Τμημα Επιστημης και Μηχανικης Υλικων
Πανεπιστημιο Κρητης

Εισαγωγή στη γλώσσα προγραμματισμού Fortran 95

Σημειώσεις Διαλέξεων

Σταματης Σταματιαδης

Η στοιχειοθεσία έγινε από το συγγραφέα με τη χρήση του XeLaTeX.
Συνεισέφερε ασκήσεις ο Χρήστος Μαθιουδάκης.

Τελευταία τροποποίηση του κειμένου έγινε στις 23 Οκτωβρίου 2024.
Η πιο πρόσφατη έκδοση βρίσκεται στο
https://raw.githubusercontent.com/sstamat/mybooks/main/fortran95.pdf

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Ένας ηλεκτρονικός υπολογιστής έχει τη δυνατότητα να *προγραμματιστεί* ώστε να εκτελέσει μια συγκεκριμένη διαδικασία.

Το πρώτο στάδιο του προγραμματισμού είναι να αναλύσουμε την επιθυμητή διεργασία σε επιμέρους στοιχειώδεις έννοιες και να προσδιορίσουμε τον τρόπο και τη σειρά αλληλεπίδρασης αυτών. Το βήμα αυτό αποτελεί την ανάπτυξη της μεθόδου, του *αλγορίθμου* όπως λέγεται, που θα επιτύχει την εκτέλεση της διεργασίας. Απαραίτητη προϋπόθεση, βέβαια, είναι να έχουμε *ορίσει* σαφώς και να έχουμε *κατανοήσει* πλήρως την επιθυμητή διαδικασία.

Ο αλγόριθμος συνήθως απαιτεί *δεδομένα*, τιμές που πρέπει να προσδιοριστούν πριν την εκτέλεσή του, τα οποία πρόκειται να επεξεργαστεί ώστε να παραγάγει κάποιο αποτέλεσμα. Όμως, η μέθοδος σχεδιάζεται και αναπτύσσεται *ανεξάρτητα* από συγκεκριμένες τιμές των δεδομένων αυτών.

Κατά την ανάπτυξη του αλγορίθμου χρησιμοποιούμε *σταθερές* και *μεταβλητές* ποσότητες, δηλαδή, θέσεις στη μνήμη του υπολογιστή, για την αποθήκευση των ποσοτήτων (δεδομένων, ενδιάμεσων τιμών και αποτελεσμάτων) του προγράμματος. Υπάρχει η δυνατότητα *εκχώρησης* τιμής στις μεταβλητές, *επιλογής* του επόμενου βήματος, ανάλογα με κάποια συνθήκη, καθώς και η δυνατότητα *επανάληψης* ενός η περισσότερων βημάτων. Ανάλογα με τις δυνατότητες της γλώσσας προγραμματισμού που θα χρησιμοποιήσουμε στο επόμενο στάδιο, μπορούμε να κάνουμε χρήση δομών για την ομαδοποίηση ποσοτήτων (π.χ. πίνακας), ομαδοποίηση και παραμετροποίηση πολλών εντολών (π.χ. συνάρτηση), ή ακόμα και ορισμό νέων τύπων (π.χ. κλάση). Επιπλέον, μπορούμε να επιλέξουμε και να εφαρμόσουμε ένα από διάφορους τρόπους οργάνωσης και αλληλεπίδρασης των βασικών εννοιών του προγράμματός μας. Έτσι, μπορούμε να ακολουθήσουμε την τεχνική του δομημένου/διαδικαστικού προγραμματισμού, τον αντικειμενοστρεφή προγραμματισμό κλπ. ανάλογα με το πρόβλημα και την πολυπλοκότητά του.

Το επόμενο στάδιο του προγραμματισμού είναι να μεταγράψουμε, σε κάποια γλώσσα, τον αλγόριθμο με τις συγκεκριμένες έννοιες και αλληλεπιδράσεις, να γράψουμε δηλαδή τον κώδικα. Κατόπιν, ο ηλεκτρονικός υπολογιστής θα μεταφράσει τον κώδικά μας από τη μορφή που κατανοούμε εμείς σε μορφή που κατανοεί ο ίδιος, ώστε να μπορέσει να ακολουθήσει τα βήματα που προσδιορίζουμε στον κώδικα και να ολοκληρώσει την επιθυμητή διεργασία.

Η επιλογή της κατάλληλης γλώσσας βασίζεται στις δομές και τις έννοιες προγραμματισμού που αυτή υλοποιεί. Ανάλογα με το πεδίο στο οποίο εστιάζει, μια γλώσσα μπορεί να παρέχει ειδικές δομές και τρόπους οργάνωσης του κώδικα, κατάλληλους για συγκεκριμένες εφαρμογές.

Στις παρούσες σημειώσεις θα περιγράψουμε έννοιες της γλώσσας προγραμματισμού Fortran 95 και θα αναφερθούμε στις βασικές δομές που αυτή παρέχει. Έμφαση θα δοθεί στην τεχνική του δομημένου προγραμματισμού για την ανάπτυξη του κώδικα.

Παρακάτω θα παραθέσουμε ένα τυπικό κώδικα σε Fortran 95 και θα περιγράψουμε τη λειτουργία του. Στα επόμενα κεφάλαια θα αναφερθούμε στις εντολές και δομές της Fortran 95 που χρειάζονται για να αναπτύξουμε σχετικά πολύπλοκους κώδικες.

1.1 Παράδειγμα

Ας εξετάσουμε μία απλή εργασία που θέλουμε να εκτελεστεί από ένα ηλεκτρονικό υπολογιστή: να μας ζητά έναν ακέραιο αριθμό και να τυπώνει στην οθόνη το διπλάσιό του. Η διαδικασία που ακολουθούμε, ο *αλγόριθμος*, είναι ο εξής:

1. Ανάγνωση αριθμού από το πληκτρολόγιο — [εισαγωγή του στη μνήμη].
2. [ανάκληση του αριθμού από τη μνήμη] — υπολογισμός του διπλάσιου — [εισαγωγή του αποτελέσματος στη μνήμη].
3. [ανάκληση του αποτελέσματος από τη μνήμη] — Εκτύπωση στην οθόνη.

Η ενέργειες που περιλαμβάνονται σε αγκύλες μπορεί να μη φαίνονται αναγκαίες σε πρώτη ανάγνωση. Είναι όμως, καθώς ο υπολογιστής κρατά στη μνήμη του (RAM) οποιαδήποτε πληροφορία, σε μεταβλητές ή σταθερές ποσότητες.

Πολλές γλώσσες προγραμματισμού, ανάμεσά τους και η Fortran, χρειάζονται ένα επιπλέον, προκαταρκτικό, βήμα αυτής της διαδικασίας. Προτού μεταφέρουμε τον αλγόριθμο πρέπει να κάνουμε το:

0. Δήλωση μεταβλητών (δηλαδή, *ρητή* δέσμευση μνήμης).

Ας δούμε καταρχήν ένα πλήρες πρόγραμμα Fortran 95 που εκτελεί την παραπάνω εργασία ακολουθώντας τα βήματα που περιγράψαμε. Με αυτό έχουμε την ευκαιρία να δούμε βασικά στοιχεία της δομής του κώδικα:

```
PROGRAM diplasio

   IMPLICIT NONE

   ! βήμα 0
   INTEGER :: a, b

   ! βήμα 1
   PRINT *, "Dose akeraio"
   READ *, a

   ! βήμα 2
   b = 2 * a

   ! βήμα 3
   PRINT *, "To diplasio einai"
   PRINT *, b

END PROGRAM diplasio
```

Ας το αναλύσουμε:

1.2 Δομή Προγράμματος

Οι εντολές του κυρίως προγράμματός μας (καλό είναι να) περιλαμβάνονται μεταξύ των **PROGRAM**...**END PROGRAM**:

```
PROGRAM diplasio

IMPLICIT NONE
...
...
...
END PROGRAM diplasio
```

Με την πρώτη εντολή το πρόγραμμά μας έχει ονομαστεί «diplasio». Καλό είναι να αποδίδουμε ονόματα στα προγράμματά μας που υποδηλώνουν με κάποιον τρόπο τι κάνουν αυτά. Το όνομα πρέπει να είναι μονολεκτικό.

Όλες οι εντολές/γραμμές μπορούν να ξεκινούν από οποιαδήποτε στήλη· πρέπει να έχουν ολοκληρωθεί στη στήλη 132 καθώς οι χαρακτήρες που υπάρχουν μετά από αυτή αγνοούνται.

Οι πρώτες γραμμές μετά το **PROGRAM** ... συγκεντρώνουν τις δηλώσεις όλων των ποσοτήτων. Η πρώτη γραμμή από αυτές (καλό είναι να) είναι η **IMPLICIT NONE**. Αφού παρατεθούν οι δηλώσεις, ακολουθούν οι υπόλοιπες εντολές, με τη σειρά που θέλουμε να εκτελεστούν.

Η τελική εντολή είναι το **END PROGRAM** ....

Σε ένα αρχείο με κώδικα σε Fortran 95 επιτρέπονται περισσότερα του ενός κενά μεταξύ λέξεων και κενές γραμμές μεταξύ εντολών.

1.3 Σχόλια

Ο compiler αγνοεί τους χαρακτήρες που υπάρχουν μεταξύ του χαρακτήρα '!' και μέχρι το τέλος της γραμμής που εμφανίζεται αυτός. Οι χαρακτήρες αυτοί αποτελούν τα *σχόλια* και πρέπει να είναι μόνο λατινικοί (σχεδόν πάντα από το σύνολο χαρακτήρων ASCII).

Γραμμές ή τμήματα γραμμών που αρχίζουν με '!' μπορούν να περιλάβουν επεξηγηματικά μηνύματα από τον προγραμματιστή, για τη δική του διευκόλυνση. Η ύπαρξη *επαρκών* και *σωστών* σχολίων σε ένα κώδικα βοηθά σημαντικά στην κατανόησή του από άλλους ή και εμάς τους ίδιους, όταν, μετά από καιρό, θα έχουμε ξεχάσει τι και πώς ακριβώς το κάνει το συγκεκριμένο πρόγραμμα. Προσέξτε όμως ότι η ύπαρξη σχολίων δυσνόητων ή υπερβολικά σύντομων, που δεν αντιστοιχούν στην τρέχουσα μορφή του κώδικα ή που δεν διευκρινίζουν τι ακριβώς γίνεται, είναι χειρότερη από την πλήρη έλλειψή τους.

1.4 Δήλωση Μεταβλητών

Οι *μεταβλητές* (variables) είναι θέσεις στη μνήμη που χρησιμοποιούνται για την αποθήκευση των ποσοτήτων (δεδομένων, αποτελεσμάτων) του προγράμματος. Προτού χρησιμοποιηθούν πρέπει να *δηλωθούν*, δηλαδή να ενημερωθεί ο compiler για το όνομά τους και τον τύπο τους. Προτού χρησιμοποιηθούν σε κάποια έκφραση πρέπει *να πάρουν τιμή*.

Η εντολή

```
INTEGER :: a, b
```

αποτελεί τη *δήλωση των μεταβλητών* του προγράμματός μας. Παρατηρήστε ότι στον αλγόριθμό μας έχουμε δύο εισαγωγές ακεραίων στη μνήμη, οπότε θέλουμε δύο κατάλληλες μεταβλητές. Η συγκεκριμένη εντολή ζητά από τον compiler να δεσμεύσει χώρο στη μνήμη για δύο ακέραιους αριθμούς (**INTEGER**) με ονόματα `a` και `b`.

1.5 Ανάγνωση δεδομένων

Η εντολή

```
READ *, a
```

«διαβάζει» από το πληκτρολόγιο έναν ακέραιο αριθμό και τον τοποθετεί στη μεταβλητή `a`. Το σύμβολο '*' υποδεικνύει στο μεταγλωττιστή ότι το πλήθος, τον τύπο

και τη μορφή των δεδομένων που θα δεχθεί, πρέπει να τα προσδιορίσει αυτόματα από τις μεταβλητές που παρατίθενται στην εντολή **READ**. Θα δούμε αργότερα πώς μπορούμε να προσδιορίσουμε εμείς ρητά τις συγκεκριμένες πληροφορίες.

Η συγκεκριμένη εντολή αποτελεί τον ένα από τρεις τρόπους για απόδοση τιμής σε μεταβλητή. Οι άλλοι είναι η *εντολή εκχώρησης* (2.2) και η απόδοση τιμής κατά τη δήλωση (αρχικοποίηση).

1.6 Υπολογισμοί και Εκχώρηση

Η εντολή

```
b = 2*a
```

εκτελείται ως εξής: *Καταρχάς*, υπολογίζεται το *δεξί* μέλος: ανακαλείται από τη μνήμη ο αριθμός που είναι αποθηκευμένος στη μεταβλητή a και εκτελείται ο πολλαπλασιασμός με το 2. Ο πολλαπλασιασμός υποδηλώνεται με το '*'. *Κατόπιν*, το αποτέλεσμα της πράξης αποθηκεύεται στη μεταβλητή του αριστερού μέλους.

