ΕΙΔΙΚΕΣ ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΕΩΝ ΚΑΙ ΕΛΕΓΧΟΥ

Απεριόριστο DO

Το Απεριόριστο DO απαιτεί εξαιρετική προσοχή και εμπειρία στο χειρισμό του. Στο Απεριόριστο DO οι εντολές μεταξύ DO και END DO εκτελούνται επ' άπειρον. Η δομή αυτή επομένως απαιτεί κάποιο τρόπο εξόδου από τον άπειρο αυτό βρόγχο. Για το λόγο αυτό τοποθετείται μια τουλάχιστον εντολή ελέγχου (IF) που να οδηγεί εκτός του βρόγχου (EXIT) όταν εκπληρωθεί κάποια συνθήκη (Λογική Έκφραση). Π.χ.:

```
DO
    b = b + 1 ; WRITE(*,*) b
    IF (b > 100) EXIT
END DO
```

Εκτός από την EXIT άλλες εντολές που τερματίζουν το Απεριόριστο DO είναι οι STOP, RETURN και GOTO.

Μια πολύ πρακτική χρήση του Απεριόριστου DO είναι η ανάγνωση μεγάλων αρχείων αγνώστου μήκους. Στη περίπτωση αυτή διαβάζουμε το αρχείο γραμμή-γραμμή και η εντολή ελέγχου εξετάζει τον κωδικό I/O Status ο οποίος γίνεται αρνητικός όταν συναντάμε το τέλος ενός αρχείου (EOF). Π.χ.:

```
OPEN(10, FILE='data.txt')
DO
   READ(10,*,IOSTAT=kwdikos) a, b, c
   IF (kwdikos < 0) EXIT
   ...
END DO
```

Το Απεριόριστο DO αν δεν συνταχθεί σωστά μπορεί να οδηγήσει μια εργασία σε συνεχή επ' άπειρον λειτουργία χωρίς αποτελέσματα ή ακόμη και τον Η/Υ σε 'κρέμασμα' και διακοπή όλων των εργασιών. Για το λόγο αυτό είναι συχνά ασφαλέστερο να χρησιμοποιούμε ένα Ορισμένο DO με πολύ μεγάλο τελικό όριο ώστε και οι επαναλήψεις να είναι αρκετές και να υπάρχει μια δικλείδα αποφυγής του άπειρου βρόγχου. Η περίπτωση αυτή είναι χαρακτηριστική στους αλγορίθμους της αριθμητικής ανάλυσης. Εκεί το πλήθος των επαναλήψεων δεν είναι γνωστό από τη αρχή αλλά όσες περισσότερες γίνουν τόσο καλύτερη θα είναι η προσέγγιση που θα πετύχουμε. Συχνά όμως οι αλγόριθμοι εγκλωβίζονται σε άπειρους βρόγχους και παλινδρομούν γύρω από τις ίδιες τιμές επ' άπειρον χωρίς να βελτιώνονται τα αποτελέσματα ή να συγκλίνουν. Στις περιπτώσεις αυτές δεν χρησιμοποιούμε το Απεριόριστο DO αλλά ένα Ορισμένο DO που έχει την ίδια συνθήκη ελέγχου και εξόδου με το Απεριόριστο DO και αποτελεί και την κανονική του έξοδο. Αν αντίθετα, το Ορισμένο DO τελειώσει λόγω του δείκτη του τότε σημαίνει ότι ο αλγόριθμός μας δεν ολοκληρώθηκε ή η σύγκλιση δεν είναι η επιθυμητή και ειδοποιείται ο χρήστης ανάλογα. Π.χ.:

```
INTEGER:: i, megisto_orio_epanalipsewn=1000
LOGICAL:: den_synekline=.TRUE.
REAL:: prosegisi, akribeia=0.000001
...
DO i = 1, megisto_orio_epanalipsewn
   ...
   IF (prosegisi <= akribeia) THEN
      den_synekline =.FALSE.
      EXIT
   END IF
   ...
END DO
IF (den_synekline) THEN
   WRITE(*,*) 'Den synekline prin to Megisto Orio Epanal.'
ELSE
   WRITE(*,*) 'Syglisi Epityxis meta apo:',i,' Epanalips.'
ENDIF
...
```

FORALL

Η FORALL είναι ένα τροποποιημένο DO για καλύτερο χειρισμό των αριθμητικών μητρώων. Μια FORALL μπορεί πάντα να αντικατασταθεί από έναν συνδυασμό από DO. Οι δύο παρακάτω δομές είναι ισοδύναμες:

```
FORALL (i=a:t:b, j=a:t:b)
    ...
END FORALL
```

```
DO i = a,t,b
   DO j = a,t,b
      ...
   END DO
END DO
```

Η κρίσιμη διαφορά της FORALL από τη DO είναι ότι οι δείκτες της πρέπει να χρησιμοποιούνται σαν δείκτες στα μητρώα για να είναι δυνατός ο παράλληλος (ταυτόχρονος) υπολογισμός όλων των στοιχείων τους.

EXIT, CYCLE και Ονόματα Βρόγχων DO

Όπως είδαμε και πιο πάνω η εντολή EXIT τερματίζει πρόωρα ένα βρόγχο οδηγώντας την εκτέλεση του προγράμματος από το μέσο του βρόγχου αμέσως μετά το END DO. Η εντολή CYCLE αντίθετα, οδηγεί την εκτέλεση από το μέσο του βρόγχου στην αρχική εντολή DO παραλείποντας όσες εντολές απέμεναν μεταξύ της CYCLE και της END DO.

