ΕΝΑΤΗ ΔΙΑΛΕΞΗ

Γενική μορφή ορισμού συνάρτησης

```
FUNCTION όνομα(παράμετροςΑ, παράμετροςΒ,...)
    IMPLICIT NONE
    τύπος_παραμέτρου_Α, INTENT (xxx) :: παράμετροςΑ
    τύπος_παραμέτρου_Β, INTENT (yyy) :: παράμετροςΒ
    τύπος_επιστρεφόμενης_ποσότητας :: όνομα

    τύπος_Α :: τοπική_μεταβλητή_Α, ...
    τύπος_Β :: τοπική_μεταβλητή_Β, ...

     ...! κώδικας
    όνομα = ...
END FUNCTION όνομα
```

Γενική μορφή ορισμού υπορουτίνας

```
SUBROUTINE όνομα(παράμετροςΑ, παράμετροςΒ,...)
    IMPLICIT NONE
    τύπος_παραμέτρου_Α, INTENT (xxx) :: παράμετροςΑ
    τύπος_παραμέτρου_Β, INTENT (yyy) :: παράμετροςΒ

    τύπος_Α :: τοπική_μεταβλητή_Α, ...
    τύπος_Β :: τοπική_μεταβλητή_Β, ...

     ...! κώδικας
END SUBROUTINE όνομα
```

Όρισμα που είναι διάνυσμα ή πίνακας

Μια παράμετρος σε υποπρόγραμμα μπορεί να είναι διάνυσμα ή πίνακας. Στην παράθεση των παραμέτρων αναφέρουμε μόνο το όνομα και η δήλωση της παραμέτρου γίνεται:

```
FUNCTION όνομα_συνάρτησης(όνομαA, όνομαB, ...)
    IMPLICIT NONE
    τύπος, INTENT (xxx) :: όνομαA(:)
    τύπος, INTENT (xxx) :: όνομαB(:,:)
    ...
END FUNCTION όνομα_συνάρτησης
```

Όρισμα που είναι διάνυσμα ή πίνακας

Μια παράμετρος σε υποπρόγραμμα μπορεί να είναι διάνυσμα ή πίνακας. Στην παράθεση των παραμέτρων αναφέρουμε μόνο το όνομα και η δήλωση της παραμέτρου γίνεται:

```
FUNCTION όνομα_συνάρτησης(όνομαA, όνομαB, ...)
    IMPLICIT NONE
    τύπος, INTENT (xxx) :: όνομαA(:)
    τύπος, INTENT (xxx) :: όνομαB(:,:)
    ...
END FUNCTION όνομα_συνάρτησης
```

Παρατηρήσεις

- Η δήλωση είναι η ίδια, ανεξάρτητα από το πώς δημιουργήσαμε το όρισμα (με γνωστό πλήθος στοιχείων κατά τη μεταγλώττιση ή με `ALLOCATE`).

Όρισμα που είναι διάνυσμα ή πίνακας

Μια παράμετρος σε υποπρόγραμμα μπορεί να είναι διάνυσμα ή πίνακας. Στην παράθεση των παραμέτρων αναφέρουμε μόνο το όνομα και η δήλωση της παραμέτρου γίνεται:

```
FUNCTION όνομα_συνάρτησης(όνομαA, όνομαB, ...)
    IMPLICIT NONE
    τύπος, INTENT (xxx) :: όνομαA(:)
    τύπος, INTENT (xxx) :: όνομαB(:,:)
    ...
END FUNCTION όνομα_συνάρτησης
```

Παρατηρήσεις

- Η δήλωση είναι η ίδια, ανεξάρτητα από το πώς δημιουργήσαμε το όρισμα (με γνωστό πλήθος στοιχείων κατά τη μεταγλώττιση ή με `ALLOCATE`).
- Η αρίθμηση των θέσεων του διανύσματος ή του πίνακα (με τις παραπάνω δηλώσεις) μέσα στο υποπρόγραμμα αρχίζει από το 1, ανεξάρτητα από το πώς αριθμήθηκαν αυτές κατά τη δημιουργία του.

