ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

ΕΙΣΑΓΩΓΗ ΣΤΗΝ FORTRAN

Σημειώσεις για το εργαστήριο του μαθήματος
ΑΡΙΘΜΗΤΙΚΗ ΑΝΑΛΥΣΗ

ΧΡΗΣΤΟΣ ΤΣΑΓΓΑΡΗΣ

ΕΕΔΙΠ
ΤΜΗΜΑΤΟΣ ΜΑΘΗΜΑΤΙΚΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΑΙΓΑΙΟΥ

ΜΑΙΟΣ 2005

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1ο: Εισαγωγή

Η γλώσσα **Fortran** είναι η πρώτη γλώσσα προγραμματισμού η οποία επέτρεψε να κωδικοποιηθεί ένας αριθμητικός υπολογισμός μ' ένα τύπο που να μοιάζει μ' ότι γράφουμε στα μαθηματικά. Αυτό φυσικά ήταν και ο σκοπός σχεδιασμού της γλώσσας Fortran, να κωδικοποιηθούν οι μαθηματικοί τύποι.

Το όνομα **Fortran** προέρχεται από τις λέξεις **FOR**mula **TRAN**slation, που σημαίνει μετάφραση τύπων. Δημιουργήθηκε την τριετία 1954-1957 στα εργαστήρια της εταιρείας IBM υπό την επίβλεψη του John Backus. Η πρώτη έκδοση της γλώσσας ήταν η **Fortran I** και χρησιμοποιήθηκε για πρώτη φορά τον Απρίλιο του 1957 σ' ένα υπολογιστή της IBM (Μοντέλο 704). Την άνοιξη του 1958 μετά από μερικές αλλαγές και προσθέσεις εμφανίστηκε η **Fortran II**. Όμως η πιο διαδεδομένη και γνωστή έκδοση της **Fortran** ήταν η **Fortran IV** που πρωτοπαρουσιάστηκε το 1962.

Το 1966 δημιουργήθηκε η πρώτη standard έκδοση της (**Fortran 66**) η οποία εφαρμόστηκε ευρέως στη διδασκαλία και η οποία είχε το πλεονέκτημα της χρήσης υπορουτινών, της ανεξάρτητης μετάφρασης και της ανεξαρτησίας από συγκεκριμένο μηχάνημα. Δηλαδή, ένα πρόγραμμα γραμμένο σε γλώσσα **Fortran 66**, μπορούσε να γίνει δεκτό χωρίς καμία αλλαγή από κάθε υπολογιστή που διέθετε μεταγλωττιστή (Compiler) **Fortran**.

Το 1978 μετά από πιέσεις για ανταγωνισμό από καινούργιες γλώσσες προγραμματισμού, όπως PASCAL, ADA, Modula 2, C C++, δημιουργήθηκε μία καινούρια standard έκδοση, η **Fortran 77**, η οποία όμως δεν μπόρεσε να ανταποκριθεί ικανοποιητικά έναντι των άλλων γλωσσών.

Στις αρχές της δεκαετίας του 90 (1991), εμφανίστηκε μια νέα standard έκδοση (**Fortran 90**) της γλώσσας **Fortran**, για να τη προσαρμόσει στις νέες απαιτήσεις της επιστήμης της Πληροφορικής.

Η τελευταία standard έκδοση της γλώσσας, με πολύ μικρές αλλαγές από την προηγούμενη, είναι η **Fortran 95**, η οποία μετά τους απαραίτητους ελέγχους διατίθεται στην αγορά από το 1997.

Θ' αναρωτηθεί κανείς, γιατί η γλώσσα Fortran επιζεί τόσα χρόνια και μάλιστα χρησιμοποιείται, σε κάποιες περιπτώσεις και πιο πολύ, από πολλές άλλες γλώσσες πιο πλούσιες, πιο δυναμικές και πιο καινούργιες;

Η κυριότερη αιτία είναι χωρίς αμφιβολία η απλότητά της.

Ένας ακόμη σοβαρός λόγος για τη διατήρηση και ανάπτυξη της γλώσσας είναι ότι τα περισσότερα ερευνητικά προγράμματα που αναπτύχθηκαν τα τελευταία 50 χρόνια, όπως επιστημονικές ή τεχνικές εφαρμογές, προγράμματα στατιστικής ή επιχειρησιακής έρευνας κ.τ.λ. είναι γραμμένα σε γλώσσα **Fortran**.

Επίσης, υπάρχουν έτοιμα και δημοσιευμένα σε διεθνή περιοδικά μικρά προγράμματα (υποπρογράμματα) σε γλώσσα **Fortran** ικανά να λύσουν κάθε πρόβλημα μαθηματικών, φυσικής, αστρονομίας, και πολλών άλλων επιστημονικών τομέων. Έτσι, ο κάθε ερευνητής, επιστήμονας, φοιτητής ή ερασιτέχνης του προγραμματισμού μπορεί να τα χρησιμοποιήσει πολύ εύκολα προσαρμόζοντάς τα στις ανάγκες του προβλήματος που τον απασχολεί.

Τι γίνεται όμως με κάθε νέα έκδοση της γλώσσας;

Κάθε νέα έκδοση είναι κι ένα υπερσύνολο των εντολών της προηγούμενης. Δηλαδή οι χρήστες και οι προγραμματιστές διαπιστώνουν τις ελλείψεις και τις αδυναμίες της γλώσσας και σχεδόν κάθε δεκαετία δημιουργούνται νέα πρότυπα που συμπληρώνουν τα προηγούμενα. Έτσι, όλα τα προγράμματα γραμμένα σε παλαιότερες εκδόσεις της **Fortran** «τρέχουν» και με τους μεταγλωττιστές (compilers) κάθε καινούργιας έκδοσης. Πολύ απλά, δίνεται η δυνατότητα για βελτιώσεις των προγραμμάτων τόσο στη σύνταξη όσο και στην εκτέλεση μέσω της σταδιακής ανανέωσης και αντικατάστασης των παλαιών προδιαγραφών της γλώσσας οι οποίες και δεν συνιστώνται για ένα σύγχρονο και αποδοτικό προγραμματισμό.

Συνοψίζοντας, μπορούμε να πούμε ότι η γλώσσα **Fortran** είναι αρκετά εύκολη στην εκμάθησή της, ότι είναι προσανατολισμένη στην έρευνα, ότι είναι περισσότερο αποτελεσματική στις μαθηματικές πράξεις, ότι υπάρχουν διαθέσιμες και πολλές φορές δωρεάν, βιβλιοθήκες έτοιμων προγραμμάτων για κάθε χρήση, ότι είναι εύκολη στη χρήση της και από τις πιο σταθερές γλώσσες, και τέλος, ότι παραμένει το πιο ισχυρό εργαλείο για τις αριθμητικές και επιστημονικές εφαρμογές.

Κεφάλαιο 2ο: Χαρακτηριστικά στοιχεία της γλώσσας Fortran

2.1 Το αλφάβητο της Fortran

Όπως κάθε φυσική γλώσσα έχει το δικό της αλφάβητο, έτσι και η **Fortran** έχει το δικό της αλφάβητο, το οποίο αποτελείται από:

- τα 52 γράμματα του λατινικού αλφάβητου (πεζά και κεφαλαία)
  A, B, ..., Z και a, b, …, z
- τα 10 αριθμητικά ψηφία
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- τα σύμβολα των αριθμητικών πράξεων
  +, (πρόσθεσης)
  -, (αφαίρεσης)
  *, (πολλαπλασιασμού)
  /, (διαίρεσης)
  ** (ύψωσης σε δύναμη)
- τα ακόλουθα ειδικά σύμβολα:
  . (Τελεία)
  , (Κόμμα)
  = (Ίσον)
  < (Μεγαλύτερο)
  > (Μικρότερο)
  $ (Δολάριο)
  ((Ανοικτή παρένθεση)
) (Κλειστή παρένθεση)
  & (Σύμβολο του «και»)
  ‘ (Απόστροφος)
  “ (Διπλή απόστροφος)
  : (Άνω και κάτω τελεία)
  :: (Διπλή άνω και κάτω τελεία)
  ! (Θαυμαστικό)
  ; (Ελληνικό Ερωτηματικό)
  ? (Αγγλικό Ερωτηματικό)
  _ Υπογράμμιση (underscore)
- Το κενό διάστημα (Space Bar).

  Το κενό διάστημα (κενό) αγνοείται από τον μεταγλωττιστή μέσα σ' ένα πρόγραμμα **Fortran**, μας επιτρέπει όμως να παρουσιάσουμε ένα κείμενο πιο ευκολοδιάβαστο και περισσότερο κατανοητό όταν εμφανίζεται στην οθόνη ή όταν είναι τυπωμένο στο χαρτί. Μπορούμε δηλαδή, να γράφουμε τις λέξεις της Fortran είτε τη μια ακριβώς μετά την άλλη είτε με ένα κενό ή και περισσότερα κενά διαστήματα μεταξύ τους.

2.2 Το λεξιλόγιο της Fortran

Το λεξιλόγιο της γλώσσας **Fortran** αποτελείται από μία σειρά «λέξεων» της αγγλικής γλώσσας οι οποίες ονομάζονται εντολές (Statements).

Διακρίνουμε δύο είδη εντολών:

Α. Τις εντολές που δεν εκτελούνται ή *εντολές προδιαγραφών και δηλώσεων* που είναι απλές ενδείξεις για το μεταγλωττιστή.

Π.χ. **Integer** (δηλωτική εντολή), **Real** (δηλωτική εντολή), **Format** (εντολή προδιαγραφών) κ.τ.λ.

Β. Τις εντολές που εκτελούνται και που μετατρέπονται από τον μεταγλωττιστή σε μια σειρά από εντολές στη γλώσσα μηχανής έτοιμες να εκτελεστούν από τον υπολογιστή.

2.3 Η γραμματική της Fortran

Όπως κάθε γλώσσα έχει τη δική της γραμματική και συντακτικό, έτσι και η γλώσσα **Fortran** διαθέτει τους δικούς της κανόνες, απόλυτους στην εφαρμογή τους, οι οποίοι και ορίζουν τη σύνταξη της γλώσσας **Fortran**.

Η σύνταξη του προγράμματος πρέπει να γίνεται με τρόπο σαφή και ακριβή, δηλαδή να περιγράφεται με τρόπο που να μην αφήνει διφορούμενες καταστάσεις στη δομή του.

Εκτός από τις «λέξεις» (εντολές) της γλώσσας **Fortran** τις οποίες και ονομάζουμε «δεσμευμένες», ο δημιουργός ενός προγράμματος σε γλώσσα **Fortran** μπορεί να δημιουργήσει και δικές του λέξεις τις οποίες τις ονομάζουμε «συμβολικά ονόματα» και τις οποίες χρησιμοποιεί ως ονόματα μεταβλητών και σταθερών, ακολουθώντας τους εξής κανόνες:

- Το συμβολικό όνομα αποτελείται από μια σειρά από αλφαριθμητικούς χαρακτήρες (δηλαδή γράμματα ή αριθμούς) και τον χαρακτήρα _ (underscore, υπογράμμιση).
- Ο πρώτος χαρακτήρας πρέπει υποχρεωτικά να είναι γράμμα.
- Δεν γίνεται διάκριση μεταξύ πεζών και κεφαλαίων. Ο compiler μετατρέπει τα πεζά σε κεφαλαία.
- Δεν επιτρέπεται να χρησιμοποιηθεί ως συμβολικό όνομα δεσμευμένη λέξη της **Fortran**.
- Η **Fortran 77** αναγνωρίζει μόνο τους 6 πρώτους χαρακτήρες του συμβολικού ονόματος και αγνοεί τους υπόλοιπους. Π.χ. τα ονόματα NUMERICALS, NUMERI και NUMERICA για την **Fortran 77** θεωρούνται ακριβώς όμοια.
- Στη **Fortran 90/95** το μήκος του συμβολικού ονόματος μπορεί να φτάσει μέχρι τους 31 χαρακτήρες αλλά δε συνιστάται η χρήση πολλών χαρακτήρων στον προσδιορισμό του συμβολικού ονόματος.
- Η απόδοση συμβολικών ονομάτων είναι ελεύθερη και γίνεται αυθαίρετα εκ μέρους του προγραμματιστή. Όμως, είναι καλό και συνιστάται να υπάρχει μια στενή σχέση μεταξύ του συμβολικού ονόματος και του φυσικού μέγεθος που εκπροσωπείται.

Κεφάλαιο 3ο: Βασικά στοιχεία ενός προγράμματος της γλώσσας Fortran

3.1 Μορφή Προγράμματος

Τα προγράμματα **Fortran** γράφονται σε αρχείο ASCII, χρησιμοποιώντας οποιοδήποτε απλό κειμενογράφο, σύμφωνα με τις παρακάτω οδηγίες:

- Γράφουμε μία εντολή ανά γραμμή. Οι εντολές πρέπει να περιέχονται μεταξύ των στηλών 7 και 72. Στη **Fortran 90/95**, σε μια γραμμή μπορούν να εμφανίζονται και περισσότερες εντολές αρκεί στο τέλος κάθε εντολής να προστεθεί ο χαρακτήρας ; (semicolon).
- Αν το μήκος μιας γραμμής ξεπερνά την $72^{η}$ στήλη μπορούμε να συνεχίσουμε την εντολή στην επόμενη σειρά βάζοντας οποιοδήποτε χαρακτήρα (**εκτός από το 0**) στην $6^{η}$ στήλη. Στη **Fortran 77** μπορούμε να χρησιμοποιήσουμε μέχρι 19 διαδοχικές γραμμές συνέχειας μιας εντολής και μια εντολή μπορεί να περιέχει το πολύ μέχρι 1320 χαρακτήρες. Ενώ, μια εντολή της **Fortran 90/95**, μπορεί να συνεχιστεί και σε περισσότερες γραμμές (μέχρι 551) αν τοποθετηθεί ένας δείκτης συνέχειας.
- Στη **Fortran 77** μόνο οι 72 πρώτες στήλες λαμβάνονται υπόψη από τον μεταγλωττιστή, ενώ οι στήλες 73- 80 εμφανίζονται στην οθόνη ή τυπώνονται από την εκτυπωτική όπως έχουν.
- Τα σχόλια μπαίνουν γράφοντας το γράμμα C ή το χαρακτήρα * στην $1^{η}$ στήλη. Όταν στην πρώτη στήλη υπάρχει το γράμμα C ή το σύμβολο *, τότε όλο το περιεχόμενο αυτής της γραμμής (στήλες 2-72) θα αγνοηθεί από το μεταγλωττιστή.
- Κάθε εντολή είναι δυνατόν να συσχετιστεί με κάποιον αυθαίρετο αριθμό που ονομάζεται ετικέτα (label), για αναφορά σε αυτήν από άλλες εντολές. Η ετικέτα μιας εντολής πρέπει να γράφεται μεταξύ των στηλών 1 και 5 (δηλαδή δεν μπορεί να περιλαμβάνει πάνω από 5 διαδοχικά ψηφία.)

3.2 Τύποι δεδομένων

Η **Fortran** υποστηρίζει του εξής τύπους δεδομένων:

- Ακέραιος (Integer)
- Πραγματικός (Real)
- Διπλής ακρίβειας (Double precision)
- Μιγαδικός (Complex)
- Λογικός (Logical)
- Χαρακτήρας (Character)

Το εύρος τιμών των ακεραίων τύπων ποικίλει ανάλογα με τον τρόπο αποθήκευσης στη μνήμη. Συγκεκριμένα το εύρος τιμών των ακεραίων (Integer) που για την αποθήκευσή τους χρησιμοποιούνται 16 bits κυμαίνεται από -2^{15} έως $2^{15} - 1$, ενώ το εύρος τιμών των ακεραίων (Integer *4) που για την αποθήκευσή τους χρησιμοποιούνται 32 bits κυμαίνεται από -2^{31} έως $2^{31} - 1$.

Το εύρος τιμών των πραγματικών (Real ή Real *4) που για την αποθήκευσή τους χρησιμοποιούνται 32 bits κυμαίνεται για τους αρνητικούς από -10^{38} έως -10^{-38} και για τους θετικούς από 10^{-38} έως 10^{38}, ενώ το εύρος τιμών των πραγματικών (Real *8 ή Double Precision) που για την αποθήκευσή τους χρησιμοποιούνται 64 bits κυμαίνεται για τους αρνητικούς από -10^{308} έως -10^{-308} και για τους θετικούς από 10^{-308} έως 10^{308}.

Όταν κατά τη διάρκεια των πράξεων, ένα δεδομένο (σταθερά ή μεταβλητή) πάρει τιμή μεγαλύτερη από τη θεωρητικώς μεγαλύτερη, τότε συμβαίνει υπερχείλιση και τυπώνεται το μήνυμα Overflow, ενώ όταν πάρει τιμή μικρότερη από τη θεωρητικώς μικρότερη, τότε συμβαίνει υποχείλιση και τυπώνεται το μήνυμα Underflow. Και στις δύο περιπτώσεις το μήνυμα σημαίνει ότι έχουμε ξεπεράσει τη μέγιστη δυνατότητα αποθήκευσης των αριθμητικών τιμών του υπολογιστή, γεγονός που θα προκαλέσει είτε αλλοίωση των αποτελεσμάτων είτε διακοπή του προγράμματος.

3.3 Μεταβλητές

Μια μεταβλητή (Variable) στον προγραμματισμό χαρακτηρίζεται από τρία βασικά στοιχεία.

α. Το όνομά της.

β. Τον τύπο της.

γ. Την αριθμητική τιμή της.

Μια μεταβλητή σ' ένα πρόγραμμα θα έχει ένα όνομα σταθερό, ένα τύπο σταθερό και μια τιμή που θα μεταβάλλεται.

Το όνομα της μεταβλητής που συνήθως λέγεται και συμβολικό όνομα και ακολουθεί όλους τους κανόνες που αναφέρθηκαν στην παράγραφο 2.3.

Ο τύπος μιας μεταβλητής δημιουργείται με δύο τρόπους, άμεσα σε σχέση με το πρώτο γράμμα του ονόματος της, είτε έμμεσα με τη χρήση δηλωτικών εντολών.

Όταν το όνομα της μεταβλητής αρχίζει μ' ένα από τα 6 γράμματα **I, J, K, L, M, N,** τότε η μεταβλητή θα έχει αυτόματα ακέραια τιμή, δηλαδή το περιεχόμενο της θέσης μνήμης που αντιστοιχεί, επεξεργάζεται από τον υπολογιστή σαν ένας ακέραιος αριθμός.

Όταν το όνομα της μεταβλητής αρχίζει μ' ένα από τα υπόλοιπα 20 γράμματα του Λατινικού αλφαβήτου (**A-H** και **O-Z**), τότε η μεταβλητή έχει πραγματική τιμή, δηλαδή το περιεχόμενο της θέσης μνήμης που αντιστοιχεί επεξεργάζεται από τον υπολογιστή σαν ένας πραγματικός αριθμός.

Μπορούμε ν' αλλάξουμε τον τύπο της μεταβλητής έμμεσα, με χρήση δηλωτικών εντολών (declaration statements). Μια δηλωτική εντολή προδιαγραφής μπαίνει πάντα στην αρχή ενός προγράμματος. Δεν εκτελείται, αλλά χρησιμεύει για να δηλώνει τον τύπο των μεταβλητών που θα χρησιμοποιηθούν στο πρόγραμμα.

Οι δηλωτικές εντολές για τον τύπο μιας μεταβλητής είναι:

- **Integer, Integer *4** για ακέραιες μεταβλητές
- **Real, Real *4** για πραγματικές μεταβλητές

- **Double Precision, Real *8** για πραγματικές μεταβλητές διπλής ακρίβειας
- **Complex** για μιγαδικές μεταβλητές
- **Logical** για λογικές μεταβλητές
- **Character** για χαρακτήρες μεταβλητές και **Character**(n) για αλυσίδες χαρακτήρων μήκους n μεταβλητές.

Συνιστάται να δηλώνονται με αυτό τον έμμεσο τρόπο όλες οι μεταβλητές ενός προγράμματος επειδή αυτός ο τρόπος παρέχει στον προγραμματιστή καλλίτερο έλεγχο των χρησιμοποιημένων μεταβλητών αλλά και γιατί η δήλωση των μεταβλητών αποτελεί τον πρωταρχικό κανόνα του σωστού και δομημένου προγραμματισμού.

Αν θέλουμε να μην ορίζεται αυτόματα ο τύπος καμίας μεταβλητή αλλά να πρέπει να δηλώνουμε υποχρεωτικά τον τύπο κάθε μεταβλητής του προγράμματος, τότε πρέπει να γράψουμε στην αρχή του προγράμματος την εντολή:

Implicit none

Η τιμή της μεταβλητής, είναι το στοιχείο εκείνο που αλλάζει κατά τη διάρκεια της εκτέλεσης του προγράμματος. Σε μία απλή μεταβλητή αντιστοιχεί μία θέση μνήμης. Το περιεχόμενο αυτής της θέσης είναι εκείνο που μεταβάλλεται κατά την εκτέλεση του προγράμματος. Το όνομα της μεταβλητής αντιστοιχεί στη «διεύθυνση» αυτής της θέσης μέσα στη μνήμη.

Στην **Fortran 77** η αρχική τιμή μιας μεταβλητής δίνεται με την εντολή ανάθεσης που θα δούμε στη συνέχεια.

Στην **Fortran 90/95**, η αρχική τιμή μιας μεταβλητής μπορεί να δοθεί και με τη χρήση του συμβόλου **::**.

Το σύμβολο **::** χρησιμοποιείται, όχι υποχρεωτικά, δίπλα από τον ορισμό του τύπου των μεταβλητών π.χ.

Integer :: A, B, C (δηλώνουμε ότι οι μεταβλητές A, B, C είναι ακέραιες),

Real :: D, E, F (δηλώνουμε ότι οι μεταβλητές A, B, C είναι πραγματικές).

Το σύμβολο **::** γίνεται υποχρεωτικό όταν θέλουμε ταυτόχρονα με τη δήλωση του τύπου να δώσουμε και μία αρχική τιμή στην μεταβλητή. Π.χ. μπορούμε να γράψουμε :

Integer :: A = 5 (δηλώνουμε την ακέραια μεταβλητή A, στην οποία δίνουμε αρχική τιμή 5),

Real :: B = 123.56 (δηλώνουμε την πραγματική μεταβλητή B, στην οποία δίνουμε αρχική τιμή 123.56),

Integer :: A = 5, B (δηλώνουμε τις ακέραιες μεταβλητές A και B, αλλά δίνουμε την αρχική τιμή 5 μόνο στην A).

3.4 Σταθερές

Μια σταθερά (Constant) στον προγραμματισμό χαρακτηρίζεται από τρία βασικά στοιχεία.

α. Το όνομά της.

β. Τον τύπο της.

γ. Την αριθμητική τιμή της.

Μια σταθερά σ' ένα πρόγραμμα θα έχει ένα όνομα σταθερό, ένα τύπο σταθερό και μια τιμή που δεν θα μεταβάλλεται.

Το όνομα της σταθεράς που συνήθως λέγεται και συμβολικό όνομα και ακολουθεί όλους τους κανόνες που αναφέρθηκαν στην παράγραφο 2.3.

Στην **Fortran** η απόδοση αρχικών τιμών σε μία σταθερά γίνεται με την δηλωτική εντολή **Parameter** στην αρχή του προγράμματος:

Parameter (Όνομα_Σταθεράς1=τιμή, Όνομα_Σταθεράς2=τιμή, ...)

Οι τιμές αυτών των μεταβλητών δεν μπορούν να αλλάξουν κατά τη διάρκεια της εκτέλεσης του προγράμματος. Έτσι, αν θέλουμε να χρησιμοποιήσουμε σε ένα πρόγραμμα την τιμή του π = 3,14159 αντί να γράφουμε σε κάθε εντολή την αριθμητική τιμή (με μεγάλη πιθανότητα να γίνει και κάποιο λάθος), μπορούμε να κάνουμε στην αρχή του προγράμματος τη δήλωση:

Parameter (Pi = 3.14159)

και στη συνέχεια να χρησιμοποιούμε τη μεταβλητή Pi στη θέση της σταθερής αριθμητικής τιμής, π.χ.

PERIF = 2 * Pi * ΑΚΤΙΝΑ

Προσοχή στο όνομα και στον τύπο της σταθεράς που δηλώνεται με την εντολή **Parameter**. Ο τύπος της σταθεράς ακολουθεί τον άμεσο τρόπο απόδοσης τύπου (που ισχύει και στις μεταβλητές) και αν θέλουμε να αλλάξουμε τον τύπο πρέπει να χρησιμοποιήσουμε μια δηλωτική εντολή πριν από τη χρήση της εντολής **Parameter**.

Οι δηλωτικές εντολές για τον τύπο μιας σταθεράς είναι:

- **Integer, Integer *4** για ακέραιες σταθερές
- **Real, Real *4** για πραγματικές σταθερές
- **Double Precision, Real *8** για πραγματικές σταθερές διπλής ακρίβειας
- **Complex** για μιγαδικές σταθερές
- **Logical** για λογικές σταθερές
- **Character** για χαρακτήρες σταθερές και **Character**(n) για αλυσίδες χαρακτήρων μήκους n σταθερές.

3.5 Αριθμητικές Εκφράσεις

Στη γλώσσα Fortran μια **αριθμητική έκφραση** (Arithmetic expression) είναι μια σειρά από μεταβλητές, από σταθερές, από συναρτήσεις, από αριθμητικούς τελεστές πράξεων και από παρενθέσεις που ακολουθούν ορισμένους αυστηρούς κανόνες.

Ένας **όρος** είναι μια μεταβλητή ή μια σταθερά ή μια συνάρτηση ή μια αριθμητική τιμή ή μια αριθμητική έκφραση τοποθετημένη μέσα σε παρενθέσεις.

Μια **αριθμητική έκφραση** είναι μια σειρά από όρους που χωρίζονται με αριθμητικούς τελεστές πράξεων.

Δύο όροι δεν μπορούν να βρίσκονται ο ένας μετά από τον άλλο, χωρίς να υπάρχει μεταξύ τους ένας αριθμητικός τελεστής πράξεων, ούτε δύο αριθμητικοί τελεστές πράξεων δίπλα - δίπλα.

Μια **συνάρτηση** αποτελείται από ένα συμβολικό όνομα ακολουθούμενο, μέσα σε παρενθέσεις, από ένα ή περισσότερα ορίσματα. Τα ορίσματα αυτά μπορεί να είναι μια αριθμητική έκφραση και πιο συχνά μία μεταβλητή ή παράμετρος.

Στον παρακάτω πίνακα παρουσιάζονται οι πιο στοιχειώδεις συναρτήσεις της γλώσσας **Fortran**.

