

- **Εντολές ελέγχου και λογικής**
- **Εντολές μεταφοράς**
- **Βρόγχοι επανάληψης εντολών**
- **Βρόγχοι επανάληψης με λογικές σχέσεις**

Εντολές Ελέγχου και Λογικής

□ Τα assignment statements είναι αρκετά απλά.

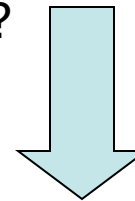
- Μεταφορά πληροφορίας από μια πράξη/σταθερά σε μεταβλητή που αντιστοιχεί σε θέση μνήμης του υπολογιστή
- Απλή δομή και η σειρά εκτέλεσης των εντολών ήταν προκαθορισμένη, μοναδική και ανεξάρτητη από τα μερικά αποτελέσματα

program trapezoid

```
c*****
c Calculate the height and area of a trapezoid
c Input the sides of the trapezoid A,B,C,D
c Output the height H and area A
c*****
  REAL  A, B, C, D
  REAL  HEIGHT, AREA
  REAL  S, X
  READ *, A, B, C, D
c Calculate the height
  S = (A - B + C + D)/2.
  X = S*(S - A + B) * (S - D) * (S - C)
  HEIGHT = (2./(A-B))*SQRT(x)
c Calculate the Area
  AREA = ((A+B)/2.) * HEIGHT
  PRINT *, HEIGHT, AREA
  END
```

□ Στην πράξη όμως ακόμη και τα πιο απλά προβλήματα απαιτούν μεγαλύτερη ευελιξία

□ Πως όμως αντιμετωπίζουμε προβλήματα όπως ο υπολογισμός της τετραγωνικής ρίζας ενός αρνητικού αριθμού χωρίς να σταματά η ροή εκτέλεσης του προγράμματος ?



Εντολές ελέγχου και λογικής
control statements

Εντολές ελέγχου – Control Statements

- Επιτρέπουν την παράλειψη ενός πλήθους εντολών (πηγαίνουμε από μια εντολή E1 σε μια άλλη E2 όχι επόμενη της πρώτης)
- Επιτρέπουν συγκρίσεις
Ανάλογα με το αποτέλεσμα, εκτελείται μια σειρά πράξεων ή κάποια άλλη πράξη
Ο Η/Υ αποκτά την δομή ανθρώπινης σκέψης (μέσω ενός προγράμματος)

- ❑ Οι κυριότερες εντολές ελέγχου και λογικής στην FORTRAN είναι:

Βασικό IF

IF ... THEN ... ELSEIF ... ELSE ...

DO

GO TO

CONTINUE, PAUSE, STOP, END

- ❑ Η μεταφορά του ελέγχου γίνεται με μια εντολή E1 στην οποία καθορίζουμε την εντολή E2 που πρέπει να εκτελέσει αμέσως μετά ο Η/Υ

➤ Η Εντολή E2 πρέπει να είναι εκτελέσιμη

Αν υπάρχουν υποπρογράμματα (επόμενο μάθημα) μέσα στο κύριο πρόγραμμα, οι E1 και E2 πρέπει να ανήκουν στο ίδιο πρόγραμμα ή υποπρόγραμμα

Βασικό IF

- Δίνει τη δυνατότητα σ' ένα πρόγραμμα να αποφασίζει ποια εντολή θα εκτελέσει στο επόμενο βήμα ανάλογα με την τιμή κάποιας μεταβλητής
- Η εντολή IF αυξάνει τις δυνατότητες ενός προγράμματος

PROGRAM IFIF

```

c=====
c THE BASIC IF STATEMENT
c =====
    INTEGER N
    PRINT *, 'Give me an integer'
    READ *, N
    IF (N.GT.0) THEN
        PRINT *, 'The number is positive'
    ENDIF
    IF (N.LT.0) THEN
        PRINT *, 'The number is negative'
    ENDIF
    STOP
    END
  
```

Η σύνταξη της εντολής είναι:

```

IF (logical expression) THEN
    block
ENDIF
  
```

Η λογική έκφραση στην παρένθεση είναι κάτι που μπορεί να ναι αληθές ή ψευδές

- Αν είναι **ψευδής** το πρόγραμμα συνεχίζει μετά την εντολή END IF
- Αν είναι **αληθής** τότε εκτελείται το group των εντολών μετά το IF.

