyên thì:

a=–22.9; b=6.1 => x=a+b=–16.8; nhưng **n=a+b=–16**
a=2.9; b=6.8 => x=a+b=9.7; nhưng **n=a+b = 9**

Như đã thấy, giá trị của **a+b** sau khi gán cho **n** thì phần thập phân sẽ bị "chặt cụt".

– Kết quả của biểu thức quan hệ và biểu thức lôgic luôn luôn nhận giá trị hoặc **.TRUE.** hoặc **.FALSE.**

Bảng 1.5 Định nghĩa các phép toán trong Fortran

Ký hiệu phép toán	Tên gọi/Ý nghĩa	Thứ tự ưu tiên	Thứ tự thực hiện	Ví dụ
Phép toán số học				
**	Phép lũy thừa	1	Phải sang trái	A ** B
*	Phép nhân	2	Trái sang phải	A * B
/	Phép chia	2	Trái sang phải	A / B
+	Phép cộng	3	Trái sang phải	A + B
–	Phép trừ	3	Trái sang phải	A – B
Phép toán quan hệ				
.EQ. (==)	Bằng	–	Không phân định	A.EQ.B; A == B
.LT. (<)	Nhỏ hơn	–	Không phân định	A.LT.B; A < B
.LE. (<=)	Nhỏ hơn hoặc bằng	–	Không phân định	A.LE.B; A <= B
.GT. (>)	Lớn hơn	–	Không phân định	A.GT.B; A > B
.GE. (>=)	Lớn hơn hoặc bằng	–	Không phân định	A.GE.B; A >= B
.NE. (/=)	Không bằng (Khác)	–	Không phân định	A.NE.B; A /= B
Phép toán lôgic				
.NOT.	Phủ định	1	Không phân định	.NOT. L1
.AND.	Và (Phép hội)	2	Trái sang phải	L1. AND. L2
.OR.	Hoặc (Phép tuyển)	3	Trái sang phải	L1. OR. L2
.XOR.	Hoặc triệt tiêu	4	Trái sang phải	L1. XOR. L2
.EQV.	Tương đương	4	Trái sang phải	L1. EQV. L2
.NEQV.	Không tương đương	4	Trái sang phải	L1. NEQV. L2
Gộp ký tự				
//	Gộp hai xâu ký tự	–	Trái sang phải	ST1 // ST2

Sau đây là môt số ví dụ ước lượng giá trị của biểu thức:

2 ** 9 ** 0.5 cho kết quả là **8**

10 + 3 * 2 ** 4 – 16 / 2 cho kết quả là **50**

3.5 > 7.2 cho kết quả là **.FALSE.**

Nếu **a** và **b** là hai biến lôgic, khi đó các phép toán giữa **a** và **b** sẽ cho kết quả:

Giá trị của a và của b	a .AND. b	a .OR. b	a .EQV. b	a .NEQV. b	a.XOR.b
a=.TRUE., b=.TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.	.FALSE.
a=.TRUE., b=.FALSE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.
a=.FALSE., b=.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.
a=.FALSE., b=.FALSE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.

Nếu **ST1** và **ST2** là hai xâu ký tự nhận các giá trị:

ST1='Hanoi'
ST2=' – Vietnam'

thì

ST1 // ST2 sẽ cho kết quả là **'Hanoi – Vietnam'**

1.5 HẰNG

Hằng là những ký hiệu qui ước được sử dụng để *biểu thị các giá trị* có kiểu riêng. Mỗi kiểu dữ liệu có một loại hằng tương ứng.

1.5.1 Hằng nguyên

Hằng nguyên được sử dụng để biểu thị các giá trị kiểu số nguyên thực sự. Biểu diễn đơn giản nhất và rõ ràng nhất là số nguyên không dấu hoặc có dấu. Trong trường hợp hằng nguyên dương, thì dấu là không bắt buộc (tuỳ ý). Ví dụ:

1000, 0, +753, –999999, 2501

là những hằng biểu diễn trong hệ cơ số thập phân (cơ số 10). Các số dương cũng có thể được biểu diễn dưới dạng nhị phân (binary – cơ số 2), bát phân (octal – cơ số 8) hoặc thập lục phân (hexa – cơ số 16), ví dụ:

trong hệ cơ số 2 (binary): **B'1011'**
trong hệ cơ số 8 (octal): **O'0767'**
trong hệ cơ số 16 (hexadecimal): **Z'12EF'**

Trong các biểu diễn trên, có thể sử dụng chữ in thường hoặc chữ in hoa. Dấu nháy kép (") có thể được sử dụng thay cho dấu nháy đơn (') như là sự phân ranh giới. Tuy nhiên, các dạng thức này không được dùng với câu lệnh **DATA**.

1.5.2 Hằng thực

Hằng thực dùng để biểu thị các giá trị có kiểu số thực và có hai dạng.

– Dạng thứ nhất được viết rất rõ ràng, gọi là *dạng dấu phẩy tĩnh*, bao gồm một dãy số có chứa dấu chấm thập phân. Nó có thể có dấu hoặc không dấu. Ví dụ:

0.09 37. 37.0 .0 -.6829135

Như vậy, khi viết một hằng thực, có thể không có số nào phía bên trái hoặc phía bên phải dấu chấm thập phân (như **37.** và **.0**), nhưng chỉ có một dấu chấm thập phân không thôi thì không được phép.

– Dạng thứ hai được gọi là dạng *dấu phẩy động*. Về cơ bản nó bao gồm hoặc một số nguyên hoặc một số thực dấu phẩy tĩnh (có thể có dấu hoặc không) và sau đó là chữ cái **E** (hoặc **e**), tiếp theo là số nguyên (có dấu hoặc không). Số nguyên đứng đằng sau E là chỉ số mũ của 10, hàm ý rằng đó là 10 luỹ thừa mà số đằng trước E phải nhân với nó. Ví dụ:

2.0E2 ($2.0 \times 10^2 = 200.0$)
2E2 ($2 \times 10^2 = 200.0$)
4.12E+2 ($4.12 \times 10^{+2} = 412.0$)
-7.321E-4 ($-7.321 \times 10^{-4} = -0.0007321$)

Hằng thực được lưu trữ dưới dạng luỹ thừa trong bộ nhớ, không quan trọng chúng được viết thực sự như thế nào. Bởi vậy, nếu số thực có thể được biểu diễn dưới dạng phân số thì chúng cũng sẽ được biểu diễn gần đúng. Thậm chí, giữa số nguyên và số thực có cùng giá trị, chúng cũng được biểu diễn khác nhau. Ví vụ **43** là số nguyên, trong khi **43.0** (hoặc **43.**) là số thực, và chúng sẽ được biểu diễn khác nhau trong bộ nhớ.

Phạm vi và độ chính xác của hằng thực không được chỉ ra một cách chuẩn xác, nhưng độ chính xác khoảng 6–7 chữ số thập phân.

1.5.3 Hằng ký tự

Hằng ký tự là một chuỗi các ký tự nằm trong cặp dấu nháy đơn (‘ ’) hoặc nháy kép (“ ”). Ngoại trừ các ký tự điều khiển (chẳng hạn, #27 là ESC), những ký tự khác thuộc bảng mã ký tự ASCII (*American Standard Code for Information Interchange – Bảng mã chuẩn dùng để trao đổi thông tin giữa các thiết bị máy tính, gồm 256 ký tự, kể cả các chữ cái, ký tự thông thường, ký tự điều khiển và ký tự đồ họa*) đều có thể được sử dụng để biểu diễn hằng ký tự. Bởi vì mỗi ký tự trong bảng mã ASCII tương ứng với số thứ tự duy nhất của nó, nên các giá trị của hằng ký tự có sự phân biệt giữa chữ thường và chữ hoa.

Ví dụ:
“HANOI” khác với **“Hanoi”,** hoặc **“Hai Phong”** khác với **“Hai phong”**,…

Cũng cần phân biệt rõ khái niệm *hằng* mà ta vừa đề cập trên đây với *hằng* được khai báo trong câu lệnh thuộc tính **PARAMETER**. Khái niệm *hằng* ở đây là những *giá trị cụ thể*, chúng có thể được gán cho *biến* hoặc *hằng có tên*. Còn *hằng* trong khai báo **PARAMETER** là *hằng* có tên được xác định bởi *tên hằng* và nhận *giá trị cụ thể* là *hằng* theo khái niệm ở đây; *hằng* đó sẽ *không bị thay đổi giá trị* trong quá trình thực hiện chương trình. Ví dụ, hãy xét đoạn chương trình sau:

```
REAL, PARAMETER :: X = 12.
```

```
REAL Y, Z
Y = 23.
Z = X + Y
PRINT*, X, Y, Z
END
```

Ở đây, **X** là *hằng* (có tên), nhận giá trị không đổi (là *hằng số*) bằng **12**. Còn **Y** và **Z** là các *biến*, trong đó **Y** được gán bởi *hằng số* có giá trị bằng **23**.

1.6 TÊN BIẾN VÀ TÊN HẰNG

Ta đã thấy rằng vị trí bộ nhớ có thể được chỉ định bởi tên tượng trưng (*symbolic names*), gọi là *tên biến* hoặc *tên hằng*, như **SoTien** và **LaiSuat**. Tên biến, tên hằng có thể gồm 1 đến 31 ký tự, và phải bắt đầu bởi một chữ cái tiếng Anh. Các ký tự được sử dụng để cấu tạo tên biến, tên hằng gồm 26 chữ cái tiếng Anh, không phân biệt chữ thường, chữ hoa, (A–Z và a–z), 10 chữ số (0–9), và *dấu gạch dưới* (_).

Ngoại trừ hằng xâu ký tự, Fortran không phân biệt tên viết bằng chữ thường hay chữ hoa, ví dụ **MYNAME** và **MyName** là như nhau. Có lẽ những người có truyền thống lập trình Fortran lâu năm thường viết chương trình chỉ bằng chữ cái in hoa. Tuy nhiên ta nên viết lẫn cả chữ thường và chữ hoa cho dễ đọc. Chẳng hạn, nếu ta viết **SoTien** chắc chắn sẽ dễ hiểu hơn là viết **SOTIEN.** Mặt khác, vì Fortran 90, và cả các phiên bản mới hơn sau này, không khống chế độ dài tên chỉ 6 ký tự như các phiên bản cũ, nên để rõ ràng, tên viết dài có thể sẽ tốt hơn tên viết ngắn, vì nó mang tính gợi nhớ hơn. Chẳng hạn, nên viết **SoTien** thay cho cách viết đơn giản **ST**.

Sau đây là một số ví dụ về cách đặt tên biến, tên hằng hợp lệ và không hợp lệ:

Tên hợp lệ	Tên không hợp lệ
X	**X+Y** (vì chứa dấu + là một phép toán)
R2D2	**SHADOW FAX** (vì chứa dấu cách)
Pay_Day	**2A** (vì ký tự đầu tiên là chữ số)
ENDOFTHEMONTH	**OBI-WAN** (vì chứa dấu – là một phép toán)

Biến là vị trí bộ nhớ mà giá trị của nó có thể bị thay đổi trong quá trình thực hiện chương trình. Tên của biến được cấu tạo theo qui tắc trên đây. Biến có kiểu và loại dữ liệu xác định, được cho bởi khai báo kiểu, ví dụ:

```
INTEGER X              ! X là biến nguyên 4 byte
REAL LaiSuat           ! LaiSuat là biến thực 4 byte
CHARACTER LETTER       ! LETTER là biến ký tự độ dài bằng 1
REAL :: A = 1          ! A là biến thực nhận giá trị khởi tạo 1
```

Chú ý rằng, biến có thể được khởi tạo khi khai báo nó, như câu lệnh cuối cùng ở ví dụ trên. Trong trường hợp này phải sử dụng dấu hai chấm kép (**::**). Giá trị của biến được khởi tạo theo cách này có thể bị thay đổi trong quá trình chương trình thực hiện.

Mặc dù các biến **X, LaiSuat** và **LETTER** đã được khai báo trong đoạn chương trình trên, nhưng giá trị của chúng vẫn chưa được xác định. Bạn đọc (đặc biệt là những người mới bắt đầu lập trình) phải chú ý tránh việc tham chiếu đến các biến chưa được xác định này, vì nó có thể sẽ dẫn đến

lỗi trong lúc thực hiện chương trình (*Run–time error*), rất khó gỡ rối. Ví dụ, khi chạy chương trình sau đây ta sẽ nhận được kết quả đúng:

```
Real x, y, z
x = 3.0
y = 2.0
z = x / y
print*, x, y, z
end
```

Nhưng nếu *bỏ đi dòng thứ hai và thứ ba* rồi chạy lại chương trình thì lỗi *Run–time error* sẽ xuất hiện do câu lệnh **z = x/y** đã tham chiếu đến các biến **x** và **y** chưa xác định.

Biến có thể được xác định bằng nhiều cách, ví dụ bằng việc khởi tạo nó (như ví dụ trước) hoặc bằng việc gán giá trị cho nó, như trong ví dụ trên đây hoặc ở những ví dụ trước.

Biến cũng có thể được gán giá trị ban đầu bằng lệnh **DATA** sau khi đã khai báo, ví dụ:

```
REAL A, B
INTEGER I, J
DATA A, B / 1, 2 / I, J / 0, -1/
```

Câu lệnh **DATA** trên đây lần lượt gán các giá trị **1** cho biến **A**, **2** cho biến **B**, **0** cho biến **I** và **–1** cho biến **J**.

Tên (kể cả tên biến, tên hằng và tên chương trình) trong chương trình phải là duy nhất. Chẳng hạn, nếu chương trình được đặt tên là **TinhTien**, thì việc khai báo một biến khác cùng tên sẽ dẫn đến lỗi.

Các biến đã mô tả trong những ví dụ ở trên gọi là các biến vô hướng, hay biến đơn, vì tại một thời điểm chúng chỉ lưu một giá trị đơn nhất. Ngoài các biến vô hướng còn có các loại biến khác, chẳng hạn biến mảng. Ta sẽ đề cập chi tiết đến các loại biến này sau.

1.7 QUI TẮC KIỂU ẨN

Các phiên bản trước của Fortran có một qui tắc đặt tên ngầm định được gọi là qui tắc kiểu ẩn (*implicit type rule*). Theo qui tắc này, các biến bắt đầu bằng các chữ cái **I, J, K, L, M, N** được tự động hiểu là biến có kiểu số nguyên (**INTEGER**), còn các biến bắt đầu bằng những chữ cái khác, nếu không được khai báo rõ ràng, sẽ được hiểu là biến thực (**REAL**). Để bảo đảm tính tương thích của các chương trình viết với các phiên bản trước, qui tắc này vẫn được áp dụng ngầm định trong Fortran 90.

Tuy nhiên, trong một số tình huống, qui tắc kiểu ẩn có thể dẫn đến lỗi chương trình trầm trọng do những sơ suất đáng tiếc khi đặt tên biến. Giá trị thực có thể được gán một cách không cố ý cho biến nguyên, làm cho phần thập phân sau dấu chấm thập phân bị chặt cụt. Ví dụ, nếu không khai báo kiểu **REAL** cho biến **LaiSuat** thì câu lệnh

```
LaiSuat = 0.12
```

trong chương trình sẽ gán giá trị **0** cho biến **LaiSuat**, vì nó được ngầm hiểu là biến nguyên.

Để đề phòng những lỗi như vậy, ngay từ đầu chương trình ta **nên đưa vào** câu lệnh sau

IMPLICIT NONE

Câu lệnh này sẽ xoá bỏ thuộc tính qui tắc kiểu ẩn, do đó tất cả các biến sử dụng trong chương trình bắt buộc phải được khai báo. Đó là cách lập trình tốt, vì có khai báo ta mới buộc phải để tâm đến biến và ý nghĩa của nó.

Sau đây ta sẽ xét một ví dụ về giải bài toán chuyển động trong trường trọng lực.

Nếu một hòn đá được tung lên thẳng đứng với tốc độ ban đầu u, quãng đường dịch chuyển thẳng đứng s của nó sau thời gian t được cho bởi công thức $s(t) = ut - \frac{gt^2}{2}$, trong đó g là gia tốc trọng trường. Bỏ qua lực cản của không khí, hãy tính giá trị của s, khi cho các giá trị của u và t.

Để lập chương trình giải bài toán này ta có thể hình dung lôgic chuẩn bị chương trình như sau:

1) Nhập các giá trị g, u và t vào máy tính

2) Tính giá trị của s theo công thức đã cho

3) In giá trị của s

4) Kết thúc

Nhìn dàn bài này có thể một số người cho là nó quá tầm thường, thậm chí họ cho rằng nó lãng phí thời gian viết ra. Và do đó, ta sẽ không lấy làm ngạc nhiên tại sao một số người trong đó, nhất là những người mới bắt đầu lập trình, lại thích làm trực tiếp trên máy tính, và lập trình bước 2 trước bước 1, để rồi lúng túng trước kết quả nhận được. Thực tế điều này rất quan trọng, vì nó tạo cho ta thói quen phân tích bài toán một cách kỹ lưỡng, thiết kế chương trình có tính lôgic, và chọn tên biến, kiểu biến để khai báo một cách phù hợp nhất.

Dựa theo các bước trên đây ta có thể viết chương trình như sau:

Ví dụ 1.4: Chuyển động trong trường trọng lực

```
PROGRAM ChuyenDongThangDung
! Chuyen dong thang dung duoi truong luc trong
IMPLICIT NONE    ! Xóa bỏ qui tắc kiểu ẩn
REAL, PARAMETER :: G = 9.8 ! Gia tốc trọng trường
REAL S           ! Quãng đường (m)
REAL T           ! Thời gian
REAL U           ! Tốc độ ban đầu (m/s)
PRINT*, ' Thoi gian    Quang duong'
PRINT*
U = 60
T = 6
S = U * T - G / 2 * T ** 2
```

```
PRINT*, T, S
END PROGRAM ChuyenDongThangDung
```

Trước hết, khai báo **G** là *hằng*, vì giá trị của nó được xác định không thay đổi trong chương trình và nhận giá trị bằng 9.8. Vì trong chương trình có sử dụng câu lệnh **IMPLICIT NONE** do đó ta phải khai báo tất cả các biến. Bạn đọc có thể kiểm chứng tác dụng của câu lệnh này bằng cách thử *bỏ qua một câu lệnh khai báo biến* nào đó (thêm dấu chấm than vào đầu dòng lệnh) và chạy lại chương trình để xem Fortran phản ứng như thế nào.

1.8 PHONG CÁCH LẬP TRÌNH

Trên thực tế có thể xảy ra tình huống ta cần sử dụng lại hoặc nâng cấp các chương trình đã lập từ rất lâu rồi, hoặc khai thác các chương trình do một người nào đó viết. Sẽ rất khó khăn nếu trong chương trình chẳng có một lời chú thích nào cả. Đối với những chương trình của mình, có thể ta đã quên đi những gì mình đã viết. Việc tìm hiểu lại một chương trình không có những lời chú thích như vậy đôi khi làm cho ta nản chí, không đủ kiên nhẫn để thực hiện.

Để tránh tình trạng đó, cần phải có một *phong cách lập trình* tốt. Nghĩa là trong chương trình phải có những lời chú thích đúng chỗ, đầy đủ, rõ ràng; trong các câu lệnh nên chèn vào những dấu cách hợp lệ, sử dụng hợp lý các ký tự in thường và in hoa; giữa các đoạn chương trình nên có các dòng trắng; nên phân cấp các câu lệnh để bố trí chúng sao cho có sự thụt, thò, dễ theo dõi.

Chẳng hạn, trong các chương trình được viết trên đây, chúng ta thường đưa vào những lời chú thích mang ý nghĩa mô tả, như dòng mô tả chương trình sẽ làm gì, các biến được khai báo có ý nghĩa gì,…

1.9 BIỂU THỨC SỐ

Trong chương trình ***ChuyenDongThangDung*** ở ví dụ 1.4 ta đã sử dụng dạng mã nguồn sau:

```
U * T - G / 2 * T ** 2
```

Đây là một ví dụ về *biểu thức số* biểu diễn bằng ngôn ngữ Fortran, là *công thức liên kết* các hằng, các biến (và các hàm, như hàm tính căn bậc hai) bằng các phép toán thích hợp. Nó chỉ ra qui tắc để tính giá trị của một biểu thức đại số thông thường. Trong trường hợp trên đây, biểu thức chỉ tính một giá trị đơn nên nó được gọi là biểu thức vô hướng.

Thứ tự thực hiện các phép tính trong một biểu thức được xác định bởi thứ tự ưu tiên của các phép toán. Tuy nhiên, nếu trong biểu thức có các bộ phận nằm trong ngoặc đơn ( ) thì chúng luôn luôn được thực hiện trước tiên. Chẳng hạn, biểu thức **1 + 2 * 3** sẽ cho kết quả là **7**, trong khi **(1 + 2) * 3** sẽ cho kết quả là **9**. Chú ý rằng **–3**2** sẽ cho kết quả là **–9** chứ không phải **9**.

Khi có các phép toán cùng bậc ưu tiên xuất hiện liên tiếp nhau trong biểu thức, chúng sẽ được thực hiện theo thứ tự từ trái sang phải, ngoại trừ phép lấy lũy thừa. Do đó, biểu thức **B/C*A** được thực hiện như **(B/C)*A** mà không phải như **B/(C * A).** Đối với các phép toán lũy thừa, thứ tự thực hiện là từ phải sang trái. Ví dụ, biểu thức **A**B**C** được thực hiện theo nguyên tắc **A**(B**C).**

1.9.1 Phép chia với số nguyên

Đối với những người mới lập trình bằng Fortran, đây quả là một vấn đề không đơn giản, bởi vì nhiều khi kết quả nhận được của các biểu thức nằm ngoài dự đoán của họ. Vấn đề là ở chỗ, khi một đại lượng có kiểu số nguyên (hằng, biến hoặc biểu thức nguyên) chia cho một đại lượng có kiểu số nguyên khác, kết quả nhận được cũng sẽ có kiểu số nguyên, do phần lẻ thập phân bị cắt bỏ. Ta hãy xét các ví dụ sau đây.

10 / 3 cho kết quả là **3**

19 / 4 cho kết quả là **4**

4 / 5 cho kết quả là **0**

-8 / 3 cho kết quả là **-2**

3 * 10 / 3 cho kết quả là **10**

10 / 3 * 3 cho kết quả là **9**

Như vậy, khi chia hai đại lượng nguyên cho nhau, kết quả nhận được là phần nguyên của thương, còn phần dư bị cắt bỏ.

1.9.2 Biểu thức hỗn hợp

Fortran 90 cho phép thực hiện phép tính với biểu thức chứa các toán hạng có kiểu khác nhau. Nguyên tắc chung là các kiểu dữ liệu "yếu hơn" hoặc "đơn giản hơn" buộc phải chuyển đổi sang kiểu dữ liệu "mạnh hơn". Vì kiểu số nguyên là đơn giản nhất, cho nên trong biểu thức có các toán hạng nguyên và thực thì các toán hạng nguyên phải chuyển thành các toán hạng có kiểu thực. Tuy nhiên, quá trình chuyển đổi này chỉ thực hiện đối với từng phép tính mà không nhất thiết áp dụng cho cả biểu thức. Ví dụ:

10 / 3.0 cho kết quả là **3.33333**

4. / 5 cho kết quả là **0.8**

2(-2)** cho kết quả là **0,** vì **2**(–2)=1/(2**2) = 1/4**

Nhưng biểu thức

3 / 2 / 3.0

sẽ cho kết quả bằng **0.333333** vì **3/2** được tính trước, nhận giá trị nguyên bằng 1.

1.10 LỆNH GÁN. GÁN HẰNG, GÁN BIỂU THỨC

Lệnh gán là câu lệnh được sử dụng phổ biến nhất trong lập trình. Cú pháp câu lệnh gán có dạng:

vname = expr

Trong đó ***vname*** là *tên* của biến hoặc hằng, ***expr*** là giá trị (hằng) hoặc biểu thức. Mục đích của câu lệnh gán là tính giá trị của biểu thức ở vế phải (nếu cần) và gán cho biến/hằng ở vế trái. Như vậy,

dấu bằng (=) trong câu lệnh gán hoàn toàn không có nghĩa như dấu bằng trong toán học, mà nó được hiểu là *dấu gán*, và nên đọc là ***vname*** được gán bởi giá trị của ***expr***. Ví dụ, câu lệnh

X = A + B

được hiểu là nội dung của biến **X** được gán bởi giá trị của tổng nội dung của biến **A** và nội dung của biến **B**. Khi thực hiện câu lệnh, máy sẽ lấy giá trị của **A** cộng với giá trị của **B**, kết quả nhận được sau đó sẽ gán cho biến **X**.

Tương tự, câu lệnh

N = N + 1

hàm nghĩa là tăng giá trị của biến **N** lên một đơn vị. Đương nhiên trong toán học biểu thức này không có ý nghĩa. Tác động của quá trình thực hiện câu lệnh là lấy nội dung của biến **N** cộng với **1**, được bao nhiêu gán lại cho biến **N**.

Nếu ***expr*** không cùng kiểu dữ với ***vname*****,** nó được chuyển đổi sang kiểu dữ liệu của ***vname*** trước khi gán. Có nghĩa là điều đó có thể dẫn đến sai số tính toán. Ví dụ, giả sử **N** là biến nguyên, còn **X** và **Y** là những biến thực thì:

N = 10. / 3 (giá trị của **N** sẽ là **3**)

X = 10 / 3 (giá trị của **X** sẽ là **3.0**)

Y = 10 / 3. (giá trị của **Y** sẽ là **3.33333**)

Sự vô ý trong lập trình nhiều lúc cũng có thể dẫn đến kết quả sai không đáng có. Chẳng hạn, khi muốn tính trung bình cộng hai số (ví dụ điểm trung bình của hai môn học), nếu đặt tên biến các môn đó là **M1** và **M2** mà *không khai báo chúng là biến thực* (tức máy sẽ hiểu đó là hai biến nguyên theo qui tắc kiểu ẩn), thì điểm trung bình được xác định bởi câu lệnh:

TBinh = (M1 + M2) / 2

sẽ *bị chặt cụt* phần thập phân do vế phải là kết quả của *phép chia hai số nguyên*. Nếu tổng **(M1+M2)** *không chia hết* cho **2** thì kết quả nhận được là *sai*. Nhưng, nếu câu lệnh trên được viết dưới dạng:

TBinh = (M1 + M2) / 2.0

thì kết quả lại hoàn toàn chính xác mặc dù **M1** và **M2** vẫn là những biến nguyên.

Sau đây là một số ví dụ về câu lệnh gán.

C = (A ** 2 + B ** 2) ** 0.5 / (2 * A)

A = P * (1 + R / 100) ** N

Câu lệnh thứ nhất có thể được viết bằng cách khác khi sử dụng hàm thư viện **SQRT** (hàm lấy căn bậc hai) của Fortran như sau:

C = SQRT (A ** 2 + B ** 2) / (2 * A)

Tuy nhiên, không được viết câu lệnh dưới dạng:

C = (A ** 2 + B ** 2) ** (1/2) / (2 * A)

Bởi vì **(1/2)** trong biểu thức lũy thừa sẽ nhận giá trị bằng **0** do phép chia hai số nguyên cho nhau.

1.11 LỆNH VÀO RA ĐƠN GIẢN

Quá trình nhận thông tin vào và kết xuất thông tin ra của máy tính được gọi là quá trình vào ra dữ liệu. Dạng vào ra dữ liệu đơn giản nhất trong Fortran là sử dụng các lệnh **READ*** và **PRINT***, và được gọi là vào ra trực tiếp. Các dạng vào ra dữ liệu phức tạp hơn sẽ được đề cập đến trong những phần sau.

Trong các mục trước ta đã gặp các câu lệnh với **READ*** và **PRINT***, nhưng chưa giải thích gì về chúng. Ở đây ta sẽ thấy rằng đó là những câu lệnh rất thường dùng mà ta cần phải tìm hiểu ngay.

1.11.1 Lệnh vào dữ liệu

Từ những ví dụ trên nhận thấy các biến được gán giá trị bằng cách sử dụng câu lệnh gán, chẳng hạn như trong chương trình ***TinhTien*:**

```
SoTien = 1000.0
LaiSuat = 0.09
```

Cách làm này không linh hoạt, vì khi muốn chạy chương trình với các giá trị *số tiền gốc* hoặc *lãi suất* khác nhau, mỗi lần như vậy ta phải thay đổi trực tiếp các câu lệnh gán này trong chương trình, sau đó biên dịch lại rồi mới thực hiện chương trình. Thay cho cách này ta có thể sử dụng câu lệnh **READ*** như sau:

```
READ*, SoTien, LaiSuat
```

Trong trường hợp này, khi chạy chương trình, máy sẽ chờ ta *gõ giá trị của các biến từ bàn phím*. Các giá trị này có thể được gõ trên cùng một dòng, phân cách nhau bởi các dấu cách, dấu phẩy hoặc trên các dòng khác nhau.

Dạng tổng quát của lệnh **READ*** như sau:

```
READ*, list
```

Trong đó ***list*** là danh sách biến; nếu có nhiều hơn một biến thì chúng được viết cách nhau bởi dấu phẩy.

Khi vào dữ liệu với lệnh **READ*** cần chú ý một số điểm sau.

– Mỗi dòng dữ liệu được gõ *liên tục* (không dùng dấu **ENTER** xuống dòng) được gọi là một bản ghi. Nếu *dòng dữ liệu* quá dài, không hiển thị đủ trên một *dòng màn hình*, nó sẽ được tự động "cuộn" xuống dòng dưới, nhưng vẫn thuộc cùng một bản ghi.

– Mỗi một lệnh **READ** khi nhận dữ liệu đòi hỏi một bản ghi mới. Khi nhập dữ liệu vào từ bàn phím, mỗi bản ghi được phân tách nhau bởi dấu ENTER (nhấn phím ENTER). Do đó, câu lệnh:

```
READ*, A, B, C
```

sẽ được thỏa mãn với một bản ghi chứa 3 giá trị:

3 4 5

Trong khi các câu lệnh:

```
READ*, A
READ*, B
READ*, C
```

đòi hỏi phải đưa vào 3 bản ghi, mỗi bản ghi chứa 1 giá trị (tức là trong khi nhập dữ liệu sẽ dùng dấu ENTER xuống dòng sau khi gõ vào một giá trị):

```
3
4
5
```

– Khi gặp một lệnh **READ** mới, những dữ liệu *chưa được đọc* trên bản ghi hiện thời (nếu còn) sẽ bị bỏ qua, và một bản ghi mới khác sẽ được tìm đến để nhận dữ liệu.

– Nếu lệnh **READ** đòi hỏi nhiều dữ liệu hơn số dữ liệu chứa trên bản ghi hiện thời nó cũng sẽ tìm đến bản ghi mới tiếp theo để nhận tiếp dữ liệu. Do đó, nếu dữ liệu không đủ đáp ứng cho lệnh **READ** thì chương trình sẽ bị kết thúc với thông báo lỗi.

Ví dụ, các câu lệnh

```
READ*, A
READ*, B, C
READ*, D
```

với các bản ghi dữ liệu đưa vào là (ở đây mỗi dòng được xem là một bản ghi):

```
1 2 3
4
7 8
9 10
```

sẽ có hiệu quả giống như các lệnh gán sau:

```
A = 1
B = 4
C = 7
D = 9
```

Tức là các giá trị **2, 3** trên bản ghi thứ nhất, **8** trên bản ghi thứ ba và **10** trên bản ghi thứ tư, bị bỏ qua.

1.11.2 Đọc dữ liệu từ file TEXT

Trên thực tế thường xảy ra tình huống, ta đang muốn kiểm tra, chỉnh sửa chương trình, trong đó mỗi lần chạy, chương trình cần phải đọc vào nhiều số liệu. Chẳng hạn, khi viết một chương trình tính trung bình của 10 số, chắc chắn ta sẽ cảm thấy rất khó chịu nếu cứ phải nhập vào 10 số từ bàn phím (bằng lệnh **READ***) mỗi khi thử lại chương trình. Đó là chưa nói đến những chương trình đòi hỏi nhiều dữ liệu hơn, như tính điểm trung bình chung học tập cho một lớp sinh viên khoảng 50 người,

100 người,… Để tránh phiền phức trong những trường hợp như vậy, Fortran cung cấp một phương thức vào dữ liệu khá đơn giản nhưng rất hữu ích, là sử dụng file số liệu.

Ý tưởng là ở chỗ, trước khi chạy chương trình, ta cần phải chuẩn bị số liệu và lưu chúng vào một file riêng biệt trên đĩa. File số liệu này có thể được tạo ra bằng một trình soạn thảo bất kỳ và được ghi lại dưới dạng file TEXT (ASCII file) với một tên file nào đó, chẳng hạn **SOLIEU.TXT**. Khi chạy chương trình, máy sẽ tìm đến file này và nhận số liệu từ đó. Muốn vậy, thay cho câu lệnh **READ***, ta sử dụng hai câu lệnh mới có chức năng tham chiếu đến file và đọc dữ liệu từ file. Để tiện trình bày, ta xét ví dụ đơn giản sau. Giả sử ta có file số liệu với tên là **SOLIEU.TXT** mà nội dung của nó chỉ gồm 3 số ở dòng đầu tiên của file:

```
3 4 5
```

Bây giờ ta hãy gõ chương trình sau đây vào máy và chạy thử:

```
PROGRAM ThuFile
REAL A, B, C
OPEN(1, FILE = 'SOLIEU.TXT') ! Mở file
READ(1, *) A, B, C          ! Đọc số liệu từ file
PRINT*, A, B, C
END
```

Câu lệnh **OPEN** kết nối số 1 với file **SOLIEU.TXT** trên đĩa. Số 1 này được gọi là **UNIT**, mang hàm nghĩa chỉ thị số hiệu file (hay *kênh* vào/ra). Câu lệnh **READ** ở đây (khác với lệnh **READ***) định hướng cho chương trình tìm và đọc số liệu trong file được kết nối với **UNIT 1**. Thông thường số **UNIT** nhận giá trị trong khoảng **1–9999**.

1.11.3 Lệnh kết xuất dữ liệu

Lệnh **PRINT*** là câu lệnh rất thuận tiện cho việc kết xuất thông tin khi lượng dữ liệu không lớn. Thông thường nó được sử dụng trong quá trình xây dựng, phát triển chương trình, hoặc đưa ra những kết quả tính toán trung gian để theo dõi tiến trình làm việc của chương trình. Dạng tổng quát của nó như sau:

PRINT*, *list*

Trong đó ***list*** có thể là danh sách hằng, biến, biểu thức và xâu ký tự, được viết cách nhau bởi dấu phẩy (,). Xâu ký tự phải được đặt trong cặp dấu nháy đơn (‘ ’) hoặc dấu nháy kép (“ ”). Nếu ***list*** là danh sách rỗng thì lệnh này có dạng đơn giản là **PRINT*** và có ý nghĩa chèn thêm một dòng trống. Ví dụ:

```
PRINT*
PRINT*, "Can bac hai cua ", 2, ' la', SQRT(2.0)
```

Sau đây là một số qui tắc chung của lệnh **PRINT**.

– Mỗi câu lệnh **PRINT*** tạo ra một bản ghi mới. Nếu nội dung bản ghi quá dài nó sẽ được "cuộn" xuống các dòng tiếp theo.

– Đối với số thực, tùy theo độ lớn giá trị của số được in mà chúng có thể được biểu diễn dưới dạng dấu phẩy tĩnh hoặc dấu phẩy động. Nếu muốn in ở dạng cầu kỳ, có qui cách, ta có thể sử dụng lệnh định dạng **FORMAT.** Ví dụ, để in số **123.4567** dưới dạng dấu phẩy tĩnh trên 8 cột, với 2 chữ số sau dấu chấm thập phân, ta có thể viết:

```
X = 123.4567
PRINT 10, X
10 FORMAT( F8.2 )
```

Lệnh định dạng **FORMAT** cho phép bố trí khuôn mẫu in theo qui cách được chỉ ra ở phần trong dấu ngoặc đơn. Trong ví dụ trên, nếu muốn in giá trị của biến **X** kèm theo những chú thích hợp lý ta có thể đưa thêm vào các hằng ký tự. Chẳng hạn, thay cho câu lệnh trên đây ta có thể viết:

```
10 FORMAT( "Gia tri bien X = ", F8.2 )
```

Hằng ký tự phải được đặt trong cặp dấu nháy đơn, hoặc dấu nháy kép. Ta sẽ đề cập chi tiết đến câu lệnh này trong các mục sau.

Lệnh **PRINT*** cũng có thể được dùng để in một thông báo (hằng ký tự) dài quá một dòng bằng cách sử dụng ký tự nối dòng. Ví dụ:

```
PRINT*, 'Day la cau thong bao duoc &
     &viet bang lenh PRINT co noi dong'
```

1.11.4 Kết xuất ra máy in

Nếu muốn kết xuất ra máy in, ta chỉ cần đặt tham số **FILE='PRN'** trong câu lệnh **OPEN** và kết hợp với việc sử dụng lệnh **WRITE**. Ví dụ:

```
OPEN (2, FILE = 'prn' )
WRITE(2, *) 'In ra may in'
PRINT*, 'In ra man hinh'
```

Chú ý rằng lệnh **WRITE** trong trường hợp này phải gắn kết với số hiệu file **UNIT** trong lệnh **OPEN.** Lệnh này tổng quát hơn lệnh **PRINT**. Ta sẽ làm quen với câu lệnh này ở những nội dung sau.

1.12 SỬ DỤNG HÀM TRONG FORTRAN

Trên đây ta đã gặp trường hợp tính căn bậc hai của một số dương bằng hàm thư viện **SQRT** của Fortran. Đó chỉ là một trong rất nhiều hàm có sẵn do trình biên dịch cung cấp. Hệ thống các hàm này (và cả những hàm do người dùng xây dựng bổ sung thêm) lập thành một thư viện các hàm trong (hay còn gọi là hàm thư viện), cho phép ta sử dụng chúng như những "hộp đen" mà không cần biết chúng được xây dựng như thế nào. Mỗi một hàm như vậy thực hiện một chức năng tính toán khác nhau (như lấy căn bậc hai, tính cosine,...) và cho một giá trị kết quả. Các hàm này được tham chiếu trực tiếp

trong các biểu thức. Khi tính biểu thức, hàm sẽ được thực hiện theo trình tự thuật toán đã xây dựng và giá trị tính được của hàm sẽ thay thế vị trí tham chiếu đến hàm.

Ví dụ, xét đoạn chương trình sau:

```
REAL X, Y
X = 16.0
Y = 5.6 + SQRT(X)
PRINT*, X, Y
END
```

Trong chương trình này, để tính giá trị của **Y**, cần phải tính **SQRT(X).** Vì **X = 16.0** nên hàm **SQRT(X) = SQRT(16.0)** sẽ cho kết quả là $\sqrt{16.0}$ = 4.0. Do đó, **Y = 5.6 + 4.0 = 9.6**. Mặc dù vậy ta hoàn toàn không biết cách tính căn bậc hai mà hàm **SQRT** thực hiện như thế nào. Và ta sử dụng hàm **SQRT** để tính căn bậc hai của một số X nào đó như là một sự thừa nhận tính đúng đắn, chính xác của nó.

Fortran cung cấp cho ta một thư viện các hàm khá phong phú. Để tiện sử dụng khi trình bày ở các phần sau, trong bảng 1.6 nêu ra một số hàm thông dụng nhất.

Khi sử dụng các hàm thư viện ta cần đặc biệt chú ý đến tính năng của chúng. Ví dụ, các hàm **INT** và **NINT** được sử dụng để đổi số thực thành số nguyên, nhưng hàm **INT** sẽ cắt bỏ phần thập phân trong khi hàm **NINT** làm tròn số thực đến số nguyên gần nhất:

INT(5.3) là 5 NINT(5.3) là 5
INT(5.8) là 5 NINT(5.8) là 6
INT(−5.3) là −5 NINT(−5.3) là −5
INT(−5.8) là −5 NINT(−5.8) là −6

Bảng 1.6 Một số hàm thư viện thường dùng của Fortran

Tên hàm và lời gọi hàm	Chức năng của hàm	Kiểu dữ liệu của đối số	Kiểu dữ liệu của kết quả
INT (X)	Chuyển số X thành số nguyên sau khi chặt cụt phần thập phân	REAL	INTEGER
NINT (X)	Làm tròn số X đến số nguyên gần nhất	REAL	INTEGER
REAL (X)	Chuyển số nguyên X thành số thực	INTEGER	REAL
ABS (X)	Tìm giá trị tuyệt đối của X	REAL	REAL
IABS (X)	Tìm giá trị tuyệt đối của X	INTEGER	INTEGER
SQRT (X)	Tính căn bậc hai của X	REAL	REAL
EXP (X)	Tính e^X	REAL	REAL
ALOG (X)	Tính lnX (logarit tự nhiên)	REAL	REAL
ALOG10 (X)	Tính lgX (logarit thập phân)	REAL	REAL
SIN (X)	Tĩnh Sine của X	REAL	REAL
COS (X)	Tính Cosine của X	REAL	REAL
TAN (X)	Tính Tang của X	REAL	REAL
MOD (X,Y)	Tính phần dư của phép chia hai số nguyên X/Y	INTEGER	INTEGER
MAX0(X_1,...,X_N)	Tìm giá trị lớn nhất của dãy số	INTEGER	INTEGER

MIN0(X_1,...,X_N)	**X_1,...,X_N Tìm giá trị nhỏ nhất của dãy số X_1,...,X_N**	**INTEGER**	**INTEGER**
AMAX1(X_1,...,X_N)	**Tìm giá trị lớn nhất của dãy số X_1,...,X_N**	**REAL**	**REAL**
AMIN1(X_1,...,X_N)	**Tìm giá trị nhỏ nhất của dãy số X_1,...,X_N**	**REAL**	**REAL**

Hàm **REAL** được sử dụng để đổi một số nguyên thành một số thực. Nếu các biến **TONG** và **N** là những biến nguyên còn **T_BINH** là biến thực, khi đó hai câu lệnh sau đây có thể cho kết quả hoàn toàn khác nhau:

```
T_BINH = TONG / N
```

và

```
T_BINH = REAL(TONG)/REAL(N)
```

Những hàm trên đây chỉ đòi hỏi có một đối số, nhưng như đã thấy trong bảng 1.6, có thể có những hàm đòi hỏi hai đối số hoặc nhiều hơn. Ví dụ, hàm **MOD** đòi hỏi hai đối số, trong khi các hàm **MAX0**, **MIN0**, **AMAX1**, **AMIN1** lại có thể có số lượng đối số lớn hơn hoặc bằng hai.

Ví dụ 1.5. Giả sử A, B, C là ba đỉnh của một tam giác. Ký hiệu AB, AC, BC là các cạnh của tam giác, ALFA là góc kẹp giữa hai cạnh AB và AC. Cho biết độ dài của các cạnh AB, AC và số đo bằng độ của góc ALFA, có thể tính độ dài của cạnh BC theo công thức:

$$BC^2 = AB^2 + AC^2 - 2.AB.AC.Cos(Alfa)$$

Viết chương trình nhập vào độ dài các cạnh AB, AC và góc ALFA (độ) rồi tính độ dài của cạnh BC.

Ta có chương trình sau:

```
REAL AB, AC, BC, ALFA
REAL PI
PI = 4.0*ATAN (1.0)
PRINT*,'Cho do dai cac canh AB, AC: '
READ*, AB,AC
PRINT*,'Cho so do goc (do) giua AB va AC: '
READ*, ALFA
BC = SQRT (AB**2 + AC**2 - 2*COS(ALFA*PI/180.0) )
PRINT*,'Do dai canh BC = ', BC
END
```

Trong chương trình trên, hàm **ATAN** để tính *Arctang*. Vì *Tang* của góc $\pi/4$ bằng 1 nên *Arctang* của 1 bằng $\pi/4$.

Một trong những hàm rất quan trọng được sử dụng trong nhiều lĩnh vực là hàm e^x, trong đó e là một hằng số, có giá trị bằng 2.718282 khi lấy tròn số đến sáu chữ số thập phân. Ví dụ, hàm mật độ xác suất của biến ngẫu nhiên tuân theo luật phân bố chuẩn chuẩn hóa có dạng:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Biểu thức ước lượng giá trị của hàm này viết bằng ngôn ngữ Fortran có thể có dạng:

```
F = 1.0/SQRT(2.0*PI)*EXP(0.5*X*X)
```

BÀI TẬP CHƯƠNG 1

1.1 Hãy cho biết trong các tên biến dưới đây những tên nào viết sai theo qui ước của Fortran, tại sao: (a) A2 (b) A.2 (c) 2A (d) 'A'ONE (e) AONE (f) X_1 (g) MiXedUp (h) Pay Day (i) U.S.S.R. (j) Pay_Day (k) min*2 (l) PRINT

1.2 Hãy xác định xem trong những hằng sau đây hằng nào viết đúng, hằng nào viết sai theo qui ước của Fortran, tại sao: (a) 9,87 (b) .0 (c) 25.82 (d) –356231 (e) 3.57*E2 (f) 3.57E2.1 (g) 3.57E+2 (h) 3,57E–2

1.3 Hãy viết các biểu thức sau đây dưới ngôn ngữ Fortran:
(a) $ax^2 + bx + c = 0$; (b) $ax^2 + bx + c > 0$; (c) $ax^2 + bx + c < 0$; (d) $ax^2 + bx + c \neq 0$; (e) $ax^2 + bx + c \geq 0$; (f) $ax^2 + bx + c \leq 0$

1.4 Tìm chỗ sai trong đoạn chương trình sau:

```
INTEGER*1 A, N
INTEGER*2 B, M
REAL X
LOGICAL L
A = 12.0
N = 150
B = −54.4
M = 33456
L = .TRUE.
```

1.5 Hãy gõ đoạn chương trình sau vào máy, chạy tính thử và khảo sát những thông báo lỗi (ERROR) khi dịch chương trình rồi sửa lại cho đúng:

```
PROGRAM Dread_ful
REAL: A, B, X
X:= 5
Y = 6,67
B = X \ Y
PRINT* 'The answer is", B
END.
```

1.6 Lập chương trình nhập vào hai số thực A và B, rồi tính tổng, hiệu, tích thương của chúng. In kết quả lên màn hình với những dòng chú thích phù hợp. Hãy khảo sát điều gì sẽ xảy ra khi thực hiện phép chia cho số 0.

1.7 Hãy lập chương trình nhập vào hai số nguyên M và N, rồi tính tổng, hiệu, tích thương của chúng. In kết quả lên màn hình với những dòng chú thích phù hợp. Chú ý theo dõi và cho biết tại sao với những cặp số M, N khác nhau lại có thể cho kết quả như nhau khi thực hiện phép chia hai số.

1.8 Cho trước giá trị của ba biến thực A=2, B=3, C=5 và hai biến nguyên I=2, J=3. Hãy cho biết giá trị của các biểu thức sau nếu chúng được tính bằng chương trình Fortran:

1) A*B + C; 2) A*(B + C); 3) B/C*A; 4) B/(C * A); 5) A/I/ J; 6) I/J/A; 7) A*B**I/A ** J * 2; 8) C + (B / A) ** 3 / B * 2.; 9) A ** B ** I; 10) –B** A ** C; J / (I / J).

1.9 Nhiệt độ thế vị θ được xác định bởi công thức $\theta = T\left(\frac{1000}{p}\right)^{R/C_p}$, trong đó T ($^oC$) và p (mb) là nhiệt độ và áp suất ban đầu của phần tử khí, $R/C_p \approx 0.288$. Hãy lập chương trình nhập vào giá trị nhiệt độ và áp suất ban đầu của một phần tử khí và tính nhiệt độ thế vị của nó.

1.10 Giả sử có các khai báo sau:

```
REAL P1, X, Y
INTEGER MAXI, A, B, I
PARAMETER (P1 = 3.14159, MAXI = 1000)
```

Hãy tính giá trị của các câu lệnh hợp lệ dưới đây, đồng thời chỉ ra những câu lệnh không hợp lệ, tại sao. Cho **A=3, B=4 và X=–1.0**

```
I = A * B
I = (990 ─ MAXI) / A
I = A*Y
x = pi*y
I = A/B
X = A / B
X = A * (A/ B)
I = B / 0
I = A * (990 — MAXI)
I = (MAXI — 990) / A
X = A / Y
I = PI*A
x = pi/y
I = B/A
I = (MAXI — 990) * A
L = A * 0
I = A * MAXI — 990)
```

1.11 Cho A, B, C và X là tên của bốn biến thực (REAL), I, J và K là tên của ba biến nguyên (INTEGER). Hãy sửa các câu lệnh dưới đây cho phù hợp với qui tắc biểu diễn biểu thức số học bằng ngôn ngữ Fortran.

```
1) X = 4.0 A* C
2) A = AC
3) I = 2X—J
4) K = 3(1 ± J)
5) X = 5A/BC
6) I = 5J3
```

1.12 Viết chương trình xác định số lần đập của quả tim trong cả cuộc đời một con người. Chương trình cho phép tính với nhịp đập bất kỳ của quả tim (ví dụ 72 lần/phút) và với tuổi thọ bất kỳ của một người (ví dụ 75 tuổi). Lấy số ngày trong một năm bằng 365.25 ngày.

1.13 Thời gian bay (t – giây) và độ cao (h – mét) đạt được của một viên đạn pháo được xác định theo các công thức:

$$t = \frac{S}{v\cos\theta}$$

$$h = vt - \frac{gt^2}{2}$$

trong đó S (m) là khoảng cách từ nơi bắn đến mục tiêu; v (m/s) là vận tốc ban đầu của viên đạn; θ (radian) là góc nâng của nòng pháo; và g (m/s^2) là gia tốc trọng trường. Cho g = 9.8 m/s^2. Hãy viết chương trình nhập vào khoảng cách đến mục tiêu, góc nâng của nòng pháo và vận tốc ban đầu của viên đạn và tính thời gian bay và độ cao đạt được của viên đạn.

1.14 Biệt thự của một gia đình là một hình chữ nhật có các kích thước là X_N và Y_N. Biệt thự được xây dựng trên một khu đất cũng là hình chữ nhật có các cạnh song song với biệt thự và có các kích thước X_D và Y_D. Ngoài biệt thự, trong khu đất còn có một vườn hoa hình tròn bán kính R_H. Khoảng trống còn lại của khu đất là cỏ. Hãy viết chương trình nhập vào những giá trị hợp lệ của các kích thước của biệt thự, của khu đất và của vườn hoa và tính xem nếu một người cắt cỏ cắt được 2 m^2/s thì phải mất bao nhiêu thời gian để cắt hết cỏ trong khu đất đó.

1.15 Viết chương trình nhập vào các tử số và mẫu số của hai phân số rồi tính tổng, hiệu, tích, thương của chúng. In kết quả dưới dạng phân số và giá trị phần trăm của phân số kết quả.

1.16 Viết chương trình đọc vào giờ, phút, giây và đổi ra giờ biểu diễn dưới dạng số thập phân (ví dụ XX giờ, YY phút, ZZ giây sẽ được đổi thành HH.TTTT giờ)

1.17 Viết chương trình nhập giá trị ba điện trở của một mạch điện mắc song song và tính điện trở tương đương của mạch theo công thức:

$$\frac{1}{R_{td}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

1.18 Bộ 3 số nguyên dương m, n, p thỏa mãn điều kiện $m^2 + n^2 = p^2$ được gọi là bộ ba số Pitago (ví dụ, ba số 3, 4, 5 là một bộ số Pitago), vì ba số này thỏa mãn điều kiện là ba cạnh của một tam giác vuông, trong đó m và n là hai cạnh góc vuông, p là cạnh huyền. Cho hai số nguyên dương x và y, với x > y, khi đó có thể tạo một bộ số Pitago theo công thức sau:

$m = x^2 - y^2$

$n = 2xy$

$p = x^2 + y^2$

Viết chương trình nhập vào hai số nguyên dương và thành lập bộ số Pitago theo các công thức trên.

1.18 Tốc độ suy giảm nhiệt độ theo phương thẳng đứng (gradient thẳng đứng của nhiệt độ) trong lớp khí quyển dưới cùng có thể lấy gần đúng bằng $0.6^oC/100m$. Viết chương trình xác định nhiệt độ khí quyển ở độ cao h (m) nào đó nếu biết rằng nhiệt độ ở mực nước biển (h=0) là T (^OC).

1.19 Hãy biểu thị dưới dạng các câu lệnh của Fortran những nội dung sau:

(a) Thêm 1 vào giá trị của biến I rồi lưu kết quả vào ô nhớ của biến I.

(b) Lấy luỹ thừa 3 của I rồi cộng với J và lưu kết quả vào ô nhớ I.

(c) Chia tổng của A và B cho tích của C và D rồi lưu vào ô nhớ của biến X.

1.20 Viết chương trình tính giá trị của biểu thức sau:

$$A=\frac{\left(x^3+\sin b-c^{0.19238}\right)\left(\cos 3x+0.20345\right)}{15.172+5^x},$$

trong đó: x nhập từ bàn phím; $b=2^x-31.769$; $c=\lg(x^4+5^x)+\ln(x^2+5^b)$

1.21 Viết chương trình nhập vào toạ độ ba điểm $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$ rồi tính tích vô hướng của các vectơ AB, AC.

1.22 Sử dụng trình soạn thảo Fortran (hoặc một trình soạn thảo bất kỳ) tạo một file TEXT có tên là SOLIEU.TXT với nội dung của file như sau:

23 12.5 65.2 21
67 89 34 56 76
32 45.6
54.6 67.8 21.3

Sau đó viết chương trình đọc file số liệu theo các yêu cầu: Đọc các giá trị thứ nhất và thứ ba ở dòng 1 và gán cho các biến A, B; Đọc các giá trị thứ hai, thứ ba và thứ tư ở dòng 2 và gán cho các biến C, D, E; Đọc hai giá trị ở dòng 3 và giá trị thứ nhất ở dòng 4 và gán cho các biến X, Y, Z. In kết quả nhận được của các biến A, B, C, D, E, X, Y, Z lên màn hình và so sánh với nội dung file số liệu.

CHƯƠNG 2. CÁC CÂU LỆNH CƠ BẢN CỦA FORTRAN

Trong chương trước chúng ta đã làm quen với một số câu lệnh của Fortran, như lệnh gán, các lệnh vào ra đơn giản với **READ*** và **PRINT***, lệnh mở file **OPEN** để nhận dữ liệu từ file TEXT hoặc kết xuất thông tin ra máy in, lệnh định dạng **FORMAT**,... Với những câu lệnh đó, ta đã có thể viết được một số chương trình đơn giản. Chương này sẽ nghiên cứu những câu lệnh phức tạp hơn.

2.1 LỆNH CHU TRÌNH (DO LOOPS)

Khi viết chương trình, ta có thể bắt gặp tình huống, một hoặc nhiều câu lệnh nào đó phải thực hiện lặp lại nhiều lần giống nhau. Chẳng hạn, muốn in 10 số nguyên liên tiếp, mỗi lần in một số, ta phải dùng đến 10 câu lệnh in ra. Điều đó làm cho ta nhiều lúc cảm thấy bất tiện. Tuy nhiên, thay cho cách làm trên đây, Fortran hỗ trợ một cấu trúc câu lệnh khá đơn giản nhưng rất hiệu quả. Đó là câu lệnh chu trình, hay chu trình lặp xác định. Cú pháp câu lệnh có thể có các dạng sau.

Dạng 1:

```
DO m bdk = TriDau, TriCuoi [, Buoc]
   Các_câu_lệnh
m Câu_lệnh_kết_thúc
```

Dạng 2:

```
DO m bdk = TriDau, TriCuoi [, Buoc]
   Các_câu_lệnh
m CONTINUE
```

Dạng 3:

```
DO bdk = TriDau, TriCuoi [, Buoc]
   Các_câu_lệnh
END DO
```

Trong đó: **bdk, TriDau, TriCuoi, Buoc** phải có cùng kiểu dữ liệu; **m** là nhãn của câu lệnh kết thúc chu trình, trong trường hợp không thể sử dụng câu lệnh kết thúc như vậy, có thể thay thế nó bằng câu lệnh **m CONTINUE** như ở dạng 2. Nếu **TriDau** < **TriCuoi** thì **Buoc** phải là một số dương, ngược lại nếu **TriDau** > **TriCuoi** thì **Buoc** phải là một số âm. Nếu **Buoc=1** thì có thể bỏ qua **Buoc**.

Cấu trúc dạng 1 và dạng 2 là của Fortran 77 và các phiên bản trước đó, nhưng chúng vẫn tương thích với Fortran 90. Mặc dù vậy, do một số đặc điểm mở rộng của câu lệnh chu trình trong Fortran 90 (mà ta sẽ đề cập ở các phần sau), hiện nay người ta ít sử dụng các cấu trúc đó.

Tập **Các_câu_lệnh** nằm giữa **DO** và

m Câu_lệnh_kết_thúc

hoặc

m CONTINUE

hoặc

ENDDO

là những câu lệnh được thực hiện lặp đi lặp lại. Số lần lặp lại được xác định bởi:

Số lần lặp = MAX { (*TriCuoi* – *TriDau* + *Buoc*) / *Buoc*, 0 }

Tác động của lệnh chu trình được mô tả trên hình 2.1. Có thể tóm tắt tác động này qua các bước sau.

1) Bắt đầu chu trình **bdk** được gán giá trị bằng **TriDau**.

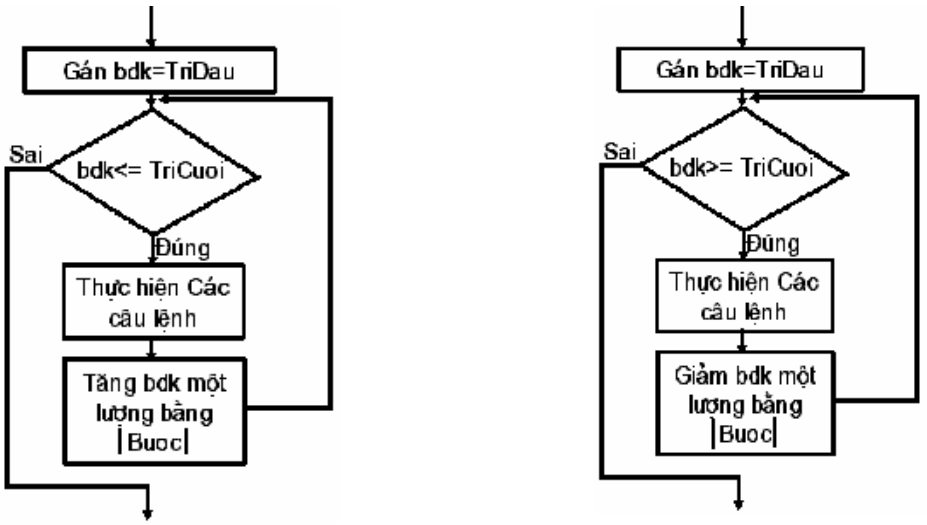
2) Sau đó chương trình sẽ thực hiện biểu thức so sánh **bdk<=TriCuoi** hoặc **bdk>=TriCuoi**:

a) Nếu biểu thức cho kết quả .**TRUE**. (đúng):

a.1) Tiếp tục thực hiện **Các_câu_lệnh**, kể cả **Câu_lệnh_kết_thúc**, nằm trong chu trình rồi tăng hoặc giảm **bdk** một lượng bằng trị tuyệt đối của **Buoc**

a.2) Quay về thực hiện bước 2)

b) Nếu biểu thức cho kết quả .**FALSE**. (sai) thì kết thúc chu trình



a) Trường hợp **TriDau<=TriCuoi** b) Trường hợp **TriDau>=TriCuoi**

Hình 2.1 Sơ đồ khối mô tả tác động của lệnh chu trình DO

Ta nhận thấy, tác động của chu trình là thực *hiện lặp đi lặp lại* **Các_câu_lệnh**, kể cả **Câu_lệnh_kết_thúc**. Mỗi lần như vậy giá trị của **bdk** sẽ thay đổi phù hợp với **Buoc**, còn **TriDau** và **TriCuoi** được giữ nguyên cho đến khi vòng lặp kết thúc. Do đó, trong phạm vi vòng lặp, tức là trong **Các_câu_lệnh** và **Câu_lệnh_kết_thúc**, tuyệt đối không được xuất hiện những câu lệnh làm thay đổi giá trị của **bdk**, **TriDau** và **TriCuoi**, nếu không sẽ dẫn đến lỗi không lường trước được.

Ví dụ 2.1. Chương trình sau đây sẽ tính tổng các số nguyên liên tiếp từ **N1** đến **N2**, trong đó **N1** và **N2** được nhập vào từ bàn phím.

```
INTEGER N1, N2, TONG, I
PRINT '(A\)', ' CHO GIA TRI N1, N2 (N1<=N2):'
```

```
READ*, N1, N2
TONG = 0
DO I = N1,N2,1
  TONG = TONG + I
  PRINT*, I
ENDDO
PRINT '(" TONG=",I5)', TONG
END
```

Khi chạy chương trình, các số nguyên liên tiếp từ **N1** đến **N2** sẽ được hiện lên màn hình và cuối cùng là thông báo kết quả tổng của các số từ **N1** đến **N2**. Các câu lệnh

```
PRINT '(A\)', ' CHO GIA TRI N1, N2 (N1<=N2):'
```

và

```
PRINT '(" TONG=",I5)', TONG
```

đã chứa trong đó lệnh định dạng **FORMAT**. Tuy nhiên, nếu cảm thấy hơi xa lạ, có thể thay thế phần định dạng này bởi dấu sao (*).

Trong câu lệnh

```
DO I = N1,N2,1
```

số 1 cuối cùng là giá trị của **Buoc**, nó có thể được bỏ qua mà không ảnh hưởng gì đến kết quả. Nhưng nếu thay nó bằng –1 thì khi nhập **N1** và **N2** cần phải lưu ý **N1>=N2**, nếu không sẽ nhận được kết quả bất ngờ (!?), vì khi đó *số lần lặp* bằng 0.

Các câu lệnh

```
DO I = N1,N2,1
  TONG = TONG + I
  PRINT*, I
ENDDO
```

cũng có thể được thay thế bởi các câu lệnh sau đây

```
DO 100 I = N1, N2
  TONG = TONG + I
100 PRINT*, I
```

Trong trường hợp này, câu lệnh

```
100 PRINT*, I
```

là câu lệnh kết thúc chu trình, và vì **Buoc** có giá trị bằng 1 nên ta đã bỏ qua nó. Ta cũng có thể dùng câu lệnh **CONTINUE** để kết thúc chu trình như sau:

```
DO 100 I = N1, N2
  TONG = TONG + I
  PRINT*, I
100 CONTINUE
```

Lệnh **CONTINUE** ở đây có thể xem là "thừa", tuy vậy trong nhiều trường hợp, để an toàn và rõ ràng hơn, ta có thể sử dụng những câu lệnh "thừa" kiểu này.