Σε περιπτώσεις πολλαπλών εσωτερικών DO, οι εντολές CYCLE & EXIT μπορούν να οδηγήσουν κατευθείαν και σε ένα από τα εξωτερικά DO. Για να γίνει όμως αυτό θα πρέπει τα DO να έχουν ονόματα (Labels). Τα ονόματα τοποθετούνται πριν την εντολή DO με μια άνω-κάτω τελεία. Π.χ.:

```
exwter: DO
   mesaio: DO
      eswter: DO
          ...
          IF (b > 100) EXIT exwter
          ...
          IF (a <= 10) CYCLE mesaio
          ...
      END DO eswter
   END DO mesaio
END DO exwter
```

Τα τρία DO εξωτερικό, μεσαίο και εσωτερικό, έχουν ονόματα, έτσι η EXIT τερματίζει τελείως όλη τη δομή, ενώ η CYCLE επιστρέφει στο μεσαίο DO αντί του εσωτερικού.

SELECT CASE

Η SELECT CASE είναι μια πολύ πρακτική παραλλαγή της IF όταν πρόκειται να πραγματοποιήσουμε πολλές ομοειδείς σε μια μεταβλητή. Τα προτερήματά της είναι πρώτον, ότι η δομή της είναι πολύ πιο καθαρή και ευανάγνωστη, και δεύτερον, ότι ο Compiler μπορεί και βελτιστοποιεί τη δομή της καλύτερα από την IF.

Η απαραίτητη προϋπόθεση στην SELECT CASE είναι ότι οι επί μέρους CASE πρέπει να περιέχουν συνθήκες αμοιβαίως αποκλειόμενες δηλ., να μην έχουν κοινό πεδίο αληθείας. Ο λόγος είναι πως ο Η/Υ δεν εκτελεί τις CASE σειριακά όπως την IF, και γι' αυτό αν δυο CASE είναι αληθείς δεν υπάρχει τρόπος να ξέρουμε ποια θα επιλεγεί.

Η μεταβλητή της SELECT CASE μπορεί να είναι INTEGER, CHARACTER ή LOGICAL. Οι τιμές των CASE με τις οποίες συγκρίνεται η μεταβλητή πρέπει να είναι ιδίου τύπου με αυτή. Ηξ κάθε CASE μέσα στη παρένθεση μπορεί να περιέχει:

- (10), ('A') μια τιμή,
- (3, 7, 10), ('Y', 'y') μια λίστα τιμών,
- (5:10), ('A': 'E') διαστήματα τιμών,
- (:10), (20:) ανοιχτά διαστήματα τιμών,
- (3, 7, 10, 12:18) μίγμα με λίστα & διαστήματα τιμών της μεταβλητής.

Π.χ.:

```
INTEGER:: a
...
SELECT CASE (a)
   CASE (20)
      ...
   CASE (30,40)
      ...
   CASE (50:90)
      ...
   CASE (:10)
      ...
```

```
   CASE (91,95,100:900)
      ...
   CASE DEFAULT
      ...
END SELECT
```

Αν η τιμή της μεταβλητής δεν ταιριάζει σε καμία από τις CASE τότε, εφόσον υπάρχει, θα εκτελεστεί η CASE DEFAULT.

WHERE

Η γενική μορφή της WHERE αποτελείται από το τμήμα της WHERE, από το τμήμα της ELSEWHERE, και τελειώνει με τη END WHERE. Όπως και η IF, μια απλή WHERE μπορεί να γραφεί σε μια μόνο γραμμή.

Η Γενική Μορφή της WHERE

```
WHERE (Logiki Ekfrash)
  ...(Εντολές για μητρώα)
ELSEWHERE
  ...(Άλλες εντολές για μητρώα)
END WHERE
```

Η Απλούστερη Μορφή της WHERE

```
...
WHERE (Logiki Ekfrash) Εντολή
...
```

Η Λογική Έκφραση στη WHERE μπορεί να αντικατασταθεί από ένα Λογικό Μητρώο. Το μητρώο αυτό λέγεται μάσκα (mask), είναι ίδιας τάξης με τα άλλα μητρώα της εντολής, επιτρέπει την εφαρμογή της WHERE μόνο για τα στοιχεία για τα οποία έχει λάβει τιμή .TRUE. ή 1. Π.χ.:

```
REAL::    A(9)   =  (/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /)
LOGICAL:: mask(9) = (/ 1, 0, 0, 1, 0, 0, 1, 0, 0 /)
...
WHERE (mask)
    A = 2.* A
ELSEWHERE
    A = A / 2.
END WHERE
...
```

Τα στοιχεία του διανύσματος A θα πολλαπλασιαστούν ή θα διαιρεθούν ανάλογα με τις τιμές των στοιχείων της μάσκας. Σε όσες θέσεις η μάσκα είναι 1 (ή .TRUE.), εκτελείται το WHERE, στις υπόλοιπες θέσεις όπου η μάσκα είναι 0 (ή .FALSE.) εκτελείται το ELSEWHERE. Το αποτέλεσμα της παραπάνω εντολής WHERE θα είναι το διάνυσμα:

A = [2.0 1.0 1.5 8.0 2.5 3.0 14.0 4.0 4.5]