Συνηθισμένη μορφή λογικής σχέσης είναι:

(Αριθ.σχέση) **Λογικός Τελεστής** (Αριθ. Σχέση)

π.χ. (A **.GT.** 0)

Βασικό IF (συνέχεια)

□ Οι τελεστές σχέσης που χρησιμοποιούνται είναι:

.GT. Greater Than ($>$)

.LT. Less Than ($<$)

.EQ. Equal ($=$)

.GE. Greater Equal (\geq)

.LE. Less Equal (\leq)

.NE. Not Equal (\neq)

Προσέξτε τις τελείες (.) που υπάρχουν πριν και μετά κάθε τελεστή σχέσης

Προσοχή: Τα σύμβολα δεν αναπαράστουν τους λογικούς τελεστές αλλά οι compilers τα θεωρούν σωστά

□ Οι λογικές σχέσεις συνδέονται ακόμα και με τους λογικούς τελεστές:

.AND. Λογικό AND

.EQV. (equivalent)

.OR. Λογικό OR

.NEQV. (not equivalent)

➤ Όταν χρησιμοποιούνται οι λογικοί τελεστές θα πρέπει κάθε λογική σχέση να βρίσκεται σε παρένθεση και το ίδιο η ολική λογική σχέση:

π.χ. IF ((A.GT.B) .AND. (B.LT.C)) THEN
 •
 •
 •
 ENDIF

Βασικό IF – Παραδείγματα (συνέχεια)

```
IF ( (A.GT.B) .OR. (B.EQ.C) ) THEN
```

```
  ⋮
  ⋮ } Block εντολών
  ⋮
```

Η ροή συνεχίζει στην εντολή μέσα στο
IF block αν είτε το $A > B$ ή το $B = C$

```
ENDIF
```

- ❑ Λογικές εκφράσεις μπορούν να αποκτήσουν αντίθετη σημασία αν προτάξουμε το **.NOT.**

(**.NOT.** (A .GT. B)) Η σχέση είναι ισοδύναμη με: $A < B$

- ❑ Δεν είναι απαραίτητο να υπάρχει κάποιο block εντολών μετά την εντολή IF
- ❑ **Καλή πρακτική:** Οι εντολές που περιέχονται μέσα σε ένα IF να γράφονται σε μετατοπισμένη στήλη προς τα δεξιά σε σχέση με την στήλη της αρχής του IF για εύκολη ανάγνωση του κώδικα δείχνοντας ότι το τμήμα αυτό του προγράμματος είναι ξεχωριστό.

```
IF (A .GT. B) THEN
```

```
  K = K + 1
```

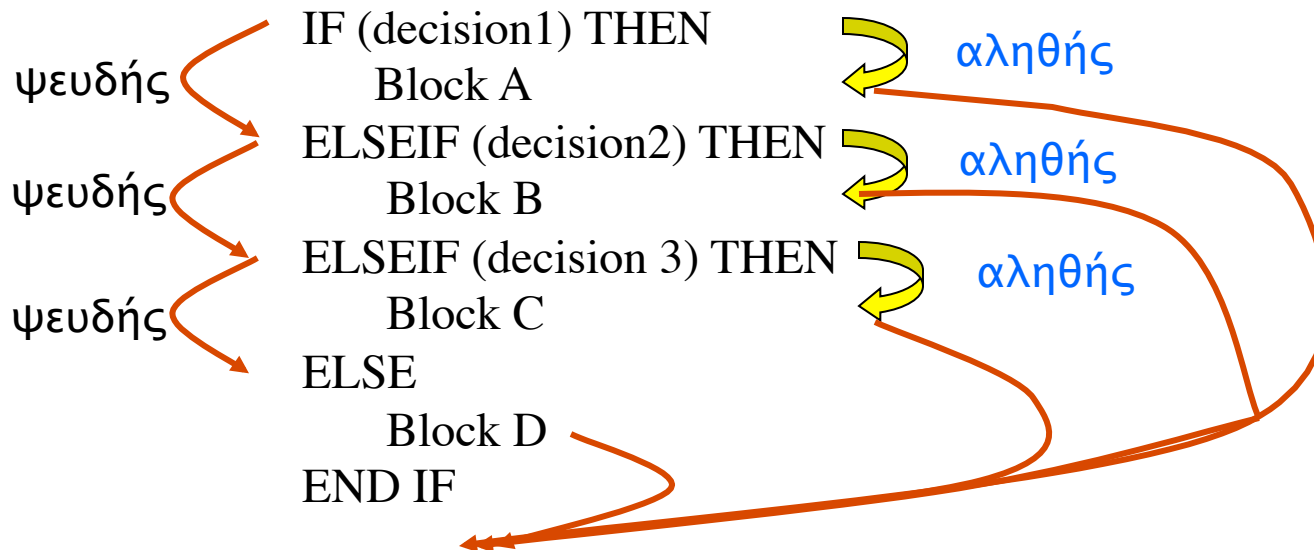
```
  L = L * K
```

```
ENDIF
```

IF...THEN ... ELSEIF ... THEN ... ELSE ...

