Εισαγωγή στη Fortran

Μάθημα 2ο

Ελευθερία Λιούκα
liouka.eleftheria@gmail.com

Περιεχόμενα

- Derived Data Types
- Intrinsic Functions
- Input, Output
- Character Operator
- Branches

Derived Data Types

- Δημιουργία δικών μας τύπων
  - Μορφή:

```
TYPE name
    declaration 1
    declaration 2
            .
            .
    declaration n
END TYPE name
```

- Κάθε declaration ορίζει μια ολόκληρη συνιστώσα που αποτελείται από έναν τύπο και ένα όνομα

Derived Data Types

Παράδειγμα

```
TYPE cource_list
        CHARACTER(15) :: First_name,Last_name
        INTEGER :: Student_ID
        REAL :: Average
        Character(1) :: Grade
END TYPE cource_list
```

Derived Data Types

- Δημιουργία μεταβλητής
  - TYPE (cource_list) cl1,cl2
- Ανάθεση τιμών

  Case 1: cl1 = cource_list(“John”,”Smith”,2378,6.7,8)

  Case 2:

  cl2%First_name = “John”

  cl2%Last_name = “Smith”

  cl2%Student_ID = 2378

  cl2%Average = 6.7

  cl2%Grade = 8

Intrinsic Functions

Μαθηματικές

- cos(x) : Συνημίτονο x
- cosh(x) : Υπερβολικό συνημίτονο x
- acos(x) : Αντίστροφο συνημίτονο x
- sin(x) : Ημίτονο x
- sinh(x) : Υπερβολικό ημίτονο x
- asin(x) : Αντίστροφο ημίτονο x
- tan(x) : Εφαπτομένη x
- tanh(x) : Υπερβολική εφαπτομένη x
- atan(x) : Αντίστροφη εφαπτομένη x
- atan2(x) : Αντίστροφη εφαπτομένη για μιγαδικούς
- sqrt(x) : Τετραγωνική ρίζα x
- exp(x) : Εκθετική συνάρτηση x
- log(x) : Φυσικός λογάριθμος x

Intrinsic Functions

Άλλες συναρτήσεις:

- abs(x) : Απόλυτη τιμή x
- complx(x,y) : Μετατροπή σε μιγαδικό
- floor(x) : Ο μεγαλύτερος ακέραιος, μικρότερος ή ίσος του x

  floor(3.4)=3 floor(-3.4)=-4
- int(x) : Ο μεγαλύτερος ακέραιος που δεν υπερβαίνει το x (χωρίς το πρόσημο)

  int(3.4)=3 int(-3.4)=-3

Intrinsic Functions

- nint(x [,kind]) : Στρογγύλευση στον πλησιέστερο ακέραιο
- real(x [,kind]) : Μετατροπή σε REAL
- mod(a,p) : Υπόλοιπο

$$a - int\left(\frac{a}{p}\right) * p$$

- modulo(a,p) : Υπόλοιπο

$$a - floor\left(\frac{a}{p}\right) * p$$

Input - Output

- Διαθέσιμες Εντολές:
  - OPEN
  - CLOSE
  - READ
  - WRITE
  - PRINT

OPEN

- OPEN(open specifiers)
  - Unit number = int: Αριθμός που δηλώνει ποιο αρχείο ανοίγουμε
    - Θα χρησιμοποιηθεί και στις READ,WRITE
  - Filename = name: Το όνομα του αρχείου που θέλουμε να ανοίξουμε
  - Status = status expression:
    - OLD: αρχείο που υπάρχει ήδη
    - NEW: νέο κενό αρχείο
    - REPLACE: νέο κενό αρχείο που αντικαθιστά το παλιό

OPEN

- Action = action expression: Περιγράφει τις ενέργειες που επιτρέπονται στο αρχείο
  - READ: Μπορούμε να διαβάσουμε αλλά όχι να τροποποιήσουμε
  - WRITE: Μπορούμε μόνο να γράψουμε
  - READWRITE: Μπορούμε να διαβάσουμε και να τροποποιήσουμε
- IOSTAT = var: «δίχτυ προστασίας» πως το αρχείο άνοιξε σωστά
  - var = 0 : άνοιξε σωστά
  - var = error number : δείχνει τον αριθμό του error που παρουσιάστηκε