1.7 Εκτύπωση δεδομένων

Οι εντολές

```
PRINT *, "To diplasio einai"
PRINT *, b
```

προκαλούν διαδοχική εκτύπωση στην οθόνη δύο πληροφοριών: συγκεκριμένου κειμένου (σειρά χαρακτήρων εντός εισαγωγικών) και *της τιμής* που είχε αποθηκευτεί στη μεταβλητή b (και η οποία ανακαλείται από τη μνήμη).

Στο επόμενο κεφάλαιο ακολουθούν πιο αναλυτικές περιγραφές όσων αναφέραμε στο παράδειγμα.

1.8 Ασκήσεις

1. Γράψτε, μεταγλωττίστε και εκτελέστε τον κώδικα του παραδείγματος.[1]

[1] Το πώς θα τα κάνετε αυτά εξαρτάται από το λειτουργικό σύστημα και τον compiler που χρησιμοποιείτε και γι' αυτό δε δίνονται εδώ λεπτομέρειες.

Κεφάλαιο 2

Τύποι και Τελεστές

2.1 Ενσωματωμένοι τύποι

Οι ενσωματωμένοι τύποι των ποσοτήτων στη Fortran είναι οι

- **INTEGER**, για ακέραιες ποσότητες,
- **REAL** και **DOUBLE PRECISION**, για πραγματικές ποσότητες απλής και διπλής ακρίβειας,
- **COMPLEX**, για μιγαδικές ποσότητες,
- **LOGICAL**, για λογικές ποσότητες,
- **CHARACTER**, για ποσότητες χαρακτήρα.

2.1.1 Δήλωση

Η δήλωση μιας ακέραιας μεταβλητής με όνομα π.χ. abc γίνεται με την ακόλουθη εντολή:

```
INTEGER :: abc
```

Η δήλωση δύο πραγματικών μεταβλητών απλής ακρίβειας με ονόματα π.χ. value1, value2, γίνεται με τις ακόλουθες εντολές:

```
REAL :: value1
REAL :: value2
```

ή, ισοδύναμα, με την

```
REAL :: value1, value2
```

Οι πραγματικές μεταβλητές απλής ακρίβειας έχουν, στους περισσότερους compilers, ακρίβεια 6 σημαντικών ψηφίων[1].

Σε επιστημονικούς κώδικες, η ακρίβεια των μεταβλητών τύπου **REAL** συχνά δεν είναι ικανοποιητική. Γι' αυτό υπάρχει ενσωματωμένος ένας πραγματικός τύπος με μεγαλύτερη ακρίβεια. Ο τρόπος που ορίζεται αυτός στη Fortran 95 είναι σχετικά πολύπλοκος (παρουσιάζεται στην §2.7), διευκολύνει όμως αρκετά την ανάπτυξη μεγάλων κωδίκων. Εδώ θα παρουσιάσουμε τον τρόπο που «κληρονομήθηκε» από τη Fortran 77.

Η δήλωση πραγματικής μεταβλητής διπλής ακρίβειας με όνομα π.χ. ab, γίνεται με την ακόλουθη εντολή:

```
DOUBLE PRECISION :: ab
```

Οι πραγματικές μεταβλητές διπλής ακρίβειας έχουν, στους περισσότερους μεταγλωττιστές, ακρίβεια 15 σημαντικών ψηφίων.

Δήλωση μιγαδικής μεταβλητής (απλής ακρίβειας) γίνεται ως εξής:

```
COMPLEX :: z
```

Περισσότερα για μιγαδικούς θα αναπτύξουμε παρακάτω.

2.1.2 Όνομα

Τα ονόματα των μεταβλητών, συναρτήσεων, προγράμματος, και, γενικότερα, των «οντοτήτων» του κώδικα, είναι μονολεκτικά και της επιλογής του προγραμματιστή. Σχηματίζονται με τους λατινικούς χαρακτήρες a–z, τα αριθμητικά ψηφία $0-9$ και το χαρακτήρα '_'. Το μέγιστο μήκος κάθε ονόματος είναι 31 χαρακτήρες. Το όνομα δεν μπορεί να αρχίζει με αριθμητικό ψηφίο. Κεφαλαία και πεζά γράμματα είναι ίδια. Καλό είναι να μην αρχίζει ούτε από '_' καθώς ονόματα με αυτόν για πρώτο χαρακτήρα χρησιμοποιούνται εσωτερικά από το μεταγλωττιστή.

Είναι προφανές ότι δεν μπορεί να χρησιμοποιηθεί το ίδιο όνομα για την αναπαράσταση διαφορετικών οντοτήτων *που συνυπάρχουν*.

Παράδειγμα
Μη αποδεκτά ονόματα:

```
ena lathos onoma, άλφα, 1234qwer, .onoma.
```

Αποδεκτά ονόματα:

```
timi, value12, ena_onoma_me_megalo_mikos, sqrt
```

[1] Σε ένα πραγματικό αριθμό γραμμένο σε δεκαδική μορφή, σημαντικά χαρακτηρίζονται τα ψηφία από το πρώτο από αριστερά μη μηδενικό ψηφίο, έως και το τελευταίο ψηφίο.

2.1.3 Εκχώρηση Τιμής

Μια μεταβλητή παίρνει τιμή

- «διαβάζοντάς» τη από το πληκτρολόγιο, από αρχείο κλπ., με την εντολή **READ**. Π.χ. για είσοδο από το πληκτρολόγιο ακέραιου αριθμού και αποθήκευση στη μεταβλητή a έχουμε

  ```
  INTEGER :: a
  .....
  READ *, a
  ```

  ή, ισοδύναμα, αλλά και πιο γενικά,

  ```
  INTEGER :: a
  .....
  READ (*,*) a
  ```

- με εντολή εκχώρησης (§2.2), της μορφής

  μεταβλητή = [έκφραση με σταθερές και μεταβλητές]

  Π.χ.

  ```
  REAL :: a, b
  a = 5.2
  b = 3.0 + 2 * a
  ```

- κατά τη δήλωσή της (αρχική τιμή). Π.χ.

  ```
  INTEGER :: k = 3
  ```

  Η παραπάνω εντολή δηλώνει ότι η μεταβλητή με όνομα k είναι ακέραια και της δίνει την αρχική τιμή 3[2].

Μπορούμε να αλλάξουμε την τιμή μιας μεταβλητής κατά τη διάρκεια εκτέλεσης του προγράμματος.

2.2 Εντολή εκχώρησης

Είναι εντολή της μορφής

μεταβλητή = [έκφραση με σταθερές και μεταβλητές]

[2] Η απόδοση αρχικής τιμής κατά τη δήλωση μετατρέπει τη μεταβλητή σε *στατική*, §8.8.

Σε αυτή την εντολή εκτελούνται καταρχάς όλες οι πράξεις, κλήσεις συναρτήσεων κλπ. που πιθανόν εμφανίζονται στο δεξί μέλος και, κατόπιν, το αποτέλεσμα *μετατρέπεται*, αν χρειάζεται, στον τύπο της μεταβλητής του αριστερού μέλους και η τιμή του εκχωρείται σε αυτή.

Η εντολή εκχώρησης μπορεί να χρησιμοποιηθεί και όταν το αριστερό μέλος είναι διάνυσμα ή πίνακας, όπως θα δούμε στο §5.6.

2.3 Αριθμητικές Σταθερές

2.3.1 Ακέραιες

Μία σειρά αριθμητικών ψηφίων χωρίς κενά ή άλλα σύμβολα, αποτελεί μία ακέραια σταθερά. Ο πρώτος χαρακτήρας της μπορεί να είναι το πρόσημο '+' ή '−'. Π.χ., τρεις ακέραιες σταθερές είναι οι παρακάτω

$$3,\ -12,\ 123456$$

2.3.2 Πραγματικές

Μία σειρά αριθμητικών ψηφίων χωρίς κενά, που περιλαμβάνει τελεία (στη θέση της υποδιαστολής) συμβολίζει πραγματική σταθερά *απλής ακρίβειας* (συνήθως 6 σημαντικών ψηφίων). Πριν ή μετά την υποδιαστολή μπορεί να μην υπάρχουν ψηφία. Ο χαρακτήρας E, αν υπάρχει, ακολουθείται από τον ακέραιο εκθέτη του 10 με τη δύναμη του οποίου πολλαπλασιάζεται ο αμέσως προηγούμενος τού E αριθμός:

$$2.034, \quad 0.23, \quad .44, \quad 23., \quad 2\mathrm{e}-4 \quad (\equiv 0.0002), \quad 2.3\mathrm{E}2 \quad (\equiv 230.0)\ .$$

Μία σειρά αριθμητικών ψηφίων χωρίς κενά, που περιλαμβάνει τελεία (στη θέση της υποδιαστολής) *και ακολουθείται από το χαρακτήρα D και τον ακέραιο εκθέτη του* 10 *με τη δύναμη του οποίου πολλαπλασιάζεται* συμβολίζει πραγματική σταθερά *διπλής ακρίβειας* (συνήθως 15 σημαντικών ψηφίων). Π.χ.,

$$2\mathrm{d}-4 \quad (\equiv 0.0002\mathrm{d}0), \qquad 2.3\mathrm{D}2 \quad (\equiv 230.0\mathrm{d}0)\ .$$

Προσέξτε ότι ο ίδιος πραγματικός αριθμός μπορεί να γραφεί στη Fortran ως πραγματικός είτε απλής είτε διπλής ακρίβειας[3]. Οι δύο μορφές *είναι διαφορετικές* παρόλο που αντιστοιχούν στον ίδιο αριθμό. Στην περίπτωση που εμφανίζονται στην ίδια έκφραση, π.χ. σε μια πρόσθεση, ο μεταγλωττιστής μετατρέπει τη σταθερά απλής ακρίβειας σε διπλής προτού προχωρήσει.

Ο συμβολισμός ενός πραγματικού αριθμού στη μορφή που περιλαμβάνει το E ή το D είναι ο λεγόμενος «επιστημονικός». Παρατηρήστε ότι μόνο με την επιστημονική μορφή μπορούμε να προσδιορίσουμε ότι μια πραγματική σταθερά είναι διπλής

[3] είτε, βέβαια, ως μιγαδικός ή και ως ακέραιος, αν τυχαίνει να είναι ακέραιος στα μαθηματικά.

ακρίβειας. Δηλαδή, το 3.2 της αριθμητικής, πρέπει να γραφεί στη Fortran ως `3.2D0` για να αποθηκευθεί ως διπλής ακρίβειας. Επίσης, το πόσα δεκαδικά ψηφία παραθέτουμε σε μία πραγματική σταθερά *δεν καθορίζει τον τύπο της*. Αν είναι απλής ακρίβειας θα *αγνοηθούν* τα ψηφία πέρα από το έκτο ή έβδομο σημαντικό. Αντίστοιχα, σε διπλή ακρίβεια τα πρώτα 15 ή 16 σημαντικά ψηφία κρατούνται, και θεωρούνται 0 όσα δεκαδικά μέχρι το 15ο ή 16ο δεν προσδιορίζονται. Έτσι, η μαθηματική σταθερά $\pi = 3.14159265358979323846\ldots$ μπορεί να γραφεί στη Fortran ως `3.141593` (τα επιπλέον δεκαδικά ψηφία αγνοούνται, ακόμα και αν τα γράψουμε) ή ως `3.14159265358979D0`.

2.3.3 Μιγαδικές

Η *μιγαδική σταθερά* απλής ακρίβειας συμβολίζεται στη Fortran με ζεύγος πραγματικών αριθμών εντός παρενθέσεων, με ',' μεταξύ τους. Ο πρώτος αριθμός είναι το πραγματικό μέρος ενώ ο δεύτερος είναι το φανταστικό. Π.χ., ο μιγαδικός αριθμός $3.1 + 2\mathrm{i}$ των μαθηματικών γράφεται ως

$$(3.1, 2.0),$$

ενώ ο i γράφεται ως

$$(0.0, 1.0).$$

2.4 Σταθερές Ποσότητες

Μια ποσότητα, οποιουδήποτε τύπου, που επιθυμούμε να πάρει τιμή που *να μην μπορεί να αλλάξει* κατά τη διάρκεια εκτέλεσης του προγράμματος, δηλώνεται με τη χρήση της λέξης **PARAMETER**. Είναι απαραίτητο να της δώσουμε αρχική (και μόνιμη) τιμή κατά τον ορισμό της (δεν μπορεί, επομένως, να διαβαστεί από το πληκτρολόγιο, αρχείο, κλπ., ή να πάρει τιμή με εντολή εκχώρησης). Π.χ. πραγματική ποσότητα διπλής ακρίβειας με σταθερή, αμετάβλητη τιμή δηλώνεται ως εξής

```
DOUBLE PRECISION, PARAMETER :: pi = 3.14159265358d0
```

Καλό είναι να χρησιμοποιούνται συμβολικές σταθερές για να αποφεύγεται η χρήση «μαγικών αριθμών» στον κώδικα. Αν μια ποσότητα που είναι σταθερή (π.χ. πλήθος στοιχείων σε διάνυσμα, φυσικές ή μαθηματικές σταθερές) χρησιμοποιείται με την αριθμητική της τιμή και όχι με συμβολικό όνομα, καθίσταται ιδιαίτερα δύσκολη η αλλαγή της καθώς πρέπει να αναγνωριστεί και να τροποποιηθεί σε όλα τα σημεία του κώδικα που εμφανίζεται.