□ Αυτή η μορφή χρησιμοποιείται για πολλαπλές αποφάσεις.

Αν καμιά από τις αποφάσεις δεν ικανοποιείται, τότε εκτελείται το block D των εντολών (μετά την εντολή ELSE).



IF χωρίς THEN και END IF

- Αν το block των εντολών που θέλουμε να εκτελέσουμε μετά ένα επιτυχημένο IF, αποτελείται μόνο από μια εντολή και δεν υπάρχουν άλλες αποφάσεις να παρθούν τότε μπορούμε να παραλείψουμε το THEN όπως επίσης και το ENDIF

```
IF (A .GT. B) PRINT *, "is greater than B" '
```

```
IF (NUM .LT. MIN) MIN= NUM
```

- Δεν μπορούμε να κάνουμε κάτι τέτοιο με THEN και ELSEIF δομές αποφάσεων.

Εντολή μεταφοράς GOTO

□ Απλή εντολή μεταφοράς από ένα σημείο του προγράμματος σε άλλο

Η σύνταξη είναι: **GOTO** (ή GO TO) E_N

όπου E_N η εντολή E με αριθμό N, την οποία ο υπολογιστής υποχρεούται να εκτελέσει αμέσως μετά.

- Δεν υπάρχει καμία προϋπόθεση για την μεταφορά αυτή και γι' αυτό η εντολή αυτή καλείται και unconditional GOTO

Ο αριθμός εντολής (π.χ. 10) μπαίνει στις πρώτες 5 στήλες της γραμμής της εντολής

```

PROGRAM TEST
REAL A, B, X
N = 0
10 READ *, A, B
   X = B**A + 5.0
   PRINT *, A,B,X
   N = N + 1
   IF (N.LT.100) GOTO 10
END
  
```



Προσοχή: Η εντολή αυτή παρ' όλη την απλότητά της πρέπει να αποφεύγεται όσο το δυνατό περισσότερο.

Ο λόγος; Οδηγεί πάρα πολλές φορές σε λάθη μια και φθάνοντας σε μια εντολή η οποία μπορεί να καλείται από διάφορα σημεία του προγράμματος με ανάλογα GOTOs δεν υπάρχει πληροφόρηση από ποιο σημείο του προγράμματος έγινε η μεταφορά

Παράδειγμα – IF και GOTO

- Να προσδιοριστούν όλες οι τριάδες (x,y,z) ακεραίων αριθμών από το 1 έως 9, για τις οποίες το άθροισμα των τετραγώνων των αριθμών, K , είναι πολλαπλάσιο του ακεραίου L .

Λογική Λύση:

Έστω ο ακεραίος K ακέραιο πολ/σιο του ακεραίου $L \Rightarrow \frac{K}{L} \times L - K = 0$

Αντίστροφα αν: $\frac{K}{L} \times L - K = 0$ τότε ο ακεραίος K είναι ακέραιο πολ/σιο του L

Μηδενισμός της $\left[\frac{K}{L} \times L - K \right]$ είναι αναγκαία και ικανή συνθήκη ώστε ο K ακέραιο πολ/σιο του L

Αναλυτική Λύση:

