Εισαγωγή στη Fortran

Μάθημα 3ο

Ελευθερία Λιούκα
liouka.eleftheria@gmail.com

Περιεχόμενα

- Loops
- External Functions
- Subroutines
- Arrays
- Common mistakes

Loops

- Ανάγκη να εκτελέσουμε τις ίδιες εντολές πολλές φορές
- Εφαρμόζονται με:
  - *Counter loops*: Για ένα συγκεκριμένο πλήθος επαναλήψεων
  - *Logical loops*: Εως ότου μια λογική συνθήκη γίνει FALSE

Loops

Counter loops

- Εκτελούνται επαναλαμβανόμενα οι εντολές που βρίσκονται μέσα στο βρόχο
- Χρησιμοποιείται μια μεταβλητή ελέγχου η οποία αυξάνεται σε κάθε βήμα με κάποιο ρυθμό
- Μόλις η σταθερά ξεπεράσει το όριο που έχουμε θέσει, η εκτέλεση σταματά

Μορφή:

DO *control-variable = initial value, limit, step-size*
    execution list
END DO

Loops

- *initial value, limit, step-size:* ακέραιοι που υποδεικνύουν το πλήθος των επαναλήψεων που θα εκτελεστούν

  (μπορούν να είναι υπάρχουσες μεταβλητές)

  – Το step-size δεν είναι απαραίτητο

    (default step-size = 1)

Δεν μπορούν να αλλάξουν τιμή μέσα στον βρόχο

Loops

Παράδειγμα1

```
INTEGER :: max = 5
DO n=1, max
    PRINT *, n, n*2
END DO
```

Output1:

```
1   2
2   4
3   6
4   8
5   10
```

Παράδειγμα2

```
DO n = 5, 1, -1
      DO m = 1,2
            PRINT *, n, m
      END DO
END DO
```

Output2:

```
5   1
5   2
4   1
4   2  .....
```

Loops

- Μπορούμε να δώσουμε ονόματα στα loops έτσι ώστε να είναι ξεκάθαρο το που αρχίζει και το που τελειώνει το καθένα

  (ειδικά σε εμφωλευμένα loops)

```
π.χ. outer: DO n = 5, 1, -1
               inner: DO m = 1,2
                        PRINT *, n, m
               END DO inner
     END DO outer
```

Loops

Logical loops

Βρόχοι οι οποίοι τερματίζουν όταν ικανοποιηθεί μια λογική συνθήκη

Ονομάζονται και DO-EXIT

Μορφή:

```
DO
        execution statements
        IF (logical criteria) EXIT
        execution statements
END DO
```

Οι εντολές εκτέλεσης μπορούν να είναι πριν, μετά ή και τα δύο από τη λογική συνθήκη

Loops

- Όταν η λογική συνθήκη γίνει .TRUE τότε οι επαναλήψεις διακόπτονται.
- Προσοχή! Στον ορισμό της λογικής συνθήκης
  Αν δεν γίνει ποτέ .TRUE θα πέσω σε ένα infinite loop

Παράδειγμα

```
n = 1
DO
   IF (n > 10) EXIT
   n = n + 1
END DO
```

Loops

- Μπορούμε να περιμένουμε από τον χρήστη να σταματήσει τις επαναλήψεις

```
DO
   PRINT *, "Enter the radius of your circle."
   READ *, r
   PRINT *, "Area is ", pi*r**2
   PRINT *, "Do you want to calculate another area?"
   READ *, response
   IF (response == "n") EXIT
END DO
```

Loops

- *DO WHILE - END DO expression*

Μορφή:

```
DO WHILE expression
    execution statements
END DO
```

Παράδειγμα

```
n = 1
DO WHILE (n <= 10)
    n = n + 1
END DO
```

Ισοδύναμο:

```
n = 1
DO
    IF (n > 10) EXIT
    n = n + 1
END DO
```

External Functions

- Συναρτήσεις που δεν ορίζει η Fortran
- Αφού οριστεί μια τέτοια συνάρτηση, χρησιμοποιείται κατά τον ίδιο τρόπο όπως οποιαδήποτε συνάρτηση της βιβλιοθήκης της Fortran

Μορφή:

```
PROGRAM name
        ...
        variable = functionname (argument_list1)
         ...
END PROGRAM

type FUNCTION function_name(argument_list2)
        type declarations
        functionname = expression
        RETURN
END FUNCTION
```