Ví dụ 2.2. Chương trình tính căn bậc hai của số ***a*** theo phương pháp Newton có thể được mô tả như sau:

1) Nhập vào số **a**

2) Khởi tạo **x** bằng **1** (gán giá trị cho **x** bằng 1)

3) Lặp lại **6** lần các bước sau đây:

a) Thay **x** bởi **(x + a/x)/2**

b) In giá trị của **x**

4) Kết thúc chương trình

Mã nguồn chương trình như sau:

```
PROGRAM Newton  ! Tinh can bac hai bang pp Newton
REAL A        ! Số sẽ lấy căn bậc hai
INTEGER I      ! Biến đếm phép lặp
REAL X        ! Giá trị gần đúng của căn bậc hai của a
WRITE( *,*) ' Cho so se lay can bac hai: '
READ*, A
PRINT*
X = 1                          ! Khởi tạo giá trị ban đầu của x (??)
DO I = 1, 6
  X = (X + A / X) / 2
  PRINT*, X
ENDDO
PRINT*
PRINT*, 'Can bac 2 cua ‘,a,’ tinh theo F90 la:',&
       SQRT(A)
END
```

Khi chạy chương trình, trên màn hình sẽ xuất hiện **6** lần giá trị của **X**. Giá trị ở dòng thứ **6** được xem là gần đúng của căn bậc hai của **a** tính bằng phương pháp lặp Newton, còn giá trị in ở dòng cuối cùng là căn bậc hai của **a** tính bằng hàm thư viện **SQRT** của Fortran. Giữa chúng có thể có sự khác nhau; khi **a** càng lớn thì sự khác nhau đó càng nhiều. Trong trường hợp này ta có thể tăng số lần lặp lại bằng cách thay số **6** ở dòng lệnh **DO I = 1, 6** bằng một số lớn hơn và chạy lại chương trình. Việc so sánh kết quả nhận được sau mỗi lần thay đổi dòng lệnh này sẽ giúp ta hiểu rõ hơn ý nghĩa của vòng lặp.

Chú ý: Nói chung Fortran cho phép các biến **bdk, TriDau, TriCuoi, Buoc** nhận kiểu dữ liệu là số nguyên hoặc số thực. Tuy nhiên ta không nên dùng kiểu dữ liệu thực, do số thực được biểu diễn ở dạng gần đúng, có thể gây nên những sai số không lường trước được.

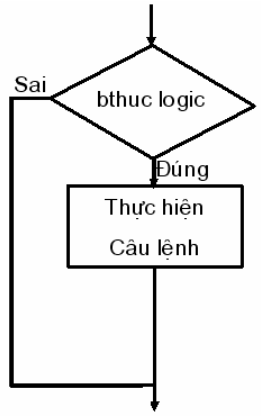
2.2 LỆNH RẼ NHÁNH VỚI IF

Cấu trúc rẽ nhánh là kiểu cấu trúc rất phổ biến đối với các ngôn ngữ lập trình. Trong Fortran, cấu trúc rẽ nhánh được cho khá đa dạng. Sau đây ta sẽ lần lượt xét từng trường hợp.

2.2.1 Dạng 1

```
IF (BThuc_Logic) Câu_lệnh
```

Trong đó **Câu_lệnh** là một câu lệnh thực hiện nào đó và không thể là một trong các câu lệnh có cấu trúc khác, như **IF, DO,**… **BThuc_Logic** là *điều kiện* rẽ nhánh. Tác động của câu lệnh **IF** là, nếu **BThuc_Logic** nhận giá trị **.TRUE.** (đúng) thì chương trình sẽ thực hiện **Câu_lệnh** ngay sau đó, ngược lại, nếu **BThuc_Logic** nhận giá trị **.FALSE.** (sai) thì **Câu_lệnh** sẽ bị bỏ qua và chương trình tiếp tục với những câu lệnh khác sau **IF**. Sơ đồ khối mô tả tác động này được cho trên hình 2.2.



Hình 2.2 Cấu trỳc IF dạng 1

Ví dụ 2.3. Hãy đọc vào một số và cho biết đó là số dương, số âm hay số 0. Chương trình có thể được viết như sau.

```
! Vi du ve lenh re nhanh
REAL X
PRINT '(A\)',' CHO MOT SO:'
READ*, X
IF (X > 0) PRINT *, ' DAY LA SO DUONG'
IF (X < 0) PRINT *, ' DAY LA SO AM'
IF (X == 0) PRINT *, ' DAY LA SO 0'
END
```

Như đã thấy, đối với cấu trúc này, khi **BThuc_Logic** nhận giá trị **.TRUE.** (đúng) thì chỉ có một câu lệnh sau đó được thực hiện.

2.2.2 Dạng 2

```
IF (BThuc_Logic) THEN
   Các_câu_lệnh
END IF
```

Về nguyên tắc, tác động của câu lệnh này hoàn toàn giống với cấu trúc dạng 1 trên đây. Sự khác nhau giữa chúng chỉ là ở chỗ, trong cấu trúc dạng 1, khi *điều kiện* được thỏa mãn (**BThuc_Logic** nhận giá trị **.TRUE.**) thì chỉ có *một* câu lệnh sau **IF** được thực hiện, còn trong trường hợp này, nếu **BThuc_Logic** nhận giá trị **.TRUE.** thì *có thể có nhiều câu lệnh* nằm giữa **IF** … **THEN** và **END IF** sẽ được thực hiện (**Các_câu_lệnh** hàm nghĩa là có thể có nhiều câu lệnh).

Ví dụ 2.4. Viết chương trình nhập vào hai số thực, nếu chúng đồng thời khác 0 thì tính tổng, hiệu, tích, thương của chúng.

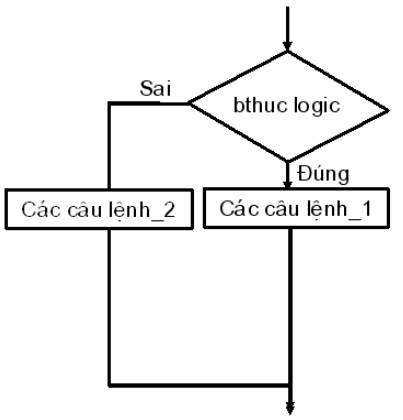
```
REAL X, Y, TONG, HIEU, TICH, THUONG
PRINT*, ‘ CHO 2 SO THUC:’
READ*, X, Y          ! Đọc các số X, Y từ bàn phím
IF (X /= 0.AND.Y /= 0) THEN
                ! X và Y đồng thời khác 0
   TONG = X + Y
   HIEU = X – Y
   TICH = X * Y
   THUONG = X / Y
   PRINT*,’ TONG CUA ’,X,’ VA ‘,Y,’ LA:’,TONG
```

```
   PRINT*,' HIEU CUA ',X,' VA ',Y,' LA:',HIEU
   PRINT*,' TICH CUA ',X,' VA ',Y,' LA:',TICH
   PRINT*,' THUONG CUA ',X,' VA ',Y,' LA:',&
         THUONG
END IF
IF (X == 0.OR.Y == 0) THEN  ! Một trong hai số = 0
  PRINT*,' MOT TRONG HAI SO VUA NHAP = 0'
END IF
END
```

2.2.3 Dạng 3

```
IF (BThuc_Logic) THEN
   Các_câu_lệnh_1
ELSE
   Các_câu_lệnh_2
END IF
```

Khác với hai cấu trúc trên, trong cấu trúc này việc thực hiện chương trình có thể rẽ về một trong hai "nhánh": Nếu **BThuc_Logic** nhận giá trị **.TRUE.** thì chương trình sẽ thực hiện **Các_câu_lệnh_1**, ngược lại, chương trình sẽ thực hiện **Các_câu_lệnh_2**. Sơ đồ khối mô tả tác động của cấu trúc này được cho trên hình 2.3



Hình 2.3 Cấu trỳc IF dạng 3

Ví dụ 2.5. Viết chương trình nhập vào từ bàn phím ba số thực. Nếu ba số đó thỏa mãn điều kiện là ba cạnh của một tam giác thì tính diện tích của tam giác. Ngược lại thì đưa ra thông báo "BA SO NAY KHONG PHAI LA 3 CANH CUA TAM GIAC".

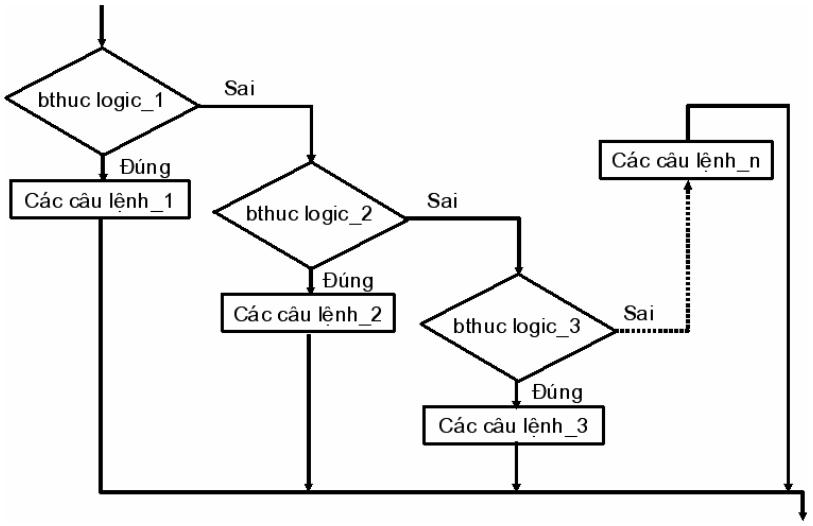
```
PROGRAM TAM_GIAC
REAL A,B,C  ! Ba số sẽ nhập vào
REAL P,S    ! Nửa chu vi và Diện tích
LOGICAL L1
LOGICAL L2
PRINT*, ' CHO 3 SO THUC:'
READ*, A,B,C
L1 = A>0.AND.B>0.AND.C>0  ! Ba số cùng Dương
L2 = A+B>C.AND.B+C>A.AND.C+A>B
      ! Ba số phải thỏa mãn bất đẳng thức tam giác
IF (L1.AND.L2) THEN ! Thỏa mãn điều kiện Tam giác
   P = (A+B+C)/2
   S = SQRT(P*(P-A)*(P-B)*(P-C))
   PRINT*,' DIEN TICH TAM GIAC = ',S
ELSE   ! Không thỏa mãn điều kiện Tam giác
   PRINT*,"BA SO NAY KHONG PHAI LA 3 CANH &
        &CUA TAM GIAC"
END IF
END
```

Trong chương trình này ta đã sử dụng hai biến lôgic **L1, L2** để xác định ba số nhập vào có thỏa mãn điều kiện là ba cạnh của một tam giác hay không. Cách dùng các biến kiểu này rất có ích, vì trong những trường hợp phức tạp nó sẽ giúp ta gỡ rối chương trình được nhanh chóng và chính xác. Hơn nữa, khi viết như vậy chương trình trông sáng sủa hơn.

2.2.4 Dạng 4

```
IF (BThuc_Logic_1) THEN
   Các_câu_lệnh_1
ELSE IF (BThuc_Logic_2) THEN
   Các_câu_lệnh_2
ELSE IF (BThuc_Logic_3) THEN
   Các_câu_lệnh_3
   ...
ELSE
   Các_câu_lệnh_n
END IF
```

Cấu trúc này được gọi là cấu trúc khối **IF** (Block **IF**). Tác động của cấu trúc này được mô tả trên hình 2.4. Trước hết, chương trình sẽ kiểm tra **BThuc_Logic_1**. Nếu **BThuc_Logic_1** nhận giá trị **.TRUE**. thì **Các_câu_lệnh_1** sẽ được thực hiện; nếu **BThuc_Logic_1** nhận giá trị **.FALSE.** thì chương trình sẽ kiểm tra đến **BThuc_Logic_2.** Nếu **BThuc_Logic_2** nhận giá trị **.TRUE.** thì **Các_câu_lệnh_2** sẽ được thực hiện; nếu **BThuc_Logic_2** nhận giá trị .**FALSE**. thì chương trình sẽ kiểm tra **BThuc_Logic_3**,… Quá trình cứ tiếp diễn như vậy cho đến khi nếu tất cả các **BThuc_Logic** đều nhận giá trị **.FALSE.** thì chương trình sẽ thực hiện **Các_câu_lệnh_n.** Nếu **Các_câu_lệnh_*** ở giai đoạn nào đó của quá trình đã được thực hiện, chương trình sẽ thoát khỏi cấu trúc **IF** và chuyển điều khiển đến những câu lệnh ngay sau **END IF**, ngoại trừ trường hợp trong **Các_câu_lệnh_*** có lệnh chuyển điều khiển **GOTO** đến một vị trí khác trong chương trình.



Hình 2.4 Cấu trúc IF dạng 4

Ví dụ 2.6. Viết chương trình nhập điểm trung bình chung học tập (TBCHT) của một sinh viên và cho biết sinh viên đó được xếp loại học tập như thế nào, nếu tiêu chuẩn xếp loại được qui định như

sau: Loại xuất sắc nếu TBCHT >=9; Loại giỏi nếu 9<TBCHT<=8; Loại khá nếu 8<TBCHT<=7; Loại trung bình nếu 7<TBCHT<=5 và loại yếu nếu TBCHT<5.

```
PROGRAM XEPLOAI_1
INTEGER DIEM
WRITE (*,'(A\)') ' CHO DIEM TBCHT: '
READ*, DIEM
IF (DIEM < 0.OR.DIEM > 10) THEN
   PRINT*, ‘ DIEM KHONG HOP LE’
   STOP
  ! Dừng chương trình nếu điểm không hợp lệ
ELSE IF (DIEM >= 9) THEN
   PRINT*,' LOAI XUAT SAC'
ELSE IF (DIEM >= 8) THEN
   PRINT*,' LOAI GIOI'
ELSE IF (DIEM >= 7) THEN
   PRINT*,' LOAI KHA'
ELSE IF (DIEM >= 5) THEN
   PRINT*,' LOAI TRUNG BINH'
ELSE
   PRINT*,' LOAI YEU'
END IF
END
```

Chương trình trên đây có thể viết bằng cách khác như sau.

```
PROGRAM XEPLOAI_2
INTEGER DIEM
WRITE (*,'(A\)') ' CHO DIEM TBCHT: '
READ*, DIEM
IF (DIEM < 0.OR.DIEM < 10) THEN
   PRINT*, ‘ DIEM KHONG HOP LE’
   STOP
END IF
IF (DIEM >= 9) PRINT*,' LOAI XUAT SAC'
IF (DIEM >= 8.AND.DIEM < 9) PRINT*,' LOAI GIOI'
IF (DIEM >= 7.AND.DIEM < 8) PRINT*,' LOAI KHA'
IF (DIEM >= 5.AND.DIEM < 7)PRINT*,' LOAI TR.BINH'
IF (DIEM < 5) PRINT*,' LOAI YEU'
END
```

Trong hai chương trình trên, lệnh **STOP** làm kết thúc (dừng hẳn) chương trình khi giá trị của **DIEM** nhập vào không hợp lệ (**DIEM < 0** hoặc **DIEM > 10**). Câu lệnh **WRITE** hàm chứa trong đó định dạng **FORMAT (‘(A\)’)** có tác dụng giữ cho con trỏ màn hình không nhảy xuống dòng dưới mà nằm ngay sau hằng ký tự **‘CHO DIEM TBCHT: ‘**. Rõ ràng, nếu sử dụng cấu trúc khối **IF** như ở chương trình **XEPLOAI_1**, các biểu thức lôgic sẽ gọn gàng hơn, và chương trình trông sáng sủa hơn so với chương trình **XEPLOAI_2**.

2.2.5 Lệnh nhảy vô điều kiện GOTO

Đây là một trong những câu lệnh được sử dụng khá phổ biến đối với Fortran 77 và các phiên bản trước. Khi lập trình với Fortran 90 lệnh **GOTO** ít được sử dụng do cái gọi là "sự phá vỡ cấu trúc" của nó. Mặc dù vậy, câu lệnh này giúp người lập trình cảm thấy nhẹ nhàng khi gặp phải những tình huống khó xử.

Cú pháp câu lệnh **GOTO** có dạng sau:

```
GOTO m
```

Trong đó **m** là nhãn của một câu lệnh nào đó sẽ được chuyển điều khiển tới trong chương trình. Khi gặp lệnh **GOTO**, ngay lập tức chương trình sẽ chuyển điều khiển tới câu lệnh có nhãn **m**. Nếu trong chương trình không có câu lệnh nào có nhãn **m** thì lỗi sẽ xuất hiện. Hơn nữa, câu lệnh sẽ được chuyển điều khiển tới (câu lệnh có nhãn **m**) không được phép nằm trong vòng kiểm soát của lệnh chu trình **DO** và cấu trúc rẽ nhánh **IF**. Chẳng hạn, những trường hợp sau đây là *không được phép*:

```
...
GOTO 123
...
DO I = 1, N
  ...
  123 X = X + Y
  ...
END DO
```

Hoặc:

```
...
GOTO 456
...
IF (L1.AND.L2) THEN
  ...
  456 A = B * C
  ...
END IF
```

Nhưng có thể dùng lệnh **GOTO** để thoát khỏi một chu trình lặp hoặc một cấu trúc rẽ nhánh nào đó, chẳng hạn:

```
DO I = 1, N
  ...
  GOTO 123
  ...
END DO
123 X = X + Y
```

Hoặc

```
IF (L1.AND.L2) THEN
  ...
  GOTO 456
  ...
```

```
END IF
456 A = B * C
```

Sau đây là một ví dụ minh họa tác động của lệnh **GOTO**. Cho giá trị của các biến lôgic **L1** và **L2**. Nếu **L1=.TRUE.** thì gán **I=1, J=2;** nếu **L1=.FALSE.** còn **L2=.TRUE.** thì gán **I=2, J=3;** nếu cả **L1** và **L2** đều nhận giá trị **.FALSE.** thì gán **I=3, J=4.** Khi đó, đoạn chương trình:

```
IF (L1) THEN
  I = 1
  J = 2
ELSE IF (L2) THEN
  I = 2
  J = 3
ELSE
  I = 3
  J = 4
END IF
```

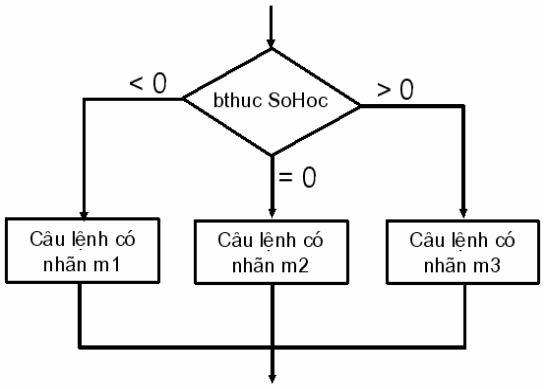
có thể được thay thế bởi đoạn chương trình sau nếu sử dụng lệnh **GOTO**

```
IF (.NOT.L1) GOTO 10
   I = 1
   J = 2
   GOTO 30
10 IF (.NOT.L2) GOTO 20
   I = 2
   J = 3
   GOTO 30
20 I = 3
   J = 4
30 CONTINUE
```

2.2.6 Lệnh IF số học

IF (BThuc_SoHoc) m1, m2, m3

Trong đó **BThuc_SoHoc** là một biểu thức số học, có thể có kiểu nguyên hoặc thực; **m1, m2, m3** là nhãn của các câu lệnh có trong chương trình. Người ta gọi cấu trúc **IF** này là ***IF số học***, vì quyết định rẽ nhánh phụ thuộc vào *dấu* của **BThuc_SoHoc**. Tác động của cấu trúc này được mô tả trên hình 2.5.



Hình 2.5 Cấu trúc IF số học

Trước hết chương trình sẽ tính giá trị của **BThuc_SoHoc**. Nếu **BThuc_SoHoc** nhận giá trị âm, chương trình sẽ chuyển điều khiển tới câu lệnh có nhãn **m1**; nếu **BThuc_SoHoc** nhận giá trị bằng 0, chương trình sẽ chuyển điều khiển tới câu lệnh có nhãn **m2**; nếu **BThuc_SoHoc** nhận giá trị dương, điều khiển sẽ được chuyển tới câu

lệnh có nhãn **m3.** Hai trong ba nhãn **m1, m2, m3** có thể trùng nhau, có nghĩa là hai nhánh của điều khiển có thể chuyển đến cùng một câu lệnh. Tuy nhiên các câu lệnh có nhãn **m1, m2, m3** không được phép nằm trong vòng kiểm soát của lệnh chu trình **DO** và cấu trúc rẽ nhánh khác. Cũng như lệnh **GOTO**, lệnh **IF** số học cũng ít được sử dụng khi lập trình với Fortran 90.

Ví dụ 2.7. Nhập vào một số nguyên và xác định xem số đó nhỏ hơn, lớn hơn hay bằng 50. Ta có chương trình sau.

```
INTEGER N
PRINT*, ' CHO MOT SO NGUYEN '
READ*, N
IF (N–50) 10, 20, 30
10 PRINT*,' SO NAY NHO HON 50'
  GOTO 40
20 PRINT*,' SO NAY BANG 50'
  GOTO 40
30 PRINT*,' SO NAY LON HON 50'
40 CONTINUE
END
```

Nếu ta chỉ quan tâm đến việc số nhập vào có lớn hơn 50 hay không, thì cấu trúc rẽ nhánh chỉ cần chuyển điều khiển đến hai nhánh. Khi đó chương trình được viết lại thành:

```
INTEGER N
PRINT*, ' CHO MOT SO NGUYEN '
READ*, N
IF (N–50) 10, 10, 30
10 PRINT*,' SO NAY NHO HON HOAC BANG 50'
  GOTO 40
30 PRINT*,' SO NAY LON HON 50'
40 CONTINUE
END
```

Các cấu trúc **IF** dạng 2, 3, 4 và **IF** số học cũng không thể là câu lệnh kết thúc của lệnh chu trình **DO**.

Ví dụ 2.8. Giả sử ta cần liệt kê tất cả các số nguyên chia hết cho 13 trong phạm vi từ **N1** đến **N2** với **N1** và **N2** được nhập từ bàn phím. Khi đó chương trình có thể viết như sau nếu sử dụng lệnh chu trình **DO** dạng 2 hoặc dạng 3 mà *không thể* sử dụng dạng 1:

```
Program Cach_1 ! Dung Chu trinh DO dang 2
write(*,'(A\)')' Cho hai so nguyen N1 va N2: '
read*, N1, N2
if (N1 > N2 .OR. N2 < 13) then
  Print*, ' So lieu khong hop le &
     & hoac khong co so nao chia het cho 13'
  Stop
end if
Do 10 i=N1,N2
```

```
  if (mod(i,13)==0) then
    print*, i
  end if
10 Continue
End
```

Hoặc

```
Program Cach_2 ! Dung Chu trinh DO dang 3
write(*,'(A\)')' Cho hai so nguyen N1 va N2: '
read*, N1, N2
if (N1 > N2 .OR. N2 < 13) then
  Print*, ' So lieu khong hop le &
      & hoac khong co so nao chia het cho 13'
  Stop
end if
Do i=N1,N2
  if (mod(i,13)==0) then
    print*, i
  end if
enddo
End
```

2.3 KẾT HỢP DO VÀ IF

Như đã thấy ở ví dụ 2.8, chu trình **DO** có thể bao hàm cả cấu trúc rẽ nhánh **IF** và ngược lại. Nghĩa là chu trình **DO** có thể kiểm soát toàn bộ cấu trúc **IF**, hoặc trong cấu trúc **IF** có chứa trọn vẹn chu trình **DO**. Nhất thiết chúng không được phép giao nhau. Cú pháp tổng quát của các cấu trúc này như sau.

Dạng 1: Cấu trúc **IF** nằm trong chu trình **DO**:

```
DO bdk = TriDau, TriCuoi, Buoc
  ...
  IF (BThuc_Logic) THEN
    ...
  END IF
  ...
END DO
```

Dạng 2: Chu trình **DO** nằm trong cấu trúc **IF**:

```
IF (BThuc_Logic) THEN
  ...
  DO bdk = TriDau, TriCuoi, Buoc
    ...
  END DO
  ...
END IF
```

Ví dụ 2.9. Chương trình sau đây sẽ kết thúc sau 20 lần lặp, mặc dù số lần lặp được qui định bởi lệnh chu trình là **10000**:

```
PROGRAM IFinDO  ! C.trúc IF nằm trong C.trình DO
Do i=1,10000
  If (mod(i,2)==0) write(*,*) i
  If (i == 20) then
    print*,' Ket thuc sau lan lap thu ',i
    stop
  End If
Enddo
END
```

Ví dụ 2.10. Đọc vào hai số nguyên **M** và **N**. Nếu **M** ≤ **N** thì in ra lần lượt các số từ **M** đến **N**, ngược lại thì in thông báo **M** > **N**.

```
PROGRAM DOinIF  ! C.trình DO nằm trong C.trúc IF
Print*,' Cho hai so nguyen: '
Read*, M, N
if (M <= N) then
  write(*,*)' Lap tu ',M, ' den ',N
  Do i=M,N
    write(*,*) i
  Enddo
Else
  write(*,*)' M > N'
End If
END
```

2.4 RẼ NHÁNH VỚI CẤU TRÚC SELECT CASE

Một trong những phương pháp hữu hiệu để chuyển điều khiển trong chương trình là sử dụng cấu trúc rẽ nhánh **SELECT CASE.** Dạng tổng quát của cấu trúc này như sau.

```
SELECT CASE (BThuc_Chon)
    CASE (Chon1)
       Các_câu_lệnh_1
    CASE (Chon2)
       Các_câu_lệnh_2
    ...
    CASE DEFAULT
       Các_câu_lệnh_n
END SELECT
```

Trong đó ***BThuc_Chon, Chon1, Chon2,…*** phải có cùng kiểu dữ liệu số nguyên, lôgic hoặc **CHARACTER*1**. ***BThuc_Chon*** là biểu thức được tính toán, nó còn được gọi là *chỉ số chọn*. ***Chon1, Chon2,…*** là các giá trị hoặc khoảng giá trị *có thể có* của ***BThuc_Chon***. Nếu có nhiều giá trị rời rạc, chúng phải được liệt kê cách nhau bởi các dấu phẩy; nếu là khoảng giá trị liên tiếp, chúng phải được biểu diễn bởi hai giá trị đầu và cuối khoảng, phân cách nhau bằng dấu hai chấm (:). ***Các_câu_lệnh_1, Các_câu_lệnh_2,…*** là tập các câu lệnh thực hiện. Nếu biểu diễn sơ đồ khối cấu trúc này nó sẽ gần giống với hình 2.4, trong đó các ***BThuc_Logic_1***,… được thay bởi mệnh đề *nếu* ***BThuc_Chon*** *thuộc tập* ***Chon1***,…

Tác động của cấu trúc này có thể mô tả như sau.

Bắt đầu: Xác định giá trị của (***Bieu_Thuc_Chon***)

Nếu giá trị của ***Bieu_Thuc_Chon*** thuộc tập (***Chon1***) thì

Thực hiện ***Các_câu_lệnh_1***

Nếu giá trị của ***Bieu_Thuc_Chon*** thuộc tập (***Chon2***) thì

Thực hiện ***Các_câu_lệnh_2***

...

Nếu ***Bieu_Thuc_Chon*** nhận các giá trị khác thì

Thực hiện ***Các_câu_lệnh_n***

Kết thúc

Ví dụ 2.11. Viết chương trình xem số ngày của một tháng nào đó trong năm.

```
INTEGER Month, Year
Print'(A\)',' Xem so ngay cua thang nao?'
Read*, Month
SELECT CASE (Month)
      CASE (1,3,5,7,8,10,12) ! Các tháng có 31 ngày
            Print*,' Thang ', Month,' co 31 ngay'
      CASE (4,6,9,11)   ! Các tháng có 30 ngày
            Print*,' Thang ', Month,' co 30 ngay'
      CASE (2)  ! Có 28 hoặc 29 ngày
            Print'(A\)',' Nam nao?'
            Read*, Year
            IF (Mod(Year,4).EQ.0.AND.  &
             & Mod(Year,100).NE.0.OR. &
             & Mod(Year,400).EQ.0) then  ! Năm nhuận
                  Print*,' Thang ', Month,' Nam ',&
                    Year, ' co 29 ngay'
            Else   ! Năm bình thường
                  Print*,' Thang ', Month,' Nam ',&
                    Year, ' co 28 ngay'
            End IF
      CASE DEFAULT
            Print*,' Khong co thang ', Month
END SELECT
END
```

Trong ví dụ 2.11, giá trị của ***Bieu_Thuc_Chon*** được xác định bởi lệnh **READ*, Month**, tức ***Month*** là ***Bieu_Thuc_Chon***. Vì ***Chon1, Chon2*** nhận các tập giá trị rời rạc (các tháng không liên tục) nên chúng được liệt kê cách nhau bởi các dấu phẩy.

Ví dụ 2.12. Gõ một ký tự và cho biết đó là chữ cái hay chữ số.

```
CHARACTER*1  char
Print*,' Hay go mot ky tu:'
Read*, Char
  SELECT CASE (char)
```

```
    CASE ('0':'9')
      WRITE (*, *) "Day la chu so ", Char
    CASE ('A':'Z','a':'z')
      WRITE (*, *) "Day la chu cai ",Char
    CASE DEFAULT
      WRITE (*, *) "Day khong phai chu so, &
            &cung khong phai chu cai."
      WRITE (*, *) "Day la ky tu ", Char
  END SELECT
END
```

Ở đây, tập các giá trị của ***Chon1, Chon2*** là những dãy giá trị liên tục trong các khoảng nên chúng được liệt kê bằng cách nối các giá trị đầu khoảng và cuối khoảng bởi dấu hai chấm (:).

2.5 THAO TÁC VỚI HẰNG VÀ BIẾN KÝ TỰ (CHARACTER)

Ở mục 1.4.2 ta đã xét kiểu dữ liệu ký tự và cách khai báo các biến, hằng có kiểu ký tự. Hằng ký tự là tập hợp các ký tự thuộc bảng mã ASCII, *không bao gồm* các ký tự điều khiển, lập thành một dãy đặt trong cặp dấu nháy đơn (‘ ’) hoặc dấu nháy kép (“ ”). Biến ký tự là biến có kiểu ký tự, được khai báo bởi lệnh **CHARACTER**. Các hằng và biến ký tự có thể được gộp với nhau để tạo thành một xâu ký tự mới.

Ví dụ 2.13. Trong chương trình làm quen ở ví dụ 1.1 ta đã gặp cách thao tác với hằng và biến ký tự nhưng chưa phân tích gì về chúng. Ta hãy trở lại với ví dụ đó. Chương trình sau đây sẽ đưa ra lời chào mừng nếu ta gõ tên mình vào khi được hỏi.

```
Program WelCome
CHARACTER *20 Name
Print *,'Ten ban la gi ?'
Read*, Name
Write(*,*) 'Xin chao ban ', Name
END
```

Trong chương trình này, lệnh **PRINT** in ra một hằng ký tự (***Ten ban la gi ?***), lệnh **READ*** đọc giá trị của biến ký tự ***Name*** do ta nhập vào, còn lệnh **WRITE** in ra một hằng ký tự (***Xin chao ban***) và giá trị của biến ký tự ***Name*** tiếp theo đó. Biến ***Name*** đã được khai báo có kiểu ký tự với độ dài cực đại là 20. Khi chạy chương trình này, nếu ta nhập một xâu có chứa dấu cách ở giữa (ví dụ ***Hoang Nam***) mà không đặt trong cặp dấu nháy, thì giá trị của biến ***Name*** có thể sẽ bị cắt bỏ phần bên phải nhất kể từ vị trí dấu cách. Chẳng hạn khi chạy lại chương trình:

```
Ten ban la gi ?
Hoang Nam
Xin chao ban Hoang
```

(chứ không phải ***Hoang Nam*** như ta mong muốn). Nhưng nếu ta gõ vào “***Hoang Nam***” (đặt trong cặp dấu nháy) thì kết quả nhận được lại hoàn toàn bình thường. Nếu thay câu lệnh

```
Read*, Name
```

bởi câu lệnh

```
Read (*,’(A)’) Name
```

thì giá trị của biến ***Name*** khi nhập vào không được đặt trong dấu nháy, vì trong trường hợp này các dấu nháy sẽ được hiểu là một bộ phận của ***Name***. Ví dụ, giả sử ta đã thay câu lệnh **READ*** như trên và chạy lại chương trình:

Ten ban la gi ?
Hoang Nam
Xin chao ban *Hoang Nam*

nhưng

Ten ban la gi ?
'Hoang Nam'
Xin chao ban *'Hoang Nam'*

Nếu độ dài xâu ký tự vượt quá độ dài khai báo cực đại của biến ký tự thì phần bên phải nhất của xâu sẽ bị cắt bỏ. Ví dụ:

```
CHARACTER *7 Name
Name = 'Hoang Nam'
```

Kết quả biến ***Name*** sẽ có giá trị là '**Hoang N**', vì độ dài cực đại của ***Name*** là 7 (ký tự), không đủ để chứa nội dung của giá trị **'Hoang Nam'.**

Ví dụ 2.14. Chương trình sau đây là một ví dụ về sử dụng phép toán *gộp* các xâu ký tự.

```
Character Ho*7,Dem*7, Ten*7, HoTen*21
Ho='Nguyen'
Dem='Van'
Ten='Thanh'
HoTen=Ho//Dem//Ten
PRINT*,'123456712345671234567'
PRINT*, HoTen
END
```

Khi chạy chương trình này ta sẽ nhận được:

```
12345671234567123456 7
Nguyen Van    Thanh
```

Dòng đầu tiên của kết quả viết một dãy các chữ số chỉ nhằm mục đích để ta đối sánh độ dài các xâu một cách dễ dàng. Biến ***HoTen*** có giá trị bằng giá trị của ba biến ***Ho, Dem*** và ***Ten***. Vì ba biến này đều có độ dài khai báo là 7 (ký tự), nên giữa ***Van*** và ***Thanh*** có 4 khoảng trống (4 dấu cách).

BÀI TẬP CHƯƠNG 2

2.1 Viết chương trình hiển thị các số nguyên trong khoảng từ 10 đến 20 và căn bậc hai tương ứng của chúng.

2.2 Viết chương trình tính tổng của các số nguyên liên tiếp trong khoảng từ 101 đến 1000 và hiển thị kết quả lên màn hình dưới dạng:

TONG CAC SO NGUYEN TU 101 DEN 1000 LA: XXXXXX

2.3 Viết chương trình tính tổng của các số nguyên chẵn liên tiếp trong khoảng từ 2 đến 1000 và hiển thị kết quả lên màn hình dưới dạng:

TONG CAC SO CHAN TU 2 DEN 1000 LA: XXXXXX

2.3 Kết quả điểm thi 10 môn học của một sinh viên được ghi trong file DIEM.TXT. Viết chương trình đọc điểm thi của từng môn, tính điểm trung bình của các môn và hiển thị lên màn hình điểm các môn và điểm trung bình của sinh viên đó.

2.4 Kết quả điểm thi 10 môn học của một sinh viên được ghi trong file DIEM.TXT. Kết quả thi được xem là đạt yêu cầu nếu điểm thi lớn hơn hoặc bằng 5. Viết chương trình đọc điểm thi của từng môn và cho biết số môn thi đạt yêu cầu của sinh viên đó.

2.5 Viết chương trình nhập vào toạ độ ba điểm $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$. Nếu ba điểm không thẳng hàng thì tính diện tích tam giác ABC, ngược lại thì đưa ra thông báo "**Ba diem A, B, C thang hang**".

2.6 Viết chương trình nhập vào toạ độ ba điểm $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$. Nếu ba điểm không thẳng hàng thì xác định toạ độ trọng tâm của tam giác ABC, ngược lại thì đưa ra thông báo "**Ba diem A, B, C thang hang**".

2.7 Viết chương trình nhập vào toạ độ ba điểm $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$. Nếu ba điểm không thẳng hàng thì tính độ dài các đường trung tuyến của tam giác ABC, ngược lại thì đưa ra thông báo "**Ba diem A, B, C thang hang**".

2.8 Viết chương trình nhập vào 3 số thực a, b, c thoả mãn điều kiện là ba cạnh của một tam giác rồi tính diện tích, các đường cao và bán kính đường tròn ngoại tiếp của tam giác đó. **Gợi ý:** Bán kính đường tròn ngoại tiếp $R=(abc)/(4S)$, với S là diện tích.

2.9 Viết chương trình nhập vào các hệ số a, b và: 1) Giải phương trình $ax + b = 0$; 2) Giải bất phương trình $ax + b > 0$.

2.10 Viết chương trình nhập vào các hệ số a_1, b_1, c_1, a_2, b_2, c_2 và giải hệ phương trình: $\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases}$

2.11 Viết chương trình nhập vào các số thực a, b, c và giải bất phương trình $ax^2 + bx + c > 0$

2.12 Tìm số nguyên dương *n* lớn nhất thoả mãn điều kiện: a) $3n^3 - 212n<10$; b) $123n^{1/2} - 3n + 4 \le 54$; $e^n - 1999\lg n < 6$.

2.13 Viết chương trình nhập vào *n* số thực và cho biết có bao nhiêu số dương, âm và bằng 0.

2.14 Điểm thi học kỳ của một lớp sinh viên được cho trong file DIEM_HK.TXT. Cấu trúc file được mô tả như sau: Dòng 1 gồm 2 số nguyên dương, chỉ số lượng sinh viên của lớp (N) và số môn học (M); Dòng 2 gồm M số nguyên dương chỉ số đơn vị học trình của M môn học; N dòng tiếp theo, mỗi dòng gồm M+1 số, số thứ nhất là một số nguyên dương, chỉ mã số của từng sinh viên, M số thực

tiếp theo tương ứng là điểm thi của M môn học (điểm thi có giá trị trong khoảng 0−10). Viết chương trình đọc số liệu trong file và tính điểm trung bình chung học tập của từng sinh viên theo công thức:

$$\text{Điểm trung bình chung} = \frac{\sum_{i=1}^{n}(\text{Điểm môn i}) x (\text{Số học trình môn i})}{\sum_{i=1}^{n}(\text{Số học trình môn i})}$$

In kết quả lên màn hình thành hai cột, tương ứng là mã số sinh viên và điểm trung bình chung học tập của sinh viên đó.

2.15 Phát triển bài tập 2.14 bằng cách, tiến hành xếp loại học tập cho sinh viên dựa vào điểm trung bình chung học tập (TBCHT) như sau:

− Nếu TBCHT < 5.0: Loại yếu (YEU)

− Nếu 5.0 ≤ TBCHT < 7.0: Loại trung bình (TRUNG BINH)

− Nếu 7.0 ≤ TBCHT < 8.5: Loại khá (KHA)

− Nếu 8.5 ≤ TBCHT < 9.0: Loại giỏi (GIOI)

− Nếu TBCHT ≥ 9.0: Loại xuất sắc (XUAT SAC)

In kết quả lên màn hình thành ba cột: cột 1 là mã số sinh viên, cột 2 là điểm trung bình chung học tập, và cột 3 là kết quả xếp loại.

2.16. Viết chương trình nhập vào số lượng môn học (M), số học trình của từng môn, họ tên và điểm thi của M môn học của N sinh viên rồi tính điểm trung bình chung học tập, xếp loại học tập theo cách thức tính và xếp loại ở các bài tập 2.14 và 2.15. In kết quả vào file KETQUA.TXT dưới dạng:

HO VA TEN DIEM TBCHT XEP LOAI

Nguyen Van A 8.7 Xuat sac

....

CHƯƠNG 3. CÁC CẤU TRÚC MỞ RỘNG

3.1 CHU TRÌNH DO TỔNG QUÁT VÀ CHU TRÌNH DO LỒNG NHAU

Trong chương 2 ta đã xét 3 dạng chu trình **DO**, trong đó dạng 1 và dạng 2 đòi hỏi phải sử dụng các dòng lệnh có nhãn để kết thúc chu trình. Điều đó làm cho ta nhiều lúc phải nhớ một cách máy móc hệ thống các nhãn này, nhất là khi chương trình có nhiều vòng lặp hoặc khi vòng lặp đòi hỏi phải kiểm soát một đoạn chương trình dài. Còn đối với cấu trúc dạng 3, nếu trong chương trình có chứa nhiều vòng lặp lồng nhau sẽ làm cho ta lúng túng khi cần phân biệt mỗi vòng lặp bắt đầu và kết thúc ở đâu. Sự bất tiện đó sẽ tăng lên khi chương trình đang có lỗi và ta đang phải gỡ rối. Để khắc phục nhược điểm này, Fortran 90 cho phép sử dụng các chu trình **DO** tổng quát, trong đó mỗi vòng lặp sẽ được gán tên, tương tự như nhãn, nhưng vì tên được đặt gắn với lệnh chu trình nên giúp ta dễ nhớ và dễ kiểm soát hơn. Cú pháp câu lệnh chu trình tổng quát như sau.

```
Ten_ChuTrinh: DO bdk = TriDau, TrCuoi [, Buoc]
   Các_câu_lệnh
END DO Ten_ChuTrinh
```

Về nguyên tắc, tác động của chu trình **DO** này hoàn toàn giống với các chu trình **DO** trước đây.

Ngoài ra, tất cả các dạng chu trình **DO** đều có thể lồng nhau sao cho chu trình ngoài kiểm soát toàn bộ chu trình trong. Có thể có các cấu trúc lồng nhau sau đây.

Dạng 1:

```
DO m1 bdk1= ...
   ...
   DO m2 bdk2=...
      Các_câu_lệnh
  m2 Câu_lệnh_kết_thúc
  [ hoặc: m2 CONTINUE ]
     ...
m1 Câu_lệnh_kết_thúc
[ hoặc: m1 CONTINUE ]
```

Dạng 2:

```
DO bdk1= ...
    ...
    DO bdk2=...
      Các_câu_lệnh
    END DO
    ...
END DO
```

Dạng 3:

```
ChuTrinh_1: DO bdk1= ...
    ...
    ChuTrinh_2: DO bdk2=...
      Các_câu_lệnh
    END DO ChuTrinh_2
    ...
END DO ChuTrinh_1
```

Dang 1 và dạng 2 là các dạng chu trình lồng nhau khi sử dụng cấu trúc **DO** ở chương 2. Dạng 3 là chu trình lồng nhau sử dụng cấu trúc **DO** tổng quát. Trong các cấu trúc trên, ***Các_câu_lệnh*** cũng có thể là các chu trình **DO** khác. Nghĩa là, về nguyên tắc ta có thể sử dụng cấu trúc có *nhiều hơn hai* chu trình lồng nhau.

Ví dụ 3.1. Lập chương trình tính tổng điểm thi đại học cho các thí sinh.

```
PROGRAM TinhDiem
WRITE(*,'(A\)')" Cho so thi sinh can tinh:"
Read*, N
ThiSinh: DO i=1,N
        TongDiem=0.0
        MonThi: DO j=1,3
                Print*, "Cho diem thi mon ", J,&
       &" cua TS thu ", I
                Read*,Diem
                TongDiem=TongDiem+Diem
    END DO MonThi
    Write(*,'(" Diem TS ",I3,"=",F5.1)')&
                I,TongDiem
END DO ThiSinh
END
```

Trong chương trình trên ta đã sử dụng hai chu trình lồng nhau dạng tổng quát, trong đó chu trình ngoài được đặt tên là ***ThiSinh***, vì nó lặp theo số thí sinh cần phải tính, còn chu trình trong được đặt tên là ***MonThi***, hàm ý là sẽ lặp theo số lượng môn thi. Cách đặt tên như vậy sẽ mang tính gợi nhớ.

3.2 CẤU TRÚC IF TỔNG QUÁT VÀ CẤU TRÚC IF LỒNG NHAU

Tương tự như cấu trúc chu trình **DO** đã nói ở mục 3.1, để giảm bớt "sức ép" vì phải nhớ máy móc trong lúc lập trình, Fortran 90 cũng đưa vào *tên* của cấu trúc rẽ nhánh **IF** và gọi là cấu trúc **IF** tổng quát. Cú pháp như sau.

```
Ten_Cau_Truc: IF (BThuc_Logic) THEN
    ...
END IF Ten_Cau_Truc
```

Hoặc

```
Ten_Cau_Truc: IF (BThuc_Logic) THEN
    ...
ELSE Ten_Cau_Truc
    ...
END IF Ten_Cau_Truc
```

Hoặc

***Ten_Cau_Truc*: IF (*BThuc_Logic_1*) THEN**
...
ELSE IF (*BThuc_Logic_2*) THEN
.....
ELSE IF (*BThuc_Logic_3*) THEN
...
ELSE *Ten_Cau_Truc*
...
END IF *Ten_Cau_Truc*

Nói chung không có gì khác biệt về chức năng giữa cấu trúc **IF** tổng quát và cấu trúc **IF** thông thường đã xét ở chương 2, ngoại trừ thêm ***Ten_Cau_Truc*** để "đánh dấu" xác định vị trí của khối cấu trúc.

Cấu trúc **IF** cũng có thể lồng nhau sao cho cấu trúc này nằm trọn vẹn trong cấu trúc kia.

IF (*BThuc_Logic_1*) THEN
...
IF (*BThuc_Logic_2*) THEN
...
END IF
...
END IF

Hoặc

***Ngoai*: IF (*BThuc–Logic_1*) THEN**
...
***Trong*: IF (*BThuc_Logic_2*) THEN**
...
END IF *Trong*
...
END IF *Ngoai*

Trong đó ***Trong*** và ***Ngoai*** tương ứng là tên các cấu trúc **IF**.

Ví dụ 3.2. Để minh họa cho cách sử dụng cấu trúc **IF** lồng nhau, sau đây sẽ đưa ra một phương án viết chương trình giải phương trình $ax^2 + bx + c = 0$ với cấu trúc **IF** tổng quát.

Ta thấy, đây là biểu thức tổng quát của một phương trình đa thức có bậc cao nhất bằng 2. Tuy nhiên, phụ thuộc vào giá trị của các hệ số *a, b, c* mà phương trình này có thể có bậc là 2, 1 hoặc 0. Do đó, để giải bài toán này trước hết ta lập một dàn bài thực hiện, gồm các bước sau.

Bước 1: Nhập các hệ số a, b, c

Bước 2: Nếu a=0: (giải phương trình bậc nhất bx + c = 0)

- Nếu b=0: (bậc của phương trình bằng 0)
  + Nếu c=0: Trả lời: Phương trình có *vô số nghiệm*
  + Nếu c≠0: Trả lời: Phương trình *vô nghiệm*

- Nếu b≠0: Trả lời: Nghiệm x=–c/b

Bước 3: Nếu a≠0: (giải phương trình bậc hai $ax^2 + bx + c = 0$)

- Tính DelTa=b*b–4*a*c
- Nếu DelTa<0: Trả lời: Vô nghiệm (hoặc nghiệm ảo)
- Nếu DelTa≥0:

  + Tính các nghiệm

  + Trả lời: Nghiệm x1, x2 =(–b±(DelTa)$^{0.5}$)/(2a)

Bước 4: Kết thúc

Dựa trên dàn bài này ta có mã chương trình như sau:

```
PROGRAM GiaiPTb2
REAL a, b, c, DelTa, x1, x2
Print*,' Cho cac he so a,b,c:'
Read*, a,b,c
XetA: IF (a==0) THEN
  XetB: IF (b==0) THEN
    XetC: IF (c==0) THEN
      Print*,' Phuong Trinh co VO SO NGHIEM'
    ELSE XetC
      Print*,' Phuong Trinh VO NGHIEM'
    END IF XetC
  ELSE XetB
    Print*, 'Ph.trinh co 1 nghiem x=', -c/b
  END IF XetB
ELSE XetA
  DelTa=b*b-4*a*c
  XetDelTa: IF (DelTa<0) THEN
                  Print*,'Phuong trinh KHONG CO NGHIEM THUC'
  ELSE XetDelTA
    DelTa=SQRT(DelTa)
    X1=( -b - DelTa) / (2*a)
    X2=( -b + DelTa) / (2*a)
    Print*,'PT co cac nghiem X1=',X1, &
      ' X2=', X2
  END IF XetDelTa
END IF XetA
END
```

Qua đó nhận thấy rằng, các cấu trúc **IF** trên đây có thể lồng nhau nhiều cấp. Hơn nữa, khi sử dụng cấu trúc **IF** tổng quát, thông qua hệ thống tên của cấu trúc, ta có thể kiểm soát được chương trình một cách dễ dàng.

3.3 CHU TRÌNH NGẦM

Trước hết ta hãy xét hai ví dụ sau, trong đó chúng đều thực hiện việc in lên màn hình 5 số nguyên.

Ví dụ 3.3.1: In 5 số trên năm dòng khác nhau

```
DO I = 1, 5
   PRINT*, I
END DO
END
```

Nếu chạy chương trình này bạn sẽ nhận được kết quả trên màn hình là:

```
1
2
3
4
5
```

Ví dụ 3.3.2: In 5 số trên cùng một dòng

```
PRINT*, (I, I = 1, 5)
END
```

Trong trường hợp này bạn sẽ nhận được kết quả là:

```
  1   2   3   4   5
```

Lệnh **PRINT*** trong ví dụ 3.3.2 cho phép in 5 giá trị của **I**, với **I** tăng dần từ 1 đến 5. Khác với ví dụ 3.3.1, trong đó lệnh **PRINT*** được thực hiện 5 lần, mỗi lần in một bản ghi, nên kết quả nhận được là mỗi số in trên một dòng, trong ví dụ 3.3.2 lệnh **PRINT*** chỉ được thực hiện một lần, tức là chỉ in một bản ghi, nên các giá trị đều nằm trên một dòng. Người ta gọi vòng lặp in các giá trị của **I** trong lệnh **PRINT*** ở ví dụ 3.3.2 là chu trình **DO** ngầm. Loại chu trình **DO** này được sử dụng rất nhiều, nhất là trong việc kết xuất dữ liệu và đọc dữ liệu từ file TEXT. Ví dụ, chương trình sau đây cho phép in 100 số nguyên dương đầu tiên theo thứ tự tăng dần trên 10 dòng, mỗi dòng 10 số.

```
PROGRAM BangSoNguyen
DO I=1,91,10
  PRINT '(10I4)',(j,j=i,i+9)
ENDDO
END
```

3.4 ĐỊNH DẠNG DỮ LIỆU BẰNG LỆNH FORMAT

Trong các chương trước ta đã gặp một số trường hợp sử dụng lệnh định dạng **FORMAT** để đọc vào hoặc kết xuất dữ liệu có qui cách. Tuy nhiên đó mới chỉ là một vài ví dụ đơn giản. Trong mục này ta sẽ đề cập chi tiết hơn về câu lệnh này. Cú pháp câu lệnh như sau.

```
m FORMAT (Mô_tả_định_dạng)
```

Trong đó **m** là nhãn câu lệnh, ***Mô_tả_định_dạng*** là những qui ước để đọc/ghi dữ liệu theo qui tắc nhất định. Fortran định nghĩa khá nhiều qui tắc định dạng, có những định dạng áp dụng cho cả đọc và ghi dữ liệu, nhưng cũng có những định dạng chỉ áp dụng cho đọc hoặc ghi dữ liệu. Bảng 3.1 dẫn ra những qui tắc định dạng được sử dụng phổ biến nhất, trong đó cột 1 mô tả ký hiệu định dạng có thể xuất hiện trong lệnh **FORMAT**, cột 2 mô tả ý nghĩa sử dụng khi nhập hoặc kết xuất dữ liệu, cột 3 đưa ra một số ví dụ đơn giản khi viết qui tắc định dạng trong lệnh **FORMAT**. Bạn đọc có thể tìm hiểu kỹ hơn qua các tài liệu tham khảo hoặc tra cứu chức năng trợ giúp của Fortran.

Ví dụ 3.4:

```
INTEGER N, M
REAL X, Y, Z
PRINT 10, ' Cho hai so nguyen: '
10 FORMAT (A\) ! Viet xong, giu con tro tren cung dong
READ (*, *) N, M
WRITE (*, '(A\)')' Cho ba so thuc: '
READ (*, *) X, Y, Z
WRITE (*, 20) N, M, X, Y, Z
20 FORMAT (20X, 'Cac so vua nhap la : '/2x,&
      &' Cac so nguyen: ',' N=', I6,&
      &' M=',I6/2x,' Cac so thuc: ',2x,&
      &' X=',F6.1,' Y=',F6.1,' Z=', F6.1)
END
```

Bảng 3.1 Qui cách mô tả định dạng FORMAT

Mô tả	Ý nghĩa	Ví dụ
Iw[.m]	Đọc/in một số nguyên	I5, I5.5, 4I6
Bw[.m]	Đọc/in một số nhị phân	B4
Ow[.m]	Đọc/in một số cơ số 8	O5
Zw[.m]	Đọc/in một số cơ số 16	Z10.3
Fw.d	Đọc/in một số thực dấu phẩy tĩnh	F10.3, F5.0, 5F7.2
Ew.d	Đọc/in một số thực dấu phẩy động	E14.7, 5E10.6
Dw.d	Đọc/in một số thực độ chính xác gấp đôi	D14.7, 5D10.6
A[w]	Đọc/in một biến ký tự	A, A1, A20, 4A7
Lw	Đọc/in một biến lôgic	L4, 3L5
nX	Bỏ qua n ký tự	1X, 5X
Tc	Chuyển con trỏ đến vị trí thứ c tính từ vị trí đầu tiên của bản ghi	T10
TLc	Chuyển con trỏ sang trái c ký tự tính từ vị trí con trỏ hiện thời	TL10
TRc	Chuyển con trỏ sang phải c ký tự tính từ vị trí con trỏ hiện thời	TR10
/	Xuống dòng	2x, 5F10.3/ 2x,7F10.3
\ hoặc $	Giữ trên cùng một bản ghi	A\
Xâu	In một xâu ký tự (đặt trong cặp dấu	‘Dong tren’/’Dong

KTự	nháy)	duoi’

Ghi chú: **w** là độ rộng trường, **d** là số chữ số sau dấu chấm thập phân, **m** là số ký tự mà một số nguyên chiếm, kể cả chữ số 0 đứng trước, **n** là số ký tự bỏ qua.

Định dạng **FORMAT** cũng có thể được mô tả ngay trong các câu lệnh **READ** và **WRITE** hoặc **PRINT** mà không nhất thiết sử dụng câu lệnh **FORMAT**. Chẳng hạn, các câu lệnh

```
WRITE (*, 30) X, Y, Z
30 FORMAT (3X, 2F10.3, E12.5)
```

tương đương với câu lệnh

```
WRITE (*, ’(3X, 2F10.3, E12.5)’) X, Y, Z
```

Khi đọc dữ liệu vào, ta có thể dùng định dạng tự do như trong các ví dụ trước đây, và cũng có thể dùng định dạng có qui cách. Ví dụ, các trường hợp sau đây sẽ cho kết quả như nhau.

Giả sử, muốn nhập vào ba số **x=12.3, y=23.45, z=123.4**. Với câu lệnh

```
READ (*, *) X, Y, Z
```

ta chỉ cần gõ vào:

12.3 23.45 123.4 (Các số cách nhau bởi các dấu cách)

Nhưng nếu viết câu lệnh đó dưới dạng:

```
READ (*,’(3F5.2)’) X, Y, Z
```

ta có thể gõ vào một dãy các số liên tục:

```
012300234512340
```

Vì định dạng được mô tả bởi **’(3F5.2)’** nên các số được nhập vào là các nhóm gồm 5 chữ số trong đó 2 chữ số cuối cùng của nhóm là 2 chữ số sau dấu chấm thập phân. Ta cũng có thể thay các số 0 không có nghĩa bởi các dấu cách.

3.5 CHU TRÌNH LẶP KHÔNG XÁC ĐỊNH

Chu trình **DO** đã xét trước đây chính là chu trình lặp với số bước lặp được xác định bởi các tham số ***TriDau, TriCuoi*** và ***Buoc.*** Trong thực tế ta thường gặp nhiều bài toán trong đó số bước tính toán cần lặp đi lặp lại không thể xác định được một cách cụ thể, mà là được xác định thông qua một điều kiện cho trước nào đó. Cấu trúc lặp này được gọi là lặp không xác định.

3.5.1 Cấu trúc kết hợp IF và GOTO

Có thể tạo ra chu trình lặp không xác định bằng việc kết hợp **IF** và **GOTO** như sau.

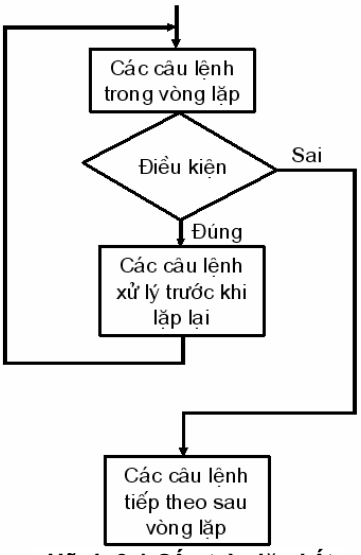
```
m Câu_lệnh_đầu_vòng_lặp
    Các_câu_lệnh_tiếp_theo_trong_vòng_lặp
 IF (BThuc_Logic) GOTO m
```

hoặc:

```
m Câu_lệnh_đầu_vòng_lặp
   Các_câu_lệnh_tiếp_theo_trong_vòng_lặp
 IF (BThuc_Logic) THEN
```

```
    Các_câu_lệnh_xử_lý_trước_khi_lặp_lại
    GOTO m
  END IF
```

Trong đó **m** là nhãn câu lệnh đầu tiên của quá trình cần lặp; ***BThuc_Logic*** là điều kiện để lặp lại quá trình. Nếu ***BThuc_Logic*** nhận giá trị **.TRUE.** thì chương trình sẽ chuyển điều khiển đến câu lệnh đầu vòng lặp, ngược lại, nếu ***BThuc_Logic*** nhận giá trị **.FALSE.** thì quá trình lặp sẽ kết thúc. Sơ đồ khối mô tả tác động của chu trình này được cho trên hình 3.1.



Hình 3.1 Cấu trúc lặp kết hợp IF và GOTO

Ví dụ 3.5. Viết chương trình nhập vào một số dương không vượt quá 10.

Với yêu cầu này, chương trình phải bảo đảm điều kiện, chừng nào số nhập vào không phải là một *số dương không vượt quá 10* thì phải nhập lại. Như vậy, điều kiện quay lại vòng lặp ở đây là số nhập vào hoặc là số âm hoặc số 0, hoặc là số lớn hơn 10; nội dung cần lặp là nhập vào một số. Ta có qui trình sau.

1) Nhập vào một số X

2) Kiểm tra số X:

– Nếu X≤0 hoặc X>10 thì

+ Thông báo lỗi

+ Quay lại bước 1)

3) Kết thúc

Chương trình có thể được viết:

```
PROGRAM IF_GOTO
REAL X
10 PRINT '(A\)',' CHO MOT SO: '
  READ*, X
  IF (X <= 0.OR.X > 10) THEN
    PRINT*,' SO VUA NHAP=',X
    PRINT*,' SAI ! NHAP LAI !'
    PRINT*
    GOTO 10
  END IF
PRINT*
PRINT*,' DUNG ROI! SO VUA NHAP=',X
END
```

3.5.2 Cấu trúc DO và EXIT

Chu trình lặp không xác định cũng có thể được cấu tạo kết hợp **DO** và **EXIT**. Cú pháp cấu trúc như sau.

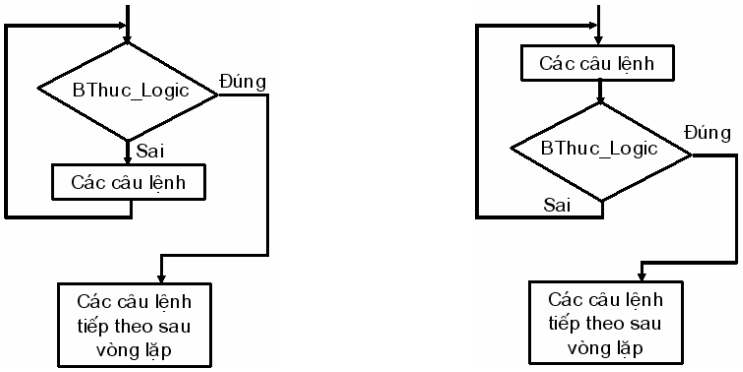
Dạng 1:

```
[TenChuTrinh:] DO
   IF (BThuc_Logic) EXIT
   Các_câu_lệnh
END DO [TenChuTrinh]
```

Dạng 2:

```
[TenChuTrinh:] DO
   Các_câu_lệnh
   IF (BThuc_Logic) EXIT
END DO [TenChuTrinh]
```

Sơ đồ khối mô tả tác động của cấu trúc được cho trên hình 3.2. Ta cần hết sức chú ý *sự khác nhau* giữa hai cấu trúc này. Đối với cấu trúc dạng 1, vì ***BThuc_Logic*** được xác định trước khi ***Các_câu_lệnh*** được thực hiện, nên có thể ***Các_câu_lệnh*** sẽ không được thực hiện một lần nào nếu ngay từ đầu ***BThuc_Logic*** nhận giá trị .**TRUE**. Trong khi đó, ở cấu trúc dạng 2, ***Các_câu_lệnh*** được thực hiện ít nhất một lần.



Dạng 1 **Dạng 2**

Hình 3.2 Cấu trúc lặp DO và EXIT

Ví dụ 3.6: Hãy làm lại ví dụ 3.5 khi sử dụng các cấu trúc **DO** và **EXIT.**

Với cấu trúc dạng 1 ta có chương trình:

```
PROGRAM LAP_1
REAL X
X = –999.  ! Khoi tao X
CauTruc1: DO
  IF (X > 0.AND.X <= 10) EXIT
  PRINT '(A\)','CHO MOT SO: '
  READ*, X
  IF (X <= 0.OR.X > 10) THEN
    PRINT*,' SO VUA NHAP=',X
    PRINT*,' SAI ! NHAP LAI !'
    PRINT*
```

```
  END IF
END DO CauTruc1
PRINT*
PRINT*,' DUNG ROI! X =',X
END
```

Trong trường hợp này, câu lệnh

X = –999.

có tác dụng khởi tạo giá trị của **X**. Nó là câu lệnh bắt buộc để tránh việc tham chiếu đến biến chưa xác định của câu lệnh

IF (X > 0.AND.X <= 10) EXIT

Bây giờ ta thay câu lệnh

X = –999.

bởi câu lệnh khác trong đó **X** thỏa mãn điều kiện $0 < \mathbf{X} \le 10$, chẳng hạn

X = 5.

và chạy lại chương trình. Kết quả nhận được thật bất ngờ! Bạn đọc hãy giải thích tại sao.