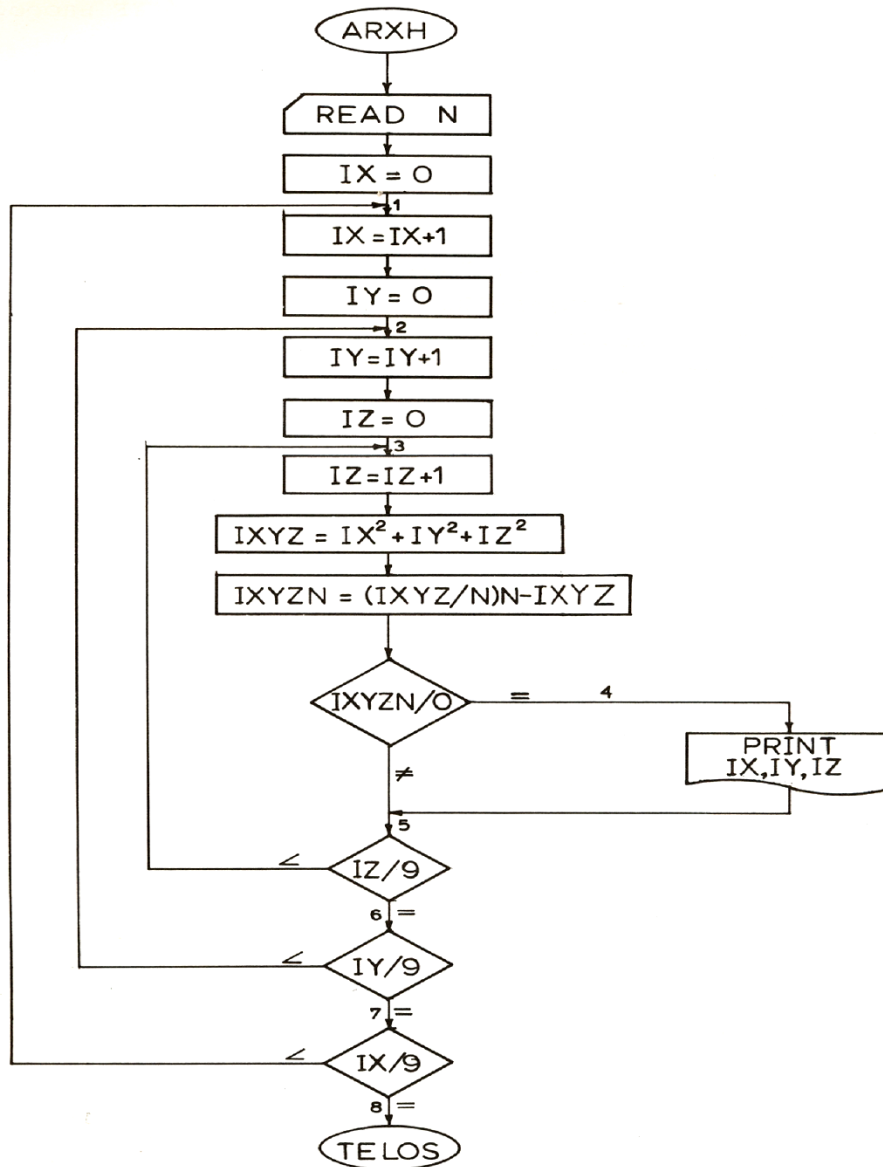
Το πρόγραμμα θα σχηματίζει για κάθε τριάδα (x,y,z) , το άθροισμα $K=x^2+y^2+z^2$ και θα εξετάζει αν κάθε άθροισμα K είναι πολ/σιο του L (**λογική λύση**)

Για ένα ζεύγος αριθμών (x, y) ο z παίρνει όλες τις τιμές από 1 ως 9 και αν $x^2+y^2+z^2$ είναι πολ/σιο του L , τυπώνεται η τριάδα x, y, z .

Αυξάνουμε το y κατά 1, (νέο ζεύγος x,y) και επαναλαμβάνουμε τη διαδικασία

Όταν εξαντληθούν οι δυνατές τιμές του y τότε αυξάνουμε το x κατά 1 και επαναλαμβάνεται η διαδικασία

Παράδειγμα



PROGRAM TRIPLET

INTEGER IX, IY, IZ

INTEGER N, IXYZ, IXYZN

READ *, L

c Initialization for IX

IX = 0

1 IX = IX + 1

c Initialization for IY

IY = 0

2 IY = IY + 1

c Initialization for IZ

IZ = 0

3 IZ = IZ + 1

c Calculate the sum of the squares

$IXYZ = IX**2 + IY**2 + IZ**2$

c Calculate the remainder

$IXYZN = (IXYZ/L)*L - IXYZ$

c Check to see if it is zero

IF (IXYZN.EQ.0) THEN

PRINT *,IX,IY,IZ

ENDIF

IF ((IZ - 9) .LT. 0) GOTO **3**

IF ((IY - 9) .LT. 0) GOTO **2**

IF ((IX - 9) .LT. 0) GOTO **1**

END

Επανάληψη εντολής – Βρόγχοι - Iterations – **DO Loop**

- ❑ Μια εντολή DO loop επιτρέπει την εκτέλεση ενός τμήματος του προγράμματος ένα καθορισμένο αριθμό επαναλήψεων.
Στα DO loops το τμήμα που θέλουμε να επαναεκτελεσθεί, ξεκινά με την εντολή DO και τελειώνει με την αριθμημένη εντολή CONTINUE.
Το τμήμα του προγράμματος που επανεκτελείται καλείται το σώμα (body) του DO loop, και ο αριθμός των επαναλήψεων καθορίζεται από την αρχική και τελική τιμή ενός μετρητή καθώς επίσης και από το πόσο αυξάνεται ο μετρητής κάθε φορά που εκτελείται το σώμα του DO loop.