Παράδειγμα (1/2)

Συνάρτηση που μετρά τα άρτια στοιχεία ενός διανύσματος ακεραίων:

```
FUNCTION even(a)
   IMPLICIT NONE
   INTEGER, INTENT (in) :: a(:)
   INTEGER :: even

   INTEGER :: k

   even = 0
   DO k = 1,SIZE(a)
      IF (MOD(a(k), 2) == 0) even = even + 1
   END DO
END FUNCTION even
```

Παράδειγμα (2/2)

Δήλωση

```
INTERFACE
   FUNCTION even(a)
      IMPLICIT NONE
      INTEGER, INTENT (in) :: a(:)
      INTEGER :: even
   END FUNCTION even
END INTERFACE
```

Κλήση

```
INTEGER :: a(100), g
... ! Δίνουμε τιμές στα a(i)
g = even(a)
```

Ειδικές περιπτώσεις (1/3)

Αν επιθυμούμε να αλλάξουμε την αρχή αρίθμησης των θέσεων σε όρισμα που είναι διάνυσμα (ή πίνακας) γράφουμε τη δήλωση

*τύπος, **INTENT** (xxx) :: όνομα(αρχή:)*

Η αρίθμηση των θέσεων του διανύσματος (ή του πίνακα) μέσα στο υποπρόγραμμα αρχίζει από το "αρχή", ανεξάρτητα από το πώς αριθμήθηκαν αυτές κατά τη δημιουργία του.

Παράδειγμα

```
SUBROUTINE f(a,b)
   IMPLICIT NONE
   DOUBLE PRECISION, INTENT (in) :: a(0:)
   INTEGER, INTENT (out) :: b(-10:,20:)
   ...
END SUBROUTINE f
```

Ειδικές περιπτώσεις (2/3)

Αν ένα υποπρόγραμμα πρόκειται να δρα σε διανύσματα συγκεκριμένου, γνωστού πλήθους στοιχείων, έστω n, n δήλωση μπορεί να γίνει

τύπος, ***INTENT*** *(xxx) :: όνομα(n)*

Αντίστοιχα ισχύουν και για πίνακα συγκεκριμένων διαστάσεων.

Παράδειγμα

```
SUBROUTINE f(a,b)
   IMPLICIT NONE
   DOUBLE PRECISION, INTENT (inout) :: a(3)
   DOUBLE PRECISION, INTENT (in) :: b(10,10)
   ...
END SUBROUTINE f
```

Ειδικές περιπτώσεις (3/3)

Μια παράμετρος που είναι διάνυσμα ή πίνακας μπορεί να δηλωθεί ως **ALLOCATABLE** και να δημιουργηθεί σε υποπρόγραμμα. Προφανώς έχει **INTENT (out)**. Η δήλωση του ορίσματος είναι

τύπος, ***INTENT (out), ALLOCATABLE*** *:: όνομαA(:)*
τύπος, ***INTENT (out), ALLOCATABLE*** *:: όνομαB(:,:)*

Παράδειγμα

```
SUBROUTINE f(a)
   IMPLICIT NONE
   INTEGER, INTENT (out), ALLOCATABLE :: a(:,:)
   INTEGER :: n
   INTEGER :: f
   ...
   READ *, n
   ALLOCATE(a(n,n))
   ...
END SUBROUTINE f
```

Το αντίστοιχο **DEALLOCATE** μπορεί να κληθεί από το τμήμα κώδικα που κάλεσε το υποπρόγραμμα δημιουργίας του διανύσματος.