Όνομα	Περιγραφή	Όρισμα	Αποτέλεσμα	Παράδειγμα
ABS(X)	Απόλυτη τιμή του X	Integer Real Complex	Integer Real	**ABS**(-3.1) = 3.1 **ABS**((3,4)) = 5.0
SQRT(X)	Τετραγωνική ρίζα του X	Real Complex	Real Complex	**SQRT**(4.) = 2.0 **SQRT**((3, 4)) = (2.0, 1.0)
SIN(X)	Το ημίτονο του X. Το X είναι εκφρασμένο σε ακτίνια	Real Complex	Real Complex	**SIN**(3.141592) = 6.27832947E-007
COS(X)	Το συνημίτονο του X. Το X είναι εκφρασμένο σε ακτίνια	Real Complex	Real Complex	**COS**(3.141592) = -1.0
TAN(X)	Η εφαπτομένη του X. Το X είναι εκφρασμένο σε ακτίνια	Real Complex	Real Complex	**TAN**(0.) = 0.0
EXP(X)	e^x	Real Complex	Real Complex	**EXP**(0.) = 1.0
LOG(X)	Φυσικός λογάριθμος του X	Real Complex	Real Complex	**LOG**(1.) = 0.0
LOG10(X)	Δεκαδικός λογάριθμος του X	Real	Real	**LOG10**(1.) = 0.0
MAX(X_1, X_2, …)	Ο μέγιστος μεταξύ των X_1, X_2, …	Integer Real	Integer Real	**MAX**(2, 4, 5) = 5
MIN(X_1, X_2, …)	Ο ελάχιστος μεταξύ των X_1, X_2, …	Integer Real	Integer Real	**MIN**(2, 4, 5) = 2
INT(X)	Μετατροπή του αριθμού X σε ακέραιο με αποκοπή του δεκαδικού μέρους	Integer Real	Integer	**INT**(2.9) = 2 **INT**(-2.9) = -2
REAL(X)	Μετατροπή του αριθμού X σε πραγματικό	Integer Real	Real	**REAL**(2) = 2.0
CMPLX(X, Y)	Μετατροπή δύο αριθμών X και Y σε μιγαδικό	Integer Real	Complex	**CMPLX**(2, 4) = (2.0, 4.0)
NINT(X)	Μετατροπή του αριθμού X στο πλησιέστερο ακέραιο	Integer Real	Integer	**NINT**(2.9) = 3 **NINT**(2.3) = 2
FRACTION(X)	Το κλασματικό (δεκαδικό) μέρος του αριθμού X	Real	Real	**FRACTION**(2.9) = 0.9
MOD(X, Y)	Το ακέραιο υπόλοιπο της διαίρεσης X/Y	Integer	Integer	**MOD**(10, 3) = 1
CONJG(X)	Ο συζυγής μιγαδικός του μιγαδικού αριθμού X	Complex	Complex	**CONJG**(2, 5) = (2.0, -5.0)
IMAG(X)	Το φανταστικό μέρος του μιγαδικού αριθμού X	Complex	Real	**IMAG**(2, 5) = 5.0
REAL(X)	Το πραγματικό μέρος του μιγαδικού αριθμού X	Complex	Real	**REAL**(2, 5) = 2.0

Η τάξη με την οποία γίνονται οι πράξεις στο εσωτερικό μιας αριθμητικής έκφρασης δίχως παρενθέσεις, είναι αυστηρά καθορισμένη.

Κατά σειρά προτεραιότητας έχουμε:

f(x)	Υπολογισμός μιας συνάρτησης	προτεραιότητα 1.
**	Ύψωση σε δύναμη	προτεραιότητα 2.
*, /	Πολ/σμός ή διαίρεση	προτεραιότητα 3.
+, -	Πρόσθεση ή αφαίρεση	προτεραιότητα 4.

Όταν έχουμε πράξεις με ίδια προτεραιότητα εκτελούνται οι πράξεις από αριστερά προς τα δεξιά.

Προσοχή: Όταν έχουμε διαδοχικές πράξεις με ύψωση σε κάποια δύναμη, τότε οι πράξεις γίνονται από δεξιά προς τα αριστερά. Π.χ.

Το X**Y**Z υπολογίζεται σαν X**(Y**Z)

Παράδειγμα: Για τον υπολογισμό της έκφρασης : X*Y**3/T–R**2+P/Q**4+T*S θα γίνουν οι πράξεις με την εξής σειρά:

```
a = Y ** 3
b = R ** 2
c = Q ** 4
d = X * a
e = d / T
f = P / c
g = T * S
h = e - b
i = h + f
j = I + g
```

Όταν υπάρχουν πράξεις με παρενθέσεις, υπολογίζεται πρώτα το περιεχόμενο κάθε παρένθεσης. Όταν υπάρχουν περισσότερες διαδοχικές παρενθέσεις, υπολογίζεται πρώτα εκείνη που είναι πιο εσωτερική και στη συνέχεια προχωρούμε προς τις εξωτερικές παρενθέσεις.

Παράδειγμα: Η έκφραση ((X*(Y+Z)+A*A)*C) υπολογίζεται ως εξής:

```
a = Y+Z
b = X*a+A*A
c = b*C
```

Σημείωση: Επειδή ο αριθμός των δυνατών παρενθέσεων θεωρητικά είναι άπειρος, γι' αυτό το λόγο μπορούμε να γράφουμε τις αριθμητικές εκφράσεις μέσα σε παρενθέσεις, αποφεύγοντας έτσι τις τυχόν αμφισβητήσεις στις πράξεις.

Αν μια αριθμητική έκφραση περιλαμβάνει μόνο ακέραιους όρους το αποτέλεσμα θα είναι ένας ακέραιος αριθμός, αν περιλαμβάνει μόνο πραγματικούς το αποτέλεσμα θα είναι ένας πραγματικός αριθμός. Αξίζει να υπενθυμίσουμε ότι η διαίρεση δύο ακεραίων δίνει σαν αποτέλεσμα το ακέραιο πηλίκο της διαίρεσης. Π.χ.

η διαίρεση 5 / 7 δίνει πηλίκο 0

και η διαίρεση 12 / 5 δίνει πηλίκο 2.

Όταν σε μια αριθμητική έκφραση υπάρχουν όροι και από τους 2 τύπους (ακέραιοι, πραγματικοί) και δεν υπάρχουν παρενθέσεις τότε, οι ακέραιοι μετατρέπονται σε πραγματικούς, πριν από την εκτέλεση των πράξεων και έτσι το αποτέλεσμα θα είναι ένας πραγματικός.

Όταν όμως υπάρχουν παρενθέσεις και το περιεχόμενο μιας παρένθεσης αποτελείται μόνο από ακέραιους, τότε υπολογίζεται η παρένθεση αυτή σαν ακέραιος και έπειτα μετατρέπεται το αποτέλεσμα σε πραγματικό.

Παράδειγμα. Για τον υπολογισμό της έκφρασης A+B*I/J όλοι οι όροι μετατρέπονται σε πραγματικούς πριν από τις πράξεις, αντίθετα για τον υπολογισμό της έκφρασης A+B*(I/J) αυτό που είναι μέσα στην παρένθεση είναι μια ακέραια διαίρεση που υπολογίζεται κανονικά σαν ακέραιος και έπειτα το πηλίκο μετατρέπεται σε πραγματικό αριθμό. Έτσι, αν έχουμε τις τιμές:

A=5.0, B=3.0, I=4 και J=7

τότε, χωρίς παρένθεση βρίσκουμε:

A + B * I / J = 6.714,

πράγματι, οι πράξεις γίνονται με την εξής σειρά:

a = B * I = 3.0 * 4 = 3.0 * 4.0 = 12.0

b = a / J = 12.0 / 7 = 12.0 / 7.0 = 1.714

c = A + b = 5.0 + 1.714 = 6.714

ενώ με τη βοήθεια των παρενθέσεων το σωστό αποτέλεσμα είναι:

A + B * (I / J) = 5.0,

πράγματι, οι πράξεις γίνονται με την εξής σειρά:

a = I / J = 4 / 7 = 0

b = B * a = 3.0 * 0 = 3.0 * 0.0 = 0.0

c = A + b = 5.0 + 0.0 = 5.0

3.6 Οι βασικές εντολές ενός προγράμματος της γλώσσας Fortran

3.6.1 Εντολή ανάθεσης

Σ' ένα πρόγραμμα **Fortran** εκείνη η εντολή που χρησιμοποιείται πιο συχνά είναι η εντολή της ανάθεσης (assignment statement). Μια εντολή ανάθεσης γράφεται ως εξής.

Μεταβλητή = Έκφραση

Κατά τη διάρκεια της εκτέλεσης της παραπάνω εντολής ο Η/Υ υπολογίζει πρώτα την τιμή της έκφρασης που βρίσκεται στα δεξιά του τελεστή ανάθεσης (=) και στη συνέχεια, αυτή τη τιμή που θα βρει, θα την αναθέσει στη μεταβλητή που βρίσκεται δεξιά του =. Αυτός είναι ουσιαστικά και ο λόγος που αναφερόμαστε σε ανάθεση και όχι ισότητα. Το περιεχόμενο της μεταβλητής που υπήρχε πριν την παραπάνω εντολή χάνεται.

Προσοχή: Το σύμβολο = δεν σημαίνει απλή ισότητα, όπως στα μαθηματικά, αλλά αντικατάσταση.

Παρατηρήσεις:

1. Σε κάθε εντολή ανάθεσης πρέπει οπωσδήποτε οι μεταβλητές που βρίσκονται στο δεξιό μέρος της (εφ' όσον υπάρχουν να έχουν ήδη πάρει τιμή μέσω είτε μιας άλλης εντολής ανάθεσης είτε κάποιας εντολής εισόδου).
2. Σε μια εντολή ανάθεσης είναι δυνατόν η μεταβλητή που βρίσκεται στο αριστερό μέρος να υπάρχει και στο δεξιό μέρος.
3. Απαγορεύεται στο αριστερό μέρος της εντολής ανάθεσης η ύπαρξη οποιασδήποτε παράστασης (που να περιέχει μία ή περισσότερες μεταβλητές) παρά μόνον μία και μόνο μεταβλητή χωρίς την ύπαρξη ουδεμίας πράξης.
4. Η μεταβλητή και η έκφραση θα πρέπει να είναι του ίδιου τύπου (αριθμητικού, λογικού, χαρακτήρας)

Προσοχή: Εκείνο που αξίζει να παρατηρήσει κανείς είναι ότι στην περίπτωση του αριθμητικού τύπου μπορεί ο τύπος της μεταβλητής να είναι διαφορετικός (π.χ. ακέραιος) από τον τύπο της αριθμητικής έκφρασης (π.χ. πραγματικός). Ο τύπος όμως της μεταβλητής δεν θα επηρεάσει τον τρόπο υπολογισμού της αριθμητικής έκφρασης.

Αν το αποτέλεσμα του υπολογισμού της αριθμητικής έκφρασης είναι διαφορετικού αριθμητικού τύπου από τη μεταβλητή, γίνεται πρώτα η αλλαγή και έπειτα η αντικατάσταση.

Παράδειγμα 1. Αν έχουμε τις εντολές:

```
Real A
Integer K, L
A = K + 8 – L
```

Οι πράξεις γίνονται μεταξύ ακέραιων και η ανάθεση της υπολογισθείσας τιμής στη μεταβλητή Α, θα γίνει αφού πρώτα μετατραπεί σε πραγματικό αριθμό το αποτέλεσμα.

Παράδειγμα 2. Αν έχουμε τις εντολές:

Real A, B
Integer K
K = A * 5.0 + B

Οι πράξεις γίνονται μεταξύ πραγματικών και η ανάθεση της υπολογισθείσας τιμής στη μεταβλητή K, θα γίνει αφού πρώτα μετατραπεί σε ακέραιο αριθμό το αποτέλεσμα.

Σημείωση: Αν γράψουμε την εντολή **A = 20.25** και η μεταβλητή **A** έχει δηλωθεί ακέραια (**Integer**), στη θέση που αντιστοιχεί η μεταβλητή **A** στη μνήμη τοποθετείται ο αριθμός **20**. Αν χρησιμοποιήσουμε αργότερα τη μεταβλητή **A**, η τιμή της δεν θα είναι **20.25** αλλά ο ακέραιος αριθμός **20**. Δηλαδή, κατά την ανάθεση μιας πραγματικής τιμής (δεκαδικός αριθμός) σε μια ακέραια μεταβλητή θα συμβεί αποκοπή του δεκαδικού μέρους του αριθμού.

3.6.2 Εντολές εισόδου / εξόδου

Οι εντολές εισόδου (Input) μας επιτρέπουν να εισάγουμε τα δεδομένα σ' έναν υπολογιστή.

Οι εντολές εξόδου (Output) μας επιτρέπουν να εμφανίσουμε στην οθόνη ή να τυπώσουμε ή ακόμη ν' αποθηκεύσουμε σ' ένα άλλο μέσο (δίσκο, ταινία κλπ) τ' αποτελέσματα που θα προκύψουν από την εκτέλεση του προγράμματος.

Θα μπορούσαμε, ακόμη πιο γενικά, να πούμε ότι με τις εντολές εισόδου/εξόδου ο υπολογιστής επικοινωνεί με τον εξωτερικό κόσμο και πραγματοποιεί τις ανταλλαγές πληροφοριών.

Η γενική μορφή των εντολών εισόδου/εξόδου είναι:

Read (m, f) "μεταβλητές" (για την είσοδο)

Write (m, f) "μεταβλητές" (για την έξοδο)

Σαν "μεταβλητές" θεωρούνται μια σειρά από μεταβλητές, χωρισμένες μεταξύ τους με υποδιαστολές και f είναι ένας ακέραιος αριθμός που αντιστοιχεί στην ετικέτα μιας εντολής προδιαγραφών **Format** (§3.6.3).

Το m είναι ένας αριθμός ο οποίος δηλώνει το μέσο από το οποίο θα γίνει η είσοδος δεδομένων ή το μέσο στο οποίο θα γίνει η έξοδος των αποτελεσμάτων Όταν μια εντολή **Read** συνοδεύεται από την τιμή m=5 σημαίνει ότι έχουμε είσοδο δεδομένων από το πληκτρολόγιο. Ενώ όταν μια εντολή **Write** συνοδεύεται από την τιμή m=6 σημαίνει ότι έχουμε έξοδο στην οθόνη των αποτελεσμάτων.

Μια εντολή **Read** γενικά, επιτρέπει να μεταφερθούν οι τιμές που δίνονται από το πληκτρολόγιο ή βρίσκονται σε κάποιο μαγνητικό ή οπτικό μέσο και να ανατεθούν στις αντίστοιχες μεταβλητές σύμφωνα με τον τρόπο που ορίζει η συνοδεύουσα εντολή προδιαγραφών **Format**.

Ενώ μια εντολή **Write** επιτρέπει να εμφανιστούν στην οθόνη ή να τυπωθούν πάνω στο χαρτί του εκτυπωτή ή ακόμη να σωθούν σε κάποιο μαγνητικό ή οπτικό μέσο, οι τιμές των

μεταβλητών με την ίδια σειρά που βρίσκονται μέσα στην λίστα των μεταβλητών σύμφωνα πάντα με τον τρόπο που ορίζει η εντολή προδιαγραφών **Format**.

Η γλώσσα **Fortran** διαθέτει επίσης και εναλλακτικές απλές μορφές σύνταξης των εντολών εισόδου/εξόδου (στις οποίες δεν χρησιμοποιείται η εντολή προδιαγραφών **Format**, ούτε ο αριθμός της μονάδας εισόδου/εξόδου (5 ή 6) αλλά στη θέση τους τοποθετείται ένα αστεράκι (*). Σ' αυτή την περίπτωση, αυτόματα, ο υπολογιστής θεωρεί σαν μονάδα εισόδου των δεδομένων μόνο το πληκτρολόγιο και σαν μονάδα εξόδου των αποτελεσμάτων μόνο την οθόνη. Όσον αφορά τις προδιαγραφές των τιμών, αυτές εισάγονται ή εμφανίζονται με τη μεγαλύτερη δυνατή ακρίβεια που μπορεί να τις επεξεργαστεί ο υπολογιστής.

Η μορφή των εναλλακτικών εντολών εισόδου/εξόδου είναι:

Read (*, *) “μεταβλητές” (για την είσοδο)

Write (*, *) “μεταβλητές” (για την έξοδο)

Επειδή η **Fortran 90/95** είναι προσανατολισμένη στην άμεση μορφή επεξεργασίας ενός προγράμματος (Interactive mode) όπου δεν απαιτούνται ειδικές μορφές εισόδου των δεδομένων ή εξόδου των αποτελεσμάτων, γι αυτό και προτείνει μια πιο απλή γραφή των εντολών εισόδου/εξόδου στην οποία ο υπολογιστής θεωρεί σαν μονάδα εισόδου των δεδομένων μόνο το πληκτρολόγιο και σαν μονάδα εξόδου των αποτελεσμάτων μόνο την οθόνη.

Η απλή εντολή εισόδου γράφεται:

Read f, “μεταβλητές”

ή

Read *, “μεταβλητές”

όταν δεν χρησιμοποιείται η εντολή προδιαγραφών **Format**.

Ενώ η απλή εντολή εξόδου γράφεται:

Print f, “μεταβλητές”

ή

Print *, “μεταβλητές”

όταν δεν χρησιμοποιείται η εντολή προδιαγραφών **Format**.

3.6.3 Η εντολή προδιαγραφών Format

Η εντολή **Format** είναι μία εντολή προδιαγραφών της **Fortran** που **δεν εκτελείται**, δηλαδή μπορεί να βρίσκεται σε κάθε σημείο του προγράμματος (αρχή, μέση ή τέλος) χωρίς να επηρεάζει την ομαλή λειτουργία του.

Συνήθως, τοποθετείται ακριβώς μετά την εντολή **Read** ή **Write** στην οποία αντιστοιχεί. Όταν όμως χρησιμοποιείται η ίδια εντολή **Format** από πολλές εντολές **Read** ή **Write** του ιδίου προγράμματος, για λόγους ομοιομορφίας και πιο εύκολης αναγνώρισης, τοποθετούνται όλες οι εντολές **Format** μαζί ή στο τέλος ή στην αρχή του προγράμματος.

Ο γενικός τύπος μιας εντολής **Format** είναι

m **Format**(προδιαγραφές)

όπου m είναι ένας ακέραιος αριθμός το πολύ πενταψήφιος (ετικέτα), που βρίσκεται στις πρώτες 5 θέσεις της γραμμής της εντολής προδιαγραφών **Format**, και αντιστοιχεί στον ίδιο αριθμό που θα τοποθετήσουμε στην εντολή **Read** ή **Write**.

Οι προδιαγραφές τοποθετούνται πάντα μέσα σε παρενθέσεις και χρησιμεύουν:

- Σαν ενδείξεις για την είσοδο/έξοδο των τιμών. Π.χ, πόσα κενά διαστήματα (ή γραμμές) θα χωρίζουν τις τιμές των μεταβλητών (ή τις γραμμές) που τυπώνονται κ.τ.λ.
- Για τον προσδιορισμό της μετατροπής των πληροφοριών (Από την δυαδική μορφή του περιεχομένου της μνήμης στη δεκαδική μορφή και αντίστροφα).

Στην πρώτη κατηγορία ανήκουν τα **Format** τίτλων και στη δεύτερη τα αριθμητικά **Format**.

Πριν προχωρήσουμε στην ανάπτυξη των προδιαγραφών της εντολής **Format** ας περιγράψουμε σύντομα, το βασικό μηχανισμό λειτουργίας.

Πρώτα μεταφέρεται (είτε για είσοδο είτε για έξοδο) η πρώτη κατά σειρά μεταβλητή μεταξύ αυτών που αναφέρονται στη λίστα των μεταβλητών μιας εντολής **Read** ή **Write** σύμφωνα με την πρώτη προδιαγραφή της αντίστοιχης εντολής.

Στη συνέχεια η δεύτερη μεταβλητή, σύμφωνα με τη δεύτερη προδιαγραφή και συνεχίζεται η ίδια διαδικασία μέχρι το τέλος της λίστας των μεταβλητών.

Αν το πλήθος των προδιαγραφών δεν είναι ακριβώς το ίδιο με το πλήθος των μεταβλητών μιας εντολής **Read** ή **Write** τότε:

- Αν το πλήθος των προδιαγραφών είναι μεγαλύτερο από το πλήθος των μεταβλητών, οι επί πλέον προδιαγραφές της εντολής **Format** που βρίσκονται μετά αγνοούνται.
- Αν όμως το πλήθος των προδιαγραφών είναι μικρότερο από το πλήθος των μεταβλητών που θέλουμε να διαβάσουμε ή να εμφανίσουμε, τότε οι προδιαγραφές ξαναχρησιμοποιούνται, προκαλώντας λάθη στην εκτέλεση του προγράμματος.

3.6.3.1 Format τίτλων

Σαν **Format** τίτλων χαρακτηρίζονται εκείνα τα **Format** που οι προδιαγραφές τους δεν ορίζουν μεταφορές αριθμητικών τιμών αλλά χρησιμεύουν κυρίως για την εμφάνιση μηνυμάτων στην οθόνη ή διευκολύνουν την αναγνωσιμότητα των αποτελεσμάτων.

Στη Fortran έχουμε τις ακόλουθες προδιαγραφές για τα **Format** τίτλων:

1. Τα κενά διαστήματα (nX)
   - Κατά την έξοδο, nX σημαίνει ότι θα έχουμε n κενά διαστήματα (κενούς χαρακτήρες) σε μία γραμμή .
   - Κατά την ανάγνωση, nX σημαίνει ότι n χαρακτήρες θα αγνοηθούν.

2. Την αλλαγή της γραμμής (/).

   - Κατά την έξοδο των αποτελεσμάτων μια πλάγια γραμμή (slash) σημαίνει πως σταματούν να εμφανίζονται ή να τυπώνονται σ' αυτή τη σειρά τα αποτελέσματα και η συνέχεια μεταφέρεται στην αρχή της αμέσως επόμενης σειράς.
   - Κατά την ανάγνωση των δεδομένων, σημαίνει πως η συνέχεια για να διαβαστούν τα δεδομένα μεταφέρεται στην αμέσως επόμενη σειρά των δεδομένων.

   Όταν θέλουμε να προχωρήσουμε περισσότερες σειρές χρησιμοποιούμε περισσότερες πλάγιες γραμμές (//...///).

3. Εμφάνιση χαρακτήρων (κειμένου) (Aw)

   Χρησιμοποιείται για μεταβλητές χαρακτήρων ή αλυσίδων χαρακτήρων και δηλώνει:

   - κατά την έξοδο, ότι θα πρέπει να τοποθετήσουμε τους χαρακτήρες που θέλουμε να εμφανίσουμε σε w θέσεις ή σε 1 θέση, αν το n απουσιάζει.
   - κατά την είσοδο, ότι θα πρέπει να διαβάσουμε τους χαρακτήρες που βρίσκονται στις πρώτες w θέσεις ή στην πρώτη θέση, αν το n απουσιάζει.

3.6.3.2 Αριθμητικά Format

Σαν αριθμητικά **Format** χαρακτηρίζονται τα **Format** που συνεπάγονται μεταφορές αριθμητικών τιμών.

1. Μεταφορά ακέραιας μεταβλητής (wI)

   Χρησιμοποιείται για ακέραιες μεταβλητές και w είναι ο αριθμός που δείχνει το σύνολο των θέσεων (στηλών) οι οποίες θα χρησιμοποιηθούν για να εμφανιστεί η αριθμητική τιμή της μεταβλητής ή ο αριθμός των θέσεων (στηλών) όπου βρίσκεται η τιμή η οποία πρόκειται να διαβαστεί.

   Κατά την έξοδο το πρόσημο - (μείον) τυπώνεται πάντα ενώ το πρόσημο + (συν) πάντοτε εννοείται. Αν η τιμή του n είναι μικρότερη από τα σημαντικά ψηφία του αριθμού, τότε στη θέση του αριθμού εμφανίζονται w-φορές αστεράκια (*). Αντίθετα, όταν ο αριθμός των σημαντικών ψηφίων είναι μικρότερος από w-1, οι χαρακτήρες που βρίσκονται αριστερά του αριθμού συμπληρώνονται με τόσα κενά διαστήματα, όσα χρειάζονται για να εξαντληθούν οι n θέσεις.

   Κατά την είσοδο τα δεδομένα πρέπει να είναι: είτε ακέραια ψηφία μεταξύ 0 και 9, είτε λευκά (κενά διαστήματα) που θεωρούνται σαν μηδενικά, είτε ακόμη τα πρόσημα + και -. Κάθε άλλος χαρακτήρας θα δώσει ένα διαγνωστικό λάθους. Η αριθμητική τιμή πρέπει να καταλαμβάνει το δεξιότερο τμήμα των w στηλών, με το αρνητικό πρόσημο (-) να προηγείται στους αρνητικούς αριθμούς. Μια μετατόπιση προς τα αριστερά κατά μία θέση, συνεπάγεται την ανάγνωση ενός αριθμού 10 φορές μεγαλύτερου.

2. Μεταφορά πραγματικής μεταβλητής (Fw.d)

   Χρησιμοποιείται για πραγματικές μεταβλητές, όπου:

w ο συνολικός αριθμός των θέσεων που καταλαμβάνει ο αριθμός είτε κατά την ανάγνωση είτε κατά την εμφάνιση (συμπεριλαμβανομένης και της υποδιαστολής) και

d ο αριθμός των ψηφίων μετά την υποδιαστολή. Πρέπει πάντα να ισχύει w > d.

Κατά την έξοδο ο αριθμός εμφανίζεται με τη συνηθισμένη δεκαδική μορφή της αριθμητικής. Πρέπει όμως να ξέρουμε εκ των προτέρων την τάξη μεγέθους του ακέραιου μέρους γιατί, αν το ακέραιο μέρος περιλαμβάνει περισσότερους από w-d-1 χαρακτήρες, τότε αντί του αριθμού εμφανίζονται w-φορές αστεράκια. Επίσης, αν ο αριθμός των ψηφίων του δεκαδικού μέρους είναι μικρότερος από d, τότε τα ψηφία του δεκαδικού μέρους συμπληρώνονται με μηδενικά έτσι ώστε ο αριθμός των ψηφίων του δεκαδικού μέρους να γίνει d. Τέλος ο αριθμός καταλαμβάνει το δεξιότερο τμήμα της ζώνης των w χαρακτήρων, το πρόσημο - τυπώνεται πάντα, ενώ το πρόσημο + εννοείται.

Κατά την είσοδο ο αριθμός πρέπει να βρίσκεται μέσα σε μία ζώνη w στηλών όπου υπάρχει μια σχετική ελευθερία. Αν υπάρχει ανάμεσα στα αριθμητικά ψηφία η τελεία (σημείο υποδιαστολής), τότε ο αριθμός μεταφέρεται στη μεταβλητή που αντιστοιχεί όπως ακριβώς έχει, δηλαδή, ανεξάρτητα της προδιαγραφής του **Format**. Όταν ανάμεσα στα αριθμητικά ψηφία δεν υπάρχει η τελεία, τότε από δεξιά προς τα αριστερά τα d ψηφία θεωρούνται σαν το δεκαδικό μέρος του αριθμού και τα w-d ψηφία που προπορεύονται σαν το ακέραιο μέρος του αριθμού.

3.6.4 Άλλες βασικές εντολές της γλώσσας Fortran

3.6.4.1 Η πρώτη εντολή ενός προγράμματος

Η πρώτη γραμμή κάθε προγράμματος **Fortran** συνιστάται να είναι μια εντολή που δεν εκτελείται και ούτε είναι υποχρεωτική. Είναι η εντολή **PROGRAM** και αν υπάρχει, πρέπει να βρίσκεται πριν από κάθε άλλη εντολή του προγράμματος.

Η εντολή αυτή γράφεται:

PROGRAM όνομα

όπου "όνομα" είναι το όνομα που δίνουμε στο πρόγραμμα και το οποίο ακολουθεί τους ίδιους κανόνες σύνταξης όπως και το συμβολικό όνομα μιας μεταβλητής.

Χρησιμεύει για να δίνει κάποιο όνομα στο κυρίως πρόγραμμα.

3.6.4.2 Η εντολή τέλος ενός προγράμματος

Η εντολή τέλος ενός προγράμματος γράφεται:

END

Αυτή η εντολή δείχνει στο μεταγλωττιστή (Compiler) ότι το κείμενο του προγράμματος τελείωσε και πρέπει ν' αρχίσει η εκτέλεσή του. Είναι μία εντολή που δεν εκτελείται, δεν μπορεί να πάρει ετικέτα και αποτελεί υποχρεωτικά την τελευταία γραμμή του προγράμματος.

Προαιρετικά, μπορεί να προστεθεί και η λέξη **PROGRAM** δίπλα στην εντολή **END** για να δείχνει ότι αποτελεί τέλος του κυρίως προγράμματος και όχι κάποιου υποπρογράμματος.

Προσοχή: Πρέπει να υπάρχει μόνο μία εντολή **END** σε ένα πρόγραμμα.

3.6.4.3 Η εντολή τέλος εκτέλεσης του προγράμματος

Η εντολή τέλος εκτέλεσης των πράξεων του προγράμματος γράφεται:

STOP

και σημαίνει ότι τελείωσε το πρόγραμμα και ο έλεγχος επιστρέφει στο λειτουργικό σύστημα.

Η εντολή αυτή είναι προαιρετική και μπορεί να τοποθετηθεί σε κάθε σημείο ενός προγράμματος. Επειδή είναι μία εντολή που εκτελείται, μπορεί να πάρει και μία ετικέτα. Έτσι μπορούμε μέσα σ' ένα πρόγραμμα **Fortran** να έχουμε περισσότερες από μία εντολές **STOP**.