Với cấu trúc dạng 2 ta có thể viết chương trình như sau:

```
REAL X
CauTruc2: DO
  PRINT '(A\)','CHO MOT SO: '
  READ*, X
  IF (X > 0.AND.X <= 10) EXIT
  PRINT*,' SO VUA NHAP=',X
  PRINT*,' SAI ! NHAP LAI !'
  PRINT*
END DO CauTruc2
PRINT*
PRINT*,' DUNG ROI! X =',X
END
```

Trong cấu trúc này ta không cần khởi tạo giá trị của **X**. Bạn đọc hãy giải thích tại sao?

3.5.3 Cấu trúc DO WHILE…END DO

Cú pháp cấu trúc này có dạng:

```
DO WHILE (BThuc_Logic)
   Các_câu_lệnh
END DO
```

Tác động của cấu trúc này hoàn toàn tương đương với cấu trúc

```
[TenChuTrinh:] DO
   IF (BThuc_Logic) EXIT
   Các_câu_lệnh
END DO [TenChuTrinh]
```

Ví dụ 3.7. Để minh họa cho cách sử dụng cấu trúc này ta hãy làm lại ví dụ ở mục trước. Mã nguồn chương trình có thể được viết:

```
REAL X
PRINT '(A\)',' CHO MOT SO: '
READ*, X
DO WHILE (X <= 0.OR.X > 10)  ! Điều kiện lặp lại
  PRINT*,' SAI ! '
  PRINT*
  PRINT '(A\)',' CHO MOT SO: '
  READ*, X    ! X nhận giá trị mới
END DO
PRINT*
PRINT*,' DUNG ROI! SO VUA NHAP=',X
END
```

Hai dòng lệnh thứ hai và thứ ba trong chương trình trên có thể được thay thế bởi câu lệnh gán giá trị khởi tạo. Nếu ngay từ đầu giá trị của **X** đã thỏa mãn điều kiện 0 < **X** ≤ 10 thì ***Các_câu_lệnh*** nằm giữa **DO WHILE** và **END DO** sẽ không bao giờ được thực hiện.

3.5.4 Lệnh CYCLE

Trong một số trường hợp thực hiện chu trình lặp, tùy theo điều kiện, cần phải bỏ qua một số bước nào đó, nhưng chưa thoát khỏi hoàn toàn chu trình, ta có thể sử dụng câu lệnh **CYCLE**. Cú pháp câu lệnh như sau:

CYCLE [Tên_Chu_Trình]

Lệnh **CYCLE** nằm trong các chu trình lặp **DO** hoặc **DO WHILE**, có tác dụng bỏ qua các câu lệnh trong vòng lặp nằm sau **CYCLE** và chuyển điều khiển về khối kiểm tra điều kiện lặp lại của chu trình có tên là ***Tên_Chu_Trình***.

Lệnh **CYCLE** có thể nằm trong các chu trình lồng nhau. Nếu không chỉ ra ***Tên_Chu_Trình*** thì **CYCLE** chỉ có tác động đối với chu trình lặp trong nhất chứa nó.

Ví dụ, ta hãy xét chương trình sau.

```
INTEGER I
INTEGER, PARAMETER :: N = 10
LapDO: DO I=1,N
                print*,'Chi so vong lap DO: ',i
                IF (i>3) CYCLE LapDO
                print*,'Lan duoc LAP boi DO:',i
END DO LapDO
END
```

Trong ví dụ trên, những câu lệnh nằm giữa hai dòng lệnh **DO** và **IF** sẽ được thực hiện **N** lần (**I=1,2,... ,N**), nhưng các câu lệnh nằm sau câu lệnh **IF** cho đến hết chu trình **DO** chỉ được thực hiện chừng nào biểu thức **(i>3)** còn nhận giá trị **.FALSE.** Câu lệnh **IF** trong trường hợp này qui định khi nào thì lệnh **CYCLE** được gọi tới, và khi nó được gọi, quá trình lặp sẽ bỏ qua những câu lệnh sau đó, chỉ số lặp được tăng lên, điều khiển được chuyển về đầu vòng lặp.

Trên thực tế có thể kết hợp giữa **CYCLE** và **EXIT** trong các chu trình lặp phức tạp hơn. Ta sẽ khảo sát kỹ ví dụ sau đây để hiểu rõ tác động của các câu lệnh này và sự khác nhau giữa chúng.

```
INTEGER i,j,k
INTEGER, PARAMETER :: N = 10
write(*,'(/A, I2)')' Dieu khien lap khi su dung &
          & CYCLE va EXIT, N = ', N
write (*,900)
Vong1: DO i = 1, n
  if (i > 3) EXIT Vong1
  write (*,910) i
  Vong2: DO j = 1, n
    if (j > 2) CYCLE Vong2
    if (i == 2.and.j > 1) EXIT Vong2
    write (*,920) j
    Vong3: DO k = 1, n
      if (k > 2) CYCLE Vong3
      if (i == 1.and.j > 1) EXIT Vong2
      write (*,930) k
    END DO Vong3
  END DO Vong2
END DO Vong1
WRITE (*,'(/A)') ' Hoan tat cac chu trinh.'
900 FORMAT(/' Vong:    1       2       3 ')
910 FORMAT(11x, i2)
920 FORMAT(21x, i2)
930 FORMAT(31x, i2)
END
```

Khi chạy chương trình này, kết quả nhận được trên màn hình có dạng sau:

```
Dieu khien lap khi su dung CYCLE va EXIT, N = 10
 Vong:    1       2       3
        1
                1
                        1
                        2
                2
        2
                1
                        1
                        2
        3
                1
                        1
                        2
                2
                        1
                        2
 Hoan tat cac chu trinh.
```

Ta thấy, mặc dù chu trình ***Vong1:*** **DO** được qui định lặp 10 lần (**N=10**), nhưng do câu lệnh **if (i > 3) EXIT** ***Vong1***, nên số lần lặp của chu trình này chỉ được thực hiện 3 lần. Nghĩa là khi **I≤3** thì các

chu trình **DO** thứ hai và thứ ba sau đó mới được thực hiện. Còn khi **I>3** thì thoát khỏi chu trình ***Vong1*** với giá trị của **I=4**. Nhưng nếu ta thay câu lệnh

if (i > 3) EXIT *Vong1*

bởi câu lệnh

if (i > 3) CYCLE *Vong1*

thì mặc dù kết quả nhận được hoàn toàn tương tự, nhưng giá trị của biến **I** sau khi thoát khỏi chu trình sẽ là 10 (bằng **N**). Sở dĩ như vậy là do sự khác nhau cơ bản của **EXIT** và **CYCLE**. Trong khi câu lệnh **EXIT** tạo ra tác động làm kết thúc chu trình một cách cưỡng bức, thì lệnh **CYCLE** chỉ *bỏ qua* việc thực hiện các câu lệnh sau nó, còn chu trình vẫn được thực hiện và kết thúc bình thường. Trong chu trình **DO** có nhãn ***Vong2***, câu lệnh

if (j > 2) CYCLE *Vong2*

có tác dụng bỏ qua những câu lệnh sau nó (nhưng vẫn tiếp tục thực hiện vòng lặp) khi **J > 2**, trong khi câu lệnh

if (i == 2.and.j > 1) EXIT *Vong2*

lại có tác dụng làm kết thúc chu trình ngay lập tức nếu biểu thức **(i==2.and.j>1)** thỏa mãn. Một cách tương tự, bạn đọc có thể đối chiếu kết quả tính của chương trình với các câu lệnh tương ứng để hiểu rõ thêm.

3.5.5 Một số ví dụ về chu trình lặp không xác định

a. Tính số PI theo công thức $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$

Đây là tổng của một chuỗi đan dấu mà số hạng tổng quát là $\frac{(-1)^{n-1}}{2n-1}$, n=1,2,… Vậy ta có các bước tính sau.

1) Khởi tạo N=1, Tmp=0, EP=0.0001 [,SS=1.0] , DAU=1
2) Tính Sn = Tmp + DAU / (2*N – 1) (Số hạng thứ n)
   (Tổng tích lũy đến số hạng thứ n)
3) Gán DAU = –DAU (đảo dấu)
4) Tính sai số tương đối SS = ABS((Sn–Tmp)/Sn)
5) So sánh SS với EP:
   - Nếu SS >= EP:
     - Lưu giá trị của Sn vào Tmp
     - Quay lại bước 2)
   - Nếu SS < EP:
     - Tính PI = Sn * 4

- In kết quả và Kết thúc chương trình

Sau đây là lời chương trình viết bằng các cách khác nhau khi sử dụng các cấu trúc lặp đã trình bày trong các mục trước. Cũng cần lưu ý rằng, các chương trình ở đây chỉ nhằm mục đích giải thích về khía cạnh lập trình mà chưa chú ý đến tính tối ưu của chương trình. Sau khi đã làm chủ được ngôn ngữ lập trình, bạn đọc có thể thay đổi hoặc viết lại cho tốt hơn.

Cách 1: Sử dụng chu trình lặp kết hợp **IF** và **GOTO**

```
PROGRAM TINHPI1 ! Cach 1: IF & GOTO
REAL EPS, SS, PI, TMP
INTEGER :: N, DAU = 1
EPS=0.0001
TMP=0.0
N=1
100 PI=TMP+DAU/FLOAT(2*N-1)
  DAU = –DAU
  SS=ABS((PI-TMP)/PI)
     PRINT*,'Vong lap thu ',N,' Sai so=',SS
  IF (SS >= EPS) THEN
    TMP = PI
    N=N+1
    GOTO 100
  ELSE
    PI=PI*4.0
    WRITE(*,300)PI
 300 FORMAT(4X,' PI = ',F10.4)
  END IF
END
```

Cách 2: Sử dụng chu trình lặp kết hợp **DO** và **EXIT**

```
PROGRAM TINHPI2 ! CACH 2: DO & EXIT
REAL EPS, SS, PI, TMP
INTEGER :: N, DAU = 1
EPS=0.0001
TMP=0.0
N=1
DO
  PI=TMP+DAU/FLOAT(2*N-1)
  DAU = –DAU
  SS=ABS((PI-TMP)/PI)
     PRINT*,'Vong lap thu ',N,' Sai so=',SS
  IF (SS < EPS) EXIT
    TMP = PI
    N=N+1
END DO
PI=PI*4.0
WRITE(*,300)PI
300  FORMAT(4X,' PI = ',F10.4)
```

```
END
```

Cách 3: Sử dụng cấu trúc **DO WHILE**

```
PROGRAM TINHPI3 ! CACH 3: DO WHILE
REAL EPS, SS, PI, TMP
INTEGER :: N, DAU = 1
EPS=0.0001
TMP=0.0
N=1
SS=1.0
DO WHILE (SS >= EPS)
  PI=TMP+DAU/FLOAT(2*N-1)
  DAU = –DAU
  SS=ABS((PI-TMP)/PI)
     PRINT*,'Vong lap thu ',N,' Sai so=',SS
  TMP = PI
  N=N+1
END DO
PI=PI*4.0
WRITE(*,300)PI
300  FORMAT(4X,' PI = ',F10.4)
END
```

b. Tìm số nguyên dương lớn nhất **n** thỏa mãn điều kiện $3n^3–212n<10$

```
INTEGER N, A
N=0
A=–999
DO WHILE (A < 10)
  N = N + 1
  A = 3*N**3 - 212*N
     PRINT*,'N = ',N,'  A=', A
END DO
PRINT*,'So phai tim la N = ', N–1
END
```

Trong ví dụ này ta cần chú ý đến cách khởi tạo giá trị của biến **N** và **A**.

BÀI TẬP CHƯƠNG 3

3.1 Viết chương trình tính số π theo công thức:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$$

với độ chính xác $\varepsilon=10^{-4}$

3.2 Viết chương trình tính số π theo công thức:

$$\frac{\pi}{8} = \frac{1}{1\times 3} + \frac{1}{5\times 7} + \frac{1}{9\times 11} + \ldots$$

với độ chính xác $\varepsilon=10^{-4}$

3.3 Viết chương trình nhập vào các số thực a, b (a<b) và tính tích phân: $I = \frac{1}{\sqrt{2\pi}}\int_a^b e^{-\frac{1}{2}x^2} dx$ theo phương pháp hình thang với độ chính xác $\varepsilon = 10^{-4}$.

3.4 Viết chương trình nhập vào các số thực a, b (a<b) và tính tích phân: $I = \int_a^b x^2 \sin x dx$ theo phương pháp hình thang với độ chính xác $\varepsilon = 10^{-4}$.

3.5 Thông tin về một cơn bão được cho trong file TRACK.TXT mà nội dung của nó được chỉ ra như sau:

Date: 14-22 JUL 2003
Typhoon #8

ADV	LAT	LON	TIME	WIND	PR	STAT
1	10.50	136.90	07/14/12Z	15	1006	TD
2	10.50	135.70	07/14/18Z	15	1006	TD
3	10.40	134.50	07/15/00Z	15	1007	TD
4	10.20	133.30	07/15/06Z	20	1004	TD
5	10.00	132.10	07/15/12Z	25	1002	TD
6	10.00	130.90	07/15/18Z	25	1002	TD
7	10.10	129.80	07/16/00Z	30	1000	TD
8	10.30	128.80	07/16/06Z	30	1000	TD
9	10.60	127.80	07/16/12Z	30	1000	TD
10	10.90	126.80	07/16/18Z	40	994	TS
11	11.00	125.70	07/17/00Z	45	991	TS
12	11.00	124.50	07/17/06Z	40	994	TS
13	11.30	123.20	07/17/12Z	40	994	TS
14	11.50	121.90	07/17/18Z	40	994	TS
15	11.80	120.50	07/18/00Z	45	991	TS
16	12.40	119.40	07/18/06Z	45	991	TS
17	13.00	118.60	07/18/12Z	45	991	TS
18	13.40	117.90	07/18/18Z	45	991	TS
19	13.90	117.30	07/19/00Z	45	991	TS
20	14.50	116.70	07/19/06Z	45	987	TS
21	15.10	116.20	07/19/12Z	45	987	TS
22	15.70	115.60	07/19/18Z	50	987	TS
23	16.30	115.00	07/20/00Z	60	980	TS
24	17.00	114.30	07/20/06Z	65	976	TY-1
25	17.60	113.60	07/20/12Z	65	976	TY-1
26	18.00	112.80	07/20/18Z	65	976	TY-1
27	18.30	112.00	07/21/00Z	65	976	TY-1
28	18.70	111.10	07/21/06Z	60	980	TS
29	18.80	110.00	07/21/12Z	55	984	TS
30	18.80	108.60	07/21/18Z	60	980	TS
31	18.90	107.30	07/22/00Z	60	980	TS
32	19.80	106.40	07/22/06Z	55	984	TS
33	20.10	105.00	07/22/12Z	40	994	TS

34 19.70 103.80 07/22/18Z 35 997 TS

Trong đó, dòng đầu tiên chỉ ngày bắt đầu và ngày kết thúc của cơn bão; dòng thứ hai chỉ số thứ tự trong năm của cơn bão; dòng thứ ba là tiêu đề các cột của các dòng tiếp theo mà ý nghĩa của chúng là: cột 1 chỉ số thứ tự, cột 2 và cột 3 lần lượt là vĩ độ và kinh độ địa lý của tâm bão; cột 4 ghi tháng, ngày, giờ của vị trí tâm bão (mm/dd/hhz), với z là ký hiệu lấy giờ chuẩn quốc tế; cột 5 là tốc độ gió cực đại (knots); cột 6 là áp suất khí quyển tại tâm bão (mb); và cột 7 là cường độ bão, với ký hiệu TD – áp thấp nhiệt đới, TS – bão, TY–1, TY–2,... – bão mạnh ở mức độ 1, 2,... Viết chương trình đọc số liệu từ file, xử lý, tính toán và hiển thị lên màn hình các thông tin sau:

– Số ngày kéo dài của cơn bão (số ngày bão hoạt động)

– Thời điểm bão có tốc độ gió cực đại lớn nhất

– Những khoảng thời gian bão đạt các cường độ TD, TS và TY.

3.6 Phương pháp Archimedes để tính số p có thể được mô tả bởi sơ đồ thuật toán sau:

1) Gán A = 1, N = 6 và SS=1.0E–6 (sai số cho phép)

2) Tiến hành:

– Thay thế N bởi 2N

– Thay thế A bởi $\sqrt{2-\sqrt{(4-A^2)}}$

– Gán L = NA/2

– Gán U = L/$\sqrt{1-A^2/2}$

– Gán PI = (U+L)/2 (là giá trị ước lượng của số p)

– Gán E = (U–L)/2 (là ước lượng của sai số)

– Nếu E>=SS: quay lại bước 2),

ngược lại thì:

In kết quả (In giá trị của PI)

3) Kết thúc

Dựa vào thuật toán trên hãy viết chương trình tính số p và so sánh kết quả với các bài tập 3.1 và 3.2.

3.7 Viết chương trình tính và in bảng các giá trị của đối số x và của hàm $f(x)=x\sin\left[\frac{\pi(1+20x)}{2}\right]$ tương ứng khi cho x biến thiên trong khoảng đóng [–1; 1] với bước nhảy $\Delta x = 0.1$.

3.8 Giả sử "sóng vuông" với chu kỳ T có thể định nghĩa bởi hàm $f(t) = \begin{cases} 1 & (0 < t < T) \\ -1 & (-T < t < 0) \end{cases}$.

Chuỗi Fourier của hàm *f(t)* được cho bởi $\frac{4}{\pi}\sum_{k=0}^{\infty}\frac{1}{2k+1}\sin\left[\frac{(2k+1)\pi t}{T}\right]$. Ta cần biết phải lấy bao nhiêu hạng tử của chuỗi để xấp xỉ tốt hàm *f(t)*. Cho T = 1, hãy viết chương trình tính và hiển thị tổng của *n* số hạng đầu tiên của chuỗi khi cho *t* biến thiên trong khoảng từ 0−1 với bước nhảy bằng 0.1. Chạy chương trình với các giá trị *n* khác nhau, chẳng hạn *n* = 3, 6, 9,...

3.9 Ký hiệu tổng của N số tự nhiên đầu tiên là S=1+2+...+N. Viết chương trình xác định số N và tổng S sao cho tổng S là số lớn nhất không vượt quá 1000.

3.10 Sự tăng trưởng dân số được giả định là tuân theo qui luật mà số lượng dân số tại thời điểm *t* nào đó được cho bởi công thức:

$$x(t) = \frac{Kx_0}{(K - x_0)e^{-rt} + x_0}$$

trong đó x_0 là số lượng dân số tại thời điểm *t=0*, *r* là tốc độ tăng trưởng và *K* là hằng số đặc trưng cho nhân tố môi trường sống. Viết chương trình tính và in giá trị của *x(t)* trong khoảng thời gian 200 năm với bước nhảy của *t* bằng 10. Lấy x_0 = 2, *r* = 0.1, *K* = 1000. Chạy chương trình với các giá trị của *K* khác nhau và so sánh kết quả để nhận xét về ý nghĩa của *K*.

3.11 Viết chương trình nhập vào một dãy các số thực với số lượng số chưa xác định. Chương trình cho phép nhập cho đến khi gặp số nhập vào bằng −999.0 thì kết thúc việc nhập. Tính giá trị trung bình số học của dãy số thực đó. In kết quả lên màn hình dưới dạng:

TRUNG BINH CUA N SO VUA NHAP = XXXXX.XX

trong đó N là số lượng các số được tính trung bình (không kể số −999.0), giá trị trung bình được in theo định dạng số thực dấu phẩy tĩnh với 2 chữ số sau dấu chấm thập phân.

3.12 Giả sử giá trị hợp lệ của nhiệt độ không khí ở một trạm quan trắc chỉ nằm trong phạm vi từ −5 độ đến 40 độ. Viết chương trình nhập vào chuỗi số liệu quan trắc nhiệt độ không khí và tính nhiệt độ trung bình của trạm đó. Chương trình cho phép chỉ nhận những giá trị số liệu hợp lệ và với độ dài chuỗi số liệu bất kỳ. In kết quả lên màn hình dưới dạng:

DO DAI CHUOI = IIII, NHIET DO TRUNG BINH = XXXX.X

trong đó IIII là số một số nguyên dương chiếm độ rộng 4 ký tự chỉ độ dài chuỗi quan trắc, nhiệt độ trung bình được in theo định dạng số thực dấu phẩy tĩnh với 1 chữ số sau dấu chấm thập phân.

3.13 Dãy số Fibonacci là dãy số 1, 1, 2, 3, 5, 8, 13,... Chúng được tạo ra theo nguyên tắc: $F_n = F_{n-1} + F_{n-2}$, trong đó $F_0 = F_1 = 1$. Viết chương trình nhập vào số nguyên dương *n* và in ra dãy *n* số Fibonacci đó.

CHƯƠNG 4. CHƯƠNG TRÌNH CON (SUBROUTINE VÀ FUNCTION) VÀ MODUL

4.1 KHÁI NIỆM

Trong lập trình, nhất là đối với những bài toán lớn, các chương trình thường bao gồm nhiều bộ phận khác nhau, trong đó có những bộ phận thường được sử dụng lặp đi lặp lại nhiều lần. Ngoài ra, những đoạn chương trình này có thể được sử dụng cho các chương trình khác. Việc viết một chương trình trong đó có nhiều đoạn trùng lặp nhau sẽ gây ra sự nhàm chán và không hiệu quả, thậm chí làm cho chương trình trở nên rối rắm hơn. Để tổ chức một chương trình gọn gàng, dễ khai thác, Fortran cho phép phân mảnh chương trình và tạo thành các chương trình con.

Có hai khái niệm chương trình con là thủ tục (**SUBROUTINE**) và hàm (**FUNCTION**). Các chương trình con cũng có thể chia thành hai loại là *chương trình con trong* và *chương trình con ngoài*. Ta cũng có thể chọn ra những chương trình con trong số các chương trình con để tạo ra một thư viện riêng cho mình. Tập hợp các chương trình con này được gọi là modul. Các *chương trình chính* (Main Program), *chương trình con ngoài* (External Subprogram) và các *modul* được gọi là các *đơn vị chương trình* (Program Unit). Về nguyên tắc, các *chương trình con trong* sẽ *nằm trong các đơn vị chương trình* khác và được biên dịch cùng với đơn vị chương trình mà nó phụ thuộc, trong khi các chương trình con ngoài có thể được biên dịch một cách độc lập. Cái khác nhau cơ bản giữa chương trình con trong và chương trình con ngoài là ở chỗ, trong khi các chương trình con trong có thể sử dụng tên biến, hằng và những khai báo của đơn vị chương trình quản lý nó, thì các chương trình con ngoài, do không được phép nằm trong đơn vị chương trình khác nên không có tính chất đó.

Cũng cần chú ý phân biệt các khái niệm đơn vị chương trình, (bộ) chương trình và file. Trong một file có thể chứa nhiều đơn vị chương trình và chúng có thể gộp lại thành một (bộ) chương trình. Nói chung ta *không nên* tổ chức như vậy, nhất là đối với những chương trình lớn, mà *nên* tách các đơn vị chương trình ra, mỗi đơn vị chương trình chứa trong một file. Nói cách khác, nếu một (bộ) chương trình gồm một chương trình chính và ***n*** chương trình con là các đơn vị chương trình thì chúng *nên* được lưu trữ trong ***n+1*** file tách biệt. Cách tổ chức này cho phép các chương trình khác nhau sử dụng chung những đơn vị chương trình như nhau. Tuy nhiên, ta *không thể* lưu trữ các *chương trình con trong* vào các file tách biệt với các đơn vị chương trình quản lý nó, ngoại trừ trường hợp sử dụng câu lệnh **INCLUDE** (sẽ được trình bày sau).

4.2 THƯ VIỆN CÁC HÀM TRONG

Trước đây, ta đã làm quen với cách sử dụng hàm thư viện trong Fortran. Trong một số ví dụ, các hàm thư viện này cũng đã được sử dụng, như hàm tính căn bậc hai, hàm tính cosin của một góc,… Để thuận tiện cho việc tham khảo, tra cứu, trong phần phụ lục đã dẫn ra những hàm thông dụng nhất

trong thư viện các chương trình con của Fortran. Sau đây là một ví dụ về cách sử dụng các hàm thư viện này.

Ví dụ 4.1. Viết chương trình lập bảng tra giá trị hàm sin và cosin của các góc nằm trong khoảng 0–90^o.

Ta có thể viết chương trình như sau.

```
PROGRAM BANG_SIN_COS
IMPLICIT NONE
REAL Pi
INTEGER I,J
PI=4.*atan(1.)
WRITE(*,'("    ",11I7)') (I,I=0,60,6)
DO j=0,89
  WRITE(*,'(I3,1X,11F7.5,I4)') &
   J,(SIN((REAL(j)+I/60.)/180.*pi),I=0,60,6), 89-J
ENDDO
WRITE(*,'("    ",11I7)') (60-I,I=0,60,6)
END
```

Trong chương trình trên đã sử dụng ba hàm thư viện của Fortran là hàm **ATAN**, hàm **SIN** và hàm **REAL**. Hàm **ATAN(x)** để tính acrtan của số **x**. Vì Fortran không định nghĩa hằng số **π** nên ta phải tính **π/4 = arctan (1)**, hàm **SIN** để tính sin của các góc từ 0–90^o mà giá trị của chúng trong khoảng này được lấy cách nhau 6 phút. Đối số của các hàm lượng giác, như sin, cos, tang, cotang, phải là *radian*. Còn hàm **REAL** dùng để đổi số nguyên **J** thành số thực, nó khác với lệnh khai báo **REAL** dùng để khai báo các biến có kiểu số thực trong phần khai báo.

4.3 CÁC CHƯƠNG TRÌNH CON TRONG

4.3.1 Hàm trong (Internal FUNCTION)

Hàm trong có thể được khai báo như sau.

```
[KiểuDL][RECURSIVE] FUNCTION TenHam  &
      ([Các_đối_số]) [RESULT (TenKetQua) ]
 [Các_câu_lệnh_khai_báo]
 [Các_câu_lệnh_thực hiện]
 [TenHam = ...]
END FUNCTION [TenHam]
```

Trong đó:

KiểuDL là kiểu dữ liệu mà hàm sẽ trả về. Ta có thể bỏ qua tùy chọn này khi sử dụng tùy chọn ***RESULT***.

RECURSIVE là tùy chọn để chỉ hàm là hàm đệ qui.

TenHam là tên của hàm, được dùng để gọi tới hàm.

Các_đối_số là danh sách các đối số hình thức, liệt kê cách nhau bởi dấu phẩy.

TenKetQua là tên biến chứa kết quả trả về của hàm. Nếu sử dụng tùy chọn này thì câu lệnh ***TenHam*** **= ...** không được phép xuất hiện. Ngược lại, nếu không sử dụng tùy chọn **RESULT** thì phải có dòng lệnh ***TenHam*** **= ...** để trả về kết quả của hàm.

Hàm có thể được gọi tới bằng cách hoặc gán giá trị hàm cho biến, hoặc hàm tham gia vào biểu thức tính:

TenBien = TenHam ([Các_đối_số])

Ví dụ, câu lệnh

Cx = COS (x)

sẽ tính giá trị cosin của **x** bằng lời gọi hàm **COS(x)** rồi gán cho biến **Cx**. Còn trong câu lệnh

Pi = 4.0 * ATAN (1.0)

giá trị của arctan(1.0) được tính thông qua lời gọi hàm **ATAN(1.0)**, sau đó lấy kết quả nhận được nhân với 4.0 rồi mới gán giá trị của biểu thức cho biến **Pi**.

Khi xây dựng hàm, ***Các_đối_số*** là những đối số hình thức, nhưng khi gọi hàm thì ***Các_đối_số*** phải được thay vào đó là danh sách đối số thực. Ví dụ, hàm **YNew** được xây dựng với ba đối số hình thức **X, Y, A**:

```
FUNCTION YNew ( X, Y, A )
  ...
  YNew = ...
END FUNCTION YNew
```

Khi hàm này được gọi tới, các ***đối số hình thức*** được thay bởi những ***đối số thực***:

```
U = ...
V = ...
Pi = ...
Y = YNew( U, V, Pi/2 )
```

Các đối số hình thức và đối số thực phải tương ứng **1-1** về thứ tự xuất hiện cũng như kiểu dữ liệu của chúng.

4.3.2 Thủ tục trong (Internal SUBROUTINE)

Về cơ bản cú pháp khai báo thủ tục giống với khai báo hàm. Chỉ có một số khác biệt sau:

– Không có giá trị nào được liên kết với tên thủ tục

– Để gọi tới thủ tục phải dùng từ khóa **CALL**

– Từ khóa **SUBROUTINE** được dùng để định nghĩa thủ tục thay cho từ khóa **FUNCTION**

– Hàm không có đối số sẽ được gọi tới bằng cách thêm vào sau tên hàm cặp dấu ngoặc đơn rỗng ( ) (Ví dụ, **MyFunction()**), nhưng nếu thủ tục không có đối số thì khi gọi tới sẽ không cần cặp dấu ngoặc đơn này (Ví dụ: **CALL MySubroutine**).

Cú pháp khai báo thủ tục như sau.

SUBROUTINE TenThuTuc [(Các_đối_số)]

[*Các_câu_lệnh_khai_báo*]

[*Các_câu_lệnh_thực_hiện*]

END SUBROUTINE [TenThuTuc]

Trong đó ***Các_đối_số*** là danh sách đối số hình thức, được liệt kê cách nhau bởi dấu phẩy.

Lời gọi thủ tục:

CALL TenThuTuc [(*Các_đối_số_thực*)]

Trong đó danh sách các đối số hình thức và danh sách các đối số thực cũng phải tương ứng 1–1 với nhau.

***Chú ý*:**

- Nói chung hàm là một chương trình con chỉ trả về duy nhất một giá trị: Giá trị của hàm ứng với các đối số. (Sau này ta sẽ thấy hàm có thể trả về nhiều giá trị)
- Trong định nghĩa hàm (**FUNCTION**), trước khi trả về chương trình gọi, giá trị của hàm luôn được xác định bởi một câu lệnh gán hoặc cho ***TenHam*** hoặc cho biến ***TenKetQua*** trong tùy chọn ***RESULT***. Đối với các thủ tục thì kết quả có thể sẽ được trả về thông qua danh sách các đối số, cũng có thể là một hoặc một số nhiệm vụ nào đó.
- Hàm (và thủ tục) kết thúc ở câu lệnh **END** cuối cùng. Tuy nhiên cũng có thể sử dụng câu lệnh **RETURN** để trả về chương trình gọi. Khi gặp câu lệnh **RETURN** chương trình con sẽ được giải phóng và quay về chương trình gọi, bất chấp sau nó có còn câu lệnh thực hiện nào hay không.

4.4 CÂU LỆNH CONTAINS

Câu lệnh **CONTAINS** là câu lệnh không thực hiện, dùng để phân cách thân chương trình chính (chính xác hơn là đơn vị chương trình) với các *chương trình con trong* thuộc nó. Các chương trình con trong được sắp xếp ngay sau câu lệnh **CONTAINS** và trước từ khóa **END** của chương trình chính. Bố cục tổng quát của chương trình có dạng như sau.

PROGRAM *TenChuongTrinh*
[*Các_câu_lệnh_khai_báo*]
[*Các_câu_lệnh_thực_hiện*]
[CONTAINS
***Các_chương_trình_con_trong*]**
END [PROGRAM [*TenChuongTrinh*]]

Ở đây, ***Các_chương_trình_con_trong*** là những hàm trong hoặc thủ tục trong chịu sự quản lý của chương trình ***TenChuongTrinh***. Ví dụ, trong chương trình sau đây, **CT_CHINH** sẽ gọi đến chương trình con trong có tên là **CT_CON**.

```
PROGRAM CT_CHINH
REAL A(10)
. . .
CALL CT_CON (A)
```

```
CONTAINS
  SUBROUTINE CT_CON (B)
  REAL B(10)
  . . .
  END SUBROUTINE CT_CON
END PROGRAM CT_CHINH
```

4.5 MỘT SỐ VÍ DỤ VỀ CHƯƠNG TRÌNH CON TRONG

Ví dụ 4.2. Tính tích phân xác định $I = \int_a^b f(x)dx$ bằng phương pháp hình thang.

Giả sử $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$. Ta lần lượt chia khoảng *(a; b)* ra làm *N* đoạn bằng nhau $\Delta x=(b-a)/N$, xác định bởi các điểm chia $x_0=a$, $x_1=x_0+\Delta x,\ldots$, $x_N=b$; mỗi lần như vậy ta tính diện tích của *N* hình thang xác định bởi các đáy $f(x_i)$, $f(x_i+\Delta x)$ và chiều cao Δx. Giá trị của tích phân sẽ được xấp xỉ bởi tổng diện tích của *N* hình thang này. Rõ ràng, khi *N* càng lớn thì tổng diện tích của các hình thang này càng tiệm cận tới giá trị tích phân. Do đó độ chính xác của phép xấp xỉ này được xác định bởi sai số tương đối $|((S2-S1)/S2) < \varepsilon|$, trong đó S1 và S2 là tổng diện tích các hình thang ứng với $N=K$ và $N=K+1$. Từ đó ta có sơ đồ tính và lời chương trình như sau.

B1) Cho giá trị của a, b (a<b), Epsilon

B2) Khởi tạo N=0, S1=0

B3) Tăng số khoảng chia lên 1: N = N+1

B4) Chia đoạn (a; b) làm N khoảng, với cự ly mỗi khoảng DelX = (b–a)/N

B5) Tính tổng diện tích N hình thang và gán cho S2:

1) Gán S2=0
2) Lặp lại N lần, mỗi lần ứng với một hình thang: j = 1, 2,…, N
   a) Xác định x1, x2, f(x1), f(x2): x1=a+(j–1)*DelX; x2=x1+DelX
   b) Tính diện tích hình thang thứ j: Tmp = (f(x1) + f(x2)) * DelX / 2
   c) Cộng dồn diện tích hình thang vừa tính vào S2: S2=S2+Tmp

B6) Tính sai số: SS=ABS((S2–S1)/S2)

B7) Kiểm tra điều kiện kết thúc:

1) Nếu SS < Epsilon: In kết quả và kết thúc chương trình
2) Nếu SS >= Epsilon:
   a) Lưu giá trị S2 vào S1: S1 = S2
   b) Lặp lại từ bước B3)

```
PROGRAM TICHPHAN
INTEGER N, J
```

```
REAL S1,S2,DELX
REAL X, F1,F2, SS,EP, HSO
REAL, PARAMETER :: EP=1.E-4, A=0., B=3.
N=0
S1=0
DO
  N=N+1
  DELX = (B-A)/REAL(N)/2.0
  S2=0
  DO J=1,N
    X = A + (J-1)*DELX
    IF (J>1) THEN
      F1 = F2
    ELSE
      F1= F(X)
    END IF
    X = X + DELX
    F2= F(X)
    S2= S2 + (F1+F2)*DELX
  END DO
  SS = ABS((S2-S1)/S2)
  IF (SS < EP ) EXIT
  S1 = S2
  PRINT*,'SO HINH THANG =',N
END DO
PRINT '('' GIA TRI TP = '',F10.4)', S2

CONTAINS
 FUNCTION F(X) RESULT (Fr)
  Fr=1.0/SQRT(2.0*(4.0*ATAN(1.)))*EXP(-0.5*X*X)
 END FUNCTION F
END
```

Trong chương trình trên, giá trị các cận tích phân và sai số cho phép được khởi tạo thông qua lệnh khai báo hằng, **F(X)** là hàm trong với đối số hình thức là **X**. Kết quả trả về của hàm được lưu trong biến **Fr** ở tùy chọn **RESULT**.

Ví dụ 4.3. Giải phương trình f(x) = 0 bằng phương pháp lặp Newton.

Nội dung phương pháp lặp Newton giải phương trình *f(x)=0* có thể tóm tắt qua các bước như sau.

1) Khởi tạo nghiệm *x* bằng một giá trị ban đầu nào đó

2) Gán *x* bởi *x–f(x)/f'(x)*, trong đó *f'(x)* là đạo hàm bậc nhất của *f(x)*

3) Tính và kiểm tra điều kiện *f(x)* ~ 0

– Nếu chưa thỏa mãn thì quay lại bước 2)

– Nếu thỏa mãn thì in kết quả và kết thúc chương trình.

Giả sử cho $f(x) = x^3 + x - 3$. Khi đó $f'(x) = 3x^2 + 1$. Ta chọn giá trị khởi tạo của x là 2. Điều kiện để xem x là nghiệm gần đúng của phương trình là: hoặc thỏa mãn $f(x) < 10^{-6}$ hoặc số lần lặp lớn hơn hoặc bằng 20. Lời chương trình như sau.

```
PROGRAM Newton
! Giai PT f(x) = 0 bang PP Newton
IMPLICIT NONE
INTEGER :: Its    = 0   ! Dem lan lap
INTEGER :: MaxIts = 20  ! So lan lap cuc dai
LOGICAL :: Converged = .false. ! Dieu kien hoi tu
REAL :: Eps = 1E-6   ! Sai so cho phep
REAL :: X = 2.       ! Gia tri nghiem khoi tao
DO WHILE (.NOT. Converged .AND. Its < MaxIts)
  X = X - F(X) / DF(X)
  PRINT*, X, F(X)
  Its = Its + 1
  Converged = ABS( F(X) ) <= Eps
END DO

IF (Converged) THEN
  PRINT*, 'Hoi tu'
ELSE
  PRINT*, 'Phan ky'
END IF
PRINT*,'Nghiem PT: X = ',X
CONTAINS
  FUNCTION F(X)
    REAL F, X
    F = X ** 3 + X - 3
  END FUNCTION F
  FUNCTION DF(X)
    REAL DF, X
    DF = 3 * X ** 2 + 1
  END FUNCTION DF
END PROGRAM Newton
```

Trong chương trình trên, các hàm trong **F(X)** và **DF(X)** được trả về thông qua lệnh gán ***TenHam*** = …, khác với ví dụ ở mục trước là giá trị của hàm được trả về thông qua biến ở tùy chọn **RESULT**.

Ví dụ 4.4. In một dãy các ký tự giống nhau.

Chương trình sau đây cho phép in ra một dãy các ký tự giống nhau, trong đó số lượng ký tự được cho ở đối số thứ nhất và mã ASCII của ký tự được cho ở đối số thứ hai của thủ tục **DayKyTu**.

```
IMPLICIT NONE
CALL DayKyTu( 5, 65 )  ! 5 chu A lien tuc
CONTAINS
  SUBROUTINE DayKyTu ( Num, Symbol )
    INTEGER I, Num, Symbol
    CHARACTER*80 Line
```

```
   DO I = 1, Num
     Line(I:I) = ACHAR( Symbol )
   END DO
   PRINT*, Line
  END SUBROUTINE
END
```

Câu lệnh **Line(I:I) = ACHAR(Symbol)** trong chương trình con trên có nghĩa là gán ký tự thứ **I** của xâu **Line** bởi ký tự **ACHAR(Symbol)**.

Như đã thấy, thủ tục **DayKyTu** trên đây nhận hai tham số đầu vào là 5 (5 ký tự) và 65 (ký tự thứ 65 trong bảng mã ASCII – chữ A) và truyền cho các đối số tương ứng **Num** và **Symbol**. Kết quả của lời gọi thủ tục này là in ra 5 chữ A liên tục.

Ví dụ 4.5. Tính tổ hợp chập $C_n^k = \frac{n!}{k!(n-k)!}$.

Để tính tổ hợp chập cần phải xây dựng hàm tính giai thừa. Chương trình sau tính và in tổ hợp chập từ 0 đến 10 của 10.

```
PROGRAM TOHOPCHAP
INTEGER I
DO I = 0, 10
  PRINT*, I, Fact(10)/(Fact(I)*Fact(10-I))
END DO
CONTAINS
  FUNCTION Fact ( N )
    INTEGER Fact, N, Temp , I
    Temp = 1
    DO I = 2, N
      Temp = I * Temp
    END DO
    Fact = Temp
  END FUNCTION
END
```

4.6 BIẾN TOÀN CỤC VÀ BIẾN ĐỊA PHƯƠNG

Hãy xét hai chương trình dưới đây, trong đó mục đích của các chương trình này là tính và in lần lượt giai thừa của các số từ 1 đến 10. Ta hãy để ý đến sự khác nhau giữa chúng.

Ví dụ 4.6.

```
PROGRAM VER1
INTEGER I
DO I = 1, 10
  PRINT*, I, Fact(I)
END DO
CONTAINS
  FUNCTION Fact ( N )
    INTEGER Fact, N, Temp
    Temp = 1
```

```
   DO I = 2, N
     Temp = I * Temp
   END DO
   Fact = Temp
  END FUNCTION
END
```

và

```
PROGRAM VER2
INTEGER I
DO I = 1, 10
  PRINT*, I, Fact(I)
END DO
CONTAINS
  FUNCTION Fact ( N )
    INTEGER Fact, N, Temp, I
    Temp = 1
    DO I = 2, N
      Temp = I * Temp
    END DO
    Fact = Temp
  END FUNCTION
END
```

Khi lần lượt chạy các chương trình này ta thấy mặc dù cả hai chương trình này viết *gần như* hoàn toàn giống nhau, nhưng kết quả lại rất khác nhau. Vì sao vậy? Vấn đề ở chỗ là sự có mặt của biến **I** trong câu lệnh khai báo của chương trình con ***Fact***:

INTEGER *Fact*, N, Temp, *I*

Vì chương trình con ***Fact*** là *chương trình con trong*, nên khi biến **I** không được khai báo, nó sẽ sử dụng biến đã được khai báo bởi chương trình chính điều khiển nó. Như vậy, ở chương trình **VER1**, biến **I** đồng thời bị điều khiển bởi cả chương trình chính lẫn chương trình con. Tác động của quá trình dùng chung biến **I** có thể được mô tả như sau.

– Khi trong chương trình chính (**CTC**) biến **I=1**, nó sẽ truyền tham số **N=I=1** cho chương trình con (**FACT**), đồng thời trong **FACT**, biến **I=2, 1** nên **Fact=Tmp=1** (Vòng **DO** không thực hiện). Khi **FACT** trả về **CTC** thì **I=2**.

– Sau khi **FACT** trả về **CTC** thì biến **I** được tăng lên do nó là biến điều khiển của chu trình **DO: I=I+1=2+1=3**, và giá trị này lại truyền cho **FACT**, nên **N=I=3**, đồng thời trong **FACT, I=2, 3** do đó **Fact=1.2.3=6** (Ra khỏi vòng **DO** của **FACT: I=N+1 = 3+1 =4**). Khi **FACT** trả về **CTC** thì **I=4**.

– Tiếp tục, trong **CTC: I=I+1=4+1=5**; khi truyền tham số cho **FACT** thì **N=I=5**, và trong **FACT: I=2, 5** nên **Fact = 1.2.3.4.5 = 120** (Ra khỏi vòng **DO** của **FACT: I=N+1=5+1=6**). **FACT** lại trả về **CTC** giá trị của **I=6**.

Quá trình cứ tiếp diễn như vậy và giai thừa của các số chẵn không được tính cho đến khi xuất hiện lỗi do biến **I** bị rối loạn.

Nhưng, nếu trong chương trình con ta khai báo thêm biến **I**, như ở chương trình **VER2**, thì quá trình tính toán diễn ra chính xác và chương trình kết thúc bình thường.

Biến **I** khai báo trong chương trình chính được gọi là *biến toàn cục*, còn biến **I** khai báo trong chương trình con là *biến địa phương*. Các chương trình con trong *được phép* tham chiếu đến các *biến toàn cục* khi các biến địa phương không được khai báo. Tuy nhiên, chương trình chính sẽ *không tham chiếu được* đến các *biến địa phương* khai báo ở các chương trình con trong.

4.7 ĐỊNH NGHĨA HÀM BẰNG CÂU LỆNH ĐƠN

Hàm có thể được định nghĩa bằng cấu trúc hàm như đã thấy trong mục 4.3.1, nhưng hàm cũng có thể được định nghĩa bằng câu lệnh khai báo hàm. Ta hãy xét ví dụ sau đây.

```
PROGRAM BT_HAM1
REAL X
F(x) = 3*x**2 - 5*x + 2
Print*,' Cho gia tri cua X: '
Read*,x
Print '(" Gia tri ham F(x)=",F10.3)', F(x)
END
```

Trong chương trình trên, câu lệnh

```
F(x) = 3*x**2 - 5*x + 2
```

là một cách định nghĩa hàm **F(x)** và gọi là biểu thức hàm, hay hàm lệnh (*Statement function*), vì nó chỉ dùng một lệnh gán để định nghĩa hàm. Chương trình này tương đương với chương trình sau.

```
PROGRAM BT_HAM2
REAL X
Print*,' Cho gia tri cua X: '
Read*,x
Print '(" Gia tri ham F(x)=",F10.3)', F(x)
CONTAINS
  FUNCTION F(X)
    F = 3*x**2 - 5*x + 2
  END FUNCTION
END
```

Định nghĩa hàm bằng biểu thức hàm cho phép làm đơn giản hóa việc viết chương trình. Tuy nhiên, cách định nghĩa này nhiều lúc gây khó khăn khi gỡ rối chương trình, nhất là đối với những chương trình lớn và đang trong quá trình xây dựng, phát triển. Bởi vậy ta không nên sử dụng cách định nghĩa này khi chương trình chưa thực sự ổn định.

4.8 CHƯƠNG TRÌNH CON NGOÀI

Các chương trình con trong là những chương trình con chỉ do một đơn vị chương trình kiểm soát (chẳng hạn, chương trình chính), chúng khu trú giữa hai câu lệnh **CONTAINS** và **END** của đơn vị chương trình.

Các chương trình con tồn tại ở ngoài dưới dạng các file độc lập được gọi là các chương trình con ngoài. Chúng có thể được tham chiếu bởi nhiều đơn vị chương trình khác nhau. Tuy nhiên, các chương trình con ngoài cũng có thể tồn tại ngay trong cùng một file với chương trình chính hoặc các đơn vị chương trình khác nhưng không nằm giữa các câu lệnh **CONTAINS** và **END**. Trong trường hợp đó, các đơn vị chương trình chứa trong các file khác sẽ không thể tham chiếu trực tiếp đến chúng được.

Các chương trình con ngoài cũng có thể có các chương trình con trong riêng của chúng. Nhưng các *chương trình con trong* lại ***không được phép*** chứa các chương trình con trong khác.

Cú pháp khai báo các chương trình con ngoài có thể có dạng.

1) Khai báo hàm:

```
[KiểuDL][RECURSIVE] FUNCTION TenHam
        ([Các_đối_số]) [RESULT (TenKetQua)]
  [Các_câu_lệnh_khai_báo]
  [Các_câu_lệnh_thực_hiện]
[CONTAINS
  Các_chương_trình_con_trong ]
END [FUNCTION [TenHam] ]
```

2) Khai báo thủ tục:

```
SUBROUTINE TenThuTuc [( Các_đối_số )]
  [Các_câu_lệnh_khai_báo]
  [Các_câu_lệnh_thực_hiện]
[CONTAINS
  Các_chương_trình_con_trong]
END [SUBROUTINE [TenThuTuc] ]
```

Như đã thấy, về cơ bản khai báo chương trình con ngoài tương tự như khai báo chương trình con trong, ngoại trừ các chương trình con ngoài được phép chứa các chương trình con trong, còn các chương trình con trong thì không được phép chứa các chương trình con trong khác. Việc tham chiếu đến các chương trình con ngoài hoàn toàn tương tự như khi tham chiếu đến các chương trình con trong. Nghĩa là giá trị của hàm có thể được gán trực tiếp cho biến hoặc là một bộ phận của biểu thức, còn thủ tục được gọi đến bởi từ khóa **CALL**. Danh sách đối số trong các lời gọi hàm hoặc thủ tục phải là danh sách các đối số thực. Ví dụ, ta có hàm tính tổng hai số nguyên sau đây:

```
INTEGER FUNCTION Tong(a, b) RESULT (X)
Integer a,b
X = a + b
END FUNCTION Tong
```

Khi đó hàm có thể được tham chiếu như sau:

```
PROGRAM GOI_HAM
IMPLICIT NONE
Integer I,j,N,M,Tong
I = 10
J = 23
N = Tong(I, J)
M = N * Tong (N, J)
```

```
print*,N, M
END
```

4.8.1 Câu lệnh EXTERNAL

Để tránh nhầm lẫn trong việc sử dụng các chương trình thư viện của Fortran và chương trình con ngoài có tên trùng nhau, ta **nên** khai báo **tên** các chương trình con ngoài bằng câu lệnh **EXTERNAL**. Ta hãy xét hai ví dụ minh họa sau đây.

Ví dụ 4.7. Chương trình sau đây định nghĩa chương trình con ngoài **COS(X)** có tên trùng với hàm **COS(X)** của thư viện Fortran. Khi chạy chương trình ta sẽ thấy chương trình con này không được gọi tới, mà thay cho nó, hàm **COS(X)** của Fortran sẽ được gọi.

```
PROGRAM EXT1
REAL A
PRINT*,'Cho so A:'
READ*, A
A = COS( A )
PRINT*, A
END
FUNCTION COS( X )
   COS = X + 5.
END FUNCTION
```

Ví dụ 4.8. Cũng chương trình trên đây, nhưng nếu ta thêm câu lệnh

```
EXTERNAL COS
```

vào ngay phần khai báo của chương trình, kết quả là chương trình con ngoài sẽ được gọi tới.

```
PROGRAM EXT2
REAL A
EXTERNAL COS
PRINT*,'Cho so A:'
READ*, A
A = COS( A )
PRINT*, A
END
FUNCTION COS( X )
   COS = X + 5.
END FUNCTION
```

Như vậy, trong quá trình xây dựng chương trình, ta cần phải hết sức thận trọng khi đặt tên cho các *chương trình con ngoài*. Biện pháp an toàn nhất là nếu không chắc chắn tên chương trình con không trùng với tên của các chương trình khác thì nên khai báo chúng bằng câu lệnh **EXTERNAL**.

4.8.2 Khai báo khối giao diện (INTERFACE BLOCK)

Trong nhiều trường hợp trình biên dịch có thể không hiểu ý đồ của ta khi ta muốn sử dụng những chương trình con ngoài có cùng tên (có thể vô tình) với các chương trình con thư viện của Fortran. Để khắc phục tình trạng đó ta có thể sử dụng câu lệnh **EXTERNAL**. Câu lệnh này cung cấp

cho trình biên dịch ***tên*** của chương trình con ngoài, cho phép nó tìm được và liên kết (**LINK**). Tuy nhiên, để trình biên dịch tạo ra lời gọi các chương trình con ngoài một cách chính xác, ngoài tên ra, nó cần phải biết chắc chắn những thông tin về chương trình con, như số biến và kiểu của biến,... Tập hợp những thông tin đó gọi là phần giao diện của chương trình con.

Đối với các chương trình con trong, chương trình con modul, và các chương trình con thư viện của Fortran, phần giao diện luôn được trình biên dịch hiểu. Nhưng khi trình biên dịch phát sinh lời gọi đến một chương trình con ngoài, những thông tin thuộc phần giao diện hoàn toàn chưa sẵn có, tức nó ở trạng thái ẩn (*implicit*), và trong nhiều trường hợp phức tạp (như các đối số tùy chọn hoặc các đối số từ khóa) đòi hỏi phải cung cấp những thông tin giao diện đầy đủ hơn. Do đó cần có khối giao diện.

Khai báo khối giao diện như sau.

```
INTERFACE
  Thân_của_khối_giao_diện
END INTERFACE
```

Trong đó ***Thân_của_khối_giao_diện*** nên được sao chép một cách chính xác phần đầu (*header*) của các chương trình con, cũng như những khai báo đối số và kết quả của chúng, và cả câu lệnh **END** của chúng.

Ví dụ 4.9. Chương trình sau đây sẽ gọi đến một thủ tục ngoài có tên **DOI_CHO**. Không quan trọng thủ tục này nằm trong cùng một file hay khác file với chương trình chính, những thông tin cơ bản về nó cần phải được khai báo trong khối giao diện ngay đầu chương trình chính để nó được tham chiếu một cách chính xác khi chương trình chính phát sinh lời gọi đến nó.

```
IMPLICIT NONE
! Phan khai bao khoi giao dien
INTERFACE
 SUBROUTINE DOI_CHO( X, Y )
 REAL X, Y
 END SUBROUTINE
END INTERFACE
! Phan khai bao bien, hang
REAL A, B
! Than chuong trinh
PRINT*,' CHO 2 SO: '
READ*, A, B
print*,A, B
CALL DOI_CHO ( A, B )
print*,A, B
END
! Chuong trinh con ngoai
SUBROUTINE DOI_CHO ( X, Y ) ! Đổi giá trị của hai biến
REAL X, Y
REAL TMP
  TMP = X
  X = Y
  Y = TMP
```

END SUBROUTINE

4.9 CÁC THUỘC TÍNH CỦA ĐỐI SỐ

4.9.1 Thuộc tính INTENT

Khi thực hiện lời gọi đến một chương trình con, các đối số hình thức sẽ được thay thế bởi các đối số thực của chương trình gọi. Quá trình tương tác giữa các đối số hình thức và đối số thực trong bộ nhớ được mô tả như sau. Trước hết, trình biên dịch sẽ cấp phát bộ nhớ cho tất cả các đối số hình thức và các biến được khai báo trong chương trình con một cách độc lập với bộ nhớ dành cho các biến ở chương trình gọi (trừ những biến, hằng được khai báo dùng chung (**COMMON**) sẽ trình bày sau). Sau đó, nội dung của các đối số thực từ chương trình gọi sẽ được sao chép một cách tương ứng sang các đối số hình thức. Chương trình con sẽ tiến hành quá trình xử lý tính toán trên các biến mà trình biên dịch đã cấp phát bộ nhớ cho nó. Kết thúc tính toán, trước khi trả về chương trình gọi, nội dung của các đối số hình thức sẽ được sao chép lại một cách tương ứng sang các đối số thực. Sau khi trả về kết quả cho chương trình gọi toàn bộ vùng bộ nhớ dành cho các đối số hình thức và các biến của chương trình con sẽ được giải phóng. Như vậy, trong nhiều trường hợp, giá trị của các biến đầu vào từ chương trình gọi có thể bị làm thay đổi một cách không cố ý thông qua việc làm thay đổi giá trị của các đối số hình thức ở chương trình con. Điều đó có thể gây nên những hậu quả rất nghiêm trọng. Để tránh những nhầm lẫn đáng tiếc này, ta có thể sử dụng từ khóa **INTENT** trong phần khai báo của các chương trình con. Cú pháp và tác động của từ khóa này như sau.

INTENT (*Mô_tả*) [::] *vname*

hoặc

Kiểu_DL*, INTENT (*Mô_tả*) :: *vname

Trong đó, ***vname*** là danh sách biến đóng vai trò đối số hình thức của chương trình con; ***Kiểu_DL*** là kiểu dữ liệu của ***vname; Mô_tả*** có thể nhận một trong các giá trị:

IN: Xác định ***vname*** là tham số chỉ truyền vào cho chương trình con, nó không thể bị làm thay đổi giá trị

OUT: Xác định ***vname*** là biến trả giá trị về chương trình gọi, nó cần phải có mặt trong danh sách đối số hình thức

INOUT: Xác định ***vname*** vừa là tham số truyền vào cho chương trình con vừa là biến trả giá trị về cho chương trình gọi, nghĩa là nó vừa cung cấp thông tin đầu vào cho chương trình con vừa có thể trả kết quả về cho chương trình gọi. Do đó giá trị của nó có thể bị làm thay đổi.

Ví dụ, hãy xét chương trình sau:

```
REAL X(20), SUM
CALL RANDOM_NUMBER (X) ! Tạo mảng số ngẫu nhiên X
PRINT*,X          ! X truyền cho chương trình con
CALL TONG (X,20,SUM)
PRINT*,SUM          ! Biến kết quả trả về từ CTCon
PRINT*,X          ! X trả về từ chương trình con

CONTAINS
```

```
  SUBROUTINE TONG (X,N,SUM)
  INTEGER, INTENT (IN) :: N    ! N chỉ IN
  REAL, INTENT (INOUT) :: X(N) ! X vừa IN vừa OUT
  REAL, INTENT (OUT) :: SUM    ! SUM chỉ OUT
  X = X + 10.   ! Làm thay đổi X
  SUM = 0.
  DO I=1,N
    SUM=SUM+X(I)
  END DO
  END SUBROUTINE TONG
END
```

Trong chương trình trên, **RANDOM_NUMBER** là một chương trình con thư viện của Fortran, dùng để tạo bộ số ngẫu nhiên. Đối số của hàm này có thể là biến đơn hoặc biến mảng. Vì biến **N** ở chương trình con là tham số chỉ truyền vào, nên ta không thể làm thay đổi giá trị của nó; **X** vừa là biến truyền vào, vừa là biến kết xuất nên nó có thể bị thay đổi; còn **SUM** chỉ là biến kết xuất.

4.9.2 Thuộc tính OPTIONAL

Một tình huống khác có thể xảy ra khi xây dựng các chương trình con là danh sách các đối số hình thức có thể rất nhiều, nhưng không phải lúc nào ta cũng tham chiếu đến tất cả các đối số này trong lời gọi. Đôi khi ta chỉ cần tham chiếu đến một vài trong số các đối số của chương trình con. Để tránh việc tham chiếu đến những đối số không cần thiết ta có thể khai báo trong chương trình con tất cả hoặc một số đối số có thuộc tính tùy chọn. Từ khóa dùng để khai báo đối số tùy chọn là **OPTIONAL** mà cú pháp và cách sử dụng của nó được mô tả như sau.

OPTIONAL [::] ***vname***

hoặc

Kiểu_DL*, OPTIONAL ::** ***vname

Trong đó, ***vname*** là danh sách biến đóng vai trò đối số hình thức của chương trình con; ***Kiểu_DL*** là kiểu dữ liệu của ***vname.*** Để minh họa ta xét ví dụ sau. Giả sử ta có chương trình con ngoài:

```
SUBROUTINE TONG (X,N,SUM, A, B, C, D, E)
INTEGER, INTENT (IN) :: N
REAL, INTENT (INOUT) :: X(N)
REAL, INTENT (OUT) :: SUM
REAL, OPTIONAL :: A, B, C, D, E ! Các đối số tùy chọn
X = X + 10.
SUM = 0.
DO I=1,N
  SUM=SUM+X(I)
END DO
A = X(1)
B = X(2)
C = A + B
```

```
D = X(3)
E = C * D
END SUBROUTINE TONG
```

Chương trình con ngoài **TONG** chứa 8 đối số hình thức, trong đó có 5 đối số là tùy chọn (có thuộc tính **OPTIONAL**), chúng có thể xuất hiện hoặc không trong lời gọi của chương trình gọi. Giả sử trong chương trình chính ta khai báo khối giao diện cho chương trình này là:

```
INTERFACE
  SUBROUTINE TONG (X,N,SUM, A, B, C, D, E)
    REAL, INTENT (INOUT) :: X(N)
    REAL, OPTIONAL :: A, B, C, D, E
  END SUBROUTINE TONG
END INTERFACE
```

Khi đó chương trình con **TONG** có thể được gọi đến như sau:

CALL TONG (X,N,SUM) ! Bỏ qua tất cả các đối số tùy chọn

CALL TONG (X,N,SUM, T) ! Bỏ qua 4 đối số tùy chọn cuối cùng

CALL TONG (X,N,SUM, T, U) ! Bỏ qua 3 đối số cuối cùng

Trong các trường hợp trên, danh sách các đối số xuất hiện theo trình tự xuất hiện của chúng trong khai báo chương trình con; những đối số bị bỏ qua nằm ở cuối danh sách. Tuy nhiên ta cũng có thể sử dụng lời gọi trong đó các đối số bị bỏ qua có thể ở vị trí bất kỳ. Khi đó cần phải dùng phép "gán" cho những đối số muốn xuất hiện. Chẳng hạn:

CALL TONG (X,N,SUM, B=T) ! Bỏ qua các đối số thứ 4, 6,7,8

CALL TONG (X,N,SUM, C=T, E=U) ! Bỏ qua các đối số thứ 4,5,7

Nếu các đối số tùy chọn đã được "gán" như trên thì những đối số nằm sau nó nhất thiết cũng phải được "gán", dù chúng được tham chiếu theo đúng trình tự xuất hiện. Ví dụ câu lệnh gọi chương trình con sau đây là ***sai***:

CALL TONG (X,N,SUM, B=T, U, V, Y) ! Sai

Câu lệnh đúng sẽ là:

CALL TONG (X,N,SUM, B=T, C=U, D=V, E=Y)

4.9.3 Thuộc tính SAVE

Như đã đề cập ở mục 4.9.1, vùng bộ nhớ cung cấp cho các biến địa phương trong các chương trình con sẽ được giải phóng ngay sau khi chương trình con trả kết quả về cho chương trình gọi. Và như vậy, giá trị của các biến này sẽ trở nên không xác định giữa các lần gọi tới chương trình con. Nếu muốn lưu giữ giá trị của chúng cho lần gọi sau ta có thể đặt thuộc tính **SAVE** cho chúng. Khi một biến nào đó trong chương trình con đã được đặt thuộc tính **SAVE**, giá trị của nó sẽ được bảo lưu cho lần gọi tiếp theo. Cú pháp khai báo thuộc tính **SAVE** như sau:

SAVE [::] ***vname***

hoặc

***Kiểu_DL*, SAVE :: vname**

Trong đó, ***vname*** là danh sách biến địa phương trong chương trình con *không phải là đối số hình thức*; ***Kiểu_DL*** là kiểu dữ liệu của ***vname.***

4.10 MODUL

Fortran định nghĩa 3 khái niệm đơn vị chương trình (Program Unit) là: chương trình chính, chương trình con ngoài, và *modul*. *Modul* khác với các chương trình con ở 2 điểm quan trọng:

– *Modul* có thể chứa trong đó nhiều hơn một chương trình con (được gọi là các chương trình con *modul*);

– *Modul* có thể chứa những câu lệnh khai báo và đặc tả mà chúng có thể tham chiếu được đối với tất cả các đơn vị chương trình có sử dụng *modul*.

