Πανεπιστήμιο Αιγαίου

Εισαγωγή στην Πληροφορική

Ενότητα 10: Πίνακες

Ανδρέας Παπασαλούρος

Τμήμα Μαθηματικών

Σάμος, Ιούνιος 2015

CC BY NC ND

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Ένας πίνακας (array) χρησιμοποιείται για την αναπαράσταση μιας ακολουθίας δεδομένων του ίδιου τύπου. Στην απλούστερη περίπτωση, οι πίνακες εκτείνονται σε μια διάσταση, όπως τα διανύσματα στα Μαθηματικά. Οι πίνακες στη Fortran είναι ορίζονται σε μία ή περισσότερες διαστάσεις.

vector(1)	vector(2)	vector(3)	vector(4)	vector(5)

matrix(1,1)	matrix(1,2)	matrix(1,3)
matrix(2,1)	matrix(2,2)	matrix(2,3)
matrix(3,1)	matrix(3,2)	matrix(3,3)

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.
- Ο *τύπος* των στοιχείων του πίνακα `vector` είναι `real` ενώ ο τύπος του πίνακα `matrix` είναι `integer`.

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.
- Ο *τύπος* των στοιχείων του πίνακα `vector` είναι `real` ενώ ο τύπος του πίνακα `matrix` είναι `integer`.
- Ο πίνακας `vector` έχει *διάσταση* (ή *τάξη*) 1, ενώ ο `matrix` έχει διάσταση 2.

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.
- Ο *τύπος* των στοιχείων του πίνακα `vector` είναι `real` ενώ ο τύπος του πίνακα `matrix` είναι `integer`.
- Ο πίνακας `vector` έχει *διάσταση* (ή *τάξη*) 1, ενώ ο `matrix` έχει διάσταση 2.
- Ο πίνακας `vector` έχει *έκταση* 4, ενώ ο `matrix` έχει *έκταση* 3 στην πρώτη και έκταση 3 επίσης στη δεύτερη διάσταση.

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.
- Ο *τύπος* των στοιχείων του πίνακα `vector` είναι `real` ενώ ο τύπος του πίνακα `matrix` είναι `integer`.
- Ο πίνακας `vector` έχει *διάσταση* (ή *τάξη*) 1, ενώ ο `matrix` έχει διάσταση 2.
- Ο πίνακας `vector` έχει *έκταση* 4, ενώ ο `matrix` έχει *έκταση* 3 στην πρώτη και έκταση 3 επίσης στη δεύτερη διάσταση.
- Δύο πίνακες οι οποίοι έχουν τον ίδιο αριθμό διαστάσεων (τάξη) και την ίδια έκταση σε κάθε διάσταση έχουν το ίδιο *σχήμα*.

- Ο `vector` είναι μονοδιάστατος πίνακας (διάνυσμα) με 5 στοιχεία, ενώ ο `matrix` είναι δισδιάστατος πίνακας.
- Ο *τύπος* των στοιχείων του πίνακα `vector` είναι `real` ενώ ο τύπος του πίνακα `matrix` είναι `integer`.
- Ο πίνακας `vector` έχει *διάσταση* (ή *τάξη*) 1, ενώ ο `matrix` έχει διάσταση 2.
- Ο πίνακας `vector` έχει *έκταση* 4, ενώ ο `matrix` έχει *έκταση* 3 στην πρώτη και έκταση 3 επίσης στη δεύτερη διάσταση.
- Δύο πίνακες οι οποίοι έχουν τον ίδιο αριθμό διαστάσεων (τάξη) και την ίδια έκταση σε κάθε διάσταση έχουν το ίδιο *σχήμα*.
- Ο συνολικός αριθμός των στοιχείων ενός πίνακα καθορίζει το *μέγεθός* του.

Δήλωση πινάκων

Οι πίνακες δηλώνονται όπως οι μεταβλητές, χρησιμοποιώντας τη λέξη `dimension` για τον ορισμό της τάξης και της έκτασης του πίνακα:

```
REAL, DIMENSION(5) :: vector
INTEGER, DIMENSION(3,3) :: matrix
```

Οι παράμετροι στη λέξη `dimension` είναι ακέραιες σταθερές, είτε ανώνυμες, όπως στα παραπάνω παραδείγματα, είτε επώνυμες, όπως στο παράδειγμα που ακολουθεί:

```
INTEGER, PARAMETER :: n = 4
REAL, DIMENSION(n) :: array
```

Οι σταθερές αυτές καθορίζουν την έκταση και τον αριθμό των διαστάσεων του πίνακα. Δεν επιτρέπεται η δήλωση πίνακα με μεταβλητή έκταση:

```
INTEGER :: m
REAL, DIMENSION(m) :: array2 ! Λάθος!
```

Εκφράσεις με πίνακες

Η πρόσβαση στο στοιχείο `i` του πίνακα μιας διάστασης, `vector`, γίνεται μέσω της έκφρασης `vector(i)`. Έτσι οι εντολές

```
vector(1) = 10
vector(2) = 20
```

δίνουν στο *πρώτο* και *δεύτερο* στοιχείο πίνακα του πίνακα `vector` τις τιμές 10 και 20, αντίστοιχα. Επιπλέον, η έκφραση

```
PRINT*, 'Το 2ο στοιχείο του πίνακα είναι ',vector(2)
```

τυπώνει την τιμή του 2ου στοιχείου του παραπάνω πίνακα, `vector`, δηλαδή το 20.

Η πρόσβαση στα στοιχεία πινάκων γίνεται συχνά με χρήση εντολών επανάληψης. Έτσι, το παρακάτω τμήμα κώδικα

```
INTEGER :: i

DO, i = 1, 5
  vector(i) = 2 * i
END DO
```

δίνει στο στοιχείο `i` του πίνακα την τιμή `2*i`. Η μεταβλητή `i` η οποία αναφέρεται στο στοιχείο `vector(i)` του πίνακα `vector` ονομάζεται *αριθμοδείκτης* (subscript). Ένας αριθμοδείκτης, σταθερά ή μεταβλητή, πρέπει να είναι *ακέραιου* τύπου.