2.5 Αριθμητικοί Τελεστές

Οι τελεστές $+, -, *, /$ μεταξύ πραγματικών αριθμών εκτελούν τις πράξεις της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης αντίστοιχα.

Οι τελεστές $+, -, *$ μεταξύ ακεραίων αριθμών εκτελούν τις πράξεις της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού αντίστοιχα. Ο τελεστής / μεταξύ ακεραίων αριθμών εκτελεί τη διαίρεση *και αποκόπτει το δεκαδικό μέρος του αποτελέσματος*, επιστρέφει, δηλαδή, το πηλίκο της διαίρεσης. Η ενσωματωμένη συνάρτηση **MOD**() επιστρέφει το υπόλοιπο της διαίρεσης ως εξής

```
INTEGER :: a, b, p, y

a = 5
b = 3
p = a / b      ! Πηλίκο: p ← 1
y = MOD(a,b)   ! Υπόλοιπο: y ← 2
```

Ο τελεστής $**$ εκτελεί την ύψωση σε δύναμη. Π.χ. το x^3 των μαθηματικών γράφεται στη Fortran ως `x**3`.

Οι τελεστές $+, -$ έχουν ίδια προτεραιότητα, η οποία είναι χαμηλότερη από την προτεραιότητα των $*, /$ (που έχουν την ίδια), όπως και στα μαθηματικά. Σε μια έκφραση που εμφανίζονται διαδοχικά τελεστές ίδιας προτεραιότητας, εκτελείται πρώτα αυτός που είναι στα αριστερά. Ο τελεστής $**$ έχει υψηλότερη προτεραιότητα από όλους. Π.χ.

```
INTEGER :: i, j, k

i = 2 + 3              ! i ← 5
j = 2 * 3              ! j ← 6
i = i + j              ! i ← 11
i = 2 * i + j          ! i ← 28
i = 17 / j             ! i ← 2
k = i * j / 3 + 2**3   ! k ← 12
```

Αλλαγή προτεραιότητας γίνεται με τη χρήση παρενθέσεων. Π.χ.

```
INTEGER :: i, j, k

i = 2
j = 6
k = 2 * i + j       ! k ← 10
k = 2 * (i + j)     ! k ← 16
k = j * i / 3       ! k ← 4
k = j * (i / 3)     ! k ← 0
```

Παρατήρηση: Όταν οι τελεστές $+, -, *, /$ εμφανίζονται μεταξύ αριθμών διαφορετικού τύπου (π.χ. ενός ακεραίου και ενός πραγματικού), γίνεται αυτόματη μετατροπή ενός από τους αριθμούς στον τύπο του άλλου ώστε μην έχουμε απώλεια ακρίβειας. Έτσι, ο ακέραιος μετατρέπεται στον ισοδύναμο πραγματικό, ο πραγματικός στον ισοδύναμο μιγαδικό, κλπ. Κατόπιν, εκτελούνται οι πράξεις. Π.χ.

```
INTEGER :: i, j
REAL :: x, y

i = 8.3        ! i ← 8
j = 2
y = 3.0

x = 4.0 / 3        ! ⇒ x ← 4.0/3.0 ⇒ x ← 1.33333

x = j + y          ! ⇒ x ← 2 + 3.0 ⇒ x ← 2.0 + 3.0 ⇒ x ← 5.0

i = 4 / 3.0        ! ⇒ i ← 4.0/3.0 ⇒ i ← 1.33333 ⇒ i ← 1

x = 4 / 3          ! ⇒ x ← 1 ⇒ x ← 1.0

x = 2.0 * 4 / 3    ! ⇒ x ← 8.0/3 ⇒ x ← 8.0/3.0
                   ! ⇒ x ← 2.66667
x = 4 / 3 * 2.0    ! ⇒ x ← 1 * 2.0 ⇒ x ← 2.0
```

Παρατήρηση: Η πεπερασμένη αναπαράσταση των πραγματικών αριθμών οδηγεί σε σφάλματα στρογγύλευσης στις πράξεις και ορισμένες μαθηματικές ιδιότητές τους (π.χ. η αντιμεταθετική και η προσεταιριστική της πρόσθεσης) δεν ισχύουν. Τι αναμένετε να τυπωθεί με τις επόμενες εντολές; Δοκιμάστε τις.

```
PRINT *, 0.1d0-0.3d0+0.2d0
PRINT *, 0.1d0+0.2d0-0.3d0
```

2.6 Άλλοι ενσωματωμένοι τύποι

Παρακάτω θα παρουσιάσουμε τους υπόλοιπους ενσωματωμένους τύπους που παρέχει η Fortran, πέρα από τους **INTEGER**, **REAL**, **DOUBLE PRECISION**.

2.6.1 Τύπος χαρακτήρα

Για την αναπαράσταση ποσοτήτων που οι δυνατές τιμές τους είναι χαρακτήρες ή σειρές χαρακτήρων, η Fortran παρέχει τον τύπο **CHARACTER**. Δήλωση μιας μεταβλητής χαρακτήρα με όνομα π.χ. ch γίνεται με την ακόλουθη εντολή:

```
CHARACTER :: ch
```

Οι τιμές που μπορεί να πάρει είναι απλοί χαρακτήρες εντός απλών (') ή διπλών (") εισαγωγικών:

```
ch = 'a'
ch = 'D'
ch = "R"
```

Για την περιγραφή σειράς χαρακτήρων, η δήλωση περιλαμβάνει το μέγιστο δυνατό μήκος των σειρών που πρόκειται να αναπαραστήσουμε. Για παράδειγμα, η σειρά `"This is a message"` αποτελείται από 17 χαρακτήρες και, επομένως, μπορεί να ανατεθεί σε μεταβλητή τύπου **CHARACTER** με τουλάχιστον αυτό το μήκος. Η κατάλληλη δήλωση είναι

```
CHARACTER (17) :: ch

ch = "This is a message"
```

2.6.2 Λογικός τύπος

Μεταβλητή λογικού τύπου (**LOGICAL**) είναι κατάλληλη για την αναπαράσταση ποσοτήτων που μπορούν να πάρουν δύο τιμές (π.χ. ναι/όχι, αληθές/ψευδές,...). Η δήλωση τέτοιας μεταβλητής, με όνομα π.χ. a, γίνεται ως εξής:

```
LOGICAL :: a
```

Οι τιμές που μπορεί να πάρει είναι .**TRUE**. ή .**FALSE**.. Η εκχώρηση π.χ. της σταθερής τιμής .**TRUE**. στην a γίνεται ως εξής

```
a = .TRUE.
```

2.6.3 Μιγαδικός Τύπος

Η Fortran υποστηρίζει με τον ενσωματωμένο τύπο **COMPLEX** την περιγραφή μιγαδικών μεταβλητών και σταθερών. Η δήλωση είναι ως εξής

```
COMPLEX :: z
```

Εσωτερικά, για τον compiler, αποτελείται από δύο πραγματικούς αριθμούς, *απλής ακρίβειας*, που αντιστοιχούν στο πραγματικό και φανταστικό μέρος. Στη Fortran 77 δεν υποστηρίζεται μιγαδικός τύπος διπλής ακρίβειας. Αυτή η δυνατότητα υπάρχει στην Fortran 95 αλλά πρέπει να χρησιμοποιηθεί ο δικός της τρόπος (§2.7) για τον προσδιορισμό του. Εναλλακτικά, είναι πιθανόν ο μεταγλωττιστής να υποστηρίζει μία επέκταση της Fortran, τον τύπο **DOUBLE COMPLEX**.

Δημιουργία μιγαδικής ποσότητας

Ένας αριθμός μιγαδικού τύπου στη Fortran μπορεί να πάρει τιμή

- με εκχώρηση *μιγαδικής σταθεράς*, δηλαδή, δύο πραγματικών αριθμών σε ένα ζεύγος παρενθέσεων, με (,) μεταξύ τους. Ο πρώτος αριθμός είναι το πραγματικό μέρος ενώ ο δεύτερος είναι το φανταστικό:

```
COMPLEX :: z

z = (1.2, 4.56)      ! z ← 1.2 + i4.56
```

Αν ανατεθεί πραγματική ή ακέραια ποσότητα θα γίνει μετατροπή συμπληρώνοντας με το 0 το φανταστικό μέρος·

```
COMPLEX :: z

z = 2        ! z ← 2.0 + i0.0
z = 2.3      ! z ← 2.3 + i0.0
```

- Με εκχώρηση έκφρασης που έχει συνολικά μιγαδική τιμή (ή πραγματική ή ακέραια, για τις οποίες θα προηγηθεί μετατροπή):

```
COMPLEX :: z, z1, z2

z1 = 3                ! z1 ← 3.0 + i0.0
z2 = (4.6, -4.2)      ! z2 ← 4.6 − i4.2
z = z1 + z2 + 5.1     ! z  ← 12.7 − i4.2
```

  Στην επόμενη παράγραφο θα δούμε περισσότερα για τις εκφράσεις.

- Με ανάγνωση μιγαδικής σταθεράς από το πληκτρολόγιο ή από αρχείο. Π.χ. ο κώδικας

```
COMPLEX :: z

READ *, z
```

  αναμένει να του δώσουμε

  (1.6, -3.8)

  (περιλαμβάνονται το κόμμα και οι παρενθέσεις).

- Με τη χρήση της ενσωματωμένης συνάρτησης **CMPLX**(). Αυτή δέχεται δύο πραγματικές ποσότητες και δημιουργεί μιγαδικό αριθμό με πραγματικό μέρος την πρώτη ποσότητα και φανταστικό τη δεύτερη

```
COMPLEX :: z
REAL :: x, y

x = 3.4
y = -9.1

z = CMPLX(x,y)       ! z ← 3.4 − i9.1
```

Προσέξτε ότι η έκφραση

```
z = (x,y)
```

που ίσως να περίμενε κανείς ότι θα έκανε σωστά τη δημιουργία, είναι λάθος.

Εκφράσεις με μιγαδικές ποσότητες

Οι αριθμητικοί τελεστές '+','-','*','/','**' εκτελούν τις αναμενόμενες πράξεις από τα μαθηματικά· ο τελευταίος συμβολίζει την ύψωση σε δύναμη. Υπενθυμίζουμε ότι αν

$$z1 = \alpha + \mathrm{i}\beta, \qquad z2 = \gamma + \mathrm{i}\delta,$$

τότε

$$z1 \cdot z2 = (\alpha\gamma - \beta\delta) + \mathrm{i}(\alpha\delta + \beta\gamma)\ .$$

Αυτήν ακριβώς την πράξη εκτελεί ο τελεστής '*' μεταξύ μιγαδικών ποσοτήτων στη Fortran. Αντίστοιχα ισχύουν και για τον τελεστή '/'.

Κατ' επέκταση όσων είπαμε στις §2.2, §2.5, ένας μιγαδικός που συμμετέχει σε μία πράξη (δεξιά ή αριστερά ενός τελεστή), προκαλεί τη μετατροπή σε μιγαδικό και του άλλου μέρους της πράξης, προτού αυτή εκτελεστεί. Επίσης, εκχώρηση του μιγαδικού αποτελέσματος μιας έκφρασης προκαλεί μετατροπή της τιμής στον τύπο της μεταβλητής στην οποία ανατίθεται. Όταν η μεταβλητή στην οποία ανατίθεται είναι πραγματική, η μετατροπή συνίσταται στην αποκοπή του φανταστικού τμήματος:

```
COMPLEX :: z
REAL    :: x

z = (2.3,4.5)       ! z ← 2.3 + i4.5
x = z               ! x ← 2.3
```

Όταν η μεταβλητή στην οποία ανατίθεται είναι ακέραια, η μετατροπή που γίνεται είναι η αποκοπή του φανταστικού τμήματος και η αποκοπή του δεκαδικού τμήματος του πραγματικού μέρους:

```
COMPLEX :: z
INTEGER :: i

z = (2.3,4.5)       ! z ← 2.3 + i4.5
i = z               ! i ← 2
```

Όλες οι ενσωματωμένες συναρτήσεις που θα δούμε παρακάτω ισχύουν και για μιγαδικούς με την αναμενόμενη από τα μαθηματικά συμπεριφορά.