Các *modul* cũng có thể được biên dịch một cách độc lập. Cấu trúc chung của *modul* có dạng như sau:

```
MODULE TenModul
  [Các_câu_lệnh_khai_báo]
[CONTAINS
  Các_chương_trình_con_modul]
END [MODULE [TenModul]]
```

Để sử dụng modul ta dùng câu lệnh khai báo **USE** ngay đầu chương trình:

USE *Tên_Modul_được_sử_dụng*

Như vậy, về cơ bản cấu trúc của modul giống như cấu trúc của chương trình con ngoài, trừ các từ khóa **SUBROUTINE** và **FUNCTION**. Modul cũng có thể có các chương trình con trong của chính nó. Modul cũng có thể sử dụng các modul khác. Vì modul có thể chứa các câu lệnh khai báo để tất cả các đơn vị chương trình khác truy cập tới, nên các biến toàn cục có thể được khai báo theo cách này cho những chương trình sử dụng modul. Tính chất này rất hữu ích để tạo ra những khai báo phức tạp, như các kiểu dữ liệu do người dùng định nghĩa,… Các khối giao diện cũng có thể được gộp vào trong các modul.

Ví dụ 4.10. Chương trình sau đây sử dụng modul **MyModul** mà nội dung của nó chứa một tham số **PI** và một thủ tục **DOI_CHO**. Trong chương trình chính ta chỉ cần khai báo **USE MyModul** là đủ để có thể sử dụng tham số **PI** và lời gọi đến thủ tục **DOI_CHO**. Nói chung modul **MyModul** nên được lưu trên một file tách biệt với file chương trình chính.

```
PROGRAM EXAMP
USE MyModul
IMPLICIT NONE
REAL A, B
PRINT*, ‘ Cho mot so: ‘
READ*, A
B = Pi          ! Khai bao tu Modul
CALL DOI_CHO( A, B ) ! Khai bao tu Modul
```

```
PRINT*, A, B
END
MODULE MyModul
REAL, PARAMETER :: Pi = 3.1415927
CONTAINS
 SUBROUTINE DOI_CHO ( X, Y )
   REAL Tmp, X, Y
   Tmp = X
   X = Y
   Y = Tmp
 END SUBROUTINE DOI_CHO
END MODULE MyModul
```

4.11 PHÉP ĐỆ QUI

Trong nhiều trường hợp phép đệ qui cho phép làm đơn giản hoá chương trình một cách đáng kể. Có thể xem phép đệ qui là một hàm hay một thủ tục có thể tham chiếu đến chính nó. Phép đệp qui bao gồm hai thành phần: Phần neo, trong đó tác động của hàm hay thủ tục được đặc tả cho một hay nhiều tham số, và phần đệ qui trong đó tác động cần được thực hiện cho giá trị hiện thời của tham số được xác định bằng các tác động hay giá trị được định nghĩa trước đó.

Ví dụ điển hình cho phép đệ qui là hàm hoặc thủ tục tính giai thừa. Xuất phát từ định nghĩa n! ta có: 0! = 1! = 1, với mọi n>0 thì n! = n.(n–1)!. Đây là một công thức truy hồi, tức là khi đã biết (n–1)! thì có thể tính được n!. Ta có hàm và thủ tục tính n! như sau:

```
RECURSIVE FUNCTION GIAITHUA1 ( N ) RESULT (Fact)
INTEGER Fact, N
IF( N == 0 .OR. N == 1 ) THEN   ! Phan neo
   Fact = 1
ELSE                        ! phan de qui
   Fact = N * GIAITHUA1( N-1 )
END IF
END FUNCTION
```

Hoặc

```
RECURSIVE SUBROUTINE GIAITHUA2( F, N )
INTEGER F, N
IF (N == 0 .OR. N == 1) THEN    ! Phan neo
   F = 1
ELSE                        ! Phan de qui
   CALL GIAITHUA2( F, N-1 )
   F = N * F
END IF
END SUBROUTINE
```

Có thể giải thích tác động đệ qui này như sau. Giả sử để tính 3!, ta gọi **GIAITHUA1(3)** (hoặc **GIAITHUA2(F,3)**). Lời gọi này sẽ tham chiếu đến **GIAITHUA1(2)** (hoặc **GIAITHUA2(F,2)**), rồi **GIAITHUA1(2)** (hoặc **GIAITHUA2(F,2))** lại tham chiếu đến **GIAITHUA1(1)** (hoặc **GIAITHUA2(F,1))** là phần neo (**IF (N==0 .OR. N==1) THEN...**).

Một ví dụ khác, trung bình số học của một dãy số $x_1, x_2,..., x_n$ có thể được tính theo công thức: $\overline{x_n} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Ta có thể biểu diễn công thức này dưới dạng khác: $\overline{x_n} = ((n-1).\overline{x_{n-1}} + x_n)/n$, trong đó $\overline{x_{n-1}}$ là trung bình của $n-1$ thành phần đầu của dãy. Phần neo có thể xác định bởi định nghĩa sau:

– Số thành phần $n>0$

– Nếu $n=1$ thì $\overline{x_n} = x_1$,

– Nếu $n=2$ thì $\overline{x_n} = (x_1 + x_2)/2$

Bạn đọc hãy viết chương trình con đệ qui cho bài toán này như là một bài tập.

BÀI TẬP CHƯƠNG 4

4.1 Làm các bài tập 3.5 và 3.6 chương 3 trong đó các hàm $f(x)$ được viết dưới dạng các chương trình con.

4.2 Viết chương trình tính sine của x theo công thức:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$$

với độ chính xác $\varepsilon=10^{-4}$. Sử dụng chương trình con hàm để tính giai thừa.

4.3 Viết chương trình tính cosine của x theo công thức:

$$\cos x = \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + ...$$

với độ chính xác $\varepsilon=10^{-4}$. Sử dụng chương trình con hàm để tính giai thừa.

4.4. Viết chương trình tìm nghiệm của phương trình $x^2 sinx + cos2x = 0$ trên đoạn $[-2; 2]$ bằng phương pháp chia đôi. Yêu cầu xây dựng chương trình con hàm hoặc thủ tục. Lấy độ chính xác của nghiệm đến 10^{-6}. **Gợi ý:** Sử dụng định lý: Hàm số $y=f(x)$ có $f(a).f(b)<0$ thì tồn tại $x=c$ thuộc $(a;b)$ sao cho $f(c)=0$.

4.5 Viết chương trình tính đạo hàm của hàm số $y=f(x)$ tại $x=x_0$ với độ chính xác đến 10^{-6}. Yêu cầu xây dựng chương trình con hàm để tính $f(x)$. Lấy ví dụ $f(x)=2x^2$, $x_0=1$. **Gợi ý:** Đạo hàm của hàm số $y=f(x)$ tại $x=x_0$ có thể được tính theo công thức $(f(x_0+h) - f(x_0))/h$ khi $h \to 0$.

4.6 Viết chương trình con hàm tính e^x theo công thức khai triển chuỗi Taylor: $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$ So sánh kết quả tính với kết quả của lời gọi hàm thư viện EXP của Fortran. Lấy độ chính xác bằng 10^{-6}.

4.7 Viết chương trình xây dựng hàm tính tổ hợp chập *k* của *n* theo công thức: $C_n^k = \frac{n!}{k!(n-k)!}$.

Chương trình cho phép kiểm tra các lỗi vào dữ liệu không hợp lệ. Ví dụ, khi cho *k* hoặc *n* là những số âm, hoặc khi *k>n*, chương trình sẽ đưa ra thông báo lỗi "Dữ liệu không hợp lệ" và kết thúc.

4.8 Viết chương trình con dạng thủ tục giải phương trình $ax^2 + bx + c = 0$. Chương trình cho phép đưa ra nghiệm ảo khi biệt thức *Delta < 0*.

4.9 Viết chương trình con dạng thủ tục xác định dãy *n* số Fibonacci (xem bài tập 3.13).

4.10 Hàm Laplas được định nghĩa bởi $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_0^x e^{-0.5t^2}\,dt$. Viết chương trình con dạng hàm tính giá trị của Φ(x). Thử chạy chương trình với một vài giá trị của x, chẳng hạn x=1.96, x=3.0.

4.11 Ba hàm đầu tiên của đa thức Legendre có dạng $P_0(x)=1$, $P_1(x)=x$, $P_2(x)=(3x^2-1)/2$. Công thức truy hồi để xác định các hàm của đa thức Legendre là

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0.$$

Ký hiệu hàm thứ *n* của đa thức Legendre là P(N, X) = $P_n(x)$. Viết chương trình con dạng thủ tục xác định giá trị các hàm $P_1(x),..., P_n(x)$ của đa thức khi cho giá trị của *n* và *x*.

4.12 Cũng với các điều kiện như ở bài tập 4.11. Hãy viết chương trình con dạng hàm tính giá trị của $P_n(x)$. Thử chạy chương trình với *n=2* và các giá trị khác nhau của *x*.

CHƯƠNG 5. MẢNG

5.1 KHÁI NIỆM VỀ MẢNG TRONG FORTRAN

Có thể định nghĩa mảng là một tập hợp các phần tử có cùng kiểu dữ liệu, được sắp xếp theo một trật tự nhất định, trong đó mỗi phần tử được xác định bởi chỉ số và giá trị của chúng. Chỉ số của mỗi phần tử mảng được xem là "địa chỉ" của từng phần tử trong mảng mà nó được dùng để truy cập/tham chiếu đến phần tử của mảng. Mỗi phần tử của mảng được xác định bởi duy nhất một "địa chỉ" trong mảng. Mảng có thể là mảng một chiều hoặc nhiều chiều. Mảng một chiều có thể hiểu là một vectơ mà mỗi phần tử mảng là một thành phần của vectơ. Địa chỉ các phần tử mảng một chiều được xác định bởi một chỉ số là số thứ tự của chúng trong mảng. Mảng hai chiều được hiểu như một ma trận mà địa chỉ các phần tử của nó được xác định bởi hai chỉ số: chỉ số thứ nhất là số thứ tự hàng, chỉ số thứ hai là số thứ tự cột. Tương tự, mảng ba chiều được xem như là tập hợp các mảng hai chiều, trong đó các phần tử mảng được xác định bởi ba chỉ số: chỉ số thứ nhất, chỉ số thứ hai (tương ứng là hàng và cột của một ma trận) và chỉ số thứ ba (lớp – số thứ tự của ma trận),...

Kiểu dữ liệu của các phần tử mảng có thể là kiểu số hoặc không phải số. Mỗi mảng được xác định bởi tên mảng, số chiều, kích thước cực đại và cách sắp xếp các phần tử của mảng. Tên mảng còn gọi là tên biến mảng, hay ngắn gọn hơn là biến mảng. Biến mảng là biến có ít nhất một chiều.

Mảng có thể là mảng tĩnh hoặc mảng động. Nếu là mảng tĩnh thì vùng bộ nhớ dành lưu trữ mảng là cố định và nó không bị giải phóng chừng nào chương trình còn hiệu lực. Kích thước của mảng tĩnh không thể bị thay đổi trong quá trình chạy chương trình. Nếu mảng là mảng động, vùng bộ nhớ lưu trữ nó có thể được gán, thay đổi và giải phóng khi chương trình đang thực hiện.

Các con trỏ (**POINTER**) là những biến động. Nếu con trỏ cũng là mảng thì kích thước của mỗi chiều cũng có thể bị thay đổi trong lúc chương trình chạy, giống như các mảng động. Các con trỏ có thể trỏ đến các biến mảng hoặc biến vô hướng.

5.2 KHAI BÁO MẢNG

Để sử dụng mảng nhất thiết cần phải khai báo nó. Khi khai báo mảng cần phải chỉ ra tên và số chiều của nó, nhưng có thể chưa cần chỉ ra kích thước và cách sắp xếp các phần tử mảng. Có rất nhiều cách khai báo biến mảng. Sau đây sẽ liệt kê một số trường hợp ví dụ.

```
REAL A(10, 2, 3)      ! Mảng các số thực 3 chiều
DIMENSION A(10, 2, 3) ! Mảng các số thực 3 chiều
ALLOCATABLE B(:, :)   ! Mảng các số thực 2 chiều
```

POINTER C(:, :, :) ! Mảng các số thực 3 chiều
REAL,DIMENSION (2,5) :: D ! Mảng các số thực 2 chiều
REAL,ALLOCATABLE :: E(:,:,:)! Mảng thực 3 chiều
REAL, POINTER :: F(:,:) ! Mảng các số thực 2 chiều

Trong các ví dụ trên, mảng **A(10, 2, 3)** là mảng ba chiều gồm các phần tử có kiểu số thực loại 4 byte, kích thước cực đại của mảng là 10 x 2 x 3 = 60 phần tử, dung lượng bộ nhớ cấp phát cho mảng là 60 x 4 (byte) = 240 byte, cách sắp xếp các phần tử là 10 hàng, 2 cột và 3 lớp, địa chỉ các hàng, cột và lớp được đánh số từ 1 (hàng 1 đến hàng 10, cột 1 đến cột 2, lớp 1 đến lớp 3). Mảng **B** là mảng động 2 chiều, trong đó kích thước và cách sắp xếp các phần tử chưa được xác định. Mảng **C** là mảng thực ba chiều có kiểu con trỏ. Ta cũng thấy rằng, có thể chỉ sử dụng các từ khóa khai báo kiểu, khai báo thuộc tính để định nghĩa mảng, nhưng cũng có thể kết hợp cả các từ khóa khai báo kiểu và khai báo thuộc tính. Khi có sự kết hợp giữa khai báo kiểu và khai báo thuộc tính, giữa chúng cần phải phân tách nhau bởi dấu phẩy và sau từ khóa thuộc tính phải có hai dấu hai chấm liền nhau phân tách chúng với tên biến. Số chiều, kích thước và cách sắp xếp phần tử mảng có thể được định nghĩa cùng với từ khóa thuộc tính hoặc tên biến.

Cách đánh số địa chỉ các phần tử mảng cũng là một trong những đặc điểm hết sức quan trọng, vì nó quyết định cách truy cập đến các phần tử mảng. Chỉ số xác định địa chỉ các phần tử mảng phụ thuộc vào giới hạn dưới và giới hạn trên dùng để mô tả cách sắp xếp các phần tử theo các chiều của mảng. Ví dụ, hai mảng

```
INTEGER M(10, 10, 10)
INTEGER K(-3:6, 4:13, 0:9)
```

đều có cùng kích thước (10 x 10 x 10), nhưng mảng **M** có chỉ số các phần tử mảng theo cả ba chiều biến thiên từ 1 đến 10 (giới hạn dưới bằng 1, giới hạn trên bằng 10), còn mảng **K** có chỉ số các phần tử mảng biến thiên theo chiều thứ nhất (hàng) là –3 đến 6, theo chiều thứ hai (cột) là 4 đến 13 và theo chiều thứ ba (lớp) là 0 đến 9. Như vậy, giới hạn dưới của chỉ số các phần tử của mảng **K** tương ứng là –3, 4 và 0, còn giới hạn trên là 6, 13 và 9. Các mảng được mô tả rõ ràng như vậy được gọi là các mảng có mô tả tường minh.

Đối với các mảng mô tả không tường minh, cách sắp xếp và đánh số địa chỉ các phần tử mảng thường được xác định trong lúc chương trình chạy hoặc sẽ được truyền qua tham số của các chương trình con. Ví dụ:

```
REAL X (4, 7, 9)
...
CALL SUB1(X)
CALL SUB2(X)
...
END
SUBROUTINE SUB1(A)
REAL A(:, :, :)
...
```

```
END SUBROUTINE SUB1
SUBROUTINE SUB2(B)
REAL B(3:, 0:, -2:)
...
END SUBROUTINE SUB2
```

Ở đây, mảng **A** trong chương trình con **SUB1** sẽ là:

A (4, 7, 9)

còn mảng **B** trong chương trình con **SUB2** sẽ là:

B (3:6, 0:6, -2:6)

Nói chung có thể có nhiều cách khai báo mảng khác nhau tùy thuộc vào yêu cầu và bối cảnh cụ thể. Sau đây là một số dạng cú pháp tổng quát của câu lệnh khai báo mảng thường được sử dụng trong lập trình.

Dạng 1:

***Kiểu_DL* Tên_biến_mảng (*Mô_tả*)**

Dạng 2:

Thuộc_tính Tên_biến_mảng (*Mô_tả*)

Dạng 3:

***Kiểu_DL*, Thuộc_tính (*Mô_tả*) :: Tên_biến_mảng**

Dạng 4:

***Kiểu_DL*, Thuộc_tính :: Tên_biến_mảng(*Mô_tả*)**

Trong đó ***Kiểu_DL*** là kiểu dữ liệu của các phần tử mảng, **Thuộc_tính** có thể là một trong các thuộc tính **DIMENSION, ALLOCATABLE, POINTER**,…, **Tên_biến_mảng** là tên của các biến mảng (nếu có nhiều hơn một biến thì chúng được liệt kê cách nhau bởi các dấu phẩy), ***Mô_tả*** là mô tả số chiều, kích thước mảng và cách sắp xếp các phần tử mảng. Nếu là mô tả ẩn thì cách sắp xếp các phần tử mảng chưa cần chỉ ra trong khai báo biến mảng. Ví dụ:

Dạng 1:

```
REAL*4 X (0:100)
REAL Y(12,34)
```

Dạng 2:

```
DIMENSION N (10,20)
ALLOCATABLE Y(:,:)
```

Dạng 3:

```
REAL, ALLOCATABLE(:,:) :: X
INTEGER, DIMENSION(12,34) :: Y
```

Dạng 4:

```
REAL, ALLOCATABLE :: X (:,:)
REAL, DIMENSION Y(12,34)
```

5.3 LƯU TRỮ MẢNG TRONG BỘ NHỚ VÀ TRUY CẬP ĐẾN CÁC PHẦN TỬ MẢNG

Nguyên tắc lưu trữ mảng trong bộ nhớ của Fortran là lưu trữ dưới dạng vectơ, cho dù đó là mảng một chiều hay nhiều chiều. Đối với mảng một chiều, các phần tử mảng được sắp xếp theo thứ tự từ phần tử có địa chỉ mảng (chỉ số) nhỏ nhất đến phần tử có địa chỉ lớn nhất. Các phần tử của mảng hai chiều cũng được xếp thành một vectơ, trong đó các "đoạn" liên tiếp của vectơ này là các cột với chỉ số cột tăng dần. Các mảng ba chiều được xem là tập hợp các mảng hai chiều với số thứ tự của các mảng hai chiều này (số thứ tự lớp) chính là chỉ số thứ ba của mảng. Các mảng nhiều chiều hơn cũng được lưu trữ theo nguyên tắc này. Nói chính xác hơn, tùy thuộc vào số chiều của mảng mà khi sắp xếp các phần tử mảng, chỉ số của chiều thứ nhất biến đổi trước, tiếp đến là chiều thứ hai, chiều thứ ba,... Các phần tử mảng được truy cập đến qua địa chỉ của chúng trong mảng.

Để rõ hơn ta xét một số ví dụ sau.

Ví dụ 5.1. Mảng một chiều.

Giả sử ta khai báo

```
REAL X(5), Y(0:5)
```

Khi đó các mảng **X** và **Y** được sắp xếp trong bộ nhớ như sau:

	X(1)	X(2)	X(3)	X(4)	X(5)	
Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	

Chương trình sau đây minh họa cách truy cập đến các phần tử của các mảng này.

```
REAL X(5), Y(0:5)
Y(0) = 1.
DO I=1,5
  X(I) = I*I     ! Gán giá trị cho các phần tử của X
  Y(I) = X(I) + I
       ! Nhận giá trị các phần tử của X, tính toán
       ! và gán cho các phần tử của Y
END DO
PRINT '(6F7.1))', (X(I), I=1,5) ! In các phần tử của X
PRINT '(6F7.1))', (Y(I), I=0,5) ! In các phần tử của Y
END
```

Khi chạy chương trình này ta sẽ nhận được kết quả trên màn hình là:

```
1.0   4.0   9.0   16.0   25.0
1.0   2.0   6.0   12.0   20.0   30.0
```

Ví dụ 5.2. Mảng hai chiều.

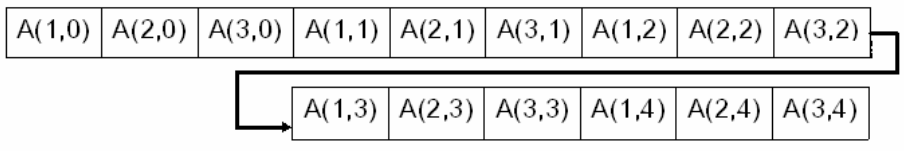
Giả sử ta khai báo

```
INTEGER, PARAMETER :: N=3, M=4
INTEGER  A(N, 0:M)
```

Khi đó có thể hiểu mảng A như là một ma trận gồm 3 hàng, 5 cột:

$$A=\begin{pmatrix} A(1,0) & A(1,1) & A(1,2) & A(1,3) & A(1,4) \\ A(2,0) & A(2,1) & A(2,2) & A(2,3) & A(2,4) \\ A(3,0) & A(3,1) & A(3,2) & A(3,3) & A(3,4) \end{pmatrix}$$

A được lưu trữ trong bộ nhớ dưới dạng:

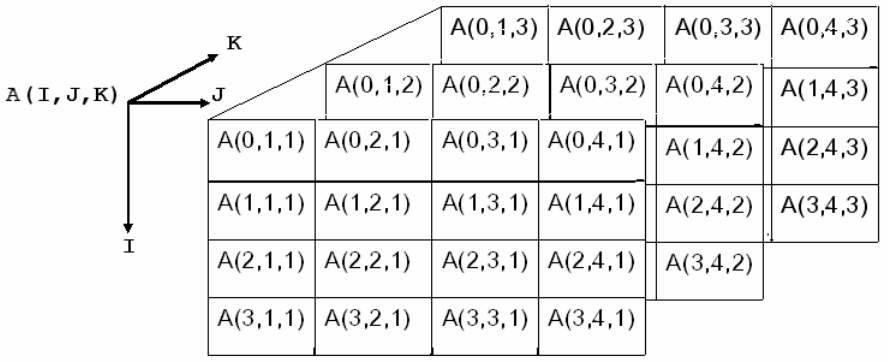


Ví dụ 5.3. Mảng ba chiều.

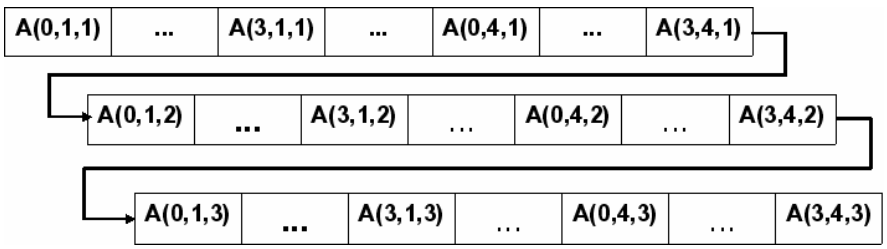
Giả sử mảng A được khai báo bởi

INTEGER, PARAMETER :: NH=3, MC=4, LLayer=3
INTEGER A(0:NH, MC, LLayer)

Khi đó **A** là mảng ba chiều gồm 4 hàng, 4 cột và 3 lớp, có cấu trúc như sau:



Và được lưu trữ trong bộ nhớ dưới dạng:



Sau đây là một số ví dụ truy cập mảng.

Nếu mảng **A** được khai báo bởi

REAL A(5,10), B(5,10)

Khi đó:

A = 3.0 **!** Gán tất cả các phần tử của A bằng 3
A(1,1) = 4. ! Gán phần tử hàng 1, cột 1 bằng 4.,
A(1,2) = 7. ! Gán phần tử hàng 1, cột 2 bằng 7.

A(2,1:8:3)=2.5 ! Gán các phần tử cột 1, 4, 7 hàng 2 bằng 2.5. Tức là A(2,1) = A(2,4) = A(2,7) = 2.5. Các chỉ số **1:8:3** của chiều thứ hai tương đương với vòng lặp **DO J=1, 8, 3**

B = SQRT(A) ! Gán tất cả các phần tử của B bằng
! căn bậc hai các phần tử tương ứng của A

Nếu khai báo

```
REAL A(10)
```

Khi đó:

```
A(1:5:2)=3.0
    !Gán các phần tử A(1), A(3), A(5) bằng 3.0.
A(:5:2)=3.0   ! Tương tự câu lệnh trên
A(2::3)=3.0
    ! Gán các phần tử A(2), A(5), A(8) bằng 3.0.
```

(Chỉ số cao nhất ngầm định bằng 10 là kích thước cực đại của **A**, tương đương với vòng lặp **DO I=2, 10, 3**)

A(7:9) = 3.0 ! Gán các phần tử A(7), A(8), A(9) bằng 3.0.
! (Bước vòng lặp ngầm định bằng 1)

A(:) = 3.0 ! Tương tự như A = 3.0;

Một ví dụ khác, nếu có khai báo

```
REAL A(10), B(5, 5)
INTEGER I(4), J(3)
```

ta có thể gán giá trị cho các phần tử của các mảng **I** và **J** bằng cách:

```
I = (/ 5, 3, 8, 2 /)
J = (/ 3, 1, 5 /)
```

Còn câu lệnh

```
A(I) = 3.0
```

có nghĩa là gán các phần tử A(5), A(3), A(8) và A(2) bằng 3.0, và câu lệnh

```
B(2,J) = 3.0
```

là gán các phần tử B(2,3), B(2,1) và B(2,5) bằng 3.

Qua các ví dụ trên ta có thể thấy cách truy cập đến các phần tử mảng của Fortran rất mềm dẻo và linh hoạt. Đó cũng là một trong những "thế mạnh" của ngôn ngữ lập trình này.

5.3.1 Sử dụng lệnh DATA để khởi tạo mảng

Trong một số trường hợp, dữ liệu ban đầu có thể được gán trực tiếp cho các phần tử mảng ngay trong chương trình mà không nhất thiết nhận từ file. Một trong những cách gán đó là sử dụng câu lệnh gán thông thường. Tuy nhiên cách làm này không hiệu quả, vì phải lặp lại nhiều lần lệnh gán, làm "giãn dài" chương trình một cách không cần thiết. Thay cho việc sử dụng những câu lệnh gán đó, ta cũng có thể sử dụng câu lệnh **DATA** để gán giá trị cho các phần tử mảng. Ví dụ:

REAL, DIMENSION(10) :: A, B, C(3,3)

DATA A / 5*0, 5*1 /

! Gán 5 phần tử đầu bằng 0 và 5 phần tử tiếp theo bằng 1

DATA B(1:5) / 4, 0, 5, 2, -1 /

! Chỉ gán giá trị cho các phần tử từ B(1) đến B(5)

DATA ((C(I,J), J= 1,3), I=1,3) /3*0,3*1, 3*2/

! Gán giá trị cho các phần tử của C lần lượt theo hàng

Điều chú ý khi sử dụng lệnh **DATA** để gán giá trị cho các phần tử mảng là số giá trị sẽ gán (nằm giữa hai dấu gạch chéo (/)) phải bằng kích thước khai báo của mảng. Nếu có nhiều giá trị bằng nhau lặp lại liên tiếp ta có thể sử dụng cách gán ***n*value,*** trong đó ***n*** là số lần lặp lại liên tiếp, ***value*** là giá trị được lặp lại. Trật tự sắp xếp các giá trị sẽ gán phải phù hợp với trật tự truy cập đến phần tử mảng. Chẳng hạn, câu lệnh sau đây:

DATA ((C(I,J), J= 1,3), I=1,3) /3*0,3*1, 3*2/

sẽ cho kết quả gán là C(1,1) = C(1,2) = C(1,3) = 0; C(2,1) = C(2,2) = C(2,3) = 1; C(3,1) = C(3,2) = C(3,3) = 2. Còn câu lệnh:

DATA C / 3*0, 3*1, 3*2 /

sẽ cho kết quả gán là C(1,1) = C(2,1) = C(3,1) = 0; C(1,2) = C(2,2) = C(3,2) = 1; C(1,3) = C(2,3) = C(3,3) = 2. Sở dĩ như vậy là vì, ở câu lệnh thứ nhất, các phần tử được truy cập lần lượt từng hàng, từ hàng 1 đến hàng 3, trong khi ở câu lệnh thứ hai, do ta không chỉ ra cụ thể nên Fortran ngầm hiểu là các phần tử của mảng được truy cập lần lượt theo cột, từ cột 1 đến cột 3.

5.3.2 Biểu thức mảng

Có thể thực hiện các phép toán trên các biến mảng. Trong trường hợp này các mảng phải có cùng cấu trúc. Ví dụ:

REAL, DIMENSION(10) :: X, Y

X + Y ! Cộng tương ứng các phần tử của X và Y: X(I) + Y(I)

X * Y ! Nhân tương ứng các phần tử của X và Y: X(I) * Y(I)

X * 3 ! Nhân tương ứng các phần tử của X với 3: X(I) * 3

X * SQRT(Y) ! Nhân các phần tử của X với căn bậc 2 của

! các phần tử tương ứng của Y: X(I) * SQRT(Y(I))

X == Y ! Phép toán so sánh, cho kết quả **.TRUE.** nếu

! **X(I) == Y(I)**, và **.FALSE.** nếu ngược lại.

5.3.3 Cấu trúc WHERE... ELSEWHERE ... END WHERE

Đây là cấu trúc chỉ dùng trong thao tác với các mảng. Cú pháp câu lệnh như sau.

WHERE (*Điều_kiện*) Câu_lệnh

Hoặc

```
WHERE (Điều_kiện)
  Các_câu_lệnh_1
ELSEWHERE
  Các_câu_lệnh_2
END WHERE
```

Tác động của câu lệnh là tìm các phần tử trong mảng thỏa mãn ***Điều_kiện***, nếu ***Điều_kiện*** được thỏa mãn thì thực hiện ***Các_câu_lệnh_1***, ngược lại thì thực hiện ***Các_câu_lệnh_2***. ***Điều_kiện*** ở đây là một biểu thức lôgic. Ví dụ:

```
REAL A (5)
A = (/ 89.5, 43.7, 126.4, 68.3, 137.7 /)
WHERE (A > 100.0) A = 100.0
```

Trong đoạn chương trình trên, tất cả các phần tử của mảng **A** có giá trị > 100 sẽ được thay bằng 100. Kết quả sẽ nhận được:

```
A = (89.5, 43.7, 100.0, 68.3, 100.0)
```

Một ví dụ khác:

```
REAL A (5), B(5), C(5)
A = (/ 89.5, 43.7, 126.4, 68.3, 137.7 /)
B = 0.0
C = 0.0
WHERE (A > 100.0)
   A = 100.0
   B = 2.3
ELSEWHERE
   A = 50.0
   C = -4.6
END WHERE
```

Ở đây, kết quả nhận được là

Mảng	PT thứ 1	PT thứ 2	PT thứ 3	PT thứ 4	PT thứ 5
A	50.0	50.0	100.0	50.0	100.0
B	0.0	0.0	2.3	0.0	2.3
C	–4.6	–4.6	0.0	–4.6	0.0

5.4 MẢNG ĐỘNG (DYNAMICAL ARRAY)

Mảng có kích thước và cách sắp xếp các phần tử không được xác định ngay từ lúc khai báo gọi là mảng động. Các mảng động luôn phải có thuộc tính **ALLOCATABLE** trong câu lệnh khai báo. Trên đây ta đã gặp một số ví dụ về khai báo và sử dụng mảng động. Một cách tổng quát, có thể có các cách khai báo như sau.

Kiểu_DL*,DIMENSION(*Mô_tả*),ALLOCATABLE :: *Tên_biến

hoặc

***Kiểu_DL*, ALLOCATABLE [::] *Tên_biến* [(*Mô_tả*)]**

hoặc

ALLOCATABLE [::] ***Tên_biến*** **[(*Mô_tả*)]**

Trong đó ***Mô_tả*** là mô tả số chiều của mảng, được xác định bởi các dấu hai chấm (:), phân cách nhau bằng dấu phẩy. Ví dụ:

```
REAL,DIMENSION(:),ALLOCATABLE :: X   ! Mảng 1 chiều
REAL, ALLOCATABLE ::  vector(:)      ! Mảng 1 chiều
INTEGER,ALLOCATABLE :: matrix(:,:)  ! Mảng 2 chiều
DIMENSION X (:,:)                    ! X là mảng hai chiều và
REAL, ALLOCATABLE :: X               ! X là mảng động, thực
ALLOCATABLE :: Y(:,:)                ! Y là mảng động 2 chiều
```

Vì các mảng động chưa được xác định kích thước ngay từ đầu nên để sử dụng ta cần phải mô tả rõ kích thước và cách sắp xếp các phần tử của chúng trước khi truy cập.

Câu lệnh **ALLOCATE** dùng để định vị kích thước và cách sắp xếp các phần tử mảng trong bộ nhớ (tức cấp phát bộ nhớ cho biến).

Câu lệnh **DEALLOCATE** dùng để giải phóng vùng bộ nhớ mà biến mảng động đã được cấp phát.

Ví dụ 5.4. Xét đoạn chương trình

```
INTEGER, ALLOCATABLE :: matrix(:,:)
REAL, ALLOCATABLE :: vector(:)
...
N = 123
ALLOCATE (matrix(3,5),vector(-2:N+2))
...
DEALLOCATE matrix, vector
```

Trong đoạn chương trình trên, ***vector*** và ***matrix*** lần lượt là các mảng động một chiều và hai chiều. Sau câu lệnh **ALLOCATE**, ***matrix*** được cấp phát một vùng nhớ gồm 3 hàng x 5 cột x 4 byte = 60 byte với cách đánh số địa chỉ các phần tử mảng bắt đầu từ 1 đến 3 (hàng) và 1 đến 5 (cột). Còn ***vector*** được cấp phát một vùng nhớ gồm (N+2 – (–2) + 1) phần tử x 4 byte = (123+2+2+1) x 4 byte = 512 byte, với địa chỉ các phần tử mảng được đánh số từ –2 đến 125.

Ví dụ 5.5. Cấp phát bộ nhớ cho mảng tuỳ thuộc tham số xác định được trong quá trình thực hiện chương trình

```
REAL A, B(:,:), C(:), D(:, :, :)
ALLOCATABLE C, D
....
READ (*, *) N, M
ALLOCATE (C(N), D(M, N, M))
```

Trong ví dụ này, kích thước các mảng C và D sẽ được xác định chỉ sau khi các giá trị của N và M đã xác định.

Ví dụ 5.6. Sử dụng hàm **ALLOCATED** để xác định mảng đã được cấp phát bộ nhớ hay chưa

```
REAL, ALLOCATABLE :: A(:)
...
IF (.NOT. ALLOCATED(A)) ALLOCATE (A (5))
```

Trong ví dụ này, mảng A sẽ được cấp phát bộ nhớ nếu nó chưa được cấp phát.

Ví dụ 5.7. Bẫy lỗi trong quá trình cấp phát bộ nhớ cho mảng

```
REAL, ALLOCATABLE :: A(:)
INTEGER ERR
ALLOCATE (A (5), STAT = ERR)
IF (ERR /= 0) PRINT *,“Khong cap phat duoc"
```

Ở đây, tham số **STAT** trong câu lệnh **ALLOCATE** sẽ trả về giá trị **ERR** (số nguyên). Nếu **ERR**=0 thì việc cấp phát bộ nhớ thực hiện thành công, ngược lại nếu không cấp phát được thì giá trị của **ERR** chính là mã lỗi lúc chạy chương trình.

Ví dụ 5.8. Chương trình sau đây nhập một mảng một chiều **X** gồm các số thực dương nhưng không biết trước số phần tử của mảng tối đa là bao nhiêu. Do đó mảng **X** sẽ được cấp phát bộ nhớ tăng dần trong khi nhập dữ liệu. Quá trình nhập dữ liệu chỉ kết thúc khi số nhập vào là một số âm. Thủ thuật thực hiện ở đây là sử dụng 2 mảng động, trong đó một mảng để lưu số liệu trung gian.

```
REAL, DIMENSION(:), ALLOCATABLE :: X, OldX
REAL     A
INTEGER   N
ALLOCATE (X(0)) ! Kích thước của X (lúc đầu bằng 0)
N = 0
DO
   Print*, ‘Cho mot so: ‘
   READ(*,*) A
   IF ( A < 0 ) EXIT        ! Nếu A<0 thì thoát
   N = N + 1                              ! Tăng N lên 1 đơn vị
   ALLOCATE(OldX(SIZE(X))) ! Cấp phát kích thước của
                                    ! OldX bằng kích thước của X
   OldX = X        ! Lưu X vào OldX
   DEALLOCATE( X ) ! Giải phóng X
   ALLOCATE(X(N))  ! Cấp phát X có kích thước bằng N
   X = OldX     ! Gán toàn bộ OldX cho X
   X(N) = A     !Gán giá trị mới cho phần tử thứ N của X
   DEALLOCATE( OldX ) ! Giải phóng OldX
END DO
PRINT*,N, ( X(I), I = 1, N )
END
```

Hàm **SIZE(X)** trong chương trình là để xác định kích thước hiện tại của mảng **X**.

5.5 KIỂU CON TRỎ

Con trỏ là một khái niệm để xác định biến có thuộc tính con trỏ. Biến con trỏ có thể là biến vô hướng hoặc biến mảng. Khai báo kiểu con trỏ như sau:

POINTER [::] *Tên_con_trỏ* [(*Mô_tả*)] [, ...]

hoặc

***Kiểu_DL*, POINTER :: Tên_con_trỏ [(*Mô_tả*)]**

Trong đó ***Tên_con_trỏ*** là tên biến có kiểu con trỏ; nếu tên biến là tên của biến mảng thì cần phải khai báo ***Mô_tả*** mảng. Ví dụ, có thể khai báo biến con trỏ như sau.

REAL A, X(:,:), B, Y(5, 5)

POINTER A, X ! A là con trỏ vô hướng, X là con trỏ mảng

hoặc

REAL, POINTER :: A (:,:)

REAL B, X(:,:)

POINTER B, X

Biến con trỏ có thể được cấp phát bộ nhớ bằng lệnh **ALLOCATE** hoặc trỏ đến một biến khác. Biến được con trỏ trỏ đến hoặc là một biến có thuộc tính đích (**TARGET**) hoặc một biến đã được xác định. Trong trường hợp biến con trỏ trỏ đến một biến khác, nó được xem như "bí danh" của biến mà nó trỏ đến. Để minh họa ta hãy xét ví dụ sau.

Ví dụ 5.9. Thao tác với biến con trỏ.

```
INTEGER, POINTER :: P1 (:)
INTEGER, POINTER :: P2 (:)
INTEGER, ALLOCATABLE, TARGET :: D (:)
ALLOCATE (D (7)) ! Cấp phát bộ nhớ cho biến ĐICH
D = 1
D (1:7:2) = 10.
PRINT*, 'DICH=',D

P1 => D  ! Con trỏ trỏ vào biến ĐICH
PRINT*,'CON TRO P1=',P1
ALLOCATE (P1(10)) ! Cấp phát bộ nhớ cho biến con trỏ
P1 = 5
P2 => P1  ! Con trỏ trỏ vào biến đã xác định
PRINT*,'CON TRO P1=',P1
print*
print*,'CON TRO P2=',P2
P2 = 8
PRINT*,'CON TRO P1=',P1
print*
print*,'CON TRO P2=',P2
END
```

Ở đây ta gặp một ký hiệu mới là (=>), nó được dùng để chỉ một con trỏ trỏ vào một biến nào đó. Như trong ví dụ trên, khi **P1** trỏ vào biến **D** nó sẽ nhận nội dung của biến **D**. Nhưng khi **P1** được cấp phát bộ nhớ và khởi tạo giá trị mới (**P1=5**), sau đó **P2** trỏ vào nó thì **P2** và **P1** đều có cùng nội dung của **P1** đã thay đổi (tức bằng 5). Bây giờ gán **P2** bằng 8 thì cả **P2** và **P1** đều nhận giá trị bằng 8.

5.5.1 Trạng thái con trỏ

Tất cả các biến con trỏ trong chương trình luôn tồn tại ở một trong ba trạng thái sau:

– *Trạng thái không xác định (undefined)*. Bắt đầu chương trình mọi con trỏ đều ở trạng thái này.

– *Trạng thái không trỏ vào đâu cả* (*null)*, tức con trỏ chưa phải là "bí danh" của biến nào cả.

– *Trạng thái đã liên kết* (*associated),* tức con trỏ đã trỏ vào một biến nào đó (đã là "bí danh" của một biến "đích")

Để đưa con trỏ về trạng thái không trỏ vào đâu cả ta dùng câu lệnh:

NULLIFY (P) **! P** là biến con trỏ

Để xác định trạng thái hiện thời của con trỏ có thể dùng hàm

ASSOCIATED (P) **! P** là biến con trỏ

Hàm này trả về giá trị **.TRUE.** nếu con trỏ đã liên kết với một biến, và trả về giá trị **.FALSE.** nếu con trỏ ở trạng thái không trỏ vào đâu cả.

5.5.2 Cấp phát và giải phóng biến con trỏ

Biến con trỏ có thể được cấp phát bộ nhớ bằng câu lệnh **ALLOCATE** và được giải phóng bởi câu lệnh **DEALLOCATE** tương tự như mảng động. Ví dụ:

```
REAL, POINTER :: P1
ALLOCATE (P1)    ! Cấp phát bộ nhớ cho P1
P1 = 17    ! Gán giá trị cho P1 như đối với biến bất kỳ
PRINT*, P1
DEALLOCATE (P1)    ! Giải phóng biến P1
```

Ta cũng có thể sử dụng tham số **STAT** cho cả hai câu lệnh **ALLOCATE** và **DEALLOCATE**, chẳng hạn:

```
ALLOCATE( P1, STAT = ERR )
DEALLOCATE( P1, STAT = ERR )
```

Số nguyên **ERR** bằng 0 nếu bộ nhớ đã được cấp phát (hoặc giải phóng) xong.

Ta cũng có thể sử dụng cả **ALLOCATABLE** và **POINTER** để khai báo mảng động, chẳng hạn:

```
REAL, DIMENSION(:), POINTER :: X
INTEGER, DIMENSION(:,:), ALLOCATABLE :: A
```

Để minh hoạ ta xét đoạn chương trình sau.

```
REAL, POINTER :: A(:), B, C
REAL, ALLOCATABLE, TARGET :: D(:)
REAL, TARGET :: E
REAL, ALLOCATABLE :: F(:, :)
...
ALLOCATE (B, D(5), F(4, 2))
```

```
A => D
C => E
...
DEALLOCATE (B, D, F)
```

Như đã thấy, **A** là mảng động có kiểu con trỏ, **C** là con trỏ đơn (vô hướng), **D** là mảng động có thuộc tính **TARGET** và **E** là biến "tĩnh". Khi đó **A** có thể trỏ đến **D** và **C** có thể trỏ đến **E**.

5.6 HÀM TRẢ VỀ NHIỀU GIÁ TRỊ

Trong mục 4.3 đã nhấn mạnh đến việc hàm chỉ trả về một giá trị duy nhất gắn với tên hàm hoặc tên tham số trong tùy chọn **RESULT**. Đó là đặc tính thông thường mà Fortran 77 và các phiên bản trước cũng như nhiều ngôn ngữ khác vẫn có. Ngoài đặc tính đó, Fortran 90 còn cho phép định nghĩa hàm với khả năng trả về nhiều giá trị. Cú pháp định nghĩa loại hàm này về cơ bản không có gì khác so với cách định nghĩa hàm thông thường, ngoài trừ một số khai báo trong định nghĩa và trong chương trình gọi. Ta sẽ xét ví dụ sau đây để minh họa.

Giả sử có hàm $f(x) = 3x^2 + 2x - 5$. Hãy tính giá trị của hàm tại các giá trị của đối số $x_1, x_2,..., x_n$. Với cách định nghĩa thông thường, $f(x)$ sẽ được xác định thông qua một hàm mà giá trị của nó được tính ứng với một giá trị của đối số x. Và như vậy, trong chương trình gọi, hàm sẽ được tham chiếu tới n lần ứng với n giá trị x_i. Thay cho cách làm này, ta xây dựng một hàm mà đầu vào là n giá trị đối số x_i, còn đầu ra là n giá trị của hàm ứng với n giá trị đối số đó. Ta có chương trình như sau.

```
INTEGER, PARAMETER :: N = 7
REAL, DIMENSION (N) :: X, FX
DATA X /-3., -2., -1., 0., 1., 2., 3./
FX = F(X,N)
PRINT*, FX

CONTAINS
  FUNCTION F(X,N)
    INTEGER, INTENT (IN) :: N
    REAL, DIMENSION(N),INTENT(IN) :: X
    REAL, DIMENSION(SIZE(X)):: F
      F(:) = 3*X(:)*X(:) + 2*X(:) - 5
  END FUNCTION
END
```

Trong chương trình trên, hàm **F(X,N)** là hàm trong, có hai đối số hình thức là mảng **X** và số nguyên **N** chỉ kích thước của **X**. Kết quả trả về của hàm cũng là một mảng có kích thước bằng kích thước của **X**. Nếu **F(X,N)** được khai báo như một hàm ngoài thì trong phần khai báo của chương trình gọi cần phải có khối giao diện. Chẳng hạn:

```
INTEGER, PARAMETER :: N = 7
REAL, DIMENSION (N) :: X, FX
INTERFACE
  FUNCTION F(X,N)
    INTEGER, INTENT (IN) :: N
    REAL, DIMENSION(N),INTENT(IN) :: X
    REAL, DIMENSION(SIZE(X)):: F
```

```
  END FUNCTION
END INTERFACE
DATA X /-3., -2., -1., 0., 1., 2., 3./
!
FX = F(X,N)
PRINT*, FX
END
!
FUNCTION F(X,N)
  INTEGER, INTENT (IN) :: N
  REAL, DIMENSION(N),INTENT(IN) :: X
  REAL, DIMENSION(SIZE(X)):: F
    F(:) = 3*X(:)*X(:) + 2*X(:) - 5
END FUNCTION
```

BÀI TẬP CHƯƠNG 5

5.1 Ký hiệu X là mảng một chiều gồm 100 phần tử. Viết chương trình: a) Gán 100 số nguyên dương đầu tiên cho các phần tử tương ứng của X, từ phần tử có chỉ số lớn nhất đến phần tử có chỉ số nhỏ nhất; b) Gán 50 số nguyên dương lẻ đầu tiên cho 50 phần tử đầu tiên và 50 số nguyên dương chẵn đầu tiên cho 50 phần tử tiếp theo của X; c) Gán 100 số tự nhiên đầu tiên chia hết cho 3 lần lượt cho các phần tử của X. Mỗi một trường hợp như vậy, hãy in kết quả lên màn hình thành 10 dòng, mỗi dòng 10 số sao cho thẳng hàng thẳng cột.

5.2 Viết chương trình nhập vào một dãy *n* số thực và sắp xếp chúng: a) theo thứ tự tăng dần; b) theo thứ tự giảm dần. In kết quả lên màn hình thành ba cột với các số được định dạng theo số thực dấu phẩy tĩnh có 2 chữ số sau dấu chấm thập phân: cột 1 là dãy chưa sắp xếp, cột 2 là dãy đã sắp xếp theo thứ tự tăng dần, cột ba – giảm dần.

5.3 Biết rằng trong dãy *n* số thực biểu thị kết quả quan trắc của biến ngẫu nhiên X có một số giá trị khuyết thiếu đã được mã hoá bằng giá trị giả tạo −999.0. Viết chương trình thay thế các giá trị giả tạo đó bằng giá trị trung bình số học của những giá trị còn lại. In kết quả lên màn hình thành 2 cột với các số được định dạng theo số thực dấu phẩy tĩnh có 2 chữ số sau dấu chấm thập phân: cột 1 là số liệu ban đầu, cột 2 là số liệu đã xử lý.

5.4 Biết rằng dãy số x_i, i=1,2,.... *n*, chứa các giá trị 0 hoặc 1 biểu thị kết quả quan trắc liên tục trong *n* ngày của một hiện tượng nào đó. x_i nhận giá trị 1 nếu hiện tượng xuất hiện và nhận giá trị 0 nếu hiện tượng không xuất hiện. Ta gọi một đợt hiện tượng kéo dài *m* (ngày) nếu có *m* phần tử liên tiếp của dãy nhận giá trị 1 còn các phần tử trước và sau *m* phần tử này nhận giá trị 0. Hãy lập bảng thống kê:

Số ngày kéo dài của đợt (ngày)	1	2	3	...	M
Số đợt	m_1	m_2	m_3	...	m_M

5.5 Ký hiệu $y_i = f(x_i)$ là giá trị của hàm $f(x)$ tại giá trị $x = x_i$. Giả sử cho trước tập n cặp giá trị $\{(x_i, y_i), i=1, 2,..., n\}$, khi đó giá trị $y_0 = f(x_0)$ ứng với x_0 cho trước có thể được ước tính khi sử dụng công thức nội suy tuyến tính: $y_0 = y_i + (x_0 - x_i)\frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)}$, trong đó $x_i \le x_0 \le x_{i+1}$. Viết chương trình nhập vào n cặp số thực (x_i, y_i) và một số thực x_0 và tính giá trị y_0 tương ứng theo công thức trên đây (Chú ý sắp xếp các cặp số theo thứ tự tăng dần theo x trước khi tính toán). Chương trình cho phép kiểm tra điều kiện hợp lệ của x_0 như sau:

Nếu $Min\{x_i, i=1,2,...,n\} \le x_0 \le Max\{x_i, i=1,2,...,n\}$ thì tính y_0 và in kết quả, ngược lại thì đưa ra thông báo "Giá trị x_0 không hợp lệ".

5.6 Các phần tử của ma trận tích của hai ma trận được xác định theo công thức: $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$. Viết chương trình nhập vào hai ma trận $A(m,n)$ và $B(n,p)$ mà các phần tử của chúng là những số thực và tính tích của chúng. In lên màn hình các ma trận ban đầu và ma trận kết quả sao cho thẳng hàng thẳng cột với các chú thích hợp lý.

5.7 Viết chương trình nhập vào giá trị các phần tử của một ma trận các số thực và tính trung bình các hàng, các cột. In kết quả lên màn hình dưới dạng: cuối mỗi hàng là trung bình của hàng tương ứng, cuối mỗi cột là trung bình của cột tương ứng, các số được bố trí thẳng hàng thẳng cột với nhau.

5.8 Ký hiệu A là một mảng các số nguyên gồm 20 phần tử. Viết chương trình đọc vào giá trị các phần tử của mảng A, tìm và in lên màn hình phần tử có giá trị lớn nhất, nhỏ nhất và vị trí tương ứng của chúng trong mảng (chỉ số trong mảng của các phần tử này).

5.9 Để thống kê tình hình tai nạn giao thông trong năm, hàng tháng cơ quan Công an phải thu thập số liệu từ N địa phương (tỉnh, thành phố) trong cả nước. Giả sử số liệu thu thập được lưu trong một mảng nguyên A gồm N hàng (N địa phương) và 12 cột (12 tháng trong năm) mà các phần tử của nó là số vụ tai nạn xảy ra ở từng địa phương trong từng tháng. Sau khi có số liệu, người ta chia số vụ tai nạn thành từng khoảng giá trị, chẳng hạn 0─50, 51─100, 101─150,..., >300, và tiến hành tính số trường hợp (tần số) có số vụ tai nạn xảy ra trong từng khoảng tương ứng cho từng tháng. Ký hiệu B là ma trận gồm M hàng (M khoảng giá trị của số vụ tai nạn), 12 cột (12 tháng) chứa tần số tai nạn giao thông theo từng khoảng của từng tháng trong năm. Giả sử số liệu thu thập của cơ quan Công an được lưu trong file DATA.TXT mà cấu trúc của file là: Dòng 1 chứa một số nguyên dương N chỉ số địa phương có số liệu, N dòng tiếp theo, mỗi dòng gồm 12 cột lưu giá trị các phần tử tương ứng của mảng A. Viết chương trình đọc số liệu từ file, tính ma trận tần số B và in B lên màn hình thành M dòng, 12 cột.

5.10 Giả sử máy tính của bạn chỉ có thể cho phép thực hiện các phép tính với những số không quá lớn (ví dụ, với số nguyên 4 byte giá trị lớn nhất chỉ có thể là 2147483647) trong khi bạn cần phải tính toán với những số lớn tùy ý. Viết chương trình cho phép đọc vào hai số nguyên (x, y) tùy ý và thực hiện phép cộng hai số nguyên này. Chạy thử chương trình với các cặp số sau: *x=1234567890123, y=4567890*; *x=98765432109876, y=567890123456789*; in kết quả và so sánh với kết quả tính bằng tay. ***Gợi ý***: Sử dụng mảng để lưu các chữ số của các số vào các phần tử mảng rồi tiến hành phép cộng

các phần tử mảng tương ứng. Nhớ rằng giá trị của các phần tử mảng, kể cả mảng chứa kết quả, là những số có một chữ số.

5.11 Số thứ tự của một ngày nào đó trong năm được đánh số theo qui ước ngày 01 tháng 01 là ngày thứ nhất, v.v., ngày 31 tháng 12 là ngày thứ 365 (hoặc 366 nếu là năm nhuận). Viết chương trình nhập vào ngày, tháng, năm của một ngày nào đó và tính xem ngày đó là ngày thứ mấy trong năm.

5.12 Cũng với các điều kiện như bài tập 5.11. Viết chương trình nhập vào số thứ tự ngày trong năm và năm rồi xác định xem đó là ngày, tháng nào.

5.13 Viết chương trình nhập vào tọa độ N đỉnh của một đa giác lồi, phẳng và sắp xếp chúng theo thứ tự liên tiếp từ đỉnh thứ nhất đến đỉnh thứ N.

5.13 Viết chương trình nhập vào tọa độ N đỉnh của một đa giác lồi, phẳng và tọa độ của một điểm M di động trên mặt phẳng và xác định xem điểm M nằm trong hay nằm ngoài miền đa giác (nếu M nằm trên một cạnh nào đó của đa giác cũng được xem là nằm trong miền đa giác. Chương trình cho phép nhập tọa độ điểm M và xác định vị trí của nó với số lần bất kỳ cho đến khi cả hai tọa độ của M đều nhận giá trị −999.0. ***Gợi ý*:** Có thể sử dụng thuật toán tính tổng diện tích của các tam giác (không giao nhau) tạo bởi điểm M với các đỉnh của đa giác và so sánh diện tích đó với diện tích của đa giác. Chú ý khi so sánh hai số thực biểu thị hai giá trị diện tích trên.

5.14 Giả sử điểm thi học kỳ của một lớp sinh viên được lưu trong file DIEM.TXT. Cấu trúc file được mô tả như sau: Dòng 1 là tiêu đề ghi tên lớp, học kỳ,...; dòng 2 gồm 2 số nguyên dương chỉ số lượng sinh viên (N) và số môn học (M); dòng 3 gồm M số nguyên dương chỉ số học trình của M môn học; N dòng tiếp theo, mỗi dòng gồm M+1 cột: cột 1 ghi họ và tên của từng sinh viên, chiếm độ rộng 30 ký tự, kể từ ký tự đầu tiên của dòng, M cột tiếp theo, bắt đầu từ ký tự thứ 31, là M số thực, viết cách nhau bởi các dấu cách, ghi kết quả thi của M môn học tương ứng với số đơn vị học trình ở dòng thứ 3. Viết chương trình tính điểm trung bình chung học tập của từng sinh viên theo công thức $\text{TBCHT} = \frac{\sum_{i=1}^{n}(\text{Điểm môn i}) x (\text{Số học trinh môn i})}{\sum_{i=1}^{n}(\text{Số học trinh môn i})}$, xếp loại học tập cho từng sinh viên (xem bài tập 2.14), sắp xếp danh sách sinh viên theo thứ tự giảm dần của điểm trung bình chung học tập, in kết quả vào một file mới theo qui cách: Dòng 1 đến dòng 3 giống với file số liệu, từ dòng 4 trở đi, mỗi dòng ghi họ và tên sinh viên, điểm thi từng môn học, điểm trung bình chung học tập và kết quả xếp loại học tập.

5.15 Cho A là một ma trận M hàng, N cột gồm các số nguyên. Định nghĩa lân cận của phần tử a_{ij} là các phần tử của A có chỉ số hàng và chỉ số cột nhỏ hơn và lớn hơn (nếu có) các chỉ số *i, j* một đơn vị (tức *i−1, i+1, j−1, j+1*). Viết chương trình lập một ma trận B gồm M hàng, N cột mà các phần tử của B là: b_{ij} = 1 nếu số lận cận của a_{ij} có giá trị lớn hơn a_{ij} nhiều hơn số lân cận của a_{ij} có giá trị nhỏ hơn a_{ij}; b_{ij} = 0 nếu ngược lại. In các ma trận A và B lên màn hình.

5.16 Giả sử thông tin về đặt chỗ vé máy bay cho một chuyến bay của một hãng hàng không được lưu trong file BOOK.TXT. Nội dung file gồm 100 hàng, biểu thị số dãy ghế của máy bay, mỗi hàng gồm 10 số là những số hoặc bằng 0 hoặc bằng 1, biểu thị số ghế ngồi trên một dãy. Giá trị bằng 0 chỉ chỗ ngồi còn trống, giá trị bằng 1 chỉ chỗ ngồi đã được đặt. Mỗi dãy ghế bị phân chia thành hai bên trái và phải bởi lối đi ở giữa, mỗi bên 5 ghế. Hai chỗ ngồi được xem là liền kề nhau nếu chúng cùng thuộc một dãy và không bị ngăn cách bởi lối đi. Viết chương trình đọc file số liệu và in ra những vị trí có ít nhất hai chỗ ngồi liền kề nhau còn trống.

CHƯƠNG 6. BIẾN KÝ TỰ

6.1 KHAI BÁO BIẾN KÝ TỰ

Hiểu một cách đơn giản, hằng ký tự là một xâu (dãy) các ký tự nằm giữa các cặp dấu nháy đơn (‘ ’) hoặc nháy kép (“ ”). Biến ký tự là biến có thể nhận giá trị là các hằng ký tự. Bởi vì mỗi ký tự chiếm một byte bộ nhớ, nên dung lượng bộ nhớ mà xâu ký tự chiếm phụ thuộc độ dài của xâu ký tự.

Nói chung có thể có nhiều cách khai báo biến ký tự như đã được đề cập đến trong mục 1.4.2. Sau đây dẫn ra một số ví dụ về cách khai báo biến ký tự thường dùng.

CHARACTER StrName [,...]

! Khai báo biến **StrName** có độ dài 1 ký tự

CHARACTER ([LEN=]*n*) StrName [,...]

! Khai báo biến **StrName** có độ dài ***n*** ký tự

CHARACTER ****n* **StrName [,...] ! Tương tự như trên

CHARACTER StrName**n* **[,...] !** Tương tự như trên

StrName là tên biến ký tự, ***n*** là một số nguyên dương chỉ độ dài cực đại của biến ***StrName***. Ví dụ:

CHARACTER ALPHA

!ALPHA là một xâu dài tối đa 1 ký tự (nhận các giá trị ‘A’,...)

CHARACTER (25) Name

!Name là một xâu dài tối đa 25 ký tự

CHARACTER Word*5 ! Word là một xâu dài tối đa 5 ký tự

...

Name = “Hanoi, Ngay...“

Word = ‘Hanoi’

...

Khi biến ký tự có thuộc tính **PARAMETER** ta còn có thể khai báo xâu có độ dài chưa xác định:

CHARACTER *(*) StrName
PARAMETER (StrName= “XauDai12KyTu”)

Hoặc:

CHARACTER *(*) , PARAMETER :: ST1 = 'ABDCEF'

Trong trường hợp này độ dài của xâu sẽ là độ dài thực của xâu được gán.

6.2 CÁC XÂU CON (SUBSTRING)

Xâu con là một bộ phận của xâu ký tự. Xâu con có thể chỉ có một ký tự cũng có thể là toàn bộ xâu. Giả sử **TEXT** là một xâu có độ dài cực đại 80 ký tự:

CHARACTER (80) TEXT

Khi đó **TEXT(I:J)** là xâu con gồm các ký tự từ ký tự thứ **I** đến ký tự thứ **J** của xâu **TEXT**. Từng ký tự riêng biệt trong xâu **TEXT** có thể được tham chiếu (truy cập) đến bằng xâu con **TEXT(I:J)**. Ví dụ:

TEXT(J:J) ! ký tự thứ J của xâu **TEXT**
TEXT(:J) ! từ ký tự thứ 1 đến ký tự thứ J
TEXT(1:J) ! từ ký tự thứ 1 đến ký tự thứ J
TEXT(J:) ! từ ký tự thứ J đến ký tự thứ 80
TEXT(J:80) ! từ ký tự thứ J đến ký tự thứ 80
TEXT(:) ! từ ký tự thứ 1 đến ký tự thứ 80 (cả xâu)
TEXT(1:80) ! (hoặc **TEXT)** tương tự, cả xâu

Nếu

TEXT = "Hanoi–Vietnam"

thì

TEXT(3:5) có giá trị là **"noi"**
TEXT(:5) có giá trị là **"Hanoi"**
TEXT(7:) có giá trị là **"Vietnam "**
TEXT(6:6) = " * " sẽ cho **TEXT= "Hanoi * Vietnam"**
TEXT(2:5) = "ANOI" sẽ cho **TEXT= "HANOI–Vietnam"**

6.3 XỬ LÝ BIẾN KÝ TỰ

Xử lý biến ký tự trong Fortran là một vấn đề khá phức tạp. Ở một số ngôn ngữ lập trình khác (chẳng hạn, PASCAL), việc xử lý biến ký tự nói chung được hỗ trợ bởi nhiều thủ tục hoặc hàm thư viện. Đối với Fortran, vấn đề này thường phải do người dùng tự lập. Sau đây ta sẽ xét một số bài toán làm ví dụ minh hoạ.