OPEN

Παράδειγμα

```
OPEN(UNIT=15,FILE="input.txt",STATUS="OLD",&
   ACTION="READWRITE",IOSTAT=Open_status)


IF(Open_status>0) STOP &
   "--- ERROR, File not opened properly ---"
```

CLOSE

- Τα αρχεία κλείνουν στις εντολές END,STOP
- Κλείνουμε τα αρχεία όπου θέλουμε:
  - CLOSE (close_list)

  Π.χ. CLOSE(15)

READ

- Μορφή:
  - READ (control specifiers) input list

  READ(*,*) hour,sec
    - Πρώτο * : unit specifier
    - Δεύτερο *: format specifier

  READ*, hour,sec
    - Όμοια (συντομογραφία)

* : default value

READ

- Control specifiers:
  - Unit specifier:
    - Δηλώνει τη συσκευή εισόδου (default keyboard)
    - Δηλώνει το unit number που αφορά ένα ανοιχτό αρχείο
  - Format specifier:
    - Σταθερά CHARACTER ή μια μεταβλητή CHARACTER των οποίων η τιμή καθορίζει τη μορφή της
    - Ετικέτα ενός format statement

READ

- Για να διαβάσω n μεταβλητές πρέπει να τις έχω δηλώσει προηγουμένως
- Η σειρά που δίνω τις τιμές των μεταβλητών πρέπει να ταιριάζει με τη σειρά εισαγωγής στην READ
- Η READ διαβάζει κάθε φορά από νέα γραμμή
- Για να διαβάσω πολλές μεταβλητές στην ίδια σειρά, πληκτολογώ μι μια αφήνοντας ενδιάμεσα ένα κενό

READ

Παράδειγμα

```
INTEGER :: Age
REAL :: Weight,Height
CHARACTER(10) :: Name

READ(*,*) Name,Age,Weight,Height
```

Input: “John”,27,83.5,182.7

Wrong: 27,182.7,“John”,83.5

READ

- Κάθε READ διαβάζει από νέα γραμμή

```
INTEGER :: I,J,K,L,M,N
READ(*,*) I,J
READ(*,*) K,L,M
READ(*,*) N
```

Input

```
100  200  300  400
500  600  700  800
900  1000 1100 1200
```

WRITE

- Χρησιμοποιείται για να τυπώσει πληροφορίες στην οθόνη ή να γράψει σε αρχείο

Μορφή:

  - WRITE(*,*) exp1,exp2,…
  - WRITE(*,*)

Κάθε φορά γράφει σε νέα γραμμή

WRITE

Παράδειγμα

```
INTEGER :: target
REAL :: angle,distance
CHARACTER(*),PARAMETER :: Time = “The time to hit target ”, Is = “ is ”, Unit = “ sec.”
Target = 10
Angle = 20.0
Distance = 1350.0

WRITE(*,*) ‘Angle = ’,angle
WRITE(*,*) ‘Distance = ’,distance
WRITE(*,*)
WRITE(*,*) Time,target,Is,angle*distance,Unit
```

Output:

```
Angle = 20.0
Distance = 1350.0

The time to hit target 10 is 27000.0 sec.
```

PRINT

Μορφή:

- PRINT format specifier, print list
  - Format specifier: όχι απαραίτητο
  - Χρήση * (default format)
  - Print list : μία ή περισσότερες μεταβλητές χωρισμένες με κόμμα

```
PRINT*, “The number pi = ”, pi
```

Η εντολή PRINT τυπώνει κάθε φορά σε νέα γραμμή

Character Operator //

- Χρησιμοποιείται για να ενώνει strings
- Αν A,B δύο strings με μήκος n,m αντίστοιχα τότε A//B έχει μήκος n+m

Παράδειγμα

```
CHARACTER(4) :: John = "John", Sam = "Sam”
CHARACTER(6) :: Lori = "Lori", Reagan = "Reagan"
CHARACTER(10) :: Ans1, Ans2, Ans3, Ans4

Ans1 = John // Lori          ! Ans1 = “JohnLori ”
Ans2 = Sam // Reagan         ! Ans2 = “Sam Reagan ”
Ans3 = Reagan // Sam         ! Ans3 = “ReaganSam ”
Ans4 = Lori // Sam           ! Ans4 = “Lori  Sam ”
```