Αρχικοποίηση στοιχείων πίνακα

Η απόδοση αρχικών τιμών στα στοιχεία ενός πίνακα είναι δυνατόν να πραγματοποιηθεί, όπως αναφέρθηκε προηγουμένως, με τη χρήση εντολών επανάληψης. Έτσι η απόδοση της τιμής 5 σε όλα τα στοιχεία του δισδιάστατου πίνακα `matrix` είναι δυνατόν να πραγματοποιηθεί ως εξής:

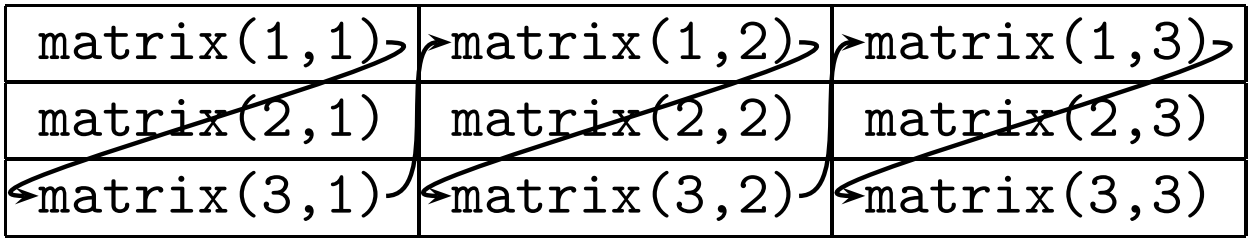
```
DO i = 1, 3
  DO j = 1, 3
    matrix(i,j) = 5
  END DO
END DO
```

Μια τέτοια δομή ονομάζεται *εμφωλευμένη*.
Συνολικά πραγματοποιούνται $3 \times 3 = 9$ επαναλήψεις.

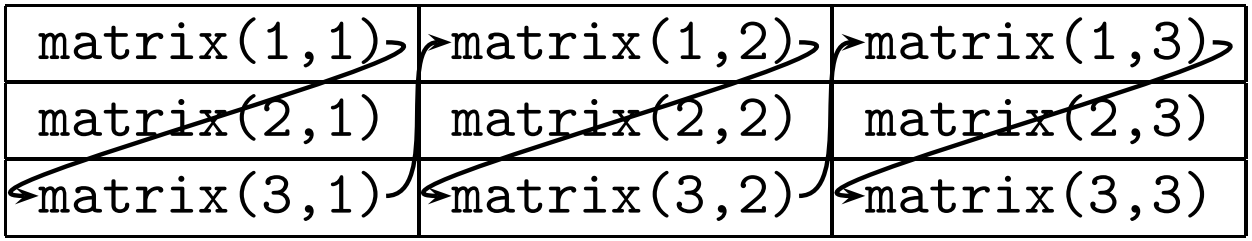
Ας σημειωθεί ότι αυτό το σχήμα της ‘επανάληψης μέσα σε επανάληψη’ είναι ιδιαίτερα συνηθισμένο κατά το χειρισμό πολυδιάστατων πινάκων, αλλά και στον προγραμματισμό γενικότερα.
Εναλλακτικά, η απόδοση αρχικών τιμών είναι δυνατόν να πραγματοποιηθεί χρησιμοποιώντας την εντολή `data` ως εξής:

```
DATA matrix /5, 5, 5, 5, 5, 5, 5, 5, 5/
```

Η απόδοση τιμών στα στοιχεία πίνακα δύο διαστάσεων γίνεται '*κατά στήλες*', δηλ. με την ακόλουθη σειρά:



Η απόδοση τιμών στα στοιχεία πίνακα δύο διαστάσεων γίνεται '*κατά στήλες*', δηλ. με την ακόλουθη σειρά:



Έτσι, για τον παραπάνω πίνακα 3×3 η σειρά απόδοσης τιμών είναι η ακόλουθη: `matrix(1,1)`, `matrix(2,1)`,`matrix(3,1)`,`matrix(1,2)`,..., `matrix(2,3)`,`matrix(3,3)`.

Ανάγνωση και εκτύπωση στοιχείων πίνακα

Η εκτύπωση των στοιχείων ενός πίνακα γίνεται με την εντολή `print`. Όπως και στην περίπτωση της αρχικοποίησης, είναι δυνατή η εκτύπωση των στοιχείων του πίνακα, ενός προς ένα, με χρήση εντολών επανάληψης, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
DO i = 1, 3
  DO j = 1, 3
    PRINT*, 'matrix(',i,',',j,') = ', matrix(i,j)
  end do
end do
```

Η εντολή

```
PRINT*, matrix
```

θα τυπώσει τα στοιχεία του πίνακα ταξινομημένα κατά στήλες, σε μια γραμμή.

Με ανάλογο τρόπο γίνεται η ανάγνωση των στοιχείων του πίνακα:

```
DO i = 1, 3
  DO j = 1, 3
    READ*, 'matrix(',i,',',j,') = ', matrix(i,j)
  end do
end do
```

ή απλά:

```
READ*, matrix
```

Οι τελεστές των τεσσάρων αριθμητικών πράξεων, `+`, `-`, `*`, `/`, είναι δυνατόν να εφαρμοστούν σε πίνακες οι οποίοι έχουν τον ίδιο αριθμό διαστάσεων και την ίδια έκταση σε κάθε διάσταση δηλ. το ίδιο *σχήμα*. Το αποτέλεσμα είναι η εφαρμογή της πράξης *σε κάθε στοιχείο* των δύο πινάκων, όπως φαίνεται στο παράδειγμα που ακολουθεί.