Εξαγωγή πραγματικού/φανταστικού μέρους –Συζυγής–Μέτρο

Το πραγματικό μέρος ενός μιγαδικού αριθμού υπολογίζεται με τη δράση της ενσωματωμένης συνάρτησης **REAL**( ) βάζοντας ως όρισμα (ποσότητα μεταξύ των παρενθέσεων) τον μιγαδικό αριθμό:

```
COMPLEX :: z
REAL :: x

z = (1.2,-8.7)        ! z ← 1.2 − i8.7
x = REAL(z)           ! x ← 1.2
```

Το φανταστικό μέρος εξάγεται με τη δράση της συνάρτησης **AIMAG**( ):

```
COMPLEX :: z
REAL :: x

z = (1.2,-8.7)        ! z ← 1.2 − i8.7
x = AIMAG(z)          ! x ← −8.7
```

Ο μιγαδικός συζυγής ενός αριθμού παράγεται με τη δράση της συνάρτησης **CONJG**( ):

```
COMPLEX :: z1, z2

z1 = (1.2,-8.7)       ! z1 ← 1.2 − i8.7
z2 = CONJG(z1)        ! z2 ← 1.2 + i8.7
```

Το (πραγματικό) μέτρο $|z|$ ενός μιγαδικού υπολογίζεται με τη συνάρτηση **ABS**( ):

```
COMPLEX :: z
REAL :: norm

z = (1.2,-8.7)        ! z ← 1.2 − i8.7
norm = ABS(z)         ! norm ← √(1.2² + 8.7²)
```

2.6.4 Ρητή μετατροπή μεταξύ αριθμητικών τύπων

Υπάρχουν περιπτώσεις που χρειάζεται να κάνουμε ρητά μετατροπή ποσότητας από ένα τύπο σε άλλον και να μη βασιζόμαστε στη μετατροπή που γίνεται κατά την εκχώρηση. Π.χ. προσέξτε τον παρακάτω κώδικα

```
DOUBLE PRECISION :: x
INTEGER :: i, j

i = 3
j = 2

x = i/j       ! x ← 1.0d0
```

Η μεταβλητή x, παρόλο που είναι πραγματικού τύπου, δεν αποκτά την επιθυμητή τιμή (1.5) καθώς της εκχωρείται το πηλίκο της διαίρεσης των δύο ακεραίων αριθμών. Η διαίρεση θα γίνει «σωστά» αν μετατρέψουμε τις ακέραιες τιμές που συμμετέχουν σε πραγματικές με κλήση της κατάλληλης συνάρτησης

```
DOUBLE PRECISION :: x
INTEGER :: i, j

i = 3
j = 2

x = DBLE(i)/DBLE(j)       ! x ← 1.5d0
```

Η ρητή μετατροπή τιμής αρκεί να γίνει στη μία από τις δύο ποσότητες· η άλλη θα μετατραπεί αυτόματα στον τύπο του έτερου μέλους. Προσέξτε ότι η μετατροπή αφορά *την τιμή* που έχει η ποσότητα στο όρισμα. Ο *τύπος* της δεν αλλάζει.

Οι συναρτήσεις ρητής μετατροπής της Fortran είναι οι εξής: **REAL**(), **DBLE**(), **INT**(), **CMPLX**(). Δέχονται μεταξύ των παρενθέσεων μία ποσότητα οποιουδήποτε αριθμητικού τύπου (**INTEGER**, **REAL**, **DOUBLE PRECISION**, **COMPLEX**) και επιστρέφουν αντίστοιχα **REAL**, **DOUBLE PRECISION**, **INTEGER**, **COMPLEX**. Η μετατροπή γίνεται σύμφωνα με όσα έχουμε αναφέρει για την εκχώρηση στους ομώνυμους τύπους.

Εναλλακτικά, αντί για τη ρητή μετατροπή μέσω των ενσωματωμένων συναρτήσεων, μπορούμε να εκμεταλλευτούμε τους αυτόματους κανόνες μετατροπής και να γράψουμε κάτι σαν

```
x = 1.0d0 * i / j
```

ή

```
x =  (i+0.0d0) / j
```

Οι τιμές της μεταβλητής x θα είναι τότε οι επιθυμητές (γιατί;).

2.7 Προσδιορισμός του τύπου στην Fortran 95

Στα προηγούμενα έχουμε παρουσιάσει όλους τους θεμελιώδεις τύπους της Fortran για την αναπαράσταση ακεραίων, πραγματικών, μιγαδικών, χαρακτήρων και λογικών ποσοτήτων. Η Fortran 95 παρέχει τουλάχιστον δύο τύπους για πραγματικές ποσότητες, αυτούς που συνήθως χαρακτηρίζουμε ως *απλής* και *διπλής* ακρίβειας. Στη Fortran 77 αυτοί είναι οι **REAL** και **DOUBLE PRECISION** αντίστοιχα. Η Fortran 95 τους διατήρησε, παρέχει όμως ένα πιο γενικό μηχανισμό για την επιλογή του επιθυμητού τύπου. Αυτόν θα παρουσιάσουμε παρακάτω.

Η επιλογή του πραγματικού τύπου στην Fortran 95 γίνεται κατά τη δήλωση ως εξής:

```
REAL (prc) :: a
```

Με την παραπάνω εντολή, το a ορίζεται ως πραγματική μεταβλητή με *είδος (kind)* prc. Η ποσότητα εντός των παρενθέσεων είναι μια *ακέραια σταθερή* ποσότητα, με όνομα της επιλογής μας. Προφανώς, προτού τη χρησιμοποιήσουμε πρέπει να την έχουμε ορίσει και να της έχουμε εκχωρήσει τιμή.

Η επιλογή του είδους του πραγματικού τύπου, η τιμή δηλαδή της ποσότητας prc στο συγκεκριμένο παράδειγμα, γίνεται με έναν από τους ακόλουθους τρόπους:

- με τη χρήση της ενσωματωμένης συνάρτησης **SELECTED_REAL_KIND**(). Ως όρισμα τής δίνουμε το πόσα ψηφία (στο δεκαδικό σύστημα) επιθυμούμε να έχουμε σωστά (όταν τη χρησιμοποιήσουμε σε δήλωση πραγματικού για τον προσδιορισμό του είδους του). Π.χ.

  ```
  INTEGER, PARAMETER :: prc = SELECTED_REAL_KIND(12)
  ```

  Στην παραπάνω δήλωση ορίσαμε μια ακέραια σταθερά με όνομα prc και της δώσαμε την κατάλληλη τιμή για να έχουμε πραγματικούς με τουλάχιστον 12 ψηφία σωστά.

- με τη χρήση της ενσωματωμένης συνάρτησης **KIND**(). Ως όρισμα τής δίνουμε έναν πραγματικό αριθμό ή πραγματική ποσότητα και μας επιστρέφει το είδος του. Επομένως η δήλωση

  ```
  INTEGER, PARAMETER :: prc = KIND(1.0D0)
  ```

  αποδίδει στη σταθερά prc το είδος των πραγματικών αριθμών διπλής ακρίβειας.

Συνοψίζοντας: επιλέγουμε πρώτα το είδος του πραγματικού αριθμού που θέλουμε, είτε με το πόσα σωστά ψηφία επιθυμούμε ή δίνοντας το είδος άλλου πραγματικού αριθμού. Κατόπιν, ορίζουμε τους δικούς μας πραγματικούς αριθμούς συμπληρώνοντας στη δήλωση το **REAL** με το επιθυμητό είδος σε παρένθεση. Συνήθης εφαρμογή των παραπάνω είναι η εξής:

```
INTEGER, PARAMETER :: prc = SELECTED_REAL_KIND(12)

REAL (prc) :: c
```

ή

```
INTEGER, PARAMETER :: prc = KIND(1.0D0)

REAL (prc) :: b
```

Οι παραπάνω εντολές ορίζουν τη μεταβλητή b ως πραγματική ποσότητα με είδος ίδιο με αυτό του αριθμού `1.0D0`, δηλαδή γίνεται διπλής ακρίβειας. Ανάλογα, η μεταβλητή c είναι πραγματική με είδος κατάλληλο για να έχουμε τουλάχιστον 12 σωστά ψηφία. Προσέξτε ότι αν ζητήσουμε πολύ μεγάλη ακρίβεια, η τιμή που θα πάρει το prc δε θα αντιστοιχεί σε κανένα είδος, οπότε θα προκύψει λάθος κατά τη μεταγλώττιση.

Τα παραπάνω ισχύουν και για τους υπόλοιπους ενσωματωμένους τύπους. Ουσιαστική χρησιμότητα έχει μόνο η δυνατότητα να ορίσουμε μιγαδικούς αριθμούς διπλής ακρίβειας:

```
INTEGER, PARAMETER :: prc = KIND(1.0D0)

COMPLEX (prc) :: z
```

Με τις παραπάνω εντολές ο z είναι μιγαδική μεταβλητή που αποτελείται από δύο πραγματικούς διπλής ακρίβειας.

2.7.1 Προσδιορισμός του είδους αριθμητικών σταθερών

Στην Fortran 95 ισχύουν όσα αναφέραμε στην §2.3 για τις πραγματικές σταθερές. Έτσι, ο αριθμός 340 γράφεται ως 3.4E2 ή 340.0 αν θέλουμε να είναι απλής ακρίβειας ενώ γράφεται ως 3.4D2 ώστε να είναι διπλής ακρίβειας. Γενικότερα, αν επιθυμούμε να έχουμε τον αριθμό ως πραγματικό με είδος prc, τον γράφουμε ως εξής

```
340.0_prc
```

ή, ισοδύναμα,

```
3.4E2_prc
```

Συνήθης εφαρμογή των παραπάνω είναι η εξής:

```
INTEGER, PARAMETER :: prc = KIND(1.0D0)

REAL (prc) :: a, b
REAL (prc), PARAMETER :: pi = 3.141592653589793_prc

a = 2.4_prc
b = a + 5.6E2_prc
```

Ένα πολύ σημαντικό πλεονέκτημα του νέου τρόπου επιλογής του είδους, αν εφαρμοστεί συστηματικά, είναι ότι η αλλαγή της τιμής *μιας ποσότητας*, της σταθεράς που αντιστοιχεί στο είδος, αλλάζει σε όλο τον κώδικά μας την ακρίβεια των υπολογισμών.

2.8 Είσοδος/Έξοδος δεδομένων

Η εκχώρηση τιμής σε κάποια μεταβλητή a από το πληκτρολόγιο γίνεται, όπως είδαμε, με την εντολή **READ**:

```
READ *, a
```

ή, ισοδύναμα, με την

```
READ (*,*) a
```

Η εκτύπωση στην οθόνη της τιμής μιας ποσότητας γίνεται με την εντολή **PRINT**

```
PRINT *, a
```

ή, ισοδύναμα και πιο γενικά, με την εντολή **WRITE**:

```
WRITE (*,*) a
```

Οι παραπάνω εντολές μπορούν να χρησιμοποιηθούν για πολλά δεδομένα ταυτόχρονα, ακόμα και διαφορετικού τύπου. Π.χ.

```
INTEGER :: a
DOUBLE PRECISION :: b

a = 4
b = 4.5d0

PRINT *, "Oi times einai", a, b
```

Το πρώτο * στα **READ**(*,*)/**WRITE**(*,*) προσδιορίζει το πληκτρολόγιο ή την οθόνη ως πηγή ή αποδέκτη δεδομένων αντίστοιχα. Σε επόμενο κεφάλαιο θα δούμε πώς συνδέουμε αρχεία για την είσοδο και έξοδο δεδομένων. Το δεύτερο * καθορίζει ότι η μορφή των δεδομένων θα αποφασιστεί από τον μεταγλωττιστή με βάση τις ποσότητες που περιλαμβάνονται στις αντίστοιχες εντολές. Στη μορφή **READ** */**PRINT** * η μεταφορά των δεδομένων γίνεται μόνο από το πληκτρολόγιο και την οθόνη αντίστοιχα, χωρίς δυνατότητα αλλαγής, ενώ το * καθορίζει τη διαμόρφωση (και υποδηλώνει ότι αυτή θα αποφασιστεί από το μεταγλωττιστή).

2.9 Ενσωματωμένες αριθμητικές συναρτήσεις

Εκτός από αυτές (**AIMAG**(), **CONJG**(), **CMPLX**()) που παρουσιάσαμε στην §2.6.3, η Fortran παρέχει μια πληθώρα αριθμητικών συναρτήσεων. Οι πιο χρήσιμες παρουσιάζονται στον Πίνακα 2.1.

Προσέξτε ότι υπάρχουν συναρτήσεις που δέχονται πραγματικό όρισμα και επιστρέφουν αριθμό με τον ίδιο τύπο με το όρισμα (πραγματικό) και *ίδιας ακρίβειας*. Επομένως, αν οι συναρτήσεις αυτές δεχθούν όρισμα τύπου **DOUBLE PRECISION** επιστρέφουν **DOUBLE PRECISION**.

Παραδείγματα

- Ένα πρόγραμμα Fortran που υπολογίζει και τυπώνει την τετραγωνική ρίζα ενός πραγματικού αριθμού που εισάγεται από το χρήστη είναι το ακόλουθο:

```
PROGRAM riza

  IMPLICIT NONE
  DOUBLE PRECISION :: x
```

Πίνακας 2.1: Επιλεγμένες ενσωματωμένες συναρτήσεις της Fortran 95 (μέρος α').