Σύνταξη της εντολής DO

Η σύνταξη της εντολής DO είναι η ακόλουθη:

Ισοδύναμη σύνταξη

```
DO N IX = ISTART, ISTOP, ISTEP  
    Κύριο σώμα του loop (εντολές)  
N CONTINUE
```



```
DO IX = ISTART, ISTOP, ISTEP  
    κύριο σώμα του loop  
ENDDO
```

Όπου:

- N** Αριθμός εντολής που ουσιαστικά οριοθετεί την περιοχή του Loop. Μπορεί να είναι ένας οποιοσδήποτε αριθμός μικρότερος από 99999
- IX** ονομάζεται δείκτης του Loop (loop index) και ουσιαστικά παίζει το **ρόλο του μετρητή**.
Η τιμή του αυξάνει κάθε φορά που ολοκληρώνεται μια εκτέλεση του Loop (περνά από την εντολή CONTINUE) **κατά την ποσότητα ISTEP**.
Η αρχική τιμή του IX καθορίζεται από την τιμή του **ISTART**, ενώ ο αριθμός των επαναλήψεων καθορίζεται από την τιμή **ISTOP**.
Η επανάληψη σταματά όταν η τιμή του IX ξεπεράσει την τιμή του **ISTOP**.

Όλες οι μεταβλητές είναι ακέραιοι.

Do Loop - Παραδείγματα

<pre> DO 12 J=1, 10 PRINT *, ' J=', J 12 CONTINUE </pre>		<p>Όταν η τιμή ISTEP παραλείπεται θεωρείται ότι είναι ίση με 1.</p>
<pre> DO 20 I=10,0,-1 PRINT *, I 20 CONTINUE </pre>		<p>Στο loop αυτό η αρχική τιμή του μετρητή I είναι 10 και κάθε φορά ελαττώνεται κατά 1 μέχρι να φθάσει στη τιμή -1 (<0) οπότε και το loop σταματά και το πρόγραμμα συνεχίζει εκτέλεση εντολών μετά την εντολή CONTINUE</p>

Ποια η τιμή του I και M στο τέλος του loop??

```

M = 0
DO 100 I=1,5
    M = M +1
100 CONTINUE

```

I = 6, M = 5

Για κάθε περίπτωση πόσες φορές θα εκτελεστεί το Loop?

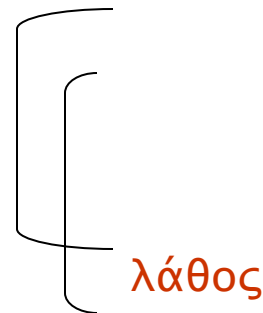
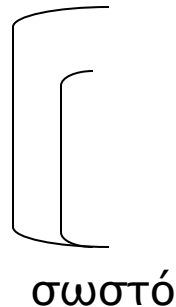
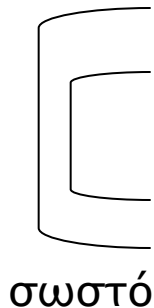
- | | |
|----------------------|----------|
| a. DO 10 I=10, 10 | a. 1 |
| b. DO 10 I=1, 10, 3 | b. 4 |
| c. DO 10 I=-2, 2 | c. 5 |
| d. DO 10 I=1, 2, 0.5 | d. Καμία |
| e. DO 10 I=10,5, -1 | e. 6 |