```
REAL, DIMENSION(2,2) :: a,b,c

DATA a /1.0, 2.5, 3.8, 4.0/
DATA b /2.0, 3.0, 4.0, 5.0/

c = a * b
```

Μετά την εκτέλεση των παραπάνω εντολών, κάθε στοιχείο του πίνακα c είναι το γινόμενο των αντίστοιχων στοιχείων των a και b. Προσέξτε ότι η παραπάνω εντολή δεν υπολογίζει το *εσωτερικό γινόμενο* των δύο πινάκων.

Το ίδιο αποτέλεσμα προκύπτει με χρήση εντολών επανάληψης:

```
DO i = 1, 3
  DO j = 1, 3
    c(i,j) = a(i,j) * b(i,j)
  END DO
END DO
```

Είναι δυνατό το πέρασμα πινάκων ως παραμέτρων σε μια συνάρτηση ή διαδικασία. Φυσικά, οι πίνακες είναι δυνατόν να χρησιμοποιηθούν τόσο ως παράμετροι εισόδου όσο και εξόδου. Επιπλέον, μια συνάρτηση είναι δυνατόν να επιστρέφει πίνακα. Η χρήση πινάκων ως παραμέτρων παρουσιάζεται στο παράδειγμα που ακολουθεί.

Παράδειγμα

Να γραφεί συνάρτηση `dotproduct(a,b,m)` η οποία δέχεται ως είσοδο δύο μονοδιάστατους πίνακες (διανύσματα), a και b, μεγέθους m και επιστρέφει το εσωτερικό τους γινόμενο:

$$a * b = \sum_{i=1}^{m} a_i b_i$$

Το εσωτερικό γινόμενο, dotproduct, υπολογίζεται από το ακόλουθο άθροισμα:

```
dotproduct = 0.
DO i = 1, m
  dotproduct = dotproduct + a(i) * b(i)
END DO
```

Η συνάρτηση dotproduct έχει τρεις παραμέτρους εισόδου: τους πίνακες a και b και το μέγεθός τους, m. Ο κώδικας της συνάρτησης δίνεται στη συνέχεια:

```
FUNCTION dotproduct(a,b,m)
  IMPLICIT NONE
  REAL :: dotproduct
  INTEGER :: m
  !Δήλωση των παραμέτων πινάκων
  REAL, DIMENSION(m),INTENT(IN) :: a,b
  INTEGER :: i

  dotproduct = 0.
  DO i = 1, m
    dotproduct = dotproduct + a(i) * b(i)
  END DO
END FUNCTION dotproduct
```

Παράδειγμα

Να γραφεί διαδικασία `multipl` *η οποία δέχεται πώς παραμέτρους έναν πίνακα* a, $n \times k$, *έναν πίνακα* b, $k \times m$ *και έναν πίνακα* c, $n \times m$. *Όλοι οι πίνακες έχουν στοιχεία πραγματικούς αριθμούς διπλής ακρίβειας. Η διαδικασία δέχεται επίσης ως παραμέτρους εισόδου τις εκτάσεις,* $m, n,$ *και* k *και υπολογίζει, ως παράμετρο εξόδου, τα στοιχεία του πίνακα* c *έτσι ώστε ο* c *να είναι το γινόμενο των πινάκων* a *και* b. *Το στοιχείο* c_{ij} *του πίνακα δίνεται από τη σχέση*

$$c_{ij} = \sum_{l=1}^{k} a_{il} b_{lj}$$

για $1 \leq i \leq n$ *και* $1 \leq j \leq m$.

Η διαδικασία (subroutine) έχει ως παραμέτρους εισόδου τους πίνακες a και b και τις διαστάσεις τους και ως παράμετρο εξόδου τον πίνακα c στον οποίο ανατίθεται το αποτέλεσμα του πολ/μού των a και b. Κάθε στοιχείο, c(i,j) του πίνακα c υπολογίζεται, μέσω μιας βοηθητικής μεταβλητής, sum σύμφωνα με το άθροισμα:

```
SUM = 0
DO l = 1, k
  SUM = SUM + a(i,l) * b(l,j)
END DO
c(i,j) = SUM
```

Ο κώδικας της διαδικασίας είναι ο ακόλουθος:

```
SUBROUTINE multipl(a,b,c,n,k,m)
IMPLICIT NONE
  ! Δηλώσεις παραμέτρων
  INTEGER, INTENT(IN) :: n,k,m
  REAL(8), DIMENSION (n,k), INTENT(IN) :: a
  REAL(8), DIMENSION (k,m), INTENT(IN):: b
  REAL(8), DIMENSION (n,m), INTENT(OUT) :: c

  !Δηλώσεις τοπικών μεταβλητών
  REAL(8):: SUM
  INTEGER:: i,j,l
```

Συνέχεια προγράμματος...

```
! Εκτελέσιμες εντολές
  DO i = 1, n
    DO j = 1, m
      SUM = 0
      DO l = 1, k
        SUM = SUM + a(i,l) * b(l,j)
      END DO
      c(i,j) = SUM
    END DO
  END DO
END SUBROUTINE multipl
```

Μια συνάρτηση είναι δυνατόν να επιστρέφει έναν ολόκληρο πίνακα. Σε αυτή την περίπτωση, ο τύπος της επιστρεφόμενης τιμής της συνάρτησης πρέπει να είναι τύπος πίνακα. Έτσι, το παράδειγμα 0.2 μπορεί να αναδιατυπωθεί έτσι ώστε το γινόμενο των δύο πινάκων, a, b να επιστρέφεται ως η τιμή μιας συνάρτησης `multpl2`. Ο κώδικας αυτής της συνάρτησης δίνεται στη συνέχεια. Η συνάρτηση ορίζεται ως *εσωτερική* σε ένα πρόγραμμα με όνομα `matmulti`, το οποίο και καλεί την παραπάνω συνάρτηση.