3.7 Παραδείγματα προγραμμάτων

Παράδειγμα 1: Να γραφεί πρόγραμμα το οποίο θα διαβάζει μια τιμή που θα δηλώνει το μήκος της ακτίνας ενός κύκλου και στη συνέχεια θα υπολογίζει το εμβαδόν του κύκλου.

```
      Program Area_Circle

C     Το παρακάτω πρόγραμμα υπολογίζει το εμβαδόν ενός κύκλου δοθέντος
C     της ακτίνας του
C     R=ακτίνα κύκλου, E=εμβαδόν κύκλου

C     Ακύρωση όλων τον τύπων των μεταβλητών και των σταθερών
C     Όλες οι μεταβλητές και οι σταθερές θα δηλώνονται
      Implicit none
C     Δήλωση σταθερών
      Real Pi
      Parameter (Pi=3.14159)
C     Δήλωση μεταβλητών
      Real E,R

C     Διάβασμα δεδομένων
      Write(*,*) 'Radius='
      Read(*,*) R

C     Υπολογισμός εμβαδού
      E=2*Pi*R

C     Εκτύπωση αποτελεσμάτων
      Write(*,10)'Area Circle=',E
10    Format(A, F10.5)

      Stop
      End
```

Παράδειγμα 2: Θεωρούμε ότι ο κορμός ενός δέντρου είναι κυλινδρικός. Συνεπώς, ο όγκος του δέντρου V αν γνωρίζουμε την περίμετρό του P και το ύψος του H θα δίνεται από τον τύπο:

$$V = \frac{P^2 \times H}{4 \times \pi}$$

όπου π = 3. 14159. Να γραφεί πρόγραμμα που να υπολογίζει τον όγκο του δέντρου να γνωρίζουμε την περίμετρό του και το ύψος του.

```
      Program Volume_Tree

C     Το παρακάτω πρόγραμμα υπολογίζει τον όγκο ενός δέντρου αν
C     γνωρίζουμε την περίμετρό του και το ύψος του
C     V=όγκος, P=περίμετρος, H=ύψος

C     Ακύρωση όλων τον τύπων των μεταβλητών και των σταθερών
C     Όλες οι μεταβλητές και οι σταθερές θα δηλώνονται
      Implicit none
C     Δήλωση σταθερών
      Real Pi
      Parameter (Pi=3.14159)
C     Δήλωση μεταβλητών
      Real V,P,H

C     Διάβασμα δεδομένων
      Write(*,*) 'Perimeter='
      Read(*,*) P
      Write(*,*) 'Height='
      Read(*,*) H

C     Υπολογισμός όγκου
      V=(P**2*H)/(4*Pi)

C     Εκτύπωση αποτελεσμάτων
      Write(*,10)'Volume Tree=',V
10    Format(A, F10.5)

      Stop
      End
```

Παράδειγμα 3: Έστω ότι έχουμε μια δασική έκταση εμβαδού Α. Έστω επίσης ότι ο ρυθμός αναδάσωσης είναι Ε. Τότε το ποσό της έκτασης αυτής μετά από Ν χρόνια θα είναι Τ και θα δίνεται από τον τύπο:

$$T = A \times (1 + E)^N .$$

Να γραφεί πρόγραμμα που να υπολογίζει τη συνολική έκταση της δασικής περιοχής μετά από Ν χρόνια, αν είναι γνωστός ο ρυθμός αναδάσωσης και η αρχική δασική έκταση.

```
      Program Area

C     Το παρακάτω πρόγραμμα υπολογίζει την έκταση δασικής περιοχής μετά
C     από Ν χρόνια αν γνωρίζουμε τον ρυθμό αναδάσωσης και την αρχική
C     δασική έκταση
C     Τ=τελική έκταση, Α=αρχική έκταση, Ε=ρυθμός αναδάσωσης, Ν=χρόνια

C     Ακύρωση όλων τον τύπων των μεταβλητών
C     Όλες οι μεταβλητές θα δηλώνονται
      Implicit none
C     Δήλωση μεταβλητών
      Real T, A, E, N

C     Διάβασμα δεδομένων
      Write(*,*) 'Years='
      Read(*,*) N
      Write(*,*) 'Initial Area='
      Read(*,*) A
      Write(*,*)'Rate='
      Read(*,*) E


C     Υπολογισμός τελικής έκτασης
      T=A*(1+E)**N

C     Εκτύπωση αποτελεσμάτων
      Write(*,10)'Final Area=',T
10    Format(A, F10.5)

      Stop
      End
```

Παράδειγμα 4: Αν δανειστούμε ένα ποσό A με επιτόκιο T = E% και υποχρεωθούμε να το ξεχρεώσουμε σε N χρόνια, τότε το ποσό της μηνιαίας δόσης M θα δίνεται από τον τύπο:

$$M = \frac{A \times T}{12} \times \frac{(1+T)^N}{(1+T)^N - 1}$$

όπου T = E/100. Να γραφεί πρόγραμμα που να υπολογίζει τη μηνιαία δόση για τις διάφορες τιμές των A, E και N.

```
      Program Monthly_Payment

C     Το παρακάτω πρόγραμμα υπολογίζει την μηνιαία δόση ενός δανείου για ποσό A
C     το οποίο θα εξοφληθεί σε N και με επιτόκιο E
C     A=κεφάλαιο δανεισμού, M=μηνιαία δόση, E=επιτόκιο δανεισμού
C     N=χρόνια αποπληρωμής

C     Ακύρωση όλων τον τύπων των μεταβλητών
C     Όλες οι μεταβλητές θα δηλώνονται
      Implicit none
C     Δήλωση μεταβλητών
      Real A, M, E, N, T

C     Διάβασμα δεδομένων
      Write(*,*) 'Initial Value='
      Read(*,*) A
      Write(*,*) 'Interest Rate='
      Read(*,*) E
      Write(*,*)'Years='
      Read(*,*) N

C     Υπολογισμός επιτοκίου
      T=E/100
C     Υπολογισμός μηνιαίας δόσης
      M=((A*T)/12)*((1+T)**N)/((1+T)**N-1)

C     Εκτύπωση αποτελεσμάτων
      Write(*,10)Monthly Payment=',M
10    Format(A, F10.5)

      Stop
      End
```

Κεφάλαιο 4ο: Εντολές επιλογής

Μέχρι τώρα παρατηρήσαμε ότι τα προβλήματα που αντιμετωπίσαμε είχαν σειριακή κίνηση, δηλαδή η μία εντολή εκτελούνταν μετά την άλλη. Στην πραγματικότητα πολύ λίγα προβλήματα μπορούν να επιλυθούν με τη σειριακή χρήση εντολών ανάθεσης και εντολών εισόδου / εξόδου. Για παράδειγμα αν θέλουμε να υπολογίσουμε την τετραγωνική ρίζα μιας έκφρασης, πρέπει να ελέγξουμε, πριν τον υπολογισμό της, αν η έκφραση αυτή είναι θετική. Αποφυγή τέτοιων ελέγχων θα οδηγούσε σε λάθη εκτέλεσης από τον υπολογιστή. Στα περισσότερα προβλήματα, πλέον, συνηθισμένη περίπτωση είναι να λαμβάνονται κάποιες αποφάσεις με βάση κάποια κριτήρια, που μπορεί να είναι διαφορετικά για κάθε διαφορετικό στιγμιότυπο ενός προβλήματος. Ακόμα και οι καθημερινές μας ενέργειες περιέχουν αυτή τη διαδικασία επιλογής με βάση κάποια κατάσταση. Για παράδειγμα, το πρόβλημα της προετοιμασίας μας για έξοδο σχετίζεται με τις καιρικές συνθήκες. Έτσι λέμε ότι, «αν βρέχει, θα πάρω ομπρέλα, αλλιώς, αλλιώς θα πάρω καπέλο και γυαλιά ηλίου». Η συνθήκη εδώ είναι το «αν βρέχει», ενώ η απόφαση είναι είτε να πάρω την «ομπρέλα» είτε το «καπέλο και γυαλιά ηλίου» με βάση την «τιμή» της συνθήκης.

4.1 Λογικές Συνθήκες

Γενικά η διαδικασία της επιλογής περιλαμβάνει τον έλεγχο κάποιας συνθήκης που μπορεί να έχει δύο τιμές (True ή False) και ακολουθεί η απόφαση εκτέλεσης κάποιας ενέργειας με βάση την τιμή της λογικής αυτής συνθήκης.

Για την σύνταξη μιας λογικής συνθήκης χρησιμοποιούνται σταθερές, μεταβλητές, αριθμητικές εκφράσεις, συγκριτικοί και λογικοί τελεστές, καθώς και παρενθέσεις. Διακρίνουμε δύο κύριες μορφές λογικών συνθηκών: τις απλές και τις σύνθετες.

Απλές Λογικές Συνθήκες:

<Εκφραση1> <Συγκριτικός τελεστής> <Εκφραση2>

Όπου Έκφραση1, Έκφραση2 είναι είτε μια σταθερά είτε μια μεταβλητή είτε μια αριθμητική παράσταση.

Ένας **συγκριτικός τελεστής** στη **Fortran 77** γράφεται με δύο γράμματα του λατινικού αλφαβήτου που τοποθετούνται ανάμεσα σε δύο τελείες. Τα δύο αυτά γράμματα, για την εύκολη απομνημόνευση και χρήση, αποτελούν συντμήσεις των σχέσεων που εκφράζουν και σχηματίζονται από τα αρχικά γράμματα των αντίστοιχων λέξεων (στην αγγλική γλώσσα). Με τη **Fortran 90/95** έχουν προστεθεί εναλλακτικά και σύμβολα για τους (6) έξι **συγκριτικούς τελεστές** όπως εμφανίζονται στον παρακάτω πίνακα.

Τελεστής Fortran 77	Έκφραση στην Αγγλική γλώσσα	Τελεστής Fortran 90/95	Σημασία
.GT.	Greater Than	>	Μεγαλύτερο
.GE.	Greater Equal	>=	Μεγαλύτερο ή ίσο
.LT.	Less Than	<	Μικρότερο
.LE.	Less Equal	<=	Μικρότερο ή ίσο
.EQ.	Equal	= =	Ίσο
.NE.	Not Equal	/=	Διάφορο

Αρχικά υπολογίζονται οι τιμές των εκφράσεων που βρίσκονται από τη μία και την άλλη πλευρά του συγκριτικού τελεστή, γίνεται έλεγχος και όταν το αποτέλεσμα ικανοποιεί τη συνθήκη που εκφράζεται από τον συγκριτικό τελεστή τότε η συνθήκη είναι αληθής (True), διαφορετικά είναι ψευδής (False).

Παρατηρήσεις:

1. Η σύγκριση μεταξύ δύο αριθμών γίνεται με τον προφανή τρόπο είτε είναι ακέραιοι είτε πραγματικοί.
2. Η σύγκριση ατομικών χαρακτήρων στηρίζεται στην αλφαβητική σειρά, για παράδειγμα ο χαρακτήρας “a” θεωρείται μικρότερος από το “b”.
3. Η σύγκριση αλφαριθμητικών δεδομένων βασίζεται στη σύγκριση χαρακτήρα προς χαρακτήρα σε κάθε θέση μέχρις ότου να βρεθεί κάποια διάφορα. Για παράδειγμα η τιμή “LessEqual” είναι μικρότερη από την τιμή “LessThan” αφού το γράμμα E προηγείται του γράμματος T.
4. Η σύγκριση λογικών δεδομένων έχει έννοια μόνο στην περίπτωση του ίσου και του διάφορου, αφού οι τιμές που μπορούν να έχουν είναι αληθής και ψευδής.

Σύνθετες Λογικές Συνθήκες:

<Λογική Συνθήκη 1> <Λογικός Τελεστής> <Λογική Συνθήκη 2>

Όπου Λογική Συνθήκη1, Λογική Συνθήκη2 είναι λογικές συνθήκες (απλές ή σύνθετες), οι οποίες μπορούν να πάρουν δύο τιμές, True ή False, και συνδέονται με έναν **λογικό τελεστή**. Οι **λογικοί τελεστές** που χρησιμοποιούνται στην Fortran παρουσιάζονται στον παρακάτω πίνακα:

Τελεστής Fortran (77 ή 90/95)	Σημασία
.AND.	ΚΑΙ (Σύζευξη)
.OR.	Ή (Διάζευξη)
.NOT.	ΟΧΙ (Άρνηση)

Αρχικά υπολογίζονται οι τιμές των λογικών συνθηκών (True ή False) που βρίσκονται από τη μία και την άλλη πλευρά του λογικού τελεστή, εκτελείται η λογική πράξη που ορίζεται από τον λογικό τελεστή.

Οι λογικές πράξεις ορίζονται από τον παρακάτω πίνακα αληθείας:

ΛΣ1	**ΛΣ2**	**ΛΣ1 ΚΑΙ ΛΣ2**	**ΛΣ1 Ή ΛΣ2**	**ΟΧΙ ΛΣ1**	**ΟΧΙ ΛΣ2**
True	True	True ς	True	False	False
True	False	False	True	False	True ς
False	True	False	True	True	False
False	False	False	False	True	True

4.2 Η εντολή If Λογικό

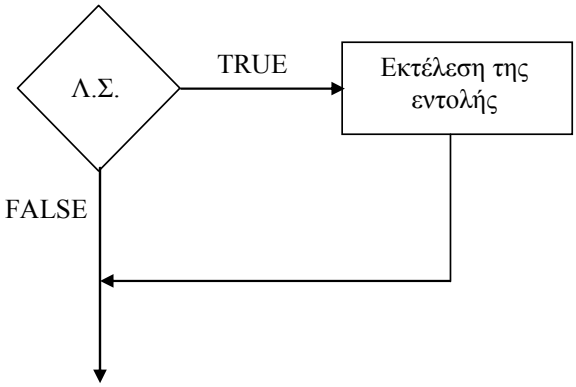
Η γενική μορφή αυτής της εντολής είναι:

If (Λογική Συνθήκη) **εντολή**

όπου **εντολή** μπορεί να είναι μια οποιαδήποτε άλλη εντολή της γλώσσας Fortran εκτός από ένα άλλο **IF** ή μια εντολή **DO**.

Αν η **λογική συνθήκη** είναι αληθής, τότε θα εκτελεστεί η **εντολή** που βρίσκεται γραμμένη δίπλα, και το πρόγραμμα θα ακολουθήσει κανονικά τη συνέχειά του. Αν όμως η λογική συνθήκη είναι ψευδής, δεν εκτελείται η **εντολή**, αλλά το πρόγραμμα συνεχίζει κανονικά με την εντολή που βρίσκεται αμέσως από κάτω.

Το διάγραμμα που ακολουθεί, παριστά τον τρόπο λειτουργίας της εντολής **If** λογικό, όπου η έκφραση **Λ.Σ.** σημαίνει Λογική Συνθήκη.



Παράδειγμα 1:

```
      REAL A, B
      A = 145.0
      B = SQRT(A-1.0)
      IF ((B+3.0) .EQ. 15.0) A = 169.5
      WRITE (*, 10) A, B
10    FORMAT (2F6.1)
```

Μετά την εκτέλεση των πέντε εντολών θα τυπωθούν οι τιμές **169.5 12.0** γιατί η λογική συνθήκη είναι αληθής και επομένως αλλάζει η τιμή της μεταβλητής Α.

Παράδειγμα 2:

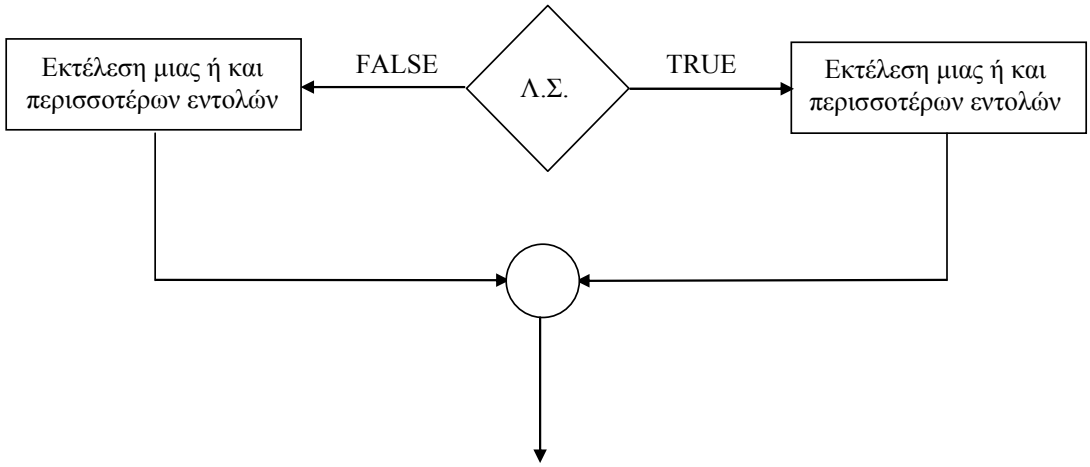
```
      REAL A, B
      A = 145.0
      B = SQRT(A-1.0)
      IF ((B+3.0) .EQ. 15.0) A = 169.5
      WRITE (*, 10) A, B
10    FORMAT (2F6.1)
```

Μετά την εκτέλεση των πέντε εντολών θα τυπωθούν οι τιμές **145.0 12.0** γιατί η λογική συνθήκη δεν είναι αληθής και επομένως δεν αλλάζει η τιμή της μεταβλητής Α.

4.3 Η εντολή Block If

Η εντολή **Block If** είναι από τις πιο ενδιαφέρουσες προσθέσεις που έχουν γίνει στη **Fortran** 77, γιατί προσπαθεί να προσαρμόσει τις αρχές του δομημένου προγραμματισμού στη γλώσσα **Fortran**. Δηλαδή, η εντολή **Block If** βοηθά στην ελαχιστοποίηση των εντολών του τύπου Goto m, οι οποίες αποτελούν και την κυριότερη αιτία λαθών στα μεγάλα προγράμματα, αλλά επίσης επιτρέπει να εκτελούνται οι πράξεις σε μικρές ομάδες. Έτσι, δίνει τη δυνατότητα να γράφονται προγράμματα κατανοητά, αξιόπιστα και αποδοτικά βελτιώνοντας συγχρόνως τον εντοπισμό και τη διόρθωση των λαθών. Επίσης, επιτρέπει τη συμπλήρωση ή αφαίρεση κάποιων εντολών μέσα σ' ένα **Block If** χωρίς τον κίνδυνο απρόβλεπτων λαθών.

Το διάγραμμα που ακολουθεί, παριστά τον τρόπο λειτουργίας της εντολής **Block If**, όπου η έκφραση **Λ.Σ.** σημαίνει Λογική Συνθήκη.



Παρατηρούμε ότι, ανάλογα με το αποτέλεσμα της Λ.Σ., δηλαδή αν είναι αληθής ή ψευδής, θα ακολουθηθεί ένας από τους δύο δρόμους, οπότε θα γίνουν οι πράξεις είτε της μιας είτε της άλλης ομάδας εντολών.

Η εντολή **Block If** συμπληρώνει την εντολή **If** λογικό με τέτοιο τρόπο ώστε σε κάθε αποτέλεσμα του ελέγχου της λογικής συνθήκης (αληθής ή ψευδής) να μπορεί να πραγματοποιηθεί μια ολόκληρη ομάδα (BLOCK) πράξεων και στο τέλος να συνεχίζονται οι πράξεις στην εντολή που ακολουθεί τις δύο ενότητες.

Η γενική μορφή της εντολής είναι:

If (Λογική Συνθήκη) **Then**

"**εντολές 1**"

Else

"**εντολές 2**"

End If

που σημαίνει ότι αν συμβεί η Λογική Συνθήκη, δηλαδή να είναι αληθής τότε θα εκτελεστούν οι "**εντολές 1"** ενώ όταν είναι ψευδής τότε ο έλεγχος θα περάσει στις "**εντολές 2**".

Όταν τελειώσουν οι πράξεις είτε της μιας είτε της άλλης ομάδας των εντολών, ο έλεγχος θα περάσει στην εντολή που βρίσκεται αμέσως μετά την εντολή **End If**. Δηλαδή, η εντολή **End If** δείχνει το σημείο όπου θα τελειώσουν οι πράξεις της εντολής **Block If**.

Η εντολή λέγεται **Block If** επειδή επιτρέπει να γίνονται οι πράξεις σε ανεξάρτητες ομάδες (Block).

Οι διάφορες μορφές σύνταξης της εντολή **Block If** είναι:

1. **If** (Λογική Συνθήκη) **Then**

   "**εντολές 1**"

   End If

   Εδώ, όταν συμβεί η λογική έκφραση να είναι αληθής εκτελούνται οι "**εντολές 1**", διαφορετικά περνάμε κατ' ευθείαν στη συνέχεια του προγράμματος.

   Η μορφή αυτή είναι μοιάζει με την εντολή **If** λογικό.

2. **If** (Λογική Συνθήκη) **Then**

   "**εντολές 1**"

   Else

   "**εντολές 2**"

   End If

   Εδώ, θα εκτελεστούν είτε οι "**εντολές 1**"είτε οι "**εντολές 2**" ανάλογα με το αποτέλεσμα της λογικής συνθήκης.

3. Η γενική μορφή της εντολής **Block If** καλύπτει την επιλογή μιας από δύο εναλλακτικές περιπτώσεις.

   Όταν οι περιπτώσεις είναι περισσότερες από δύο, τότε μπορούν να χρησιμοποιηθούν πολλές εντολές **Block If** η μία μέσα στη άλλη, οι εμφωλευμένες εντολές **Block If**, όπως ονομάζονται.

   Εμφωλευμένα **Block If** ονομάζονται δύο ή περισσότερες εντολές **Block If** που περιέχονται η μία μέσα στην άλλη.

   Δεν υπάρχει γενική μορφή των εμφωλευμένων **Block If**.

   Στη συνέχεια δίνεται ένα παράδειγμα εμφωλευμένων **Block If**.

   If (Λογική Συνθήκη 1) **Then**

   If (Λογική Συνθήκη 2) **Then**

   "**εντολές**"

   End If

   End If

Εδώ, κατ' αρχήν εξετάζεται η Λογική Συνθήκη 1. Αν είναι αληθής εξετάζεται η Λογική Συνθήκη 1και αν είναι και αυτή αληθής τότε εκτελούνται οι "**εντολές**". Σε όλες τις άλλες περιπτώσεις περνάμε στη συνέχεια του προγράμματος χωρίς να εκτελεστούν οι "**εντολές**".

Ένα άλλο παράδειγμα εμφωλευμένων **Block If**, το οποίο συνδυάζει τις δύο προηγούμενες μορφές της εντολής **Block If** είναι:

If (Λογική Συνθήκη 1) **Then**

    If (Λογική Συνθήκη 2) **Then**

        "**εντολές 1**"

    Else

        "**εντολές 2**"

    End If

Else

    If (Λογική Συνθήκη3) **Then**

        "**εντολές 3**"

    End If

End If

Παρατήρηση: Για τον καλλίτερο έλεγχο της αρχής και του τέλους μιας εντολής **Block If** συνιστάται να τοποθετείται το **If**, το **Else** και το **End If** στην ίδια κατακόρυφο ευθεία

4. Όταν οι περιπτώσεις είναι περισσότερες από δύο, τότε μπορούν να χρησιμοποιηθούν εκτός από εμφωλευμένα **Block If** και εντολές **Block If** του τύπου **If – Else If**.

Η γενική μορφή της εντολής **Block If** του τύπου **If – Else If** είναι:

If (Λογική Συνθήκη 1) **Then**

    "**εντολές 1**"

Else If (Λογική Συνθήκη 2) **Then**

    "**εντολές 2**"

Else If (Λογική Συνθήκη3) **Then**

    "**εντολές 3**"

……..

Else

    "**εντολές n+1**"

End If

Αν κάποια συνθήκη i ∈ [1, n] είναι αληθής, τότε εκτελείται η ομάδα (block) εντολών "**εντολές i**", διαφορετικά εκτελείται η ομάδα εντολών "**εντολές n+1**", στη συνέχεια, και στις δύο περιπτώσεις, εκτελείται η εντολή που ακολουθεί την εντολή **End If**. Στην περίπτωση που δεν υπάρχει η εντολή **Else**, τότε, αν όλες οι συνθήκες είναι ψευδείς, εκτελείται η εντολή που ακολουθεί την εντολή **End If**.

4.4 Παραδείγματα

Παράδειγμα 1: Να γραφεί πρόγραμμα που θα διαβάζει δύο ακέραιους αριθμούς a, b και στη συνέχεια θα υπολογίζει και θα εκτυπώνει τον μεγαλύτερο.

```
Program Maximum_Number_1
Implicit None
Integer a, b

Write(*,*) 'Give two integers'
Read(*,*) a, b

If (a.GE.b) Then
   Write(*,*) 'Maximum=',a
Else
   Write(*,*) 'Maximum=',b
End if

End
```

Παράδειγμα 2: Να τροποποιηθεί το προηγούμενο πρόγραμμα έτσι ώστε να εκτυπώνεται και η σχέση των a, b.

```
      Program Maximum_Number_2
C     Με χρήση της εντολής Block If του τύπου If – Else If
      Implicit None
      Integer a, b
      Write(*,*) 'Give two integers'
      Read(*,*) a, b

      If (a.GT.b) Then
         Write(*,*) 'a > b'
         Write(*,*) 'Maximum=',a
      Else if (a.EQ.b) Then
         Write(*,*) 'a = b'
         Write(*,*) 'Maximum=',a
      Else
         Write(*,*) 'a < b'
         Write(*,*) 'Maximum=',b
      End if

      End
```

```
      Program Maximum_Number_3
C     Με χρήση εμφωλευμένων εντολών Block If
      Implicit None
      Integer a, b

      Write(*,*) 'Give two integers'
      Read(*,*) a, b

      If (a.GE.b) Then
         If (a.GT.b) Then
            Write(*,*) 'a > b'
            Write(*,*) 'Maximum=',a
         Else
            Write(*,*) 'a = b'
            Write(*,*) 'Maximum=',a
         End if
      Else
         Write(*,*) 'a < b'
         Write(*,*) 'Maximum=',b
      End if

      End
```

Παράδειγμα 3: Να γραφεί πρόγραμμα που θα διαβάζει έναν ακέραιο αριθμούς a και στη συνέχεια θα υπολογίζει και θα εκτυπώνει το πρόσημό του.

```
      Program Sign_Number_1
C     Με χρήση της εντολής Block If του τύπου If – Else If
      Implicit None
      Integer a, sign

      Write(*,*) 'Give an integer'
      Read(*,*) a

      If (a.GT.0) Then
         sign = 1
         Write(*,*) 'The sign of number', a, 'is', sign
      Else if (a.EQ.0) Then
         sign = 0
         Write(*,*)'The sign of number', a, 'is', sign
      Else
         sign = -1
         Write(*,*)'The sign of number', a, 'is', sign
      End if

      End
```

```
      Program Sign_Number_2
C     Με χρήση εμφωλευμένων εντολών Block If
      Implicit None
      Integer a, sign

      Write(*,*) 'Give an integer'
      Read(*,*) a

      If (a.GE.0) Then
        If (a.GT.0) Then
          sign = 1
          Write(*,*)'The sign of number', a, 'is', sign
        Else
          sign = 0
          Write(*,*)'The sign of number', a, 'is', sign
        End if
      Else
        sign = -1
        Write(*,*)'The sign of number', a, 'is', sign
      End if

      End
```

Παράδειγμα 4: Να γραφεί πρόγραμμα που θα διαβάζει τον βαθμό πτυχίου ενός φοιτητή και θα εμφανίζει τον χαρακτηρισμό του.

```
      Program Graduate

      Implicit None
      Real degree

      Write(*,*) 'Give the degree'
      Read(*,*) degree

      If ((degree.GE.5.0).AND.(degree.LT.6.5) Then
        Write(*,*) 'Kalos'
      Else if ((degree.GE.6.5).AND.(degree.LT.8.5) Then
        Write(*,*)'Poly Kala'
      Else if ((degree.GE.8.5).AND.(degree.LE.10.0) Then
        Write(*,*)'Arista'
      Else
        Write(*,*) 'Lathos dedomena'
      End if

      End
```

Παράδειγμα 5: Να γραφεί πρόγραμμα που θα διαβάζει ένα ποσό θα υπολογίζει και θα εμφανίζει τον συντελεστή φόρου σύμφωνα με τον παρακάτω πίνακα:

Ποσό	Συντελεστής Φόρου
<2.500	0.02
[2.500, 5000)	0.03
>= 5.000	0.04

```
Program Taxis
Implicit None
Real taxrate
Integer poso

Write(*,*) 'Give the poso'
Read(*,*) poso

If (poso.LT.2500) Then
   taxrate = 0.02
Else if (poso.LT.5000) Then
   taxrate = 0.03
Else
   taxrate = 0.04
End if

Write(*,*)'Taxrate=', taxrate

End
```

Παράδειγμα 6: Να γραφεί πρόγραμμα το οποίο θα δέχεται ως είσοδο 2 αριθμούς a, b και στη συνέχεια θα επιλύει την πρωτοβάθμια εξίσωση ax + b = 0.

```
Program Prwtobathmia
Implicit None
Real a, b, x

Write(*,*) 'Give a and b'
Read(*,*) a, b

If (a.NE.0.0) Then
   x = -b/a
   Write(*,*)'x=', x
Else
   If (b.EQ.0.0) Then
      Write(*,*)'Aoristi'
   Else
      Write(*,*)'Adynati'
   End if
End if

End
```

Παράδειγμα 7: Να γραφεί πρόγραμμα το οποίο θα δέχεται ως είσοδο 3 αριθμούς a, b, c και στη συνέχεια θα επιλύει την δευτεροβάθμια εξίσωση $ax^2 + bx + c = 0$.