Κανόνες και περιορισμοί για DO loops

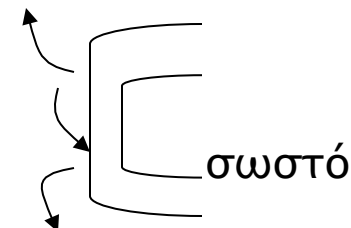
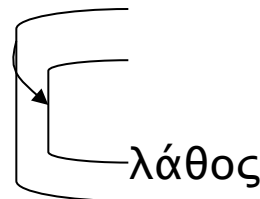
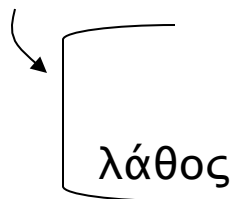
- ❑ Το DO loop επιφέρει μια ομαδοποίηση εντολών. Έτσι υπάρχουν περιορισμοί για την είσοδο/έξοδο από το loop. Υπάρχουν 2 τρόποι εξόδου: Η κανονική έξοδος όπου ο δείκτης εξαντλεί τις τιμές που μπορεί να πάρει. Η εξαναγκασμένη έξοδος όπου κάποια από τις εντολές του loop περιέχει μια εντολή IF ή GOTO που μεταφέρει τον έλεγχο έξω από το loop. Αυτό μπορεί να συμβεί και την πρώτη φορά που εκτελείται το loop ή και την τελευταία.
Οι παράμετροι ISTART, IFIN, ISTEP καθορίζουν το μέγιστο αριθμό εκτελέσεων του Loop αλλά οι εντολές GOTO και IF τον πραγματικό.
- ❑ Ο δείκτης του loop μπορεί να χρησιμοποιηθεί αφού βγούμε από το loop. Ιδιαίτερη προσοχή χρειάζεται για την τιμή του. Στην εξαναγκασμένη έξοδο η τιμή είναι η τρέχουσα τιμή του (όταν ήταν στο loop) για την κανονική ωστόσο η τιμή του έχει αυξηθεί κατά ISTEP μετά την τελευταία φορά εκτέλεση.
- ❑ Η πρώτη εντολή της περιοχής ενός loop (μετά την εντολή DO) πρέπει να είναι εκτελέσιμη εντολή.
- ❑ Η τελευταία εντολή της περιοχής ενός loop πρέπει να είναι υποχρεωτικά η εντολή CONTINUE (επίσης χρησιμοποιείται η END DO χωρίς αρίθμηση).
- ❑ Οι παράμετροι ISTART, IFIN, ISTEP όπως επίσης και ο δείκτης του loop δεν πρέπει να μεταβάλλονται κατά την εκτέλεση των εντολών του loop.
- ❑ Ένα DO loop επιτρέπεται να περιέχει ένα άλλο DO loop (nested loops). **ΑΛΛΑ...**

Κανόνες και περιορισμοί για DO loops

- Ένα εσωτερικό loop πρέπει να περιέχεται εξ' ολοκλήρου στο εξωτερικό loop. Το πολύ, θα πρέπει το εσωτερικό loop να καταλήγει στην ίδια CONTINUE εντολή με το εξωτερικό. (Είναι καλή πρακτική να χρησιμοποιείτε διαφορετική καταληκτική εντολή για τα διάφορα nested loops). Σημαντική η καλή μετατόπιση του κύριου τμήματος κάθε loop ώστε να ξεχωρίζει που ξεκινά και που τελειώνει.
- Τα παρακάτω σχήματα δηλώνουν nested loops με σωστή δομή:



- Απαγορεύεται η μεταφορά ελέγχου του προγράμματος μέσα στο loop από μια εντολή IF ή GOTO που δεν ανήκει στο loop. (Εξαίρεση η κλήση συνάρτησης ή υποπρογράμματος – subroutines – όπως θα δούμε)



Παράδειγμα – Οι τριάδες (x,y,z) που είδαμε πριν

PROGRAM TRIPLET

```
INTEGER IX, IY, IZ
INTEGER L, IXYZ, IXYZN
READ *, L
```

c Initialization for IX

```
IX = 0
```

```
1 IX = IX + 1
```

c Initialization for IY

```
IY = 0
```

```
2 IY = IY + 1
```

c Initialization for IZ

```
IZ = 0
```

```
3 IZ = IZ + 1
```

c Calculate the sum of the squares

```
IXYZ = IX**2 + IY**2 + IZ**2
```

c Calculate the remainder

```
IXYZN = (IXYZ/L)*L - IXYZ
```

c Check to see if it is zero

```
IF (IXYZN.EQ.0) THEN
```

```
PRINT *,IX,IY,IZ
```

```
ENDIF
```

```
IF ( (IZ - 9) .LT. 0) GOTO 3
```

```
IF ( (IY - 9) .LT. 0) GOTO 2
```

```
IF ( (IX - 9) .LT. 0) GOTO 1
```

```
END
```

PROGRAM TRIPLET

```
INTEGER IX, IY, IZ
INTEGER L, IXYZ, IXYZN
READ *, L
```

```
DO 32 IX = 1, 9
```

```
DO 31 IY = 1, 9
```

```
DO 30 IZ = 1, 9
```

```
IXYZ = IX**2 + IY**2 + IZ**2
```

```
IXYZN = (IXYZ/L)*L - IXYZ
```

```
IF (IXYZN.EQ.0) PRINT *,IX, IY, IZ
```

```
30 CONTINUE
```

```
31 CONTINUE
```

```
32 CONTINUE
```

```
END
```

Loop (επαναληπτική διαδικασία) με λογική σχέση

Υπάρχουν αρκετά προβλήματα για τα οποία οι μηχανισμοί ελέγχου που είδαμε μέχρι τώρα δεν επαρκούν.