```
PROGRAM matmulti
  IMPLICIT NONE
  REAL(8), DIMENSION(3,4) :: a
  REAL(8), DIMENSION(4,2) :: b
  REAL(8), DIMENSION(3,2) :: d

  DATA a /1,2,3,4,5,6,7,8,9,10,11,12/
  DATA b /2,3,4,5,6,8,9,10/
  d = multipl2(a,b,3,4,5)
  PRINT*,d

CONTAINS
```

Συνέχεια

```
FUNCTION multipl2(a,b,n,k,m)
    ! Δηλώσεις παραμέτρων
    INTEGER, INTENT(IN) :: n,k,m
    REAL(8), DIMENSION (n,k), INTENT(IN) :: a
    REAL(8), DIMENSION (k,m), INTENT(IN):: b
    REAL(8), DIMENSION (n,m) :: multipl2

    !Δηλώσεις τοπικών μεταβλητών
    REAL(8):: SUM
    INTEGER:: i,j,l
```

Συνέχεια προγράμματος (3)

```
! Εκτελέσιμες εντολές
    DO i = 1, n
      DO j = 1, m
        SUM = 0
        DO l = 1, k
          SUM = SUM + a(i,l) * b(l,j)
        END DO
        multipl2(i,j) = SUM
      END DO
    END DO
  END FUNCTION multipl2

  END PROGRAM matmulti
```

Για την εύρεση του μεγαλύτερου στοιχείου σε έναν πίνακα, a με n στοιχεία κάνουμε τα εξής: Αν ο πίνακας είναι *κενός* , δηλ. αν ο αριθμός των στοιχείων του είναι $n = 0$, τότε δεν ορίζεται μεγαλύτερο στοιχείο για τον πίνακα. Αν ο πίνακας έχει τουλάχιστον ένα στοιχείο, τότε για την εύρεση του μεγαλύτερου στοιχείου, `max`, θεωρούμε αρχικά ως μέγιστο το *πρώτο στοιχείο* του πίνακα.

```
MAX = a(1)
```

Στη συνέχεια επισκεπτόμαστε τα στοιχεία του πίνακα ξεκινώντας από το δεύτερο, καθώς το πρώτο το έχουμε ήδη επισκευθεί. Αν κάποιο από τα στοιχεία του πίνακα, `a(i)`, είναι μεγαλύτερο από την τρέχουσα τιμή του `max`, αντικαθιστούμε το `max` με αυτό το στοιχείο.

Ευρεση μεγαλυτερου στοιχειου σε πινακα

```
DO i = 2, n
  IF (a(i) > MAX) THEN
    MAX = a(i)
  END IF
END DO
```

```
SUBROUTINE maximum(a, n, MAX, pos)
IMPLICIT NONE
  INTEGER, INTENT(IN) :: n ! Μέγεθος του πίνακα
  REAL, DIMENSION(n), INTENT(IN) :: a ! ο πίνακας
  REAL, INTENT(OUT)  :: MAX ! Παράμετροι εξόδου
  INTEGER, INTENT(OUT) :: pos

  INTEGER :: i ! Τοπική μεταβλητή
```

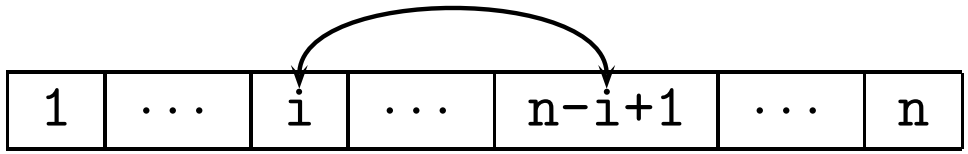
```
    ! Αν ο πίνακας a έχει μη αποδεκτό μέγεθος ...
    IF (n <= 0) THEN
      pos = -1
      RETURN ! Η διαδικασία δεν συνεχίζεται μετά από αυτή
την εντολή
    END IF

    MAX = a(1); pos = 1
    DO i = 2, n
      IF (a(i) > MAX) THEN
        MAX = a(i)
        pos = i
      END IF
    END DO
  END SUBROUTINE maximum
```

```
PROGRAM MaxTest
IMPLICIT NONE
  INTEGER, PARAMETER :: SIZE = 10
  REAL, DIMENSION(SIZE) :: array
  REAL :: maxValue
  INTEGER :: posit

  PRINT *, 'Δώστε δέκα στοιχεία του πίνακα:  '
  READ *, x
  CALL maximum(array, SIZE, maxValue, posit)
  IF (posit > 0) THEN
    PRINT *, 'Το μέγιστο στοιχείο είναι το ', maxValue,&
      'στη θέση ', posit
  ELSE
    PRINT *,'Ο πίνακας είναι κενός '
  END IF
END PROGRAM MaxTest
```

Αντιστροφη των στοιχειων μονοδιαστατου πινακα



Για την αντιστροφή, αντιμεταθέτουμε κάθε στοιχείο στο *πρώτο μισό* του πίνακα με το συμμετρικό του. Όπως έχει αναφερθεί στο κεφάλαιο ;;, η αντιμετάθεση των τιμών δύο μεταβλητών γίνεται με χρήση μιας τρίτης, βοηθητικής μεταβλητής, σύμφωνα με το παρακάτω:

```
DO i = 1, n/2
  temp = a(i)
  a(i) = a(n-i+1)
  a(n-i+1) = temp
END DO
```

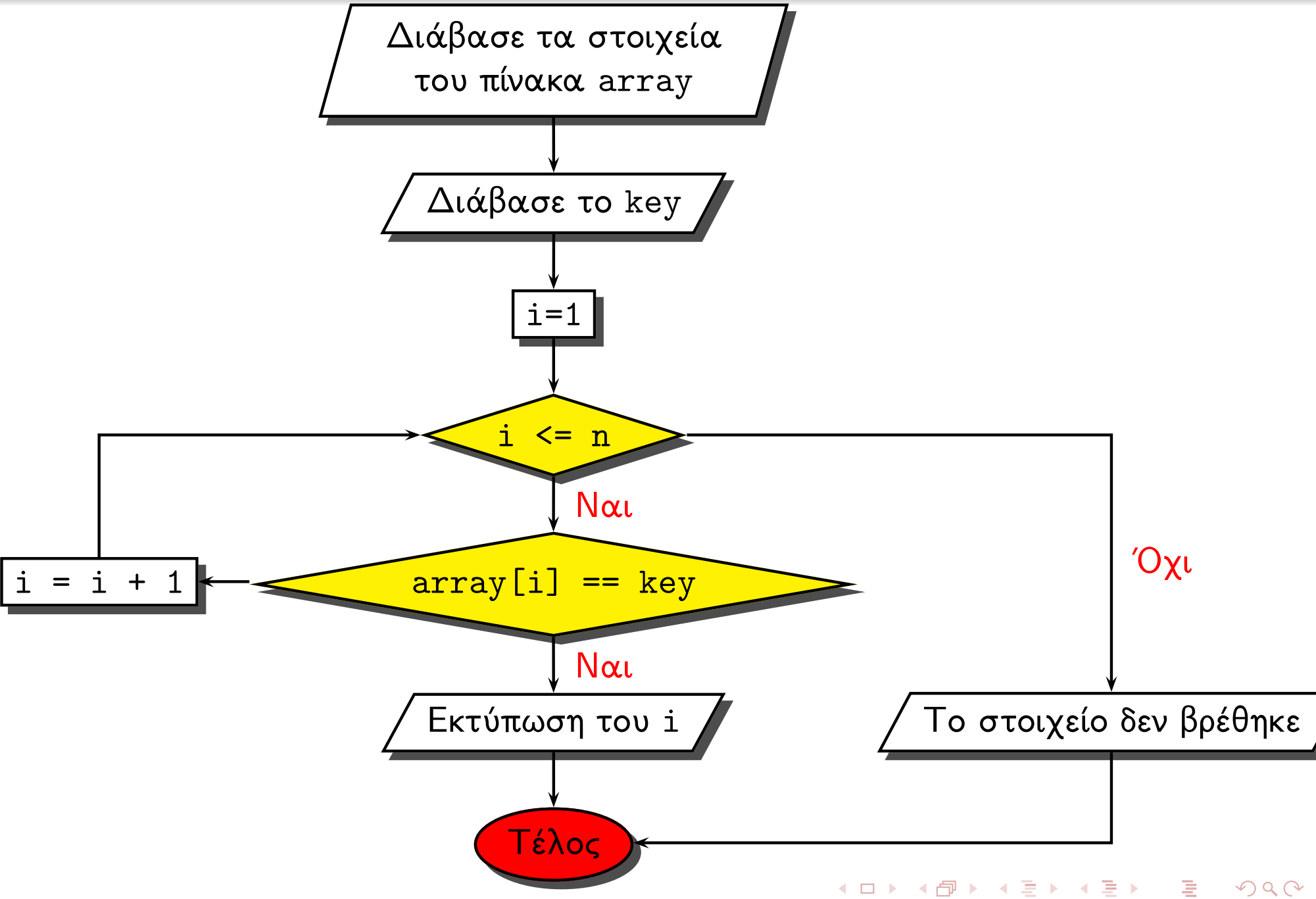
```
SUBROUTINE reverse(a,n)
IMPLICIT NONE
  INTEGER :: n
  REAL, DIMENSION(n), INTENT(INOUT) :: a ! Παράμετρος εισόδ
  REAL :: temp
  INTEGER :: i

  DO i = 1, n/2
    temp = a(i)
    a(i) = a(n-i+1)
    a(n-i+1) = temp
  END DO
END SUBROUTINE reverse
```

Αναζητηση στοιχειου σε πινακα

Ένα σημαντικό πρόβλημα στην Επιστήμη των Υπολογιστών είναι η *αναζήτηση* ενός στοιχείου, το οποίο ονομάζεται *κλειδί* από μια λίστα. Για το πρόβλημα αυτό έχουν προταθεί διάφορες λύσεις, ανάλογα με τον τρόπο διευθέτησης των στοιχείων στη λίστα, για παράδειγμα, το αν είναι ταξινομημένη ή όχι.

Για μια τυχαία λίστα (πίνακα) με μη ταξινομημένα στοιχεία, ο αλγόριθμος για την εύρεση ενός στοιχείου σε αυτή είναι ο εξής: Συγκρίνουμε το στοιχείο το οποίο αναζητάται (*κλειδί*) διαδοχικά με κάθε στοιχείο της λίστας. Αν το κλειδί είναι ίσο με κάποιο στοιχείο στη λίστα, τότε η αναζήτηση σταματά και επιστρέφεται η *θέση* στην οποία βρέθηκε. Αν μετά την επίσκεψη κάθε στοιχείου του πίνακα το κλειδί δεν βρεθεί, επιστρέφεται μια ειδική τιμή που επισημαίνει την αποτυχία της αναζήτησης. Ο αλγόριθμος αυτός ονομάζεται αλγόριθμος *γραμμικής αναζήτησης* και περιγράφεται από το *διάγραμμα ροής* που ακολουθεί:


Διάβασε τα στοιχεία του πίνακα array
Διάβασε το key
i=1
i <= n
Ναι
array[i] == key
i = i + 1
Ναι
Εκτύπωση του i
Όχι
Το στοιχείο δεν βρέθηκε
Τέλος

Ο παραπάνω αλγόριθμος υλοποιείται από το πρόγραμμα που ακολουθεί:

```
PROGRAM SimpleSearch
IMPLICIT NONE

  INTEGER, PARAMETER :: N = 10
  INTEGER, DIMENSION(N) :: array
  INTEGER :: i, key
  LOGICAL :: found

  PRINT *, 'Δώστε 10 στοιχεία:'; READ*, array
  PRINT *, 'Δώστε στοιχείο για αναζήτηση:', READ*, key
```