Συνάρτηση	**Επιστρεφόμενη τιμή**	**Παρατηρήσεις**
`ABS(x)`	Απόλυτη τιμή.	Το όρισμα είναι ακέραιος ή πραγματικός.
`ABS(x)`	Μέτρο ($\sqrt{xx^*}$).	Το όρισμα είναι μιγαδικός.
`CEILING(x)`	Ο μικρότερος ακέραιος που δεν είναι μικρότερος από το όρισμα.	Το όρισμα είναι πραγματικός.
`FLOOR(x)`	Ο μεγαλύτερος ακέραιος που δεν είναι μεγαλύτερος από το όρισμα.	Το όρισμα είναι πραγματικός.
`INT(x)`	Ο μεγαλύτερος ακέραιος που δεν ξεπερνά το |x|, με το πρόσημο του x.	Το όρισμα είναι ακέραιος, πραγματικός ή μιγαδικός (οπότε περνά το πραγματικό μέρος του).
`NINT(x)`	Η πλησιέστερη ακέραια τιμή στο όρισμα.	Το όρισμα είναι πραγματικός, το αποτέλεσμα ακέραιος.
`AINT(x)`	Ο μεγαλύτερος ακέραιος που δεν ξεπερνά το |x|, με το πρόσημο του x.	Το όρισμα και το αποτέλεσμα είναι πραγματικά.
`ANINT(x)`	Αν x>0 επιστρέφει `AINT(x+0.5)` αλλιώς `AINT(x-0.5)`.	Το όρισμα και το αποτέλεσμα είναι πραγματικά.
`SQRT(x)`	Η τετραγωνική ρίζα του ορίσματος.	Το όρισμα πρέπει να είναι μιγαδικός ή μη αρνητικός πραγματικός.
`EXP(x)`	Εκθετικό (e^x).	
`LOG(x)`	Φυσικός λογάριθμος ($\ln x$).	Το όρισμα πρέπει να είναι πραγματικός θετικός ή μιγαδικός με φανταστικό μέρος $\in (-\pi, \pi]$.
`LOG10(x)`	Δεκαδικός λογάριθμος ($\log x$).	Το όρισμα πρέπει να είναι θετικό.
`MOD(x,y)`	Το υπόλοιπο της διαίρεσης x/y, δηλαδή το `x-INT(x/y)*y`.	Τα x,y ίδιου τύπου, ακέραιου ή πραγματικού. Πρέπει y$\neq 0$.
`MODULO(x,y)`	Επιστρέφει το $R \in [0, y)$ ώστε $x = Q * y + R$ με Q ακέραιο.	Τα x,y ακέραια, με y$\neq 0$.
`MODULO(x,y)`	Επιστρέφει το `x - FLOOR(x/y)*y`.	Τα x,y πραγματικά, με y$\neq 0$.

Πίνακας 2.1: Επιλεγμένες ενσωματωμένες συναρτήσεις της Fortran 95 (μέρος β').

Συνάρτηση	**Επιστρεφόμενη τιμή**	**Παρατηρήσεις**
`MAX(x1,x2,...)`	Ο μεγαλύτερος από τα δύο ή περισσότερα ορίσματα.	Τα ορίσματα πρέπει να είναι ίδιου τύπου, ακέραια ή πραγματικά.
`MIN(x1,x2,...)`	Ο μικρότερος από τα δύο ή περισσότερα ορίσματα.	Τα ορίσματα πρέπει να είναι ίδιου τύπου, ακέραια ή πραγματικά.
`DIM(x,y)`	Επιστρέφει το `MAX(x-y,0.0)`.	Τα ορίσματα πρέπει να είναι ίδιου τύπου, ακέραια ή πραγματικά.
`REAL(x)`	Το όρισμα ως πραγματικός απλής ακρίβειας.	Αν το όρισμα είναι μιγαδικός επιστρέφεται το πραγματικό του μέρος.
`DBLE(x)`	Το όρισμα ως πραγματικός διπλής ακρίβειας.	Αν το όρισμα είναι μιγαδικός επιστρέφεται το πραγματικό του μέρος.
`CONJG(x)`	Ο μιγαδικός συζυγής του ορίσματος.	Το όρισμα είναι μιγαδικός.
`AIMAG(x)`	Το φανταστικό μέρος του ορίσματος.	Το όρισμα είναι μιγαδικός.
`COS(x)`	Συνημίτονο.	Το πραγματικό ή μιγαδικό όρισμα σε rad.
`SIN(x)`	Ημίτονο.	Το πραγματικό ή μιγαδικό όρισμα σε rad.
`TAN(x)`	Εφαπτομένη.	Το πραγματικό ή μιγαδικό όρισμα σε rad.
`ACOS(x)`	Τόξο συνημιτόνου.	Το όρισμα στο $[-1 : 1]$, το αποτέλεσμα στο $[0 : \pi]$ σε rad.
`ASIN(x)`	Τόξο ημιτόνου.	Το όρισμα στο $[-1 : 1]$, το αποτέλεσμα στο $[-\pi/2 : \pi/2]$ σε rad.
`ATAN(x)`	Τόξο εφαπτομένης.	Το αποτέλεσμα στο $[-\pi/2 : \pi/2]$ σε rad.
`ATAN2(y, x)`	Τόξο εφαπτομένης $\tan^{-1}(y/x)$.	Τα πρόσημα των ορισμάτων καθορίζουν το τεταρτημόριο. Το αποτέλεσμα στο $(-\pi : \pi]$ σε rad.
`COSH(x)`	Υπερβολικό συνημίτονο.	
`SINH(x)`	Υπερβολικό ημίτονο.	
`TANH(x)`	Υπερβολική εφαπτομένη.	

```
  PRINT *, "Dwse ena pragmatiko"
  READ (*,*) x

  PRINT *, "H tetragwnikh riza einai"
  WRITE (*,*) SQRT(x)

END PROGRAM riza
```

- Ο υπολογισμός του ημιτόνου της γωνίας που δίνει ο χρήστης μπορεί να γίνει με το εξής πρόγραμμα

```
PROGRAM imitono

  IMPLICIT NONE
  DOUBLE PRECISION :: theta, x

  PRINT *, "Dwse ena pragmatiko"
  READ (*,*) theta

  x = SIN(theta)

  PRINT *, "To imitono einai"
  WRITE (*,*) x

END PROGRAM imitono
```

Εντός των παρενθέσεων που ακολουθούν το όνομα της συνάρτησης μπορεί να εμφανίζεται μία σταθερά ή μεταβλητή ή οποιαδήποτε πολύπλοκη έκφραση (που μπορεί να περιλαμβάνει άλλες συναρτήσεις), αρκεί να μπορεί να μετατραπεί στον κατάλληλο τύπο για το όρισμα, όπως αυτός προσδιορίζεται στον Πίνακα 2.1.

Προσέξτε ότι οι τριγωνομετρικές συναρτήσεις δέχονται ως όρισμα ή επιστρέφουν γωνία εκφρασμένη σε *ακτίνια (rad)*.

2.9.1 Γεννήτρια τυχαίων αριθμών

Η ενσωματωμένη υπορουτίνα **RANDOM_NUMBER**() δέχεται ως όρισμα μια πραγματική μεταβλητή. Κάθε φορά που καλείται, εκχωρεί στο όρισμά της ένα τυχαία επιλεγμένο αριθμό στο διάστημα $[0, 1)$. Θα αναφέρουμε στο Κεφάλαιο 8 ότι η κλήση μιας υπορουτίνας γίνεται με την εντολή **CALL** ακολουθούμενη από το όνομα της υπορουτίνας και τα ορίσματά της εντός παρενθέσεων. Επομένως, μπορούμε να παραγάγουμε δύο τυχαίους αριθμούς με το παρακάτω πρόγραμμα:

```
PROGRAM rand
  IMPLICIT NONE
```

```
  DOUBLE PRECISION :: x, y

  CALL RANDOM_NUMBER(x) ! to x einai tyxaios metaksy 0, 1
  CALL RANDOM_NUMBER(y) ! to y einai tyxaios metaksy 0, 1
END PROGRAM rand
```

Μπορούμε να επηρεάσουμε την ακολουθία τυχαίων που παράγεται από τη **RANDOM_NUMBER**() αν καλέσουμε την υπορουτίνα **RANDOM_SEED**(). Δεν θα επεκταθούμε στη χρήση της.

Αν επιθυμούμε να έχουμε τυχαίο πραγματικό σε κάποιο διάστημα $[a, b)$ τότε πρέπει να κάνουμε κατάλληλο μετασχηματισμό. Αν r είναι τυχαίος πραγματικός στο $[0, 1)$ τότε κάνουμε τη μετατροπή $x = \alpha r + \beta$ και επιλέγουμε τους συντελεστές α, β ώστε η ποσότητα x να βρίσκεται εντός των επιθυμητών ορίων. Μπορείτε να επαληθεύσετε ότι το $x = (b - a)r + a$ ικανοποιεί τις σχέσεις $a \leq x < b$.

Η υπορουτίνα **RANDOM_NUMBER**() είναι **ELEMENTAL** (§8.10) άρα μπορεί να δεχθεί ως όρισμα και ολόκληρο πραγματικό διάνυσμα ή πίνακα. Η εκχώρηση τυχαίων τιμών στα N στοιχεία του με μία κλήση της υπορουτίνας είναι πιο γρήγορη από N κλήσεις της για κάθε ένα στοιχείο.

2.10 Δομή Προγράμματος

Όπως αναφέραμε, οι εντολές του κυρίως προγράμματός μας περιλαμβάνονται μεταξύ των **PROGRAM**...**END PROGRAM**:

```
PROGRAM onoma

  IMPLICIT NONE
  ...
  ...
  ...
END PROGRAM onoma
```

Το όνομα του προγράμματος (εδώ «onoma») πρέπει να εμφανίζεται, είναι της επιλογής του προγραμματιστή και σχηματίζεται με τους κανόνες που ισχύουν για τα ονόματα των μεταβλητών. Όλες οι γραμμές μπορούν να ξεκινούν από οποιαδήποτε στήλη. Σε ένα αρχείο με κώδικα σε Fortran 95, επιτρέπονται κενά μεταξύ λέξεων και κενές γραμμές μεταξύ εντολών. Στην περίπτωση που κάποια γραμμή είναι πολύ μεγάλη, πάνω από 132 χαρακτήρες ή από το πλάτος της οθόνης μας, μπορούμε να την «σπάσουμε» σε οποιοδήποτε σημείο και να συνεχιστεί στην επόμενη γραμμή του αρχείου, αρκεί στο σημείο του «σπασίματος» στην *πρώτη γραμμή* να τοποθετήσουμε το χαρακτήρα '&'. Π.χ. η ακόλουθη εντολή

```
  INTEGER :: a, b
```

μπορεί να γίνει, αν το επιθυμούμε,

```
INTEGER :: a, &
b
```

Με άλλα λόγια, ο χαρακτήρας '&' στο τέλος μιας γραμμής υποδηλώνει ότι αυτή συνεχίζεται στην επόμενη (και αποτελούν μία εντολή).

Οι πρώτες γραμμές μετά το **PROGRAM** onoma συγκεντρώνουν τις δηλώσεις όλων των ποσοτήτων. Η πρώτη γραμμή από αυτές (καλό είναι να) είναι η **IMPLICIT NONE**. Αφού τελειώσουν οι δηλώσεις, ακολουθούν οι υπόλοιπες εντολές, με τη σειρά που θέλουμε να εκτελεστούν.

Πριν ή μετά το κυρίως πρόγραμμα μπορούμε να έχουμε ορισμούς συναρτήσεων, υπορουτινών και modules, §8. Αυτοί εκτελούνται μόνο αν χρησιμοποιηθούν από το κυρίως πρόγραμμα.

2.10.1 Εντολή STOP

Η εκτέλεση ενός προγράμματος ολοκληρώνεται κανονικά, μόλις η ροή συναντήσει το **END PROGRAM**. Εναλλακτικά, μπορούμε να δώσουμε την εντολή **STOP** σε οποιοδήποτε σημείο του. Η εκτέλεσή της προκαλεί την διακοπή του προγράμματος και την εκτύπωση σχετικού μηνύματος. Το **STOP** μπορεί να ακολουθείται από ένα σύνολο χαρακτήρων εντός εισαγωγικών, απλών ή διπλών, ή ένα ακέραιο αριθμό με έως 5 ψηφία. Σε τέτοια περίπτωση, το πρόγραμμα κατά τη διακοπή θα εκτυπώσει ό,τι ακολουθεί το **STOP**. Με αυτόν τον τρόπο μπορούμε να διακρίνουμε ποιο **STOP** (αν έχουμε πολλά) προκάλεσε τη διακοπή.

2.11 Ασκήσεις

1. Το παρακάτω πρόγραμμα υπολογίζει και τυπώνει στην οθόνη το άθροισμα δύο ακεραίων αριθμών που διαβάζει από το πληκτρολόγιο. Γράψτε το (όχι με copy-paste!) στο αρχείο *athroisi.f90*, μεταγλωττίστε το και εκτελέστε το.

```
PROGRAM athroisi
IMPLICIT NONE

INTEGER :: a, b
INTEGER :: c

PRINT *, "Dose dyo akeraious"
READ *, a,b

c = a + b

PRINT *, "To athroisma tous einai: "
PRINT *, c

END PROGRAM athroisi
```

2. Συμπληρώστε τον παραπάνω κώδικα ώστε να υπολογίζει και τη διαφορά, το γινόμενο, το πηλίκο και το υπόλοιπο της διαίρεσης των ακέραιων αριθμών εισόδου.

3. Επαναλάβετε το παραπάνω αλλά για πραγματικούς αριθμούς (προσέξτε ότι δεν ορίζεται πηλίκο και υπόλοιπο για πραγματικούς αλλά μόνο ο λόγος τους).