Ωστόσο υπάρχουν δύο μηχανισμοί που δεν έχουν άμεση μορφή αλλά πρέπει να κατασκευαστούν από περισσότερες βασικές μορφές

Οι δύο μηχανισμοί ελέγχου είναι:

- WHILE (λογική έκφραση) DO (group εντολών)
 - REPEAT (group εντολών) UNTIL (λογική έκφραση)
- } Προσοχή: Δεν είναι εντολές

- ❑ Η πρώτη δομή μπορεί να μην εκτελεστεί ποτέ
(αφού η έκφραση μπορεί να μην ικανοποιείται)
Η δομή REPEAT θα εκτελεστεί τουλάχιστον μια φορά

- ❑ Η δομή WHILE ισοδυναμεί με:

```
<label> IF (λογική έκφραση) THEN
    ·
    · } Ομάδα
    · } εντολών
    ·
    GOTO <label>
ENDIF
```

- Η δομή REPEAT ισοδυναμεί με:

```
<label> CONTINUE
    ·
    · } Ομάδα
    · } εντολών
    ·
    IF (λογική έκφραση) GOTO <label>
```

Δομή **WHILE** (λογική έκφραση) **DO** (group εντολών)

PROGRAM FIND

C... Αυτό το πρόγραμμα βρίσκει την
 C... πρώτη εμφάνιση ενός αριθμού σε μια λίστα
 C... Χρησιμοποιούμε 100 αριθμούς ενώ ο πίνακας
 C... είναι κατά 1 μεγαλύτερος από τη λίστα

```
REAL A(101)
INTEGER A, MARK
INTEGER IEND, I
```

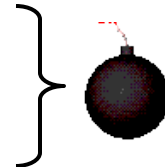
```
READ(1,*) MARK
READ(1,*) IEND
READ(1,*) (A(I), I=1, IEND)
```

```
I = 1
```

```
A(IEND+1) = MARK
```

```
100 IF (MARK.NE.A(I)) THEN } While
    I = I + 1                } DO
    GOTO 100
ENDIF
```

Με την εντολή αυτή σίγουρα θα βρούμε κάποιο αριθμό !



Επικίνδυνο αφού μπορεί
να συνεχιστεί επ' άπειρο

```
IF (I.EQ.(IEND+1)) THEN
  PRINT *, 'STOIXEIO DEN VRETHIKE'
ELSE
  PRINT *, 'STOIXEIO STI THESI ', I
ENDIF
```

```
END
```

Δομή REPEAT UNTIL...

Συνήθως χρησιμοποιείται σε προβλήματα που χρειάζονται αριθμητική επίλυση. Τέτοιου είδους προβλήματα συνήθως απαιτούν την επανάληψη ενός υπολογισμού μέχρι τα αποτελέσματα από δύο διαδοχικούς υπολογισμούς διαφέρουν κάποια μικρή ποσότητα που αποφασίζετε ανάλογα με το πρόβλημα.

➤ Για παράδειγμα ο υπολογισμός του e^x

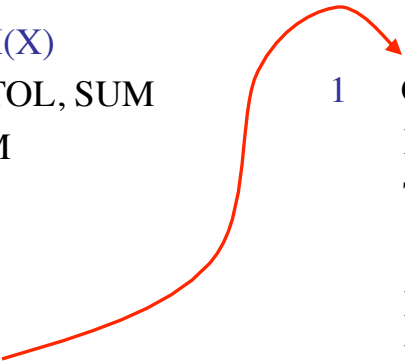
Μπορεί να γραφεί:
$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = 1 + \sum_{n=1}^{\infty} \frac{x^{n-1}}{(n-1)!} \frac{x}{n}$$

- Κάθε όρος προκύπτει από τον προηγούμενο πολ/ζοντας με x/n
- Για κάποια τιμή του n , οι όροι γίνονται πολύ μικροί και προσφέρουν ελάχιστο στην τιμή του αθροίσματος
- ❖ Οι όροι υπολογίζονται μέχρι το x/n να γίνει μικρότερο από το όριο ανοχής
- Το πλήθος των υπολογισμών δεν είναι γνωστός από πριν

PROGRAM EXPX(X)

```
REAL TERM, X, TOL, SUM
INTEGER NTERM
TOL=0.001
SUM = 1.0
TERM = 1.0
NTERM = 0
```

```
1 CONTINUE
  NTERM = NTERM + 1
  TERM = (X/NTERM)*TERM
  SUM = SUM + TERM
  IF (TERM.GT.TOL) GOTO 1
END
```