```
  found = .FALSE.
  DO i= 1, N
    IF (array(i) == key)
      found = .TRUE.
      EXIT
    END IF
  END DO
  IF (found) THEN
    PRINT*, 'Το στοιχείο:', key,'βρέθηκε στη θέση ', i
  ELSE
     PRINT*, 'Το στοιχείο:', key,'δεν βρέθηκε'
  END IF
END PROGRAM SimpleSearch
```

Αλγόριθμος δυαδικής αναζήτησης

Ο αλγόριθμος *γραμμικής* αναζήτησης που παρουσιάστηκε προηγουμένως διασχίζει ολόκληρο τον πίνακα στην περίπτωση που το κλειδί δεν υπάρχει στη λίστα. Επιπλέον, είναι εύκολο να αποδειχθεί ότι στην περίπτωση τυχαία κατανεμημένων στοιχείων στη λίστα, ο αριθμός των βημάτων που απαιτούνται για την εύρεση ενός στοιχείου είναι ανάλογος του μεγέθους του πίνακα. Μικρότερος αριθμός βημάτων απαιτείται όταν ο πίνακας στον οποίο γίνεται η αναζήτηση είναι *ταξινομημένος*, δηλ. αν τα στοιχεία του πίνακα διατάσσονται κατά αύξουσα ή φθίνουσα σειρά.

Στην περίπτωση αυτή είναι δυνατόν να χρησιμοποιηθεί ο αλγόριθμος *δυαδικής αναζήτησης*, ο οποίος παρουσιάζεται σε αυτή την ενότητα. Στο εξής θεωρούμε ότι ο πίνακας στον οποίο πραγματοποιείται η αναζήτηση είναι ταξινομημένος κατά *αύξουσα* σειρά, δηλ. κάθε στοιχείο είναι μεγαλύτερο ή ίσο από το προηγούμενό του.

Η δυαδική αναζήτηση βασίζεται στην εξής τεχνική: Αρχικά

Παρά το γεγονός ότι το παραπάνω σχέδιο αλγορίθμου είναι σχετικά απλό στην περιγραφή και την κατανόηση, οι λεπτομέρειες του αλγορίθμου απαιτούν προσοχή. Η λεπτομερής περιγραφή του αλγορίθμου, σε μορφή ψευδοκώδικα, παρουσιάζεται στον αλγόριθμο 1:

Αλγόριθμος 1 Αλγόριθμος δυαδικής αναζήτησης

Είσοδος: Ένας πίνακας ακεραίων, *array*
Είσοδος: Το μέγεθος, *size*, του πίνακα *array*
Είσοδος: Ένας ακέραιος, *key*, ο οποίος αναζητάται στον πίνακα
Έξοδος: Η θέση πρώτης εμφάνισης του *key* στον πίνακα, ή -1 αν το στοιχείο δεν βρέθηκε

```
Διάβασε τον πίνακα, το μέγεθός του και το key
lower ← 1
upper ← size
όσο lower ≤ upper εκτέλεσε
    m ← (lower+upper)/2
    αν array_m = key τότε
        Βγες από το βρόχο
    αλλιώς αν array_m < key τότε
        lower ← m + 1
    αλλιώς
        upper ← m − 1
    τέλος αν
τέλος όσο
αν array_m = key τότε
    Position ← m
αλλιώς {Το στοιχείο key δεν βρέθηκε}
    Position ← −1
τέλος αν
```

Όπως φαίνεται στον αλγόριθμο 1, οι μεταβλητές *lower* και *upper* παίρνουν ως αρχικές τιμές τα όρια του πίνακα, 1 και *size*, αντίστοιχα. Προσέξτε ότι η $\frac{lower+upper}{2}$ είναι η *ακέραια* διαίρεση του αθροίσματος των ορίων της περιοχής αναζήτησης με το 2 και δίνει την ενδιάμεση θέση στον πίνακα.
Για την κατανόηση του αλγορίθμου, έστω ότι αναζητείται η τιμή 90 στον πίνακα με δεδομένα 10, 20, 30, 50, 60, 90, 91. Τότε, η εκτέλεση του αλγορίθμου δίνει τιμές στις μεταβλητές σύμφωνα με το παρακάτω:

Επανάληψη	m	$lower$	$upper$	$array_m$	key
0	-	1	7	-	90
1	4	5	7	50	90
2	6	5	7	50	90

Μετά το τέλος της δεύτερης επανάληψης, η *medium* έχει πάρει την τιμή 6, όπου βρίσκεται η τιμή που αναζητείται. Με άλλα λόγια, η συνθήκη $array_m = key$ ικανοποιείται και ο βρόχος τερματίζεται.

Έστω τώρα ότι αναζητείται η τιμή $key = 85$, η οποία δεν υπάρχει στον πίνακα. Τότε η εκτέλεση του αλγορίθμου έχει ως εξής:

Επανάληψη	m	$lower$	$upper$	$array_m$	key
0	-	1	7	-	85
1	4	5	7	50	85
2	6	5	5	90	85
3	5	6	5	60	85

Στην παραπάνω περίπτωση, ο αλγόριθμος εκτελείται μέχρι ο αριθμοδείκτης *lower* να γίνει μεγαλύτερος του *upper*. Στην περίπτωση αυτή, το στοιχείο δεν υπάρχει στη λίστα και επιστρέφεται κατάλληλη τιμή (-1).
Είναι φανερό ότι σε κάθε επανάληψη του αλγορίθμου δυαδικής αναζήτησης το μέγεθος της περιοχής στην οποία πραγματοποιείται η αναζήτηση μειώνεται στο μισό. Ο μέσος χρόνος αναζήτησης ενός στοιχείου αποδεικνύεται ότι είναι ανάλογος του $\log(n)$, όπου n το μέγεθος του πίνακα. Είναι φανερό ότι για μεγάλες τιμές του n η δυαδική αναζήτηση είναι κατά πολύ ταχύτερη της γραμμικής, όπου ο χρόνος αναζήτησης είναι ανάλογος του n.

Ένα πρόγραμμα Fortran το οποίο υλοποιεί τον αλγόριθμο δυαδικής αναζήτησης παρατίθεται στη συνέχεια. Προσέξτε ότι ο κώδικας του αλγορίθμου υλοποιείται ως μια *εξωτερική* συνάρτηση, `BinSearch`, η οποία καλείται από το κυρίως πρόγραμμα `BinTest`.

```
PROGRAM BinTest
IMPLICIT NONE
  INTEGER :: BinSearch
  INTEGER, DIMENSION (6) :: anArray

  DATA anArray /1, 2, 3, 4, 5, 7, 12/

  PRINT*, BinSearch(anArray,7,5)

END PROGRAM BinTest
```

```
FUNCTION BinSearch(array,SIZE, key)
IMPLICIT NONE
  INTEGER:: SIZE
  INTEGER, DIMENSION (SIZE) :: array
  INTEGER :: key
  INTEGER ::binary
  INTEGER:: lower, upper, medium

  lower = 1
  upper = SIZE
```

```
  DO WHILE (lower <= upper)
    medium = (lower + upper) / 2
    IF (array(medium) == key) THEN
      EXIT
    ELSE IF (array(medium) < key) THEN
      lower = medium + 1
    ELSE
      upper = medium - 1
    END IF
  END DO