Ví dụ 6.1. Một trong những thủ thuật xử lý biến ký tự là chèn một xâu vào một xâu ký tự khác. Có thể nêu nguyên tắc thực hiện như sau: Để chèn một xâu ***SubStr*** vào một vị trí nào đó của xâu ***Str*** cho trước cần phải dịch chuyển các ký tự phía sau vị trí cần chèn của xâu ***Str*** sang phải một số vị trí bằng độ dài xâu ***SubStr.***

Giả sử có xâu **TEXT = "Hanoi – Saigon"**, nếu muốn chèn xâu con " **– Hue**" vào xâu này để nhận được xâu "**Hanoi – Hue – Saigon**" ta có thể lập trình như sau:

```
CHARACTER (50) TEXT
!       12345678901234
TEXT = 'Hanoi - Saigon'
print*,TEXT
I = 6          ! Chèn vào vị trí thứ 6 (trước dấu "-")
```

```
LENSub = 6     ! Độ dài xâu cần chèn (“ – Hue”) là 6
LenTEXT = LEN_TRIM( TEXT )
DO J = LenTEXT, I, -1
 ! Bắt đầu từ cuối xâu đến vị trí thứ I
  TEXT( J+LENSub:J+LENSub ) = TEXT(J:J)
 ! Sao chép ký tự thứ J đến vị trí thứ J+LENSub
END DO
TEXT(I:I+LENSub) = ' - Hue'
print*,TEXT
END
```

Hàm **LEN_TRIM (TEXT)** trong chương trình trả về độ dài xâu **TEXT** *sau khi đã loại bỏ các dấu cách ở bên phải nhất* của xâu. Có thể mô tả tác động của chương trình như sau. Cho **J** lần lượt nhận các giá trị từ độ dài thực (**LenTEXT**) của xâu **TEXT** đến vị trí cần chèn xâu con (**I**), mỗi lần như vậy ta sao chép ký tự thứ **J** của xâu **TEXT** đến vị trí **J+LENSub**. Kết quả của vòng lặp này đã dịch chuyển (sao chép) nội dung **TEXT(I:LenTEXT)** lùi sang phải **LENSub** vị trí. Tiếp đó ta ghi đè nội dung của xâu con **' – Hue'** vào xâu **TEXT** kể từ vị trí thứ **I** đến vị trí thứ **I+LENSub**.

Ví dụ 6.2. Thay thế khoảng trống giữa các từ bằng một dấu cách (space bar). Giả sử ta định nghĩa một từ là một dãy ký tự liên tiếp không chứa dấu cách. Khi đó trong một xâu có thể có nhiều từ. Khi gõ văn bản, giữa hai từ chỉ được phép để một dấu cách. Hãy tìm những khoảng trống giữa các từ có nhiều hơn một dấu cách và thay thế chúng bởi chỉ một dấu cách. Ta có chương trình sau.

```
CHARACTER (Len=80) ST, ST1, ST2, ALLTRIM
INTEGER L
LOGICAL OK
ST='  Ha noi   la   Thu  do   cua VIET NAM  '
PRINT*, ST
ST1 = ALLTRIM(ST)  ! Cắt bỏ các dấu cách ở hai đầu
OK = .FALSE.
DO WHILE (.NOT. OK)
  L = INDEX(TRIM(ST1),'  ') ! Tìm vị trí có 2 dấu cách
  IF (L > 0) THEN          ! Nếu còn tìm thấy:
    OK = .FALSE.
    ST2 = ' '             !Gán các dấu cách cho ST2
    ST2(1:L-1) = ST1(1:L-1)! Sao chép nội dung ST1
    ST2(L:) = ST1(L+1:)    ! vào ST2 sau khi loại
                  ! bỏ bớt 1 dấu cách
    ST1 = ST2              ! Sao chép ST2 vào ST1
  ELSE
    OK = .TRUE.     ! Nếu không tìm thấy thì thoát
  END IF
END DO
ST = ‘ ‘
ST = ST1
PRINT*,ST
END

FUNCTION ALLTRIM (ST)
```

```
CHARACTER *80 ST, ALLTRIM
INTEGER I, J
J=LEN_TRIM(ST)
I=0
DO
  I=I+1
  IF (ST(I:I) /= ' ') EXIT
ENDDO
ALLTRIM = ST(I:J)
RETURN
END
```

Trong chương trình trên ta đã xây dựng một hàm **ALLTRIM** có chức năng loại bỏ tất cả những ký tự trống ở cả bên phải nhất và bên trái nhất của xâu ký tự. Còn **INDEX** và **TRIM** là các hàm thư viện của Fortran. Hàm **INDEX** trả về vị trí lần gặp đầu tiên một xâu con trong xâu ký tự; hàm **TRIM** trả về xâu ký tự đã cắt bỏ những ký tự trống ở bên phải nhất của xâu ký tự.

Ví dụ 6.3. Tìm và tách các từ trong một xâu. Giả sử có xâu ký tự **ST**. Hãy xác định xem trong xâu có bao nhiêu từ và cho biết nội dung của chúng.

```
CHARACTER (Len=80) ST, ST1, ST2, ALLTRIM
CHARACTER*10 S(20)  ! Mảng chứa nội dung các từ
INTEGER L, I, K
LOGICAL OK
ST='  Ha noi   la   Thu  do   cua VIET NAM  '
PRINT*, ST
CALL NO_DOUBLE_SPACES(ST)
ST1 = ST
I=0              ! Biến đếm số từ có trong xâu
L=1              ! Vị trí của các từ trong xâu
OK=.FALSE.
DO WHILE (.NOT.OK)
  K=INDEX( TRIM(ST1(L:)), ' ')
  IF (K > 0) THEN  ! Nếu tìm thấy dấu cách giữa 2 từ
    I=I+1
    S(I)=ST1(L:L+K-1) ! Lưu nội dung từ thứ I
    L=L+K
  ELSE
    OK=.TRUE.
  END IF
END DO
I = I + 1
S(I)=ST1(L:LEN_TRIM(ST1))
PRINT*,' So tu trong xau = ', I
PRINT*,' Noi dung cac tu trong xau la '
DO L=1,I
  PRINT*,S(L)
END DO
END
```

```
SUBROUTINE NO_DOUBLE_SPACES(ST)
CHARACTER*80 ST,ST1,ST2, ALLTRIM
INTEGER L
LOGICAL OK
ST1=ALLTRIM(ST)
OK = .FALSE.
DO WHILE (.NOT. OK)
  L = INDEX(TRIM(ST1),'  ')
  IF (L > 0) THEN
    OK = .FALSE.
    ST2 = ' '
    ST2(1:L-1) = ST1(1:L-1)
    ST2(L:) = ST1(L+1:)
    ST1 = ST2
  ELSE
    OK = .TRUE.
  END IF
END DO
ST=' '
ST=ST1
RETURN
END
```

Thủ tục **NO_DOUBLE_SPACES** thực chất là nội dung của ví dụ 6.2 trên đây, nhưng ta đã xây dựng thành một chương trình con để tiện sử dụng. Thủ tục này cũng tham chiếu tới hàm **ALLTRIM** đã đề cập đến trong ví dụ 6.2.

Ví dụ 6.4. Cho một xâu chứa họ tên đầy đủ của một người. Hãy cho biết rõ họ, tên và tên đệm (họ đệm) của người đó.

Để giải bài toán này ta giả thiết rằng, họ của người đó là từ đầu tiên trong xâu, tên của người đó là từ cuối cùng trong xâu, phần còn lại của xâu nằm giữa họ và tên là tên đệm. Ta có thể viết chương trình như sau.

```
CHARACTER (Len=80) ST, ST1
CHARACTER *80 FIRST_WORD, END_WORD, ALLTRIM
CHARACTER *80 HO, DEM, TEN, ST2
ST=' Nguyen  Le  Hoang  Viet  '
CALL NO_DOUBLE_SPACES(ST)
      ! Cắt bỏ những khoảng trống thừa
ST1=ST
HO  = FIRST_WORD (ST1)  ! Họ là từ đầu tiên
TEN = END_WORD (ST1)    ! Tên là từ cuối cùng
L1 = LEN_TRIM (HO)
L2 = LEN_TRIM(TEN)
L3 = LEN_TRIM(ST1)
ST2 = ST1(L1+1:L3-L2)
DEM = ALLTRIM(ST2)      ! Đệm là phần giữa Họ và Tên
```

```
print*,HO
print*,DEM
print*,TEN
END

!!!!!!!!!!!!!!!!!!!!!!!!

FUNCTION FIRST_WORD (ST)
CHARACTER*80 ST, FIRST_WORD
I=0
DO
  I=I+1
  IF (ST(I:I) == ' ') EXIT
ENDDO
FIRST_WORD = ST(:I-1)
RETURN
END

FUNCTION END_WORD (ST)
CHARACTER*80 ST, END_WORD
K = 0
J = LEN(TRIM(ST))
DO
  K = K + 1
  J = J - 1
  IF (ST(J:J) == ' ') EXIT
ENDDO
J = LEN_TRIM(ST)
END_WORD = ST(J-K+1:J)
RETURN
END
```

Các hàm **FIRST_WORD** và **END_WORD** trong chương trình tương ứng sẽ trả về các từ đầu tiên và cuối cùng của xâu.

Một số hàm xử lý xâu ký tự trong thư viện Fortran.

LEN (Str): trả về độ dài cực đại (khai báo) của xâu ***Str***

LEN_TRIM (Str): trả về độ dài xâu ***Str*** sau khi đã loại bỏ các ký tự trống (dấu cách) ở bên phải nhất

ACHAR (I): trả về ký tự thứ **I** trong bảng mã ASCII

IACHAR(c): trả về số thứ tự trong bảng mã ASCII của ký tự ***c***

INDEX (Str, SubStr [, back]): trả về vị trí đầu tiên của xâu con ***SubStr*** trong xâu ***Str***. Tham số tùy chọn ***back*** có ý nghĩa như sau:

Nếu ***back*** = **.TRUE.**: tìm ***SubStr*** từ cuối xâu ***Str***

Nếu ***back*** = **.FALSE.**: tìm ***SubStr*** từ đầu xâu ***Str***

Giá trị ngầm định là ***back*** = **.FALSE.**

REPEAT (Str, ncopies): trả về một xâu gồm ***ncopies*** lần sao chép ***Str***

TRIM (Str): trả về xâu ***Str*** sau khi đã cắt bỏ các ký tự trống ở bên phải nhất.

6.4 PHÉP TOÁN GỘP XÂU KÝ TỰ

Phép toán gộp hai xâu ký tự được ký hiệu là //. Giả sử ta muốn tạo một tên file từ hai xâu là ***Name*** chứa tên và ***Ext*** chứa phần mở rộng. Có thể cả hai xâu này còn chứa các dấu cách ở đầu và cuối xâu. Trước khi gộp hai xâu này thành một xâu có ý nghĩa *tên của một file* ta cần phải cắt bỏ các dấu cách đó. Việc cắt bỏ này có thể thực hiện bằng lời gọi hàm **ALLTRIM** như các ví dụ trong mục 6.3. Để bạn đọc có thể nắm bắt được những tình huống khác nhau khi xử lý biến ký tự, chương trình sau đây sẽ đưa ra một phương án khác.

Ví dụ 6.5. Tạo tên file từ hai xâu.

```
CHARACTER (80) FName, Name, Ext
!      123456789012345
Name = '   gl04012200  '
Ext  = '  .dat  '
  ! Cả 2 xâu trên đều có chứa dấu cách ở đầu và cuối
Len1 = INDEX(TRIM(Name),' ',.true.)
Len2 = INDEX(TRIM(Ext),' ',.true.)
  ! Xác định vị trí dấu cách cuối cùng bên trái
  ! của hai xâu (Kết quả: Len1=3, Len2=2)
Len1 = Len1 + 1
Len2 = Len2 + 1
  ! Ký tự tiếp theo sau dấu cách
Len3 = LEN(TRIM(Name))
Len4 = LEN(TRIM(Ext))
  ! Xác định độ dài xâu sau khi đã cắt bỏ dấu cách
  ! bên phải của hai xâu (Kết quả: Len3=13, Len4=8)
FName = Name(Len1:Len3) // Ext(Len2:Len4)
  ! Gộp tên và phần mở rộng để tạo thành tên file
PRINT*, FName
END
```

Khi chạy chương trình này, ta sẽ nhận được kết quả trên màn hình là "**gl04012200.dat**".

6.5 TẠO ĐỊNH DẠNG FORMAT BẰNG XÂU KÝ TỰ

Biểu thức xâu ký tự có thể được sử dụng để tạo định dạng **FORMAT** tự động trong chương trình. Ví dụ sau đây cho phép in một số thực dạng dấu phẩy tĩnh với độ rộng trường bằng 9, còn số chữ số thập phân cần in ra được lựa chọn tùy ý (tối đa là 4 chữ số):

```
CHARACTER (1), DIMENSION(0:4) :: TP = &
          & (/'0','1','2','3','4'/)
!                   123456
CHARACTER (8) :: FMT = "(F9.?)"
PRINT*,'Cho so X:'
READ*,X
PRINT*,'Cho so chu so thap phan can in:'
```

```
READ*, N
FMT(5:5)=TP(N) ! Thay dấu (?) bởi số chữ số thập phân
PRINT FMT, X
END
```

Chương trình sau sẽ in **N** số nguyên dương đầu tiên trên cùng một dòng, mỗi số chiếm 4 vị trí (độ rộng trường bằng 4):

```
!                  1234567890
CHARACTER *11 :: FMT = '(2X, ???I4)'
CHARACTER *3 SubSt
PRINT*,'CHO SO N:'
READ*,N
WRITE(SubSt,'(I3.3)') N ! Đổi số N thành ký tự
FMT(6:8)=SubSt
WRITE(*, FMT) (I,I=1,N)
END
```

Sau đây là một ví dụ về kết xuất thông tin dạng mảng ra file TEXT có qui cách. Giả sử trong khi thực hiện chương trình ta muốn in một mảng hai chiều ra file TEXT dưới dạng ma trận, tức dữ liệu lưu trữ trong file phải được bố trí thẳng hàng thẳng cột, trong khi kích thước của mảng không được biết trước mà chỉ được xác định trong quá trình tính toán. Để làm điều đó ta có thể sử dụng đoạn chương trình sau.

```
PROGRAM In_Co_Dinh_Dang
REAL, ALLOCATABLE :: A(:,:)
INTEGER M, N, I, J
CHARACTER FMT*80
...
M = ...
N = ...
ALLOCATE (A(N, M))
...
OPEN (3, FILE=”OUT.TXT”)
WRITE (FMT,'(A1,I2.2,A6)') '(', M, 'F15.8)'
DO I=1,N
   WRITE (3,FMT) (A(I,J),J=1,M)
ENDDO
...
END
```

6.6 MẢNG XÂU KÝ TỰ

Xâu ký tự có thể khai báo ở dạng biến đơn cũng có thể khai báo ở dạng biến mảng. Mảng xâu ký tự là mảng trong đó mỗi phần tử là một xâu ký tự. Các phần tử trong mảng xâu ký tự phải có độ dài giống nhau. Như vậy, nếu mỗi phần tử trong mảng có độ dài là **n** ký tự, thì mảng một chiều gồm **m** phần tử sẽ có kích thước **n** x **m** ký tự. Ví dụ, chương trình sau đây định nghĩa các ngày trong tuần là các xâu ký tự đươc xác định bởi các phần tử tương ứng của một mảng:

```
CHARACTER (8), DIMENSION(7) :: DayOfWeek = &
   &(/ ’Thu 2’,                    ‘Thu 3’, ‘Thu 4’, ‘Thu 5’,&
```

```
   &   'Thu 6','Thu 7','Chu nhat' /)
PRINT*,'Cac ngay trong tuan la:'
DO I = 1,7
  PRINT*, DayOfWeek (I)
END DO
END
```

Trong ví dụ này, mảng ***DayOfWeek*** là mảng một chiều gồm 7 phần tử, mỗi phần tử là một xâu có độ dài cực đại bằng 8 ký tự.

Ta cũng có thể truy cập đến từng ký tự riêng biệt trong các phần tử của mảng. Ví dụ, ***DayOfWeek(1)(5:5)*** là ký tự thứ 5 của phần tử thứ nhất của mảng, nên nó có giá trị là "**2**".

Bằng cách tương tự, ta có thể định nghĩa mảng ký tự hai chiều, ba chiều,…

BÀI TẬP CHƯƠNG 6

6.1 Một từ được định nghĩa như là một dãy ký tự khác dấu cách đứng liền nhau. Giả thiết giữa các từ chỉ được phân cách nhau bởi các dấu cách. Viết chương trình nhập vào một xâu ký tự có độ dài tùy ý và cho biết trong xâu đó có bao nhiêu từ, mỗi từ dài bao nhiêu ký tự.

6.2 Định nghĩa một câu là một dãy các từ được kết thúc bằng dấu chấm (.). Viết chương trình nhập vào một xâu ký tự và cho biết trong xâu đó có bao nhiêu câu.

6.3 Theo qui định về gõ văn bản, các dấu phân cách như chấm câu, dấu phẩy, dấu ngoặc mở, dấu ngoặc đóng, dấu chấm than, dấu hỏi,… phải viết liền ngay sau ký tự kết thúc của một từ. Giả sử có file văn bản (TEXT file) có tên là DOC.TXT mà nội dung của nó gồm N dòng, mỗi dòng dài không quá 80 ký tự. Viết chương trình đọc file văn bản và cho biết trong file có bao nhiêu lỗi xảy ra khi gõ các dấu chấm câu và dấu phẩy không đúng qui định.

6.4 Phát triển bài tập 6.3 cho các trường hợp dấu phân cách khác và sửa các lỗi đó cho file văn bản.

6.5 Viết chương trình đọc vào một câu (kết thúc bởi dấu chấm) và in lên màn hình (không in dấu chấm) theo thứ tự nghịch đảo: a) các từ; b) các ký tự. Ví dụ, "Ha Noi." ➔ "Noi Ha" và "ioN aH".

6.6 Công thức đồng dư Zeller có thể được dùng để tính ngày trong tuần có dạng:

$$f = \left(\left[2.6m - 0.2\right] + k + y + \left[y/4\right] + \left[c/4\right] - 2c\right) \text{ modulo } 7.$$

Trong đó m là số thứ tự tháng, với qui ước tháng 1 và tháng 2 tương ứng là tháng thứ 11 và tháng thứ 12 của năm trước, tháng 3 là tháng thứ 1,…, tháng 12 là tháng thứ 10; k là số thứ tự ngày trong tháng; c là số thứ tự thế kỷ; y là số thứ tự năm trong thế kỷ; $f=0$ là Chủ Nhật, $f=1$ là Thứ Hai,…; dấu ngoặc vuông là ký hiệu lấy phần nguyên. Ví dụ, ngày 23 tháng 8 năm 1963 được biểu diễn bởi $m = 6$, $k = 23$, $c = 19$, $y = 63$; ngày 01 tháng 01 năm 1980 được biểu diễn bởi $m = 11$, $k = 1$, $c = 17$, $y = 99$. Viết chương trình đọc một xâu ký tự mô tả thời gian là một ngày nào đó, chẳng hạn, "*Today is 08/03/2005*", chuyển thông tin ngày, tháng, năm từ xâu ký tự này thành dạng số và sử dụng công thức Zeller để xác định ngày đó là ngày thứ mấy trong tuần.

6.7 Viết chương trình đọc họ và tên (bao gồm cả tên đệm) của một người và in ra chỉ Họ và Tên (không có tên đệm) của người đó.

6.8 Theo qui ước ghi số liệu quan trắc mưa, nếu có mưa nhưng không tiến hành đo lượng mưa thì ghi dấu (X), nghĩa là mất số liệu, nếu không mưa thì ghi dấu (−), trong những trường hợp khác lượng mưa được biểu thị bằng một số thực không âm. Giả sử file số liệu RAIN.TXT lưu giá trị quan trắc tổng lượng mưa ngày trong 10 năm, mỗi năm 12 tháng, của một trạm nào đó, trong đó mỗi dòng gồm tối đa 31 số, cách nhau bởi các dấu cách, ghi số liệu từng ngày trong một tháng. Viết chương trình đọc file số liệu và cho biết có bao nhiêu ngày có mưa nhưng không được quan trắc, bao nhiêu ngày không mưa.

6.9 Viết chương trình con dạng thủ tục cho phép in một ma trận các số thực có kích thước M hàng N cột vào một file TEXT sao cho trên mỗi hàng có đúng N phần tử, với M và N bất kỳ. Các số được in dưới dạng số thực dấu phẩy tĩnh (*Fw.d*) với độ rộng trường (*w*) và số chữ số sau dấu chấm thập phân (*d*) cũng được xác định qua truyền tham số cho chương trình con.

CHƯƠNG 7. KIỂU FILE

7.1 KHÁI NIỆM

Trong hệ thống vào/ra của Fortran, dữ liệu được lưu trữ và chuyển đổi chủ yếu thông qua các **file**. Tất cả các nguồn vào/ra cung cấp và kết xuất dữ liệu được xem là **các file**. Các thiết bị như màn hình, bàn phím, máy in được xem là những **file ngoài** (*external files*), kể cả các file số liệu lưu trữ trên đĩa. Các biến trong bộ nhớ cũng có thể đóng vai trò như các file, đặc biệt chúng được sử dụng để chuyển đổi từ dạng biểu diễn mã ASCII sang số nhị phân (*binary*). Khi các biến được sử dụng theo cách này, chúng được gọi là các **file trong**.

Các **file trong** hoặc **file ngoài** đều được liên kết với cái gọi là thiết bị lôgic. Thiết bị lôgic là một khái niệm được sử dụng để tham chiếu đến các file. Ta có thể nhận biết một thiết bị lôgic liên kết với một file bằng định danh (**UNIT=**).

Định danh **UNIT** đối với một file trong là tên của một biến ký tự liên kết với nó. Định danh **UNIT** đối với một file ngoài hoặc là một số nguyên dương được gán trong lệnh **OPEN,** hoặc là một số kết nối trước như là định danh **UNIT** đối với thiết bị, hoặc dấu sao (*). Các định danh **UNIT** ngoài được kết nối với các thiết bị nhất định không được mở (**OPEN**). Các **UNIT** ngoài đã kết nối sẽ bị ngắt kết nối khi kết thúc thực hiện chương trình hoặc khi **UNIT** bị đóng bởi lệnh **CLOSE**.

Tại một thời điểm **UNIT** không thể kết nối với nhiều hơn một file, và file cũng không kết nối với nhiều hơn một thiết bị.

Định danh **UNIT** liên kết với một file ngoài phải là một số nguyên, một biểu thức nguyên hoặc dấu sao (*). Nếu là số nguyên hoặc biểu thức nguyên, giá trị của nó sẽ liên kết với một file trên đĩa; nếu là dấu sao (*) thì khi đọc vào nó được hiểu là bàn phím, còn khi in ra ngầm định là màn hình. Ví dụ:

```
OPEN (UNIT = 10, FILE = 'TEST.dat')
WRITE(10,'(A18,\)')' Ghi vao File TEST.dat &
          & da lien ket voi UNIT 10'
WRITE (*, '(1X, A30,\)') ' In ra man hinh.'
```

Fortran ngầm định một số thiết bị chuẩn liên kết với định danh **UNIT** như sau:

– Dấu sao (*): Màn hình hoặc bàn phím

– **UNIT** = 0: Màn hình hoặc bàn phím

– **UNIT** = 5: Bàn phím

– **UNIT** = 6: Màn hình

Ví dụ 7.1. Trong chương trình sau đây, **UNIT** 6 nếu không liên kết với file ngoài nó sẽ được hiểu là màn hình. Tuy nhiên khi muốn liên kết nó với file ngoài ta phải sử dụng lệnh mở file (**OPEN**), và để loại bỏ liên kết đó ta dùng lệnh đóng file (**CLOSE**).

```
REAL a, b
 ! In ra màn hình (UNIT 6 đã kết nối trước).
WRITE(6, '(" Day la UNIT 6")')
 ! Sử dụng lệnh OPEN để kết nối UNIT 6
 ! với file ngoài có tên 'COSINES'.
OPEN (UNIT = 6, FILE = 'COSINES', STATUS = 'NEW')
DO a = 0.1, 6.3, 0.1
   b = COS (a)
! Ghi vào file 'COSINES'.
   WRITE  (6, 100) a, b
 100 FORMAT (F3.1, F5.2)
END DO
! Đóng file, cắt bỏ kết nối với file trên đĩa.
CLOSE (6)
! Kết nối lại UNIT 6 với màn hình bằng cách
!                ghi ra màn hình
WRITE(6,' (" Ket thuc chuong trinh ")')
END
```

Định danh **UNIT** liên kết với file trong là xâu ký tự hoặc mảng ký tự. Đối với các file trong ta chỉ có thể sử dụng các câu lệnh **READ** hoặc **WRITE**. Ta không thể mở hoặc đóng file trong như đối với các file ngoài.

Có thể đọc và ghi các file trong với việc sử dụng lệnh định dạng **FORMAT** như đối với các file ngoài. Trước khi câu lệnh vào/ra được thực hiện, các file trong được định vị tại vị trí đầu của bản ghi đầu tiên.

Bằng khái niệm file trong, Fortran cho phép ta chuyển đổi giữa các dạng dữ liệu, chẳng hạn đổi ký tự sang số hoặc đổi số sang dạng ký tự.

Ví dụ 7.2. Chuyển đổi dữ liệu từ ký tự thành số và từ số thành ký tự khi sử dụng khái niệm file trong.

```
CHARACTER(10) str
INTEGER n1, n2, n3
CHARACTER(14) fname
INTEGER  I
str = " 1  2  3"  ! Xâu ký tự
READ(str, *) n1, n2, n3
! Đọc n1, n2, n3 từ xâu str (Chuyển ký tự thành số)
PRINT*,n1,n2,n3
I = 4
WRITE (fname, 200) I
! Ghi giá trị của I vào fname (Chuyển số thành ký tự)
200 FORMAT ('FM', I3.3, '.DAT')
```

```
PRINT*,fname
END
```

Trong chương trình trên, ***Str*** và ***Fname*** là các file trong. Kết quả chạy chương trình ta sẽ nhận được **n1=1, n2=2, n3=3, fname = "FM004.DAT"**.

7.2 PHÂN LOẠI FILE

Fortran hỗ trợ hai phương pháp truy cập file là truy cập tuần tự và truy cập trực tiếp, và ba dạng cấu trúc file là có định dạng (*Formatted*), không định dạng (*Unformatted*), và dạng nhị phân (*Binary*).

7.2.1 File có định dạng (Formatted Files)

Có thể tạo file có định dạng bằng lệnh **OPEN** với tùy chọn **FORM** = "**FORMATTED**", hoặc bỏ qua tham số **FORM** khi file được mở ở chế độ truy cập tuần tự. Các bản ghi của file có định dạng được lưu trữ như các ký tự ASCII. Bởi vậy ta có thể nói file có định dạng là ASCII file, hay TEXT file. Mỗi bản ghi kết thúc bằng các ký tự (ASCII) điều khiển **RETURN (CR)** và xuống dòng **(LF – line feed).** Để xem nội dung file có thể sử dụng các trình soạn thảo văn bản thông thường.

Ví dụ 7.3. Tạo một file có định dạng (TEXT file)

```
OPEN (UNIT=3, FILE= "TEST.TXT", FORM= "FORMATTED")
WRITE (3, *) "Day la file co dinh dang"
CLOSE (3)
END
```

7.2.2 File không định dạng (Unformatted Files)

Để tạo một file không định dạng có thể sử dụng lệnh **OPEN** với tùy chọn **FORM**="**UNFORMATTED**", hoặc bỏ qua tham số **FORM** khi file được mở ở chế độ truy cập trực tiếp. File không định dạng là một chuỗi bản ghi các khối vật lý. Mỗi bản ghi chứa tuần tự các giá trị lưu trữ gần giống với những gì sử dụng trong bộ nhớ chương trình. Tốc độ truy cập dữ liệu trong các file này nhanh hơn và chúng được tổ chức chặt chẽ hơn các file có định dạng. Nếu các file không định dạng lưu trữ các số, chúng sẽ không thể đọc được bằng các trình soạn thảo văn bản thông thường. Nói chính xác hơn, khi sử dụng các trình soạn thảo để đọc các file không định dạng, những thông tin bằng số sẽ không thể xem được một cách rõ ràng.

Ví dụ 7.4. Tạo file không định dạng và đọc thông tin từ file không định dạng.

```
CHARACTER *50 St
INTEGER A,B,C,D
OPEN (UNIT=3, FILE= "TEST.TXT", FORM= "UNFORMATTED")
St="Day la file khong dinh dang truy cap tuan tu"
WRITE (3) St       ! Bản ghi thứ nhất
WRITE (3) 1,2,3,4  ! Bản ghi thứ 2
WRITE (3) 5,6,7,8  ! Bản ghi thứ ba
REWIND (3)         ! Quay về vị trí bản ghi thứ nhất
READ (3) St        ! Đọc bản ghi thứ nhất
PRINT*,ST
```

```
READ (3) A, B, C, D  ! Đọc bản ghi thứ 2
PRINT*, A, B, C, D
READ (3) A, B, C, D  ! Đọc bản ghi thứ 3
PRINT*, A, B, C, D
CLOSE (3)
END
```

Trong ví dụ trên, câu lệnh **REWIND(3)** có tác dụng đưa con trỏ file định vị ở vị trí đầu file.

7.2.3 File dạng nhị phân (Binary Files)

Có thể tạo một file nhị phân bằng lệnh **OPEN** với tùy chọn **FORM**= '**BINARY**'. File nhị phân là dạng file chặt chẽ nhất, rất tốt cho việc lưu trữ số liệu có dung lượng lớn.

Ví dụ 7.5. Đọc và ghi file nhị phân truy cập tuần tự.

```
CHARACTER *50 St
INTEGER A,B,C,D
OPEN (UNIT=3, FILE= "TEST.TXT", FORM= "BINARY")
St= "Day la file dang nhi phan truy cap tuan tu"
WRITE (3) St      ! Bản ghi thứ nhất
WRITE (3) 1,2,3,4 ! Bản ghi thứ hai
REWIND (3)        ! Quay lại đầu bản ghi thứ nhất
READ (3) St       ! Đọc bản ghi thứ nhất
PRINT*,ST
READ (3) A, B, C, D ! Đọc bản ghi thứ hai
PRINT*, A, B, C, D
CLOSE (3)
END
```

Về hình thức, nói chung không có sự khác nhau nhiều trong cách tạo file và truy cập file giữa file không định dạng và file nhị phân. Sự khác nhau cơ bản giữa hai loại file này là tổ chức dữ liệu trong file. Ta sẽ xét đến vấn đề này ở những mục tiếp theo.

7.2.4 File truy cập tuần tự (Sequential-Access Files)

Dữ liệu trong file tuần tự cần phải được truy cập hợp lệ, bản ghi này tiếp nối sau bản ghi khác, trừ khi ta thay đổi vị trí con trỏ file bằng các câu lệnh **REWIND** hoặc **BACKSPACE**. Các phương pháp vào/ra có thể sử dụng đối với file truy cập tuần tự là **NONADVANCING**, **LIST–DIRECTED**, và **NAMELIST–DIRECTED**. Các file trong cần phải là file tuần tự. Đối với các file liên kết với các thiết bị tuần tự cần phải sử dụng cách truy cập tuần tự. Các thiết bị tuần tự là thiết bị lưu trữ vật lý. Bàn phím, màn hình và máy in là những thiết bị tuần tự.

7.2.5 File truy cập trực tiếp (Direct-Access Files)

Dữ liệu trong file truy cập trực tiếp có thể được đọc và ghi theo một trình tự bất kỳ. Các bản ghi được đánh số một cách tuần tự, bắt đầu từ 1. Tất cả các bản ghi có độ dài được chỉ ra bởi tham số tùy chọn **RECL** trong câu lệnh **OPEN**. Số liệu trong file truy cập trực tiếp được truy cập đến bằng việc chỉ ra số thứ tự bản ghi trong file.

7.3 TỔ CHỨC DỮ LIỆU TRONG FILE

Như đã thấy ở trên, với hai phương pháp truy cập file và ba dạng cấu trúc file, một cách tương đối ta có thể phân chia thành sáu dạng tổ chức dữ liệu trong file:

1) File truy cập tuần tự có định dạng (*Formatted Sequential*)

2) File truy cập trực tiếp có định dạng (*Formatted Direct*)

3) File truy cập tuần tự không định dạng (*Unformatted Sequential*)

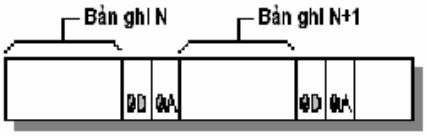
4) File truy cập trực tiếp không định dạng (*Unformatted Direct*)

5) File truy cập tuần tự dạng nhị phân (*Binary Sequential*)

6) File truy cập trực tiếp dạng nhị phân (*Binary Direct*)

7.3.1 File truy cập tuần tự có định dạng

File tuần tự có định dạng là một chuỗi các bản ghi có định dạng được ghi một cách tuần tự (hình 7.1) và được đọc theo thứ tự xuất hiện trong file. Các bản ghi có thể có độ dài biến đổi và có thể rỗng. Chúng được phân cách nhau bởi ký tự điều khiển **RETURN** (**$0D** hay **#13**) và ký tự xuống dòng (**$0A** hay **#10**).



Hình 7.1 Cấu trúc file tuần tự có định dạng

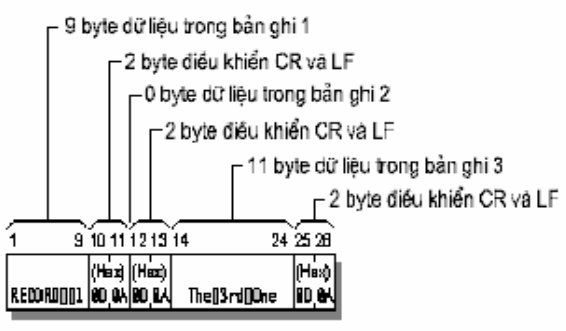
Ví dụ 7.6. Tạo file truy cập tuần tự có định dạng

```
OPEN (3, FILE='TEST1.TXT')
! TEST1.TXT ngầm định là file tuần tự có định dạng
WRITE (3, '(A, I3)') 'RECORD', 1
! Kết quả ghi ra là: RECORD001 (9 ký tự=9 byte)
WRITE (3, '()')  ! Bản ghi trống (0 byte)
WRITE (3, '(A11)') 'The 3rd One' ! 11 ký tự
CLOSE (3)
END
```

Như vậy, khi bỏ qua các tuỳ chọn **FORM=** và **ACCESS=** trong câu lệnh **OPEN** thì file sẽ được ngầm hiểu là file tuần tự có định dạng. Mô tả cấu trúc dữ liệu trong file **TEST1.TXT** được cho trên hình 7.2. Vì giữa các bản ghi được phân cách nhau bởi các ký tự điều khiển nên các bản ghi có thể có độ dài khác nhau tùy ý. Nếu sử dụng một trình soạn thảo nào đó để xem nội dung file ta sẽ thấy file có 3 dòng, trong đó dòng thứ hai là dòng trống:

```
RECORD  1

The 3rd One
```



Hình 7.2 Cấu trúc của file TEST1.TXT

7.3.2 File truy cập trực tiếp có định dạng

Trong file truy cập trực tiếp có định dạng, tất cả các bản ghi có cùng độ dài và có thể được ghi hoặc đọc theo thứ tự bất kỳ. Kích thước bản ghi được chỉ ra bởi tùy chọn **RECL=** trong câu lệnh **OPEN** và nên *bằng hoặc lớn hơn* số byte của bản ghi dài nhất. Các ký tự **RETURN** (**CR**) và xuống dòng (**LF**) là những ký tự phân cách giữa các bản ghi và không tính vào giá trị của **RECL**. Một khi bản ghi truy cập trực tiếp đã được ghi, nó không thể bị xóa nhưng vẫn có thể bị ghi đè.

Khi kết xuất (*output*) ra file truy cập trực tiếp có định dạng, nếu số liệu không lấp đầy hoàn toàn bản ghi, trình biên dịch sẽ đệm vào phần còn lại của bản ghi các dấu cách (**BLANK SPACES**). Các dấu cách bảo đảm rằng file chỉ chứa những bản ghi đã lấp đầy hoàn toàn và tất cả các bản ghi đều có cùng độ dài.

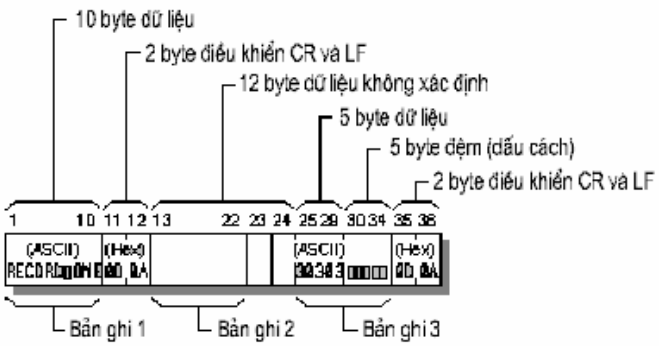
Khi đọc vào (*input*), trình biên dịch cũng ngầm định là có đệm các dấu cách vào nếu danh sách đọc vào và định dạng đòi hỏi nhiều dữ liệu hơn bản ghi đã chứa. Có thể bỏ qua ngầm định việc đệm vào các dấu cách ở dữ liệu vào bằng cách đặt tùy chọn **PAD**= "**NO**" trong câu lệnh **OPEN**. Khi đặt **PAD**= "**NO**", bản ghi đọc vào cần phải chứa lượng dữ liệu được chỉ ra bởi danh sách đầu vào và định dạng **FORMAT**, nếu không sẽ xuất hiện lỗi. **PAD**= "**NO**" không có ảnh hưởng đối với kết xuất.

Ví dụ 7.7. Đọc và ghi file truy cập trực tiếp có định dạng

```
character st*10
integer n
OPEN (3,FILE='TEST2.TXT',FORM='FORMATTED',&
    ACCESS='DIRECT',RECL=10)
st = 'RECORD ONE'
WRITE (3, '(A10)', REC=1) st ! Bản ghi thứ nhất
n= 30303
WRITE (3, '(I5)', REC=3) n   ! Bản ghi thứ ba
CLOSE (3)
OPEN (3,FILE='TEST2.TXT', FORM='FORMATTED', &
    ACCESS='DIRECT',RECL=10)
st = ' '
n=0
read(3,'(A10)',rec=1) st  ! Đọc bản ghi thứ nhất
read(3,'(I5)', rec=3) n   ! Đọc bản ghi thứ ba
print*,st
```

```
print*,n
END
```

Qua đó thấy rằng, để làm việc với file truy cập trực tiếp có định dạng, câu lệnh **OPEN** phải chứa các tham số tuỳ chọn **FORM** = '**FORMATTED**', **ACCESS** = '**DIRECT**' và **RECL** = ***Độ_dài_bản_ghi***. Mô tả tổ chức dữ liệu theo chương trình ở ví dụ 7.7 được cho trên hình 7.3. Độ dài của mỗi bản ghi là 10 byte, cộng với với 2 byte chứa ký tự điều khiển, nên những vị trí bản ghi chưa có dữ liệu sẽ có "khoảng trống" 12 byte không xác định. Đối với những bản ghi có dữ liệu chiếm ít hơn 10 byte, số byte còn lại sẽ được lấp đầy (đệm) bằng các dấu cách.



Hình 7.3 Cấu trúc file TEST2.TXT

7.3.3 File truy cập tuần tự không định dạng

File tuần tự không định dạng được tổ chức hơi khác nhau một ít giữa các dòng máy khác nhau cũng như giữa các trình biên dịch Fortran khác nhau. Sau đây ta sẽ xét đến loại file này đối với trình biên dịch Microsoft Fortran PowerStation.

Các bản ghi trong file tuần tự không định dạng có thể có độ dài biến đổi. File tuần tự không định dạng được tổ chức thành từng khúc 130 byte hoặc nhỏ hơn, được gọi là các khối vật lý. Mỗi khối vật lý bao gồm dữ liệu gửi vào file (cho đến 128 byte) và 2 byte chỉ độ dài do trình biên dịch chèn vào. Các byte độ dài cho biết mỗi bản ghi bắt đầu và kết thúc ở đâu. Mỗi bản ghi lôgic tham chiếu đến một bản ghi không định dạng chứa một hoặc nhiều hơn các khối vật lý. Các bản ghi lôgic có thể lớn tùy ý; trình biên dịch sẽ biết cung cấp số khối vật lý cần thiết để chứa.

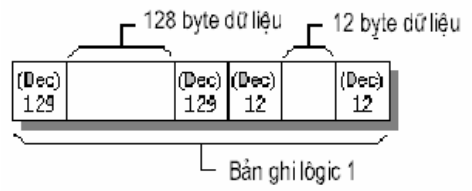
Khi tạo một bản ghi lôgic gồm nhiều hơn một khối vật lý, trình biên dịch đặt byte độ dài bằng 129 để chỉ rằng số liệu trong khối vật lý hiện tại sẽ nối tiếp vào khối vật lý tiếp theo. Ví dụ, một bản ghi lôgic có độ dài 140 byte sẽ được tổ chức như trên hình 7.4.

Ví dụ 7.8. Chương trình sau đây sẽ tạo một file tuần tự không định dạng. Cấu trúc dữ liệu trong file được mô tả trên hình 7.5.

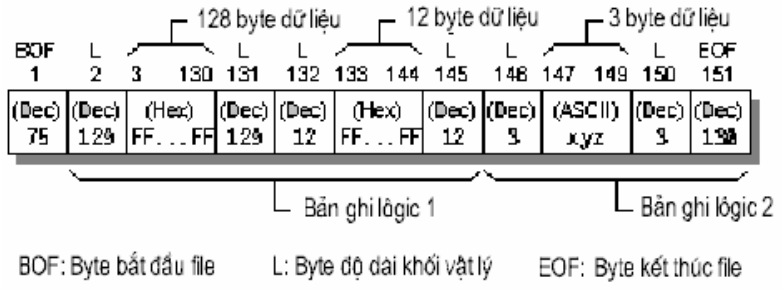
```
CHARACTER xyz(3)
INTEGER(4) idata(35)
DATA  idata /35 * -1/, xyz /'x', 'y', 'z'/
OPEN (3, FILE='TEST3.TXT',FORM='UNFORMATTED')
WRITE (3) idata
WRITE (3) xyz
```

```
CLOSE (3)
END
```

Ta thấy file dữ liệu được tạo gồm hai bản ghi lôgic. Bản ghi thứ nhất chứa dữ liệu của mảng ***idata*** gồm **35 x 4 = 140** byte. Bản ghi thứ hai chứa dữ liệu của mảng ***xyz,*** chiếm **3** byte. Vì bản ghi thứ nhất có độ dài lớn hơn 128 byte, nên nó được lưu trữ trên hai khối vật lý. Khối thứ nhất: từ byte thứ 2 đến byte thứ 131, với 128 byte dữ liệu và 2 byte chỉ độ dài được đặt bằng 129, hàm nghĩa rằng dữ liệu của bản ghi này vẫn còn được chứa tiếp ở khối tiếp theo. Khối thứ hai: từ byte thứ 132 đến byte thứ 145, gồm 12 byte dữ liệu và 2 byte độ dài được đặt bằng 12. Bản ghi thứ hai chỉ gồm 3 byte nên nó được chứa trọn vẹn trên một khối vật lý.



Hình 7.4 Cấu trúc file tuần tự không định dạng



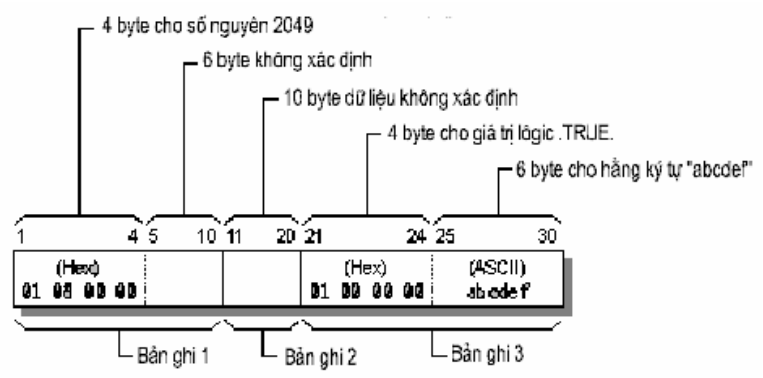
Hình 7.5 Cấu trúc file TEST3.TXT

7.3.4 File truy cập trực tiếp không định dạng

File truy cập trực tiếp không định dạng là một chuỗi các bản ghi không định dạng; có thể ghi hoặc đọc các bản ghi theo thứ tự tùy ý. Tất cả các bản ghi có cùng độ dài được cho bởi tham số **RECL=** trong câu lệnh **OPEN**. Giữa các bản ghi không có byte phân định ranh giới, hay nói cách khác, trong file truy cập trực tiếp không định dạng *không chứa thông tin về cấu trúc bản ghi*. Có thể ghi một phần bản ghi vào file truy cập trực tiếp không định dạng. Trình biên dịch Fortran sẽ đệm vào các bản ghi này những ký tự rỗng (**NULL**) **ASCII** để cho độ dài bản ghi là cố định. Những bản ghi trong file không ghi gì cả sẽ chứa các số liệu không xác định.

Ví dụ 7.9. Chương trình sau đây tạo một file truy cập trực tiếp không định dạng chứa hai bản ghi dữ liệu được ghi vào bản ghi thứ nhất và thứ ba. Bản ghi thứ hai không chứa dữ liệu. Mô tả cấu trúc dữ liệu trong file được cho trên hình 7.6.

```
OPEN (3, FILE='TEST4.TXT', RECL=10,    &
    FORM = 'UNFORMATTED', ACCESS = 'DIRECT')
WRITE (3, REC=3) .TRUE., 'abcdef'
WRITE (3, REC=1) 2049
CLOSE (3)
END
```



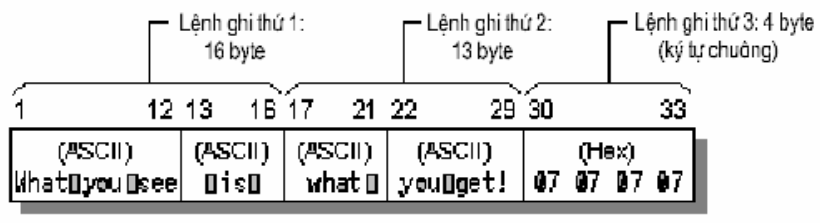
Hình 7.6 Cấu trúc file TEST4.TXT

7.3.5 File truy cập tuần tự dạng nhị phân

File truy cập tuần tự dạng nhị phân là một chuỗi các giá trị được ghi và đọc theo cùng trình tự và được lưu trữ như những số nhị phân. Trong file tuần tự dạng nhị phân không tồn tại ranh giới bản ghi, và không có byte đặc biệt để chỉ ra cấu trúc file. Số liệu được đọc và ghi không bị thay đổi dạng hoặc độ dài. Đối với mọi hạng mục vào/ra, tuần tự các byte trong bộ nhớ cũng chính là tuần tự các byte trong file.

Ví dụ 7.10. Chương trình sau đây tạo một file truy cập tuần tự dạng nhị phân gồm ba bản ghi có độ dài khác nhau. Cấu trúc dữ liệu trong file được mô tả trên hình 7.7.

```
INTEGER(1) Chuong(4)
CHARACTER(4) V1(3)
CHARACTER(4) V2
DATA   Chuong /4*7/
DATA  V2 /' is '/, V1 /'What',' you',' see'/
OPEN (3, FILE='TEST5.TXT',FORM='BINARY')
WRITE (3) V1, V2
WRITE (3) 'what ', 'you get!'
WRITE (3) Chuong
CLOSE (3)
END
```



Hình 7.7 Cấu trúc file TEST5.TXT

7.3.6 File truy cập trực tiếp dạng nhị phân

File truy cập trực tiếp dạng nhị phân lưu trữ các bản ghi như là một chuỗi các số nhị phân, có thể truy cập được theo trình tự bất kỳ. Các bản ghi trong file có cùng độ dài được chỉ ra bởi tham số

RECL= của câu lệnh **OPEN**. Có thể ghi một phần bản ghi vào file trực tiếp dạng nhị phân; phần chưa sử dụng của bản ghi sẽ chứa dữ liệu không xác định.

Một câu lệnh đơn **READ** hoặc **WRITE** có thể truyền tải dữ liệu nhiều hơn một bản ghi liên tiếp trong file. Tuy nhiên điều đó có thể gây nên lỗi.

Ví dụ 7.11. Chương trình sau đây tạo một file truy cập trực tiếp dạng nhị phân gồm 4 bản ghi. Sau đó đọc các thông tin từ file theo một trình tự và qui cách khác với cách tạo file. Bạn đọc cần nghiên cứu và phân tích lời chương trình một cách kỹ càng, sau đó chạy chương trình rồi nhận xét kết quả để hiểu rõ hơn nguyên tắc thao tác với file loại này.

```
character*20 st
OPEN(3, FILE='TEST6.TXT',RECL=10,FORM='BINARY', &
   ACCESS='DIRECT')
WRITE (3, REC=1) 'abcdefghijklmno'
  ! Ghi đè sang cả bản ghi thứ 2
WRITE (3) 4,5   ! Ghi vào bản ghi thứ 3
WRITE (3, REC=4) 'pq'
CLOSE (3)
OPEN (3, FILE='TEST6.TXT',RECL=10,FORM='BINARY',&
    ACCESS='DIRECT')
read(3,rec=3) i,j   ! Đọc bản ghi thứ 3
write(*,*) i,j
read(3,rec=1) st  ! Đọc sang cả bản ghi thứ 2
print*,st
read(3,rec=2) st  ! Đọc sang cả bản ghi thứ 3
write(*,*) st
END
```

Cấu trúc dữ liệu trong file được mô tả trên hình 7.8. Khi chạy chương trình ta sẽ nhận được những kết quả ngoài mong đợi. Khai báo độ dài của bản ghi là 10 byte, của biến ***st*** bằng 20 byte, gấp hai lần độ dài của một bản ghi, trong khi nội dung của ***st*** chỉ có 15 byte. Khi ghi vào file, ***st*** sẽ chiếm hai bản ghi, trong đó bản ghi thứ hai sẽ chứa 5 byte nội dung cuối cùng của ***st*** và 5 byte còn lại có nội dung không xác định. Do đó, khi đọc nội dung của bản ghi thứ hai (***rec=2***) và gán cho biến ký tự ta sẽ nhận được kết quả "***klmno***". Bản ghi thứ ba lưu hai số nguyên 4 byte, nên còn dư 2 byte không xác định. Tương tự như vậy đối với bản ghi thứ 4.

7.4 LỆNH MỞ (OPEN) VÀ ĐÓNG (CLOSE) FILE

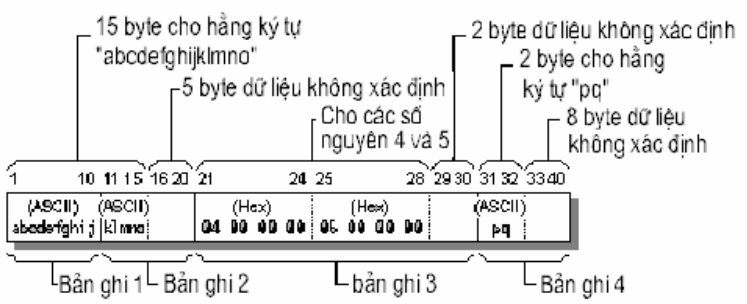
Ta đã làm quen với lệnh mở file (**OPEN**) và lệnh đóng file (**CLOSE**) qua các ví dụ mà chưa giải thích gì thêm. Trong mục này ta sẽ khảo sát kỹ hơn các câu lệnh này.

7.4.1 Lệnh mở file

Một cách tổng quát, cú pháp câu lệnh mở file có dạng:

```
OPEN ([UNIT=] unit
    [, ACCESS=access]
    [, ACTION=action]
    [, BLANK=blanks]
```

[, BLOCKSIZE=*blocksize*]
[, CARRIAGECONTROL=*carriagecontrol*]
[, DELIM=*delim*]
[, ERR=*err*]
[, FILE=*file*]
[, FORM=*form*]
[, IOFOCUS=*iofocus*]
[, IOSTAT=*iostat*]
[, PAD=*pad*]
[, POSITION=*position*]
[, RECL=*recl*]
[, SHARE=*share*]
[, STATUS=*status*])



Hình 7.8 Cấu trúc file TEST6.TXT

Trong mô tả trên đây, những từ viết in hoa được xem là các từ khoá xác định tham số, chúng phải được viết một cách chính xác, những từ viết in thường sau dấu (=) là giá trị các tham số. Như đã thấy, cú pháp đầy đủ của câu lệnh khá phức tạp bởi nó chứa rất nhiều tham số. Tuy vậy, hầu hết các tham số này đều là tùy chọn, ta có thể bỏ qua những tùy chọn không cần thiết trong quá trình sử dụng. Sau đây là ý nghĩa và cách sử dụng các tham số.

UNIT là tham số dùng để khai báo thiết bị lôgic sẽ được liên kết với file. Nếu bỏ qua **UNIT=** thì ***unit*** cần phải là tham số đầu tiên. Ngược lại, các tham số có thể xuất hiện theo thứ tự bất kỳ.

unit: Là một số nguyên (**INTEGER(4)**) lớn hơn hoặc bằng 0, dùng như một thiết bị lôgic để liên kết với file ngoài hoặc thiết bị ngoài.

access: Chỉ ra cách truy cập vào file, có thể nhận một trong các giá trị "**APPEND**" (ghi tiếp vào cuối file), "**DIRECT**" (truy cập trực tiếp), hoặc "**SEQUENTIAL**" (truy cập tuần tự – ngầm định).

action: Là tham số mô tả tác động dự định đối với file. Có thể nhận giá trị '**READ**' (file chỉ đọc), '**WRITE**' (chỉ để ghi vào file), hoặc **'READWRITE'** (cả đọc từ file và ghi vào file). Nếu bỏ qua ***action***, chương trình sẽ cố gắng mở file với **'READWRITE'**. Nếu không được, trước hết chương trình sẽ mở file với **'READ'**, sau đó với **'WRITE'**. Việc bỏ qua ***action*** không giống với **ACTION = 'READWRITE'**. Nếu **ACTION = 'READWRITE'**, khi mà file không thể truy cập bằng cả đọc và ghi thì việc mở file sẽ không thành công. Do đó việc bỏ qua ***action*** sẽ mềm dẻo và linh động hơn.

blanks: Có dạng ký tự (**Character*(*)**), dùng để điều khiển cách biểu diễn các ký tự trống (dấu cách) đối với vào/ra có định dạng, nhận một trong các giá trị **'NULL'** hoặc **'ZERO'**. Giá trị **'NULL'**

(ngầm định) để bỏ qua ký tự trống, giá trị '**ZERO**' để xử lý các ký tự trống (các dấu cách ở đầu) như là những số 0.

blocksize: Ngầm định là một số nguyên ((**INTEGER(4)**), dùng để biểu diễn kích thước vùng đệm (tính bằng byte).

carriagecontrol: Có dạng ký tự (**Character*(*)**), dùng để chỉ việc hiểu ký tự đầu tiên của bản ghi trong file có định dạng như thế nào; nhận một trong các giá trị '**FORTRAN**' hoặc '**LIST**'. Ngầm định đối với các **UNIT** kết nối với thiết bị ngoài như máy in hoặc màn hình là '**FORTRAN**'; ngầm định đối với các **UNIT** kết nối với các file là '**LIST**'. ***carriagecontrol*** bị bỏ qua khi sử dụng tùy chọn **FORM**= '**UNFORMATTED**' hoặc **FORM**='**BINARY**'.

delim: Có dạng ký tự (**Character*(*)**), dùng để chỉ cách định ranh giới các bản ghi trong ***list-directed*** hoặc ***namelist-formatted***; nhận một trong các giá trị '**APOSTROPHE**', '**QUOTE**', hoặc '**NONE**' (ngầm định).

err: Là nhãn của câu lệnh thực hiện trong chương trình. Khi gặp lỗi mở file chương trình sẽ chuyển điều khiển đến câu lệnh có nhãn ***err***. Nếu bỏ qua, hiệu ứng lỗi vào/ra sẽ được xác định bởi ***iostat***.

file: Có dạng ký tự (**Character*(*)**), dùng để chỉ ra tên file cần mở; có thể là dấu cách, tên file hợp lệ, tên thiết bị hoặc tên biến xác định file trong. Đối với **Windows NT** và **Windows 9x** trở lên, tên file cho phép dài hơn 8 ký tự, phần mở rộng dài hơn 3 ký tự. Nếu ***file*** bị bỏ qua, trình biên dịch sẽ tạo một file tạp nham (file nháp) nào đó chỉ có tên (không có phần mở rộng) và file này sẽ bị xóa khi gặp lệnh đóng file hoặc khi chương trình kết thúc.

form: Xác định kiểu file sẽ được mở, nhận một trong các giá trị '**FORMATTED**' (file có định dạng), '**UNFORMATTED**' (file không định dạng), hoặc '**BINARY**' (file nhị phân). Đối với file truy cập tuần tự, giá trị ngầm định là '**FORMATTED**'; đối với file truy cập trực tiếp giá trị ngầm định là '**UNFORMATTED**'.

iofocus: Tham số lôgic, dùng để chỉ việc có đặt cửa sổ con (child window) là cửa sổ hoạt động hay không. Giá trị **.TRUE**. (ngầm định) tạo ra lời gọi **SETFOCUSQQ** ngay lập tức trước khi thực hiện các lệnh **READ**, **WRITE**, hoặc **PRINT**.

iostat: Là tham số kết xuất, ngầm định là một số nguyên (**INTEGER(4)**); cho giá trị bằng *0* nếu không xuất hiện lỗi mở file, giá trị *âm* nếu gặp lỗi kết thúc bản ghi, hoặc một số xác định mã lỗi.

pad: Có dạng ký tự (**Character*(*)**), dùng để chỉ ra việc có đệm các dấu cách vào bản ghi khi đọc hay không nếu định dạng bản ghi có độ dài lớn hơn độ dài dữ liệu. ***pad*** nhận giá trị bằng '**YES**' (ngầm định) hoặc '**NO**'. Tham số này chỉ tác động khi đọc dữ liệu.

position: Có dạng ký tự (**Character*(*)**), chỉ ra vị trí con trỏ file truy cập tuần tự, nhận một trong các giá trị '**ASIS**' (ngầm định), '**REWIND**', hoặc '**APPEND**'. Nếu ***position*** nhận giá trị '**REWIND**', con trỏ file sẽ định vị tại đầu file; nếu nhận giá trị '**APPEND**', con trỏ file sẽ định vị tại

cuối file; nếu nhận giá trị '**ASIS**', con trỏ file không thay đổi vị trí (tức giữ nguyên vị trí hiện thời trong file). Vị trí con trỏ file đối với file mới luôn luôn ở đầu file.

recl: Ngầm định là số nguyên, để chỉ độ dài tính bằng byte của một bản ghi trong file truy cập trực tiếp, hoặc độ dài bản ghi cực đại đối với file tuần tự. Đối với file truy cập trực tiếp đây là tham số đòi hỏi phải có.

share: Có dạng ký tự (**Character*(*)**), chỉ ra quyền truy cập đối với file được mở. Các giá trị của ***share*** có ý nghĩa như sau:

share ='**DENYRW**': Cấm đọc/ghi

share ='**DENYWR**': Cấm ghi

share ='**DENYRD**': Cấm đọc

share ='**DENYNONE**': Cho phép cả đọc và ghi (ngầm định)

status: Có dạng ký tự (**Character*(*)**), dùng để mô tả trạng thái của file được mở. Sau đây là các giá trị có thể của tham số này.

status = **'OLD'**: File đang tồn tại. Mở thành công nếu file tồn tại, ngược lại sẽ xuất hiện lỗi;

status**='NEW'**: File mới. Nếu file không tồn tại nó sẽ được tạo mới. Nếu file đang tồn tại sẽ xuất hiện lỗi;

status**='SCRATCH'**: Nếu bỏ qua tham số ***file*** thì giá trị của ***status*** ngầm định là '**SCRATCH**'. File **SCRATCH** là file tạm thời, nó sẽ bị xóa khi đóng file hoặc khi chương trình kết thúc;

status**='REPLACE'**: File được mở thay thế file có cùng tên. Nếu file cùng tên không tồn tại thì một file mới được tạo ra.

status**='UNKNOWN'** (ngầm định): Trong lúc chương trình chạy hệ thống sẽ cố gắng mở file với ***status*** = **'OLD'**, và sau đó với ***status*** = **'NEW'**. Nếu file tồn tại thì nó được mở, nếu không tồn tại thì nó được tạo mới. Sử dụng ***status*** = **'UNKNOWN'** để tránh các lỗi xảy ra trong lúc chạy chương trình liên quan đến việc mở một file đang tồn tại với ***status*** ='**NEW**' hoặc mở một file không tồn tại với ***status*** = '**OLD**'.

Các giá trị của ***status*** chỉ ảnh hưởng tới các file trên đĩa, và được bỏ qua đối với các thiết bị như bàn phím và màn hình hoặc máy in.

7.4.2 Lệnh đóng file

Cú pháp lệnh đóng file **CLOSE** có dạng:

CLOSE ([UNIT=] ***unit***
[, ERR=*err***]**
[, IOSTAT=*iostat***]**
[, STATUS=*status***])**

Trong đó: nếu **UNIT=** bị bỏ qua thì ***unit*** phải là tham số đầu tiên, ngược lại các tham số có thể xuất hiện theo thứ tự bất kỳ.

unit: Là một số nguyên chỉ thiết bị lôgic liên kết với file được mở. Sẽ không xuất hiện lỗi nếu file không mở.

err: Là nhãn của câu lệnh thực hiện trong chương trình sẽ được chuyển điều khiển tới nếu lỗi xuất hiện khi đóng file. Nếu bỏ qua, hiệu ứng lỗi vào/ra được xác định bởi sự có mặt hoặc không có mặt của tham số ***iostat***.

iostat: Là tham số kết xuất, ngầm định là một số nguyên (**INTEGER*4**), nhận giá trị bằng 0 nếu không xuất hiện lỗi khi đóng file, hoặc một số chỉ mã lỗi.

status: Tham số vào, là một biểu thức ký tự (**CHARACTER*(*)**), nhận các giá trị "**KEEP**" hoặc "**DELETE**"; ngầm định là "**KEEP**", ngoại trừ file nháp. Các file được mở không có tham số **FILE=** được gọi là các file nháp ("***scratch***" files). Đối với những file này giá trị ngầm định của ***status*** là '**DELETE**'. Nếu đặt ***status*** ='**KEEP**' đối với những file nháp sẽ gây nên lỗi run-time.

7.5 CÁC LỆNH VÀO RA DỮ LIỆU VỚI FILE

7.5.1 Lệnh đọc dữ liệu từ file (READ)

Cú pháp lệnh **READ** làm việc với file có dạng:

READ { { *fmt* , | *nml* } |
([UNIT=] *unit*
[, { [FMT=] *fmt* |
[NML=] *nml* }]
[, ADVANCE=*advance*]
[, END=*end*]
[, EOR=*eor*]
[, ERR=*err*]
[, IOSTAT=*iostat*]
[, REC=*rec*]
[, SIZE=*size*]) } *iolist*

Trong đó dấu gạch đứng (|) dùng để phân chia các tham số thuộc một nhóm, có nghĩa là chỉ được phép chọn một trong các tham số của nhóm. Ví dụ, nếu đã chọn **[FMT=] *fmt*** thì không được phép chọn **[NML=] *nml***. Nếu bỏ qua **UNIT=**, thì ***unit*** phải là tham số đầu tiên. Nếu bỏ qua **FMT=** hoặc **NML=**, thì ***fmt*** hoặc ***nml*** phải là tham số thứ hai. Nếu muốn sử dụng ***fmt*** không có **FMT=**, thì phải bỏ qua **UNIT=**. Trong các trường hợp khác các tham số có thể xuất hiện theo thứ tự bất kỳ. Hoặc ***fmt*** hoặc ***nml*** cần được chỉ ra nhưng không phải cả hai. Sau đây là ý nghĩa các tham số.

unit: Là tên thiết bị lôgic. Khi đọc từ một file ngoài ***unit*** là một biểu thức nguyên. Khi đọc từ một file trong ***unit*** là một xâu ký tự, một biến ký tự hoặc một phần tử của mảng ký tự,...

fmt: Là chỉ thị định dạng, có thể nhận một trong các trường hợp: 1) Nhãn của lệnh **FORMAT**; 2) Biểu thức ký tự (biến, thủ tục, hoặc hằng) xác định định dạng đọc vào, phân định bởi các dấu nháy đơn (') hoặc dấu nháy kép ("); 3) Dấu sao (*) đối với trường hợp định dạng tự do; 4) hoặc một biến nguyên **ASSIGN**. Không được dùng ***fmt*** đối với **NAMELIST**. Nếu lệnh **READ** bỏ qua các tùy chọn

UNIT=, **END=**, **ERR=**, và **REC=**, và chỉ có ***fmt*** và ***iolist***, thì câu lệnh sẽ đọc từ **UNIT (*)** là bàn phím.

nml: Chỉ tên của **NAMELIST**. Các tùy chọn ***iolist*** và ***fmt*** phải được bỏ qua. Lệnh đọc **NAMELIST** chỉ có thể được thực hiện đối với file truy cập tuần tự.

advance: Có dạng ký tự (**Character*(*)**), dùng để chỉ ra cách đọc vào từ file tuần tự có định dạng, nhận giá trị hoặc '**YES**' (ngầm định) hoặc '**NO**'. Nếu ***advance*** = **'YES'**: Chương trình sẽ đọc theo định dạng ***fmt*** hết bản ghi này sang bản ghi khác và gán giá trị cho ***iolist***. Nếu ***advance*** = **'NO'**: Chương trình sẽ đọc các giá trị theo định dạng chỉ trên một dòng và gán cho ***iolist***. Nếu số giá trị chứa trong bản ghi ít hơn số biến trong ***iolist*** sẽ xuất hiện lỗi. Yêu cầu tham số định dạng ***fmt*** phải khác (*)

end: Nhãn của câu lệnh trong chương trình sẽ được chuyển điều khiển đến nếu con trỏ file đặt ở cuối bản ghi kết thúc file. Nếu bỏ quan ***end***, sẽ xuất hiện lỗi khi con trỏ file đã ở cuối file nhưng quá trình đọc vẫn cố gắng đọc tiếp, trừ trường hợp có chỉ ra ***err*** hoặc ***iostat***.

eor: Nhãn của câu lệnh trong chương trình sẽ được chuyển điều khiển đến nếu con trỏ file đặt ở cuối bản ghi. Tham số này chỉ dùng khi đưa vào tham số **ADVANCE='NO'**. Nếu bỏ qua ***eor***, hiệu ứng lỗi vào/ra sẽ được xác định bởi ***iostat***.

err: Nhãn của câu lệnh trong chương trình sẽ được chuyển đến nếu gặp lỗi trong quá trình đọc. Nếu bỏ qua ***err***, hiệu ứng lỗi vào/ra sẽ được xác định bởi ***iostat***.

iostat: Là tham số kết xuất, ngầm định là một số nguyên (**INTEGER(4)**). ***iostat*** = **0** nếu không có lỗi, = **–1** nếu gặp kết thúc file (*end-of-file*), hoặc bằng một số chỉ thị thông báo lỗi.

rec: Tham số vào, ngầm định là một số nguyên dương (**INTEGER(4)**) chỉ số thứ tự bản ghi cần đọc đối với file truy cập trực tiếp. Sẽ xuất hiện lỗi khi sử dụng tham số này cho file truy cập tuần tự hoặc file trong. Khi sử dụng tham số ***rec*** cần bỏ qua các tham số ***end*** và ***nml***. Con trỏ file sẽ được định vị đến bản ghi có số thứ tự ***rec*** trước khi dữ liệu được đọc. Số thứ tự bản ghi bắt đầu từ 1. Ngầm định của ***rec*** là số thứ tự bản ghi hiện thời.

size: Ngầm định là một số nguyên (**INTEGER(4)**), trả về số lượng ký tự được đọc và chuyển cho các biến nhận dữ liệu vào. Các dấu cách chèn đệm vào sẽ không được tính. Nếu sử dụng tham số này thì cần phải đặt tùy chọn **ADVANCE='NO'**.

iolist: Danh sách các biến sẽ được nhận dữ liệu từ file.

7.5.2 Lệnh ghi dữ liệu ra file (WRITE)

Cú pháp câu lệnh như sau.

```
WRITE ( [UNIT=] unit
      [, {[ FMT=] fmt  |
         [ NML=] nml}]
      [, ADVANCE=advance]
      [, ERR=err]
```

[, IOSTAT=*iostat*]
[, REC=*rec*]) *iolist*

Trong đó, dấu gạch đứng có ý nghĩa phân cách các tham số trong một nhóm mà chỉ có thể một trong chúng được phép xuất hiện.

Nếu bỏ qua **UNIT=** thì ***unit*** phải là tham số đầu tiên và ***fmt*** hoặc ***nml*** phải là tham số thứ hai (**FMT=** hoặc **NML=** có thể được bỏ qua). Ngược lại, các tham số có thể xuất hiện theo thứ tự bất kỳ. Trong hai tham số ***fmt*** và ***nml*** chỉ được phép xuất hiện một.

unit: Là tên thiết bị lôgic. Khi ghi ra file ngoài ***unit*** là một biểu thức nguyên gắn với định danh **UNIT**. Khi ghi ra file trong ***unit*** phải là xâu ký tự, biến ký tự, mảng hoặc phần tử mảng ký tự,... Nếu ***unit*** chưa liên kết với một file cụ thể thì lệnh mở file ẩn (*implicit*) được thực hiện. Ví dụ như câu lệnh sau:

OPEN (*unit*, FILE = ' ', STATUS = 'OLD',&
ACCESS = 'SEQUENTIAL', FORM = *form*)

trong đó ***form*** là '**FORMATTED**' đối với lệnh đọc/ghi có định dạng hoặc '**UNFORMATTED**' đối với lệnh đọc/ghi không định dạng.

fmt: Chỉ thị định dạng, có thể là nhãn câu lệnh **FORMAT**; biến, hàm hoặc hằng ký tự, trong đó kiểu định dạng được chỉ ra trong các cặp dấu nháy đơn (') hoặc nháy kép ("); biến nguyên **ASSIGN**; hoặc dấu sao (*).

nml: Chỉ ra tên của **NAMELIST**. Nếu tham số này được chọn thì các tham số ***iolist*** và ***fmt*** phải được bỏ qua. File chứa **NAMELIST** phải là file truy cập tuần tự.

advance: Có dạng ký tự (**Character*(*)**), chỉ ra cách ghi ra file là tiến (*advancing*) hay không. Nếu ***advance*****='YES'** (ngầm định) sẽ tạo ra đánh dấu vị trí ở cuối bản ghi; nếu ***advance*****='NO'** sẽ ghi một phần của bản ghi (tức chưa tạo ra kết thúc bản ghi).

err: Nhãn của câu lệnh thực hiện trong chương trình sẽ được chuyển điều khiển đến khi gặp lỗi. Nếu bỏ qua tham số này, hiệu ứng lỗi vào/ra sẽ được xác định bởi tham số ***iostat***.

iostat: Là tham số kết xuất, ngầm định là một số nguyên (**INTEGER(4)**), bằng 0 nếu không có lỗi, hoặc bằng một số xác định mã lỗi.

rec: Tham số vào, ngầm định là một số nguyên dương (**INTEGER(4)**), chỉ số thứ tự bản ghi trong file sẽ được ghi vào file truy cập trực tiếp. Khi sử dụng tham số ***rec*** thì các tham số ***end*** và ***nml*** cần phải bỏ qua. Con trỏ file phải được định vị tại bản ghi ***rec*** trước khi dữ liệu được ghi. Giá trị ngầm định của ***rec*** là số thứ tự bản ghi hiện thời.

iolist: Danh sách các biến sẽ được ghi, liệt kê cách nhau bởi dấu phẩy (,).

7.5.3 Vào ra dữ liệu với NAMELIST

Vào ra dữ liệu bằng **NAMELIST** là một phương pháp rất hữu hiệu của Fortran. Bằng cách chỉ ra một hoặc nhiều biến trong nhóm tên danh sách ta có thể đọc hoặc ghi các giá trị của chúng chỉ với một câu lệnh đơn giản.

Nhóm ***namelist*** được tạo bởi câu lệnh **NAMELIST** có dạng:

NAMELIST / *namelist* / *variablelist*

Trong đó ***namelist*** là tên nhóm và ***variablelist*** là danh sách các biến.

Lệnh **NAMELIST** khi đọc vào sẽ kiểm duyệt tên nhóm trong file ***NAMELIST***. Sau khi tìm thấy tên nhóm nó sẽ kiểm duyệt các câu lệnh gán giá trị cho các biến trong nhóm.

Trong file ***NAMELIST***, các nhóm được bắt đầu bởi dấu và (**&**) hoặc dấu đôla (**$**), và kết thúc bằng dấu gạch chéo (/), dấu và (**&**), hoặc dấu đôla (**$**).

Từ khóa **END** có thể xuất hiện liền ngay sau các dấu kết thúc (**&**) hoặc (**$**) (không có dấu cách) nhưng không được phép xuất hiện sau dấu gạch chéo (/).

Ví dụ, giả sử có:

```
INTEGER a, b
NAMELIST /mynml/ a, b
...
```

Các lệnh gán trong file ***NAMELIST*** sau đây là hợp lệ:

```
&mynml a = 1 /
$mynml a = 1, b = 2, $
$mynml a = 1, b = 2, $end
&mynml a = 1, b = 2, &
&mynml a = 1, b = 2, $END
&mynml
  a = 1
  b = 2
/
```

a. Lệnh đọc nội dung NAMELIST

Cú pháp:

READ (UNIT=*unit*, [NML=] *namelist*)

Trong đó ***unit*** là thiết bị lôgic lưu trữ thông tin của **NAMELIST**, có thể là file trên đĩa hoặc bàn phím, ***namelist*** là tên của **NAMELIST**. Nếu ***unit*** là dấu sao (*) thì **NAMELIST** được nhận từ bàn phím. Trong trường hợp này cần phải gõ vào theo đúng qui cách. Ví dụ:

```
INTEGER a, b
NAMELIST /mynml/ a, b
read(*,mynml)
WRITE(*,mynml)
END
```

Khi chạy chương trình này ta phải gõ vào, chẳng hạn:

&mynml a = 1, b = 2, /

Một ví dụ khác, giả sử **NAMELIST** có tên ***example*** chứa trong file liên kết với thiết bị lôgic 4 (***unit*=4**) với nội dung:

```
&example
    Z1 = (99.0,0.0)
    INT1=99
    array(1)=99
    REAL1 = 99.0
    CHAR1='Z'
    CHAR2(4:9) = 'Inside'
    LOG1=.FALSE.
/
```

Ta có thể dùng câu lệnh sau để đọc nội dung **NAMELIST** này:

READ (UNIT = 4, example)

Hoặc, giả sử ta có chương trình

```
INTEGER, DIMENSION(4) :: A = 7
NAMELIST /MYOUT/A, X, Y
X = 1
Y = 1
PRINT*,'Cho noi dung NAMELIST (A(1:4),X,Y):'
READ( *, MYOUT )
WRITE( *, NML = MYOUT )
END
```

Khi chạy chương trình này ta nhập vào (từ bàn phím):

&MYOUT A(1:2) = 2*1 Y = 3 /

và sẽ nhận được (trên màn hình):

```
 &MYOUT A = 1 1 7 7, X = 1.0000000, Y = 3.0000000
```

b. Lệnh in nội dung NAMELIST

Để in nội dung các nhóm **NAMELIST** có thể sử dụng lệnh:

WRITE (UNIT=*unit,* [NML=] *namelist*)

NML= là một tùy chọn, chỉ đòi hỏi phải có nếu các từ khóa khác được sử dụng.

Ví dụ, chương trình sau đây gán các giá trị của **NAMELIST** và in nội dung lên màn hình.

```
INTEGER(1) int1
INTEGER int2, int3, array(3)
LOGICAL(1) log1
LOGICAL log2, log3
REAL       real1
REAL(8)    real2
COMPLEX    z1, z2
CHARACTER(1)  char1
CHARACTER(10) char2
```

```
NAMELIST /example/ int1, int2, int3, &
    log1, log2, log3, real1, real2, &
    z1, z2, char1, char2, array
int1    = 11
int2    = 12
int3    = 14
log1    = .TRUE.
log2    = .TRUE.
log3    = .TRUE.
real1   = 24.0
real2   = 28.0d0
z1      = (38.0, 0.0)
z2      = (316.0d0, 0.0d0)
char1   = 'A'
char2   = '0123456789'
array(1) = 41
array(2) = 42
array(3) = 43
WRITE (*, example)
END
```

Khi chạy chương trình ta sẽ nhận được trên màn hình:

```
&EXAMPLE
INT1 =        11
INT2 =        12
INT3 =        14
LOG1 =  T
LOG2 =  T
LOG3 =  T
REAL1 =      24.000000
REAL2 =      28.000000
Z1 =          (38.000000,0.000000E+00)
Z2 =         (316.000000,0.000000E+000)
CHAR1 =  A
CHAR2 =  0123456789
ARRAY =          41         42         43
/
```

Ta cũng có thể sửa đổi chương trình để ghi **NAMELIST** vào file.

7.5.4 Một số ví dụ thao tác với file

1) Chương trình sau đây tạo một file có tên file do ta xác định, sau đó đọc nội dung từng bản ghi trong file và lần lượt hỏi ta có xóa hay không. Kết quả trung gian được ghi vào một file nháp. Nội dung của file được tạo sẽ được phục hồi lại từ file nháp này.

```
CHARACTER(80) Name, FileName, Ans
WRITE( *, '(A)', ADVANCE = 'NO' ) "Name of file: "
READ*, FileName
```

```
OPEN( 1, FILE = FileName )
OPEN( 2, STATUS = 'SCRATCH' )
write (1,'(A)') 'TEST1 Only'
write (1,'(A)') 'TEST2 Only'
write (1,'(A)') 'TEST3 Only'
rewind(1)
IO = 0
DO WHILE (IO == 0)
 READ( 1, '(A)', IOSTAT = IO ) Name
 print*,Name
 IF (IO == 0) THEN
  WRITE(*,'(A)',ADVANCE='NO')“Xoa khong (Y/N)?"
  READ*, Ans
  IF(Ans/='Y'.AND.Ans/='y') WRITE( 2, * ) Name
 END IF
END DO
REWIND( 2 )
CLOSE( 1, STATUS = 'DELETE' )
OPEN( 1, FILE = FileName )
IO = 0
DO WHILE (IO == 0)
 READ( 2, '(A)', IOSTAT = IO ) Name
 IF (IO == 0) WRITE( 1, * ) Name
END DO
CLOSE( 1 )
CLOSE( 2 )
END
```

2) Chương trình sau đây tạo một file tuần tự không định dạng. Chú ý cách truy cập đến các bản ghi của nó.

```
INTEGER, DIMENSION(10) :: A = (/ (I, I = 1,10) /)
INTEGER, DIMENSION(10) :: B = (/ (I, I = 11,20) /)
OPEN( 1, FILE = 'TEST', FORM = 'UNFORMATTED' )
WRITE (1) A    ! ghi A trước
WRITE (1) B    ! ghi B sau
REWIND (1)
A = 0
B = 0
READ (1) B    ! đọc B trước
PRINT*, B
READ (1) A    ! đọc A sau
PRINT*, A
CLOSE (1)
END
```

3) Chương trình sau đây tạo file truy cập trực tiếp.

```
CHARACTER (20) NAME
INTEGER I
LEN=20
```

```
OPEN( 1, FILE = 'TEST.txt', STATUS = 'REPLACE',&
    ACCESS = 'DIRECT', RECL = LEN )
DO I = 1, 6
  PRINT*,' Cho xau ky tu thu ', I
  READ*, NAME
  WRITE (1, REC = I) NAME
END DO
PRINT*,' Cac xau da nhap:'
DO I = 1, 6
  READ( 1, REC = I ) NAME
  PRINT*,'Xau thu ',I,': ', NAME
END DO
 ! Ghi de (thay doi) noi dung ban ghi thu 3
WRITE (1, REC = 3) 'Ban ghi thu 3'
PRINT*,' Cac xau sau khi sua doi:'
DO I = 1, 6
  READ( 1, REC = I ) NAME
  PRINT*, NAME
END DO
CLOSE (1)
END
```

4) Chương trình sau đọc từng ký tự trong từng bản ghi và in ra nội dung và số ký tự của bản ghi. Hãy lưu ý cách sử dụng các tham số trong lệnh **READ**.

```
CHARACTER ch*1, ST(100)
INTEGER IO, Num, EOR
OPEN( 1, FILE = 'TEST.TXT' )
DO I=1,10
   WRITE(1,'(10I3)') (J,J=I+1,I+10)
END DO
REWIND (1)
IO = 0
DO WHILE (IO /= -1)      ! EOF
! DO WHILE (IO == 0)     ! Khong co loi
Num = 0
do
  READ (1, '(A1)', IOSTAT = IO,&
     ADVANCE = 'NO',EOR=10) ch  ! Doc tung ky tu
  Num = Num + 1
  ST(Num) = ch  ! Luu vao bien ST
end do
10 PRINT*,Num   ! In so ky tu
PRINT*,ST       ! In noi dung ban ghi
END DO
CLOSE (1)
END
```

5) Chương trình sau đây nhập vào một mảng số nguyên và một mảng số thực với số phần tử tùy ý (trong chương trình chỉ hạn chế tối đa là 10 phần tử). Sau đó in các phần tử của mỗi mảng này trên

cùng dòng. Chú ý cách tạo lệnh định dạng **FORMAT** sao cho có thể in được số phần tử và độ rộng trường tùy ý.

```
INTEGER WI, N(10), ICOUNT, XCOUNT
REAL X(10)
ICOUNT=1
DO
  PRINT*,' CHO GIA TRI N(',ICOUNT,') (-999=THOAT):'
  READ*,N(ICOUNT)
  IF (N(ICOUNT)==-999.OR.ICOUNT==10) EXIT
  ICOUNT=ICOUNT+1
END DO
XCOUNT=1
DO
  PRINT*,' CHO GIA TRI X(',XCOUNT,') (-999=THOAT):'
  READ*,X(XCOUNT)
  IF (X(XCOUNT)==-999.OR.XCOUNT==10) EXIT
  XCOUNT=XCOUNT+1
END DO
PRINT*,' CHO DO RONG TRUONG SO NGUYEN:'
READ*, WI
WRITE(*, 11) (X(I),I=1,XCOUNT-1)
11 FORMAT(2X,<XCOUNT-1>F8.1)
WRITE(*, 10) (N(I),I=1,ICOUNT-1)
10 FORMAT(<ICOUNT-1>I <WI>)
WRITE(*,*)
END
```

BÀI TẬP CHƯƠNG 7

7.1 Khảo sát đoạn chương trình sau và cho biết cấu trúc của các file đang xử lý:

```
INTEGER, PARAMETER :: NX=201, NY=161, NZ=16
INTEGER NDIG, MDIG
CHARACTER :: FMT*20
REAL*4 X (NX, NY, NZ), OUT(NX, NY)
...
OPEN (1, FILE=”RAIN.DAT”, FORM=”UnFormatted”,&
    ACCESS=”Direct”, RECL=NX*NY*4, STATUS=”Old”)
DO K=1,NZ
   READ(1,REC=K) OUT
   X(:,:,K) = OUT(:,:)
END DO
...
OPEN (3, FILE=”RAIN.TXT”,STATUS=”UnKnown”)
NDIG=12
MDIG=4
WRITE(FMT,"(A,I3.3,A,I2.2,A,I1,A)") &
        '(', NX,'F', NDIG,'.', MDIG,')'
DO J=NY, 1, −1
   WRITE(3,FMT) ((X(I,J,K), I=1,NX),K=1,NZ)
```

```
END DO
END
```

7.2 Khảo sát đoạn chương trình sau và cho biết cấu trúc của các file đang xử lý:

```
INTEGER NX, NY, NZ
REAL*4 X (NX, NY, NZ), OUT(NX, NY)
...
OPEN (1, FILE=”DATA.TXT”,STATUS=”Old”)
READ (1, *) NX, NY, NZ
DO K=1, NZ
   DO J=1,NY
     READ (1,*) (X(I,J,K), I=1,NX)
   END DO
END DO
...
OPEN (3, FILE=”DATA.DAT”, FORM=”UnFormatted”,&
          STATUS=”UnKnown”)
DO K=1,NZ
   OUT (:,:) = X(:,:,K)
   WRITE (3) OUT
END DO
...
END
```

7.3 Cho file số liệu dạng TEXT chứa kết quả quan trắc của các biến X_1, X_2, …, X_m. Cấu trúc file như sau. Dòng 1 là tiêu đề mô tả nội dung file. Dòng 2 là hai số nguyên dương (N, M) chỉ số lần quan trắc (N – dung lượng mẫu) và số biến (M). Các dòng tiếp theo mỗi dòng chứa M số thực là giá trị quan trắc x_{i1}, x_{i2}, …, x_{im} lần thứ i (i=1,2,…,N) của các biến X_1, X_2, …, X_m, các giá trị được viết cách nhau ít nhất một dấu cách. Hãy viết chương trình đọc file số liệu và ghi vào một file mới theo qui cách không định dạng truy cập tuần tự với đầy đủ nội dung của file đã cho trừ dòng tiêu đề.

7.4 Viết chương trình đọc file không định dạng truy cập tuần tự đã tạo ra ở bài tập 7.3 và ghi vào một file mới theo qui cách không định dạng truy cập trực tiếp. Hãy kiểm tra lại kết quả bằng cách đọc lại nó.

7.5 Cho file số liệu dạng TEXT chứa kết quả quan trắc của biến Y (biến phụ thuộc) và các biến X_1, X_2, …, X_m (biến độc lập). Cấu trúc file như sau. Dòng 1 là tiêu đề mô tả nội dung file. Dòng 2 là hai số nguyên dương (N, M) chỉ số lần quan trắc (N – dung lượng mẫu) và số biến độc lập (M). Các dòng tiếp theo mỗi dòng chứa M+1 số thực là giá trị quan trắc y_i, x_{i1}, x_{i2}, …, x_{im} lần thứ i (i=1,2,…,N) của các biến Y, X_1, X_2, …, X_m, các giá trị được viết cách nhau ít nhất một dấu cách. Hãy viết chương trình đọc file số liệu vào ghi vào một file mới theo qui cách định dạng nhị phân (BINARY) truy cập tuần tự với đầy đủ nội dung của file đã cho, kể cả dòng tiêu đề.

7.6 Viết chương trình đọc file định dạng nhị phân (BINARY) đã tạo ra ở bài tập 7.5 và ghi vào một file dạng TEXT như đã mô tả trong bài tập đó.

7.7 Viết chương trình tạo một file không định dạng truy cập trực tiếp chứa 20 mảng một chiều gồm các số nguyên, kích thước mảng là 10. Đọc các mảng thứ 5, 10, 15 và 20 từ file và in ra để kiểm tra.

7.8 Viết chương trình nhập vào 2 mảng một chiều kích thước 10 phần tử là những số nguyên và thay thế nội dung của mảng thứ 5 và thứ 10 trong file tạo ra ở bài tập 7.7 tương ứng bởi hai mảng này. Đọc lại nội dung các mảng này từ file và in ra để kiểm tra.

CHƯƠNG 8. MỘT SỐ KIẾN THỨC MỞ RỘNG

8.1 KHAI BÁO DÙNG CHUNG BỘ NHỚ

Trong nhiều lớp bài toán, vấn đề dùng chung bộ nhớ sẽ quyết định khả năng giải được của bài toán liên quan đến tài nguyên bộ nhớ và tốc độ của máy tính. Fortran hỗ trợ một vài phương thức dùng chung bộ nhớ, làm tăng khả năng chia sẻ dữ liệu giữa các đơn vị chương trình cũng như làm tăng tốc độ truy cập bộ nhớ. Trong mục này ta sẽ xét hai phương thức dùng chung bộ nhớ là sử dụng lệnh **COMMON** và lệnh **EQUIVALENT**.

8.1.1 Lệnh COMMON

Cú pháp câu lệnh **COMMON** có dạng:

COMMON [/[*cname*]/] *list* [[,]/[*cname*]/ *list*] ...

Trong đó: ***cname***: (Tùy chọn) là tên của khối dùng chung mà các biến trong ***list*** thuộc khối đó. Nếu bỏ qua ta nói đó là khối chung "trắng" ("*blank common*"); ***list***: Là một hoặc nhiều tên biến, tên mảng được phép dùng chung vùng nhớ, chúng được liệt kê cách nhau bởi dấu phẩy.

Các biến, mảng trong các khối dùng chung giữa các đơn vị chương trình phải tương ứng về vị trí và độ dài. Các mảng phải có cùng kích thước. Các biến trong khối chung không thể là đối số hình thức, mảng động, tên hàm,... Chúng cũng không thể là những hằng được khai báo bởi lệnh **PARAMETER**.

Tác động của khối dùng chung là ở chỗ, khi các đơn vị chương trình khác nhau có khai báo dùng chung cùng một tên khối chung ***cname*** thì, mặc dù danh sách tên biến trong ***cname*** ở các đơn vị chương trình có thể khác nhau, chúng vẫn tham chiếu đến cùng những vùng bộ nhớ.

Ví dụ, giả sử ta có các đơn vị chương trình sau:

```
PROGRAM MyProg
COMMON i, j, x, k(10)
COMMON /mycom/ a(3)
A=5
CALL MySub
PRINT*,A
END
!
SUBROUTINE MySub
COMMON je, mn, z, idum(10)
COMMON /mycom/ B(3)
B = B + 5
END
```

Khi chương trình thực hiện, các biến **je, mn, z, idum** trong **MySub** sẽ tham chiếu đến các vùng bộ nhớ tương ứng của các biên **I, J, X, K** trong **MyProg**; biến **B** trong **MySub** và biến **A** trong **MyProg** cùng dùng chung một vùng bộ nhớ. Do đó, lời gọi **CALL MySub** trong đó **B** bị thay đổi cũng có nghĩa là **A** bị biến đổi.

8.1.2 Lệnh EQUIVALENT

Phương thức khai báo dùng chung **EQUIVALENCE** làm cho hai hoặc nhiều biến hoặc mảng chiếm cùng một vùng bộ nhớ. Cú pháp câu lệnh có dạng:

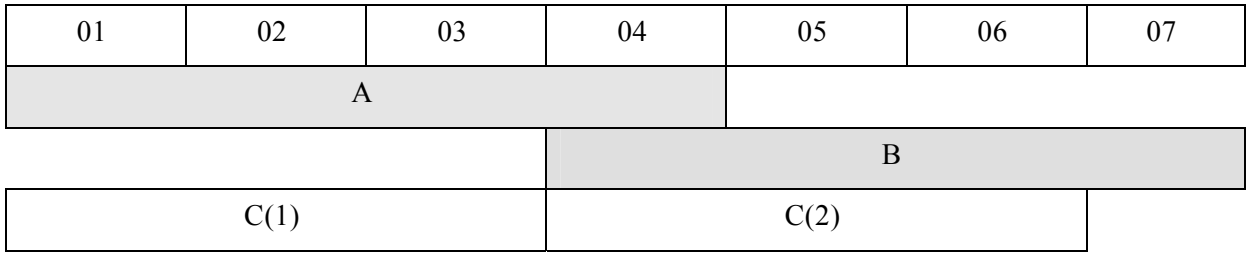
EQUIVALENCE (*nlist*) [, (*nlist*)] ...

Trong đó ***nlist*** là hai hoặc nhiều hơn các biến, mảng hoặc phần tử mảng, viết cách nhau bởi dấu phẩy (,). Các chỉ số mảng cần phải là những hằng số nguyên và phải nằm trong miền giá trị của kích thước mảng. Những tên biến mảng không có chỉ số được ngầm hiểu là phần tử có chỉ số đầu tiên của mảng. Tất cả các phần tử trong ***nlist*** đều có cùng vị trí đầu tiên trong vùng bộ nhớ. Để minh họa ta hãy xét ví dụ sau.

Giả sử ta có khai báo:

```
CHARACTER  a*4, b*4, c(2)*3
EQUIVALENCE (a, c(1)), (b, c(2))
```

Khi đó định vị bộ nhớ sẽ có dạng:



Nếu không khai báo dùng chung, dung lượng bộ nhớ phải cấp cho các biến **A, B, C** sẽ là **4 + 4 + 2*3 = 14** byte. Nhưng khi sử dụng khai báo dùng chung bằng **EQUIVALENT**, dung lượng bộ nhớ cấp cho 3 biến này chỉ cần **7** byte như trên hình vẽ. Từ hình vẽ, có thể hình dung rằng, **3** ký tự của phần tử **C(1)** dùng chung bộ nhớ với **3** ký tự đầu của xâu **A**; ký tự thứ tư của xâu **A**, ký tự thứ nhất của xâu **B** và ký tự thứ nhất của phần tử **C(2)** dùng chung **1** byte; hai cặp ký tự tiếp theo của xâu **B** và của **C(2)** dùng chung các byte thứ **5** và thứ **6**; chỉ có ký tự cuối cùng của xâu **B** là không dùng chung. Do đó, tùy theo biến nào bị thay đổi giá trị cuối cùng mà các biến khác sẽ bị biến đổi theo. Ví dụ, trong khai báo trên, nếu tuần tự câu lệnh là:

```
A='abcd'
B='efgh'
C(1)='ijk'
C(2)='LMN'
```

thì kết quả cuối cùng sẽ là:

C(1)='ijk' **B='LMNh'**
C(2)='LMN' **A='ijkL'**

Một ví dụ khác, giả sử ta có chương trình:

```
INTEGER M(6), N(10), MN(8)
EQUIVALENCE (N,M), (N(7),MN)
N=4
M=3
MN=5
PRINT*,M
PRINT*,N
PRINT*,MN
END
```

Khi chạy chương trình này ta sẽ nhận được kết quả:

```
MN=(5, 5, 5, 5, 5, 5, 5, 5)
N =(3, 3, 3, 3, 3, 3, 5, 5, 5, 5)
M =(3, 3, 3, 3, 3, 3)
```

8.2 CHƯƠNG TRÌNH CON BLOCK DATA

Chương trình con **BLOCK DATA** là một loại đơn vị chương trình cho phép xác định các giá trị khởi tạo cho các biến trong các khối dùng chung. Chương trình con này chứa các câu lệnh không thực hiện. Cấu trúc chương trình có dạng:

```
BLOCK DATA [name]
  [Các_câu_lệnh]
END [BLOCK DATA [name]]
```

Thông thường các biến được khởi tạo bằng các lệnh **DATA**. Các biến trong khối dùng chung cũng có thể được khởi tạo. **BLOCK DATA** có thể chứa các câu lệnh: **COMMON, USE, DATA, PARAMETER, DIMENSION, POINTER, EQUIVALENCE, SAVE, IMPLICIT**.

Ví dụ:

```
COMMON /WRKCOM/ X, Y, Z (10,10)
PRINT*,X
PRINT*,Y
PRINT*,Z
END
BLOCK DATA WORK
  COMMON /WRKCOM/ A, B, C (10,10)
  DATA A /1.0/, B /2.0/ , C /100*5.0/
END BLOCK DATA WORK
```

8.3 CÂU LỆNH INCLUDE

Trong nhiều trường hợp, một số đoạn chương trình hoặc đã được chuẩn hóa, ít thay đổi, hoặc có thể xuất hiện ở các đơn vị chương trình khác nhau,… nhưng chúng chưa đủ để tạo lập riêng một chương trình con, ta có thể tách chúng ra và mỗi đoạn gồm những dòng lệnh liên tiếp nhau được ghi vào một file riêng biệt. Sau đó thay thế vào vị trí của các đoạn chương trình này câu lệnh **INCLUDE** để "trả lại" nội dung nguyên bản của nó. Cú pháp câu lệnh như sau.

```
INCLUDE filename
```

trong đó ***filename*** là hằng ký tự chỉ tên file, bao gồm cả đường dẫn, chứa nội dung của đoạn chương trình đã được tách từ đơn vị chương trình. Cách làm này giúp ta tổ chức chương trình gọn nhẹ, dễ bao quát hơn. Khi gặp lệnh **INCLUDE** trình biên dịch sẽ tìm đến file có tên là ***filename*** và chèn nội dung của file vào vị trí của dòng lệnh rồi mới tiến hành biên dịch cùng với đơn vị chương trình. Như vậy, tác dụng của lệnh **INCLUDE** chỉ làm cho chương trình gọn hơn về hình thức.

Ví dụ, giả sử ta có file "**PARAM.INC**" lưu tại thư mục hiện thời với nội dung là:

```
INTEGER, PARAMETER :: NMAX=200, MMAX=100
REAL, PARAMETER :: Re=6731, G=9.8
```

Khi đó chương trình sau đây:

```
PROGRAM CT1
INCLUDE "PARAM.INC"
…
END
```

sẽ tương đương với chương trình

```
PROGRAM CT2
INTEGER, PARAMETER :: NMAX=200, MMAX=100
REAL, PARAMETER :: Re=6731, G=9.8
…
END
```

8.4 LỆNH INQUIRE

Chức năng của câu lệnh này là truy vấn về trạng thái, thuộc tính của file hoặc dung lượng chiếm giữ bộ nhớ của biến. Cú pháp tổng quát của câu lệnh khá dài, tương tự như câu lệnh OPEN. Ở đây sẽ đưa ra ba dạng đơn giản với các tham số tùy chọn thường được sử dụng.

Dạng 1:

INQUIRE (FILE = *Tên_file*, *Tùy_chọn* = *Chọn*)

Dạng 2:

INQUIRE ([*UNIT* =] *Unit*, *Tùy_chọn* = *Chọn*)

Dạng 3:

INQUIRE (INLENGTH = *Len*) *vname*

Trong đó: ***Tên_file*** là tên của file sẽ được truy vấn; ***UNIT*** là định danh chỉ số hiệu file; ***Unit*** là số hiệu file; ***Chọn*** là biến nhận giá trị trả về của ***Tùy_chọn***; ***vname*** là tên của biến/bản ghi; ***Len*** là dung lượng chiếm giữ bộ nhớ (độ dài) của biến/bản ghi; và ***Tùy_chọn*** là tham số tùy chọn, có thể nhận các dạng sau đây (kiểu dữ liệu trả về được viết trong dấu ngoặc đơn tương ứng):

EXIST (*logical*): **TRUE** nếu file tồn tại, **FALSE** nếu file không tồn tại.

OPENED (*logical*): **TRUE** nếu file đã được kết nối, **FALSE** nếu file chưa được kết nối.

NUMBER (*integer*): Giá trị chỉ số hiệu file được kết nối, hoặc bằng –1 nếu không có số hiệu file nào được kết nối.

NAMED (*logical*): **TRUE** nếu file đã được đặt tên, **FALSE** nếu file chưa được đặt tên.

NAME (*character*): Trả về tên file nếu file đã được đặt tên.

ACCESS (*character*): Nhận giá trị **SEQUENTIAL**, **DIRECT**, hoặc **UNDEFINED** (nếu chưa kết nối).

SEQUENTIAL và ***DIRECT*** (*character*): Nhận giá trị **YES**, **NO** hoặc **UNKNOWN**, tùy thuộc kiểu truy cập cho phép.

FORM (*character*): Nhận giá trị **FORMATTED**, **UNFORMATTED**, hoặc **UNDEFINED**.

RECL (*integer*): Độ dài cực đại của bản ghi.

NEXTREC (*integer*): Số thứ tự của bản ghi vừa mới được đọc hoặc ghi.

POSITION (*character*): **REWIND**, **APPEND**, **ASIS** hoặc **UNDEFINED** (như lệnh **OPEN**).

ACTION (*character*): **READ, WRITE, READWRITE** hoặc **UNDEFINED**.

READ, ***WRITE*** và ***READWRITE*** (*character*): **YES**, **NO** hoặc **UNKNOWN**.

Ví dụ 8.1. Chương trình sau đây đòi hỏi ta nhập vào tên file và sẽ kiểm tra sự tồn tại của file đó. Nếu file có tên được nhập vào không tồn tại, chương trình sẽ yêu cầu nhập lại cho đến khi hoặc đã tìm thấy file hoặc ta muốn thoát ra để kiểm tra lại.

```
CHARACTER*80  fname
CHARACTER     answer
LOGICAL       exists, OK
OK = .FALSE.
DO WHILE (.NOT.OK)
  WRITE (*, '(1X, A\)') ' Cho ten file : '
  READ  (*, '(A)') fname
  INQUIRE (FILE = fname, EXIST = exists)
  IF (.NOT. exists) THEN
    WRITE(*,'(2A)')' Khong tim thay file ', fname
    WRITE (*,'(A\)')' Nhap lai hay thoat (L/T)? '
    READ (*,'(A)') answer
    IF (answer == 't'.OR.answer == 'T') OK=.TRUE.
  END IF
END DO
END
```

Ví dụ 8.2. Chương trình sau sẽ trả về dung lượng bộ nhớ chiếm giữ của biến mảng X tùy thuộc vào kích thước mảng mà ta nhập vào cho biến N.

```
REAL, ALLOCATABLE :: X(:)
INTEGER N, LEN
WRITE (*,'(A\)') ' Cho kich thuoc mang (N): ‘
READ*, N
ALLOCATE (X(N))
INQUIRE (IOLENGTH = LEN) X
PRINT*,’ Mang X chiem ‘,LEN,’ byte bo nho.’
```

END

8.5 ĐIỀU KHIỂN CON TRỎ FILE

8.5.1 Lệnh REWIND

Chức năng của lệnh này là định vị lại con trỏ file về vị trí đầu file, bất chấp hiện tại nó đang ở vị trí nào. Cú pháp câu lệnh như sau.

REWIND {*unit*|([UNIT=]*unit* &
[,ERR=*err*][,IOSTAT=*iostat*])}

Trong đó nếu bỏ qua **UNIT=** thì ***unit*** phải là tham số đầu tiên. ***unit*** là số hiệu file, nếu file chưa được mở thì **REWIND** không có hiệu lực. ***err*** là nhãn của một câu lệnh thực hiện trong chương trình; nếu chỉ ra, khi xuất hiện lỗi vào/ra, chương trình sẽ chuyển điều khiển đến câu lệnh có nhãn này. ***iostat*** nhận giá trị bằng 0 nếu lệnh thực hiện thành công, ngược lại sẽ trả về số nguyên biểu diễn mã lỗi.

8.5.2 Lệnh BACKSPACE

Chức năng của lệnh là đưa con trỏ file lùi về một bản ghi so với vị trị hiện thời. Cú pháp câu lệnh là:

BACKSPACE {*unit*([UNIT=]*unit*[,ERR=*err*] &
[,IOSTAT=*iostat*])}

Ý nghĩa của các tham số ở đây tương tự như đối với lệnh **REWIND**.

8.5.3 Lệnh ENDFILE

Chức năng của lệnh là ghi vào vị trị hiện thời của con trỏ file bản ghi kết thúc file. Cú pháp câu lệnh là:

ENDFILE {*unit* | ([UNIT=] *unit* [, ERR=*err*] &
[, IOSTAT= *iostat*])}

Ý nghĩa của các tham số ở đây tương tự như đối với lệnh **REWIND** và lệnh **BACKSPACE**.

Ví dụ 8.3. Chương trình sau đây cho thấy tác dụng của các câu lệnh **REWIND**, **BACKSPACE** và **ENDFILE**.

```
CHARACTER (LEN=50) ST
OPEN (1, FILE = “TEST.TXT”)  ! Mở file
WRITE (1,”(A)”) “Dong 1”
WRITE (1,”(A)”) “Dong 2”
WRITE (1,”(A)”) “Dong 3”
WRITE (1,”(A)”) “Dong 4”  ! File có 4 bản ghi tất cả
REWIND (1)     ! Đưa con trỏ file về đầu file
READ (1,”(A)”) ST
PRINT*, ST     ! Kết quả trên màn hình là: Dong 1
READ (1,”(A)”) ST
PRINT*, ST     ! Kết quả trên màn hình là: Dong 2
READ (1,”(A)”) ST
PRINT*, ST     ! Kết quả trên màn hình là: Dong 3
```

```
BACKSPACE (1)  ! Lùi lại bản ghi vừa đọc (Dong 3)
READ (1,"(A)") ST
PRINT*, ST     ! Kết quả trên màn hình là: Dong 3
BACKSPACE (1) !Lùi lại bản ghi vừa đọc (vẫn là Dong 3)
ENDFILE (1)   ! Ghi bản ghi kết thúc file vào vị trí
          ! Dong 3 (File chỉ còn 2 bản ghi đầu)
CLOSE (1)
END
```

8.6 CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG ĐỊNH NGHĨA

Cho đến nay chúng ta mới chỉ giới hạn xét 5 kiểu dữ liệu cơ bản của Fortran. Ta cũng đã biết cách sử dụng các biến đơn và biến mảng với các kiểu dữ liệu này để giải quyết nhiều bài toán khác nhau. Tuy nhiên ở một chừng mực nào đó ta cũng có thể xem mảng như là một loại cấu trúc dữ liệu mà nó là tập hợp các phần tử gồm cùng một kiểu dữ liệu đơn giản.

Để hỗ trợ người dùng tạo ra những kiểu dữ liệu tùy ý, Fortran 90 cho phép ta tự thiết kế các cấu trúc dữ liệu cho riêng mình. Trong mục này ta sẽ xét cách tạo ra cấu trúc dữ liệu kiểu này cũng như cách truy cập, sử dụng chúng trong lập trình.

Trước hết ta hãy xét một ví dụ. Giả sử ta muốn tạo ra những bản ghi chứa thông tin về các sinh viên của một trường nào đó. Trên thực tế, hồ sơ về một sinh viên có thể gồm rất nhiều mục, nhưng ở đây ta chỉ xét một số thông tin chính, như họ và tên, địa chỉ, số điện thoại, mã số sinh viên, giới tính, ngày sinh, điểm của các môn học kể từ năm thứ nhất cho đến lúc ra trường. Khi đó ta có thể định nghĩa một bản ghi lưu trữ thông tin của một sinh viên như sau:

```
TYPE HOSOSV
 CHARACTER (30) HoTen   ! bao gom ca ho va ten.
 CHARACTER (20), DIMENSION(4) :: DiaChi
        ! Tinh, huyen, xa, thon/xom
 CHARACTER (10) DienThoai
 CHARACTER (9) MaSo ! Vi du, K45003504
 LOGICAL GioiTinh   ! .TRUE. neu la Nu,
              ! .FALSE. doi voi Nam (!)
 INTEGER NgaySinh   ! Vi du, 19870308
 REAL, DIMENSION(40) :: Diem ! Diem cac mon hoc
END TYPE
```

Trong đoạn chương trình trên, **TYPE** là một từ khóa, được dùng để định nghĩa cấu trúc **HOSOSV**. Ta sẽ gọi **HOSOSV** được định nghĩa như trên là một kiểu dữ liệu có cấu trúc. Các biến khai báo nằm giữa **TYPE** và **END TYPE** được gọi là các thành phần, hay các trường, của cấu trúc. Để khai báo một biến nào đó có kiểu dữ liệu này ta có thể viết:

```
TYPE (HOSOSV) SVien
```

Với cách khai báo này, ***SVien*** là một biến có kiểu **HOSOSV**. Như vậy, theo cấu trúc trên, thông tin về một sinh viên sẽ được cung cấp qua 7 trường cơ bản, là họ tên, địa chỉ,... Ta cũng có thể nhận thấy các trường của cấu trúc có thể là biến đơn cũng có thể là biến mảng. Chẳng hạn, trường ***Diem*** là một mảng gồm 40 phần tử lưu kết quả học tập của sinh viên trong thời gian ở trường; hoặc trường

DiaChi cũng là một mảng ký tự gồm 4 phần tử, mỗi phần tử là một xâu có độ dài 20 ký tự. Để tham chiếu đến từng trường ta sử dụng dấu phần trăm (%) nối giữa tên biến và tên các trường tương ứng. Ví dụ, các câu lệnh sau đây sẽ gán giá trị cho các trường:

```
SVien%HoTen = “Hoang Anh Dung”
SVien%DiaChi(1) = “Ha Noi”
SVien%DiaChi(2) = “Hoan Kiem”
SVien%DiaChi(3) = “Hang Bac”
SVien%DiaChi(4) = “Dinh Tien Hoang”
SVien%GioiTinh = .FALSE.
...
```

Ta cũng có thể khai báo biến mảng có kiểu **HOSOSV** theo cách tương tự như đối với các kiểu dữ liệu cơ bản khác của Fortran. Ví dụ, để lưu trữ hồ sơ sinh viên của một lớp không quá 100 người ta có thể khai báo:

```
TYPE (HOSOSV), DIMENSION (100) :: K45_KTTV
```

Khi đó **K45_KTTV** sẽ là một biến mảng gồm tối đa 100 phần tử, mỗi phần tử có kiểu **HOSOSV**. Bởi vậy, câu lệnh

```
K45_KTTV%GioiTinh = .FALSE.
```

sẽ gán trường ***GioiTinh*** cho tất cả 100 sinh viên là ***Nam*** (!). Câu lệnh gán này tương đương với câu lệnh:

```
K45_KTTV(:)%GioiTinh = .FALSE.
```

Từ đó, ta có thể đưa ra cú pháp định nghĩa kiểu dữ liệu có cấu trúc như sau:

```
TYPE [[, Quyen_truy_cap] ::] Ten_Cau_Truc
  [ PRIVATE | SEQUENCE]
 Kieu_DL [, Thuoc_tinh [::]] Ten_truong1
 Kieu_DL [, Thuoc_tinh [::]] Ten_truong2
 ...
END TYPE [Ten_Cau_Truc]
```

Trong đó ***Quyen_truy_cap*** ngầm định là **PUBLIC**, trừ khi nó được khai báo **PRIVATE** trong ***modul***. Còn câu lệnh khai báo **PRIVATE** chỉ xuất hiện nếu kiểu cấu trúc dữ liệu được định nghĩa trong các ***modul***. Nếu ***Quyen_truy_cap*** là **PRIVATE** thì kiểu cấu trúc, kể cả tên và các trường của nó, chỉ có thể được truy cập trong chính ***modul*** chủ. Nếu câu lệnh **PRIVATE** xuất hiện trong khai báo kiểu cấu trúc thì tất cả các trường của nó chỉ được phép truy cập trong chính ***modul*** chủ. Nếu câu lệnh **SEQUENCE** được chỉ ra thì tất cả các trường được lưu trữ theo trình tự đã liệt kê.

Giả sử ta định nghĩa một cấu trúc mới:

```
TYPE Student_Type
 CHARACTER (20) NAME
 REAL Mark
END TYPE
```

và khai báo biến **Student** có kiểu dữ liệu **Student_Type:**

```
TYPE (Student_Type) Student
```

Kiểu cấu trúc này rất đơn giản chỉ có hai trường là họ tên (**NAME**) và điểm (**Mark**). Trong trường hợp này, để gán giá trị cho các trường ta có thể sử dụng cách tham chiếu sau:

```
Student = Student_Type( "Hoang Nam", 9.5 )
```

Giá trị các trường trong câu lệnh gán trên cần phải xuất hiện theo thứ tự như trong định nghĩa kiểu. Với cấu trúc **HOSOSV** trong ví dụ trước, việc truy cập và gán giá trị cho các trường sẽ phức tạp hơn. Ví dụ, chương trình sau đây gán nội dung thông tin đầy đủ về một sinh viên theo mẫu cấu trúc dữ liệu **HOSOSV**:

```
TYPE HOSOSV
 CHARACTER (30) HoTen
 CHARACTER (20), DIMENSION(4) :: DiaChi
 CHARACTER (10) DienThoai
 CHARACTER (9) MaSo
 LOGICAL GioiTinh  ! .TRUE. neu la Nu,
                   ! .FALSE. doi voi Nam (!)
 INTEGER NgaySinh  ! Vi du, 19870308
 REAL, DIMENSION(5) :: Diem  ! Diem 5 mon hoc
END TYPE

TYPE (HOSOSV) SVien
SVien = HOSOSV(”Hoang Nam”,(/”Ha Noi”,”Hoan Kiem”,&
          ”Hang Bac”, ”Dinh Tien Hoang”/), &
          “048234567”, ”KT04505432”,      &
          .FALSE., 19780203, (/6,9,8,10,8/))
Print*, SVien
END
```

Giá trị ban đầu của các biến thuộc kiểu dữ liệu có cấu trúc cũng có thể được khởi tạo qua câu lệnh khai báo, ví dụ:

```
TYPE (Student_Type) :: Student = &
            Student_Type( "Hoang Nam", 9.5 )
```

Việc kết xuất thông tin các biến thuộc kiểu dữ liệu có cấu trúc cũng được thực hiện như đối với các biến có kiểu dữ liệu cơ bản. Ví dụ:

```
PRINT '(A20, F6.1)', Student
```

hoặc

```
PRINT *, Student%Name
```

Nếu hai biến có cùng một kiểu dữ liệu có cấu trúc ta có thể thực hiện câu lệnh gán đối với chúng. Khi đó nội dung tất cả các trường của hai biến sẽ được gán tương ứng cho nhau. Ví dụ:

```
TYPE Student_Type
 CHARACTER (20) NAME
 REAL Mark
END TYPE
TYPE (Student_Type) Student1, Student2
...
```

```
Student1 = Student_Type( "Hoang Nam", 9.5 )
Student2 = Student1
...
END
```

Ngoài ra, ta cũng có thể định nghĩa các kiểu dữ liệu có cấu trúc phức tạp hơn mà các trường của kiểu đó sẽ là các kiểu đã định nghĩa trước. Chẳng hạn, đối với kiểu **HOSOSV** trên đây, ta có thể định nghĩa lại như sau:

```
!
TYPE Name          ! Ho ten
 CHARACTER (7)  Ho
 CHARACTER (20) Dem
 CHARACTER (7)  Ten
END TYPE Name

TYPE Address
 CHARACTER (20) Tinh
 CHARACTER (20) Huyen
 CHARACTER (20) Xa
 CHARACTER (20) Xom
END TYPE Address

TYPE DiemHK
 REAL, DIMENSION (10) :: HK1, HK2
END TYPE DiemHK

TYPE KetQuaHT
 TYPE (DiemHK) NamThu1, NamThu2, NamThu3, NamThu4
END TYPE KetQuaHT

TYPE HOSOSV1
 TYPE (Name) HoTen
 TYPE (Address) DiaChi
 CHARACTER (10) DienThoai
 CHARACTER (9) MaSo
 LOGICAL GioiTinh  ! .TRUE. = Nu, .FALSE. = Nam
 INTEGER NgaySinh  ! Vi du, 19870308
 TYPE (KetQuaHT) Diem
END TYPE

TYPE (HOSOSV1) SVien
...
SVien%HoTen = Name (“Tran”, “Luu”, “Danh”)
SVien%Diem%NamThu1%HK1 = (/10,8,8,7,9,6,7,8,9,10/)
...
```

Trở lại với bài toán về việc lưu trữ hồ sơ sinh viên. Hãy tưởng tượng rằng ta có một tập hồ sơ của hàng nghìn sinh viên trong một trường. Mỗi hồ sơ như vậy ta có thể liệt kê các thông tin trong đó thành một dòng của một bảng thống kê. Bảng có thể có nhiều cột, trong mỗi cột lại có các cột nhỏ

hơn, mỗi cột nhỏ hơn lại có thể có các cột nhỏ hơn nữa. v.v. Ví dụ, với cấu trúc trên đây, cột địa chỉ sẽ bao gồm ba cột con. Cột điểm có bốn cột con ứng với bốn năm học, mỗi năm học lại chia thành hai học kỳ, do đó trong cột điểm sẽ có ba "tầng",… Với suy luận như vậy ta có thể thiết kế và xây dựng một cấu trúc dữ liệu phù hợp chứa đầy đủ thông tin về một sinh viên.

BÀI TẬP CHƯƠNG 8

8.1 Viết chương trình nhập hai mảng hai chiều A(N,M) và B(M,P), thực hiện phép nhân hai ma trận. In các ma trận A, B và ma trận tích. Yêu cầu: Nhập các ma trận, nhân ma trận và in ma trận được tổ chức thành các chương trình con và giữa chương trình chính và các chương trình con phải dùng chung bộ nhớ.

8.2 Một tập số liệu lịch sử nhiều năm lưu trữ thông tin về các cơn bão hoạt động trong một khu vực, gồm: Tên cơn bão (tên quốc tế) là một xâu ký tự dài tối đa 20 ký tự, số thứ tự cơn bão trong năm (số nguyên dương), ngày tháng năm hình thành và tan rã, vị trí tâm bão đổ bổ vào đất liền (nếu có) là một cặp hai số thực chỉ kinh độ và vĩ độ tâm bão, áp suất tại tâm vào thời điểm mạnh nhất, tốc độ gió cực đại vào thời điểm mạnh nhất, bán kính ảnh hưởng (là những số thực). Hãy viết chương trình tổ chức bộ số liệu bão đó sao cho thuận tiện khi truy cập, khai thác. ***Gợi ý***: Sử dụng kiểu dữ liệu có cấu trúc TYPE.

8.3 Viết chương trình lưu trữ hồ sơ sinh viên của một lớp, trong đó thông tin về mỗi sinh viên gồm: Họ tên, giới tính, ngày tháng năm sinh, quê quán, chỗ ở hiện nay, số điện thoại, kết quả học tập qua các năm học (từ năm thứ nhất đến năm thứ tư, mỗi năm hai học kỳ) được cho bởi tên môn học, số đơn vị học trình, điểm thi.

CHƯƠNG 9. MỘT SỐ BÀI TOÁN THÔNG DỤNG

Trong chương này sẽ trình bày cách lập chương trình giải một số bài toán thường gặp trong thực tế bằng ngôn ngữ Fortran. Như đã nói trước đây, những chương trình được viết trong chương này chủ yếu nhấn mạnh khía cạnh lập trình, nghĩa là chú trọng vào việc sử dụng ngôn ngữ Fortran, mà chưa đề cập đến sự tối ưu về thuật toán. Sau khi đã thành thạo về ngôn ngữ, bạn đọc có thể thay đổi các chương trình này để tạo thành các thư viện cho riêng mình. Vì sự hạn chế về trình độ và kiến thức nên ở đây chỉ giới thiệu một số bài toán sau:

– Lớp các bài toán thống kê, bao gồm tính các đặc trưng thống kê đơn giản, tính hệ số tương quan tuyến tính (tương quan cặp, tương quan bội, tương quan riêng), xây dựng các phương trình hồi qui,…

– Các bài toán giải tích số, như tìm nghiệm phương trình, tính đạo hàm, tích phân, phương trình truyền nhiệt,…

– Xây dựng cơ sở dữ liệu.

Hơn nữa, để tiện trình bày, các chương trình được viết kèm theo phần giới thiệu thuật toán.

9.1 CÁC BÀI TOÁN THỐNG KÊ CƠ BẢN

9.1.1 Tính trung bình số học của một chuỗi số liệu

Giả sử có chuỗi số liệu $x_1, x_2,..., x_n$. Khi đó trung bình số học của chuỗi được tính bởi công thức:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (9.1.1)$$

Chương trình tính $\bar{x}$ được viết dưới dạng chương trình con hàm **AVER**:

```
FUNCTION AVER(X,N)
!... HAM TINH TRUNG BINH SO HOC CUA CHUOI
! INPUT: + X MANG DO DAI N CHUA SO LIEU QUAN TRAC
!        + N DUNG LUONG MAU
! OUTPUT: TRUNG BINH SO HOC CUA X
!
INTEGER N, I
REAL X(N)    ! Mảng chứa số liệu ban đầu
REAL TMP
 TMP = X(1)
 DO I = 2,N
   TMP = TMP + X(I)
 END DO
 AVER = TMP/REAL(N)
```

```
RETURN
END FUNCTION
```

9.1.2 Tính độ lệch chuẩn của một chuỗi số liệu

Bên cạnh trung bình số học, một đặc trưng khác rất được quan tâm khi xử lý số liệu là độ lệch chuẩn. Giả sử có chuỗi số liệu $x_1, x_2,..., x_n$. Độ lệch chuẩn của chuỗi là căn bậc hai của phương sai mẫu (hay phương sai thực nghiệm) và được tính bởi công thức:

$$s_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2 - \left(\bar{x}\right)^2} \qquad (9.1.2)$$

Theo công thức này, để tính độ lệch chuẩn cần phải tính trung bình số học của chuỗi. Do đó, trong chương trình sau, hàm AVER trên đây sẽ được gọi tới.

```
FUNCTION STDEV (X,N)
!... HAM TINH DO LECH CHUAN CUA CHUOI X
! INPUT: + X MANG DO DAI N CHUA SO LIEU QUAN TRAC
!       + N DUNG LUONG MAU
! OUTPUT: DO LECH CHUAN CUA X
! FUNCTION/SUBROUTINE DUOC GOI TOI: AVER(X,N)
INTEGER N, I
 REAL X(N)     ! Mảng chứa số liệu ban đầu
 REAL TMP, TMP1
 TMP = X(1)*X(1)
 DO I = 2,N
   TMP = TMP + X(I)*X(I)
 END DO
 TMP1 = AVER(X,N)
 STDEV = SQRT(TMP/REAL(N) - TMP1*TMP1)
RETURN
END FUNCTION
```

9.1.3 Sắp xếp chuỗi theo thứ tự tăng dần và xác định giá trị lớn nhất, nhỏ nhất của chuỗi

Giả sử có chuỗi số liệu $x_1, x_2,..., x_n$. Cần phải sắp xếp chuỗi theo một trình tự nhất định và xác định các giá trị lớn nhất, nhỏ nhất của chuỗi. Trong nhiều trường hợp, và nhất là trong các bài toán thống kê, người ta thường sắp xếp chuỗi theo thứ tự tăng dần. Chuỗi được sắp xếp theo thứ tự tăng dần gọi là chuỗi trình tự. Hay nói cách khác, chuỗi trình tự là chuỗi được cấu thành từ tập tất cả các phần tử của chuỗi ban đầu nhưng đã sắp xếp lại theo thứ tự tăng dần về trị số của các phần tử. Khi đó giá trị nhỏ nhất sẽ là phần tử đầu tiên và giá trị lớn nhất sẽ là phần tử cuối cùng của chuỗi trình tự. Ta có chương trình con dạng thủ tục sau đây.

```
SUBROUTINE XMAXMIN (X,N, AMAX, AMIN)
!... CT NAY TIM MAX, MIN CUA CHUOI X
! INPUT: + X MANG DO DAI N CHUA SO LIEU QUAN TRAC
!       + N DUNG LUONG MAU
! OUTPUT:+ AMAX,AMIN lA MAX, MIN CUA X
!      + MANG X(N) DA SAP XEP THEO THU TU TANG DAN
```

```
! FUNCTION/SUBROUTINE DUOC GOI TOI: KHONG
INTEGER N,I,J
REAL, INTENT(INOUT) :: X(N)
  ! Khi vào: Mảng chứa số liệu ban đầu
  ! Khi ra: Mảng đã được sắp xếp
REAL AMAX, AMIN, TMP
 DO I=1,N-1
  K = I
  DO J=I+1,N
    IF (X(J) < X(K)) K = J
  ENDDO
  IF (K /= I) THEN
      TMP = X(K)
      X(K) = X(I)
      X(I) = TMP
  ENDIF
ENDDO
 AMAX = X(N)
 AMIN = X(1)
RETURN
END SUBROUTINE
```

Chú ý rằng trong chương trình này, khi vào mảng **X** là số liệu ban đầu, còn khi trả về chương trình gọi mảng **X** là mảng mà các phần tử của nó đã được sắp xếp theo thứ tự tăng dần.

9.1.4 Xác định các phân vị của chuỗi

Giả sử có chuỗi số liệu { $x_1, x_2,..., x_n$}. Ký hiệu chuỗi trình tự của chuỗi này là { $x_{(1)}, x_{(2)},..., x_{(n)}$} với $x_{(1)} \le x_{(2)} \le ... \le x_{(n)}$. Khi đó, phân vị q_p của chuỗi ứng với xác suất p được xác định bởi: $q_p = x(F(x)=p)$. Khi $p=0.5$, phân vị $q_{0.5}$ được gọi là trung vị và được xác định bởi:

$$Me = q_{0.5} = \begin{cases} x_{((n+1)/2)} & \text{với } n \text{ lẻ} \\ \dfrac{x_{(n/2)} + x_{(n/2+1)}}{2} & \text{với } n \text{ chẵn} \end{cases} \qquad (9.1.3)$$

Nếu $p=0.25$ ta gọi $q_{0.25}$ là phân vị dưới và khi $p=0.75$ thì $q_{0.75}$ được gọi là phân vị trên. Thực chất các phân vị này tương ứng là các trung vị của nửa dưới và nửa trên của chuỗi. Chúng còn được gọi là những *tứ vị*, vì chúng cùng với trung vị sẽ chia chuỗi số liệu tương ứng thành bốn "đoạn". Chương trình con dạng hàm **Q05** sau đây sẽ tính trung vị của chuỗi X độ dài N phần tử, còn chương trình con dạng thủ tục **QUANTILES** sẽ tính cả trung vị và các phân vị dưới, phân vị trên của chuỗi.

```
FUNCTION Q05(X,N)
! HAM NAY TINH TRUNG VI CUA CHUOI X
! INPUT: + X LA MANG SO LIEU BAN DAU
!        + N LA DO DAI CUA X (DUNG LUONG MAU)
! OUTPUT:+ TRUNG VI CUA CHUOI
! FUNCTION/SUBROUTINE DUOC GOI TOI: XMAXMIN
!
```

```
REAL X(N), AMAX, AMIN
INTEGER N,NC
  CALL XMAXMIN (X,N, AMAX, AMIN)
  NC=N/2
  IF (MOD(N,2)==0) THEN
    Q05=(X(NC)+X(NC+1))/2.0
  ELSE
    Q05=X(NC+1)
  ENDIF
RETURN
END FUNCTION
!
SUBROUTINE QUANTILES(X,N,Q50,Q25,Q75)
! CT NAY TINH CAC PHAN VI CHINH CUA CHUOI X(N)
! INPUT: + X MANG CHUA CHUOI SO LIEU BAN DAU
!        + N DUNG LUONG MAU
! OUTPUT: + TRUNG VI Q50
!         + CAC PHAN VI DUOI VA TREN Q25, Q75
! FUNCTION/SUBROUTINE DUOC GOI TOI: Q05(X,N)
!
REAL X(N), X1(N)
INTEGER N,I,NC1,NC2,NT,J
REAL Q50,Q25,Q75
Q50=Q05(X,N)
NT=N
NC1=NT/2
IF (MOD(NT,2)==0) THEN
  NC2=NC1+1
ELSE
  NC1=NC1+1
  NC2=NC1
ENDIF
DO I=1,NC1
  X1(I)=X(I)
ENDDO
Q25=Q05(X1,NC1)
J=0
DO I=NC2,N
  J=J+1
  X1(J)=X(I)
ENDDO
Q75=Q05(X1,NC1)
RETURN
END SUBROUTINE
```

9.1.5 Tính các mômen phân bố

Giả sử chuỗi số liệu $\{ x_1, x_2,..., x_n\}$ là kết quả quan trắc thực nghiệm của biến ngẫu nhiên **X**. Khi đó các mômen phân bố của **X** có thể được ước lượng bởi các công thức:

– Mômen gốc bậc r: $a_r = \frac{1}{n}\sum_{t=1}^{n} x_t^r$ (9.1.4)

– Mômen trung tâm bậc r: $m_r = \frac{1}{n}\sum_{t=1}^{n} (x_t - \bar{x})^r$ (9.1.5)

Trong đó r là một số nguyên dương. Khi $r=1$ ta có $a_1=\bar{x}$ và $m_1=0$. Khi $r=2$ ta có $a_2=\frac{1}{n}\sum_{t=1}^{n} x_t^2$ và $m_2 = \frac{1}{n}\sum_{t=1}^{n} (x_t - \bar{x})^2 = \tilde{D}_x = s_x^2$, trong đó $\tilde{D}_x$ là phương sai thực nghiệm, s_x là độ lệch chuẩn. Giữa mômen gốc và mômen trung tâm liên hệ với nhau qua công thức:

$$m_r = \sum_{k=0}^{r} (-1)^k C_r^k a_1^k a_{r-k} \quad (9.1.6)$$

hay dưới dạng cụ thể hơn:

$$m_r = \sum_{k=0}^{r} \frac{1}{n}\sum_{t=1}^{n} (-1)^k C_r^k x_t^{r-k} \left(\bar{x}\right)^k \quad (9.1.7)$$

Các chương trình sau đây sẽ tính các mômen gốc và mômen trung tâm của chuỗi.

```
FUNCTION AMMGOC(X,N,K)
!
! HAM NAY TINH MOMEN GOC BAC K CUA CHUOI SO LIEU
! INPUT: + X MANG SO LIEU BAN DAU DO DAI N
!        + N DUNG LUONG MAU
!        + K BAC CUA MOMEN
! OUTPUT: MOMEN GOC BAC K
!
REAL X(N)
INTEGER N,K,I
REAL TMP
TMP=0.0
DO I=1,N
  TMP=TMP+X(I)**K
ENDDO
AMMGOC=TMP/REAL(N)
RETURN
END FUNCTION
!
FUNCTION AMMTAM(X,N,K)
! HAM NAY TINH MOMEN TRUNG TAM BAC K CUA CHUOI X
! INPUT: + X MANG SO LIEU BAN DAU DO DAI N
!        + N DUNG LUONG MAU
!        + K BAC CUA MOMEN
! OUTPUT: MOMEN TRUNG TAM BAC K
! FUNCTION/SUBROUTINE DUOC GOI TOI:
```

```
!              AMMGOC(X,N,K), COMBIN(N,K)
REAL X(N)
INTEGER N,K,I
REAL TMP,TB
TB=AMMGOC(X,N,1)
TMP=0
DO I=0,K
  TMP=TMP+(-1)**I*COMBIN(K,I)*TB**I*AMMGOC(X,N,K-I)
ENDDO
AMMTAM=TMP
RETURN
END FUNCTION
```

Chương trình **AMMTAM** cần gọi chương trình tính tổ hợp chập *k* của *n*. Để tính tổ hợp chập ta lại cần phải tính giai thừa. Do đó sau đây dẫn ra hai chương trình con hàm tính tổ hợp chập và tính giai thừa.

```
FUNCTION COMBIN(N,K)
! TINH TO HOP CHAP K CUA N
! INPUT: + N >= K LA NHUNG SO NGUYEN
! OUTPUT: TO HOP CHAP K CUA N
! FUNCTION/SUBROUTINE DUOC GOI TOI: FAC(N)
!
IF (N<0.OR.K<0.OR.N<K) THEN
   WRITE(*,*)' INVALID NUMERIC INPUT...'
   STOP
ELSE
   COMBIN=FAC(N)/FAC(K)/FAC(N-K)
   RETURN
ENDIF
END FUNCTION
!
FUNCTION FAC(N)
! TINH N! (N GIAI THUA)
! INPUT: + N SO NGUYEN KHONG AM
! OUTPUT: + N!
!
REAL TMP
IF (N.LT.0) THEN
  WRITE(*,*)' INVALID NUMERIC INPUT IN FAC FUNC.'
  STOP
ELSE IF (N==0.OR.N==1) THEN
  FAC=1.0
  RETURN
ELSE
  TMP=1.0
  DO I=2,N
    TMP=TMP*REAL(I)
  ENDDO
  FAC=TMP
```

```
  RETURN
ENDIF
END FUNCTION
```

9.1.6 Tính một số đặc trưng thống kê khác

Giả sử có chuỗi số liệu $\{x_1, x_2,..., x_n\}$. Các đặc trưng thống kê cơ bản của chuỗi bao gồm: Trung bình số học, trung vị, trimean, trung bình hiệu chỉnh, phương sai và độ lệch chuẩn, chỉ số biên độ phần tư, phương sai hiệu chỉnh, độ bất đối xứng, chỉ số Yull−Kendall. Trong các mục trước ta đã biết cách lập chương trình tính một số đặc trưng trên. Các đặc trưng còn lại được tính theo các công thức sau.

− Hệ số bất đối xứng: $A = \dfrac{m_3}{s_x^3} = \dfrac{\frac{1}{n}\sum_{t=1}^{n}(x_t - \bar{x})^3}{s_x^3}$ (9.1.8)

− Trimean: $Trimean = \dfrac{q_{0.25} + 2q_{0.5} + q_{0.75}}{4}$ (9.1.9)

− Biên độ phần tư: $IQR = q_{0.75} - q_{0.25}$ (9.1.10)

− Trung bình hiệu chỉnh: $\overline{x_\alpha} = \dfrac{1}{n-2k}\sum_{i=k+1}^{n-k} x_{(i)}$ (9.1.11)

− Phương sai hiệu chỉnh: $s_\alpha^2 = \dfrac{1}{n-2k}\sum_{i=k+1}^{n-k}\left(x_{(i)} - \overline{x_\alpha}\right)^2$ (9.1.12)

trong đó k, là số nguyên làm tròn của tích αn, là số thành phần bị cắt bỏ, tính từ hai đầu mút, của chuỗi trình tự; α là số phần trăm thành phần sẽ bị cắt bỏ ở mỗi đầu mút và được gọi là bậc hiệu chỉnh; $x_{(i)}$ là các thành phần của chuỗi trình tự.

− Chỉ số Yule-Kendall:

$$\gamma_{yk} = \frac{(q_{0.75} - q_{0.5}) - (q_{0.5} - q_{0.25})}{IQR} = \frac{q_{0.25} - 2q_{0.5} + q_{0.75}}{IQR} \quad (9.1.13)$$

Chương trình sau đây cho phép tính tất cả các đặc trưng nói trên của chuỗi $\{x_1, x_2,..., x_n\}$.

```
SUBROUTINE ANOVA(X,N,ALFA,TB,MEDIAN,TRIMEAN,TBHC, &
                 DX,SX,IQR, SHC,A,YULKED)
! CHUONG TRINH NAY TINH CAC DAC TRUNG TK DON GIAN
! INPUT: + X MANG SO LIEU BAN DAU DO DAI N
!        + N DUNG LUONG MAU
!        + ALFA (%) PHAN TRAM SO LIEU BI CAT BO
! OUTPUT: + TB: TRUNG BINH SO HOC
!         + MEDIAN: TRUNG VI
!         + TRIMEAN
!         + TBHC: TRUNG BINH HIEU CHINH
```

```
!       + DX: PHUONG SAI
!       + SX: DO LECH CHUAN
!       + IQR: CHI SO BIEN DO PHAN TU
!       + SHC: PHUONG SAI HIEU CHINH
!       + A: DO BAT DOI XUNG
!       + YULKED: CHI SO YULE-KENDALL
! FUNCTION/SUBROUTINE DUOC GOI TOI: QUANTILES
!
REAL X(N)
REAL ALFA,TB,MEDIAN,TRIMEAN,TBHC
REAL DX,SX,IQR, SHC,A,YULKED
REAL Q50,Q25,Q75,TMP
INTEGER N,I,N1,K
!
TB=AMMGOC(X,N,1)
DX=AMMTAM(X,N,2)
SX=SQRT(DX)
A =AMMTAM(X,N,3)/(DX*SX)
CALL QUANTILES(X,N,Q50,Q25,Q75)
MEDIAN=Q50
TRIMEAN=(Q25+2.0*Q50+Q75)/4.0
IQR=Q75-Q25
YULKED=(Q25-2.0*Q50+Q75)/IQR
!
K=INT(REAL(N)*ALFA)
TMP=0.0
DO I=K+1,N-K
  TMP=TMP+X(I)
ENDDO
TBHC=TMP/REAL(N-2*K)
!
TMP=0.0
DO I=K+1,N-K
  TMP=TMP+(X(I)-TBHC)**2
ENDDO
SHC=TMP/REAL(N-2*K)
RETURN
END SUBROUTINE
```

9.1.7 Tính mômen tương quan và hệ số tương quan

Khi xét đồng thời hai hay nhiều chuỗi số liệu, ngoài các đặc trưng thống kê cơ bản của từng chuỗi, ta cần tính hệ số tương quan hoặc các mômen tương quan giữa các cặp chuỗi. Giả sử có cặp chuỗi số liệu { $(x_1,y_1), (x_2,y_2), ..., (x_n,y_n)$ }. Khi đó mômen tương quan giữa chúng có thể được tính theo công thức:

$$R_{xy} = \frac{1}{n}\sum_{t=1}^{n}\left(x_t - \bar{x}\right)\left(y_t - \bar{y}\right) = \frac{1}{n}\sum_{t=1}^{n} x_t y_t - \bar{x}\bar{y} \qquad (9.1.14)$$

trong đó $\bar{x}$, $\bar{y}$ tương ứng là trung bình số học của các chuỗi $\{x_t\}$ và $\{y_t\}$. Hệ số tương quan giữa hai chuỗi sẽ được xác định bởi:

$$r_{xy} = R_{xy}/(s_x s_y) \qquad (9.1.15)$$

với s_x và s_y tương ứng là độ lệch chuẩn của $\{x_t\}$ và $\{y_t\}$.

Nếu xét đồng thời m chuỗi $\{x_{tj},\ t=1,2,...,n;\ j=1,2,...,m\}$ thì ứng với mỗi cặp $\{x_{tj},\ x_{tk}\}$, $j,k=1,2,...,m$ sẽ có một mômen tương quan. Tất cả các mômen tương quan này lập thành một ma trận m hàng, m cột, gọi là ma trận tương quan:

$$(R_{jk}) = \begin{pmatrix} R_{11} & \dots & R_{1m} \\ \dots & \dots & \dots \\ R_{m1} & \dots & R_{mm} \end{pmatrix} \qquad (9.1.16)$$

trong đó $R_{jk} = R_{x_j x_k}$ là mômen tương quan giữa $\{x_{tj}\}$ và $\{x_{tk}\}$, được tính theo công thức tương tự trên đây. Hệ số tương quan giữa $\{x_{tj}\}$ và $\{x_{tk}\}$ cũng sẽ được xác định bởi $r_{jk}=R_{jk}/s_j s_k$, với $s_j=s_{xj}$, $s_k=s_{xk}$ tương ứng là độ lệch chuẩn của $\{x_{tj}\}$ và $\{x_{tk}\}$. Các hệ số tương quan này sẽ lập thành ma trận tương quan chuẩn hóa:

$$(r_{jk}) = \begin{pmatrix} r_{11} & \dots & r_{1m} \\ \dots & \dots & \dots \\ r_{m1} & \dots & r_{mm} \end{pmatrix} \qquad (9.1.17)$$

Các chương trình sau đây cho phép tính các mômen tương quan và ma trận tương quan giữa các chuỗi số liệu.

```
FUNCTION HSTQ(X,Y,N)
!HAM NAY TINH HE SO TUONG QUAN GIUA HAI BIEN X VA Y
! INPUT: + X(N) MANG CHUA SO LIEU CUA X
!        + Y(N) MANG CHUA SO LIEU CUA Y
!        + N DUNG LUONG MAU
! OUTPUT: R: HE SO TUONG QUAN GIUA X VA Y
! CAC HAM DUOC GOI TOI: AVER, STDEV
REAL X(N),Y(N)
REAL TBX,TBY,SX,SY,RXY
TBX=AVER (X,N)
TBY=AVER (Y,N)
SX=STDEV (X,N)
SY=STDEV (Y,N)
RXY=0.0
DO I=1,N
  RXY=RXY+X(I)*Y(I)
ENDDO
RXY=RXY/REAL(N)-TBX*TBY
HSTQ=RXY/SX/SY
RETURN
```

```
END FUNCTION
!
SUBROUTINE COVAR(X,N,M,R,TB)
! C/TRINH NAY TINH MA TRAN TUONG QUAN CUA M CHUOI
! INPUT: + X(N*M) MANG 1 CHIEU CHUA S/LIEU BAN DAU
!        LUU THEO COT CUA MA TRAN XX(N,M)
!        (X(1,1), X(2,1),...,X(N,1),X(1,2),...)
!      + N DUNG LUONG MAU
!      + M SO BIEN
! OUTPUT:+ R(M*M) MANG MOT CHIEU
!              CHUA MA TRAN TUONG QUAN
!      + TB(M) MANG 1 CHIEU (TRUNG BINH CAC BIEN)
!
REAL X(N*M),XX(N,M),R(M*M),RR(M,M),TB(M)

K=0
DO J=1,M
  TB(J)=AVER(X((J−1)*N+1,J*N),N)
  DO I=1,N
    K=K+1
    XX(I,J)=X(K)
  ENDDO
ENDDO
DO J=1,M
  DO K=J,M
   RR(J,K)=0.0
   DO I=1,N
    RR(J,K)=RR(J,K)+(XX(I,J)-TB(J))*(XX(I,K)-TB(K))
   ENDDO
   RR(J,K)=RR(J,K)/REAL(N)
   IF (K /= J) RR(K,J)=RR(J,K)
  ENDDO
ENDDO
L=0
DO K=1,M
  DO J=1,M
    L=L+1
    R(L)=RR(J,K)
  ENDDO
ENDDO
RETURN
END SUBROUTINE
!
SUBROUTINE CORRE(X,N,M,R)
!CHUONG TRINH NAY TINH MA TRAN TUONG QUAN CHUAN HOA
! CUA TAP M BIEN TU SO LIEU BAN DAU CO DUNG LUONG N
!INPUT:+ X MANG MOT CHIEU KICH THUOC N*M LUU TRU SO
!       LIEU BAN DAU DANG MA TRAN N HANG M COT
!       (X(1,1), X(2,1),...,X(N,1),X(1,2),...)
!      + N DUNG LUONG MAU
```

```
!       + M SO BIEN
! OUTPUT: + R(M*M) MANG MOT CHIEU LUU TRU MA TRAN
!          TUONG QUAN CHUAN HOA CUA M BIEN
!
REAL X(N*M),XX(N,M),R(M*M),TB(M),SX(M)
K=0
DO J=1,M
  TB(J)=AVER(X((J−1)*N+1,J*N),N)
  SX(J)=STDEV(X((J−1)*N+1,J*N),N)
  DO I=1,N
    K=K+1
    XX(I,J)=X(K)
  ENDDO
ENDDO
JK=0
DO J=1,M
  DO K=1,M
    JK=JK+1
    R(JK)=0.0
    DO I=1,N
      R(JK)=R(JK)+XX(I,J)*XX(I,K)
    ENDDO
    R(JK)=R(JK)/REAL(N)-TB(J)*TB(K)
    R(JK)=R(JK)/(SX(J)*SX(K))
  ENDDO
ENDDO
RETURN
END SUBROUTINE
```

Chú ý rằng trong hai chương trình **COVAR** và **CORRE** trên đây, mặc dù số liệu ban đầu được hiểu là lưu trên các ma trận **N** hàng, **M** cột, nhưng do các phần tử mảng trong bộ nhớ được Fortran lưu dưới dạng vectơ, nên trước khi gọi các chương trình này ta cần chuyển chúng về dạng mảng một chiều. Nếu không thực hiện việc chuyển đổi này, kết quả tính toán có thể sẽ sai do kích thước thực của mảng vào từ chương trình gọi và của mảng khai báo trong chương trình con khác nhau. Câu lệnh chuyển được thực hiện ở chương trình gọi có thể là:

```
DO J=1,M
  X((J-1)*N+1:J*10) = XX(1:N,J)
ENDDO
```

Trong đó **XX** là mảng hai chiều lưu số liệu của **M** biến, với dung lượng mẫu là **N**, còn **X** là mảng một chiều kích thước **NxM** sẽ được truyền vào chương trình con.

Ta có thể khắc phục nhược điểm này bằng cách sử dụng mảng động. Ý tưởng là ở chỗ, vì mảng động chỉ xác định kích thước và cách sắp xếp phần tử khi ta cấp phát bộ nhớ cho nó bằng câu lệnh **ALLOCATE**, nên nếu trong chương trình gọi ta dùng mảng động thì kích thước và cách sắp xếp các phần tử của chúng trong cả chương trình gọi và chương trình con sẽ giống nhau. Ví dụ, trong chương trình gọi ta khai báo:

```
REAL, ALLOCATABLE :: A(:,:),B(:,:),C(:)
...
ALLOCATE (A(N,M),B(M,M),C(M))
...
CALL COVAR_1 (A, N, M, B, C)
...
CALL CORRE_1 (A, N, M, B)
...
```

Khi đó, các chương trình **COVAR** và **CORRE** tương ứng được đổi thành **COVAR_1** và **CORRE_1** sau đây:

```
SUBROUTINE COVAR_1(X,N,M,R,TB)
!
!INPUT:+ X(N,M) MANG HAI CHIEU CHUA SO LIEU BAN DAU
!       + N DUNG LUONG MAU
!       + M SO BIEN
! OUTPUT:+ R(M,M) MANG HAI CHIEU
!              CHUA MA TRAN TUONG QUAN
!     + TB(M) MANG MOT CHIEU (TRUNG BINH CAC BIEN)
!
REAL X(N,M), R(M,M),TB(M)
DO J=1,M
  TB(J)=AVER(X(:,J),N)
ENDDO
DO J=1,M
  DO K=J,M
    R(J,K)=0.0
    DO I=1,N
      R(J,K)=R(J,K)+X(I,J)*(X(I,K)
    ENDDO
    R(J,K)=R(J,K)/REAL(N)─TB(J)*TB(K)
    R(K,J)=R(J,K)
  ENDDO
ENDDO
RETURN
END SUBROUTINE
!
SUBROUTINE CORRE_1(X,N,M,R)
!
! INPUT: + X(N,M) MANG HAI CHIEU LUU TRU SO LIEU
!       + N DUNG LUONG MAU
!       + M SO BIEN
! OUTPUT: + R(M,M) MANG HAI CHIEU LUU TRU MA TRAN
!          TUONG QUAN CHUAN HOA CUA M BIEN
!
REAL X(N,M), R(M,M),TB(M),SX(M)
DO J=1,M
  TB(J)=AVER (X(:,J),N)
```

```
  SX(J)=STDEV(X(:,J),N)
ENDDO
DO J=1,M
  DO K=J,M
    R(J,K)=HSTQ (X(:,J),X(:,K),N)
    R(K,J)=R(J,K)
  ENDDO
ENDDO
RETURN
END SUBROUTINE
```

9.2 MỘT SỐ BÀI TOÁN VỀ MA TRẬN

9.2.1. Tích hai ma trận

Cho ma trận A(N,M) gồm N hàng, M cột và ma trận B(M,P) gồm M hàng, P cột. Ma trận tích của hai ma trận này sẽ có kích thước N hàng, P cột mà các phần tử của nó được xác định bởi hệ thức sau:

$$c_{ik} = \sum_{j=1}^{M} a_{ij} b_{jk} \qquad (9.2.1)$$

trong đó c_{ik} là phần tử hàng i cột k của ma trận C(N,P), a_{ij} và b_{jk} tương ứng là các phần tử hàng i cột j và hàng j cột k của các ma trận A và B.

Với công thức đó ta có chương trình tính tích hai ma trận như sau.

```
SUBROUTINE MULTIP(A,B,C,N,M,P)
! CHUONG TRINH NAY TINH TICH CUA HAI MA TRAN
! INPUT: + A(N*M) MANG MOT CHIEU (N HANG, M COT)
!          LUU MA TRAN A THEO COT
!        + B(M*P) MANG MOT CHIEU (M HANG, P COT)
!          LUU MA TRAN B THEO COT
!        + N SO HANG CUA MA TRAN A
!        + M SO COT CUA A VA LA SO HANG CUA B
!        + P SO COT CUA B
! OUTPUT: + C(N*P) MANG MOT CHIEU (N HANG, P COT)
!           LUU MA TRAN C THEO COT
INTEGER N,M,P
REAL A(N*M),B(M*P),C(N*P)
DO I=1,N
DO K=1,P
  IK=(K-1)*N+I
  C(IK)=0.0
  DO J=1,M
    IJ=(J-1)*N+I
    JK=(K-1)*M+J
    C(IK)=C(IK)+A(IJ)*B(JK)
  ENDDO
```

```
ENDDO
ENDDO
RETURN
END SUBROUTINE
```

Nếu sử dụng cách biểu diễn mảng hai chiều, chương trình **MULTIP** có thể được viết lại thành:

```
SUBROUTINE MULTIP_1(A,B,C,N,M,P)
! INPUT: + A(N,M) MANG MOT CHIEU (N HANG, M COT)
!        + B(M,P) MANG MOT CHIEU (M HANG, P COT)
!        + N SO HANG CUA MA TRAN A
!        + M SO COT CUA A VA LA SO HANG CUA B
!        + P SO COT CUA B
! OUTPUT: + C(N,P) MANG MOT CHIEU (N HANG, P COT)
INTEGER N,M,P
REAL A(N,M),B(M,P),C(N,P)
DO I=1,N
  DO K=1,P
    C(I,K)=0.0
    DO J=1,M
      C(I,K)=C(I,K)+A(I,J)*B(J,K)
    ENDDO
  ENDDO
ENDDO
RETURN
END SUBROUTINE
```

Nếu ma trận B chỉ gồm một cột, B(M,1), tức B là vectơ cột, khi đó chương trình trên cho phép nhân ma trận với một vectơ.

9.2.2. Định thức của ma trận

Nếu A là một ma trận vuông gồm N hàng và N cột thì định thức của ma trận A là số được xác định bởi:

$$D = \det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum \pm a_{1r_1} a_{2r_2} \dots a_{nr_n} \qquad (9.2.2)$$

trong đó các chỉ số $r_1, r_2, \dots, r_n$ chạy qua tất cả $n!$ hoán vị có thể có của các số $1,2,\dots,n$, còn dấu của mỗi số hạng là (+) hay (–) tuỳ theo hoán vị tương ứng là chẵn hay lẻ. Số n được gọi là cấp của định thức.

Có nhiều thuật toán để tính định thức của A. Bạn đọc có thể tham khảo chẳng hạn trong [...]. Sau đây là một phương án tính.

Ký hiệu $D=D_n$ ta có:

$$D_n = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = a_{11} \begin{vmatrix} 1 & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

với $a_{1j}^{(1)} = a_{1j} / a_{11}, j = 1,2,...,n$

Sử dụng phép biến đổi Gauss ta nhận được:

$$D_n = a_{11} \begin{vmatrix} 1 & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = a_{11} \begin{vmatrix} 1 & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{vmatrix} =$$

$$= a_{11} \begin{vmatrix} a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \dots & \dots & \dots \\ a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{vmatrix} = a_{11} D_{n-1}$$

Quá trình cứ tiếp tục như vậy cho đến khi ta nhận được hệ thức cuối cùng: $D = a_{11} a_{22}^{(1)} ... a_{nn}^{(n-1)}$.

Dựa trên thuật toán này ta có chương trình tính dưới đây.

```
FUNCTION DET(X,N)
! HAM NAY TINH DINH THUC CUA MA TRAN A(N,N)
!INPUT:+ X(N*N) MANG CHUA CAC PHAN TU CUA MA TRAN A
!     LUU DANG COT: X(1)=A(1,1), X(2)=A(2,1),...,
!     X(N)=A(N,1),X(N+1)=A(1,2), X(N+2)=A(2,2),...
!    + N KICH THUOC MA TRAN A
! OUTPUT: DINH THUC CUA A
!
REAL, PARAMETER :: EP=1.0E-6
REAL X(N*N),A(N,N)
K=0
DO J=1,N
  DO I=1,N
    K=K+1
    A(I,J)=X(K)
  ENDDO
ENDDO
D=1.0
N1=N-1
DO K=1,N1
  AM=0.0
  DO I=K,N
    T=A(I,K)
    IF (ABS(T)>=ABS(AM)) THEN
```

```
      AM=T
      J=I
    ENDIF
  ENDDO
  IF (ABS(AM)<=EP) THEN
    DT=0.0
    DET=DT
    RETURN
  ELSE
    IF (J/=K) THEN
      D=-D
      DO I=K,N
        T=A(J,I)
        A(J,I)=A(K,I)
        A(K,I)=T
      ENDDO
    ENDIF
  ENDIF
  M=K+1
  DO I=M,N
    T=A(I,K)/AM
    DO J=M,N
      A(I,J)=A(I,J)-T*A(K,J)
    ENDDO
  ENDDO
  D=D*A(K,K)
ENDDO
DT=D*A(N,N)
DET=DT
RETURN
END FUNCTION
```

Trong chương trình trên ma trận đầu vào được lưu dưới dạng mảng một chiều. Nếu ma trận đầu vào được lưu dưới dạng mảng hai chiều ta chỉ cần thay đổi phần đầu của chương trình này để nhận được một chương trình khác tương đương:

```
FUNCTION DET_1(A,N)
! HAM NAY TINH DINH THUC CUA MA TRAN A(N,N)
!INPUT:+ A(N,N) MANG CHUA CAC PHAN TU CUA MA TRAN A
!       + N KICH THUOC MA TRAN A
! OUTPUT: DINH THUC CUA A
!
REAL, PARAMETER :: EP=1.0E-6
REAL A(N,N)
D=1.0
N1=N-1
...
RETURN
END FUNCTION
```

9.2.3. Phần phụ đại số

Phần phụ đại số D_{ij} của phần tử a_{ij} của ma trận vuông A gồm N hàng, N cột, là định thức của ma trận con của A sau khi đã loại bỏ hàng i cột j nhân với $(-1)^{i+j}$. Chương trình sau đây cho phép tính phần phụ đại số này.

```
FUNCTION PPDS(A,N,IR,JC)

! HAM NAY TINH PHAN PHU DAI SO CUA PHAN TU B(IR,JC)
! (HANG IR, COT JC) CUA MA TRAN B(N,N)
! MA CAC PHAN TU CUA NO DUOC LUU TRONG MANG A(N*N)
! THEO QUI CACH COT
! TRUOC DONG SAU
! INPUT: + MANG A(N*N) CHUA MA TRAN DAU VAO CUA B
!        + N KICH THUOC CUA B
!        + IR CHI SO HANG
!        + JC CHI SO COT
! OUTPUT: PHAN PHU D/SO CUA PHAN TU HANG IR COT JC
!
REAL A(N*N), B(N,N)
K=0
DO J=1,N
  DO I=1,N
    K=K+1
    B(I,J)=A(K)
  ENDDO
ENDDO
K=0
DO J=1,N
  IF (J/=JC) THEN
    DO I=1,N
      IF (I/=IR) THEN
        K=K+1
        A(K)=B(I,J)
      ENDIF
    ENDDO
  ENDIF
ENDDO
IF (MOD(IR+JC,2)==0) THEN
   PPDS=DET(A,N-1)
ELSE
   PPDS=-DET(A,N-1)
ENDIF
RETURN
END FUNCTION
```

9.2.4. Ma trận nghịch đảo

Nếu A là một ma trận vuông không suy biến cấp **NxN** (định thức khác 0) thì ma trận nghịch đảo của $\boldsymbol{A}$, ký hiệu là $\boldsymbol{A^{-1}}$, sẽ được xác định bởi hệ thức: $\boldsymbol{A.A^{-1} = I}$, trong đó $\boldsymbol{I}$ là ma trận đơn vị.

Ký hiệu phần tử hàng *i* cột *j* của ma trận $\boldsymbol{A^{-1}}$ là $a_{ij}^{(-1)}$ ta có công thức tính như sau:

$$a_{ij}^{(-1)} = \frac{D_{ij}}{D},\ i,j=1,2,...,\ n$$

trong đó $D = detA$ là định thức của ma trận $\boldsymbol{A}$, D_{ij} là phần phụ đại số của phần tử a_{ij} của ma trận $\boldsymbol{A}$.

Ta cũng có thể tính ma trận nghịch đảo của $\boldsymbol{A}$ bằng phương pháp khử Gauss như sau. Giả sử

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Ta lập một ma trận mới $\boldsymbol{B}$ bằng cách ghép một ma trận đơn vị có cùng kích thước vào bên phải $\boldsymbol{A}$:

$$B = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{pmatrix}$$

Sau đó thực hiện các phép biến đổi thông thường để đưa ma trận $\boldsymbol{B}$ về dạng:

$$B' = \begin{pmatrix} 1 & 0 & \dots & 0 & b_{11} & b_{12} & \dots & b_{1n} \\ 0 & 1 & \dots & 0 & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Khi đó, nửa bên phải của $\boldsymbol{B'}$ chính là nghịch đảo của $\boldsymbol{A}$:

$$A^{-1} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Sau đây là hai chương trình tính ma trận nghịch đảo.

```
SUBROUTINE MINVERT(AA,N)
! C/TRINH TINH MA TRAN NGHICH DAO CUA MA TRAN A
! INPUT: + AA(N*N) MANG MOT CHIEU LUU A THEO COT
```

```
!       + N SO HANG/COT CUA MA TRAN A
! OUTPUT: AA MA TRAN NGHICH DAO CUA A
!
REAL AA(N*N), A(N,N)
K=0
DO J=1,N
  DO I=1,N
    K=K+1
    A(I,J)=AA(K)
  ENDDO
ENDDO
M=N-1
DO I=1,N
  R=A(I,1)
  DO K=1,M
    A(I,K)=A(I,K+1)/R
  ENDDO
  A(I,N)=1.0/R
  DO J=1,N
    IF (I/=J) THEN
      R=A(J,1)
      DO K=1,M
        A(J,K)=A(J,K+1)-R*A(I,K)
      ENDDO
      A(J,N)=-R*A(I,N)
    ENDIF
  ENDDO
ENDDO
K=0
DO J=1,N
  DO I=1,N
    K=K+1
    AA(K)=A(I,J)
  ENDDO
ENDDO
RETURN
END SUBROUTINE
!
SUBROUTINE NDMT_GAUS (A,N)
! INPUT: + A(N,N) MANG 2 CHIEU CHUA A
!       + N: KICH THUOC CUA A
! OUTPUT: A(N,N) MA TRAN NGHICH DAO CUA A
REAL A(N,N), B(N,2*N), X(2*N)
integer i,j,k
REAL Tmp1, Tmp2
B = 0.                  ! Khởi tạo B
B( 1:N, 1:N ) = A(1:N,1:N) ! Nửa đầu của B
DO I = 1, N             ! Đường chéo nửa sau của B
  B( I, N+I ) = 1.
END DO
```

```
DO I = 1, N
  Tmp1 = B( I, I )
  IF (Tmp1 == 0) THEN ! Đổi chỗ các hàng
    K = I + 1
    DO WHILE (Tmp1 == 0 .AND. K <= N)
      Tmp1 = B( K, I )
      K = K + 1
    END DO
    IF (Tmp1 == 0) THEN
      PRINT*, "EXIT Here"
      STOP
    ELSE
      X = B( I, 1:2*N )
      K = K - 1
      B( I, 1:2*N ) = B( K, 1:2*N )
      B( K, 1:2*N ) = X
    END IF
  END IF
! Chuẩn hóa hàng I cho các phần tử trên đường chéo
  B( I, 1:2*N ) = B( I, 1:2*N ) / Tmp1
! Khử ngoài đường chéo nửa đầu của B
  DO J = 1, N
    IF (J /= I) THEN
      Tmp2 = B( J, I )
      B( J, 1:2*N ) = B( J, 1:2*N ) &
          - B( I, 1:2*N ) * Tmp2
    END IF
  END DO
END DO
! Nghịch đảo của A(1:N, 1:N)
A = B( 1:N, N+1:2*N )
END SUBROUTINE
```

9.2.5. Giải hệ phương trình đại số tuyến tính

Trong mục này ta sẽ tìm hiểu một số phương pháp lập trình giải hệ phương trình đại số tuyến tính. Giả sử có hệ phương trình:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad (9.2.3)$$

trong đó a_{ij}, b_i, $(i,j=1,2,\ldots,n)$ là các hệ số hằng số; $x_1, x_2,\ldots,x_n$ là các ẩn số phải tìm. Hệ phương trình trên có thể được biểu diễn dưới dạng ma trận:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} \tag{9.2.4}$$

Hay ở dạng gọn hơn:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{9.2.5}$$

trong đó:

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} \tag{9.2.6}$$

Nếu $\boldsymbol{A}$ là ma trận không suy biến, nghiệm của hệ có thể được xác định bởi các phương pháp sau:

a. Phương pháp nghịch đảo ma trận

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{9.2.7}$$

với $\boldsymbol{A}^{-1}$ là ma trận nghịch đảo của $\boldsymbol{A}$.

b. Phương pháp Cramer

$$x_i = \frac{D_i}{D},\ i=1,2,...,n \tag{9.2.8}$$

trong đó:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} \tag{9.2.9}$$

là định thức của ma trận $\boldsymbol{A}$,

$$D_i = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1,i-1}\, b_1 & a_{1,i+1} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2,i-1}\, b_2 & a_{2,i+1} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{n,i-1}\, b_n & a_{n,i+1} & \dots & a_{nn} \end{vmatrix} \tag{9.2.10}$$

là định thức của ma trận $\boldsymbol{A}$ sau khi đã thay cột thứ $\boldsymbol{i}$ bởi vectơ vế phải $\boldsymbol{b}$.

c. Phương pháp khử Gauss

Nguyên tắc chung của phương pháp này là sử dụng các phép biến đổi để đưa hệ phương trình (9.2.3) về dạng "nửa đường chéo" như sau:

$$\begin{cases} a'_{11}x_1 + a'_{12}x_2 + \ldots + a'_{1n}x_n = b'_1 \\ \qquad a'_{22}x_2 + \ldots + a'_{2n}x_n = b'_2 \\ \ldots \\ \qquad\qquad\qquad a'_{nn}x_n = b'_n \end{cases}$$

Khi đó nghiệm của hệ sẽ được xác định bởi các hệ thức:

$$x_n = \frac{b'_n}{a'_{nn}},\ x_{n-1} = \frac{b'_{n-1} - a'_{n-1}x_n}{a'_{n-1}} = \frac{b'_{n-1} - a'_{n-1}\dfrac{b'_n}{a'_{nn}}}{a'_{n-1}},\ldots,\ x_n = \frac{b'_1 - \sum_{j=2}^{n} a'_{1j}x_j}{a'_{11}}$$

Nếu hệ (9.2.3) được đưa về dạng "đường chéo" ta sẽ có:

$$\begin{cases} 1x_1 + \ 0x_2 + \ldots + \ 0x_n = b''_1 \\ 0x_1 + 1x_2 + \ldots + \ 0x_n = b''_2 \\ \ldots \\ 0x_1 + \ 0x_2 + \ldots + 1x_n = b''_n \end{cases}$$

Trong trường hợp này, nghiệm của hệ đơn giản là:

$$x_1 = b''_1,\ x_2 = b''_2,\ldots,\ x_n = b''_n$$

Sau đây là ba chương trình giải hệ phương trình đại số tuyến tính (9.2.3) tương ứng với ba phương pháp trình bày ở trên.

```
SUBROUTINE GIAIHE_ND(A,B,X,N)
! CHUONG TRINH NAY GIAI HE PHUONG TRINH DSTT Ax=b
! BANG PP NGHICH DAO MA TRAN
!
!INPUT:+ A(N*N) MANG MOT CHIEU CHUA MA TRAN HE SO A
!          (LUU THEO COT)
!       + B(N) VECTO HE SO b
!       + N SO PHUONG TRINH
! OUTPUT: + X(N) VECTO NGHIEM
!
REAL A(N*N),B(N), X(N)
CALL MINVERT(A,N)
CALL MULTIP(A,B,X,N,N,1)
RETURN
END SUBROUTINE
!
SUBROUTINE GIAIHE_CRM(A,B,X,N)
! CHUONG TRINH NAY GIAI HE PHUONG TRINH DSTT Ax=b
! BANG PP CRAMER
!
!INPUT:+ A(N*N) MANG MOT CHIEU CHUA MA TRAN HE SO A
!          (LUU THEO COT)
!       + B(N) VECTO HE SO b
!       + N SO PHUONG TRINH
```

```
! OUTPUT: + X(N) VECTO NGHIEM
!
REAL A(N*N),B(N),X(N), AA(N*N)
REAL D, DI
D = DET(A,N)
DO I = 1,N
  AA(:)=A(:)
  AA((I-1)*N+1:I*N) = B(:)
  DI = DET(AA,N)
  X(I) = DI/D
ENDDO
RETURN
END SUBROUTINE
!
SUBROUTINE GIAIHE_GAUSE (AA,B,X,N)
integer N,i,j,k
REAL AA(N*N), B(N), X(N), A(N, N+1)

DO I=1,N
  A(:,I) = AA((I-1)*N+1:I*N)
ENDDO
A(:,N+1) = B(:)
DO I=1,N
  Tmp = A( I, I ) ! Sắp xếp lại để A (I,I) <> 0
  IF (Tmp == 0) THEN
    K = I + 1
    DO WHILE (Tmp == 0 .AND. K < N)
      Tmp = A( K, I )
      K = K + 1
    END DO
    IF (Tmp == 0) THEN
      PRINT*, "EXIT Here"
      STOP
    ELSE
      X = A( I, 1:N+1 )
      K = K - 1
      A( I, 1:N+1 ) = A( K, 1:N+1 )
      A( K, 1:N+1 ) = X
    END IF
  END IF
  Tmp = A(i,i)   ! Chuẩn hóa hàng I
  A(i,1:N+1) = A(i,1:N+1) / Tmp
  do i1=1,N      ! Khử các phần tử ngoài đường chéo
    if (i1 /= i) then
      Tmp = A(i1,i)
      A(i1,1:N+1)=A(i1,1:N+1)-A(i,1:N+1)*Tmp
    endif
  enddo
Enddo
X(1:N) = A(1:N,N+1)  ! Nghiệm
```

END SUBROUTINE

9.3 TƯƠNG QUAN VÀ HỒI QUI TUYẾN TÍNH

9.3.1. Xây dựng phương trình hồi qui tuyến tính

Đây là một trong những bài toán khá phổ biến được ứng dụng nhiều trong thực tế. Nội dung bài toán có thể được phát biểu như sau. Giả sử $\{x_{t1},x_{t2},...,x_{tm};\ t=1,2,...,n\}$ là tập số liệu quan trắc thực nghiệm của m biến ngẫu nhiên $X_1, X_2,..., X_m$. Xét mối quan hệ tương quan giữa biến X_1 (biến phụ thuộc) với tập $m-1$ biến còn lại (các biến độc lập). Hãy xây dựng phương trình hồi qui tuyến tính giữa biến phụ thuộc X_1 với tập các biến độc lập $X_2, X_3,..., X_m$.

Với yêu cầu của bài toán, ta cần xác định các hệ số $a_1, a_2,..., a_m$ của phương trình hồi qui:

$$\hat{x}_1 = a_1 + a_2x_2 + ... + a_mx_m \qquad (9.3.1)$$

Về lý thuyết xây dựng phương trình (9.3.1) bạn đọc có thể tham khảo, chẳng hạn, trong [4]. Sau đây sẽ dẫn ra các công thức tính các hệ số a_j, $j=1,2,...,m$ và một số đại lượng liên quan đến việc phân tích phương sai và đánh giá chất lượng phương trình hồi qui.

Hệ số a_1 của (9.3.1) được tính theo công thức:

$$a_1 = \overline{x_1} - \sum_{j=2}^{m} a_j \overline{x_j} \qquad (9.3.2)$$

Trong đó $\overline{x_j} = \frac{1}{n}\sum_{t=1}^{n} x_{tj}$ là trung bình số học của biến X_j, $j=1,2,...$

Các hệ số $a_2, a_3,..., a_m$ được xác định bằng việc giải hệ phương trình đại số tuyến tính $m-1$ phương trình, $m-1$ ẩn số sau:

$$\sum_{k=2}^{m} R_{jk}a_k = R_{1j}\ ,j=2,3,...,\ m \qquad (9.3.3)$$

Hay viết dưới dạng ma trận:

$$\begin{pmatrix} R_{22} & R_{23} & ... & R_{2m} \\ R_{32} & R_{33} & ... & R_{3m} \\ ... & ... & ... & ... \\ R_{m2} & R_{m3} & ... & R_{mm} \end{pmatrix} \begin{pmatrix} a_2 \\ a_3 \\ ... \\ a_m \end{pmatrix} = \begin{pmatrix} R_{12} \\ R_{13} \\ ... \\ R_{1m} \end{pmatrix} \qquad (9.3.3')$$

Trong đó R_{jk}, $j,k=2,3,...,m$ là môмen tương quan giữa các biến X_j và X_k, được tính theo công thức:

$$R_{jk} = \frac{1}{n}\sum_{t=1}^{n}\left(x_{tj} - \overline{x_j}\right)\left(x_{tk} - \overline{x_k}\right) \qquad (9.3.4)$$

và R_{1j}, $j=1,2,...,m$ là môмen tương quan giữa X_1 và X_j, được xác định bởi:

$$R_{1j} = \frac{1}{n}\sum_{t=1}^{n}\left(x_{t1} - \overline{x_1}\right)\left(x_{tj} - \overline{x_j}\right) \qquad (9.3.5)$$

Tổng bình phương các biến sai hồi qui (U) và biến sai thặng dư (Q) và sai số chuẩn (s) của ước lượng theo phương trình hồi qui (9.3.1) được tính theo các công thức:

$$U = \sum_{t=1}^{n}\left(\hat{x}_{t1} - \overline{x_1}\right)^2 \qquad (9.3.6)$$

$$Q = \sum_{t=1}^{n}\left(x_{t1} - \hat{x}_{t1}\right)^2 \qquad (9.3.7)$$

$$s = \sqrt{\frac{Q}{n-m}} \qquad (9.3.8)$$

với $\hat{x}_{t1} = a_1 + a_2 x_{t2} + ... + a_m x_{tm}$, $t=1,2,...,\ n$ là ước lượng của các quan trắc x_{t1} theo phương trình hồi qui (9.3.1).

Trên cơ sở đó ta có chương trình tính sau đây.

```
SUBROUTINE REGRE(X,N,M,A,S,Q,U,RR,F)
! CHUONG TRINH TINH CAC HE SO HOI QUI CUA P/TRINH
! HOI QUI GIUA X1 VA CAC X2,...,XM
! VA XAC DINH CAC DAI LUONG CHO PHAN TICH P/SAI.
! DANG PHUONG TRINH HQ: X1=A1+A2*X2+...+AM*XM
!
! INPUT: + X(N*M) MANG MOT CHIEU LUU SO LIEU
!          QUAN TRAC CUA CAC X1..XM (LUU THEO COT)
!        + N DUNG LUONG MAU
!      + M SO BIEN (1 BIEN P.THUOC VA M-1 BIEN DLAP)
!      + A MANG MOT CHIEU KICH THUOC M LUU CAC HE SO
!          A1, A2,..., AM
!        + S SAI SO CHUAN (CHUAN SAI THANG DU)
!        + Q TONG BINH PHUONG CAC BIEN SAI THANG DU
!        + U TONG BINH PHUONG CAC BIEN SAI HOI QUI
!        + RR HE SO TUONG QUAN BOI
!        + F BIEN CO PHAN BO FISHER
!
REAL X(N*M),A(M)
REAL R(M*M),RXX(M*M),RY(M),TB(M),RX(M*M)

CALL COVAR(X,N,M,R,TB)
L=0
J=0
DO I=M+1,M*M
  IF (MOD(I,M)/=1) THEN
    L=L+1
    RXX(L)=R(I)
  ELSE
    J=J+1
```

```
      RY(J)=R(I)
    ENDIF
  ENDDO
  RX=RXX
  CALL MINVERT(RX,M-1)
  CALL MULTIP(RX,RY,A,M-1,M-1,1)
  TMP=0.0
  DO J=1,M-1
    TMP=TMP+A(J)*TB(J+1)
  ENDDO
  TMP=TB(1)-TMP
  DO J=M,2,-1
    A(J)=A(J-1)
  ENDDO
  A(1)=TMP

  RX=R
  D=DET(RX,M)
  RX=R
  D11=PPDS(RX,M,1,1)
  TMP=D/D11

  RR=SQRT(1-TMP/R(1))

  Q=0.0
  DO I=1,N
    YHQ=0.0
    DO J=2,M
      IJ=(J-1)*N+I
      YHQ=YHQ+A(J)*X(IJ)
    ENDDO
    YHQ=YHQ+A(1)
    TMP=X(I)-YHQ
    Q=Q+TMP*TMP
  ENDDO
  U=REAL(N)*R(1)-Q
  F=U*REAL(N-M)/(Q*REAL(M-1))
  S=SQRT(Q/REAL(N-M))

  RETURN
  END SUBROUTINE
```

9.3.2. Tính hệ số tương quan riêng

Giả sử $\{x_{t1},x_{t2},...,x_{tm};\ t=1,2,...,n\}$ là tập số liệu quan trắc thực nghiệm của m biến ngẫu nhiên X_1, X_2,..., X_m. Lần lượt xét mối quan hệ tương quan giữa các biến X_1 và X_2 với tập $m-2$ biến còn lại. Giả thiết tất cả các biến đều có trung bình số học bằng 0 ($\overline{x_j} = 0, \forall j$). Khi đó phương trình hồi qui tuyến tính giữa X_1 và X_2 và các biến X_3,..., X_m có thể được viết:

$$\hat{x}_1 = a_3 x_3 + ... + a_m x_m \qquad (9.3.9)$$

$$\hat{x}_2 = b_3 x_3 + ... + b_m x_m \qquad (9.3.10)$$

Ký hiệu q_1 và q_2 tương ứng là thặng dư của X_1 và X_2 đối với các biến $X_3,..., X_m$, ta có:

$$q_1 = x_1 - \hat{x}_1,\ q_2 = x_2 - \hat{x}_2 \qquad (9.3.11)$$

Khi đó hệ số tương quan riêng $r_{12.34...m}$ giữa X_1 và X_2 đối với các biến $X_3,..., X_m$ là hệ số tương quan giữa q_1 và q_2, và được xác định bởi hệ thức:

$$r_{12.34...m} = -\frac{D_{12}}{\sqrt{D_{11}D_{22}}} \qquad (9.3.12)$$

Trong đó D_{12}, D_{11}, D_{22} tương ứng là phần phụ đại số của các phần tử R_{12}, R_{11}, R_{22} của ma trận tương quan mà các phần tử của nó là các mômen tương quan giữa các X_j và X_k, $j,k=1,2,...,m$:

$$R_{jk} = \frac{1}{n}\sum_{t=1}^{n}\left(x_{tj} - \overline{x_j}\right)\left(x_{tk} - \overline{x_k}\right), j,k=1,2,...,m \qquad (9.3.13)$$

$$\left(R_{jk}\right) = \begin{pmatrix} R_{11} & R_{12} & ... & R_{1m} \\ R_{21} & R_{22} & ... & R_{2m} \\ ... & ... & ... & ... \\ R_{m1} & R_{m2} & ... & R_{mm} \end{pmatrix} \qquad (9.3.13')$$

Một cách tổng quát, hệ số tương quan riêng $r_{jk.12...m}$ giữa X_j và X_k đối với các biến $X_1,..., X_m$ còn lại được xác định bởi:

$$r_{jk.12..j-1,j+1...k-1,k+1...m} = -\frac{D_{jk}}{\sqrt{D_{jj}D_{kk}}}, j,k=1,2,...,m \qquad (9.3.14)$$

Sau đây là chương trình tính hệ số tương quan riêng giữa hai biến đối với những biến còn lại trong số m biến được xét.

```
FUNCTION HSTQR(X,N,M,J,K)

! HAM NAY TINH HE SO TQUAN RIENG GIUA BIEN XJ VA XK
! KHI XET DONG THOI M BIEN X1,X2,...,XM
! INPUT: + X(N*M) MANG MOT CHIEU CHUA
!          SO LIEU CAC BIEN
!       + N DUNG LUONG MAU
!       + M SO BIEN DUOC XET
!       + J,K CHI SO BIEN DUOC TINH
!         TUONG QUAN RIENG
!OUTPUT: HE SO TQUAN RIENG GIUA BIEN THU J VA THU K
!
```

```
REAL X(N*M), R(M*M), RT(M*M), TB(M)
CALL COVAR(X,N,M,R,TB)
RT=R
RJK=PPDS(RT,M,J,K)
RT=R
RJJ=PPDS(RT,M,J,J)
RT=R
RKK=PPDS(RT,M,K,K)
HSTQR=-RJK/(SQRT(RJJ*RKK))
RETURN
END FUNCTION
```

9.3.3. Tính hệ số tương quan bội

Giả sử $\{x_{t1},x_{t2},...,x_{tm};\ t=1,2,...,n\}$ là tập số liệu quan trắc thực nghiệm của m biến ngẫu nhiên X_1, X_2,..., X_m. Tương quan giữa X_1 với tập các biến còn lại X_2, X_3,..., X_m được gọi là tương quan bội. Khi đó hệ số tương quan bội giữa X_1 đối với các biến X_2, X_3,..., X_m là hệ số tương quan giữa $\hat{x}_1$ và x_1, và được xác định bởi:

$$r_{1.234...m} = \sqrt{1 - \frac{D}{R_{11}D_{11}}} \qquad (9.3.15)$$

Trong đó D là định thức của ma trận tương quan

$$\left(R_{jk}\right) = \begin{pmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ R_{21} & R_{22} & \dots & R_{2m} \\ \dots & \dots & \dots & \dots \\ R_{m1} & R_{m2} & \dots & R_{mm} \end{pmatrix} \qquad (9.3.16)$$

và D_{11} là phần phụ đại số của phần tử R_{11}.

Sau đây là chương trình tính.

```
FUNCTION HSTQB(R,M,J)
! HAM TINH HE SO TQUAN BOI GIUA BIEN XJ VA TAP HOP
!       M-1 BIEN X1,X2,X(J-1),X(J+1),..,XM
! INPUT: + X(N*M) MANG 1 CHIEU CHUA S/LIEU CAC BIEN
!       + N DUNG LUON MAU
!       + M SO BIEN DUOC XET
!       + J CHI SO BIEN DUOC TINH T/QUAN VOI TAP
!          M-1 BIEN CON LAI
! OUTPUT: HE SO TQUAN BOI GIUA BIEN THU J VA M−1
!        BIEN KHAC
!
REAL X(N*M), R(M*M), RT(M*M), TB(M)
CALL COVAR(X,N,M,R,TB)
RT=R
```

```
RJJ=PPDS(RT,M,J,J)
RT=R
RR=DET(RT,M)
HSTQB=SQRT(1-RR/RJJ)
RETURN
END FUNCTION
```

9.4 PHƯƠNG PHÁP SỐ

9.4.1. Tìm nghiệm phương trình

Trong mục này ta sẽ đề cập đến việc lập chương trình tìm nghiệm của phương trình $F(x) = 0$. Nghiệm của phương trình là giá trị x_0 nào đó mà $F(x_0) = 0$. Nếu $F(x)$ là một hàm đơn giản ta có thể giải và tìm được nghiệm đúng (x_0) của nó. Nếu $F(x)$ là một hàm phức tạp, việc tìm nghiệm đúng của nó gặp rất nhiều khó khăn, và thậm chí là không tìm được bằng phương pháp giải tích. Trong trường hợp đó ta có thể tìm nghiệm gần đúng của phương trình bằng cách lập chương trình cho máy tính. Ở ví dụ 4.3 ta đã xét phương pháp lặp Newton để tìm nghiệm gần đúng của phương trình, trong đó cần phải biết đạo hàm của hàm $F(x)$. Tuy nhiên, trong nhiều trường hợp, việc tính đạo hàm của hàm $F(x)$ cũng khá phức tạp, nên người ta thường sử dụng phương pháp chia đôi. Nói chung, dù sử dụng phương pháp nào, việc tìm nghiệm phương trình $F(x)=0$ bằng phương pháp số đều gắn liền với sự hội tụ nghiệm. Thời gian tính của máy lúc đó sẽ phụ thuộc vào tốc độ hội tụ của phương pháp. Ví dụ phương pháp lặp Newton sẽ cho thời gian tính ít hơn so với phương pháp chia đôi, vì tốc độ hội tụ của nó nhanh hơn. Điều kiện hội tụ có thể được xác định bởi các chỉ tiêu sau:

a) $|F(x_i)| < Epsilon$ (9.4.1)

b) $|x_i - x_0| < Epsilon$ (9.4.2)

c) $|x_i - x_{i-1}| < Epsilon$ (9.4.3)

Trong đó *Epsilon* là một số dương vô cùng bé tùy ý, x_0 là nghiệm chính xác của phương trình, x_i là các giá trị có thể lấy làm xấp xỉ của nghiệm. Ta thấy, chỉ tiêu a) chỉ quan tâm đến độ chính xác của hàm, tức là x_i được lấy làm nghiệm gần đúng nếu nó làm cho giá trị hàm tại đó đủ nhỏ; chỉ tiêu b) xem x_i là nghiệm gần đúng nếu nó rất gần với nghiệm thực; còn chỉ tiêu c) lấy x_i làm nghiệm gần đúng nếu nó rất gần với giá trị xấp xỉ ở bước trước đó. Rõ ràng ta không hy vọng tìm được nghiệm chính xác của phương trình, bởi vì quá trình lặp để tìm nghiệm sẽ kết thúc nếu chỉ tiêu hội tụ thỏa mãn. Ta cũng chưa biết trước nghiệm chính xác x_0 nên chỉ tiêu b) nói chung không được sử dụng trong thực tế. Sau đây ta sẽ xét phương pháp chia đôi tìm nghiệm phương trình $F(x)=0$ khi sử dụng các chỉ tiêu a) và c).

Về nguyên tắc, để tìm nghiệm của phương trình $F(x)=0$, trước hết ta nên khảo sát sơ bộ để xác định số nghiệm của phương trình, các khoảng giá trị của x mà nghiệm có thể nằm trong những khoảng đó. Giả sử ta xét một khoảng được giới hạn bởi hai đầu mút ***XL*** và ***XR*** (điểm bên trái và điểm bên phải). Nếu ***F(XL).F(XR)<0*** thì tồn tại nghiệm trong khoảng này. Để xác định nghiệm, ta tiến hành chia đôi khoảng (***XL, XR***) bởi điểm ***XC*** và xét xem nghiệm rơi vào khoảng nào trong hai khoảng trên

bằng cách xác định dấu của ***F(XL).F(XC)*** và ***F(XC).F(XR)***. Quá trình cứ tiến hành cho đến khi thỏa mãn chỉ tiêu hội tụ nghiệm.

```
SUBROUTINE NGHIEM_CHIA_DOI (XL, XR,EPSILON,X0, ERROR)
!Chuong trinh tim nghiem PT f(x)=0
! INPUT: + XL,XR: Khoang chua nghiem (neu có)
!       + EPSILON: Do chinh xac
! OUTPUT: + X0: Nghiem phuong trinh (neu co)
!        + ERROR=.FALSE. neu có nghiem,
!              =.TRUE. neu khong co nghiem
LOGICAL ERROR
REAL XL,XR,X0,EPSILON
REAL SS,FXL,FXR,FX0
ERROR = .FALSE.     ! Có nghiệm trong khoảng (XL,XR)
IF (FF(XL)==0.) THEN                ! Nghiệm tại đầu mút trái
  X0 = XL
  RETURN
ELSE IF(FF(XR)==0.) THEN ! Nghiệm tại đầu mút phải
  X0 = XR
  RETURN
ELSE IF(XL==XR) THEN  ! Hai đầu mút trùng nhau và
  ERROR = .TRUE.                    ! không phải là nghiệm
  RETURN
ELSE IF(XL>XR) THEN                 ! Đổi vị trí hai đầu mút
  SS = XL
  XL = XR
  XR = SS
END IF
SS = 999. ! Khởi tạo SS cho vòng lặp WHILE
DO WHILE (SS >= EPSILON)
  X0 = (XR+XL)/2.   ! Điểm giữa của khoảng
  FX0=FF(X0)
  FXL=FF(XL)
  FXR=FF(XR)
  IF (FX0 == 0) THEN                ! Nghiệm đúng tại X0
    EXIT
  ELSE IF (FXL*FX0 < 0) THEN ! Nghiệm ở nửa bên trái
    XR = X0
  ELSE  IF (FXR*FX0 < 0) THEN ! Nghiệm ở nửa bên phải
    XL = X0
  END IF
  SS = ABS(XR-XL) ! Chỉ tiêu hội tụ nghiệm là c)
ENDDO
CONTAINS
 FUNCTION FF(X)
  REAL X
!  Định nghĩa hàm F(X) ở đây !!!
  FF = X*X-5.*X+6.
 END FUNCTION
END SUBROUTINE
```

Trong chương trình trên, nếu muốn sử dụng chỉ tiêu a) ta chỉ cần thay câu lệnh gán để tính ***SS*** bởi câu lệnh

SS = ABS(FXR-FXL)

9.4.2. Tính tích phân xác định

Trong mục này ta sẽ xét phương pháp lập trình tính tích phân xác định:

$$I = \int_a^b f(x)dx \qquad (9.4.4)$$

Ở ví dụ 4.2, mục 4.5 ta đã khảo sát cách tính tích phân này bằng phương pháp hình thang. Tuy nhiên đó là một cách tính chưa hiệu quả. Bạn đọc có thể sửa đổi chương trình theo thuật toán sau đây, chạy và so sánh kết quả với chương trình ở ví dụ 4.2. Cũng như ở ví dụ 4.2, để tính tích phân này ta tiến hành chia đoạn (a,b) thành N khoảng bằng nhau bởi các điểm chia x_i, $i=0,2,...,N$. Khi đó, $x_0=a$, x_1, $x_2,...,x_{N-1}$, $x_n=b$. Ký hiệu độ dài mỗi khoảng chia là Δx, ta có $\Delta x = (b-a)/N$. Tích phân I được xấp xỉ bởi tổng diện tích các hình thang giới hạn bởi trục hoành, đường $f(x)$ và các đường thẳng $x=x_i$.

$$I = \int_a^b f(x)dx \approx \frac{\Delta x}{2}\left[\left(f(x_0)+f(x_1)\right)+\left(f(x_1)+f(x_2)\right)+...+\left(f(x_{N-1})+f(x_N)\right)\right]$$

$$=\frac{\Delta x}{2}\left(f(a)+f(b)+2\sum_{i=1}^{N-1}f(x_i)\right) \qquad (9.4.5)$$

Khác với phương pháp hình thang, phương pháp Simpson chia đoạn (a,b) thành ***2N*** khoảng bằng nhau, và thay cho việc nối các điểm $f(x_i)$ liên tiếp bởi các đoạn thẳng trong phương pháp hình thang, ở đây ta sẽ xấp xỉ từng ba điểm liên tiếp bởi đường parabol. Ký hiệu các điểm chia là x_i, $i=0,1,2,...,2N$, ($x_0=a$, x_1, $x_2,...,x_{2N-1}$, $x_{2N}=b$), và độ dài mỗi khoảng chia là $\Delta x = (b-a)/2N$. Khi đó:

$$I = \int_a^b f(x)dx \approx \frac{\Delta x}{3}\left[f(a)+f(b)+2\sum_{i=1}^{N-1}f(x_{2i})+4\sum_{i=1}^{N}f(x_{2i-1})\right] \qquad (9.4.6)$$

Sau đây là chương trình tính tích phân xác định (9.4.4) theo phương pháp Simpson viết dưới dạng chương trình con hàm.

```
FUNCTION SIMPSON (A, B, EP)
! Tinh tich phan xac dinh cua ham f(x) tu A den B
! INPUT: + A, B: Can duoi va can tren cua tich phan
!        + EP: Do chinh xac
! OUTPUT: Gia tri cua tich phan
REAL A,B,DX,S1,S2,SS, EP
INTEGER N, I
N = 0
S1 = 0.
```

```
DO
  N = N + 2
  DX = (B - A)/(2*N)
  S2 = 0.
  DO I = 1, N-1
    S2 = S2 + FF(A + 2*I*DX)
  END DO
  SS = 0.
  DO I = 1, N
    SS = SS + FF(A + (2*I-1)*DX)
  END DO
  S2 = DX/3*(FF(A) + FF(B) + 2*S2 + 4*SS)
  SS = ABS ((S2-S1)/S2)
  IF (SS < EP) EXIT
  S1 = S2
END DO
SIMPSON = S2
CONTAINS
 FUNCTION FF(X)
   REAL X
!  DINH NGHIA HAM F(X) O DAY !!!
   FF = 1/SQRT(8*ATAN(1.))*EXP(−0.5*X*X)
 END FUNCTION
END FUNCTION
```

9.4.3. Sai phân hữu hạn và đạo hàm

Giả sử giá trị của hàm *u(x)* đã biết tại những điểm rời rạc cách đều nhau một khoảng Δx. Khi đó các đạo hàm của *u(x)* có thể nhận được nếu sử dụng sai phân hữu hạn. Thật vậy, nếu khai triển Taylor hàm *u(x)* tại lân cận của *x* ta được:

$$u(x+\Delta x)=u(x)+\left.\frac{du}{dx}\right|_x \frac{\Delta x}{1!}+\left.\frac{d^2u}{dx^2}\right|_x \frac{\Delta x^2}{2!}+\ldots+\left.\frac{d^nu}{dx^n}\right|_x \frac{\Delta x^n}{n!} \qquad (9.4.7)$$

hoặc

$$u(x-\Delta x)=u(x)-\left.\frac{du}{dx}\right|_x \frac{\Delta x}{1!}+\left.\frac{d^2u}{dx^2}\right|_x \frac{\Delta x^2}{2!}+\ldots+(-1)^n\left.\frac{d^nu}{dx^n}\right|_x \frac{\Delta x^n}{n!} \qquad (9.4.8)$$

Từ những hệ thức này ta có thể nhận được các công thức tính đạo hàm của hàm *u(x)* sau:

1) Đạo hàm bậc nhất:

$$\left.\frac{du}{dx}\right|_x = \frac{u(x+\Delta x)-u(x)}{\Delta x}+\frac{d^2u(x)}{dx^2}\frac{\Delta x}{2!}+\ldots \qquad (9.4.9)$$

$$\left.\frac{du}{dx}\right|_x = \frac{u(x)-u(x-\Delta x)}{\Delta x} + \frac{d^2u(x)}{dx^2}\frac{\Delta x}{2!} + \ldots \qquad (9.4.10)$$

$$\left.\frac{du}{dx}\right|_x = \frac{u(x+\Delta x)-u(x-\Delta x)}{2\Delta x} + 2\frac{d^3u(x)}{dx^3}\frac{\Delta x^2}{3!} + \ldots \qquad (9.4.11)$$

Hay có thể viết dưới dạng khác khi bỏ qua các hạng bậc cao:

$$\left.\frac{du}{dx}\right|_x = \frac{u(x+\Delta x)-u(x)}{\Delta x} + \varepsilon(\Delta x) \qquad (9.4.9')$$

$$\left.\frac{du}{dx}\right|_x = \frac{u(x)-u(x-\Delta x)}{\Delta x} + \varepsilon(\Delta x) \qquad (9.4.10')$$

$$\left.\frac{du}{dx}\right|_x = \frac{u(x+\Delta x)-u(x-\Delta x)}{2\Delta x} + \varepsilon(\Delta x)^2 \qquad (9.4.11')$$

trong đó $\varepsilon(\Delta x)$ và $\varepsilon(\Delta x^2)$ tương ứng biểu thị sai số khi xác định đạo hàm và được gọi là sai số bậc nhất và sai số bậc hai. Các công thức (9.4.9'), (9.4.10') được gọi là các đạo hàm có độ chính xác bậc nhất, còn (9.4.11') có độ chính xác bậc hai. Ba công thức tương ứng này còn được gọi là các sơ đồ sai phân tiến, sai phân lùi và sai phân trung tâm.

Như vậy, để tính đạo hàm bậc nhất của $u(x)$ tại x theo (9.4.9') và (9.4.10') ta cần biết giá trị của hàm tại x và tại một điểm lân cận trước hoặc sau x, trong khi để tính theo (9.4.11') ta cần biết cả hai điểm lân cận trước và sau x.

Tương tự, bằng cách khai triển Taylor hàm $u(x)$ ở bốn điểm lân cận ta cũng có thể nhận được công thức tính đạo hàm bậc nhất của $u(x)$ với độ chính xác bậc bốn:

$$\left.\frac{du}{dx}\right|_x = \frac{4}{3}\left[\frac{u(x+\Delta x)-u(x-\Delta x)}{2\Delta x}\right] - \frac{1}{3}\left[\frac{u(x+2\Delta x)-u(x-2\Delta x)}{4\Delta x}\right] \qquad (9.4.12)$$

2) Đạo hàm bậc hai:

Các công thức tính đạo hàm bậc hai của $u(x)$ cũng có thể nhận được bằng cách khai triển thành chuỗi Taylor hàm $u(x)$ tại các lân cận x. Tùy theo cách biểu diễn, ta có thể có các công thức tính với độ chính xác khác nhau. Sau đây là hai công thức thương dùng:

– Độ chính xác bậc hai:

$$\left.\frac{d^2u}{dx^2}\right|_x = \frac{u(x+\Delta x)-u(x-\Delta x)-2u(x)}{\Delta x^2} + \varepsilon(\Delta x^2) \qquad (9.4.13)$$

– Độ chính xác bậc bốn:

$$\frac{d^2u}{dx^2} = \frac{1}{\Delta x^2}\left[-\frac{1}{5}u(x) + \frac{4}{3}\left[u(x+\Delta x) - u(x-\Delta x)\right] - \right.$$
$$\left. -\frac{1}{12}\left[u(x+2\Delta x) - u(x-2\Delta x)\right]\right] + \varepsilon(\Delta x)^4 \qquad (9.4.14)$$

Sau đây là hai chương trình tính đạo hàm bậc nhất và bậc hai của hàm *u(x)* cho tại *n* điểm cách đều nhau. Trong chương trình tính đạo hàm bậc nhất, tham số ***SODO*** cho phép lựa chọn một trong bốn sơ đồ đã trình bày với các độ chính xác khác nhau. Ngoài ra, chương trình còn đưa vào tùy chọn ***CYC*** cho điều kiện biên là tuần hoàn hoặc không tuần hoàn.

```
SUBROUTINE DH1(U,UX,N,DX,SODO,CYC)
! CHUONG TRINH TINH DAO HAM BAC NHAT CUA HAM U(X)
! INPUT: + U(N) MANG 1 CHIEU CHUA GIA TRI CUA U(X)
!        + N: SO DIEM CO GIA TRI CUA U(X)
!        + DX: DO DAI KHOANG CHIA CUA X
!        + SODO: =1 SAI PHAN TIEN
!                =2 SAI PHAN LUI
!                =3 SAI PHAN TRUNG TAM DCX=2
!                =4 SAI PHAN TRUNG TAM DCX=4
!        + CYC: = .TRUE. NEU BIEN TUAN HOAN
!               = .FALSE. NEU BIEN KHONG TUAN HOAN
! OUTPUT: UX: DAO HAM CUA U(X)
!          SODO=1,CYC=.FALSE.: UX(1:N-1)
!          SODO=2,CYC=.FALSE.: UX(2:N)
!          SODO=3,CYC=.FALSE.: UX(2:N-1)
!          CYC=.TRUE.: UX(1:N)
!
INTEGER N,SODO, N1,N2,N3
LOGICAL CYC
REAL U(N),UX(N),C1,C2,DX2,DX4
SELECT CASE (SODO)
  CASE (1)
    DO I=1,N-1
      UX(I) = (U(I+1)-U(I))/DX
    ENDDO
    IF (CYC) UX(N) = UX(1)
  CASE (2)
    DO I=2,N
      UX(I) = (U(I)-U(I-1))/DX
    ENDDO
    IF (CYC) UX(1) = UX(N)
  CASE (3)
    DX2=2*DX
    DO I=2,N-1
      UX(I) = (U(I+1)-U(I-1))/DX2
    ENDDO
    UX(1) = (U(2)-U(1))/DX
    UX(N) = (U(N)-U(N-1))/DX
```

```
      IF (CYC) THEN
        UX(1) = (U(2)-U(N-1))/(2*DX)
        UX(N) = UX(1)
      END IF
    CASE (4)
      N1 = N-1
      N2 = N-2
      N3 = N-3
      C1 = 4./3.
      C2 = 1./3.
      DX2 = 1./(2.*DX)
      DX4 = 1./(4.*DX)
      DO I = 3, N2
        UX(I) = C1*((U(I+1)-U(I-1))*DX2) &
              -C2*((U(I+2)-U(I-2))*DX4)
      ENDDO
      IF (CYC) THEN
        UX(2)  = C1*((U(3)-U(1))*DX2)-C2 &
               *((U(4)-U(N-1))*DX4)
        UX(1)  = C1*((U(2)-U(N-1))*DX2)-C2 &
               *((U(3)-U(N-2))*DX4)
        UX(N)  = UX(1)
        UX(N-1)= C1*((U(N)-U(N-2))*DX2)-C2 &
               *((U(2)-U(N-3))*DX4)
      END IF
    END SELECT
END SUBROUTINE
!-----------------------------------
!
SUBROUTINE DH2 (U,UX,N,DX,SODO)
! CHUONG TRINH TINH DAO HAM BAC HAI CUA HAM U(X)
! INPUT: + U(N) MANG 1 CHIEU CHUA GIA TRI CUA U(X)
!        + N: SO DIEM CO GIA TRI CUA U(X)
!        + DX: DO DAI KHOANG CHIA CUA X
!        + SODO: =1 DO CHINH XAC BAC 2
!                =2 DO CHINH XAC BAC 4
! OUTPUT: UX: DAO HAM CUA U(X)
!           SODO=1: UX(2:N-1)
!           SODO=2: UX(3:N-2)
INTEGER N,SODO
REAL U(N),UX(N), DX
REAL DX2,C15,C43,C12

DX2=DX*DX
SELECT CASE (SODO)
  CASE (1)
    DO I=2,N-1
      UX(I) = (U(I+1)-U(I-1)-2.*U(I))/DX2
    ENDDO
  CASE (2)
```

```
      C15=1./5.
      C43=4./3.
      C12=1./12.
      DO I=3,N-2
        UX(I)=(-C15*U(I)+C43*(U(I+1)-U(I-1)) &
            -C12*(U(I+2)-U(I-2)))/DX2
      ENDDO
    END SELECT
END SUBROUTINE
!
```

9.4.4. Toán tử Laplaxian

Giả sử *u(x,y)* là hàm hai biến đối với *x* và *y*. Khi đó hệ thức:

$$\nabla^2 u(x,y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \equiv \nabla^2 u \qquad (9.4.15)$$

được gọi là Laplaxian của hàm *u(x,y)*. Đây là một trong những toán tử được dùng khá phổ biến trong một số lĩnh vực.

Để xác định toán tử này, hàm *u(x,y)* thường được cho trên một hệ thống lưới điều hòa của các biến *x* và *y*: $u_{ij} = u(x_i,y_j)$, trong đó các (x_i,y_j) là các điểm nút lưới. Giả thiết khoảng cách giữa các nút lưới không đổi và bằng *h* theo cả chiều *x* và *y*. Khi đó, bằng phép khai triển Taylor lân cận điểm *(x,y)* và thực hiện các phép biến đổi đơn giản ta có thể nhận được các công thức tính Laplaxian của *u(x,y)* sau đây:

1) Sơ đồ 5 điểm, độ chính xác bậc hai:

$$\nabla^2 u = \frac{1}{h^2}\left[u(x,y+h) + u(x-h,y) + u(x,y-h) + u(x+h,y) - 4(u(x,y)\right] + \varepsilon(h^2) \qquad (9.4.16)$$

2) Sơ đồ 9 điểm, độ chính xác bậc hai:

$$\nabla^2 u = \frac{1}{6h^2}\left[4\{4u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h)\} + \right.$$
$$+ \{u(x+h,y+h) + u(x-h,y+h) + u(x+h,y-h) +$$
$$\left. + u(x-h,y-h) - 20u(x,y)\}\right] + \varepsilon(h^2) \qquad (9.4.17)$$

Chương trình tính toán tử Laplaxian theo các sơ đồ trên được trình bày dưới đây.

```
SUBROUTINE LAPLA52 (U,LAPU,H,N,M)
! CHUONG TRINH TINH LAPLAXIAN CUA U(X,Y)
! THEO SO DO 5 DIEM, DO CHINH XAC BAC 2
! INPUT: + U(N,M) GIA TRI CUA U
!        + N,M SO DIEM NUT THEO X VA Y
!        + H KHOANG CACH GIUA CAC DIEM NUT
! OUTPUT: LAPU(2:N-1,2:M-1) LAPLAXIAN CUA U
INTEGER N,M
```

```
REAL U(N,M),LAPU(N,M),H
INTEGER IP1, IM1, JP1, JM1

N1=N-1
M1=M-1
DO I=2,N1
  DO J=2,M1
    IP1=I+1
    IM1=I-1
    JP1=J+1
    JM1=J-1
    LAPU(I,J)=(U(IP1,J)+U(IM1,J)+U(I,JP1) &
          +U(I,JM1)-4.*U(I,J))/H**2
  ENDDO
ENDDO
RETURN
END SUBROUTINE

!-----------------------------------------

SUBROUTINE LAPLA92 (U,LAPU,H,N,M)
! CHUONG TRINH TINH LAPLAXIAN CUA U(X,Y)
! THEO SO DO 9 DIEM, DO CHINH XAC BAC 2
! INPUT: + U(N,M) GIA TRI CUA U
!       + N,M SO DIEM NUT THEO X VA Y
!       + H KHOANG CACH GIUA CAC DIEM NUT
! OUTPUT: LAPU(2:N-1,2:M-1) LAPLAXIAN CUA U
INTEGER N,M
REAL U(N,M),LAPU(N,M),H
INTEGER IP1, IM1, JP1, JM1

N1=N-1
M1=M-1
DO I=2,N1
  DO J=2,M1
    IP1=I+1
    IM1=I-1
    JP1=J+1
    JM1=J-1
    LAPU(I,J)=(U(IP1,JP1)+U(IP1,JM1)+U(IM1,JP1) &
     +U(IM1,JM1)+4.*(U(IP1,J)+U(IM1,J)+U(I,JP1) &
     +U(I,JM1))-20.*U(I,J))/(6.*H**2)
  ENDDO
ENDDO
RETURN
END SUBROUTINE
```

9.4.5. Giải phương trình truyền nhiệt

Bài toán truyền nhiệt trên thanh cũng là một trong những bài toán khá phổ biến. Phương trình truyền nhiệt dạng tổng quát có thể được viết dưới dạng:

$$\frac{\partial U}{\partial t} = \frac{\partial}{\partial x}\left(A\frac{\partial U}{\partial x} \right) \qquad (9.4.18)$$

trong đó A là hệ số dẫn nhiệt, t là thời gian, x là tọa độ các điểm trên thanh. Nếu A là hằng số, phương trình (9.4.18) có thể được viết lại:

$$\frac{\partial U}{\partial t} = A\frac{\partial^2 U}{\partial x^2} \qquad (9.4.19)$$

Để đơn giản trong trình bày, ở đây ta giả thiết $A=1$, và do đó phương trình truyền nhiệt sẽ có dạng đơn giản:

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} \qquad (9.4.20)$$

Giả sử xét sự truyền nhiệt trên thanh đồng nhất có độ dài hữu hạn bằng L. Bài toán yêu cầu xác định sự phân bố của U theo thời gian t và vị trí trên thanh x, tức xác định hàm $U(x,t)$. Để giải bài toán bằng phương pháp sai phân hữu hạn, ta chia thanh thành các đoạn đều nhau $\Delta x = h$. Ký hiệu bước thời gian tích phân là $\Delta t = k$. Khi đó tại điểm x_i và thời điểm t_j ta có $U(x_i,t_j) = U(ih,jk) = U_{ij}$, với $i,j=0,1,2,...$ Gọi u_{ij} là xấp xỉ của U_{ij} tại (x_i,t_j), ta cần tìm tất cả các u_{ij} tại các điểm nút (x_i,t_j). Thuật toán để giải bài toán có thể được mô tả như sau.

Xấp xỉ các đạo hàm hai vế của (9.4.20) bởi các sai phân hữu hạn, ta có:

$$\frac{\partial U}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{k} \qquad (9.4.21)$$

Tại thời điểm t_j:

$$\frac{\partial^2 U}{\partial x^2} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{h^2} \qquad (9.4.22)$$

Tại thời điểm t_{j+1}:

$$\frac{\partial^2 U}{\partial x^2} = \frac{u_{i+1,j+1} + u_{i-1,j+1} - 2u_{i,j+1}}{h^2} \qquad (9.4.23)$$

Cộng từng vế (9.4.22) và (9.4.23) ta được:

$$\frac{\partial^2 U}{\partial x^2} = \frac{1}{h^2}\left(u_{i+1,j+1} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j} + u_{i-1,j} - 2u_{i,j} \right) \qquad (9.4.24)$$

Ký hiệu $r = \frac{k}{h^2}$, kết hợp (9.4.21) và (9.4.24) ta được:

$$-ru_{i-1,j+1} + (2+2r)u_{i,j+1} - ru_{i+1,j+1} = ru_{i-1,j} + (2-2r)u_{i,j} + ru_{i+1,j}$$

(9.4.25)

Nếu khoảng thời gian tích phân là T, thì số bước thời gian tính sẽ là $NT = T/k$. Vì độ dài thanh là L nên số khoảng chia trên thanh sẽ là $NX = L/h$. Do đó $i=0,1,2,...,NX$, $j=0,1,2,...,NT$. Từ (9.4.25) ta có hệ phương trình:

$$\begin{cases} -ru_{0,j+1} + (2+2r)u_{1,j+1} - ru_{2,j+1} = ru_{0,j} + (2-2r)u_{1,j} + u_{2,j} \\ -ru_{1,j+1} + (2+2r)u_{2,j+1} - ru_{3,j+1} = ru_{1,j} + (2-2r)u_{2,j} + u_{3,j} \\ -ru_{2,j+1} + (2+2r)u_{3,j+1} - ru_{4,j+1} = ru_{2,j} + (2-2r)u_{3,j} + u_{4,j} \\ \ldots \\ -ru_{Nx-2,j+1} + (2+2r)u_{Nx-1,j+1} - ru_{Nx,j+1} = ru_{Nx-2,j} + \\ \qquad + (2-2r)u_{Nx-1,j} + u_{Nx,j} \end{cases} \quad (9.4.26)$$

Hệ gồm $NX–1$ phương trình, với $NX–1$ ẩn số là các giá trị $u_{i,j+1}$ $(i=1,...,NX–1)$ tại thời điểm t_{j+1}, trong đó các giá trị $u_{i,j}$ của bước thời gian trước được xem là đã biết.

Khi cho trước các giá trị trên biên $u_{0,j+1}$ và $u_{NX,j+1}$, ma trận các hệ số vế trái của hệ (9.4.26) là một ma trận ba đường chéo có dạng:

$$A = \begin{pmatrix} (2+2r) & -r & 0 & \ldots & 0 \\ -r & (2+2r) & -r & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & -r & (2+2r) & -r \\ 0 & 0 & 0 & -r & (2+2r) \end{pmatrix} \quad (9.4.27)$$

Đây là một ma trận dạng "ba đường chéo", nghĩa là ma trận trong đó chỉ có các phần tử thuộc đường chéo chính và hai dãy trên và dưới nó là khác không.

Dưới dạng ma trận, hệ (9.4.26) có thể được viết:

$$\begin{pmatrix} (2+2r) & -r & 0 & \ldots & 0 \\ -r & (2+2r) & -r & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & -r & (2+2r) & -r \\ 0 & 0 & 0 & -r & (2+2r) \end{pmatrix} \begin{pmatrix} u_{1,j+1} \\ u_{2,j+1} \\ \ldots \\ u_{Nx-2,j+1} \\ u_{Nx-1,j+1} \end{pmatrix} =$$

$$= \begin{pmatrix} ru_{0,j} + (2-2r)u_{1,j} + u_{2,j} + ru_{0,j+1} \\ ru_{1,j} + (2-2r)u_{2,j} + u_{3,j} \\ \ldots \\ ru_{Nx-3,j} + (2-2r)u_{Nx-2,j} + u_{Nx-1,j} \\ ru_{Nx-2,j} + (2-2r)u_{Nx-1,j} + u_{Nx,j} + ru_{Nx,j+1} \end{pmatrix} \quad (9.4.28)$$

Để giải hệ này ta cần biết giá trị của $U(x, t)$ tại mọi x khi $t=0$ ($j=0$, điều kiện ban đầu) và tại mọi t khi x nhận giá trị hai đầu mút ($x=0$ và $x=L$, tức $i=0$ và $i=NX$ – điều kiện biên). Khi đã biết u tại $j=0$ và u tại $i=0$ và $i=NX$, vế phải của hệ sẽ được xác định. Giải hệ ta sẽ nhận được $u_{i,1}$ (tại bước thời gian $j=1$). Sau khi nhận được các $u_{i,1}$, thay vào vế phải và giải hệ ta sẽ nhận được $u_{i,2}$. Quá trình cứ tiếp tục cho đến khi nhận được $u_{i,NT}$ là giá trị của u tại bước thời gian cuối cùng.

Cuối cùng ta có các bước giải như sau.

1) Xác định giá trị độ dài thanh L, độ dài khoảng chia h, khoảng thời gian tích phân T và bước thời gian k

2) Tính số khoảng chia theo x: $NX = L/h$, và số bước tích phân thời gian $NT = T/k$

3) Tính giá trị của $r = k/h^2$

4) Xác định ba đường chéo của ma trận A (9.4.27)

5) Xác định điều kiện ban đầu $U(0:NX, 0)$ và các điều kiện biên $U(0, 0:NT)$, $U(NX, 0:NT)$. Nếu những điều kiện này được cho dưới dạng các biểu thức giải tích ta phải tính giá trị của chúng tại các điểm rời rạc, tức là tính các giá trị $u_{i,0}=U(x_i,0)$, $u_{0,j}=U(0,t_j)$, $u_{NX,j}=U(L,t_j)$, $i=0,1,2,...,NX$, $j=0,1,2,...,NT$. Trong thực tế, điều kiện ban đầu và điều kiện biên được cho bởi tập các giá trị rời rạc của u: $\{u_{0,0}, u_{1,0},..., u_{NX,0}\}$, $\{u_{0,0}, u_{0,1},..., u_{0,NT}\}$ và $\{u_{NX,0}, u_{NX,1},..., u_{NX,NT}\}$.

6) Thực hiện vòng lặp tích phân thời gian:

a) Xuất phát từ $j = 0$

b) Tính vectơ vế phải $G(1:NX–1)$ tại j

c) Giải hệ để xác định $U(1:NX, j+1)$

d) Nếu $j<NT$ thì tăng j lên 1 đơn vị

e) Quay lại bước b)

Quá trình lặp cứ tiếp tục cho đến khi $j=NT$ thì kết thúc.

Để minh họa ta sẽ khảo sát ví dụ sau đây. Giả sử $L = 1$ ($0 \le x \le 1$), $h = 0.1$ và $k = 0.01$, khi đó $r = k/h^2 = 1$. Số khoảng chia theo x là $NX = 1/0.1 = 10$ ($i = 0,1,...,10$).

– Điều kiện biên được chọn là: $U(0, t) = U(1, t) = 0$ hay $u_{0,j} = u_{NX,j} = 0$ (với mọi j)

– Điều kiện ban đầu là: $U(x,0) = \begin{cases} 2x, & 0 \le x \le 1/2 \\ 2(1-x), & 1/2 \le x \le 1 \end{cases}$

Hay $u_{i,0} = \{ 0, 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2, 0 \}$.

Khi đó ta có chương trình tính như sau.

```
PROGRAM PT_Truyen_Nhiet
IMPLICIT NONE
```

```
INTEGER N, NT
REAL, ALLOCATABLE :: A(:), B(:), C(:),  &
                     G(:), U(:,:), UX(:)
INTEGER I, J
REAL H, K, R, T, L, X
L = 1.   ! Độ dài thanh
K = 0.01 ! Bước thời gian
H = 0.1  ! Khoảng chia theo x
R = K / H ** 2
N = NINT(L/H) ! Số khoảng chia theo x
NT = 100      ! Số bước tích phân
ALLOCATE(A(2:N-1), B(1:N-1), C(1:N-2), &
         G(1:N-1), U(0:N,0:NT), UX(1:N-1))
A = -R        ! Xác định ba đường chéo của A
B = 2 + 2 * R
C = -R
DO I = 0, N   ! Điều kiện ban đầu
  X = I*H
  U(I,0) = 2 * X
  IF (X > 0.5) U(I,0) = 2*(1-X)
END DO
U(0,:) = 0.   ! Điều kiện biên
U(N,:) = 0.
T = 0
PRINT "(11F7.4)", (I * H, I = 1, N-1)
DO J = 1, NT  ! Thời điểm tích phân
  G = R * (U(0:N-2,0) + U(2:N,0)) + &
    (2 - 2 * R) * U(1:N-1,0)    ! Vế phải
  G(1) = G(1) + R * U(0,J)
  G(N-1) = G(N-1) + R * U(N,J)

  CALL BaDuongCheo( A, B, C, UX, G )
  U(1:N-1,J) = UX
  U(1:N-1,0) = UX
END DO
DO J=1, NT
   PRINT "(11F7.4)", U(1:N,J)
ENDDO
CONTAINS
SUBROUTINE BaDuongCheo ( A, B, C, X, G )
IMPLICIT NONE
  REAL B(:)              ! Đường chéo chính
  REAL A(2:)             ! Đường chéo dưới
  REAL C(:)              ! Đường chéo trên
  REAL, INTENT(OUT) :: X(:)  ! Ẩn
  REAL G(:)              ! Vế phải
  REAL W( SIZE(B) )      ! Mảng làm việc trung gian
  REAL T
  INTEGER I, J, N
  N = SIZE(B)
```

```
  W = B
  DO I = 2, N
    T = A(I) / W(I-1)
    W(I) = W(I) - C(I-1) * T
    G(I) = G(I) - G(I-1) * T
  END DO
  X(N) = G(N) / W(N)
  DO I = 1, N-1
    J = N-I
    X(J) = (G(J) - C(J) * X(J+1)) / W(J)
  END DO
END SUBROUTINE BaDuongCheo
END
```

Trong chương trình trên, thủ tục ***BaDuongCheo*** là chương trình con trong giải hệ phương trình đại số tuyến tính mà ma trận hệ số vế trái là ma trận "ba đường chéo".

9.4.6. Xây dựng cơ sở dữ liệu

Có rất nhiều phần mềm chuyên dụng về cơ sở dữ liệu hiện đang được lưu hành. Mỗi một cơ sở dữ liệu đều có một kiểu cấu trúc dữ liệu riêng, nói chung không giống nhau. Và do đó, trong nhiều trường hợp ta không thể truy cập được các cơ sở dữ liệu này bằng Fortran. Bởi vậy ta cần phải xây dựng cơ sở dữ liệu cho riêng mình. Trong mục này ta sẽ tìm hiểu cách xây dựng một cơ sở dữ liệu bằng ngôn ngữ Fortran.

Việc xây dựng một cơ sở dữ liệu có thể bao gồm các nhiệm vụ sau:

1) Tạo một cấu trúc dữ liệu. Tùy thuộc vào từng mục đích cụ thể cũng như yêu cầu lưu trữ, khai thác thông tin, cấu trúc dữ liệu cần phải bảo đảm các nguyên tắc: rõ ràng, đầy đủ, dễ truy cập và chiếm ít không gian bộ nhớ nhất (cả bộ nhớ trong và bộ nhớ ngoài). Thực chất đây là giai đoạn thiết kế cấu trúc cơ sở dữ liệu.

2) Xây dựng chương trình quản trị, khai thác dữ liệu. Đây là một bộ chương trình cho phép thực hiện các chức năng tạo mới, cập nhật, bổ sung, sửa chữa, hiển thị và kết xuất thông tin từ cơ sở dữ liệu. Trong nhiều trường hợp bộ chương trình này còn có thể có các chức năng tính toán, xử lý dữ liệu. Để có được một bộ chương trình hoàn thiện, ta cần phải tiến hành việc phân tích, thiết kế chương trình một cách kỹ lưỡng, tỷ mỷ, và có thể cần có cả những thuật toán tối ưu.

3) Tổ chức lưu trữ dữ liệu. Nếu khối lượng dữ liệu lớn, vấn đề tổ chức lưu trữ là rất quan trọng, vì nó liên quan đến sự an toàn, khả năng bảo mật (nếu cần), tính thuận tiện trong truy cập, khai thác,... Có lẽ đây là vấn đề mà người xây dựng cơ sở dữ liệu cần phải lường được trước khi bắt tay vào thiết kế, xây dựng.

Ở đây ta sẽ chỉ để cập đến một số khía cạnh rất nhỏ liên quan đến nhiệm vụ thứ hai. Hơn nữa, ta cũng sẽ chỉ kết hợp kiểu dữ liệu có cấu trúc **TYPE** với các file truy cập trực tiếp và xây dựng một chương trình minh họa những nguyên tắc cơ bản khi thiết lập, hiển thị và cập nhật cơ sở dữ liệu.

Giả sử ta muốn thiết lập một cơ sở dữ liệu về sinh viên, trong đó thông tin đầy đủ của mỗi sinh viên được mô tả bằng một bản ghi. Để đơn giản, ta giả thiết thông tin về mỗi sinh viên chỉ bao gồm họ tên và một số nguyên chỉ điểm thi nào đó. Trên thực tế, thông tin mô tả đầy đủ về một sinh viên khá phức tạp, ví dụ điểm có thể được mô tả bởi một mảng hoặc một cấu trúc như đã thấy ở chương trước, nhưng vì mục đích của ta chỉ tập trung vào việc xây dựng cơ sở dữ liệu, nên nếu đưa vào quá nhiều thông tin sẽ làm phức tạp chương trình một cách không cần thiết. Bạn đọc có thể tự phát triển nó cho riêng mình.

Giả sử cấu trúc dữ liệu của một bản ghi trong cơ sở dữ liệu được khai báo bởi:

```
TYPE MauHSSV
  CHARACTER (NameLen) Name
  INTEGER  Mark
END TYPE MauHSSV
```

Ta cần phải tạo ra một số chương trình con để đọc và ghi các biến của cấu trúc này đối với file truy cập trực tiếp. Muốn vậy, trước hết ta phác thảo một "khung" chương trình bao gồm chương trình chính và các chương trình con có thể có. Nội dung chi tiết của chương trình sẽ được viết dần dần. Ngoài ra, để tiện trình bày, các chương trình con đều được khai báo như là những chương trình con trong, và file chỉ được mở một lần ngay đầu chương trình chính và sẽ được đóng lại trước khi kết thúc chương trình. Trong thực tế, các chương trình con nên được tổ chức thành *modul* với một số khai báo biến toàn cục, như định nghĩa kiểu dữ liệu (**TYPE**), tên file,… Trong trường hợp đó, mỗi chương trình con khi thao tác với file có thể mở và đóng file ngay trong đó. Có thể phác họa chương trình như sau:

```
PROGRAM HO_SO_SINH_VIEN
IMPLICIT NONE
INTEGER, PARAMETER :: NameLen = 20
TYPE MauHSSV
  CHARACTER (NameLen) Name
  INTEGER  Mark
END TYPE MauHSSV
TYPE (MauHSSV) Student
INTEGER  EOF, RecLen, RecNo
LOGICAL  IsThere
CHARACTER (NameLen) FileName
CHARACTER  Ans
CHARACTER (7) FileStatus
CHARACTER (*), PARAMETER :: NameChars = &
          " abcdefghijklmnopqrstuvwxyz" &
           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
!
INQUIRE (IOLENGTH = RecLen) Student
WRITE (*, "('Ten File: ')", ADVANCE = "NO")
READ*, FileName
INQUIRE (FILE = FileName, EXIST = IsThere)
IF (IsThere) THEN
  WRITE (*, "('File dang ton tai. &
       Xoa va Thay the? (Y/N)? ')", &
       ADVANCE = "NO")
  READ*, Ans
```

```
  IF (Ans == "Y") THEN
    FileStatus = "REPLACE" ! Xóa file và tạo mới
  ELSE
    FileStatus = "OLD"  ! Cập nhật vào file cũ
  END IF
ELSE
  FileStatus = "NEW"  ! File không tồn tại. Tạo file mới.
END IF
OPEN (1, FILE = FileName, STATUS = FileStatus, &
        ACCESS = 'DIRECT', RECL = RecLen)
Ans = ""    ! Khởi tạo giá trị bằng trống rỗng
DO WHILE (Ans /= "Q")
  PRINT*
  PRINT*, "A: Them ban ghi moi"
  PRINT*, "D: Hien thi tat ca cac ban ghi"
  PRINT*, "Q: Thoat"
  PRINT*, "U: Cap nhat ban ghi dang co"
  PRINT*
  WRITE (*, "('Ban chon? (ENTER): ')", &
          ADVANCE = "NO")
  READ*, Ans
  SELECT CASE (Ans)
    CASE ("A", "a")
      CALL ThemBanGhi
    CASE ("D", "d")
      CALL HienThi
    CASE ("U", "u")
      CALL CapNhat
  END SELECT
END DO
CLOSE (1)
CONTAINS
  SUBROUTINE ThemBanGhi
  ...
  SUBROUTINE HienThi
  ...
  SUBROUTINE DocSoNguyen( Num )
  ...
  SUBROUTINE XoaDauCach( Str )
  ...
  SUBROUTINE CapNhat
  ...
END
```

Trong chương trình trên, độ dài trường ***Name*** của ***MauHSSV*** được khai báo là hằng vì nó sẽ được sử dụng trong các khai báo khác nữa. Biến cơ bản trong chương trình là ***Student*** có kiểu dữ liệu ***MauHSSV***. Câu lệnh **INQUIRE** để xác định độ dài bản ghi cho câu lệnh **OPEN** sau đó. Chương trình sẽ dùng hội thoại để người sử dụng nhập tên file. Câu lệnh **INQUIRE** tiếp theo xác định xem file có

tồn tại không. Tùy thuộc vào trạng thái tồn tại của file mà tham số ***STATUS*** trong lệnh **OPEN** sẽ mở file cho hợp lý.

Đoạn chương trình tiếp theo tạo thực đơn (***Menu***) dạng đối thoại, cho phép người sử dụng lựa chọn công việc sẽ làm. Trong thực đơn của chương trình người sử dụng có thể gõ ký tự in thường hoặc in hoa. Tuy nhiên, sẽ thuận lợi hơn nếu ta xây dựng thêm một chương trình con đổi chữ thường thành chữ hoa như sau.

```
FUNCTION ChToUpper( Ch )
 ! Chuong trinh doi chu in thuong thanh chu in hoa
  CHARACTER Ch, ChToUpper
  ChToUpper = Ch
  SELECT CASE (Ch)
    CASE ( "a":"z" )
    ChToUpper = CHAR( ICHAR(Ch) + &
          ICHAR("A") - ICHAR("a") )
  END SELECT
END FUNCTION ChToUpper
```

Chương trình con ***ThemBanGhi*** làm nhiệm vụ chèn thêm một bản ghi mới vào cuối file dữ liệu đang tồn tại hoặc chèn vào đầu một file mới.

```
SUBROUTINE ThemBanGhi
RecNo = 0
EOF = 0
DO WHILE (EOF == 0)
  READ( 1, REC = RecNo+1, IOSTAT = EOF )
  IF (EOF == 0) THEN
    RecNo = RecNo + 1
  END IF
END DO
RecNo = RecNo + 1
Student = MauHSSV( "a", 0 )
DO WHILE ((VERIFY( Student % Name,NameChars )== 0))
  PRINT*, "Cho ten SV: "
  READ "(A20)", Student % Name
  IF (VERIFY(Student % Name,NameChars ) == 0) THEN
    PRINT*, "Mark: "
    CALL DocSoNguyen( Student % Mark )
    WRITE (1, REC = RecNo) Student
    RecNo = RecNo + 1
  END IF
END DO
END SUBROUTINE ThemBanGhi
```

Vì Fortran không có khả năng xác định số bản ghi trong file, nên ta phải thực hiện việc "đọc bỏ qua" các bản ghi để nhận biết số bản ghi này. Cách đọc bỏ qua này không tốn nhiều thời gian. Số bản ghi sẽ được xác định bởi biến ***RecNo*** trong vòng lặp **DO WHILE**. Câu lệnh **WRITE** sẽ ghi vào file toàn bộ thông tin của một sinh viên chứa trong bản ghi. Chương trình con **DocSoNguyen** được gọi để nhập điểm là một số nguyên hợp lệ. Chương trình con **HienThi** sử dụng vòng lặp **DO WHILE** để đọc

và hiển thị nội dung file. Chương trình con **CapNhat** thực hiện việc tìm tên một sinh viên và sửa đổi nội dung thông tin về sinh viên này. Trong trường hợp ở đây, thông tin chỉ có một trường điểm (**Mark**). Chương trình con **XoaDauCach** dùng để xóa bỏ các dấu cách (nếu có) trong biến đưa vào để tìm (tên sinh viên) và trường ***Name*** của bản ghi để đảm bảo việc so sánh hai xâu ký tự.

```
SUBROUTINE DocSoNguyen( Num )
 INTEGER Err, Num
 Err = 1
 DO WHILE (Err > 0)
   READ (*, *, IOSTAT = Err) Num
   IF (Err > 0) PRINT*, "Sai du lieu! Vao lai."
 END DO
 END SUBROUTINE DocSoNguyen

SUBROUTINE HienThi
 RecNo = 1
 EOF = 0
 DO WHILE (EOF == 0)
   READ (1, REC = RecNo, IOSTAT = EOF) Student
   IF (EOF == 0) THEN
     PRINT "(A20, I3)", Student
   END IF
   RecNo = RecNo + 1
 END DO
 END SUBROUTINE HienThi

SUBROUTINE CapNhat
 CHARACTER (NameLen) Item, Copy
 LOGICAL Found
 Found = .false.
 EOF = 0
 PRINT*, "Sua diem cho ai?"
 READ "(A20)", Item
 CALL XoaDauCach( Item )
 RecNo = 1
 DO WHILE (EOF == 0 .AND. .NOT. Found)
   READ (1, IOSTAT = EOF, REC = RecNo) Student
   IF (EOF == 0) THEN
     Copy = Student % Name
     CALL XoaDauCach( Copy )
     IF (Item == Copy) THEN
       Found = .true.    ! Tìm thấy
       PRINT*, 'Found at recno', RecNo, &
             ' Enter new mark:'
       CALL DocSoNguyen( Student % Mark )
       WRITE (1, REC = RecNo) Student
                    ! Ghi vào file
     ELSE
       RecNo = RecNo + 1
```

```
      END IF
    END IF
  END DO
  IF (.NOT. Found) THEN
    PRINT*, Item, ' Khong tim thay...'
  END IF
  END SUBROUTINE CapNhat

SUBROUTINE XoaDauCach( Str )
  CHARACTER (*) Str
  INTEGER I
  I = 1
  DO WHILE (I < LEN_TRIM( Str ))
    IF (Str(I:I) == " ") THEN
      Str(I:) = Str(I+1:)
    ELSE
      I = I + 1
    END IF
  END DO
  END SUBROUTINE XoaDauCach
```

BÀI TẬP CHƯƠNG 9

9.1 Viết chương trình nhập vào một mảng một chiều gồm N phần tử là những số thực chứa giá trị quan trắc của một biến ngẫu nhiên. Tính các đặc trưng trung bình số học, phương sai, độ lệch chuẩn, độ bất đối xứng, trung vị và các tứ vị của chuỗi. In kết quả theo qui cách mỗi một đặc trưng trên một dòng với những chú thích hợp lý.

9.2 Cho file số liệu dạng TEXT chứa kết quả quan trắc của các biến $X_1, X_2, ..., X_m$. Cấu trúc file như sau. Dòng 1 là tiêu đề mô tả nội dung file. Dòng 2 là hai số nguyên dương (N, M) chỉ số lần quan trắc (N – dung lượng mẫu) và số biến (M). Các dòng tiếp theo mỗi dòng chứa M số thực là giá trị quan trắc $x_{i1}, x_{i2}, ..., x_{im}$ lần thứ i ($i=1,2,...,N$) của các biến $X_1, X_2, ..., X_m$, các giá trị được viết cách nhau ít nhất một dấu cách. Hãy đọc file số liệu và tính các đặc trưng thống kê của các biến $X_1, X_2, ..., X_m$: trung bình số học, trung vị (median), phương sai, độ lệch chuẩn, các mômen gốc và mômen trung tâm bậc 2, 3, 4. In kết quả vào một file mới dưới dạng thích hợp.

9.3 Cũng với file số liệu như ở bài tập 9.2, hãy viết chương trình tính: Trung bình số học, độ lệch chuẩn của các biến $X_1, X_2, ..., X_m$ và các ma trận tương quan, ma trận tương quan chuẩn hóa của chúng. In kết quả vào một file mới.

9.4 Cho file số liệu dạng TEXT chứa kết quả quan trắc của biến Y (biến phụ thuộc) và các biến $X_1, X_2, ..., X_m$ (biến độc lập). Cấu trúc file như sau. Dòng 1 là tiêu đề mô tả nội dung file. Dòng 2 là hai số nguyên dương (N, M) chỉ số lần quan trắc (N – dung lượng mẫu) và số biến độc lập (M). Các dòng tiếp theo mỗi dòng chứa $M+1$ số thực là giá trị quan trắc $y_i, x_{i1}, x_{i2}, ..., x_{im}$ lần thứ i ($i=1,2,...,N$) của các biến $Y, X_1, X_2, ..., X_m$, các giá trị được viết cách nhau ít nhất một dấu cách. Hãy viết chương trình đọc file số liệu và tính: Trung bình số học và độ lệch chuẩn của các biến $Y, X_1, X_2, ..., X_m$, các hệ số $a_0, a_1, ..., a_m$ của phương trình hồi qui $y = a_0 + a_1x_1 + ... + a_mx_m$. In kết quả vào file mới.

9.5 Cho hàm số *f(x) = 3sin2x*. Sử dụng công thức giải tích và các sơ đồ sai phân với độ chính xác bậc nhất: $f'(x)=(f(x+\Delta x)-f(x))/\Delta x$; độ chính xác bậc hai: $f'(x)= (f(x+\Delta x) - f(x-\Delta x))/2\Delta x$; và độ chính xác bậc bốn: $f'(x)= (4/3) (f(x+\Delta x) - f(x-\Delta x)) / 2\Delta x - (1/3) (f(x+2\Delta x) - f(x-2\Delta x))/4\Delta x$, tính đạo hàm bậc nhất của hàm số trên đoạn $[-\pi/2; \pi/2]$. Lấy $\Delta x = 0.1$ radian. In kết quả vào một file mới dưới dạng (cột cuối cùng là giá trị đạo hàm tính theo công thức giải tích):

HO TEN:..... …

DAO HAM BAC NHAT CUA HAM SO F(X) = 3*SIN(2*X)

X	**DH1_1**	**DH1_2**	**DH1_4**	**ANAL**
...	**...**	**...**	**...**	**...**

9.6 Cho hàm số *f(x) = 2cosx*. Sử dụng công thức giải tích và các sơ đồ sai phân với độ chính xác bậc hai và độ chính xác bậc bốn, tính đạo hàm bậc hai của hàm số trên đoạn $[-\pi; \pi]$. Lấy $\Delta x = 0.1$ radian. In kết quả vào một file mới dưới dạng tương tự như ở bài tập 9.5.

9.7 Cho hàm số $f(x,y)=\sin(x)+\cos(y)+\sin(x+y)$, với $x\in[0 ; 2\pi]$, $y\in [-\pi/2 ; \pi/2]$. Sử dụng công thức giải tích và các sơ đồ sai phân 5 điểm độ chính xác bậc hai và 9 điểm độ chính xác bậc hai, tính Laplaxian của hàm số khi cho *h = 0.1* radian. In kết quả vào một file mới dưới dạng tương tự như ở bài tập 9.5.

9.8 Giả sử quá trình truyền nhiệt xuống các lớp đất sâu được mô tả bởi phương trình $\frac{\partial T}{\partial t} = K\frac{\partial^2 T}{\partial z^2}$, trong đó *T = T(z, t),* với *z* là độ sâu tính từ bề mặt, *t* là thời gian trong ngày tính bằng giây (*s*); *K* là hệ số truyền nhiệt. Hãy viết chương trình tính sự phân bố nhiệt độ theo độ sâu và theo thời gian. Cho biết $z \in [0; 1 (m)]$, $t \in [0; 24 (h)]$; $K = 3\times10^{-7}$. Điều kiện ban đầu: $T(z,0) = \begin{cases} 3z+18.5 & z \le 0.5 \\ 20 & z > 0.5 \end{cases}$; điều kiện biên: $T(0, t) = 3\cos(2\pi/ (24*3600) * t + 2\pi/3) + 20$; $T(1, t) = 20$. In kết quả vào file mới dưới dạng thích hợp.

9.9 Viết chương trình xây dựng một cơ sở dữ liệu lưu trữ hồ sơ cán bộ của một cơ quan.

TÀI LIỆU THAM KHẢO

1. ***Brian D. Hahn***: Fortran 90 for scientists and engeneers. *British Library Cataloguing in Publication Data*, 1996, 352pp
2. ***Elliot B. Koffman, Frank L. Friedman***: Fortran with engineering applications. *Addison–Wesley Publishing Company, Inc.*, 1993, 664pp
3. ***Krishnamutri T. N., L. Bounoua***: An introduction to numerical weather prediction techniques. *CRC Press, Inc.*, 1996, 293 pp
4. ***Phan Văn Tân***: Các phương pháp thống kê trong khí hậu. *NXB Đại học Quốc gia Hà Nội, 2003*, 208 tr.
5. Tuyển tập các chương trình máy tính (*Ứng dụng trong giao thông vận tải*). Tập 1. *NXB Giao thông vận tải, Hà Nội*, 1987.

PHỤ LỤC

1. TRÌNH TỰ CÁC CÂU LỆNH TRONG MỘT ĐƠN VỊ CHƯƠNG TRÌNH FORTRAN

<table>
<tr><td colspan="3">Câu lệnh PROGRAM, FUNCTION, SUBROUTINE hoặc MODUL</td></tr>
<tr><td colspan="3">Câu lệnh USE</td></tr>
<tr><td rowspan="3">Các lệnh định dạng FORMAT</td><td colspan="2">Câu lệnh IMPLICIT NONE</td></tr>
<tr><td>Các câu lệnh PARAMETER và DATA</td><td>Các câu lệnh định nghĩa kiểu dữ liệu, khối giao diện, khai báo biến, hằng và kiểu dữ liệu</td></tr>
<tr><td colspan="2">Các câu lệnh thực hiện</td></tr>
<tr><td colspan="3">Câu lệnh CONTAINS</td></tr>
<tr><td colspan="3">Các chương trình con trong
hoặc các chương trình con modul</td></tr>
<tr><td colspan="3">Câu lệnh END</td></tr>
</table>

2. TÓM TẮT CÁC CÂU LỆNH CỦA FORTRAN

Tên câu lệnh	*Mô tả*
ALLOCATABLE	Chỉ định thuộc tính động cho biến mảng
ALLOCATE	Cấp phát bộ nhớ cho biến mảng động hoặc con trỏ động
BACKSPACE	Đưa con trỏ file lùi về một bản ghi
BLOCK DATA	Chương trình con đặc biệt dùng để khởi tạo dữ liệu
CALL	Lời gọi chương trình con SUBROUTINE
CASE	Chỉ định tập giá trị được chọn trong câu lệnh SELECT CASE
CHARACTER	Lệnh khai báo biến, hằng kiểu ký tự
CLOSE	Lệnh đóng file
COMMON	Lệnh khai báo dùng chung bộ nhớ
COMPLEX	Lệnh khai báo biến, hằng kiểu số phức
CONTAINS	Lệnh phân tách giữa phần thân đơn vị chương trình và khối các chương trình con trong
CONTINUE	Lệnh không thực hiện, thường dùng để kết thúc chu trình hoặc chuyển tiếp giữa các đoạn trong chương trình
CYCLE	Chuyển điều khiển đến câu lệnh kết thúc chu trình (END DO)

Tên câu lệnh	*Mô tả*
DATA	Lệnh khởi tạo dữ liệu cho biến
DEALLOCATE	Giải phóng bộ nhớ cho biến mảng động hoặc con trỏ động
DIMENSION	Chỉ định thuộc tính mảng cho biến, có thể dùng như lệnh khai báo mảng
DO	Lệnh mở đầu cho một chu trình lặp
DO WHILE	Lệnh mở đầu cho một chu trình lặp có điều kiện
DOUBLE PRECISION	Lệnh khai báo biến, hằng thực có độ chính xác gấp đôi
END	Lệnh kết thúc đơn vị chương trình hoặc chương trình con
Tên câu lệnh	*Mô tả*
ENDFILE	Ghi vào file tuần tự bản ghi kết thúc file tại vị trí con trỏ file hiện thời
ENTRY	Khi chèn lệnh này kèm theo tên mới và danh sách đối số của chương trình con vào một vị trí nào đó trong chương trình con, nó có thể làm thay đổi vị trí bắt đầu của chương trình con khi dùng lời gọi với tên mới
EQUIVALENCE	Lệnh khai báo dùng chung bộ nhớ
EXIT	Lệnh thoát khỏi chu trình có điều kiện
EXTERNAL	Khai báo tên của chương trình con ngoài
FORMAT	Khai báo định dạng vào/ra dữ liệu
FUNCTION	Từ khóa khai báo đó là chương trình con dạng hàm
GOTO	Lệnh nhảy vô điều kiện
IF	Lệnh rẽ nhánh
IMPLICIT	Khai báo danh sách các biến, hằng có ký tự ký tự đầu được chỉ ra là những biến, hằng có thuộc tính khai báo ẩn
INCLUDE	Chỉ ra tên file (cả đường dẫn) chứa đoạn chương trình sẽ chèn vào vị trị của lệnh
INQUIRE	Lệnh truy vấn về trạng thái và thuộc tính của file hoặc kích thước bộ nhớ chiếm giữ của biến/bản ghi
INTEGER	Lệnh khai báo biến, hằng có kiểu dữ liệu số nguyên
INTENT	Lệnh khai báo thuộc tính dự định cho các đối số hình thức của chương trình con
INTERFACE	Từ khóa mở đầu khai báo khối giao diện
LOGICAL	Lệnh khai báo kiểu dữ liệu lôgic
MODULE	Từ khóa chỉ đơn vị chương trình là loại modul
NAMELIST	Lệnh khai báo danh sách các khối và biến trong namelist
NULLIFY	Đưa biến con trỏ về trạng thái không trỏ vào đâu cả
Tên câu lệnh	*Mô tả*
OPEN	Lệnh mở file

Tên câu lệnh	*Mô tả*
OPTIONAL	Lệnh chỉ ra các đối số có thuộc tính tùy chọn trong chương trình con
PARAMETER	Khai báo chỉ định thuộc tính hằng
PAUSE	Lệnh tạm dừng chương trình
POINTER	Khai báo chỉ định biến có thuộc tính con trỏ
PRINT	Lệnh kết xuất thông tin ra thiết bị chuẩn (thường là màn hình)
PRIVATE	Khai báo biến, hằng có thuộc tính riêng chỉ trong nội bộ của modul
PROGRAM	Từ khóa chỉ đơn vị chương trình là chương trình chính
PUBLIC	Khai báo biến, hằng có thuộc tính công cộng, có thể truy cập được từ các đơn vị chương trình khác có sử dụng modul
READ	Lệnh đọc dữ liệu vào từ thiết bị
REAL	Lệnh khai báo biến, hằng có kiểu dữ liệu số thực
RECURSIVE	Chỉ định thủ tục đệ qui cho chương trình con
RETURN	Lệnh chuyển điều khiển về chương trình gọi từ chương trình con
REWIND	Đưa con trỏ file trở về đầu file của file tuần tự
SAVE	Khai báo thuộc tính bảo lưu giá trị của các biến trong chương trình con
SELECT CASE	Lệnh chỉ định cấu trúc rẽ nhánh
SEQUENCE	Chỉ định thuộc tính lưu trữ theo trình tự xuất hiện của kiểu dữ liệu do người dùng định nghĩa
STOP	Lệnh dừng hẳn chương trình tại một thời điểm nào đó khi chương trình chưa kết thúc
SUBROUTINE	Từ khóa khai báo đó là một chương trình con dạng thủ tục
TARGET	Chỉ định thuộc tính đích cho biến mà nó là đích của con trỏ
TYPE	Từ khóa định nghĩa kiểu dữ liệu của người dùng tự thiết lập
USE	Từ khóa khai báo tên modul sẽ được sử dụng trong chương trình
WHERE	Câu lệnh thực hiện việc tìm kiếm trong mảng
WRITE	Lệnh kết xuất thông tin ra thiết bị

3. MỘT SỐ HÀM VÀ THỦ THỤC CỦA FORTRAN

Tên hàm, thủ tục	*Chức năng*
ABS(A)	Giá trị tuyệt đối của số nguyên, số thực hoặc số phức A
ACOS(X)	Arccosine (hàm ngược của cosine) của X
AIMAG(Z)	Phần ảo của số phức Z
AINT(A [,KIND])	Phần nguyên (là số thực) lớn nhất không vượt quá A
ANINT(A [,KIND])	Phần nguyên (là số thực) gần nhất của A
ASIN(X)	Arcsine (hàm ngược của sine) của X

Tên hàm, thủ tục	*Chức năng*
ATAN(X)	Arctang (hàm ngược của tang) của X, trong phạm vi $-\pi/2$ đến $\pi/2$
CEILING(A)	Số nguyên nhỏ nhất không nhỏ hơn A
CMPLX(X[,Y][,KIND])	Đổi số X hoặc (X, Y) ra số phức
CONJG(Z)	Liên hợp phức của Z
COS(X)	Cosine của X
COSH(X)	Cosine hyperbol của X
DIM(X, Y)	max(X−Y, 0)
EXP(X)	e^x
FLOOR(A)	Số nguyên lớn nhất không vượt quá A
INT(A [,KIND])	Đổi số A thành số nguyên và chặt cụt phần thập phân
LOG(X)	Lôgarit cơ số tự nhiên của X
LOG10(X)	Lôgarit cơ số 10 của X
MAX(A1,A2[,A3,...])	Giá trị lớn nhất của các số A1, A2, A3,…
MIN(A1,A2[,A3,...])	Giá trị nhỏ nhất của các số A1, A2, A3,…
MOD(A, P)	Số dư của phép chia A cho P, bằng A-INT(A/P)*P
NINT(A [,KIND])	Số nguyên gần nhất với A
REAL(A [,KIND])	Đổi số A thành số thực
SIGN(A, B)	Trị tuyệt đối của A nhân với dấu của B
SIN(A)	Sine của A
SINH(A)	Sine hyberbol của A
SQRT(A)	Căn bậc hai của A
TAN(A)	Tang của A
TANH(A)	Tang hyberbol của A
ACHAR(I)	Ký tự có mã ASCII là I với I trong khoảng 0−127
ADJUSTL(STR)	Trả về xâu STR có cùng độ dài nhưng đã căn lề trái
ADJUSTR(STR)	Trả về xâu STR có cùng độ dài nhưng đã căn lề phải
CHAR(I [,KIND])	Ký tự có vị trí là I của hệ thống sắp xếp thứ tự được cho bởi KIND
IACHAR(C)	Mã ASCII của ký tự C
ICHAR(C)	Vị trí của ký tự C trong hệ thống sắp xếp thứ tự
INDEX(STR, SUBSTR [BACK])	Vị trí bắt gặp đầu tiên của SUBSTR trong STR, tính từ bên trái (nếu BACK=FALSE−ngầm định) hoặc bên phải (nếu BACK=TRUE), bằng 0 nếu không tìm thấy
LEN_TRIM(STR)	Độ dài của xâu STR khi đã cắt bỏ các dấu cách bên phải
LGE(STR_A, STR_B)	Bằng TRUE nếu STR_A tiếp sau STR_B theo thứ tự ASCII hoặc bằng nhau (về mặt từ vựng), bằng FALSE nếu ngược lại
LGT(STR_A, STR_B)	Bằng TRUE nếu STR_A tiếp sau STR_B theo thứ tự ASCII, bằng FALSE nếu ngược lại

Tên hàm, thủ tục	*Chức năng*
LLE(STR_A, STR_B)	Bằng TRUE nếu STR_A đứng trước STR_B theo thứ tự ASCII hoặc bằng nhau (về mặt từ vựng), bằng FALSE nếu ngược lại
LLT(STR_A, STR_B)	Bằng TRUE nếu STR_A đứng trướcc STR_B theo thứ tự ASCII, bằng FALSE nếu ngược lại
LEN(STR)	Số ký tự của STR nếu là biến vô hướng, hoặc số phần tử của STR nếu nó là biến mảng
REPEAT(STR,NCOPIES)	Gộp NCOPIES lần xâu STR
TRIM(STR)	Trả về xâu STR đã cắt bỏ các dấu cách bên phải nhất
EPSILON(X)	Số mà hầu như có thể bỏ qua so với 1 (số vô cùng bé 2^{1-p})
HUGE(X)	Giá trị lớn nhất của biến X có kiểu thực hoặc nguyên
PRECISION(X)	Độ chính xác thập phân (số chữ số thập phân biểu diễn chính xác) của số thực hoặc số phức
TINY(X)	Số dương nhỏ nhất của số thực
BIT_SIZE(I)	Số bit lớn nhất biểu diễn số nguyên
BTEST(I, POS)	Bằng TRUE nếu bít thứ POS của số nguyên I bằng 1 (Chú ý: Số thứ tự bít đánh số từ 0 tính từ bên phải sang của dãy bít biểu diễn số I)
IAND(I, J)	Trả về số nguyên biểu diễn các bít của I và J tương ứng bằng 1, ví dụ IAND(255, 128)=128, vì bít thứ 7 của hai số đều bằng 1, tức 128 = $1.2^7 + 0.2^6 + \ldots 0.2^0$.
ISHFT(I, SHIFT)	Giá trị của I khi dịch chuyển tất cả các bít của I sang trái (SHIFT dương) hoặc sang phải (SHIFT âm) SHIFT vị trí
ALLOCATED(ARRAY)	Nhận giá trị TRUE nếu ARRAY đã được cấp phát bộ nhớ
LBOUND(ARRAY[,DIM])	Trả về chỉ số mảng đầu tiên (nếu bỏ qua DIM) hoặc chỉ số đầu tiên của chiều DIM của ARRAY
SHAPE(SOURCE)	Trả về kích thước các chiều của mảng SOURCE, nếu SOURCE là vô hướng thì kích thước bằng không
SIZE(ARRAY [,DIM])	Trả về kích thước [chiều DIM] của mảng ARRAY
UBOUND(ARRAY[,DIM])	Tương tự như LBOUND nhưng là chỉ số cuối cùng
MAXLOC(ARRAY[,MASK])	Trả về địa chỉ phần tử mảng có giá trị lớn nhất. Nếu có đối số MASK thì MASK là mảng các phần tử lôgic có cùng kích thước với ARRAY; trong trường hợp này chỉ có các phần tử TRUE mới được xét đến.
MERGE(TSOURCE, FSOURCE, MASK)	Trả về mảng có cùng kích thước với cả ba tham số. Các phần tử của mảng kết quả sẽ là những giá trị lấy từ mảng TSOURCE hoặc FSOURCE tùy thuộc phần tử tương ứng của MASK là TRUE hay FALSE.
MINLOC(ARRAY[,MASK])	Tương tự như MAXLOC nhưng là giá trị nhỏ nhất.
TRANSPOSE(MATRIX)	Trả về ma trận chuyển vị của MATRIX
ASSOCIATED(POINTER [,TARGET])	Nếu không có TARGET, kết quả là TRUE nếu POINTER được liên kết với một đích, là FALSE nếu ngược lại.

Tên hàm, thủ tục	*Chức năng*
	Trạng thái POINTER phải là chưa xác định. Nếu có TARGET, kết quả là TRUE nếu POINTER được liên kết với nó. Nếu TARGET cũng chính là con trỏ thì đích của nó được so sánh với đích của POINTER, và sẽ trả về FALSE nếu hoặc POINTER hoặc TARGET chưa được liên kết.
KIND(X)	Trả về giá trị tham số loại dữ liệu của X
SELECTED_INT_KIND(R)	Giá trị tham số loại đối với dữ liệu kiểu số nguyên có thể biểu diễn tất cả các giá trị nguyên trong khoảng $-10^R < n < 10^R$ với R là một số nguyên.
SELECTED_REAL_KIND([P] [,R])	Giá trị tham số loại đối với dữ liệu kiểu số thực có độ chính xác thập phân ít nhất là P, và phạm vi số mũ thập phân ít nhất là R. Ít nhất một trong hai tham số P, R phải xuất hiện.
RANDOM_NUMBER (X)	Thủ tục tạo bộ số ngẫu nhiên ($0 \le X < 1$)
RANDOM_SEED ()	Thủ tục khởi tạo giá trị gốc bộ số ngẫu nhiên của bộ xử lý
DATE_AND_TIME([DATE] [,TIME] [,ZONE] [,VALUES])	Thủ tục trả về các giá trị (là trống rỗng hoặc HUGE(0) nếu không có đồng hồ): – DATE (Character) dạng CCYYMMDD (thế kỷ–ngày) – TIME (Character) dạng HHMMSS.SSS (giờ–mili giây) – ZONE (Character) dạng Shhmm (hiệu giữa giờ địa phương và giờ UTC, S là dấu – VALUES mảng ít nhất 8 phần tử, mà giá trị của chúng tương ứng là Năm, Tháng, Ngày, hiệu thời gian theo phút so với UTC, giờ, phút, giây và mili giây.