DO WHILE ...

```
PROGRAM SROOT
IMPLICIT NONE
REAL X, GUESS
INTEGER COUNT, I

PRINT *, 'DWSTE MIA TIMH X'
READ *, X
PRINT *, 'DWSTE POSOYS OROUS'
READ *, COUNT

GUESS = 1
DO I = 1, COUNT
    GUESS = (GUESS + X/GUESS)/2.
ENDDO

PRINT *, 'Root = ', GUESS

END
```


```
PROGRAM SROOT
IMPLICIT NONE
REAL X, GUESS, PRECISION

PRINT *, 'DWSTE MIA TIMH X'
READ *, X
PRINT *, 'DWSTE TIN AKRIBEIA'
READ *, PRECISION

GUESS = 1
DO WHILE (ABS(X - GUESS*GUESS).GT.PRECISION)
    GUESS = (GUESS + X/GUESS)/2.
ENDDO

PRINT *, 'Root = ', GUESS

END
```



ελεγχόμενη
επαναλαμβανόμενη
εκτέλεση
Το loop εκτελείται
έως να γίνει ψευδής
η λογική έκφραση.

Μπορεί να εκτελείται
επάπειρο

READ/WRITE από/σε αρχεία (files)

- ❑ Πολλές φορές τα δεδομένα ενός προβλήματος είναι πάρα πολλά και είναι αδύνατη η εισαγωγή τους από το πληκτρολόγιο ή η εκτύπωση των αποτελεσμάτων τους προγράμματος στην οθόνη
- ❑ Τα δεδομένα βρίσκονται σε κάποια αρχεία (files) και πρέπει να τα διαβάσουμε ή θα πρέπει να τα γράψουμε σε κάποιο file
- ❑ Ο απλούστερος τρόπος για να το κάνουμε είναι μέσω των ακόλουθων εντολών

`OPEN(unit=<unit number>, file= '<file name>', status= 'status code')`

`READ(<unit number>,*) variable list`

`WRITE(<unit number>,*) variable list`

Η πρώτη εντολή ανοίγει κάποιο file με τυχαίο όνομα <file name> και το συνδέει σε μια λογική μονάδα (unit number) του υπολογιστή

Αν το file προϋπάρχει τότε θα πρέπει να χρησιμοποιήσουμε `status = 'OLD'`

Χρησιμοποιείται όταν διαβάζουμε δεδομένα από ένα file

Για να γράψουμε σε ένα νέο file τότε πρέπει να χρησιμοποιήσουμε `status = 'NEW'`

Αν δε ξέρουμε αν το file προϋπάρχει τότε χρησιμοποιούμε `status = 'UNKNOWN'`

Στην περίπτωση αυτή, αν το file προϋπήρχε το περιεχόμενό του θα αντικατασταθεί από τα νέα δεδομένα που θα γράψουμε

READ/WRITE από/σε αρχεία (files)

Για παράδειγμα

```
PROGRAM TEST  
REAL X,Y
```

```
OPEN(UNIT=60,file='phy145.dat',status='OLD')
```

```
OPEN(UNIT=61,file='phy145.out',status='NEW')
```

```
DO I = 1, 1000
```

```
  READ(60,*)X
```

```
  Y = X*10
```

```
  WRITE(61,*)Y
```

```
END DO
```

```
END
```

Ανοίγουμε κάποιο file (phy145.dat) που προϋπάρχει και το θέτουμε στη μονάδα 60

Ανοίγουμε κάποιο νέο file (phy145.out) και το θέτουμε στη μονάδα 61

Από τη μονάδα 60 (phy145.dat) θα διαβάσουμε 1000 στοιχεία καθένα από τα οποία θα τα βάλουμε στη μεταβλητή X

Τα αποτελέσματα του προγράμματος (μεταβλητή Y) θα γράφουν στην μονάδα 61 (phy145.out)

Βασική προϋπόθεση στα παραπάνω ότι το file που διαβάζουμε έχει τουλάχιστον 1000 στοιχεία, αλλιώς όταν το πλήθος των στοιχείων τελειώσει θα μας δώσει error

Θα δούμε αργότερα πως μπορούμε να αποφύγουμε το πρόβλημα αυτό

Αν το file περιέχει περισσότερα από ένα στοιχεία σε κάθε σειρά του ή θέλουμε να γράψουμε περισσότερα από ένα στοιχεία σε κάθε σειρά του τότε μπορούμε να ορίσουμε περισσότερες μεταβλητές στην εντολή READ/WRITE