Αναδρομικός αλγόριθμος

Αναδρομή

Σε κάποιο πρόβλημα ο αλγόριθμος επίλυσής του μπορεί να γραφτεί με τέτοιο τρόπο ώστε

- ο υπολογισμός του αποτελέσματος να χρειάζεται την εφαρμογή του ίδιου αλγόριθμου αλλά σε διαφορετικές «τιμές» για τα δεδομένα εισόδου από αυτές που δέχτηκε αρχικά,
- ο αλγόριθμος να μπορεί να υπολογίσει το αποτέλεσμα για ένα συγκεκριμένο σύνολο «τιμών» με άλλο τρόπο και όχι με εφαρμογή του εαυτού του. Το συγκεκριμένο σύνολο πρέπει να μπορεί να το «φτάσει» σε κάποια από τις διαδοχικές εφαρμογές του εαυτού του.

Ο αλγόριθμος με αυτά τα χαρακτηριστικά λέγεται αναδρομικός.

Αναδρομικός αλγόριθμος

Αναδρομή

Σε κάποιο πρόβλημα ο αλγόριθμος επίλυσής του μπορεί να γραφτεί με τέτοιο τρόπο ώστε

- ο υπολογισμός του αποτελέσματος να χρειάζεται την εφαρμογή του ίδιου αλγόριθμου αλλά σε διαφορετικές «τιμές» για τα δεδομένα εισόδου από αυτές που δέχτηκε αρχικά,
- ο αλγόριθμος να μπορεί να υπολογίσει το αποτέλεσμα για ένα συγκεκριμένο σύνολο «τιμών» με άλλο τρόπο και όχι με εφαρμογή του εαυτού του. Το συγκεκριμένο σύνολο πρέπει να μπορεί να το «φτάσει» σε κάποια από τις διαδοχικές εφαρμογές του εαυτού του.

Ο αλγόριθμος με αυτά τα χαρακτηριστικά λέγεται αναδρομικός.

Ένας αναδρομικός αλγόριθμος

- είναι συχνά πιο αποδοτικός (εύκολος, γρήγορος, απλός) από τον ισοδύναμο μη αναδρομικό.
- υλοποιείται με υποπρόγραμμα που πρέπει να καλεί τον εαυτό του.

Στη Fortran 95 τα υποπρογράμματα έχουν τη δυνατότητα να καλούν τον εαυτό τους, αν τροποποιηθούν κατάλληλα.

Παράδειγμα: παραγοντικό

Δύο ισοδύναμοι ορισμοί για το παραγοντικό ακέραιου μη αρνητικού αριθμού:

Μη αναδρομικός

$$n! = \begin{cases} 1 \times 2 \times \cdots \times (n-1) \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

Αναδρομικός

$$n! = \begin{cases} (n-1)! \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

Αναδρομική συνάρτηση παραγοντικού (1/2)

Όχι απόλυτα σωστός κώδικας

Στις εντολές γράφουμε ακριβώς ό,τι μας λέει ο μαθηματικός ορισμός:

$$n! = \begin{cases} (n-1)! \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αναδρομική συνάρτηση παραγοντικού (1/2)

Όχι απόλυτα σωστός κώδικας

Στις εντολές γράφουμε ακριβώς ό,τι μας λέει ο μαθηματικός ορισμός:

$$n! = \begin{cases} (n-1)! \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Ο παραπάνω κώδικας δεν είναι ΑΠΟΛΥΤΑ ΣΩΣΤΟΣ. Χρειάζεται τροποποίηση. Ας δούμε πρώτα πώς λειτουργεί.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με n=2.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).
- Το αποτέλεσμα της `par(0)` είναι `1`. Αυτό επιστρέφεται.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n > 0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).
- Το αποτέλεσμα της `par(0)` είναι `1`. Αυτό επιστρέφεται.