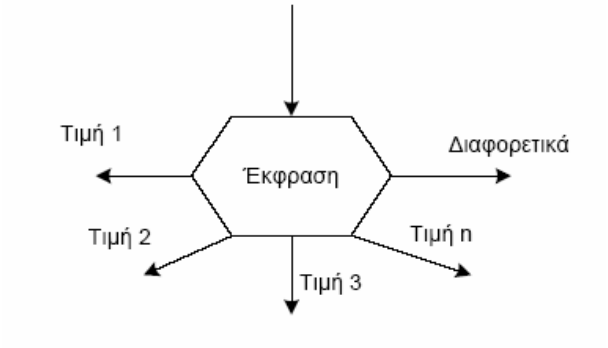
```
Program Deyterobathmia

Implicit None
Real a, b, c, d, x, x1, x2

Write(*,*) 'Give a'
Read(*,*) a
Write(*,*) 'Give b'
Read(*,*) b
Write(*,*) 'Give c'
Read(*,*) c

If (a.NE.0.0) Then
   d = b**2 – 4*a*c
   If (d.GE.0.0) Then
      x1 = (-b + SQRT(d))/(2*a)
      x2 = (-b - SQRT(d))/(2*a)
      Write(*,*) 'x1=', x1
      Write(*,*) 'x2=', x2
   Else
      x1 = -b/(2*a)
      x2 = SQRT(-d)/(2*a)
      Write(*,*) x1, '+I*', x2
      Write(*,*) x1, '-I*', x2
   End if
Else
   If (b.NE.0.0) Then
      x = -c/b
      Write(*,*)'H eksisosi einai prwtobathmia me riza x=', x
   Else
      If (c.EQ.0.0) Then
         Write(*,*)'Aoristi'
      Else
         Write(*,*)'Adynati'
      End if
   End if
End if

End
```

4.5 Η εντολή Select Case

Μια εντολή **Block If**, όπως είδαμε, μπορεί να περιέχει πολλούς ενσωματωμένους ελέγχους (εμφωλευμένα **Block If**, **Block If** τύπου **If – Else If**) με κίνδυνο σε ορισμένα προβλήματα να είναι δύσκολη η ανίχνευση της σωστής λειτουργίας του συνόλου των περιπτώσεων.

Όταν συμβεί να απαιτούνται περισσότερες από 2 επιλογές τότε, συνιστάται να χρησιμοποιούμε την εντολή **Select Case** η οποία έχει εισαχθεί με τη Fortran 90 και επιτρέπει πολλαπλές επιλογές.

Το διάγραμμα που ακολουθεί, παριστά τον τρόπο λειτουργίας μιας εντολής **Select Case**.



Δηλαδή, η εντολή **Select Case** ελέγχει την τιμή της **έκφρασης** και επιλέγει μόνο μια από τις διαφορετικές εξόδους για τη συνέχεια των πράξεων.

Η γενική μορφή της εντολής είναι:

Select Case (έκφραση)
Case (τιμή 1)
"**εντολές 1**"
Case (τιμή 2)
"**εντολές 2**"
……
……
Case (τιμή n)
"**εντολές n**"
Case Default
"**εντολές**"
End Select

Η **έκφραση** μπορεί να είναι μια ακέραια ή λογική μεταβλητή ή χαρακτήρας και υποχρεωτικά όλες οι συγκεκριμένες τιμές που τοποθετούνται δίπλα στη λέξη **Case** πρέπει να είναι και αυτές του ιδίου τύπου.

Ο τρόπος λειτουργίας της εντολής είναι απλός. Κατ' αρχήν υπολογίζεται η τιμή της **έκφρασης** και επιλέγεται να εκτελεστούν οι εντολές μετά από εκείνη τη εντολή **Case** όπου η αντίστοιχη τιμή ταιριάζει με την τιμή της έκφρασης, και στη συνέχεια εκτελείται η επόμενη εντολή που ακολουθεί την εντολή **End Select**.

Όταν υπάρχει η εντολή **Case Default** τότε θα εκτελεστούν οι εντολές που ακολουθούν, μόνο στην περίπτωση που καμία από τις προηγούμενες τιμές δεν συμφωνεί με την τιμή της έκφρασης, και στη συνέχεια εκτελείται η επόμενη εντολή που ακολουθεί την εντολή **End Select**. Αν απουσιάζει η γραμμή **Case Default** τότε εκτελείται απευθείας η επόμενη εντολή που ακολουθεί την εντολή **End Select**.

Στη θέση της τιμής δίπλα στη λέξη CASE μπορούν να τοποθετηθούν οι ακόλουθες παραλλαγές:

Α. Μια συγκεκριμένη τιμή π.χ**. 5** ή **'ΝΑΙ'**

Β. Τιμές μικρότερες ή ίσες από μια τιμή π.χ. **:15** που σημαίνει για όλες τις τιμές της έκφρασης τις μικρότερες ή ίσες με το 15.

Γ. Τιμές μεγαλύτερες ή ίσες από μια τιμή π.χ. **75:** που σημαίνει για όλες τις τιμές της έκφρασης τις μεγαλύτερες ή ίσες με το 75.

Δ. Τιμές που να είναι μεταξύ δύο ορίων π.χ. **15:75** που σημαίνει για όλες τις τιμές της έκφρασης τις μικρότερες του 75 και μεγαλύτερες του 15.

Παράδειγμα 1: Να γραφεί πρόγραμμα (με χρήση της εντολής **Select Case**) που θα διαβάζει ένα ποσό θα υπολογίζει και θα εμφανίζει τον συντελεστή φόρου σύμφωνα με τον παρακάτω πίνακα:

Ποσό	Συντελεστής Φόρου
<2.500	0.02
[2.500, 5000)	0.03
>= 5.000	0.04

```
Program Taxis
Implicit None
Real taxrate
Integer poso

Write(*,*) 'Give the poso'
Read(*,*) poso

Select Case (poso)
   Case (:2499)
      taxrate = 0.02
   Case (2500:4999)
      taxrate = 0.03
   Case Default
      taxrate = 0.04
End Select

Write(*,*)'Taxrate=', taxrate

End
```

Κεφάλαιο 5ο: Εντολές Επανάληψης

Η διαδικασία της επανάληψης είναι ιδιαίτερη συχνή, αφού πλήθος προβλημάτων μπορούν να επιλυθούν με κατάλληλες επαναληπτικές διαδικασίες. Η λογική των επαναληπτικών διαδικασιών εφαρμόζεται στις περιπτώσεις, όπου μια ομάδα εντολών πρέπει να εφαρμοσθεί σε ένα σύνολο περιπτώσεων, που έχουν κάτι κοινό. Για παράδειγμα, όλες οι τράπεζες κάθε εξάμηνο αποδίδουν τόκους των καταθέσεων ταμιευτηρίου. Ο υπολογισμός των τόκων πρέπει να γίνει για όλους τους λογαριασμούς της τράπεζας, άρα η πράξη

$$\text{τόκος} = \text{ποσό} * \text{επιτόκιο}$$

πρέπει να εκτελεσθεί για όλους τους τραπεζικούς λογαριασμούς. Οι επαναληπτικές διαδικασίες μπορεί να έχουν διάφορες μορφές και συνήθως περιέχουν συνθήκες επιλογών.

5.1 Η εντολή Do στη Fortran 77

Επειδή στα μαθηματικά και γενικότερα στις επιστημονικές εφαρμογές, έχουμε πολύ συχνά επανάληψη των ίδιων ακριβώς πράξεων, γι' αυτό και η γλώσσα **Fortran** διαθέτει μια εντολή με την οποία μπορούμε να ελέγχουμε αυτές τις επαναλήψεις.

Ο γενικός τύπος της εντολής είναι:

Do n i = m1, m2, m3

ή πιο αναλυτικά :

Do "ετικέτα" "μεταβλητή" = "κατώτερο όριο", "ανώτερο όριο", "βήμα"

και όπου ισχύουν :

- **n**, είναι η ετικέτα μιας άλλης εντολής που πρέπει να βρίσκεται μετά την εντολή **Do** και δεν πρέπει να είναι μόνο μία άλλη εντολή **Do**.
- **i**, είναι μία ακέραια μεταβλητή που λέγεται και μεταβλητή ελέγχου. Δεν πρέπει ποτέ μέσα σε μια ανακύκλωση να ορίσουμε ξανά την τιμή της. Όταν τελειώσουν οι επαναλήψεις (ανακυκλώσεις) της εντολής **Do** μπορούμε να δώσουμε στη μεταβλητή **i** οποιαδήποτε νέα τιμή θέλουμε.
- **m1**, είναι η αρχική τιμή που θα πάρει η μεταβλητή i πριν από την πρώτη εκτέλεση της εντολής **Do**.
- **m2**, είναι η τελική τιμή που μπορεί να πάρει η μεταβλητή i και να εκτελεστεί η επανάληψη.
- **m3**, είναι το βήμα, δηλαδή η τιμή αύξησης της μεταβλητής i σε κάθε επανάληψη της ανακύκλωσης. Μπορεί να παραληφθεί το βήμα και τότε εννοείται η μονάδα αλλά ποτέ δεν πρέπει να πάρει την τιμή μηδέν.

π.χ. μπορούμε να γράψουμε:

```
      M = 0
      Do 10 K = 1, 15, 1
10    M=M+K
      Write(6, 20) M
20    Format(1X, I5)
```

Εδώ βρίσκουμε και εμφανίζουμε την τιμή του αθροίσματος των 15 πρώτων ακέραιων αριθμών.

Ο μηχανισμός λειτουργίας της εντολής είναι ο εξής :

Οι εντολές που βρίσκονται ανάμεσα στη γραμμή της εντολής **Do** ..., και την εντολή (συμπεριλαμβανομένης και της εντολής αυτής) που έχει ετικέτα τον αριθμό που ακολουθεί τη λέξη **Do**, επαναλαμβάνονται τόσες φορές, όσες χρειάζεται για να πάμε από τον πρώτο δείκτη στο δεύτερο δείκτη με βήμα τον τρίτο δείκτη.

Συνολικά ο αριθμός των ανακυκλώσεων που εκτελούνται είναι:

$$\left[\frac{m_2 - m_1}{m_3}\right] + 1$$

όπου [ ] δηλώνει το ακέραιο μέρος.

Τα m1, m2 , m3 μπορούν να πάρουν μόνο ακέραιες τιμές (στη **Fortran 90/95**) ενώ στη **Fortran 77** μπορούν να πάρουν και πραγματικές τιμές και μπορούν να είναι είτε αριθμητικές σταθερές είτε μεταβλητές αλλά όχι μεταβλητές με δείκτες.

Μια εντολή **Do** εκτελείται από τον υπολογιστή, με τον ακόλουθο τρόπο:

Περίπτωση $1^{η}$: $m_3 > 0$.

Αν $m_1 > m_2$, τότε μεταφερόμαστε εκτός της ανακύκλωσης. Διαφορετικά, η μεταβλητή i παίρνει την αρχική τιμή m_1. Εκτελούνται οι εντολές που βρίσκονται ανάμεσα στη γραμμή της εντολής **Do** ..., και την εντολή (συμπεριλαμβανομένης και της εντολής αυτής) που έχει ετικέτα τον αριθμό που ακολουθεί τη λέξη **Do**. Στη συνέχεια, προσθέτουμε στη μεταβλητή i το βήμα m_3 και ελέγχουμε αν $i \leq m_2$. Αν ικανοποιείται ο έλεγχος επαναλαμβάνεται όλη η προηγούμενη διαδικασία. Αν δεν ικανοποιείται ο έλεγχος τότε μεταφερόμαστε εκτός της ανακύκλωσης.

Περίπτωση $1^{η}$: $m_3 < 0$.

Αν $m_1 < m_2$, τότε μεταφερόμαστε εκτός της ανακύκλωσης. Διαφορετικά, η μεταβλητή i παίρνει την αρχική τιμή m_1. Εκτελούνται οι εντολές που βρίσκονται ανάμεσα στη γραμμή της εντολής **Do** ..., και την εντολή (συμπεριλαμβανομένης και της εντολής αυτής) που έχει ετικέτα τον αριθμό που ακολουθεί τη λέξη **Do**. Στη συνέχεια, προσθέτουμε στη μεταβλητή i το βήμα m_3 και ελέγχουμε αν $i \geq m_2$. Αν ικανοποιείται ο έλεγχος επαναλαμβάνεται όλη η προηγούμενη διαδικασία. Αν δεν ικανοποιείται ο έλεγχος τότε μεταφερόμαστε εκτός της ανακύκλωσης.

Περίπτωση $3^{η}$: $m_3 = 0$.

Δεν υπάρχει τέτοια περίπτωση.

Παραδείγματα:

- **Do** 15 I = 5, 5, 2 Η επανάληψη των εντολών εκτελείται μια φορά για την τιμή I = 5
- **Do** 32 K = 1, 10, 5 Η επανάληψη των εντολών εκτελείται δύο φορές:
  μια για την τιμή K = 1 και
  μια για την τιμή K = 6

- **Do** 35 L = 1, 3, 3 Η επανάληψη των εντολών εκτελείται μια φορά για την τιμή L = 1
- **Do** 64 I = 3, 1, 1 Η επανάληψη των εντολών δεν θα εκτελεσθεί καμία φορά επειδή $m_1 > m_2$ και $m_3 > 0$.
- **Do** 18 IV = -1, 1 Η επανάληψη των εντολών θα εκτελεστεί 3 φορές (εννοείται βήμα = 1):

  μια για IV= -1,

  μία για IV=0 και

  μία για IV=1.

Παρατηρήσεις:

- Μπορούμε μέσα σε μια ανακύκλωση να έχουμε και άλλες ανακυκλώσεις, οι οποίες πρέπει να πληρούν τον κανόνα: Το εύρος κάθε εσωτερικής ανακύκλωσης να βρίσκεται πάντα μέσα στο εύρος της αμέσως εξωτερικής της ανακύκλωσης και να μην χρησιμοποιείται η ίδια μεταβλητή (i) ελέγχου.

- Δύο ή και περισσότερες είτε και όλες οι ανακυκλώσεις μπορούν να έχουν την ίδια τελευταία εντολή.

- Όταν έχουμε πολλές ανακυκλώσεις, τη μια μέσα στην άλλη, πρώτα εκτελείται εκείνη που βρίσκεται στο εσωτερικό και στη συνέχεια μια-μια από το εσωτερικό προς το εξωτερικό εκτελούνται οι υπόλοιπες μέχρις ότου φθάσουμε στην πλέον εξωτερική ανακύκλωση.

5.1.2 Η εντολή Continue

Η εντολή **Continue** είναι μια εντολή που ενώ εκτελείται δεν έχει καμιά αποτελεσματικότητα.

Συνήθως μπαίνει στο τέλος μιας ανακύκλωσης για να αποφύγουμε τις απαγορευμένες εντολές και επειδή μπορεί να πάρει ετικέτα.

Το παράδειγμα, για τον υπολογισμό της τιμής του αθροίσματος των 15 πρώτων ακέραιων αριθμών, μπορεί να γραφτεί τώρα και με τον ακόλουθο τρόπο:

```
      M = 0
      Do 10 K = 1, 15, 1
      M=M+K
10    Continue
      Write(6, 20) M
20    Format(1X, I5)
```

Βλέπουμε λοιπόν ότι η εντολή CONTINUE δεν επηρεάζει το αποτέλεσμα αλλά μπορεί να χρησιμεύσει σαν την τελευταία εντολή μιας ανακύκλωσης.

5.2 Νέες μορφές εντολών επανάληψης (Do) στη Fortran 90/95

Με τη **Fortran 90/95**, για να ελαττωθεί η χρήση των ετικετών και για να προσαρμοσθεί η Fortran στο δομημένο προγραμματισμό, έχει εισαχθεί μια νέα μορφή γραφής της εντολής **Do**, η **Do** – **End Do**, η οποία μπορεί να χρησιμοποιηθεί με τρεις μορφές σύνταξης.

5.2.1 Η πρώτη μορφή της Do – End Do (Do (Exit))

Η γενική μορφή αυτής της εντολής είναι:

Do

Εντολές της Fortran

End Do

Οι **εντολές της FORTRAN** που περιλαμβάνονται σε μια παρόμοια εντολή **Do** θα επαναλαμβάνονται μέχρις ότου συναντηθεί μια εντολή εξόδου από τις επαναλήψεις.

Επομένως, πρέπει να είμαστε πολύ προσεκτικοί στη χρήση αυτής της εντολής για να μην υποπέσουμε σε άπειρες επαναλήψεις (πρόγραμμα χωρίς τέλος).

Για την καλλίτερη και πιο αποτελεσματική αντιμετώπιση παρομοίων προβλημάτων έχει εισαχθεί στην **Fortran 90/95** η εντολή **Exit**.

Η εντολή **Exit** χρησιμεύει για να διακόπτεται η προκαθορισμένη σειρά επανάληψης των εντολών μέσα σε μια ενότητα εντολής **Do**. Δηλαδή, η εντολή **Exit** μεταφέρει αυτόματα τον έλεγχο για τη συνέχιση των πράξεων, έξω από την ενότητα των επαναλαμβανόμενων εντολών.

Μια εντολή **Exit** μπορεί να χρησιμοποιηθεί ΜΟΝΟ μέσα σε οποιαδήποτε εντολή DO και ΟΧΙ σε άλλο σημείο του προγράμματος.

Αν υπάρχουν πολλές ανακυκλώσεις, η μια μέσα στην άλλη, μια εντολή **Exit** σε εσωτερική ανακύκλωση, σταματά τον έλεγχο των ανακυκλώσεων μόνο στη συγκεκριμένη εσωτερική ανακύκλωση και μεταφέρει τον έλεγχο στην αμέσως πιο εξωτερική από αυτήν ανακύκλωση.

Επομένως η γενική μορφή αυτής της εντολής με τη χρήση της εντολής **Exit** είναι:

Do

Εντολές της Fortran

If (Λογική Συνθήκη) **Exit**

End Do

Σημείωση: Παρατηρούμε ότι με τη πρώτη μορφή της **Do – End Do** και με τη χρήση της εντολής **Exit** έχουμε επανάληψη εντολών έως ότου ικανοποιηθεί μια συνθήκη.

Για να μην υποπέσουμε σε άπειρες επαναλήψεις (πρόγραμμα χωρίς τέλος), θα πρέπει Λογική Συνθήκη που οδηγεί στην έξοδο (Exit) από την ανακύκλωση, να αλλάζει σε κάθε επανάληψη τιμή, όπως επίσης και να έχει αρχική τιμή.

Παράδειγμα: Να γραφεί πρόγραμμα το οποίο θα διαβάζει ένα σύνολο ακεραίων αριθμών και θα υπολογίζει το άθροισμά τους έως ότου εμφανιστεί αρνητικός αριθμός.

5.2.2 Η δεύτερη μορφή της Do – End Do (Do While)

Με τη δεύτερη μορφή της **Do – End Do** έχουμε επανάληψη εντολών όσο ικανοποιείται μια συνθήκη.

Η γενική μορφή αυτής της εντολής είναι:

Do While (Λογική Συνθήκη)

Εντολές της Fortran

End Do

Οι **Εντολές της Fortran** που περιλαμβάνονται σε μια εντολή **Do While** θα επαναλαμβάνονται μέχρις ότου η Λογική Συνθήκη γίνει ψευδής. Όσο η τιμή της Λογικής Συνθήκης παραμένει αληθής, θα επαναλαμβάνονται οι ανακυκλώσεις. Επομένως, για να έχει κάποιο ενδιαφέρον η χρήση αυτής της εντολής και για να μην υποπέσουμε σε άπειρες επαναλήψεις (πρόγραμμα χωρίς τέλος), θα πρέπει η τιμή της Λογικής Συνθήκης να αλλάξει τιμή και να γίνει ψευδής κατά τη διάρκεια της εκτέλεσης των **εντολών της Fortran**, καθώς επίσης και να έχει αρχική τιμή. Αν η αρχική τιμή της Λογικής Συνθήκη είναι ψευδής δεν θα εκτελεστεί ούτε μια από τις εντολές δηλαδή, δεν θα γίνει καμία επανάληψη.

Παράδειγμα: Να γραφεί πρόγραμμα το οποίο θα διαβάζει ένα σύνολο ακεραίων αριθμών και θα υπολογίζει το άθροισμά τους έως ότου εμφανιστεί αρνητικός αριθμός. (Με χρήση της εντολής **Do While**).

Παρατηρούμε ότι ένα πρόγραμμα που γράφεται με χρήση της **Do (Exit)** μπορεί να γραφεί και με τη χρήση της **Do While**. Με τη μόνη διαφορά ότι ως Λογική Συνθήκη της **Do While** χρησιμοποιούμε την άρνηση της Λογικής Συνθήκης που μας οδηγεί σε έξοδο σε μία **Do (Exit)**.

5.2.3 Η τρίτη μορφή της Do – End Do (Γενική μορφή)

Με τη τρίτη μορφή της **Do – End Do** έχουμε επανάληψη των εντολών για ένα συγκεκριμένο πλήθος φορών.

Η γενική μορφή της εντολής είναι:

Do i = m1, m2, m3

Εντολές της Fortran

End Do

όπου:

- **i**, είναι μία ακέραια μεταβλητή που λέγεται και μεταβλητή ελέγχου. Δεν πρέπει ποτέ μέσα σε μια ανακύκλωση να ορίσουμε ξανά την τιμή της. Όταν τελειώσουν οι επαναλήψεις (ανακυκλώσεις) της εντολής **Do – End Do** μπορούμε να δώσουμε στη μεταβλητή **i** οποιαδήποτε νέα τιμή θέλουμε.

- **m1**, είναι η αρχική τιμή που θα πάρει η μεταβλητή i πριν από την πρώτη εκτέλεση της εντολής **Do – End Do**.
- **m2**, είναι η τελική τιμή που μπορεί να πάρει η μεταβλητή i και να εκτελεστεί η επανάληψη.
- **m3**, είναι το βήμα, δηλαδή η τιμή αύξησης της μεταβλητής i σε κάθε επανάληψη της ανακύκλωσης. Μπορεί να παραληφθεί το βήμα και τότε εννοείται η μονάδα αλλά ποτέ δεν πρέπει να πάρει την τιμή μηδέν.

Η γενική εντολή **Do – End Do** εκτελείται από τον υπολογιστή, με τον ακόλουθο τρόπο:

Περίπτωση $1^{η}$: $m_3 > 0$.

Αν $m_1 > m_2$, τότε μεταφερόμαστε εκτός της ανακύκλωσης. Διαφορετικά, η μεταβλητή i παίρνει την αρχική τιμή m_1. Εκτελούνται οι **εντολές της Fortran** που βρίσκονται μέσα στην εντολής **Do – End Do**. Στη συνέχεια, προσθέτουμε στη μεταβλητή i το βήμα m_3 και ελέγχουμε αν $i \leq m_2$. Αν ικανοποιείται ο έλεγχος επαναλαμβάνεται όλη η προηγούμενη διαδικασία. Αν δεν ικανοποιείται ο έλεγχος τότε μεταφερόμαστε εκτός της ανακύκλωσης.

Περίπτωση $1^{η}$: $m_3 < 0$.

Αν $m_1 < m_2$, τότε μεταφερόμαστε εκτός της ανακύκλωσης. Διαφορετικά, η μεταβλητή i παίρνει την αρχική τιμή m_1. Εκτελούνται οι **εντολές της Fortran** που βρίσκονται μέσα στην εντολής **Do – End Do**. Στη συνέχεια, προσθέτουμε στη μεταβλητή i το βήμα m_3 και ελέγχουμε αν $i \geq m_2$. Αν ικανοποιείται ο έλεγχος επαναλαμβάνεται όλη η προηγούμενη διαδικασία. Αν δεν ικανοποιείται ο έλεγχος τότε μεταφερόμαστε εκτός της ανακύκλωσης.

Περίπτωση $3^{η}$: $m_3 = 0$.

Δεν υπάρχει τέτοια περίπτωση.

Παράδειγμα: Να εκτυπωθούν οι αριθμοί από το 1 ως το 5.

```
Do i = 1, 5, 1
    Write(*, *) i
End Do
```

Το i παίρνει αρχική τιμή 1. Στη συνέχεια γίνεται ο έλεγχος αν το i (= 1) ≤ 5. Εφόσον η συνθήκη είναι αληθής, εκτυπώνεται το i (δηλαδή το 1). Επειδή το βήμα είναι 1, συνεπώς το i παίρνει τη τιμή i = 1 + 1 = 2. Επαναλαμβάνεται ο έλεγχος αν i (= 2) ≤ 5 κ.τ.λ. Στο τέλος, εφόσον εκτυπωθεί και το 5, η τιμή του i αυξάνει κατά 1, δηλαδή i = 5 + 1 που δεν είναι μικρότερο ή ίσο του 5 και συνεπώς, μεταφέρεται ο έλεγχος εκτός ανακύκλωσης. Σε ενδεχόμενη λοιπόν ερώτηση για την τιμή του i μετά την ανακύκλωση, η σωστή απάντηση θα ήταν 6 και όχι 5 όπως πιθανόν να φαινόταν.

Παρατηρήσεις:

- Το βήμα θα πρέπει να είναι πάντα διαφορετικό από το μηδέν.
- Κατά τη διάρκεια της ανακύκλωσης δεν θα πρέπει να αλλάξουμε τις τιμές των m_1, m_2 και m_3.

- Θα πρέπει να προσέχουμε να μην παραλείπουμε το βήμα όταν είναι αρνητικό π.χ. -1.

- Η μεταβλητή που χρησιμοποιούμε στην ανακύκλωση, καθώς και οι τιμές m_1 (αρχική τιμή), m_2 (τελική τιμή) και m_3 (βήμα), θα πρέπει να είναι ακέραιες, πρώτον, διότι διαφορετικά μπορεί να δημιουργηθούν λάθη αποκοπής και δεύτερον, διότι σε κάποιες εκδόσεις της Fortran χρησιμοποιούνται μόνο ακέραιες μεταβλητές.

- Ο αριθμός των φορών που θα εκτελεστεί μια ανακύκλωση με την γενική μορφή της εντολής **Do – End Do** είναι:

$$\left[\frac{m_2 - m_1}{m_3}\right] + 1$$

όπου [ ] δηλώνει το ακέραιο μέρος.

- Μπορούμε μέσα σε μια ανακύκλωση να έχουμε και άλλες ανακυκλώσεις, οι οποίες πρέπει να πληρούν τον κανόνα: Το εύρος κάθε εσωτερικής ανακύκλωσης να βρίσκεται πάντα μέσα στο εύρος της αμέσως εξωτερικής της ανακύκλωσης και να μην χρησιμοποιείται η ίδια μεταβλητή (i) ελέγχου.

- Όταν έχουμε πολλές ανακυκλώσεις, τη μια μέσα στην άλλη, πρώτα εκτελείται εκείνη που βρίσκεται στο εσωτερικό και στη συνέχεια μια-μια από το εσωτερικό προς το εξωτερικό εκτελούνται οι υπόλοιπες μέχρις ότου φθάσουμε στην πλέον εξωτερική ανακύκλωση.

- Μια εντολή **Do – End Do** γενικής μορφής μπορεί να γραφεί με τη μορφή **Do While** ή **Do (Exit)**. Πράγματι, αν $m_2 \geq m_1$ και $m_3 \geq 0$, θα έχουμε:

Γενική μορφή:

```
Do i = m1, m2, m3
      Εντολές της Fortran
End Do
```

Μορφή Do While:

```
i = m1
Do While (i ≤ m2)
      Εντολές της Fortran
      i = i + m3
End Do
```

Μορφή Do (Exit)

```
i = m1
Do
      Εντολές της Fortran
      i = i + m3
      If (i > m2) Exit
End Do
```

5.3 Παραδείγματα

Παράδειγμα 1: Να γραφεί πρόγραμμα το οποίο θα διαβάζει ένα σύνολο ακεραίων αριθμών και θα υπολογίζει το άθροισμα έως ότου εμφανιστεί ένας αρνητικός αριθμός.

```
      Program Sum_Positive1
C     Με χρήση της εντολής Do – End Do της μορφής Do (Exit)
      Implicit None
      Integer:: x, s = 0

      Do
         Write(*,*) 'Give an Integer'
         Read(*,*) x
         If (x.LT.0) Exit
         s = s + x
      End Do

      Write(*,*) 'Sum=', s

      End


      Program Sum_Positive2
C     Με χρήση της εντολής Do – End Do της μορφής Do While
      Implicit None
      Integer:: x, s = 0

      Write(*,*) 'Give an Integer'
      Read(*,*) x

      Do While (x.GE.0)
         s = s + x
         Write(*,*) 'Give an Integer'
         Read(*,*) x
      End do

      Write(*,*) 'Sum=', s

      End
```

Παράδειγμα 2: Να γραφεί πρόγραμμα το οποίο θα υπολογίζει το μέγιστο κοινό διαιρέτη δύο αριθμών a, b βάσει του αναδρομικού τύπου

$$\text{ΜΚΔ}(a,b)=\begin{cases}\text{ΜΚΔ}(a-b,b), & \alpha\nu\ \ a>b\\ \text{ΜΚΔ}(a,b-a) & \alpha\nu\ \ a<b\\ a & \alpha\nu\ \ a=b\end{cases}.$$

```
Program MKD

Implicit None
Integer a, b

Write(*,*) 'Give two integers'
Read(*,*) a, b

Do While (a.NE.b)
   If (a.GT.b) Then
      a = a – b
   Else
      b = b – a
   End if
End Do

Write(*,*) 'MKD=', a

End
```

Παράδειγμα 3: Να γραφεί πρόγραμμα που θα υπολογίζει το ελάχιστο κοινό πολλαπλάσιο δύο αριθμών a, b βάσει του εξής αλγορίθμου: Τοποθετούμε στη θέση του a το μεγαλύτερο από τους 2 αριθμούς και στη θέση του b το μικρότερο. Αν ο a είναι πολλαπλάσιο του b, τότε ο a είναι το ΕΚΠ, διαφορετικά διπλασιάζουμε, τριπλασιάζουμε κ.τ.λ. τον a, έως ότου το πολλαπλάσιο του a να είναι και πολλαπλάσιο του b. Το πρώτο πολλαπλάσιο του a που είναι και πολλαπλάσιο του b, είναι το ΕΚΠ που ψάχνουμε.

```
Program EKP

Implicit None
Integer a, b, temp, i

Write(*,*) 'Give two integers'
Read(*,*) a, b

If (a.LT.b) Then
   temp = a
   a = b
   b = temp
End if

i = 1

Do While (MOD(i*a,b).NE.0)
   i = i + 1
End Do

Write(*,*) 'EKP=',i*a

End
```

Παράδειγμα 4: Ο υπολογισμός της τετραγωνικής ρίζας του πραγματικού αριθμού a γίνεται από τον αναδρομικό τύπο:

$$x_n = \frac{1}{2}\left(x_{n-1} + \frac{a}{x_{n-1}} \right).$$

Να γραφεί πρόγραμμα που θα υπολογίζει την τετραγωνική ρίζα ενός πραγματικού αριθμού a με ακρίβεια $\varepsilon = 10^{-6}$, αν πάρουμε ως αρχική τιμή της προσέγγισης το $x_0 = \frac{a}{3}$.

```
Program Tetragoniki_Riza

Implicit None
Real e
Parameter (e = 0.000001)
Real a, xold, xnew

Write(*,*) 'Give a real'
Read(*,*) a

xold = a/3
xnew = (1./2)*(xold+a/xold)

Do While (ABS(xnew-xold).GE.e)
   xold = xnew
   xnew = (1./2)*(xold+a/xold)
End Do

Write(*,*) 'Tetragoniki Riza =', xnew

End
```

Παράδειγμα 5: Να γραφεί πρόγραμμα που θα υπολογίζει το άθροισμα:

$$S = 1 + 2 + \ldots + 100.$$

```
Program Athroisma

Implicit None
Integer:: s = 0, i

Do i = 1, 100
   s = s + i
End Do

Write(*,*) 'S=', s

End
```

Παράδειγμα 6: Να γραφεί πρόγραμμα που θα υπολογίζει το N! (N! = 1*2*…*N)

```
Program Paragontiko

Implicit None
Real*8:: p = 1
Integer i, n

Write(*,*)'Give the n'
Read(*,*) n

If (n.LT.0) Then
   Write(*,*)'Error, n must be positive'
Else if (n.EQ.0) Then
   Write(*,*) '0! = 1'
Else
   Do i = 1, n
      p = p * i
   End Do
   Write(*,*) n,'!=', p
End if

End
```

Παρατήρηση: Κανονικά η μεταβλητή p στην οποία θα αποθηκεύονται τα μερικά γινόμενα θα έπρεπε να είναι ακέραια αφού έχουμε γινόμενα ακεραίων αριθμών. Όμως, όπως είδαμε (§3.2), ακόμα και **Integer*4** να δηλώσουμε την μεταβλητή p, το εύρος τιμών της μεταβλητής θα κυμαίνεται από -2^{31} έως $2^{31} - 1$. Επειδή, όσο μεγαλώνει το n τόσο πιο πολύ μεγαλώνει η τιμή της μεταβλητής p, δηλώνουμε τη μεταβλητή p ως **Real*8** για να έχει μεγαλύτερο εύρος τιμών.

Παράδειγμα 7: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο το πλήθος N > 2 αριθμών και στη συνέχεια αφού διαβασθούν οι N αριθμοί να υπολογισθούν και να εμφανιστούν τα εξής: α) ο μεγαλύτερος αριθμός, β) ο μικρότερος αριθμός και γ) ο μέσος όρος τους.

```
      Program Max_Min_Average

      Implicit None
      Real x, s, max, min
      Integer i, n

C     Έλεγχος ότι η τιμή του N θα είναι μεγαλύτερη από 2
      Do
         Write(*,*) 'Give the N'
         Read(*,*) n
         If (n.GT.2) Exit
      End Do

      Write(*,*) 'Give a number'
      Read(*,*) x

      s = x
      max = x
      min = x
      Do i = 2, n
         Write(*,*) 'Give a number'
         Read(*,*) x
         If (max.LT.x) max = x
         If (min.GT.x) min = x
         s = s + x
      End Do

      Write(*,*) 'MAX=', max
      Write(*,*) 'MIN=',min
      Write(*,*) 'AVERAGE=', s/n

      End
```

Παράδειγμα 8: Να γραφεί πρόγραμμα που θα εμφανίζει τη γνωστή προπαίδεια.

```
      Program Propaideia

      Implicit None
      Integer i, j

      Do i = 1, 10
         Do j = 1, 10
            Write(*,*) i, '  *', j, '  =', i*j
         End Do
      End Do

      End
```

Παράδειγμα 9: Να γραφεί πρόγραμμα που θα βρίσκει και θα εμφανίζει τους ακέραιους αριθμούς a, b, c (Πυθαγόρειοι αριθμοί) που ανήκουν στο διάστημα [1, 10] και ικανοποιούν την ιδιότητα:

$$a^2 + b^2 = c^2.$$

```
Program Pythagorioi_Arithmoi1

Implicit None
Integer a, b, c

Do a = 1, 10
   Do b = 1, 10
      Do c = 1, 10
         If ((a**2+b**2).EQ.(c**2)) Then
            Write(*,*) a, b, c
         End if
      End Do
   End Do
End Do

End
```

Παρατήρηση: Παρατηρούμε ότι στο αποτέλεσμα του προηγούμενου προγράμματος εμφανίζονται ίδιες τριάδες αριθμών με διαφορετική βέβαια σειρά. Μπορούμε να τροποποιήσουμε το παραπάνω πρόγραμμα (δες το παρακάτω πρόγραμμα) έτσι ώστε να μην έχουμε επανάληψη τριάδων.

```
Program Pythagorioi_Arithmoi2

Implicit None
Integer a, b, c

Do a = 1, 10
   Do b = a, 10
      Do c = b, 10
         If ((a**2+b**2).EQ.(c**2)) Then
            Write(*,*) a, b, c
         End if
      End Do
   End Do
End Do

End
```

Κεφάλαιο 6ο: Πίνακες

Στο κεφάλαιο αυτό θα ασχοληθούμε με μια από πιο ενδιαφέρουσες δομές δεδομένων, τους πίνακες. Οι πίνακες είναι σύνθετες δομές δεδομένων που μας δίνουν τη δυνατότητα να αποθηκεύσουμε μεγάλη ομάδα πληροφοριών στη μνήμη του Η/Υ με ένα ξεχωριστό όνομα και ένα δείκτη που θα προσδιορίζει τη θέση του στοιχείου μέσα στην ομάδα. Η σημασία τους έγκειται στο ότι μας δίνουν τη δυνατότητα να αναλύσουμε τα δεδομένα εύκολα μέσω ανακυκλώσεων. Επειδή τα δεδομένα αυτά αποθηκεύονται σε ξεχωριστές θέσεις της μνήμης του Η/Υ, μπορούμε να έχουμε πρόσβαση σε αυτά χωρίς να χρειαστεί να τα ξαναδιαβάσουμε.

6.1 Ορισμός

Πίνακας είναι μια σύνθετη μεταβλητή που καταλαμβάνει παραπάνω από μία θέση στην μνήμη του Η/Υ, έχει ένα συγκεκριμένο όνομα και δέχεται ένα συγκεκριμένο τύπο δεδομένων.

Τα ονόματα των πινάκων ακολουθούν τους ίδιους κανόνες με αυτά των απλών μεταβλητών. Οι πίνακες, όπως οι απλές μεταβλητές, χωρίζονται σε αριθμητικούς και αλφαριθμητικούς, ανάλογα με το τύπο δεδομένων που δέχονται και ο διαχωρισμός τους γίνεται με το ίδιο τρόπο, δηλαδή με τις δηλωτικές εντολές **Real**, **Integer** κ.τ.λ. Αν δεν οριστεί ο τύπος, τότε αν το όνομα αρχίζει με ένα από τα 6 γράμματα I, J, K, L, M, N τα στοιχεία του πίνακα είναι ακέραιοι, διαφορετικά θα είναι πραγματικοί.

Τη θέση των στοιχείων μέσα σε ένα πίνακα την προσδιορίζουμε με τη βοήθεια των **δεικτών**. Έτσι, αν έχω ένα αριθμητικό πίνακα με όνομα Α που καταλαμβάνει 5 συνεχόμενες θέσεις στη μνήμη του Η/Υ, θα αναφερόμαστε στο πρώτο στοιχείο του πίνακα με το όνομα Α(1), στο δεύτερο με το όνομα Α(2) κ.τ.λ. Ο δείκτης ενός πίνακα μπορεί να είναι μια σταθερά ή μεταβλητή ή παράσταση που παίρνει ακέραιες τιμές και φυσικά δεν θα πρέπει να ξεπερνά τα όρια της διάστασης του πίνακα. Στους δυσδιάστατους πίνακες, ο πρώτος δείκτης αναφέρεται σε μια γραμμή του πίνακα, ενώ ο δεύτερος σε μια στήλη του πίνακα.

6.2 Δήλωση πινάκων

Το πλήθος των δεικτών και οι μεγαλύτερες τιμές που μπορούν να πάρουν ορίζεται με μία δηλωτική εντολή που λέγεται **Dimension**.

Η εντολή **Dimension** ανήκει στις εντολές προδιαγραφών που δεν εκτελούνται και δηλώνει - ορίζει το μέγεθος ενός ή περισσοτέρων πινάκων. Δηλαδή, μ' αυτή την εντολή ζητάμε να δεσμευτεί στη μνήμη μια ζώνη από συνεχείς θέσεις μνήμης οι οποίες να έχουν όλες το ίδιο όνομα. Η εντολή **Dimension** πρέπει να βρίσκεται πριν από την πρώτη φορά που θα συναντήσουμε τη μεταβλητή αυτή.

Έτσι, από συνήθεια και ομοιομορφία, συγκεντρώνουμε όλες τις εντολές **Dimension** στην αρχή του προγράμματος.

Ο σκοπός αυτής της εντολής είναι να προειδοποιήσει το μεταγλωττιστή της γλώσσας **Fortran** για να δεσμεύσει τις απαραίτητες θέσεις γι' αυτή τη μεταβλητή στη μνήμη πριν από την χρησιμοποίησή της.
Μέχρι και τη **Fortran 77**, η γενική μορφή της εντολής DIMENSION είναι:

Dimension "όνομα πίνακα" (X : Y)

Το X εκφράζει την κατώτερη τιμή που μπορεί να πάρει ο δείκτης και όταν λείπει εννοείται η μονάδα.

Το Y εκφράζει την ανώτερη τιμή και πρέπει να υπάρχει υποχρεωτικά.

Ο τύπος των πινάκων ακολουθεί τους ίδιους κανόνες με μία απλή μεταβλητή.

Μπορούμε να δηλώσουμε ταυτόχρονα και περισσότερα ονόματα πινάκων χωρισμένα με υποδιαστολή.

Π.χ. μπορούμε να γράψουμε :

Dimension A(15), B(10, 10), C(-5 : 5), K(3, 2, 2)

Η τιμή που είναι μέσα σε παρένθεση δηλώνει τον αριθμό των διαδοχικών θέσεων μνήμης που θα δεσμευτούν. Όταν υπάρχουν περισσότερες τιμές, που μπορεί να φθάσουν μέχρι και τις 7, αυτό σημαίνει ο πίνακας είναι περισσοτέρων διαστάσεων. Π.χ.

Για τον πίνακα A(15) κρατάμε 15 διαδοχικές θέσεις μνήμης.

Για τον πίνακα B(10,10) κρατάμε 10*10 = 100 διαδοχικές θέσεις μνήμης.

Για τον πίνακα C(-5:5) κρατάμε συνολικά 11 θέσεις μνήμης.

Για τον πίνακα K(3, 2, 2) κρατάμε συνολικά 3*2*2 = 12 θέσεις μνήμης.

Με τη **Fortran 90/95** η γενική μορφή της εντολής **Dimension** γίνεται :

Τύπος, **Dimension** (X : Y) **::** "όνομα πίνακα"

Ο **Τύπος**, είναι ένας από τους γνωστούς και αποδεκτούς τύπους των μεταβλητών.

Μπορούμε να δηλώσουμε ταυτόχρονα και περισσότερα ονόματα πινάκων χωρισμένα με υποδιαστολή.

Π.χ. μπορούμε να γράψουμε :

Real, **Dimension** (15) **::** A, B, C(-5 : 5), K(3, 2, 2)

Εδώ, όλοι οι πίνακες θα δέχονται μόνο πραγματικές τιμές, αλλά επιπλέον, οι μεταβλητές C και K δεν θα ακολουθούν τη γενική δήλωση. Δηλαδή, οι πίνακες A και B θα είναι μονοδιάστατοι και 15 θέσεων ο κάθε ένας, ενώ ο πίνακας C θα είναι μονοδιάστατος 11 θέσεων και ο πίνακας K τρισδιάστατος. Επομένως, έχουμε την ευχέρεια να δηλώνουμε τους πίνακες που θα χρειαστούμε στο πρόγραμμα με μεγάλη ευελιξία.

Αν γράψουμε :

Integer, **Dimension** (10, 10) **::** A, B, C

τότε, όλοι οι πίνακες θα είναι δυσδιάστατοι (10, 10) και θα δέχονται μόνο ακέραιες τιμές.

Εναλλακτικά, μπορούμε να δηλώσουμε τους πίνακες χωρίς την χρήση της εντολής **Dimension**, μέσω των, γνωστών από τις μεταβλητές, δηλωτικών εντολών.

Π.χ. μπορούμε να γράψουμε τις ακόλουθες εντολές:

Real X(5, 5), Y(5, 5)

Integer Tab(100), Pin(10, 20)

αντί για τις εντολές (Fortran 77):

Dimension X(5, 5), Y(5, 5)

Integer Tab, Pin

Dimension Tab(100), Pin(10, 20)

ή τις εντολές (Fortran 90/95):

Real, **Dimension** (5, 5) **::** X, Y

Integer, **Dimension** (100) **::** Tab, Pin(10, 20)

6.3 Αποθήκευση πινάκων

Για έναν πίνακα μιας διάστασης είναι απλό. Το στοιχείο π.χ. A(3), βρίσκεται στην τρίτη κατά σειρά θέση μνήμης.

Για ένα πίνακα δύο διαστάσεων π.χ. B(3 , 2), οι θέσεις στη μνήμη βρίσκονται με την ακόλουθη διάταξη: B(1,1) B(2,1), B(3,1) και στη συνέχεια B(1,2), B(2,2), B(3,2), δηλαδή οι πίνακες δύο διαστάσεων κρατούν τόσες θέσεις όσα τα στοιχεία τους, που είναι διατεταγμένα μέσα στη μνήμη, πρώτα όλα τα στοιχεία της πρώτης στήλης, κατόπιν όλα τα στοιχεία της δεύτερης στήλης και ούτω καθ' εξής μέχρι την τελευταία στήλη. Δηλαδή η Fortran καθορίζει ότι η διάταξη με την οποία θα τοποθετηθούν τα στοιχεία ενός πίνακα στην μνήμη του H/Y, είναι κατά στήλες.

6.4 Εισαγωγή τιμών σε πίνακα

Όταν ορίζουμε τον πίνακα με έναν από τους τρόπους της παραγράφου 6.2:

- Αν ο πίνακας είναι αριθμητικός, έχουμε αυτόματο μηδενισμό όλων των στοιχείων του.
- Αν ο πίνακας είναι αλφαριθμητικός, όλα του τα στοιχεία έχουν μηδενικό μήκος (κενές συμβολοσειρές).

Στη συνέχεια θα δούμε πως μπορούμε να δώσουμε τιμές σε έναν πίνακα, διαχωρίζοντας του πίνακες σε μονοδιάστατους και δυσδιάστατους.

6.4.1 Εισαγωγή τιμών σε μονοδιάστατο πίνακα

Μπορούμε να δώσουμε τιμές σ' ένα μονοδιάστατο πίνακα με τους ακόλουθους τρεις τρόπους.

Α' τρόπος. Με Εντολές Εισόδου

Αν θέλουμε να δώσουμε δυναμικά, κατά τη διάρκεια της εκτέλεσης του προγράμματος, τιμές οποίες θα αποθηκευτούν σε πίνακες μπορούμε να χρησιμοποιήσουμε τις εντολές εισόδου της **Fortran**. Συγκεκριμένα μπορούμε να χρησιμοποιήσουμε την εντολή **Do – End Do**:

```
Integer A
Dimension A(5)
Write(*, *) "Πληκτρολογείστε πέντε τιμές"
Do i =1, 5
   Read(*, *) A(i)
End Do
```

Να έχουμε απευθείας ανάγνωση του πίνακα Α

```
Integer A
Dimension A(5)
Write(*, *) "Πληκτρολογείστε πέντε τιμές"
Read(*, *) A
```

Οι πέντε τιμές που θα πρέπει να πληκτρολογηθούν, όταν εμφανιστεί το μήνυμα στην οθόνη, θα τοποθετηθούν στις πέντε (5) θέσεις του πίνακα Α.

Ή να χρησιμοποιήσουμε την υπονοούμενη (implied) εντολή **Do**:

```
Integer A
Dimension A(5)
Write(*, *) "Πληκτρολογείστε πέντε τιμές"
Read(*, *) (A(i), i = 1, 5)
```

Η γενική μορφή της εντολής για το υπονοούμενο **Do** είναι:

("μεταβλητή" , "μεταβλητή ελέγχου" = "κατώτερο όριο", "ανώτερο όριο", "βήμα")

Π.χ. **Read**(*, *) (A(i), i = 1, 10, 2)

Έχουμε δηλαδή, τις ίδιες έννοιες μ' εκείνες που είδαμε για την εντολή **Do**.

Β' τρόπος. Με εντολές ανάθεσης

Αν έχουμε τον πίνακα Α(5) και θέλουμε να δώσουμε τις τιμές 10, 20, 30, 40, 50 μπορούμε να γράψουμε (**Fortran 77**):

```
Integer A
Dimension A(5)
A(1) = 10
A(2) = 20
A(3) = 30
A(4) = 40
```

```
A(5) = 50
```

ή να γράψουμε (**Fortran 90/95**):

```
Integer, Dimension (5) :: A
A(1) = 10
A(2) = 20
A(3) = 30
A(4) = 40
A(5) = 50
```

Μπορούμε να βελτιώσουμε αυτόν τον κώδικα με την χρήση της εντολής **Do**.

Π.χ.

```
Integer A
Dimension A(5)
Do i = 1, 5
   A(i) = i*10
End Do
```

ή

```
Integer, Dimension (5) :: A
Do i = 1, 5
   A(i) = i*10
End Do
```

Μπορούμε να δώσουμε και σε μεμονωμένες θέσεις ενός πίνακα τιμές, δηλαδή μπορούμε να γράψουμε :

```
A(3) = 3055
```

Γ' τρόπος. Με την εντολή Data

Όταν πρέπει να δώσουμε αρχικές τιμές σε ένα πρόγραμμα οι οποίες δεν θα αλλάζουν σε κάθε νέα εκτέλεση του προγράμματος ή κατά τη διάρκεια της εκτέλεσης, θεωρείται άστοχο να τις πληκτρολογούμε σε κάθε νέα επανεκκίνηση του προγράμματος. Οι λόγοι της αποφυγής αυτής της μορφής απόδοσης τιμών είναι κυρίως δύο.

Ο πρώτος λόγος είναι ότι προκαλείται καθυστέρηση στην έκδοση των αποτελεσμάτων, από τη χρονοβόρο διαδικασία της πληκτρολόγησης και ο δεύτερος, ίσως και ο πιο σημαντικός λόγος, είναι ότι μια λανθασμένη πληκτρολόγηση μπορεί να προκαλέσει λανθασμένα αποτελέσματα.

Συνιστάται λοιπόν, η χρησιμοποίηση της εντολής **Data**, η οποία επιτρέπει τη μαζική απόδοση τιμών σε απλές μεταβλητές και σε μεταβλητές με δείκτες (πίνακες).

Η γενική μορφή της εντολής **Data**, είναι :

Data "μεταβλητές" / "σταθερές τιμές" /.

Μπορούμε, στη θέση των "μεταβλητών" να τοποθετήσουμε μια σειρά από απλές μεταβλητές ή/και μεταβλητές με δείκτες (πίνακες) του προγράμματος.

Στη θέση των "σταθερών τιμών" γράφουμε τις αντίστοιχες τιμές των "μεταβλητών".

Π.χ. η εκτέλεση των εντολών:

```
Real X(4)
Data X / 1.2, 15.5, 7.0, 14.1 /
```

θα προκαλέσει αντικατάσταση των τεσσάρων αριθμητικών τιμών στις τέσσερις θέσεις του πίνακα Χ.

Η συνέχεια των εντολών:

```
Integer X(4), Z
Data X, Z / 1, 2, 3, 4, 5 /
```

θα προκαλέσει αντικατάσταση των 4 πρώτων τιμών (1, 2, 3, 4) στις 4 θέσεις του πίνακα Χ ως εξής: Χ(1)=1, Χ(2)=2, Χ(3)=3 και Χ(4)=4, και την τιμή 5 στην απλή μεταβλητή Z.

Η εντολή **Data** μπορεί να βρίσκεται σε οποιοδήποτε σημείο του κώδικα αλλά, τοποθετείται, τις περισσότερες φορές, στην αρχή του προγράμματος και χρησιμεύει κυρίως για την απόδοση αρχικών ή σταθερών τιμών στις μεταβλητές.

6.4.2 Εισαγωγή τιμών σε δυσδιάστατο πίνακα

Μπορούμε να δώσουμε τιμές σ' ένα δυσδιάστατο πίνακα με τους ακόλουθους τρεις τρόπους.

Α' τρόπος. Με Εντολές Εισόδου

Αν θέλουμε να δώσουμε δυναμικά, κατά τη διάρκεια της εκτέλεσης του προγράμματος, τιμές οποίες θα αποθηκευτούν σε πίνακες μπορούμε να χρησιμοποιήσουμε τις εντολές εισόδου της **Fortran**. Συγκεκριμένα μπορούμε να χρησιμοποιήσουμε την εντολή **Do – End Do**:

Διάβασμα τιμών ενός δυσδιάστατου πίνακα κατά γραμμές:

```
Integer A
Dimension A(3, 2)
Do i =1, 3
   Do j =1, 2
      Read(*, *) A(i, j)
   End Do
End Do
```

Διάβασμα τιμών ενός δυσδιάστατου πίνακα κατά στήλες:

```
Integer A
```

```
Dimension A(3, 2)
Do j =1, 2
   Do i =1, 3
      Read(*, *) A(i, j)
   End Do
End Do
```

Να έχουμε απευθείας ανάγνωση του πίνακα Α

```
Integer A
Dimension A(3, 2)
Read(*, *) A
```

Οι έξι πρώτες τιμές που θα πληκτρολογηθούν, θα τοποθετηθούν στις 6 θέσεις του πίνακα Α.

Προσοχή: Επειδή η **Fortran** αποθηκεύει τις τιμές ενός δυσδιάστατου πίνακα κατά στήλες θα πρέπει να δίνουμε τις τιμές του πίνακα Α κατά στήλες.

Ή να χρησιμοποιήσουμε την υπονοούμενη (implied) εντολή **Do**. Όπως μπορούμε να χρησιμοποιήσουμε πολλές ανακυκλώσεις τη μία μέσα στην άλλη, σε μια εντολή **Do**, έτσι και σε μια υπονοούμενη **Do** μπορούμε να γράφουμε:

```
Integer A
Dimension A(3, 2)
Read(*, *) ((A(i,j), j = 1, 2), i = 1, 3)
```

αν θέλουμε να διαβάσουμε τις τιμές του πίνακα κατά γραμμές. Από τους δύο δείκτες i και j, πρώτα μεταβάλλεται εκείνος που είναι εσωτερικά δηλαδή ο j και κατόπιν ο i.

Ή να γράφουμε:

```
Integer A
Dimension A(3, 2)
Read(*, *) ((A(i,j), i = 1, 3), j = 1, 2)
```

αν θέλουμε να διαβάσουμε τις τιμές του πίνακα κατά στήλες.

Β' τρόπος. Με εντολές ανάθεσης

Αν έχουμε τον πίνακα Α(3, 2) και θέλουμε να δώσουμε τις τιμές

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}$$

μπορούμε να γράψουμε:

```
Integer A
Dimension A(3, 2)
```

A(1, 1) = 1

A(1, 2) = 2

A(2, 1) = 2

A(2, 2) = 4

A(3, 1) = 3

A(3, 2) = 6

Μπορούμε να βελτιώσουμε αυτόν τον κώδικα με την χρήση της εντολής **Do**.

Π.χ.

```
Integer A
Dimension A(3, 2)
Do i = 1, 3
   Do j = 1, 2
      A(i,j) = i*j
   End Do
End Do
```

Μπορούμε να δώσουμε και σε μεμονωμένες θέσεις ενός πίνακα τιμές, δηλαδή μπορούμε να γράψουμε :

A(3, 2) = 6.

Γ' τρόπος. Με την εντολή Data

Αν έχουμε τον πίνακα A(3, 2) και θέλουμε να δώσουμε τις τιμές

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}$$

μπορούμε να γράψουμε:

Data A / 1, 2, 3, 2, 4, 6 /

Προσοχή: Επειδή η Fortran αποθηκεύει τις τιμές ενός δυσδιάστατου πίνακα κατά στήλες θα πρέπει να δίνουμε τις τιμές του πίνακα Α κατά στήλες.

6.5 Εμφάνιση (εκτύπωση) τιμών ενός πίνακα

Αν και το επόμενο στάδιο μετά την εισαγωγή τιμών σε ένα πίνακα, είναι η επεξεργασία των τιμών του πίνακα και στην συνέχεια η εμφάνιση των τιμών του πίνακα, για λόγους ευκολίας θα αναφερθούμε στην συνέχεια στην εμφάνιση των τιμών ενός πίνακα. Ο κύριος λόγος για την επιλογή αυτή, είναι ότι η εμφάνιση των τιμών ενός πίνακα μοιάζει πάρα πολύ με την εισαγωγή (ανάγνωση) των τιμών του πίνακα. Η κύρια διαφορά έγκειται ότι στην ανάγνωση

χρησιμοποιούμε την εντολή **Read**, ενώ στην εκτύπωση χρησιμοποιούμε την εντολή **Write** ή **Print**.

6.5.1 Εκτύπωση τιμών μονοδιάστατου πίνακα

***Α' τρόπος. Με χρήση της εντολής* Do – End Do**

```
Integer A
Dimension A(5)
Do i =1, 5
   Write*, *) A(i)
End Do
```

***Β' τρόπος. Με χρήση της υπονοούμενης εντολής* Do**

```
Integer A
Dimension A(5)
Write(*, *) (A(i), i = 1, 5)
```

Γ' τρόπος. Με απευθείας εκτύπωση

```
Integer A
Dimension A(5)
Write(*, *) A
```

6.5.2 Εκτύπωση τιμών δυσδιάστατου πίνακα

***Α' τρόπος. Με χρήση της εντολής* Do – End Do**

```
Integer A
Dimension A(3, 2)
Do i =1, 3
   Do j = 1, 2
      Write*, *) A(i, j)
   End Do
End Do
```

Εκτυπώνονται οι τιμές του πίνακα η μία κάτω από την άλλη κατά γραμμές.

```
Integer A
Dimension A(3, 2)
Do j =1, 2
   Do i = 1, 3
```

```
            Write*, *) A(i, j)
        End Do
    End Do
```

Εκτυπώνονται οι τιμές του πίνακα η μία κάτω από την άλλη κατά στήλες.

***Β' τρόπος. Με χρήση της υπονοούμενης εντολής* Do**

```
Integer A
Dimension A(3, 2)
Do i =1, 3
    Write*, *) (A(i, j), j = 1, 2)
End Do
```

Εκτυπώνεται ο πίνακας με τη συνήθη μορφή.

Γ' τρόπος. Με απευθείας εκτύπωση

```
Integer A
Dimension A(3, 2)
Write(*, *) A
```

Εκτυπώνονται οι τιμές του πίνακα η μία κάτω από την άλλη κατά στήλες.

Σημείωση: Από τους παραπάνω τρεις τρόπους εμφάνισης των τιμών ενός πίνακα, ο προτιμότερος είναι ο Β' ο οποίος εκτυπώνει τις τιμές του πίνακα με τη συνήθη μορφή.

6.6 Νέες εντολές της Fortran 90/95

Με τη **Fortran 90/95** έχουν εισαχθεί νέες εντολές και συναρτήσεις για την επεξεργασία των πινάκων οι οποίες εμπλουτίζουν τη γλώσσα και παρέχουν μεγαλύτερη ευελιξία στους προγραμματιστές.

Οι νέες εντολές επιτρέπουν τη δυναμική δέσμευση της μνήμης ενώ οι νέες συναρτήσεις χρησιμοποιούνται στην επεξεργασία των τιμών ενός πίνακα.

6.6.1 Δυναμική δέσμευση μνήμης

Λέμε δυναμική δέσμευση της μνήμης την ενέργεια του προγραμματιστή με την οποία δεσμεύει και χρησιμοποιεί εκείνη την περιοχή της μνήμης που πραγματικά έχει ανάγκη και μάλιστα τη στιγμή που την έχει ανάγκη και όσο χρόνο αυτό απαιτείται από τον αλγόριθμο επίλυσης του προβλήματος. Όταν δεν χρειάζεται πια αυτή η περιοχή της μνήμης μπορεί να αποδεσμευτεί και να χρησιμοποιεί για άλλες πράξεις.

Μέχρι και τη **Fortran 77** δεν υπήρχε πρόβλεψη για δυναμική δέσμευση της μνήμης και οι προγραμματιστές ήταν αναγκασμένοι να καταφεύγουν στη στατική δέσμευση της μνήμης δηλαδή, να δηλώνουν στην αρχή του προγράμματος τη μέγιστη διάσταση των πινάκων και να χρησιμοποιούν πολλές φορές ένα μέρος από τη δεσμευμένη περιοχή της μνήμης. Αυτό είχε

σαν αποτέλεσμα την άσκοπη απασχόληση της μνήμης και την αδυναμία εκτέλεσης προγραμμάτων με μεγάλες απαιτήσεις σε διαθέσιμο χώρο μνήμης.

Ειδικά, μέχρι της αρχές της δεκαετία του '90 που οι τιμές αγοράς της μνήμης ήταν πολύ υψηλές και οι δυνατότητες των υπολογιστών περιορισμένες, η λύση της στατικής δέσμευσης της μνήμης δεν επέτρεπε την ορθολογική χρήση των υπολογιστών και τη βέλτιστη απόδοση των αλγορίθμων.

Η γενική μορφή δήλωσης δυναμικής δέσμευσης της μνήμης είναι:

Τύπος, Dimension (:), Allocatable :: "όνομα μεταβλητής με δείκτες"

Ο **Τύπος,** είναι ένας από τους γνωστούς και αποδεκτούς τύπους των μεταβλητών. Ο τύπος των μεταβλητών με δείκτες ακολουθεί τους ίδιους κανόνες με μία απλή μεταβλητή.

Η λέξη **Allocatable** υποδεικνύει στο μεταγλωττιστή ότι οι μεταβλητές με δείκτες που ακολουθούν δεν θα έχουν από την αρχή του προγράμματος καθορισμένο μέγεθος αλλά η διάστασή τους θα οριστεί αργότερα κατά τη διάρκεια του προγράμματος.

Αν γράψουμε :

Real, **Dimension** (:), **Allocatable** **::** TAB

και στη συνέχεια, τη στιγμή που θα πρέπει να χρησιμοποιήσουμε τον πίνακα TAB, π.χ. για την επίλυση ενός συστήματος εκατό εξισώσεων με εκατό αγνώστους, μπορούμε να γράψουμε:

Allocate (TAB(100, 100))

Όταν ολοκληρωθεί ο αλγόριθμος της επίλυσης του συστήματος και δεν έχουμε πλέον ανάγκη τις 10.000 θέσεις μνήμης, μπορούμε να απελευθερώσουμε τη δεσμευμένη μνήμη με τη βοήθεια της εντολή **Deallocate**.

Αρκεί λοιπόν, να γράψουμε:

Deallocate (TAB)

Αν χρειαστεί, μπορούμε να ξαναχρησιμοποιήσουμε τον πίνακα TAB με μικρότερη ή μεγαλύτερη διάσταση. Π.χ. αν χρειαζόμαστε ένα νέο πίνακα μονοδιάστατο, 7500 θέσεων μνήμης, μπορούμε να χρησιμοποιήσουμε την ίδια περιοχή μνήμης και να γράψουμε:

Allocate (TAB(7500))

και όταν δεν χρειαζόμαστε πλέον τον πίνακα TAB, μπορούμε να ξαναγράψουμε:

Deallocate (TAB)

Μπορούμε επίσης, να δηλώσουμε τη διάσταση του πίνακα όχι με μια σταθερή τιμή αλλά με την τιμή μιας μεταβλητής η οποία θα παίρνει τιμή π.χ. από μια εντολή εισόδου.

Μπορούμε λοιπόν, να έχουμε τις ακόλουθες εντολές:

Real, **Dimension** (:), **Allocatable** **::** XX
……..
……..
Write (*, *) "Δώστε τη διάσταση του πίνακα εργασίας"

```
Read (*, *), K
Allocate ( XX( K, K) )
……..
……..
Deallocate ( XX )
```

Εδώ, κατά τη διάρκεια εκτέλεσης του προγράμματος και ανάλογα με τις απαιτήσεις του προβλήματος, ο χρήστης μπορεί να πληκτρολογήσει την ακριβή διάσταση του πίνακα XX τη στιγμή που την έχει ανάγκη και επομένως, δεσμεύει τόση περιοχή μνήμης όση έχει πραγματικά ανάγκη με την εντολή:

Allocate (XX(K, K))

Όταν ολοκληρωθεί η επεξεργασία και δεν πρόκειται να ξαναχρησιμοποιεί ο πίνακας XX τότε, με την εντολή:

Deallocate (XX)

αποδεσμεύεται η δεσμευμένη περιοχή της μνήμης για τον πίνακα εργασίας XX με αποτέλεσμα την ορθολογιστική διαχείριση και η εκμετάλλευση της μνήμης.

6.6.2 Συναρτήσεις για την επεξεργασία των τιμών ενός πίνακα

Όνομα	Περιγραφή
MAXLOC(X)	Υπολογίζει τη θέση του μέγιστου στοιχείου του πίνακα X
MINLOCX)	Υπολογίζει τη θέση του ελάχιστου στοιχείου του πίνακα X
MAXVAL(X)	Υπολογίζει τη μέγιστη τιμή του πίνακα X
MINVAL(X)	Υπολογίζει την ελάχιστη τιμή του πίνακα X

6.7 Παραδείγματα

Παράδειγμα 1: Να γραφεί πρόγραμμα που θα διαβάζει 30 ακέραιους αριθμούς από το πληκτρολόγιο, θα τους τοποθετεί σε ένα πίνακα A και στη συνέχεια θα βρίσκει και θα εμφανίζει:

α) το πλήθος των αριθμών που βρίσκονται έξω από το διάστημα [-10, 10],

β) το γινόμενο των αριθμών που είναι διάφοροι του μηδενός και βρίσκονται στο διάστημα [-5, 5],

γ) το άθροισμα των αριθμών που είναι πολλαπλάσια του 5 και

δ) ο μέσος όρος των άρτιων αριθμών.

```
      Program Example1

      Implicit None
      Integer n
      Parameter (n=30)
      Integer:: A(n), i, counter = 0, p = 1, s = 0, counter_even = 0
      Real:: MO, s_evev = 0.0

      Do i = 1, n
         Write(*,*) 'Give an integer'
         Read(*,*) A(i)
      End Do

      Do i = 1, n
C        Ερώτημα (α)
         If ((A(i).LT.-10).OR.(A(i).GT.10)) Then
            counter = counter + 1
         End if
C        Ερώτημα (β)
         If (((A(i).GE.-5).AND.(A(i).LE.5)).AND.(A(i).NE.0)) Then
            p = p*A(i)
         End if
C        Ερώτημα (γ)
         If (MOD(A(i),5).EQ.0) Then
            s = s + A(i)
         End if
C        Ερώτημα (δ)
         If (MOD(A(i),2).EQ.0) Then
            s_even = s_even + a(i)
            counter_even = counter_even + 1
         End if
      End Do

      Write(*,*) 'Plithos aritmwn exw apo to diastima [-10, 10] =', counter
      Write(*,*) 'Ginomeno mi midenikwn arithmwn sto diastima [-5, 5]=', p
      Write(*,*) 'Athroisma pollaplasiwn tou 5 =', s
      If (counter_even.EQ.0) Then
         Write(*,*) 'Den yparxoun artioi arithmoi'
      Else
         MO = s_even/counter_even
         Write(*,*) 'Mesos oros artiwn =', MO
      End if

      End
```

Παράδειγμα 2: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία ενός διανύσματος Α ακεραίων αριθμών διάστασης Ν και στη συνέχεια θα βρίσκει και θα εμφανίζει το μέγιστο στοιχείο και τη θέση του (την τελευταία εμφάνιση του μέγιστου) καθώς και το ελάχιστο στοιχείο και τη θέση του (την τελευταία εμφάνιση του ελαχίστου).

```
Program Max_Min_Vector

Implicit None
Integer n
Parameter (n = 8)
Integer A(n), i, max, pos_max, min, pos_min

Do i = 1, n
   Write(*,*) 'Give the', i, 'element'
   Read(*,*) A(i)
End Do

max = A(1)
pos_max = 1
min = A(1)
pos_min = 1

Do i = 1, n
   If (max.LE.A(i)) Then
      max = A(i)
      pos_max = i
   End if
   If (min.GE.A(i)) Then
      min = A(i)
      pos_min = i
   End if
End Do

Write(*,*) 'MAX =', max, '   Position =', pos_max
Write(*,*) 'MIN =', min, '   Position =', pos_min

End
```

Παράδειγμα 3: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο διανυσμάτων Α και Β ακεραίων αριθμών διάστασης Ν και θα υπολογίζει και θα εμφανίζει το άθροισμά τους.

```
Program Sum_Vector

Implicit None
Integer n
Parameter (n = 8)
Integer A(n), B(n), C(n), i

Write(*,*) 'Give the elements of vector A'
Do i = 1, n
   Write(*,*) 'Give the', i, ' element'
   Read(*,*) A(i)
End Do

Write(*,*) 'Give the elements of vector B'
Do i = 1, n
   Write(*,*) 'Give the', i, ' element'
   Read(*,*) B(i)
End Do

Do i = 1, n
   C(i)=A(i)+B(i)
End Do

Write(*,*) 'The elements of vector C'
Do i = 1, n
   Write(*,*) C(i)
End Do

End
```

Παράδειγμα 4: Να γραφεί πρόγραμμα που θα δημιουργεί τους παρακάτω πίνακες:

$$A = I_4 \quad \text{και} \quad B = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

```
Program Create_Arrays

Implicit None
Integer n
Parameter (n = 4)
Integer A(n, n), B(n, n), i, j

Do i = 1, n
   Do j = 1, n
      If (i.EQ.j) Then
         A(i,j) = 1
      Else
         A(i,j) = 0
      End if
   End Do
End Do

Do i = 1, n
   Do j = 1, n
      B(i,j) = 0
   End Do
End Do

Do i = 1, n-1
   B(i,i+1) = 1
End Do

Write(*,*) 'Array A'
Do i = 1, n
   Write(*,*) (A(i,j), j = 1, n)
End Do

Write(*,*) 'Array B'
Do i = 1, n
   Write(*,*) (B(i,j), j = 1, n)
End Do

End
```

Παράδειγμα 5: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία ενός πίνακα Α ακεραίων αριθμών διάστασης NxM και στη συνέχεια θα βρίσκει και θα εμφανίζει το μέγιστο στοιχείο και τη θέση του (την τελευταία εμφάνιση του μέγιστου) καθώς και το ελάχιστο στοιχείο και τη θέση του (την τελευταία εμφάνιση του ελαχίστου).

```
Program Max_Min_Array

Implicit None
Integer n, m
Parameter (n = 4, m =3)
Integer A(n,m), i, j, max, posi_max, posj_max, min, posi_min, posj_min

Do i = 1, n
   Do j = 1, m
      Write(*,*) 'Give the',i,',',j, 'element'
      Read(*,*) A(i,j)
   End Do
End Do

max = A(1,1)
posi_max = 1
posj_max = 1
min = A(1,1)
posi_min = 1
posj_min = 1

Do i = 1, n
   Do j = 1, m
      If (max.LE.A(i,j)) Then
         max = A(i,j)
         posi_max = i
         posj_max = j
      End if
      If (min.GE.A(i,j)) Then
      min = A(i,j)
      posi_min = i
      posj_min = j
      End if
   End Do
End Do

Write(*,*) 'MAX =', max, '   Position = (', posi_max,',', posj_max,')'
Write(*,*) 'MIN =', min, '   Position = (', posi_min,',', posj_min,')'

End
```

Παράδειγμα 6: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο πινάκων Α και Β ακεραίων αριθμών διάστασης NxM και θα υπολογίζει και θα εμφανίζει το άθροισμά τους.

```
Program Sum_Array

Implicit None
Integer n, m
Parameter (n = 4, m = 3)
Integer A(n,m), B(n,m), C(n,m), i, j

Write(*,*) 'Give the elements of array A'
Do i = 1, n
   Do j = 1, m
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) A(i,j)
   End Do
End Do

Write(*,*) 'Give the elements of array B'
Do i = 1, n
   Do j = 1, m
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) B(i,j)
   End Do
End Do

Do i = 1, n
   Do j = 1, m
      C(i,j)=A(i,j)+B(i,j)
   End Do
End Do

Write(*,*) 'The elements of array C'
Do i = 1, n
   Write(*,*) (C(i,j), j = 1, m)
End Do

End
```

Παράδειγμα 7: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο πινάκων Α και Β ακεραίων αριθμών διάστασης NxM και θα υπολογίζει και θα εμφανίζει το γινόμενό τους.

```
Program Multiplication_Array

Implicit None
Integer n, m, l
Parameter (n = 4, m = 3, k=2)
Integer A(n,k), B(k,m), C(n,m), i, j, z

Write(*,*) 'Give the elements of array A'
Do i = 1, n
   Do j = 1, k
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) A(i,j)
   End Do
End Do

Write(*,*) 'Give the elements of array B'
Do i = 1, k
   Do j = 1, m
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) B(i,j)
   End Do
End Do

Do i = 1, n
   Do j = 1, m
      C(i,j) = 0
      Do z = 1, k
         C(i,j) = C(i,j) + A(i,z)*B(z,j)
      End Do
   End Do
End Do

Write(*,*) 'The elements of array C'
Do i = 1, n
   Write(*,*) (C(i,j), j = 1, m)
End Do

End
```

Παράδειγμα 8: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία ενός πινάκα Α ακεραίων αριθμών διάστασης NxM και θα υπολογίζει και θα εμφανίζει το άθροισμα της κάθε γραμμής του και το άθροισμα της κάθε στήλης του.

```
Program Sum_Row_Sum_Column

Implicit None
Integer n, m
Parameter (n = 4, m = 3)
Integer A(n,m), i, j, s

Write(*,*) 'Give the elements of array A'
Do i = 1, n
   Do j = 1, m
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) A(i,j)
   End Do
End Do

Do i = 1, n
   s = 0
   Do j = 1, m
      s = s + A(i,j)
   End Do
   Write(*,*) 'To athrisma tis ',i,' grammis einai:', s
End Do

Do j = 1, m
   s = 0
   Do i = 1, n
      s = s + A(i,j)
   End Do
   Write(*,*) 'To athrisma tis ',j,' stilis einai:', s
End Do

End
```

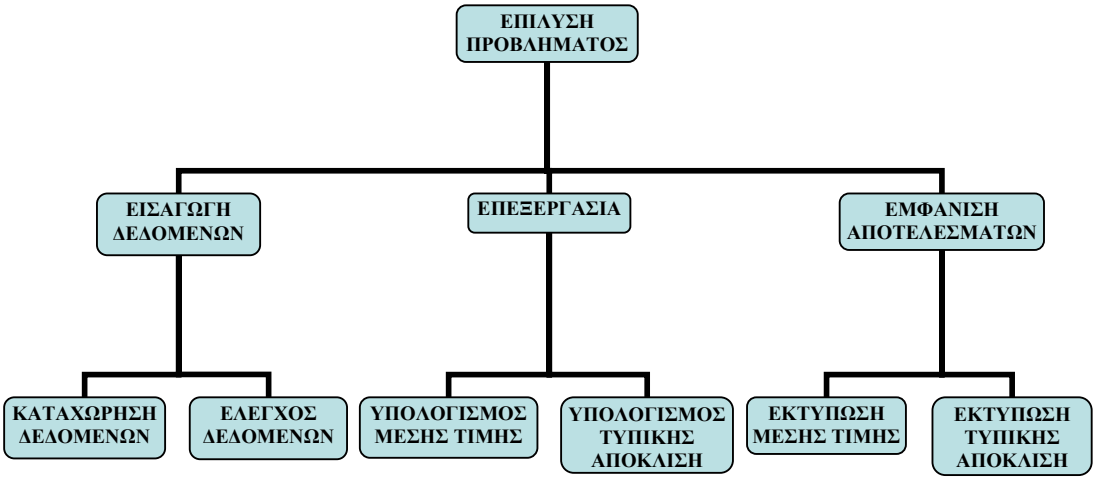
Κεφάλαιο 7ο: Συναρτήσεις και Υπορουτίνες

7.1 Ο Τμηματικός Προγραμματισμός

Η επίλυση ενός προβλήματος πολλές φορές ανάγεται στην επίλυση άλλων απλούστερων προβλημάτων, τα οποία με τη σειρά τους μπορούν να ανάγονται σε επίσης απλούστερα προβλήματα. Η τεχνική αυτή είναι γνωστή ως τεχνική του **τμηματικού προγραμματισμού** και αποτελεί ένα από τα βασικότερα χαρακτηριστικά του δομημένου προγραμματισμού.

Παράδειγμα: Να γραφεί πρόγραμμα το οποίο θα διαβάζει τους βαθμούς και τα ονοματεπώνυμα Ν φοιτητών και στη συνέχεια θα υπολογίζει και θα εμφανίζει τον μέσο όρο και την τυπική απόκλιση των βαθμών.

Η επίλυση του παραπάνω προβλήματος μπορεί να αναχθεί στα επιμέρους υποπροβλήματα:



Όπως φαίνεται το αρχικό πρόβλημα διασπάστηκε σε αρκετά απλούστερα υποπροβλήματα. Η δημιουργία λοιπόν του τελικού προγράμματος ανάγεται στη δημιουργία των επί μέρους τμημάτων προγραμμάτων και τη σύνδεση αυτών μεταξύ τους. Μερικά από αυτά τα τμήματα, όπως ο υπολογισμός της μέσης τιμής, μπορεί να τα έχουμε αντιμετωπίσει και σε άλλα προβλήματα, οπότε μας είναι γνωστά.

Όταν ένα τμήμα προγράμματος επιτελεί αυτόνομο έργο και έχει γραφεί ξεχωριστά από το υπόλοιπο πρόγραμμα, τότε αναφερόμαστε σε **υποπρόγραμμα** (Subprogram).

7.1.1 Χαρακτηριστικά των υποπρογραμμάτων

Ο χωρισμός ενός προγράμματος σε υποπρογράμματα προϋποθέτει (όπως είδαμε) την ανάλυση του αρχικού προβλήματος σε μικρότερα υποπροβλήματα, τα οποία μπορούν να αντιμετωπισθούν ανεξάρτητα το ένα από το άλλο. Η ανάλυση αυτή δεν είναι πάντα εύκολη και δεν υπάρχουν συγκεκριμένοι κανόνες για την επιτυχή ανάλυση. Η δυσκολία αυξάνεται όσο πιο μεγάλο και πιο σύνθετο είναι το πρόβλημα. Η σωστή εφαρμογή του τμηματικοί

προγραμματισμού απαιτεί μελέτη στην ανάλυση του προβλήματος, εμπειρία στο προγραμματισμό, ταλέντο και φυσικά γνώσεις.

Υπάρχουν πάντως τρεις ιδιότητες που πρέπει να διακρίνουν τα υποπρογράμματα:

- ***Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο.*** Στην πραγματικότητα κάθε υποπρόγραμμα ενεργοποιείται με την είσοδο σε αυτό που γίνεται πάντοτε από την αρχή του, εκτελεί ορισμένες ενέργειες, και απενεργοποιείται με την έξοδο από αυτό που γίνεται πάντοτε από το τέλος του.

- ***Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα.*** Αυτό σημαίνει ότι κάθε υποπρόγραμμα μπορεί να σχεδιαστεί, να αναπτυχθεί και να συντηρηθεί αυτόνομα χωρίς να επηρεαστούν άλλα υποπρογράμματα. Στην πράξη βέβαια η απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί.

- ***Κάθε υποπρόγραμμα πρέπει να μην είναι πολύ μεγάλο.*** Η έννοια του μεγάλου προγράμματος είναι υποκειμενική, αλλά πρέπει κάθε υποπρόγραμμα να είναι τόσο, ώστε να είναι εύκολα κατανοητό για να μπορεί να ελέγχεται. Γενικά κάθε υποπρόγραμμα πρέπει να εκτελεί μόνο μία λειτουργία. Αν εκτελεί περισσότερες λειτουργίες, τότε συνήθως μπορεί και πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

7.1.2 Πλεονεκτήματα του τμηματικού προγραμματισμού

Η σωστή χρήση του τμηματικού προγραμματισμού, δηλαδή ο σωστός χωρισμός ενός σύνθετου προγράμματος σε υποπρογράμματα εξασφαλίζει τέσσερα βασικά χαρακτηριστικά του σωστού προγραμματισμού:

Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος.

Επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι στην αντιμετώπιση του συνολικού προβλήματος. Με τη σταδιακή επίλυση των υποπροβλημάτων και τη δημιουργία των αντιστοίχων υποπρογραμμάτων τελικά επιλύεται το συνολικό πρόβλημα.

Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.

Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση ενός συγκεκριμένου τμήματος του χωρίς οι αλλαγές αυτές να επηρεάσουν όλο το υπόλοιπο πρόγραμμα. Επίσης διευκολύνει οποιονδήποτε χρειαστεί να διαβάσει και να κατανοήσει τον τρόπο που λειτουργεί το πρόγραμμα. Αυτό είναι ένα πολύ σημαντικό χαρακτηριστικό του σωστού προγραμματισμού, αφού ένα μεγάλο πρόγραμμα στον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.

Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.

Πολύ συχνά χρειάζεται η ίδια λειτουργία σε διαφορετικά σημεία ενός προγράμματος. Από τη στιγμή που ένα υποπρόγραμμα έχει γραφεί, μπορεί το ίδιο να καλείται από πολλά σημεία του προγράμματος. Έτσι μειώνονται το μέγεθος του προγράμματος, ο χρόνος που απαιτείται για τη συγγραφή του και οι πιθανότητες λάθους, ενώ ταυτόχρονα το πρόγραμμα γίνεται πιο εύληπτο και κατανοητό.

Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.

Ένα υποπρόγραμμα που έχει γραφεί μπορεί να χρησιμοποιηθεί πολύ εύκολα και σε άλλα προγράμματα. Από τη στιγμή που έχει δημιουργηθεί, η χρήση του δεν διαφέρει από τη χρήση των ενσωματωμένων συναρτήσεων που παρέχει η γλώσσα προγραμματισμού, όπως για τον υπολογισμό του ημίτονου ή του συνημίτονου ή την εντολή με την οποία εκτελεί μία συγκεκριμένη διαδικασία. Αν λοιπόν χρειάζεται συχνά κάποια λειτουργία που δεν υποστηρίζεται απευθείας από τη γλώσσα, όπως για παράδειγμα η εύρεση του μικρότερου δύο αριθμών, τότε μπορεί να γραφεί το αντίστοιχο υποπρόγραμμα. Η συγγραφή πολλών υποπρογραμμάτων και η δημιουργία βιβλιοθηκών με αυτά, ουσιαστικά επεκτείνουν την ίδια τη γλώσσα προγραμματισμού.

7.1.3 Παράμετροι

Τα υποπρογράμματα ενεργοποιούνται από κάποιο άλλο πρόγραμμα ή υποπρόγραμμα για να εκτελέσουν συγκεκριμένες λειτουργίες.

Κάθε υποπρόγραμμα για να ενεργοποιηθεί καλείται, όπως λέγεται, από ένα άλλο υποπρόγραμμα ή το αρχικό πρόγραμμα, το οποίο ονομάζεται κύριο πρόγραμμα. Μέχρι και τη **Fortran 77** δεν ήταν δυνατόν ένα υποπρόγραμμα να καλεί τον εαυτό του, δηλαδή να προκαλείται αναδρομή (Recursion). Με τη **Fortran 90/95** αυτό επιτρέπεται όπως και σε πολλές άλλες γλώσσες.

Το υποπρόγραμμα είναι αυτόνομο και ανεξάρτητο τμήμα προγράμματος, αλλά συχνά πρέπει να επικοινωνεί με το υπόλοιπο πρόγραμμα. Συνήθως δέχεται τιμές από το τμήμα προγράμματος που το καλεί και μετά την εκτέλεση επιστρέφει σε αυτό νέες τιμές, αποτελέσματα.

Οι τιμές αυτές που περνούν από το ένα υποπρόγραμμα στο άλλο λέγονται παράμετροι.

Οι παράμετροι λοιπόν είναι σαν τις κοινές μεταβλητές ενός προγράμματος με μία ουσιώδη διαφορά, χρησιμοποιούνται για να περνούν τιμές στα υποπρογράμματα.

Οι παράμετροι χωρίζονται σε δύο κατηγορίες:

α) **Τυπικές** παράμετροι (Formal arguments)

β) **Συγκεκριμένες** παράμετροι (Actual arguments).

Θα λέμε ότι μία παράμετρος είναι τυπική όταν εμφανίζεται κατά τη σύνταξη του υποπρογράμματος και μπορεί να είναι μια μεταβλητή οποιουδήποτε τύπου.

Θα λέμε ότι μία παράμετρος είναι συγκεκριμένη όταν εμφανίζεται στην κλήση του υποπρογράμματος και μπορεί να είναι είτε μια μεταβλητή είτε μια σταθερά είτε ακόμη μια αριθμητική ή λογική παράσταση.

Προσοχή: Πρέπει πάντοτε οι αντίστοιχες τυπικές και συγκεκριμένες παράμετροι να συμφωνούν ως προς το πλήθος, τη σειρά και τον τύπο. Και αυτό επειδή κατά την εκτέλεση του προγράμματος οι τιμές των συγκεκριμένων παραμέτρων καταλαμβάνουν τις θέσεις μνήμης των τυπικών παραμέτρων.

Διακρίνουμε ακόμη τις παραμέτρους σε 2 κατηγορίες:

α) Παράμετροι **εισόδου**

β) Παράμετροι **εξόδου**

Παράμετροι εισόδου, είναι εκείνες οι τυπικές ή συγκεκριμένες παράμετροι που χρησιμεύουν για να περάσουν τα δεδομένα στο υποπρόγραμμα.

Παράμετροι εξόδου, είναι εκείνες οι τυπικές ή συγκεκριμένες παράμετροι που χρησιμεύουν για να περάσουν τα αποτελέσματα από το υποπρόγραμμα στο κύριο πρόγραμμα.

Σημείωση: Η ονομασία των τυπικών παραμέτρων δεν έχει καμιά σχέση με τα ονόματα των μεταβλητών μέσα στο κύριο πρόγραμμα. Έτσι μπορούμε να έχουμε το ίδιο όνομα και σαν τυπική παράμετρο και σαν μεταβλητή του προγράμματος απ' όπου γίνεται η κλήση. Η εξήγηση είναι απλή: Σ' άλλο χώρο της μνήμης αποθηκεύεται το κύριο πρόγραμμα και σ' άλλο το υποπρόγραμμα με τις τυπικές παραμέτρους.

7.1.4 Διαδικασίες και συναρτήσεις

Υπάρχουν δύο ειδών υποπρογράμματα, οι **διαδικασίες** και οι **συναρτήσεις**. Το είδος κάθε υποπρογράμματος καθορίζεται από το είδος της λειτουργίας που καλείται να επιτελέσει.

Οι διαδικασίες μπορούν να εκτελέσουν οποιαδήποτε λειτουργία από αυτές που μπορεί να εκτελέσει ένα πρόγραμμα. Να εισάγουν δεδομένα, να εκτελέσουν υπολογισμούς, να μεταβάλλουν τις τιμές των μεταβλητών και να τυπώσουν αποτελέσματα. Με τη χρήση των παραμέτρων αυτές τις τιμές μπορούν να τις μεταφέρουν και στα άλλα υποπρογράμματα.

Αντίθετα η λειτουργία των συναρτήσεων είναι πιο περιορισμένη. Οι συναρτήσεις υπολογίζουν μόνο μία τιμή, αριθμητική, χαρακτήρα ή λογική και μόνο αυτήν επιστρέφουν στο υποπρόγραμμα που την κάλεσε. Οι συναρτήσεις μοιάζουν με τις συναρτήσεις των μαθηματικών και η χρήση τους είναι όμοια με τη χρήση των ενσωματωμένων συναρτήσεων που υποστηρίζει η γλώσσα προγραμματισμού.

Ο τρόπος κλήσης καθώς και ο τρόπος σύνταξης των δύο αυτών τύπων των υποπρογραμμάτων είναι διαφορετικός. Τόσο οι συναρτήσεις όσο και οι διαδικασίες τοποθετούνται μετά το τέλος του κυρίου προγράμματος. Οι συναρτήσεις εκτελούνται απλά με την εμφάνιση του ονόματος τους σε οποιαδήποτε έκφραση, ενώ για να εκτελεστούν οι διαδικασίες χρησιμοποιείται η ειδική εντολή και το όνομα της διαδικασίας. Για τη Fortran, αυτή η εντολή είναι η **Call**.

Συνοψίζοντας, μπορούμε να πούμε ότι

- Η **συνάρτηση** είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή με το όνομά της (όπως οι μαθηματικές συναρτήσεις).

- Η **διαδικασία** είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος.

Στην Fortran τα δύο είδη υποπρογραμμάτων είναι οι **συναρτήσεις (Functions)** και οι **υπορουτίνες (Subroutines)**, όπως ονομάζονται οι διαδικασίες.

7.2 Συναρτήσεις (Functions) στην Fortran

Η γενική μορφή δήλωσης ενός υποπρογράμματος τύπου **Function** (συνάρτηση) είναι:

τύπος Function "όνομα" ("τυπικές παράμετροι")
Δηλώσεις των μεταβλητών
Κώδικας της συνάρτησης
Return
End

όπου:

τύπος είναι ένας από τους γνωστούς και αποδεκτούς τύπους των μεταβλητών,

"όνομα" είναι το όνομα που δίνουμε στη **Function**,

"**τυπικές παράμετροι**" είναι μια σειρά από μεταβλητές χωρισμένες με υποδιαστολές.

Όταν απουσιάζει η δήλωση του **τύπου** της **Function** τότε, το **"όνομα"** δείχνει και τον τύπο της. Αν το **"όνομα"** αρχίζει μ' ένα από τα γράμματα **I, J, K, L, M, N** τότε είναι μία συνάρτηση που δίνει σαν αποτέλεσμα ακέραια τιμή και αν αρχίζει μ' ένα από τα υπόλοιπα γράμματα είναι μία συνάρτηση που δίνει σαν αποτέλεσμα πραγματική τιμή.

Σημείωση: Όταν δηλώνουμε μια συνάρτηση με ένα συγκεκριμένο τύπο (π.χ. **Integer**), θα πρέπει και στο κυρίως πρόγραμμα, το όνομα αυτής της συνάρτησης να δηλωθεί με τον ίδιο τύπο.

Στις συναρτήσεις οι τυπικές παραμέτρους είναι μόνο παράμετροι εισόδου.

Αν μια ή περισσότερες τυπικές παράμετροι αντιστοιχούν σε πίνακες, τότε αυτές στο τμήμα δηλώσεων των μεταβλητών μπορούν να δηλωθούν είτε:

- με αριθμητικές σταθερές ίσες με την πραγματική διάσταση των πινάκων,
- με το όνομα **ακέραιων** μεταβλητών που πρέπει να βρίσκονται ανάμεσα στις τυπικές παραμέτρους.

Παρατήρηση: Όταν η μεταβλητή πίνακας δεν αναφέρεται σε τυπική παράμετρο αλλά σε μια τοπική μεταβλητή του υποπρογράμματος θα πρέπει να δηλωθεί κανονικά η διάσταση του πίνακα με αριθμητικές σταθερές.

Αμέσως μετά τις **δηλώσεις των μεταβλητών** γράφουμε τον **κώδικα της συνάρτησης** ο οποίος περιγράφει το τμήμα του αλγόριθμου που πρόκειται να πραγματοποιήσουμε με την συνάρτηση.

Μια **Function** τελειώνει πάντα με την εντολή **End**, όπως και το κυρίως πρόγραμμα.

Μετά την εκτέλεση των πράξεων μιας **Function** πρέπει ο έλεγχος για τη συνέχιση των πράξεων να επιστρέψει στο κυρίως πρόγραμμα. Αυτό το υλοποιεί η εντολή **Return**. Η εντολή **Return**, δείχνει στο μεταγλωττιστή ότι τελείωσαν οι πράξεις του υποπρογράμματος και επιτρέπει την επιστροφή στο κυρίως πρόγραμμα. Μπορεί να υπάρχουν και περισσότερες από

μία εντολές **Return** ή και καμία, μέσα σε μια **Function**, ανάλογα με τις διακλαδώσεις του υποπρογράμματος.

Η **Function** στο μοναδικό σημείο που διαφέρει ουσιαστικά από μια **Subroutine** είναι ότι η **Function** υπολογίζει μόνο μία τιμή η οποία μεταφέρεται μέσω του ονόματος της **Function**. Πρέπει λοιπόν, να υπάρχει το λιγότερο μια φορά, μια εντολή που να δίνει τιμή στο όνομα της **Function** και πιο συχνά είναι μια εντολή αντικατάστασης.

7.2.1 Κλήση μιας Function

Η κλήση μιας **Function** γίνεται πολύ απλά με την εμφάνιση του ονόματός της σε οποιαδήποτε έκφραση, σαν μια συνάρτηση βιβλιοθήκης, π.χ.

X = RIZA (5.2, 3.5, 0.3)

όπου RIZA είναι το όνομα της **Function** και οι αριθμητικές σταθερές είναι οι συγκεκριμένες παράμετροι.

Δεν πρέπει να ξεχνάμε ότι οι συγκεκριμένες παράμετροι είναι πάντοτε εισόδου και ότι το αποτέλεσμα μεταφέρεται με την απλή κλήση της **Function** μέσα από το όνομά της.

Έτσι, αν το όνομα μιας **Function** που θα υπολογίζει τη λύση της εξίσωσης 2ου βαθμού με τρεις παραμέτρους είναι RIZA τότε, μετά την εκτέλεση της εντολής:

X = RIZA (5.2, 3.5, 0.3)

η λύση της εξίσωσης θα είναι το X.

7.3 Υπορουτίνες (Subroutines) στην Fortran

Η γενική μορφή δήλωσης ενός υποπρογράμματος τύπου **Subroutine** (υπορουτίνα) είναι:

Subroutine "όνομα" ("τυπικές παράμετροι")

Δηλώσεις των μεταβλητών

Κώδικας της υπορουτίνας

Return

End

όπου:

"**όνομα**" είναι το όνομα που δίνουμε στη **Subroutine**,

"**τυπικές παράμετροι**" είναι μια σειρά από μεταβλητές χωρισμένες με υποδιαστολές.

Οι τυπικές παράμετροι μπορεί να είναι εισόδου και εξόδου ή μόνο εισόδου ή μόνο εξόδου. Μπορεί ακόμη και να μην υπάρχουν καθόλου παράμετροι. Τέλος μπορεί μια τυπική παράμετρος να είναι ταυτόχρονα παράμετρος εισόδου και εξόδου.

Αν μια ή περισσότερες τυπικές παράμετροι αντιστοιχούν σε πίνακες, τότε αυτές στο τμήμα δηλώσεων των μεταβλητών μπορούν να δηλωθούν είτε:

- με αριθμητικές σταθερές ίσες με την πραγματική διάσταση των πινάκων,

- με το όνομα **ακέραιων** μεταβλητών που πρέπει να βρίσκονται ανάμεσα στις τυπικές παραμέτρους.

Παρατήρηση: Όταν η μεταβλητή πίνακας δεν αναφέρεται σε τυπική παράμετρο αλλά σε μια τοπική μεταβλητή του υποπρογράμματος θα πρέπει να δηλωθεί κανονικά η διάσταση του πίνακα με αριθμητικές σταθερές.

Αμέσως μετά τις **δηλώσεις των μεταβλητών** γράφουμε τον **κώδικα της υπορουτίνας** ο οποίος περιγράφει το τμήμα του αλγόριθμου που πρόκειται να πραγματοποιήσουμε με την υπορουτίνα.

Μια **Subroutine** τελειώνει πάντα με την εντολή **End**, όπως και το κυρίως πρόγραμμα.

Μετά την εκτέλεση των πράξεων μιας **Subroutine** πρέπει ο έλεγχος για τη συνέχιση των πράξεων να επιστρέψει στο κυρίως πρόγραμμα. Αυτό το υλοποιεί η εντολή **Return**. Η εντολή **Return**, δείχνει στο μεταγλωττιστή ότι τελείωσαν οι πράξεις του υποπρογράμματος και επιτρέπει την επιστροφή στο κυρίως πρόγραμμα. Μπορεί να υπάρχουν και περισσότερες από μία εντολές **Return** ή και καμία, μέσα σε μια **Subroutine**, ανάλογα με τις διακλαδώσεις του υποπρογράμματος.

7.3.1 Κλήση μιας Subroutine

Η κλήση μιας **Subroutine** γίνεται με τη βοήθεια της εντολής **Call**.

Η γενική μορφή της εντολής **Call** είναι:

Call “όνομα” (“συγκεκριμένες παράμετροι”)

Το **“όνομα”** είναι εκείνο το συμβολικό όνομα που ήδη έχουμε δώσει κατά τη σύνταξη της **Subroutine** με τη δηλωτική εντολή:

Subroutine “όνομα” (“τυπικές παράμετροι”)

“Συγκεκριμένες παράμετροι” είναι μια σειρά από μεταβλητές ή σταθερές για τις οποίες έχει ζητηθεί η εκτέλεση του υποπρογράμματος.

Προσοχή: πρέπει να αντιστοιχούν οι συγκεκριμένες παράμετροι στο ίδιο πλήθος, στην ίδια σειρά και στον ίδιο τύπο με τις τυπικές παραμέτρους.

Αντίθετα από τις τυπικές παραμέτρους, οι συγκεκριμένες παράμετροι εισόδου εκτός από μεταβλητές μπορεί να είναι και αριθμητικές ή λογικές εκφράσεις, σταθερές ή ακόμη και κλήσεις άλλων συναρτήσεων.

Κάθε συγκεκριμένη παράμετρος εισόδου πρέπει πριν από την κλήση του υποπρογράμματος να έχει ήδη πάρει μία τιμή έτσι ώστε, να μη δημιουργηθεί πρόβλημα με τις απροσδιόριστες τιμές των μεταβλητών. Έτσι, μία συγκεκριμένη παράμετρος εισόδου μπορεί και πρέπει να πάρει τιμή είτε από μια εντολή αντικατάστασης, είτε από μια εντολή **Read** είτε ακόμη να είναι μια παράμετρος εξόδου άλλου υποπρογράμματος.

7.4 Λειτουργία της κλήσης των υποπρογραμμάτων.

Για να γίνει πιο κατανοητή η λειτουργία της κλήσης των υποπρογραμμάτων ας παρακολουθήσουμε το παράδειγμα που ακολουθεί:

Έστω ότι, το κυρίως πρόγραμμα καλεί δύο φορές το υποπρόγραμμα ALFA (αδιάφορο αν είναι **Subroutine** ή **Function**) και το υποπρόγραμμα ALFA καλεί μία φορά το υποπρόγραμμα ΒΗΤΑ. Έστω ακόμη ότι, στη μνήμη έχουν καταχωρηθεί οι εντολές σύμφωνα με το παρακάτω σχήμα.

Η διαδικασία που ακολουθείται στη μνήμη του υπολογιστή είναι:

Σταδιακά εκτελούνται μία-μία οι εντολές του κυρίως προγράμματος από πάνω προς τα κάτω.

Όταν ο έλεγχος φθάσει στην πρώτη κλήση του υποπρογράμματος ALFA, η αμέσως επόμενη εντολή η οποία θα εκτελεστεί θα είναι η πρώτη εντολή του υποπρογράμματος ALFA δηλαδή, ο έλεγχος μεταφέρεται μέσα σ' αυτό το υποπρόγραμμα.

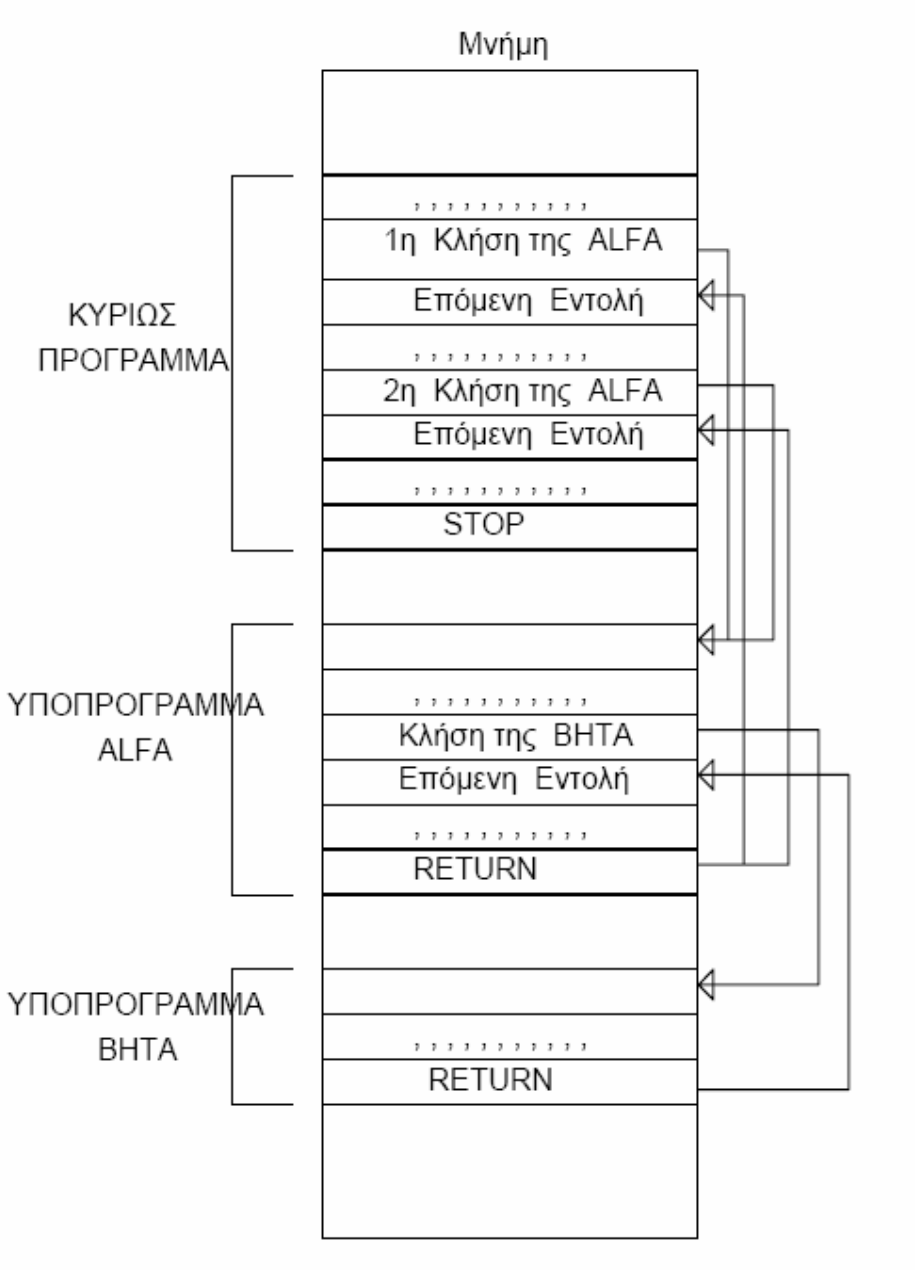
Όταν συναντηθεί η κλήση του υποπρογράμματος ΒΗΤΑ, μέσα στο υποπρόγραμμα ALFA, πάλι η αμέσως επόμενη εντολή που θα εκτελεστεί θα είναι η πρώτη εντολή του υποπρογράμματος ΒΗΤΑ.

Όταν τώρα ο έλεγχος φθάσει στην εντολή **Return** (ή στην εντολή **End**) του υποπρογράμματος ΒΗΤΑ, τότε γυρίζει στην επόμενη από την κλήση εντολή του υποπρογράμματος ALFA.

Συνεχίζονται να εκτελούνται οι εντολές του υποπρογράμματος ALFA και όταν ο έλεγχος φθάσει στην εντολή **Return** (ή στην εντολή **End**) τότε επιστρέφει για τη συνέχιση, στην εντολή που βρίσκεται ακριβώς μετά από την πρώτη κλήση του υποπρογράμματος ALFA.

Ακριβώς η ίδια διαδικασία ακολουθείται όταν συναντηθεί η δεύτερη κλήση του υποπρογράμματος ALFA, όπως χαρακτηριστικά δείχνουν τα βέλη.

Η εκτέλεση των πράξεων, δηλαδή το πρόγραμμα, σταματά όταν θα έρθει η ώρα να εκτελεστεί η εντολή του κυρίως προγράμματος **Stop** (ή η εντολή **End**).


Μνήμη
ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ
1η Κλήση της ALFA
Επόμενη Εντολή
2η Κλήση της ALFA
Επόμενη Εντολή
STOP
ΥΠΟΠΡΟΓΡΑΜΜΑ ALFA
Κλήση της ΒΗΤΑ
Επόμενη Εντολή
RETURN
ΥΠΟΠΡΟΓΡΑΜΜΑ ΒΗΤΑ
RETURN

7.4 Παραδείγματα

Παράδειγμα 1: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο δύο πραγματικούς αριθμούς x, x_0 και θα υπολογίζει την αριθμητική παράσταση

$$y = \frac{f(x) - f(x_0)}{x - x_0}$$

όπου

$$f(x) = \begin{cases} x^2 + 2, & \alpha\nu \quad x \geq 2 \\ x + 4 & \alpha\nu \quad x < 2 \end{cases}.$$

```
Program fun1

Implicit None
Real x, x0, y, f

Write(*,*)'Give 2 real numbers'
Read(*,*) x, x0

If (x.EQ.x0) Then
   Write(*,*)'Oi arithmoi prepei na einai diaforetikoi'
Else
   y=(f(x)-f(x0))/(x-x0)
   Write(*,*) 'y=',y
End if

End


Real Function f(x)

Implicit None
Real x

If (x.GE.2) Then
   f = x**2 + 2
Else
   F = x + 4
End if

End
```

Παράδειγμα 2: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τα στοιχεία ενός πινάκα Α ακεραίων αριθμών διάστασης NxN και θα υπολογίζει (με τη βοήθεια συνάρτησης) και θα εμφανίζει το άθροισμα των διαγωνίων στοιχείων του.

```
Program fun2

Implicit None
Integer n
Parameter (n=3)
Integer A(n,n), i, j, diagon

Write(*,*) 'Give the elements of array A'
Do i = 1, n
   Do j = 1, n
      Write(*,*)'Give the',i,',',j, 'element'
      Read(*,*) A(i,j)
   End Do
End Do

Write(*,*)'To athroisma tvn diagwniwn stoixeien einai:'
Write(*,*) diagon(A,n)

End


Integer Function diagon(x,dim)

Implicit None
Integer dim, i
Integer x(dim,dim)

diagon=0
Do i=1,dim
   diagon=diagon+x(i,i)
End Do

End
```

Παράδειγμα 3: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο δύο ακεραίους αριθμούς, θα τοποθετεί στη μεταβλητή n το μεγαλύτερο από τους 2 αριθμούς και στη μεταβλητή m το μικρότερο και στη συνέχεια θα υπολογίζει την παράσταση:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}.$$

```
Program fun3

Implicit None
Integer n, m, temp
Real*8 x, paragontiko

Write(*,*) 'Give two integer numbers'
Read(*,*) n, m

If ((n.LT.0).OR.(m.LT.0)) Then
   Write(*,*) 'Oi arithmoi prepei na einai thetikoi'
Else
   If (n.LT.m) Then
      temp = n
      n = m
      m = temp
   End if
   x = paragontiko(n)/(paragontiko(m)*paragontiko(n-m))
   Write(*,*) 'H timi tis parastasis einai =', x
End if

End


Real*8 Function paragontiko (x)

Implicit None
Real*8 p
Integer i, x

If (x.EQ.0) Then
   paragontiko = 1
Else
   p = 1
   Do i = 1, x
      p = p * i
   End Do
   paragontiko = p
End if

End
```

Παράδειγμα 4: Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο δύο ακεραίους αριθμούς a, b και θα υπολογίζει (με τη βοήθεια υπορουτίνας) και θα εμφανίζει το άθροισμά τους.

```
Program SubRout1

Implicit None
Integer a, b, c

Write(*,*) 'Give two integer numbers'
Read(*,*) a, b

Call summ(a,b,c)

Write(*,*)'Sum=',c

End


Subroutine summ (x, y, z)

Implicit None
Integer x, y, z

z=x+y

End
```

Παράδειγμα 5: Να γραφεί πρόγραμμα (με τη βοήθεια υπορουτινών) που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο διανυσμάτων Α και Β ακεραίων αριθμών διάστασης Ν και θα υπολογίζει και θα εμφανίζει το άθροισμά τους.

```
Program SumVec

Implicit None
Integer n
Parameter (n=8)
Integer A(n), B(n), C(n)

Call read_vec (A,n)
Call read_vec (B,n)
Call sum_vec (A,B,C,n)
Call write_vec(C,n)

End
```

```
Subroutine read_vec (x, dim)

Implicit None
Integer dim, i, x(dim)

Do i=1,dim
   Write(*,*) 'Give the',i, ' element'
   Read(*,*) x(i)
End Do

End


Subroutine write_vec (x, dim)

Implicit None
Integer dim, i, x(dim)

Do i=1,dim
   Write(*,*) x(i)
End Do

End


Subroutine sum_vec (x, y, z, dim)

Implicit None
Integer dim, i, x(dim), y(dim), z(dim)

Do i=1,dim
   z(i)=x(i)+y(i)
End Do

End
```

Παράδειγμα 6: Να γραφεί πρόγραμμα (με τη βοήθεια υπορουτινών) που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο πινάκων Α και Β ακεραίων αριθμών διάστασης NxM και θα υπολογίζει και θα εμφανίζει το άθροισμά τους.

```
Program SumArray

Implicit None
Integer n, m
Parameter (n=3, m=2)
Integer A(n,m), B(n,m), C(n,m)

Call read_array (A, n, m)
Call read_array (B, n, m)
Call sum_array (A, B, C, n, m)
Call write_array (C, n, m)

End

Subroutine read_array (x, dim1, dim2)
Implicit None
Integer dim1, dim2, i, j, x(dim1,dim2)

Do i=1,dim1
   Do j=1,dim2
      Write(*,*) 'Give the',i,',',j,' element'
      Read(*,*) x(i,j)
   End Do
End Do

End

Subroutine write_array (x, dim1, dim2)
Implicit None
Integer dim1, dim2, i, j, x(dim1,dim2)

Do i=1,dim1
   Write(*,*) (x(i,j),j=1,dim2)
End Do

End

Subroutine sum_array (x, y, z, dim1, dim2)
Implicit None
Integer dim1, dim2, i, j, x(dim1,dim2), y(dim1,dim2), z(dim1,dim2)

Do i=1,dim1
   Do j=1,dim2
      z(i,j)=x(i,j)+y(i,j)
   End Do
End Do

End
```

Παράδειγμα 7: Να γραφεί πρόγραμμα (με χρήση συναρτήσεων και υπορουτινών) που θα διαβάζει από το πληκτρολόγιο τα στοιχεία δύο πινάκων Α, Β πραγματικών αριθμών διάστασης NxM, και θα εκτυπώνει τον πίνακα που εμφανίζει το μεγαλύτερο μέγιστο στοιχείο.

```
Program SubRout2

Implicit None
Integer n, m
Parameter (n=3, m=2)
Real A(n,m), B(n,m), maximum

Call read_array (A, n, m)
Call read_array (B, n, m)

If (maximum(A,n,m).GE.maximum(B,n,m)) Then
   Call write_array (A, n, m)
Else
   Call write_array (B, n, m)
End if

End

Subroutine read_array (x, dim1, dim2)

Implicit None
Integer dim1, dim2, i, j
Real x(dim1,dim2)

Do i=1,dim1
   Do j=1,dim2
      Write(*,*) 'Give the',i,',',j,' element'
      Read(*,*) x(i,j)
   End Do
End Do

End

Subroutine write_array (x, dim1, dim2)

Implicit None
Integer dim1, dim2, i, j
Real x(dim1,dim2)

Do i=1,dim1
   Write(*,*) (x(i,j),j=1,dim2)
End Do

End
```

```
Real Function maximum(x, dim1, dim2)

Implicit None
Integer dim1, dim2, i, j
Real x(dim1,dim2)

maximum = x(1,1)

Do i=1,dim1
   Do j=1,dim2
      If (maximum.LE.x(i,j)) Then
         maximum = x(i,j)
      End if
   End Do
End Do

End
```

Κεφάλαιο 8ο: Αρχεία

Ας υποθέσουμε ότι σε μια στατιστική έρευνα, φοιτητές συλλέγουν 100 διαφορετικές τιμές μιας τυχαίας μεταβλητής. Κύριος σκοπός της έρευνας είναι ο υπολογισμός της μέσης τιμής και της τυπικής απόκλισης των παραπάνω τιμών. Προκειμένου να υλοποιήσουμε ένα πρόγραμμα για το υπολογισμό της μέσης τιμής και της τυπικής απόκλισης των παραπάνω τιμών, θα χρησιμοποιήσουμε απλές μεταβλητές ή μεταβλητές-πίνακες για να αποθηκεύσουμε προσωρινά τις παραπάνω τιμές πριν τις επεξεργαστούμε. Τα πιθανά προβλήματα που θα αντιμετωπίσουμε κατά τη υλοποίηση και εκτέλεση του προγράμματος, είναι τα εξής:

α) πιθανό λάθος κατά την είσοδο των δεδομένων

β) πιθανό λάθος στο πρόγραμμα,

γ) ξαναεκτέλεση του προγράμματος.