  IF (array(medium) == key) THEN
    BinSearch = medium
  ELSE
    BinSearch = -1
  END IF
END FUNCTION
```

Όρια πινάκων

Τα όρια ενός πίνακα έκτασης n είναι,σύμφωνα με τα προηγούμενα, από 1 έως n. Είναι δυνατή η τροποποίηση αυτών των ορίων. Ο παρακάτω πίνακας array

```
INTEGER, DIMENSION(-2:2):: array
```

έχει 11 στοιχεία, τα array(-2),array(-1), array(0),array(4)array(4).
Ανάλογες δηλώσεις υποστηρίζονται για πίνακες δύο ή περισσότερων διαστάσεων. Η δήλωση

```
INTEGER, DIMENSION(-1:1,0:2) :: m
```

δημιουργεί έναν πίνακα με την ακόλουθη διάταξη στοιχείων:

Αναπαράσταση πολυωνύμων με πίνακες

Ένα πολυώνυμο βαθμού n έχει τη μορφή $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$. Ένα πολυώνυμο καθορίζεται από τους συντελεστές του. Άρα μπορεί να παρασταθεί στον υπολογιστή ως ένας πίνακας πραγματικών, a, μεγέθους n+1 όπου το στοιχείο a(i) του πίνακα αντιστοιχεί στον συντελεστή a_i. Τα όρια του πίνακα είναι 0 το κατώτερο και $n+1$ το ανώτερο. Ένας τέτοιος πίνακας, για ένα πολυώνυμο τρίτου βαθμού, δηλώνεται ως εξής:

```
INTEGER, PARAMETER :: n = 3
REAL, DIMENSION(0:n) :: amatrix
```

Στη συνέχεια δίνονται συναρτήσεις οι οποίες υλοποιούν βασικές πράξεις πολυωνύμων: πρόσθεση, και πολ/μό.

Παράδειγμα

Να γραφεί συνάρτηση `addpoly(a,b,m,n)` *η οποία επιστρέφει το άθροισμα των πολυωνύμων* `a` *και* `b` *βαθμού* `m` *και* `n`*, αντίστοιχα. Κάθε πολυώνυμο παριστάνεται με ένα διάνυσμα όπως παραπάνω.*

Το άθροισμα δύο πολυωνύμων a και b είναι ένα νέο πολυώνυμο βαθμού ίσου με το μεγαλύτερο βαθμό των δύο. Αν υποθέσουμε ότι το πολυώνυμο a έχει τον μικρότερο βαθμό, $(m < n)$, τότε το πολυώνυμο του αθροίσματος, c, έχει βαθμό n με τους εξής συντελεστές:

$$c_i = \begin{cases} a_i + b_i & \text{αν } 0 \leq i \leq m \\ b_i & \text{αν } m < i \leq n \end{cases}$$

Στη γενική περίπτωση, ο συντελεστής i του αθροίσματος είναι ίσος με το άθροισμα των στοιχείων $a_i + b_i$, για $i <= min(m, n)$. Για τα υπόλοιπα στοιχεία το άθροισμα πρέπει να συμπληρωθεί με τα στοιχεία του *μεγαλύτερου* των δύο πινάκων.
Η συνάρτηση που υλοποιεί την πρόσθεση επιστρέφει έναν πίνακα πραγματικών. Ο κώδικας της συνάρτησης είναι ο ακόλουθος:

```
FUNCTION addpoly(a,b,m,n)
  IMPLICIT NONE
  INTEGER :: m,n
  REAL, DIMENSION(0:MAX(m,n)) :: addpoly
  REAL, DIMENSION(0:m), INTENT(IN):: a
  REAL, DIMENSION(0:n), INTENT(OUT) :: b
  INTEGER :: i

  DO i = 0, MIN(m,n)
     addpoly(i) = a(i) + b(i)
  END DO
```

```
  IF (m < n) THEN
     DO i = m + 1, n
        addpoly(i) = b(i)
     END DO
  ELSE ! m >= n
     DO i= n+ 1, m
        addpoly(i) = a(i)
     END DO
  END IF

END FUNCTION addpoly
```

Προσέξτε ότι η συνάρτηση επιστρέφει έναν πίνακα πραγματικών με έκταση από 0 έως max(m+n). Αυτό καθορίζεται με τη δήλωση

```
REAL, DIMENSION(0:MAX(m,n)) :: addpoly
```

Παράδειγμα

Να γραφεί συνάρτηση `multipoly(a,b,m,n)` *η οποία επιστρέφει το γινόμενο των πολυωνύμων* `a` *και* `b` *βαθμού* `m` *και* `n`*, αντίστοιχα.*

Το γινόμενο είναι ένα νέο πολυώνυμο βαθμού $m+n$. Ας πάρουμε το παράδειγμα του γινομένου δύο πολυωνύμων, a, b βαθμού 2. Το γινόμενό τους είναι το παρκάτω πολυώνυμο:

$$a_2b_2x^4+(a_2b_1+a_1b_2)x^3+(a_2b_0+a_1b_1+a_0b_2)x^2+(a_1b_0+a_0b_1)x+a_0b_0$$

Στη γενική περίπτωση, ο συντελεστής βαθμού k, c_k, του γινομένου ισούται με το παρακάτω άθροισμα:

$$c_k = \sum_i \sum_j a_i b_j \text{ για όλα τα} i,j \text{ για τα οποία} i+j=k$$

Ο συντελεστής k μπορεί να υπολογιστεί με πολλούς τρόπους. Ένας έξυπνος και αποδοτικός τρόπος υπολογισμού *όλων* των συντελεστών του γινομένου είναι με χρήση του παρακάτω διπλού βρόχου:

```
DO i = 0, m
  DO j = 0, n
    c(i+j) = c(i+j) + a(i) * b(j)
  END DO
END DO
```

Ο παραπάνω αλγόριθμος λειτουργεί ως εξής: Αρχικά όλα τα στοιχεία του γινομένου, `c`, αρχικοποιούνται στην τιμή 0. Στη συνέχεια, μέσα στον διπλό βρόχο προκύπτει κάθε πιθανός συνδυασμός τιμών των `i` και `j`. Για κάθε συνδυασμό, προστίθεται στο στοιχείο `i+j` το γινόμενο `a(i) * b(j)`. Αυτό έχει ως αποτέλεσμα μετά το τέλος των επαναλήψεων, κάθε στοιχείο `c(k)` έχει ως τιμή το άθροισμα των γινομένων `a(i) * b(j)` για τα οποία $i + j = k$.