Άρα

*par(2) → par(1)*2 → (par(0)*1)*2 → (1*1)*2 = 2!*

Γενικός ορισμός αναδρομικής συνάρτησης

Στον ορισμό (και τη δήλωση) της συνάρτησης πρέπει δηλώσουμε ότι είναι αναδρομική και να διαχωρίσουμε το όνομα με το οποίο γίνεται η κλήση της συνάρτησης από τη μεταβλητή του ονόματος της συνάρτησης:

```
RECURSIVE FUNCTION όνομα(παράμετροςA, παράμετροςB,...) RESULT (όνομα2)
    IMPLICIT NONE
    τύπος_παραμέτρου_A, INTENT (xxx) :: παράμετροςA
    τύπος_παραμέτρου_B, INTENT (yyy) :: παράμετροςB
    τύπος_επιστρεφόμενης_ποσότητας :: όνομα2

    τύπος_A :: τοπική_μεταβλητή_A, ...
    τύπος_B :: τοπική_μεταβλητή_B, ...

     ...! κώδικας
    όνομα2 = ...όνομα( ..., ..., ...)
END FUNCTION όνομα
```

Αναδρομική συνάρτηση παραγοντικού

Σωστός ορισμός

```
RECURSIVE FUNCTION par(n) RESULT(p)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: p

  IF (n >  0) p = par(n-1) * n
  IF (n == 0) p = 1
END FUNCTION par
```

Ο παραπάνω κώδικας είναι ΣΩΣΤΟΣ.

Γενικός ορισμός αναδρομικής υπορουτίνας

Σε υπορουτίνα που καλεί τον εαυτό της χρειάζεται μόνο η προσθήκη της λέξης **RECURSIVE** πριν το **SUBROUTINE**:

```
RECURSIVE SUBROUTINE όνομα(παράμετροςΑ, παράμετροςΒ,...)
    IMPLICIT NONE
    τύπος_παραμέτρου_Α, INTENT (xxx) :: παράμετροςΑ
    τύπος_παραμέτρου_Β, INTENT (yyy) :: παράμετροςΒ

    τύπος_Α :: τοπική_μεταβλητή_Α, ...
    τύπος_Β :: τοπική_μεταβλητή_Β, ...

     ...! κώδικας
END SUBROUTINE όνομα
```

Παράδειγμα: πολυώνυμα Hermite (1/2)

Ορισμός

Στη Μαθηματική Φυσική χρησιμοποιείται η οικογένεια πολυωνύμων Hermite:

$$\begin{aligned} H_0(x) &= 1 \,, \\ H_1(x) &= 2x \,, \\ H_2(x) &= 4x^2 - 2 \,, \\ H_3(x) &= 8x^3 - 12x \,, \\ H_4(x) &= 16x^4 - 48x^2 + 12 \,, \\ &\vdots \end{aligned}$$

Τα πολυώνυμα $H_n(x)$ ικανοποιούν την αναδρομική σχέση:

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \,, \qquad n \geq 2 \,.$$

Παράδειγμα: πολυώνυμα Hermite (2/2)

Κώδικας

Αναδρομική συνάρτηση που υπολογίζει τα $H_n(x)$:

```
RECURSIVE FUNCTION hermite(n,x) RESULT (h)
  IMPLICIT NONE
  INTEGER,          INTENT (in) :: n
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION              :: h

  IF (n==0) h = 1.0d0
  IF (n==1) h = 2.0d0 * x
  IF (n>=2) h = 2.0d0 * &
    (x * hermite(n-1,x) - (n-1) * hermite(n-2,x))
END FUNCTION hermite
```

Υποπρογράμματα κατά στοιχείο (ELEMENTAL)

- Στις ενσωματωμένες συναρτήσεις το όρισμα μπορεί να είναι μία τιμή αλλά επιτρέπεται να είναι και διάνυσμα:

```
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
y = SQRT(x)
a = 3.2d0
b = SQRT(a)
```

Υποπρογράμματα κατά στοιχείο (ELEMENTAL)

- Στις ενσωματωμένες συναρτήσεις το όρισμα μπορεί να είναι μία τιμή αλλά επιτρέπεται να είναι και διάνυσμα:

```
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
y = SQRT(x)
a = 3.2d0
b = SQRT(a)
```

- Επιθυμούμε το ίδιο να μπορεί να γίνει και σε δικά μας υποπρογράμματα που δέχονται απλές μεταβλητές. Γι' αυτό συμπληρώνουμε τη δήλωση με το **ELEMENTAL** πριν τη λέξη **FUNCTION** ή **SUBROUTINE**.