External Functions

- Τα *argument_list1, argument_list2 δεν είναι ανάγκη να είναι ίδια, θα πρέπει όμως να περιέχουν ίσο πλήθος και ίδιους τύπους μεταβλητών*
- *Αν τα ονόματα είναι διαφορετικά τότε το argument_list1 αντιστοιχίζεται με τις τιμές των argument_list2*

External Functions

Παράδειγμα

```
PROGRAM testfunc
    IMPLICIT NONE
    REAL :: a, b, c, func
    a = 3.0
    b = 4.0
    c = func(a,b)
END PROGRAM testfunc

REAL FUNCTION func(x,y)
    REAL :: func
    REAL, INTENT (IN) :: x, y
    func = x ** 2 + 2 * y
    RETURN
END FUNCTION func
```

External Functions

- Καλό είναι όσο μπορούμε να αποφεύγουμε περιττές εντολές

z = f(x)/(1.0 + f(x))

Ισοδύναμα:

```
y = f(x)
z = y/(1.0 + y)
```

```
REAL FUNCTION f(x)
      .
      .
      .
```

Subroutines

Παρόμοιες με τις External Functions (πιο εύστροφες)

External Functions	Subroutines
• Παίρνουν ορίσματα και επιστρέφουν έναν αριθμό, ένα string, μια λογική σταθερά ή έναν πίνακα	• Μπορούν να επιστρέψουν πολλά δεδομένα ή καθόλου δεδομένα
• Το όνομα της συνάρτησης ορίζεται να είναι και η επιστρεφόμενη τιμή στο κυρίως πρόγραμμα	• Το όνομα της υπορουτίνας δεν έχει τιμή
• Καλούνται με το όνομά τους μέσα από το κυρίως πρόγραμμα	• Καλούνται μέσω της εντολής CALL

Subroutines

Μορφή:

SUBROUTINE *name(dummy argument-list)*
specification part
execution part
END SUBROUTINE

Η επίσημη *argument-list είναι μια λίστα από identifiers για την είσοδο και την έξοδο στην υπορουτίνα*

Την ονομάζουμε dummy argument-list

Subroutines

- *Κλήση υπορουτίνας:*

  CALL *name(actual argument-list)*
- *actual argument-list*
  - *Μεταβλητές*
  - *Σταθερές*
  - *Εκφράσεις*

  Που χρησιμοποιούνται από την υπορουτίνα
- Οι τιμές αυτές μπορούν είτε απλά να χρησιμοποιηθούν είτε να αλλάξουν τιμή και να επιστραφούν στο αρχικό πρόγραμμα
- Οι τύποι των μεταβλητών των *actual argument-list και dummy argument-list πρέπει να ταιριάζουν*

Subroutines

```
MAIN PROGRAM
    IMPLICIT NONE
    CHARACTER(20) :: file_name
    REAL :: vel, dens, conc
    !  Calling the subroutine.
    ! values for vel, dens, and conc will be returned.
    CALL Openfile(file_name, vel, dens, conc)
END PROGRAM

SUBROUTINE Openfile(name, a, b, c)
    CHARACTER(20), INTENT(IN) :: name
    REAL, INTENT (OUT) :: a, b, c
    INTEGER :: OpenStatus
    OPEN (UNIT = 10, FILE = name, ACTION = "READ", &
                IOSTAT = Openstatus)
    IF (OpenStatus /= 0) STOP “error opening file”
    READ (10,*) a, b, c
END SUBROUTINE
```

One-dimensional Arrays

Μορφή:

type, DIMENSION(*min:max*) :: *array names*

Π.χ.

```
REAL, DIMENSION(1:20) :: A, B
REAL, DIMENSION(20) :: A, B
INTEGER, DIMENSION(-1:3) :: C
```

Ανάθεση τιμών από αρχείο:

```
REAL, DIMENSION(5) :: A
DO I = 1, 5
      READ (10, *) A(I)
END DO
```

Προσοχή! Κάθε τιμή σε ξεχωριστή γραμμή

One-dimensional Arrays

Ανάθεση τιμών από αρχείο:

```
READ (10, *) A
```

Αναφορά σε στοιχείο: A(i)

```
Sum = A(3) + A(5)
```

Π.χ.

```
Sum = 0
DO I = 1,5
      Sum = Sum + A(I)
END DO
```

One-dimensional Arrays

Arrays VS Variables

- Άμεση πρόσβαση
  - REAL :: input_1, input_2, ... input_20
    READ (10, *) input_1, input_2, ... input_20
  - REAL, DIMENSION (20) :: input
    READ (10, *) (input (I), I = 1, 20)
- Περίπτωση Α: Διαβάζει όλες τις προηγούμενες για να φτάσει σε αυτήν που ψάχνω
- Περίπτωση Β: Πηγαίνει κατευθείαν στην τιμή που ψάχνω