```
FUNCTION multipoly(a,b,m,n)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: m,n
  REAL, DIMENSION(m+n) :: multipoly,c
  REAL, DIMENSION(m), INTENT(IN) :: a
  REAL, DIMENSION(n), INTENT(IN) :: b
  INTEGER :: i,j

  DO i = 0, m + n
    c = 0.
  END DO

  DO i = 0, m
    DO j = 0, n
      c(i+j) = c(i+j) + a(i) * b(j)
    END DO
  END DO
  multipoly = c
```

Αλγόριθμοι με πολυδιάστατους πίνακες

Στη συνέχεια παρουσιάζονται περισσότερα παραδείγματα με πίνακες τάξης μεγαλύτερης από ένα.

Παράδειγμα

Δίνεται ένας τετραγωνικός πίνακας $n \times n$. Να γραφεί πρόγραμμα το οποίο τυπώνει κατάλληλο μήνυμα ανάλογα με το αν ο πίνακας είναι συμμετρικός ως προς την κύρια διαγώνιο.

Ένας πίνακας a είναι συμμετρικός ως προς την κύρια διαγώνιο αν για κάθε στοιχείο του είναι ίσο με το συμμετρικό του ως προς την κύρια διαγώνιο. Το συμμετρικό του a_{ij}, είναι το a_{ji}. Άρα πρέπει να ισχύει η εξής συνθήκη:

$$a_{ij} = a_{ji}, \text{ για κάθε } 1 < i, j \leq n$$

Ο αλγόριθμος για τον έλεγχο της παραπάνω συνθήκης αρκεί να περιοριστεί στα στοιχεία *πάνω* ή *κάτω* από την κύρια διαγώνιο. Τα στοιχεία της *κάτω* διαγωνίου είναι αυτά για τα οποία $i > j$. Άρα ο έλεγχος γίνεται μέσα στον ακόλουθο διπλό βρόχο

```
DO i = 1, n
  DO j = 1, i-1
     ...
  END DO
END DO
```

Στο παραπάνω τμήμα κώδικα, η μεταβλητή j παίρνει τιμές μικρότερες του i, οπότε το στοιχείο `a(i,j)` βρίσκεται κάτω από την κύρια διαγώνιο του πίνακα. Κάθε στοιχείο `a(i,j)` συγκρίνεται με το συμμετρικό του, `a(j,i)`. Αν βρεθεί ένα ζευγάρι συμμετρικών στοιχείων τα οποία δεν είναι ίσα, τότε ο έλεγχος *τερματίζεται*.
Η προφανής υλοποίηση του ελέγχου αυτού περιλαμβάνει μια εντολή `exit`:

```
IF (a(i,j) /= a(j,i)) THEN
  EXIT
END IF
```

Όμως μια τέτοια εντολή `exit` θα είχε ως αποτέλεσμα την έξοδο μόνο από τον εσωτερικό βρόχο `do`. Για την έξοδο από τον εξωτερικό βρόχο χρησιμοποιείται μια μορφή της `exit` με χρήση *ετικέτας* (label).
Κάθε εντολή στη Fortran επιτρέπεται να επισημαίνεται με ένα συμβολικό όνομα το οποίο ονομάζεται ετικέτα ως εξής:

```
athroisma: x = a + b
```

Στην παραπάνω εντολή `x = a + b` έχει τεθεί μια ετικέτα με συμβολικό όνομα `athroisma`.

Ετικέτες είναι δυνατόν να τεθούν σε εντολές επανάληψης πριν την αρχή και μετά το τέλος της δομής do, όπως φαίνεται στο παρακάτω παράδειγμα:

```
loop : DO i = 1, n
          ...
          EXIT loop
          ...
       END DO loop
```

Η εκτέλεση μιας εντολής exit ακολουθούμενης από μια ετικέτα έχει ως αποτέλεσμα την έξοδο όχι από τον πιο εσωτερικό βρόχο, αλλά από τον βρόχο με την ετικέτα loop. Έτσι, στο παραπάνω παράδειγμα η εκτέλεση της

```
EXIT loop
```

έχει ως αποτέλεσμα την έξοδο από τον βρόχο με ετικέτα loop.

Επιστρέφοντας στο παράδειγμα, ο βρόχος για τον έλεγχο της συνθήκης συμμετρίας για τον πίνακα είναι ο εξής:

```
check : DO i = 1, n
   DO j = 1, i-1
      IF (a(i,j) /= a(j,i)) THEN
         EXIT check
      END IF
   END DO
END DO check
```

```
PROGRAM symmetric
  IMPLICIT NONE
  INTEGER, PARAMETER :: n = 3
  ! Εναλλακτικός τρόπος δήλωσης πίνακα
  INTEGER :: a(n,n) ! χωρίς τη λέξη DIMENSION
  INTEGER :: i,j
  LOGICAL :: symmetric

  DATA a /1,4,7,&
          4,5,8,&
          7,8,9/
```

```
! Αρχικα θεωρούμε ότι ο πίνακας είναι συμμετρικός
 symmetric = .TRUE.
 check : DO i = 1, n
    DO j = 1, i-1
       IF (a(i,j) /= a(j,i)) THEN
          symmetric = .FALSE.
          EXIT check
       END IF
    END DO
 END DO check
 IF (symmetric) THEN
    PRINT*, "Συμμετρικός"
 ELSE
    PRINT*, "΄Οχι συμμετρικός"
 END IF
END PROGRAM symmetric
```

Ο έλεγχος για γίνεται με χρήση μιας λογικής μεταβλητής `symmetric`. Αρχικά θεωρούμε ότι ο πίνακας είναι συμμετρικός. Αν βρεθούν δύο στοιχεία για τα οποία η συνθήκη συμμετρίας δεν ισχύει, τότε η μεταβλητή `symmetric` παίρνει την τιμή `false` και ο έλεγχος των επόμενων στοιχείων παραλείπεται.