Υποπρογράμματα κατά στοιχείο (ELEMENTAL)

- Στις ενσωματωμένες συναρτήσεις το όρισμα μπορεί να είναι μία τιμή αλλά επιτρέπεται να είναι και διάνυσμα:

```
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
y = SQRT(x)
a = 3.2d0
b = SQRT(a)
```

- Επιθυμούμε το ίδιο να μπορεί να γίνει και σε δικά μας υποπρογράμματα που δέχονται απλές μεταβλητές. Γι' αυτό συμπληρώνουμε τη δήλωση με το **ELEMENTAL** πριν τη λέξη **FUNCTION** ή **SUBROUTINE**.
- Μια συνάρτηση που είναι **ELEMENTAL** πρέπει να έχει όλα τα ορίσματά της προσδιορισμένα με το **INTENT (in)**. Μία υπορουτίνα μπορεί να είναι **ELEMENTAL** αν στα ορίσματά της έχουμε προσδιορίσει κάποιο **INTENT**, όχι απαραίτητα **INTENT (in)**.

Παράδειγμα υπορουτίνας ELEMENTAL (1/2)

Ορισμός

```
ELEMENTAL SUBROUTINE swap(a,b)
   IMPLICIT NONE
   DOUBLE PRECISION, INTENT (inout) :: a,b

   DOUBLE PRECISION                 :: c

   c = a
   a = b
   b = c
END SUBROUTINE swap
```

Παράδειγμα υπορουτίνας ELEMENTAL (2/2)

Δήλωση και κλήση

```
INTERFACE
   ELEMENTAL SUBROUTINE swap(a,b)
      IMPLICIT NONE
      DOUBLE PRECISION, INTENT (inout) :: a,b
   END SUBROUTINE swap
END INTERFACE
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
y = 1.3d0
a = 3.2d0
b = 5.3d0
CALL swap(x,y)
CALL swap(a,b)
```

Υπορουτίνα παραγωγής τυχαίων αριθμών (1/2)

Η υπορουτίνα **RANDOM_NUMBER** δέχεται μια πραγματική μεταβλητή. Κάθε φορά που καλείται αποδίδει στο όρισμα τυχαία τιμή στο διάστημα $[0, 1)$. Η υπορουτίνα είναι **ELEMENTAL** άρα δέχεται και διάνυσμα πραγματικών στους οποίους αποδίδει τυχαίες τιμές.

Υπορουτίνα παραγωγής τυχαίων αριθμών (1/2)

Η υπορουτίνα **RANDOM_NUMBER** δέχεται μια πραγματική μεταβλητή. Κάθε φορά που καλείται αποδίδει στο όρισμα τυχαία τιμή στο διάστημα $[0, 1)$. Η υπορουτίνα είναι **ELEMENTAL** άρα δέχεται και διάνυσμα πραγματικών στους οποίους αποδίδει τυχαίες τιμές.

Παράδειγμα

```
DOUBLE PRECISION :: x, y, z(100)
CALL RANDOM_NUMBER(x)  ! x τυχαίος
CALL RANDOM_NUMBER(y)  ! y άλλος τυχαίος
CALL RANDOM_NUMBER(z)  ! z άλλοι 100 τυχαίοι
```

Υπορουτίνα παραγωγής τυχαίων αριθμών (2/2)

Αλλαγή διαστήματος

Αν r είναι τυχαίος πραγματικός αριθμός στο $[0, 1)$, τότε κάνουμε τη μετατροπή $x = \kappa r + \lambda$ και επιλέγουμε τους συντελεστές κ, λ ώστε η ποσότητα x να βρίσκεται εντός των επιθυμητών ορίων.
Εύκολα επαληθεύεται ότι ο $x = (b - a)r + a$ ικανοποιεί τις σχέσεις $a \leq x < b$, άρα ο x είναι τυχαίος πραγματικός στο διάστημα $[a, b)$.

ΤΕΛΟΣ