Character Operator //

CHARACTER(4) :: John = "John", Sam = "Sam”

CHARACTER(6) :: Lori = "Lori", Reagan = "Reagan"

CHARACTER(10) :: Ans1, Ans2, Ans3, Ans4

Ans1 = John // Lori

Ans2 = Sam // Reagan

Ans3 = Reagan // Sam

Ans4 = Lori // Sam

J	O	H	N	L	O	R	I		
S	A	M		R	E	A	G	A	N
R	E	A	G	A	N	S	A	M	
L	O	R	I			S	A	M	

Branches

- *IF (IF-THEN)*

Μορφή:

IF (*logical-criteria*) execution statement

- Αν logical-criteria = TRUE εκτελείται
- Αν logical-criteria = FALSE παρακάμπτεται

```
IF (2.0 < x .AND x < 3.0) PRINT *, x
```

- Αν έχω παραπάνω εκτελέσιμες εντολές:

```
IF (x  >= 0) THEN
    z = x * y
    PRINT *, "x is a positive number."
END IF
```

Branches

- *IF-ELSE*

Μορφή:

IF (*logical -criteria*) THEN
    execution statements for true result
ELSE
    execution statements for false result
END IF

- Αν logical –criteria = TRUE εκτελείται το πρώτο μέρος και το δεύτερο παρακάμπτεται
- Αν logical –criteria = FALSE εκτελείται το δεύτερο μέρος και το πρώτο παρακάμπτεται

Π.χ.
```
IF (x > 0) THEN
    PRINT *, "The value is greater than zero."
ELSE
    PRINT *, "The value is not greater than zero."
END IF
```

Branches

- *IF-ELSE IF*

Εμφωλευμένα if statements

Μορφή:

IF (*logical -criteria1*) THEN
    execution statements 1
ELSE IF (*logical -criteria2*) THEN
    execution statements 2
ELSE IF (*logical -criteria3*) THEN
    execution statements 3
  .....
ELSE
    execution statements n
END IF

Το τελευταίο ELSE δεν είναι απαραίτητο.

Branches

Παράδειγμα

```
IF (x > 0) THEN
        PRINT *, "Value is greater than zero."
ELSE IF (x < 0) THEN
       PRINT *, "Value is less than zero."
ELSE IF (x == 1) THEN
        PRINT *, "Value is one."
ELSE
        PRINT *, "Value is zero."
END IF
```

Αν το x=1 το output θα είναι Value is greater than zero.

Προσοχή πως φτιάχνουμε τα ELSE IF.

Branches

- *SELECT CASE*

Μορφή:

```
SELECT CASE (selector)
        CASE (list1)
            execution statements 1
        CASE (list2)
            execution statements 2
        ...
         CASE (listn)
           execution statements n
END SELECT
```

Ο selector μπορεί να είναι INTEGER, CHARACTER, LOGICAL

Δεν μπορεί να είναι REAL.

Branches

Παράδειγμα

```
SELECT CASE (INT(grade))  ! The real value grade is converted to an integer.
      CASE (90:)          ! 90: indicates values of 90 or above.
         PRINT *, "Your grade is an A."
      CASE (80:89)        ! 80:89 means 80 to 89.
         PRINT *, "Your grade is a B."
      CASE (70:79)        ! 70:79 means 70 to 79.
         PRINT *, "Your grade is a C."
      CASE (60:69)        ! 60:69 means 60 to 69.
         PRINT *, "Your grade is a D."
      CASE (:59)          ! 59: indicates 59 or below.
         PRINT *, "Your grade is an F."
END SELECT
```

Naming Branches

- Μπορούμε να ονομάσουμε τα branches μας προκειμένου να βελτιωθεί η αναγνωσιμότητα του κώδικα μας

- ```
  positive: IF (x >= 1) THEN
        PRINT *, "The value is greater than or equal to one."
  ELSE
        negative:  IF (x > 0) THEN
           PRINT *, "The value is between zero and one."
        ELSE
           PRINT *, "The value is less than or equal to zero."
        END IF negative
  END IF positive
  ```
  ```

Naming Branches

- name: SELECT CASE (*selector*)
  CASE (*list1*)
  execution statements 1
  CASE (*list2*)
  execution statements 2
  …
  CASE (*listn*)
  execution statements n

  END SELECT *name*

Τέλος