One-dimensional Arrays

- Ανάθεση τιμών μέσα στο πρόγραμμα:
  - Εισαγωγή τιμών ανάμεσα από / ... /

```
INTEGER, DIMENSION (5) :: A
A = (/ 2, 4, 6, 8, 10/)
A = (/  (2*I, I = 1,5) /)
A = (/ 2, (2*I, I = 2, 4), 10 /)
```

- Ίδια τιμή σε όλο τον πίνακα:
  - A = 0

One-dimensional Arrays

- Δυνατότητα χρήσης υποσυνόλων πινάκων
  - *array_name(min: max: step)*

```
REAL, DIMENSION (5) :: A
REAL, DIMENSION (3) :: B, C
A = (/ 10, 20, 30, 40, 50/)
B = A(1:5:2)
C = A(2:4)
```

Αποτέλεσμα:

```
B = (/ 10, 30, 50/)
C = (/ 20, 30, 40 /)
```

Arrays

- INTRINSIC FUNCTIONS

DOT_PRODUCT(A, B)	Returns the dot product of A and B
MAXVAL(A)	Returns the maximum value in array A
MINVAL(A)	Returns the minimum value in array A
PRODUCT(A)	Returns the product of all the elements in A
SIZE(A)	Returns the number of elements in A
SUM(A)	Returns the sum of all the elements in A

Multi-dimensional Arrays

Μορφή:

type, DIMENSION(*min1*:*max1*, *min2*:*max2*, ..., *minn*:*maxn*) :: *array_names*

Π.χ.

REAL, DIMENSION(0:4, 3:12, 5) :: A

INTEGER, DIMENSION(3, 4) :: B

Ή

REAL :: A (0:4, 3:12, 5)

INTEGER :: B (3, 4)

Multi-dimensional Arrays

- Ανάθεση τιμών ανά γραμμές:

```
DO row = 1, max_rows
    DO col = 1, max_cols
        READ (10,*) array (row, col)
    END DO
END DO
```

- Ανάθεση τιμών ανά στήλες:

```
DO col = 1, max_cols
    DO row = 1, max_rows
        READ (10,*) array (row, col)
    END DO
END DO
```

Multi-dimensional Arrays

Προσοχή! Η Fortran διαβάζει ανά στήλες

$$A = \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix}$$

```
INTEGER, DIMENSION(3,4) :: array
READ (10,*) array
```

Αποτέλεσμα:

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Multi-dimensional Arrays

Μετατροπή one-dimensional array σε multi-dimensional array

- RESHAPE (*one-dimensionalspecifier*, *shape*, *pad*, *order*)

Π.χ.

A = RESHAPE ((/ 1, 2, 3, 4/), (/2,3/), PAD = (/0/), ORDER = (/2, 1/))

(Old) A = 1 2 3 4

(New) A = 1 2 3
4 0 0

Common mistakes

- Μικτές εκφράσεις που δεν παράγουν τα αποτελέσματα που περιμένουμε
  - a = 4.0
    b = 1/2 + a**2

  Correct: b = 1.0/2.0 + a**2

- Λανθασμένη ενημέρωση μεταβλητών
  - DO
      xnext = 1.08*xcurrent + 0.02*ycurrent
      ynext = ycurrent + 0.14*xcurrent
      znext = 1.01*zcurrent + 0.01* xcurrent**2 + 0.04*ycurrent
      xcurrent = xnext
      ycurrent = ynext
      zcurrent = znext
    END DO

Common mistakes

- Αρχικοποίηση των μεταβλητών πριν τις χρησιμοποιήσουμε
- Όταν καλούμε subroutines προσοχή να δίνουμε ίσο πλήθος και ίδιους τύπους μεταβλητών
  - CALL tempcalc(temperature,distance)

    !Υπόποιπες εντολές

    SUBROUTINE tempcalc(temperature,x,y,z)

Common mistakes

- Αν θέλουμε η subroutine να επιστρέφει κάποια τιμή τότε θα πρέπει να οριστεί η αντίστοιχη μεταβλητή στην subroutine
  - SUBROUTINE tempcalc(x,y,z)

```
IMPLICIT NONE
REAL temperatures, distance, x, y, z
! Commands follow …
temperatures = …
! Commands follow …
RETURN
END
```

Correct: SUBROUTINE tempcalc(temperatures,x,y,z)

Τέλος