Στις περισσότερες περιπτώσεις, λοιπόν, θα έχουμε επανάληψη της εισαγωγής των δεδομένων. Σίγουρα αυτό δε θα αποτελούσε πρόβλημα αν τα δεδομένα ήταν λίγα.

Ας φανταστούμε, επίσης, ότι δημιουργούμε ένα πρόγραμμα που παράγει ένα μεγάλο πλήθος από αποτελέσματα, π.χ τιμές μιας συνάρτησης f(x) για διαφορετικές τιμές του x = 1, 1.1, 1.2, …, 100. Έστω ότι θέλουμε να χρησιμοποιήσουμε τα αποτελέσματα αυτά από ένα άλλο πρόγραμμα της Fortran ή ακόμα και από άλλα προγράμματα, όπως το MATLAB, για να σχεδιάσουμε τη γραφική παράσταση της συνάρτησης. Με ποιο τρόπο θα συλλέξουμε τις τιμές αυτές;

Παρατηρούμε, λοιπόν, ότι αν έχουμε μεγάλο πλήθος δεδομένων να επεξεργαστούμε ή αν τα αποτελέσματα του προγράμματός μας πρέπει να χρησιμοποιηθούν από άλλα προγράμματα ή αν ακόμα το πρόγραμμά μας πρέπει να εκτελεστεί πολλές φορές (πιθανώς λόγω λαθών) για το ίδιο σύνολο δεδομένων, θα πρέπει να βρούμε ένα τρόπο μόνιμης αποθήκευσης των δεδομένων και την δυνατότητα ανάκλησή τους. Τη λύση σε προβλήματα τέτοιας φύσης δίνουν τα αρχεία τα οποία θα μελετήσουμε στο κεφάλαιο αυτό.

8.1 Τι είναι αρχείο;

Το **αρχείο** (file) δεν είναι τίποτα άλλο από μια οργανωμένη συλλογή δεδομένων. Τα δεδομένα αυτά αποθηκεύονται σε μια δευτερεύουσα μνήμη του Η/Υ όπως η δισκέτα, ο σκληρός δίσκος κ.τ.λ. και χαρακτηρίζονται από ένα μοναδικό όνομα. Ο χρήστης δε χρειάζεται να γνωρίζει πως είναι τοποθετημένο ένα αρχείο στη δευτερεύουσα μνήμη, αρκεί να γνωρίζει το όνομά του. Κάθε αρχείο αποτελείται από εγγραφές. Μια **εγγραφή** (record) είναι ένα σύνολο από δεδομένα που ανήκουν στην ίδια λογική ενότητα. Κάθε εγγραφή αποτελείται από επιμέρους περιοχές, οι οποίες διατηρούνται για ένα συγκεκριμένο τύπο δεδομένων. Οι περιοχές αυτές στις οποίες χωρίζεται η εγγραφή ονομάζονται **πεδία** (fields). Χαρακτηριστικά του πεδίου είναι η μορφή (αριθμητική, αλφαριθμητική) και το μήκος του.

Με το όρο **οργάνωση** (organization) αρχείου αναφερόμαστε:

α) στον τρόπο με τον οποίο είναι τοποθετημένες οι εγγραφές σε ένα αρχείο και

β) στη μεθοδολογία που χρησιμοποιούμε για εντοπίσουμε που έχει αποθηκευτεί μια εγγραφή σε ένα αρχείο.

Με τον όρο **προσπέλαση** (access) αρχείου, αναφερόμαστε στον τρόπο με τον οποίο γράφουμε εγγραφές σε ένα αρχείο ή στον τρόπο με τον οποίο διαβάζουμε εγγραφές από ένα αρχείο. Ο τρόπος με τον οποίο είναι οργανωμένες οι εγγραφές ενός αρχείου στη δευτερεύουσα μνήμη του Η/Υ, συνεπάγεται και τον τρόπο με τον οποίο θα προσπελάσουμε τις εγγραφές του.

Υπάρχουν δύο διαφορετικοί τρόποι προσπέλασης αρχείων:

Σειριακή προσπέλαση (sequential access): Η προσπέλαση σε ένα αρχείο ονομάζεται σειριακή, όταν κατά τη δημιουργία του οι εγγραφές τοποθετούνται η μία μετά την άλλη στη σειρά και η ανάγνωση των εγγραφών του γίνεται επίσης με τον ίδιο τρόπο, διαβάζοντας δηλαδή τη μια εγγραφή μετά την άλλη, ξεκινώντας από την πρώτη εγγραφή του αρχείου. Αν θέλουμε να προσθέσουμε μια εγγραφή, στα αρχεία σειριακής προσπέλασης, θα πρέπει να την τοποθετήσουμε στη σειρά σε σχέση με τις ήδη υπάρχουσες, δηλαδή στο τέλος του αρχείο. Κάθε άλλη απόπειρα θα συνεπάγονταν την αποκοπή του μέρους του αρχείου που ακολουθεί τη συγκεκριμένη εγγραφή. Αν επίσης θέλουμε να διαβάσουμε την $100^{η}$ εγγραφή ενός αρχείο, θα πρέπει πρώτα να διαβάσουμε τις πρώτες 99. Αν πάλι έχουμε διαβάσει την $100^{η}$ εγγραφή και θέλουμε να διαβάσουμε τη $10^{η}$, θα πρέπει να επιστρέψουμε στην πρώτη εγγραφή του αρχείου και να διαβάσουμε τις πρώτες 9 εγγραφές πριν την $10^{η}$. Σειριακό τρόπο προσπέλασης θα χρησιμοποιούσαμε για να επεξεργαστούμε όλες τις εγγραφές ενός αρχείο π.χ. υπολογισμός των αποδοχών όλων των υπαλλήλων, καταχώρηση των βαθμολογιών όλων των φοιτητών ή και των στοιχείων τους. Επίσης, σειριακό τρόπο προσπέλασης θα χρησιμοποιούσαμε αν είχαμε λίγες εγγραφές σε ένα αρχείο, π.χ. για την αρχειοθέτηση των CD-ROMS που έχουμε σπίτι μας. Τέλος, είναι αρκετά εύκολο για κάποιον να γράψει ένα πρόγραμμα διαχείρισης ενός σειριακού αρχείου.

Άμεση προσπέλαση (random or direct access): Η προσπέλαση σε ένα αρχείο ονομάζεται άμεση, όταν έχουμε τη δυνατότητα να προσπελάσουμε κάθε εγγραφή χωρίς απαραίτητα να προσπελάσουμε οποιαδήποτε άλλη έγγραφη. Αυτό επιτυγχάνεται συνήθως χρησιμοποιώντας ένα πεδίο που χαρακτηρίζει μοναδικά κάθε εγγραφή και ονομάζεται πεδίο κλειδί. Η τιμή του πεδίου αυτού συνδέεται μοναδικά με τη διεύθυνση μνήμης στην οποία είναι καταχωρημένη η συγκεκριμένη εγγραφή. Ας υποθέσουμε ότι έχουμε ένα αρχείο με τα στοιχεία φοιτητών. Η κάθε εγγραφή σε ένα τέτοιο αρχείο θα αποτελούνταν από πεδία, όπως το ονοματεπώνυμο του φοιτητή, ο αριθμός μητρώου και πιθανοί βαθμοί που είχε στα μαθήματα του Τμήματος. Σε αυτή τη περίπτωση το πεδίο που χαρακτηρίζει μοναδικά το φοιτητή είναι ο αριθμός μητρώου του. Χρησιμοποιώντας, λοιπόν, το πεδίο αυτό ως πεδίο κλειδί, μπορούμε να δημιουργήσουμε ένα αρχείο άμεσης προσπέλασης. Τα αρχεία άμεσης προσπέλασης έχουν το πλεονέκτημα σε σχέση με τα αρχεία σειριακής προσπέλασης να μας δίνουν τη δυνατότητα να προσπελαύνουμε μία έγγραφη πολύ πιο γρήγορα (γιατί δε χρειάζεται να προσπελάσουμε όλες τις προηγούμενες εγγραφές).

Αρχεία άμεσης προσπέλασης συνήθως χρησιμοποιούμε όταν έχουμε πολλές εγγραφές, π.χ. στο αρχείο πελατών μιας τράπεζας, στα αρχεία των φοιτητών ενός Τμήματος κ.τ.λ. Επίσης χρησιμοποιούμε αρχεία άμεσης προσπέλασης, όταν μας ενδιαφέρει πολύ ο χρόνος απόκρισης του υπολογιστή, π.χ. στα μηχανήματα ΑΤΜ.

8.2 Εντολές της Fortran για τη διαχείριση αρχείων

Τα βασικά βήματα προσπέλασης ενός αρχείου είναι τα παρακάτω:

1. **Άνοιγμα του αρχείου**. Κατά τη διαδικασία αυτή δηλώνουμε αν το αρχείο είναι σειριακής ή άμεσης προσπέλασης, αν θα δημιουργηθεί για πρώτη φορά ή αν είναι παλιό αρχείο, αν θα γραφεί ή θα διαβαστεί με format και τέλος, αντιστοιχούμε ένα μοναδικό ακέραιο αριθμό στο αρχείο αυτό, ώστε η επικοινωνία μας με το αρχείο αυτό στη συνέχεια του προγράμματος να γίνεται με το συγκεκριμένο αριθμό.

2. **Επεξεργασία αρχείου**. Στη φάση αυτή εισάγουμε εγγραφές στο αρχείο ή διαβάζουμε εγγραφές από αυτό, προκειμένου να τις επεξεργαστούμε.

3. **Κλείσιμο του αρχείου**. Εφόσον τελειώσουμε με την επεξεργασία του αρχείου, κλείνουμε το αρχείο.

8.2.1 Η εντολή Open

Η εντολή **Open** έχει ως σκοπό την σύνδεση ενός ήδη υπάρχοντος αρχείου με μια λογική συσκευή ή τη δημιουργία ενός καινούργιου αρχείου και τη σύνδεσή του με μια λογική συσκευή. Ο προσδιορισμός της λογικής συσκευής η οποία συσχετίζεται με το αρχείο, γίνεται μέσω ενός θετικού ακέραιου αριθμού που δίνεται στη παράμετρο Unit.

Σύνταξη:

Open ([**Unit** =] ακέραιος θετικός αριθμός, **File** = ‘αλφαριθμητική παράσταση’,

[**Access** = {‘Sequential’, ‘Direct’, ‘Append’}], [**Status** = {‘New’, ‘Old’, ‘Scratch’}],

[**Form** = {‘Formatted’, ‘Unformatted’}])

Παρατήρηση: Ότι υπάρχει μέσα σε αγκύλες ([ ]) μπορεί να παραληφθεί κατά τη σύνταξη της εντολής **Open**. Δηλαδή, παρατηρούμε ότι, στη σύνταξη της εντολής **Open** απαραίτητα πρέπει να υπάρχει ο ακέραιος αριθμός μέσω του οποίου το αρχείο συσχετίζεται με μια λογική συσκευή και η παράμετρος **File** με την τιμή της (αλφαριθμητική παράσταση) που είναι το όνομα του αρχείου.

Παράμετροι της εντολής Open:

Unit: Κάθε αρχείο που χρησιμοποιείται σε ένα πρόγραμμα συσχετίζεται με μια λογική συσκευή. Για να προσδιορίσουμε τη λογική αυτή συσκευή, χρησιμοποιούμε την παράμετρο **Unit**. Η λογική συσκευή αυτή συσκευή ορίζεται με ένα θετικό ακέραιο αριθμό, τον οποίο και θα αντιστοιχίσουμε μοναδικά στο αρχείο. Στη συνέχεια όταν θέλουμε να αναφερθούμε στο αρχείο, θα χρησιμοποιούμε τον αριθμό αυτό αντί του ονόματος του αρχείου. Ο αριθμός αυτός είναι προσωρινός και ισχύει όσο το αρχείο είναι ανοικτό. Αν το αρχείο κλείσει, έχουμε τη δυνατότητα να ανοίξουμε το αρχείο με διαφορετικό αριθμό. Επίσης ο αν το αρχείο κλείσει μπορεί ο αριθμός να χρησιμοποιηθεί στο άνοιγμα κάποιου άλλου αρχείου. Ιδιαίτερη προσοχή θα πρέπει να δοθεί στα εξής:

α) Δε θα πρέπει να δώσουμε το ίδιο αριθμό σε δύο διαφορετικά αρχεία που χρησιμοποιούνται την ίδια στιγμή.

β) Επειδή ο αριθμός 5 και το σύμβολο * χρησιμοποιούνται για το πληκτρολόγιο και οι αριθμοί 0, 6 και το σύμβολο * χρησιμοποιούνται για την οθόνη, θα πρέπει, λοιπόν, αν θελήσουμε να χρησιμοποιήσουμε έναν αριθμό για ένα αρχείο, να χρησιμοποιούμε αριθμούς εκτός από 0, 5, 6.

File: Στη θέση αυτή ανάμεσα σε απλά εισαγωγικά τοποθετούμε το όνομα του αρχείου που θέλουμε να δημιουργήσουμε ή το όνομα του αρχείου που θέλουμε να καλέσουμε. Το όνομα του αρχείου θα αποτελείται από δύο μέρη, τα οποία θα χωρίζονται με τελεία. Το πρώτο μέρος είναι υποχρεωτικό και θα πρέπει να αρχίζει με γράμμα της αγγλικής αλφαβήτου, ενώ θα ακολουθεί συνδυασμός γραμμάτων και αριθμών. Το δεύτερος μέρος, που ονομάζουμε και επέκταση, είναι προαιρετικό και θα πρέπει να έχει έως 3 χαρακτήρες. Στην περίπτωση που δεν δηλώσουμε όνομα αρχείου, τότε δημιουργείται ένα προσωρινό σειριακό αρχείο, στο οποίο μπορούμε να γράψουμε (διαβάσουμε) εγγραφές). Σε περίπτωση όμως που κλείσουμε το αρχείο ή διακοπή η εκτέλεση του προγράμματος, τότε το αρχείο αυτό διαγράφεται.

Access: Εδώ θα πρέπει να τοποθετήσουμε τη λέξη 'Sequential' αν πρόκειται για αρχείο σειριακής προσπέλασης ή 'Direct' αν πρόκειται για αρχείο άμεσης προσπέλασης. Σε περίπτωση που παραλείψουμε να δηλώσουμε το είδος προσπέλασης, τότε το αρχείο θεωρείται σειριακής προσπέλασης. Επίσης, χρησιμοποιείται και το 'Append', για να δηλώσουμε τα αρχεία σειριακής προσπέλασης στα οποία θέλουμε να προσθέσουμε δεδομένα στο τέλος.

Status: Εδώ θα πρέπει να τοποθετήσουμε τη λέξη 'New' αν πρόκειται να δημιουργήσουμε καινούριο αρχείο, τη λέξη 'Old' αν πρέπει να καλέσουμε ένα ήδη υπάρχον αρχείο και τέλος τη λέξη 'Scratch' για αρχείο εξόδου που θα διαγραφεί μετά την εκτέλεση του προγράμματος. Σε περίπτωση που δε δηλώσουμε την κατάσταση του αρχείου, τότε ο υπολογιστής αναζητεί αρχείο με το συγκεκριμένο όνομα. Αν δε βρει το αρχείο αυτό, τότε δημιουργεί ένα αρχείο με το όνομα αυτό.

Form: Εδώ θα πρέπει να τοποθετήσουμε τη λέξη 'Formatted' αν θα χρησιμοποιήσουμε τις εντολές **Read** ή **Write** με **Format** για να διαβάσουμε ή να γράψουμε στο αρχείο αυτό, ενώ 'Unformatted' σε κάθε άλλη περίπτωση. Στην περίπτωση που ξεχάσουμε να δηλώσουμε την παράμετρο αυτή, τότε αν το αρχείο είναι σειριακής προσπέλασης, θεωρείται ότι **Form** = 'Formatted', ενώ αν το αρχείο είναι άμεσης προσπέλασης, τότε θεωρείται ότι **Form** = 'Unformatted'.

Παρατήρηση: Στην περίπτωση που δηλώσουμε **Form** = 'Formatted', τα δεδομένα μας τοποθετούνται σε ASCII μορφή στο αρχείο και συνεπώς, μπορούμε να τα δούμε ανοίγοντας το αρχείο με ένα απλό επεξεργαστή κειμένου (editor) π.χ. το Notepad. Σε αντίθετη περίπτωση, τα δεδομένα αποθηκεύονται σε binary μορφή και συνεπώς δεν θα μπορούμε να διαβάσουμε τα δεδομένα του αρχείου ανοίγοντάς το με έναν απλό επεξεργαστή.

8.2.2 Η εντολή Close

Σύνταξη:

Close ([**Unit** =] ακέραιος θετικός αριθμός, [**Status** = {'Keep', 'Delete'}])

Η εντολή **Close** χρησιμοποιείται για την αποσύνδεση του αρχείου που αντιστοιχεί στη λογική συσκευή που ορίζεται στην παράμετρο **Unit** από το πρόγραμμα. Σε περίπτωση που αυτή η

εντολή παραληφθεί, τότε το αρχείο αποσυνδέεται από το πρόγραμμα με το τέλος εκτέλεσης του προγράμματος. Στη παράμετρο **Status** τοποθετούμε τη λέξη 'Keep' αν θέλουμε το αρχείο να διατηρηθεί μετά το κλείσιμο του αρχείου, ενώ τη λέξη 'Delete' αν θέλουμε το αρχείο να διαγραφεί μετά το κλείσιμο του.

8.2.3 Η εντολή Rewind

Σύνταξη:

Rewind ([**Unit** =] ακέραιος θετικός αριθμός)

Επαναφέρει το αρχείο σειριακής προσπέλασης στην πρώτη εγγραφή.

Μπορούμε να φανταστούμε την ύπαρξη ενός νοητού δείκτη, ο οποίος, όταν ανοίγουμε το αρχείο δείχνει την πρώτη εγγραφή του αρχείου. Κάθε φορά που διαβάζουμε μια εγγραφή, ο νοητός αυτός δείκτης δείχνει στην αμέσως επόμενη εγγραφή που πρόκειται να διαβαστεί. Η διαδικασία αυτή συνεχίζεται έως ότου φτάσουμε στην τελευταία εγγραφή, οπότε και ο δείκτης δείχνει σε ένα σημάδι που δηλώνει το τέλος του αρχείου, το οποίο και ονομάζουμε **EOF** (End Of File). Η εντολή **Rewind** τοποθετεί το νοητό αυτό δείκτη στην πρώτη εγγραφή του αρχείου.

8.2.4 Η εντολή Backspace

Σύνταξη:

Backspace ([**Unit** =] ακέραιος θετικός αριθμός)

Μετατοπίζει τον νοητό δείκτη που αναφέραμε παραπάνω, στην τελευταία εγγραφή που διαβάσαμε.

Έστω ότι διαβάσαμε τη ν-οστή εγγραφή ενός αρχείου. Συνεπώς, ο νοητός δείκτης πηγαίνει στην επόμενη εγγραφή του αρχείου, δηλαδή στην ν+1 εγγραφή. Η εντολή **Backspace** μεταφέρει το νοητό δείκτη μια θέση πίσω, δηλαδή στη ν-οστή εγγραφή και συνεπώς ξαναδιαβάζουμε τη ν-οστή εγγραφή.

8.2.5 Η εντολή Endfile

Σύνταξη:

Endfile ([**Unit** =] ακέραιος θετικός αριθμός)

Τοποθετεί στο τέλος ενός αρχείου σειριακής προσπέλασης μια εγγραφή που δηλώνει το τέλος του αρχείου **EOF** (End Of File).

Επίσης η εγγραφή **EOF** (End Of File) που δηλώνει το τέλος του αρχείου τοποθετείται στο τέλος του αρχείου και με την εντολή **Close**.

8.2.6 Η εντολή Read

Σύνταξη:

Read ([**Unit** =] ακέραιος θετικός αριθμός, f)

όπου f είναι ένας ακέραιος αριθμός που αντιστοιχεί στην ετικέτα μιας εντολής προδιαγραφών **Format** (§3.6.3)

Αυτό που αλλάζει στην παραπάνω εντολή ανάγνωσης σε σχέση με την εντολή **Read** που παρουσιάσαμε στο κεφάλαιο 3, είναι:

η λογική συσκευή από την οποία διαβάζουμε δεδομένα και η οποία αντιστοιχεί σε αρχείο, σε αντιδιαστολή με τη γνωστή **Read**, όπου είχαμε το νούμερο 5 ή το σύμβολο * που σήμαινε ότι θα διαβάσουμε τα δεδομένα μας από το πληκτρολόγιο.

8.2.7 Η εντολή Write

Σύνταξη:

Write ([**Unit** =] ακέραιος θετικός αριθμός, f)

όπου f είναι ένας ακέραιος αριθμός που αντιστοιχεί στην ετικέτα μιας εντολής προδιαγραφών **Format** (§3.6.3)

Αυτό που αλλάζει στην παραπάνω εντολή ανάγνωσης σε σχέση με την εντολή **Write** που παρουσιάσαμε στο κεφάλαιο 3, είναι:

η λογική συσκευή στην οποία γράφουμε τα δεδομένα και η οποία αντιστοιχεί σε αρχείο, σε αντιδιαστολή με τη γνωστή **Write**, όπου είχαμε το νούμερο 0 ή το νούμερο 6 ή το σύμβολο * που σήμαινε ότι θα εμφανίσουμε τα δεδομένα μας στη οθόνη.

8.3 Παραδείγματα

Παράδειγμα 1: Να γραφεί πρόγραμμα που θα διαβάζει από το αρχείο input1.dat τα στοιχεία του διανύσματος Α ακεραίων αριθμών διάστασης N, από το αρχείο input2.dat τα στοιχεία του διανύσματος Β ακεραίων αριθμών διάστασης N και θα υπολογίζει και θα εκτυπώνει στο αρχείο output.dat το άθροισμά τους.

```
Program Sum_Vector

Implicit None
Integer n
Parameter (n = 8)
Integer A(n), B(n), C(n), i

Open (8, File = 'input1.dat')
Open (9, File = 'input2.dat')
Open (10, File = 'output.dat')

Do i = 1, n
   Read(8,*) A(i)
End Do

Do i = 1, n
   Read(9,*) B(i)
End Do

Do i = 1, n
   C(i)=A(i)+B(i)
End Do

Write(10,*) 'The elements of vector C'
Do i = 1, n
   Write(10,*) C(i)
End Do

Close(8)
Close(9)
Close(10)

End
```

Παρατήρηση: Τα δεδομένα στα αρχεία input1.dat και input2.dat θα πρέπει να δίνονται το ένα στοιχείο κάτω από το άλλο.

Παράδειγμα 2: Να γραφεί πρόγραμμα που θα διαβάζει από το αρχείο input.dat τα στοιχεία ενός πινάκα Α ακεραίων αριθμών διάστασης NxN και θα υπολογίζει (με τη βοήθεια συνάρτησης) και θα εκτυπώνει στο αρχείο output.dat το άθροισμα των διαγωνίων στοιχείων του.

```
Program fun2

Implicit None
Integer n
Parameter (n=3)
Integer A(n,n), i, j, diagon

Open (8, File = 'input1.dat')
Open (9, File = 'output.dat')

Do i = 1, n
   Do j = 1, n
      Read(8,*) A(i,j)
   End Do
End Do

Write(9,*)'To athroisma tvn diagwniwn stoixeien einai:'
Write(9,*) diagon(A,n)

Close(8)
Close(9)

End


Integer Function diagon(x,dim)

Implicit None
Integer dim, i
Integer x(dim,dim)

diagon=0
Do i=1,dim
   diagon=diagon+x(i,i)
End Do

End
```

Παρατήρηση: Τα δεδομένα στο αρχείο input1.dat θα πρέπει να δίνονται κατά γραμμές, το ένα στοιχείο κάτω από το άλλο, αφού για το διάβασμα των στοιχείων του πίνακα χρησιμοποιούμε μια διπλή επανάληψη.

Παράδειγμα 3: Να γραφεί πρόγραμμα που θα διαβάζει από το αρχείο input1.dat τα στοιχεία του πίνακα Α ακεραίων αριθμών διάστασης NxM, από το αρχείο input2.dat τα στοιχεία του πίνακα Β ακεραίων αριθμών διάστασης NxM και θα υπολογίζει και θα εκτυπώνει στο αρχείο output.dat το άθροισμά τους.

```
Program Sum_Array

Implicit None
Integer n, m
Parameter (n = 4, m = 3)
Integer A(n,m), B(n,m), C(n,m), i, j

Open (8, File = 'input1.dat')
Open (9, File = 'input2.dat')
Open (10, File = 'output.dat')

Do i = 1, n
   Read(8,*) (A(i,j), j = 1, m)
End Do

Do i = 1, n
   Read(9,*) (B(i,j), j = 1, m)
End Do

Do i = 1, n
   Do j = 1, m
      C(i,j)=A(i,j)+B(i,j)
   End Do
End Do

Write(10,*) 'The elements of array C'
Do i = 1, n
   Write(10,*) (C(i,j), j = 1, m)
End Do

Close(8)
Close(9)
Close(10)

End
```

Παρατήρηση: Τα δεδομένα στα αρχεία input1.dat και input2.dat θα πρέπει να δίνονται κατά γραμμές, αφού για το διάβασμα των στοιχείων των πινάκων χρησιμοποιούμε μια υπονοούμενη επανάληψη μέσα σε μια επανάληψη.

Βιβλιογραφία

- Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μάλαμας, Ι. Μανωλόπουλος, Π. Πολίτης. «*Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*», Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων, Παιδαγωγικό Ινστιτούτο, 2000.

- Αλέξανδρος Καράκος. «*Προγραμματισμός Υπολογιστών Ι (Fortran)*», http://pericles.ee.duth.gr/Fortran/default.htm.

- Νικόλαος Καραμπετάκης. «*Εισαγωγή στη Fortran 90/95*», Εκδόσεις Ζήτη, Οκτώβριος 2002.