4. Γράψτε κώδικα στον οποίο θα δηλώνετε μεταβλητές κατάλληλου τύπου και θα εκχωρείτε σε αυτές τους αριθμούς 2.3167895443, $4 \cdot 10^3$, 10^{-2}, $3/2$. Τυπώστε τις τιμές των μεταβλητών. Είναι αυτές που αναμένετε;

5. Γράψτε κώδικα που να υπολογίζει τις παρακάτω εκφράσεις και να τυπώνει τις τιμές τους, αφού διαβάσει από το χρήστη τους πραγματικούς αριθμούς x, y, z:

   - d = $x^2 + y^2 + z^2$
   - d = $x^2/y + z$
   - d = $2.45(x + 1.5) + 3.1(y + 0.4) + 5.2 - z/2$
   - d = $(12.8x + 5y)/(11.3y + 4z)$
   - d = $x^{2/3} + y^{2/3} + z^{2/3}$

   Αν δώσετε $x = 1.5$, $y = 2.5$, $z = 3.5$ οι εκφράσεις πρέπει να έχουν τις τιμές: 20.75, 4.4, 19.79, 0.7502958579881658, 5.457604592453865.

6. Να επαληθεύσετε με πρόγραμμα ότι για δύο ακέραιους a, b, με b>0, ισχύει ότι n έκφραση `(a/b) * b + MOD(a,b)` είναι ίση με a. Επομένως, διαβάστε από τον χρήστη δύο ακέραιους αριθμούς a, b, υπολογίστε την έκφραση και τυπώστε τη στην οθόνη μαζί με το a. Οι δύο αριθμοί που θα τυπώσετε πρέπει να είναι ίδιοι.

7. Να γραφεί κώδικας ο οποίος θα κάνει τα παρακάτω:

   (α΄) θα εμφανίζει το μήνυμα «Δώστε την ακτίνα του κύκλου»,

   (β΄) θα διαβάζει από το πληκτρολόγιο την ακτίνα,

   (γ΄) θα υπολογίζει και θα εμφανίζει την περίμετρο του κύκλου (μαζί με κατάλληλο μήνυμα),

   (δ΄) θα υπολογίζει και θα εμφανίζει το εμβαδόν του κύκλου (μαζί με κατάλληλο μήνυμα).

   Δίνεται ότι η περίμετρος ενός κύκλου δίνεται από τη σχέση $\Gamma = 2\pi R$ και το εμβαδόν από τη σχέση $E = \pi R^2$.

8. Τρεις θετικοί ακέραιοι αριθμοί a, b, c που ικανοποιούν τη σχέση $a^2 + b^2 = c^2$ αποτελούν μία Πυθαγόρεια τριάδα. Μπορούμε να παραγάγουμε μια τέτοια τριάδα από δύο οποιουσδήποτε ακέραιους m, n με $m > n$, σχηματίζοντας τους αριθμούς $a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$.[4] Γράψτε πρόγραμμα που να διαβάζει δύο ακεραίους και να τυπώνει την αντίστοιχη Πυθαγόρεια τριάδα.

9. Να γράψετε κώδικα που θα δέχεται έναν τριψήφιο ακέραιο αριθμό και θα εμφανίζει το άθροισμα των ψηφίων του.

   Υπόδειξη: Βρείτε το πηλίκο και το υπόλοιπο της διαίρεσης του αριθμού με το 10. Τι παρατηρείτε;

10. Γράψτε πρόγραμμα που να δέχεται ένα ακέραιο αριθμό από το πληκτρολόγιο. Υποθέστε ότι ο αριθμός θα έχει τέσσερα ψηφία. Το πρόγραμμα θα υπολογίζει τα ψηφία αυτά και δημιουργεί τον «ανάστροφο» ακέραιο: το ψηφίο των μονάδων του αρχικού αριθμού θα είναι το ψηφίο των χιλιάδων του νέου, το ψηφίο των δεκάδων του αρχικού θα είναι το ψηφίο των εκατοντάδων του νέου κλπ. Στο τέλος, το πρόγραμμά σας θα τυπώνει το νέο, ανάστροφο ακέραιο.

11. Να γραφεί κώδικας ο οποίος θα δέχεται ένα χρονικό διάστημα σε δευτερόλεπτα και θα εμφανίζει τις μέρες, τις ώρες, τα λεπτά και τα υπόλοιπα δευτερόλεπτα στα οποία αντιστοιχεί.

    Για παράδειγμα: εάν δώσουμε ως είσοδο 200000, θα πρέπει να εμφανιστεί στην οθόνη το μήνυμα: “2 days, 7 hours, 33 min & 20 seconds”.

[4] Μπορείτε να το επαληθεύσετε εύκολα κάνοντας την αντικατάσταση.

12. Γράψτε πρόγραμμα που θα δέχεται από το πληκτρολόγιο ένα θετικό ακέραιο αριθμό. Ο αριθμός αυτός είναι χρονικό διάστημα μετρημένο σε δευτερόλεπτα. Το πρόγραμμά σας να υπολογίζει και να τυπώνει στην οθόνη σε πόσα χρόνια, μήνες, μέρες, ώρες, λεπτά και δευτερόλεπτα αντιστοιχεί αυτό. Υποθέστε ότι κάθε μήνας έχει 30 μέρες.

    Παράδειγμα: Τα 2034938471 δευτερόλεπτα είναι

    65 χρόνια, 5 μήνες, 2 ημέρες, 12 ώρες, 41 λεπτά, 11 δευτερόλεπτα.

13. Ρομπότ με σταθερό μήκος βήματος καταφθάνει στον πλανήτη Άρη για να περισυλλέξει πετρώματα. Κάθε βήμα του είναι $80\,\mathrm{cm}$. Το ρομπότ διαθέτει μετρητή βημάτων. Διένυσε στον Άρη μία ευθεία από σημείο Α σε σημείο Β και ο μετρητής βημάτων καταμέτρησε Ν βήματα.

    Να γραφεί πρόγραμμα που:

    (α΄) να διαβάζει τον αριθμό N των βημάτων του ρομπότ,

    (β΄) να υπολογίζει και να τυπώνει την απόσταση ΑΒ που διανύθηκε σε cm,

    (γ΄) να αναλύει και να τυπώνει αυτή την απόσταση σε km, m και cm.

    Για παράδειγμα, αν τα βήματα είναι 1253 τότε θέλουμε να τυπώνει: απόσταση $100\,240\,\mathrm{cm}$ ή $1\,\mathrm{km}$, $2\,\mathrm{m}$, $40\,\mathrm{cm}$. (ΟΕΦΕ, 2001)

14. Ένας αλγόριθμος για τον υπολογισμό της ημερομηνίας του Πάσχα των Ορθοδόξων σε συγκεκριμένο έτος (μέχρι το 2099) είναι ο εξής:

    - Θεωρούμε ως δεδομένο εισόδου το έτος που μας ενδιαφέρει.
    - Ορίζουμε κάποιες ακέραιες ποσότητες σύμφωνα με τους ακόλουθους τύπους:

      (α΄) r_1 = υπόλοιπο διαίρεσης του έτους με το 19.

      (β΄) r_2 = υπόλοιπο διαίρεσης του έτους με το 4.

      (γ΄) r_3 = υπόλοιπο διαίρεσης του έτους με το 7.

      (δ΄) $r_a = 19r_1 + 16$.

      (ε΄) r_4 = υπόλοιπο διαίρεσης του r_a με το 30.

      (στ΄) $r_b = 2(r_2 + 2r_3 + 3r_4)$.

      (ζ΄) r_5 = υπόλοιπο διαίρεσης του r_b με το 7.

      (η΄) $r_c = r_4 + r_5$.
    - Το r_c είναι πόσες ημέρες μετά την 3η Απριλίου του συγκεκριμένου έτους πέφτει το Πάσχα.

    Γράψτε σε κώδικα τον παραπάνω αλγόριθμο. Φροντίστε να τυπώνει κατάλληλο μήνυμα και τον αριθμό r_c, αφού τον υπολογίσει.

15. Γράψτε κώδικα που

(α΄) να διαβάζει τιμές σε δύο ακέραιες μεταβλητές,

(β΄) να εναλλάσσει τις τιμές αυτών των μεταβλητών,

(γ΄) να τυπώνει στην οθόνη τις νέες τους τιμές.

16. Δύο σωματίδια με φορτία q_1 και q_2 βρίσκονται ακίνητα στα σημεία με συντεταγμένες (x_1, y_1, z_1) και (x_2, y_2, z_2) αντίστοιχα. Η ηλεκτρική δύναμη που ασκείται στο δεύτερο σωματίδιο εξαιτίας του πρώτου είναι διάνυσμα με συνιστώσες

$$F_x = \frac{q_1 q_2}{4\pi\epsilon_0}\frac{x_2 - x_1}{d^{3/2}}\,, \qquad F_y = \frac{q_1 q_2}{4\pi\epsilon_0}\frac{y_2 - y_1}{d^{3/2}}\,, \qquad F_z = \frac{q_1 q_2}{4\pi\epsilon_0}\frac{z_2 - z_1}{d^{3/2}}\,,$$

όπου $d = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$. Η ποσότητα ϵ_0 είναι η ηλεκτρική σταθερά και έχει τιμή $8.854 \cdot 10^{-12}$ σε μονάδες SI. Θεωρήστε ότι $\pi \approx 3.14159$.

Γράψτε πρόγραμμα που να διαβάζει από το πληκτρολόγιο τα φορτία και τις συντεταγμένες θέσης για τα δύο σωματίδια και να τυπώνει στην οθόνη τις τιμές των συνιστωσών της δύναμης (σε μονάδες SI).

17. Από τα Μαθηματικά γνωρίζουμε ότι ισχύει

$$\begin{aligned} \pi &= \cos^{-1}(-1)\,, \\ \pi &= 8\tan^{-1}(1/3) + 4\tan^{-1}(1/7)\,, \\ 4\sin(\pi/12) &= \sqrt{6} - \sqrt{2}\,, \\ \tan(\pi/2) &= \infty\,. \end{aligned}$$

Γράψτε πρόγραμμα που να υπολογίζει και να τυπώνει το π από τις παραπάνω σχέσεις. Στην τελευταία θεωρήστε ότι $\infty = 1/0$.

Η ακριβής τιμή του π είναι $3.14159265358979323846264338\ldots$. Πόσα ψηφία προκύπτουν σωστά με το πρόγραμμά σας; Φροντίστε να έχετε τουλάχιστον 15 ψηφία σωστά.

18. Γράψτε πρόγραμμα που να δέχεται το πραγματικό και το φανταστικό μέρος δύο μιγαδικών αριθμών από το πληκτρολόγιο και να τυπώνει στην οθόνη, *στην πολική αναπαράσταση* (δηλαδή, με μέτρο και φάση), το άθροισμα, τη διαφορά, το γινόμενο και το λόγο τους.

19. Εξηγήστε γιατί `'1'+'2'` δεν κάνει `'3'`.

Κεφάλαιο 3

Εντολές Ελέγχου Ροής

3.1 Τελεστές σύγκρισης

Η Fortran υποστηρίζει τη σύγκριση αριθμητικών ποσοτήτων (ή ποσοτήτων τύπου χαρακτήρα) με τη βοήθεια των *σχεσιακών* τελεστών (Πίνακας 3.1). Το αποτέ-

`>`	μεγαλύτερο	`<`	μικρότερο
`>=`	μεγαλύτερο ή ίσο	`<=`	μικρότερο ή ίσο
`==`	ίσο	`/=`	άνισο

Πίνακας 3.1: Τελεστές σύγκρισης στη Fortran.

λεσμα της σύγκρισης είναι λογική ποσότητα, §2.6.2, και επομένως έχει τιμή `.TRUE.` ή `.FALSE.`. Π.χ. το `3.0>2.0` είναι `.TRUE.` ενώ το `2/=1+1` είναι `.FALSE.`. Οι αριθμητικοί τελεστές έχουν μεγαλύτερη προτεραιότητα από τους τελεστές σύγκρισης. Προσέξτε ότι, όπως και στα μαθηματικά, μόνο οι τελεστές `==,/=` έχουν νόημα για μιγαδικές ποσότητες.

Αν σε μία λογική έκφραση εμφανίζονται ποσότητες διαφορετικού τύπου, γίνονται οι προβλεπόμενες μετατροπές, §2.5, ώστε όλες οι ποσότητες να είναι ίδιου τύπου.

Παρατήρηση: Να αποφεύγετε τη σύγκριση για ισότητα μεταξύ πραγματικών ποσοτήτων· η πεπερασμένη αναπαράσταση των πραγματικών αριθμών οδηγεί σε σφάλματα στρογγύλευσης.

Για τη σύνδεση λογικών εκφράσεων η Fortran παρέχει τους λογικούς τελεστές `.NOT.`, `.AND.`, `.OR.`, `.EQV.`, `.NEQV.`. Όλοι, εκτός από τον πρώτο, δρουν μεταξύ δύο λογικών ποσοτήτων ή εκφράσεων και σχηματίζουν μια νέα λογική ποσότητα:

- Ο τελεστής `.NOT.` δρα στη λογική έκφραση που τον ακολουθεί και της αλλάζει την τιμή:

  Το `.NOT. (4 > 3)` είναι `.FALSE.`
  Το `.NOT. (4 < 3)` είναι `.TRUE.`

- Η λογική έκφραση που σχηματίζεται συνδέοντας δύο άλλες εκφράσεις με τον τελεστή `.AND.` έχει τιμή `.TRUE.` μόνο αν και οι δύο ποσότητες είναι `.TRUE.`. Σε άλλη περίπτωση είναι `.FALSE.`:

  Το `(4 > 3) .AND. (3.0 > 2.0)` είναι `.TRUE.`
  Το `(4 < 3) .AND. (3.0 > 2.0)` είναι `.FALSE.`

- Ο τελεστής `.OR.` μεταξύ δύο λογικών εκφράσεων σχηματίζει μια νέα ποσότητα με τιμή `.TRUE.` αν έστω και μία από τις δύο ποσότητες είναι `.TRUE.`, αλλιώς είναι `.FALSE.`:

  Το `(4 > 3) .OR. (3.0 < 2.0)` είναι `.TRUE.`
  Το `(4 < 3) .OR. (3.0 < 2.0)` είναι `.FALSE.`

- Η λογική έκφραση που σχηματίζεται συνδέοντας δύο άλλες λογικές εκφράσεις με τον τελεστή `.EQV.` έχει τιμή `.TRUE.` αν οι δύο ποσότητες έχουν την *ίδια* τιμή· αλλιώς είναι `.FALSE.`:

  Το `(4 < 3) .EQV. (3.0 < 2.0)` είναι `.TRUE.`
  Το `(4 < 3) .EQV. (3.0 > 2.0)` είναι `.FALSE.`

- Ο τελεστής `.NEQV.` μεταξύ λογικών εκφράσεων σχηματίζει ποσότητα με τιμή `.TRUE.` αν οι δύο ποσότητες έχουν *διαφορετική* τιμή· αλλιώς είναι `.FALSE.`:

  Το `(4 < 3) .NEQV. (3.0 < 2.0)` είναι `.FALSE.`
  Το `(4 < 3) .NEQV. (3.0 > 2.0)` είναι `.TRUE.`

Παρατηρήστε ότι οι δύο τελευταίοι τελεστές, `.EQV.` και `.NEQV.`, ουσιαστικά ελέγχουν την ισότητα ή ανισότητα λογικών ποσοτήτων. Δεν επιτρέπεται να χρησιμοποιήσουμε τους τελεστές `==`, `/=` για σύγκριση λογικών ποσοτήτων.

Μια έκφραση με σχεσιακούς ή λογικούς τελεστές της Fortran μπορεί να ανατεθεί σε μεταβλητές τύπου `LOGICAL`:

```
LOGICAL :: a, b
INTEGER :: i, max

...

a = 3==2
b = ( i > 0 .AND. i < max )
```

Όσον αφορά τις προτεραιότητες, οι λογικοί τελεστές έχουν χαμηλότερη προτεραιότητα από τους τελεστές σύγκρισης (και επομένως και τους αριθμητικούς τελεστές). Μεταξύ των λογικών τελεστών το .**NOT**. έχει την υψηλότερη προτεραιότητα, ακολουθεί το .**AND**. και τέλος το .**OR**., που έχει τη χαμηλότερη προτεραιότητα από όλους.

3.2 Εντολές Ελέγχου Ροής

3.2.1 IF

Η εντολή **IF** είναι μία από τις βασικές δομές διακλάδωσης κάθε γλώσσας προγραμματισμού. Ελέγχει τη ροή του κώδικα ανάλογα με τη (λογική) τιμή μιας συνθήκης, δηλαδή μιας ποσότητας ή έκφρασης λογικού τύπου. Στη Fortran συντάσσεται ως εξής:

```
IF (condition) THEN
    ...
    ...       // block A
ELSE
    ...
    ...       // block B
END IF
```

Εάν η συνθήκη («condition»), είναι αληθής, εκτελούνται οι εντολές του τμήματος A. Αν η συνθήκη είναι ψευδής, τότε εκτελείται το block των εντολών μεταξύ του **ELSE** και του **END IF** (τμήμα B).

Το κάθε τμήμα μπορεί να αποτελείται από καμία, μία, ή περισσότερες εντολές. Π.χ.

```
IF (val > 0.0d0) THEN
   PRINT *, "O arithmos einai thetikos"
   max = val
ELSE
   PRINT *, "O arithmos den einai thetikos"
END IF
```

Το κάθε τμήμα μπορεί να περιλαμβάνει οποιεσδήποτε άλλες εντολές και βέβαια άλλο **IF**. Παρατηρήστε ότι η στοίχιση των εντολών υποδηλώνει ότι βρίσκονται στο «εσωτερικό» μιας (σύνθετης) εντολής. Δεν είναι υποχρεωτική αλλά διευκολύνει τον προγραμματιστή στην κατανόηση του κώδικα.

Στην περίπτωση που το τμήμα μετά το **ELSE** είναι κενό, η λέξη **ELSE** μπορεί να παραληφθεί:

```
IF (condition) THEN
    ...
    ...
```

```
END IF
```

Εναλλακτική σύνταξη της περίπτωσης χωρίς τμήμα **ELSE** και με μία μόνο εντολή στο σώμα του **IF**, δηλαδή της

```
IF (condition) THEN
   command
END IF
```

είναι η ακόλουθη:

```
IF (condition) command
```

Παρατηρήστε ότι παραλείπονται τα **THEN**, **END IF**.

Αναφέραμε ότι στο σώμα ενός **IF** μπορεί να περιέχεται άλλο **IF**, π.χ.

```
IF (cond1) THEN
  ...
ELSE
  IF (cond2) THEN
    ...
  ELSE
    ...
  END IF
END IF
```

Συντετμημένη μορφή της παραπάνω δομής είναι η

```
IF (cond1) THEN
  ...
ELSE IF (cond2) THEN
    ...
ELSE
    ...
END IF
```

Γενικότερα, μπορούμε να έχουμε μια ακολουθία από **ELSE IF**:

```
IF (cond1) THEN
  ...
ELSE IF (cond2) THEN
  ...
ELSE IF (cond3) THEN
  ...
ELSE IF (cond4) THEN
  ...
ELSE
  ...
END IF
```

Στη συγκεκριμένη μορφή του **IF**, γίνεται έλεγχος των συνθηκών cond1, cond2, ...*προτού εκτελεστεί η εντολή*. Η πρώτη από τις συνθήκες που θα βρεθεί αληθής, θα καθορίσει ποιο τμήμα εντολών θα εκτελεστεί (είναι αυτές παρατίθενται μετά το αντίστοιχο **IF**). Αν δεν ικανοποιείται καμία συνθήκη, εκτελούνται οι εντολές μετά το **ELSE** (αν υπάρχει) ή την επόμενη εντολή του **IF**.

3.2.2 SELECT CASE

Η δομή **SELECT CASE** αποτελεί κάποιες φορές, μια πιο κομψή και κατανοητή εκδοχή πολλαπλών εσωκλειόμενων ή διαδοχικών **IF**. Η σύνταξή της είναι:

```
SELECT CASE (i)
  CASE (value1)
  ...
  CASE (value2)
  ...
  ...
  ...
  CASE (valueN)
  ...
  CASE default
  ...
END SELECT
```

Η ποσότητα ελέγχου i πρέπει να είναι μεταβλητή (ή έκφραση) ακέραιου τύπου (ή λογικού ή χαρακτήρα), *όχι πραγματικού*. Οι τιμές value1, value2, ...,valueN πρέπει να είναι σταθερές αντίστοιχου τύπου και διακριτές μεταξύ τους (να μην επαναλαμβάνονται).

Κατά την εκτέλεση, η τιμή ελέγχου i συγκρίνεται με κάθε μία από τις value1, value2,...,valueN. Αν η τιμή της περιλαμβάνεται σε αυτές, τότε εκτελούνται οι εντολές που ακολουθούν το αντίστοιχο **CASE**. Όταν αυτές εξαντληθούν, η εκτέλεση συνεχίζει με την πρώτη εντολή μετά το **END SELECT**. Αν δεν βρεθεί η τιμή της στις value1, value2,...,valueN εκτελείται το τμήμα των εντολών του **default**, αν υπάρχει. Αλλιώς, η εκτέλεση συνεχίζει μετά το **END SELECT**.

Οι σχετικές θέσεις των **CASE** μπορούν να είναι οποιεσδήποτε.

Παράδειγμα
Έστω ότι σε ένα παιχνίδι θέλουμε να αντιστοιχίσουμε ένα αριθμό που θα πληκτρολογεί ο χρήστης με την κατεύθυνση στην οποία θα κινηθεί ο ήρωας του παιχνιδιού. Έστω ακόμη ότι η συσχέτιση είναι $0 \rightarrow$ «επάνω», $1 \rightarrow$ «κάτω», $2 \rightarrow$ «αριστερά», $3 \rightarrow$ «δεξιά», ενώ για άλλη τιμή να μένει ακίνητος. Μπορούμε να γράψουμε τον εξής κώδικα

```
INTEGER :: i
CHARACTER (5) :: direction
```

```
READ *, i

SELECT CASE (i)
  CASE (0)
      direction = "up"
  CASE (1)
      direction = "down"
  CASE (2)
      direction = "left"
  CASE (3)
      direction = "right"
  CASE default
      direction = "none"
END SELECT

PRINT *, "H kateythinsi einai ", direction
```

Όταν επιθυμούμε να εκτελεστούν οι ίδιες εντολές για πολλές τιμές του i, μπορούμε να τις ομαδοποιήσουμε. Π.χ., έστω ότι οι αριθμοί $(0, 4, 8)$ είναι ισοδύναμοι και αντίστοιχα οι $(1, 5, 9)$, $(2, 6)$ και $(3, 7)$. Ο κώδικας δεν επαναλαμβάνεται αν τον τροποποιήσουμε ως εξής

```
SELECT CASE (i)
  CASE (0,4,8)
      direction = "up"
  CASE (1,5,9)
      direction = "down"
  CASE (2,6)
      direction = "left"
  CASE (3,7)
      direction = "right"
  CASE default
      direction = "none"
END SELECT
```

Τέλος, στην περίπτωση που έχουμε συνεχόμενες επιθυμητές τιμές, μπορούμε να τις δώσουμε με τη μορφή

χαμηλή τιμή : υψηλή τιμή

Έτσι, ένα **CASE** με τη μορφή

```
CASE (0:9)
  ...
```

εκτελείται όταν η τιμή ελέγχου είναι στο διάστημα $[0, 9]$. Αν παραληφθεί το ένα από τα δύο όρια, θεωρείται ότι έχουμε βάλλει το μικρότερο ή μεγαλύτερο (ανάλογα) αριθμό της Fortran. Π.χ.

```
SELECT CASE (i)
  CASE (:-1)
  ...    ! entoles otan i < 0
  CASE (0)
  ...    ! entoles otan i == 0
  CASE (1:)
  ...    ! entoles otan i > 0
END SELECT
```

3.3 Ασκήσεις

1. Γράψτε πρόγραμμα που θα δέχεται ένα ακέραιο αριθμό από το χρήστη και θα ελέγχει αν είναι άρτιος, τυπώνοντας κατάλληλο μήνυμα.

   Υπόδειξη: Άρτιος είναι ο ακέραιος που το υπόλοιπο της διαίρεσής του με το 2 είναι 0.

2. Γράψτε πρόγραμμα που θα υπολογίζει τον όγκο και το εμβαδόν της επιφάνειας μιας σφαίρας, αφού ζητήσει από το χρήστη την ακτίνα της. Αν ο χρήστης δώσει κατά λάθος αρνητικό αριθμό, να τυπώνει κατάλληλο μήνυμα.

3. Γράψτε πρόγραμμα που να επιλύει την πρωτοβάθμια εξίσωση $ax = b$, με τιμές των a, b που θα παίρνει από το χρήστη. Προσέξτε να κάνετε διερεύνηση ανάλογα με τις τιμές των a, b, δηλαδή: Ποια είναι η λύση αν (α΄) $a \neq 0$, (β΄) αν το $a = 0$ και $b = 0$, (γ΄) αν το $a = 0$ και $b \neq 0$.

4. Γράψτε πρόγραμμα που να τυπώνει τις λύσεις της δευτεροβάθμιας εξίσωσης $ax^2 + bx + c = 0$, με τιμές των (πραγματικών) a, b, c που θα παίρνει από το χρήστη. Προσέξτε να κάνετε διερεύνηση ανάλογα με τις τιμές των a, b, c, τυπώνοντας πέρα από τις λύσεις και κατάλληλα μηνύματα. Όταν οι λύσεις είναι μιγαδικές (δηλαδή, όταν η διακρίνουσα είναι αρνητική), το πρόγραμμα να πληροφορεί το χρήστη για αυτό, χωρίς να τις υπολογίζει.

5. Να τροποποιήσετε τον κώδικα που γράψατε για την επίλυση του τριωνύμου ώστε να λάβετε υπόψη την περίπτωση που υπάρχουν μιγαδικές λύσεις.

6. Γράψτε πρόγραμμα που να υπολογίζει το μέγιστο/ελάχιστο από 5 ακέραιους αριθμούς.

   Υπόδειξη: Θεωρήστε ότι ο μεγαλύτερος είναι ο πρώτος. Αποθηκεύστε τον σε μια μεταβλητή. Συγκρίνετε τον τρέχοντα μεγαλύτερο (τη νέα μεταβλητή) με το δεύτερο αριθμό ώστε να βρείτε το νέο μεγαλύτερο. Αποθηκεύστε τον στη μεταβλητή. Συγκρίνετε τον τρέχοντα μεγαλύτερο με τον τρίτο αριθμό ώστε να βρούμε τον νέο μεγαλύτερο, κλπ.

7. Το Υπουργείο Οικονομικών ανακοίνωσε ότι φορολογεί τα εισοδήματα των μισθωτών που αποκτήθηκαν κατά το έτος 2015 με βάση την παρακάτω κλίμακα:

Εισόδημα (σε Ευρώ)	Συντελεστής Φόρου
0 - 20000	22%
20000,01 - 30000	29%
30000,01 - 40000	37%
από 40000,01 και πάνω	45%

   Για παράδειγμα, εάν κάποιος έχει εισόδημα 48000€, για τα πρώτα 20000€ θα φορολογηθεί με ποσοστό 22% (φόρος 4400€), για τα επόμενα 10000€ θα

φορολογηθεί με ποσοστό 29% (φόρος 2900€), για τα επόμενα 10000€ θα φορολογηθεί με ποσοστό 37% (φόρος 3700€) και για τα υπόλοιπα 8000€ θα φορολογηθεί με ποσοστό 45% (φόρος 3600€). Συνολικά θα πληρώσει 14600€.

Γράψτε κώδικα που θα διαβάζει από το πληκτρολόγιο ένα ποσό (το εισόδημα) και θα υπολογίζει το φόρο που του αναλογεί.

8. Ο αλγόριθμος του Zeller υπολογίζει την ημέρα (Κυριακή, Δευτέρα, ...) κάποιας ημερομηνίας (στο Γρηγοριανό ημερολόγιο) ως εξής:

   Έστω d είναι η ημέρα του μήνα $(1, 2, 3, \ldots, 31)$, m ο μήνας $(1, 2, \ldots, 12)$ και y το έτος. Αν ο μήνας είναι 1 (Ιανουάριος) ή 2 (Φεβρουάριος) προσθέτουμε στο m το 12 και αφαιρούμε 1 από το έτος y. Κατόπιν,

   (α′) Ορίζουμε το a να είναι το πηλίκο της διαίρεσης του $13(m + 1)$ με το 5.

   (β′) Ορίζουμε τα j, k να είναι το πηλίκο και το υπόλοιπο αντίστοιχα, της διαίρεσης του έτους y με το 100.

   (γ′) Ορίζουμε το b να είναι το πηλίκο της διαίρεσης του j με το 4.

   (δ′) Ορίζουμε το c να είναι το πηλίκο της διαίρεσης του k με το 4.

   (ε′) Ορίζουμε το h να είναι το άθροισμα των a, b, c, d, k και του πενταπλάσιου του j.

   Το υπόλοιπο της διαίρεσης του h με το 7 είναι η ημέρα: αν είναι 0 η ημέρα είναι Σάββατο, αν είναι 1 η ημέρα είναι Κυριακή, κλπ.

   Γράψτε πρόγραμμα που να δέχεται μια ημερομηνία από το χρήστη και να τυπώνει την ημέρα της εβδομάδας αυτής της ημερομηνίας.

9. Έχετε τις εξής πληροφορίες:

   - Οι μήνες Ιανουάριος, Μάρτιος, Μάιος, Ιούλιος, Αύγουστος, Οκτώβριος, Δεκέμβριος έχουν 31 ημέρες.
   - Οι μήνες Απρίλιος, Ιούνιος, Σεπτέμβριος, Νοέμβριος έχουν 30 ημέρες.
   - Ο Φεβρουάριος έχει 28 ημέρες εκτός αν το έτος είναι δίσεκτο, οπότε έχει 29.
   - Η αλλαγή από το παλαιό στο νέο ημερολόγιο έγινε στις 16 Φεβρουαρίου 1923 (με το παλαιό) που ορίστηκε ως 1η Μαρτίου 1923 (στο νέο). Συνεπώς, ο Φεβρουάριος του 1923 είχε 15 ημέρες.
   - Πριν το 1923, δίσεκτα είναι τα έτη που διαιρούνται ακριβώς με το 4.
   - Μετά το 1923, δίσεκτα είναι τα έτη που διαιρούνται ακριβώς με το 4, εκτός από τις εκατονταετίες. Οι εκατονταετίες είναι δίσεκτες όταν διαιρούνται με το 400. Επομένως: ένα έτος μετά το 1923 που διαιρείται ακριβώς με το 4 αλλά όχι με το 100 είναι δίσεκτο. Είναι επίσης δίσεκτο αν διαιρείται ακριβώς με το 400.

Γράψτε κώδικα, χρησιμοποιώντας το **`SELECT CASE`**, που να διαβάζει μήνα και έτος από το χρήστη και να τυπώνει στην οθόνη τις ημέρες του συγκεκριμένου μήνα.

10. Γράψτε πρόγραμμα που να υπολογίζει την τιμή της παρακάτω συνάρτησης για κάποια δεδομένη τιμή του x που θα διαβάζει από το πληκτρολόγιο:

$$y = \begin{cases} (x^2+1)\mathrm{e}^{-x} & x > 3 \\ \sqrt{x-1} & 1 \leq x \leq 3 \\ \dfrac{x-2}{x+5} + \dfrac{x-3}{x+7} & x < 1, x \neq -5, x \neq -7 \end{cases}$$

11. Δύο σωματίδια με μάζες m_1 και m_2 βρίσκονται ακίνητα στα σημεία με συντεταγμένες (x_1, y_1, z_1) και (x_2, y_2, z_2) αντίστοιχα. Η βαρυτική δύναμη μεταξύ τους έχει μέτρο

$$F = G\frac{m_1 m_2}{r^2} \ ,$$

όπου $r = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$ η απόσταση των σωματιδίων και G η βαρυτική σταθερά με τιμή $6.674 \cdot 10^{-11}$ σε μονάδες SI.

Γράψτε πρόγραμμα που

- να διαβάζει από το πληκτρολόγιο τις μάζες και τις συντεταγμένες θέσης για τα δύο σωματίδια στις μεταβλητές m1, x1, y1, z1 και m2, x2, y2, z2.
- Να υπολογίζει και να τυπώνει την απόστασή τους.
- να υπολογίζει το μέτρο της μεταξύ τους δύναμης. Να το τυπώνει στην οθόνη μόνο αν είναι μεγαλύτερο από 10^{-10} αλλιώς να τυπώνει 0.

Κεφάλαιο 4

Εντολή επανάληψης

4.1 Εισαγωγή

Βασική ανάγκη ύπαρξης ενός υπολογιστή είναι να εκτελέσει επαναληπτικά μια διαδικασία, εξασφαλίζοντας την ακρίβεια και ταχύτητα στην εκτέλεσή της. Η σύνθετη εντολή **DO** υλοποιεί στη Fortran τον επαναληπτικό βρόχο, το βασικό χαρακτηριστικό κάθε γλώσσας προγραμματισμού. Παρουσιάζεται με δύο παραλλαγές:

- για συγκεκριμένο, γνωστό, αριθμό επαναλήψεων, και
- για άγνωστο εκ των προτέρων, πλήθος επαναλήψεων.

4.2 DO για συγκεκριμένο πλήθος επαναλήψεων

Για να καταλάβουμε την ανάγκη ύπαρξης της εντολής επανάληψης ας δούμε το εξής πρόβλημα: Έστω ότι θέλουμε να τυπώσουμε στην οθόνη τους αριθμούς $1, 2, 3, 4, 5$, σε ξεχωριστή γραμμή τον καθένα. Η εκτύπωση, σύμφωνα με όσα ξέρουμε μέχρι τώρα, μπορεί να γίνει ως εξής

```
WRITE (*,*) 1
WRITE (*,*) 2
WRITE (*,*) 3
WRITE (*,*) 4
WRITE (*,*) 5
```

Η προσέγγιση αυτή είναι εφικτή καθώς το πλήθος των αριθμών είναι πολύ μικρό. Πώς θα μπορούσαμε να επεκτείνουμε αυτές τις εντολές, σύντομα και σωστά, αν θέλαμε να τυπώσουμε μέχρι π.χ. το 500;

Μπορούμε να τροποποιήσουμε τις παραπάνω εντολές ώστε να τις φέρουμε σε κατάλληλη μορφή για επανάληψη. Έτσι, η εκτύπωση μπορεί να γίνει ως εξής

```
INTEGER :: i

i=1
WRITE (*,*) i

i=2
WRITE (*,*) i

i=3
WRITE (*,*) i

i=4
WRITE (*,*) i

i=5
WRITE (*,*) i
```

Παρατηρήστε ότι φέραμε τον κώδικα στη μορφή που μια εντολή επαναλαμβάνεται αυτούσια ενώ, πριν από τη συγκεκριμένη εντολή, μια ακέραια μεταβλητή αυξάνεται διαδοχικά κατά σταθερό βήμα, 1. Η χρήση της εντολής **DO** απλοποιεί τον κώδικα ως εξής

```
INTEGER :: i
DO i = 1,5
   WRITE (*,*) i
END DO
```

Σε αυτή τη μορφή, η τροποποίηση του κώδικα ώστε να εκτυπώνει τους αριθμούς π.χ. μέχρι το 500 είναι απλή: αλλάζουμε το άνω όριο της μεταβλητής `i`.

4.2.1 Σύνταξη

Η γενικευμένη σύνταξη της **DO** για συγκεκριμένο πλήθος επαναλήψεων είναι η ακόλουθη

DO μεταβλητή = αρχική τιμή, τελική τιμή, βήμα αύξησης
....
....
END DO

Η «μεταβλητή» είναι *υποχρεωτικά ακέραια* και αναφέρεται ως μεταβλητή ελέγχου. Οι ποσότητες «αρχική τιμή», «τελική τιμή», «βήμα αύξησης» είναι σταθεροί *ακέραιοι* αριθμοί, *ακέραιες* μεταβλητές ή σύνθετες εκφράσεις που δίνουν *ακέραια* τιμή. Το

«βήμα αύξησης» αν παραλείπεται (μαζί με το (,) που προηγείται) θεωρείται ότι είναι 1. Αν δίνεται, επιτρέπεται να είναι θετικό ή αρνητικό αλλά όχι 0.

Η εκτέλεση της σύνθετης εντολής γίνεται σε βήματα ως εξής:

1. η «μεταβλητή» αποκτά αρχικά την «αρχική τιμή».
2. γίνεται ο ακόλουθος έλεγχος:
   - Αν το «βήμα αύξησης» είναι θετικό, ελέγχεται αν η «μεταβλητή» έχει τιμή μικρότερη ή ίση από την «τελική τιμή».
   - Αν το «βήμα αύξησης» είναι αρνητικό, ελέγχεται αν η «μεταβλητή» έχει τιμή μεγαλύτερη ή ίση από την «τελική τιμή».

   Ισοδύναμα, μπορούμε να πούμε ότι ελέγχεται αν με διαδοχικές προσθέσεις του «βήματος αύξησης» στη «μεταβλητή» μπορούμε να φτάσουμε ή να ξεπεράσουμε την «τελική τιμή».
3. Αν ο έλεγχος στο βήμα 2 είναι αρνητικός, η ροή του προγράμματος συνεχίζει με την πρώτη εντολή μετά το **`END DO`**, διακόπτοντας την επανάληψη.
4. Αν ο έλεγχος στο βήμα 2 είναι θετικός, εκτελούνται οι εντολές (καμία, μία ή περισσότερες) που περικλείονται μεταξύ των **`DO`**, **`END DO`**.
5. Προστίθεται στη «μεταβλητή» το «βήμα αύξησης» και επαναλαμβάνεται η διαδικασία από το βήμα 2.

Εύκολα συμπεραίνουμε ότι θα γίνουν τελικά (το πολύ)

$$\max((\text{«τελική τιμή»} - \text{«αρχική τιμή»} + \text{«βήμα αύξησης»})/\text{«βήμα αύξησης»}),0)$$

επαναλήψεις. Επιπλέον, συνέπεια της παραπάνω διαδικασίας είναι ότι η «μεταβλητή» (η οποία φυσικά πρέπει να έχει δηλωθεί) θα έχει την «αρχική τιμή» αν δεν εκτελεστεί καθόλου η επανάληψη, την τιμή

> «τελική τιμή» − υπόλοιπο της διαίρεσης της ποσότητας («τελική τιμή» − «αρχική τιμή») με την απόλυτη τιμή της ποσότητας «βήμα αύξησης»

αν εκτελεστεί πλήρως, ενώ θα έχει μια ενδιάμεση τιμή αν η επανάληψη διακοπεί (π.χ. με την εντολή **`EXIT`**, §4.4.1) πριν την κανονική λήξη της.

Υπάρχει η δυνατότητα (αν και είναι κακή πρακτική) να τροποποιηθούν, αν είναι μεταβλητές, οι ποσότητες «αρχική τιμή», «τελική τιμή», «βήμα αύξησης» κατά τη διάρκεια της επανάληψης. Οι νέες τους τιμές, όμως, *δεν παίζουν κανένα ρόλο* στο πλήθος των επαναλήψεων· αυτό καθορίζεται από τις αρχικές τιμές τους. