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.

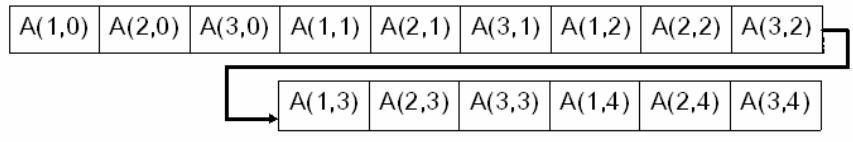
Giả sử ta khai báo

```
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:

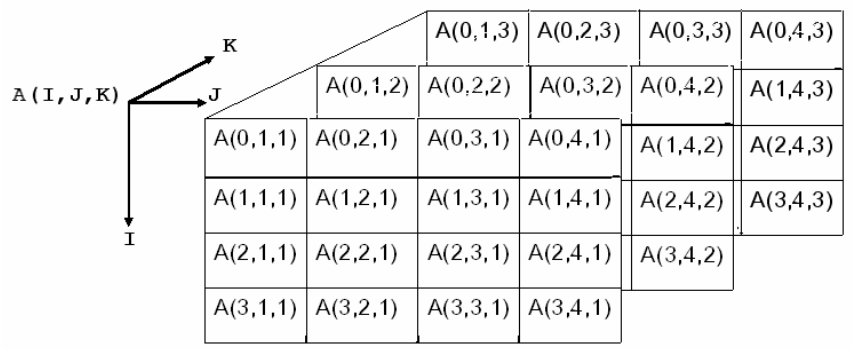


Ví dụ 5.3. Mảng ba chiều.

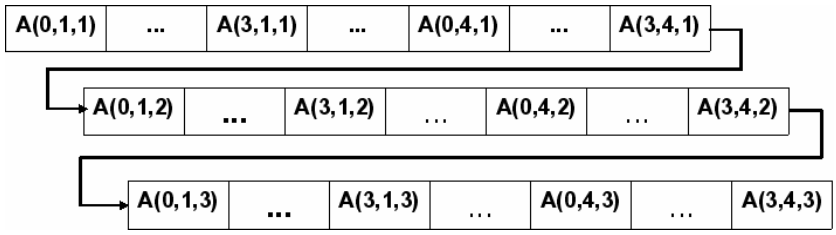
Giả sử mảng A được khai báo bởi

```
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 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
```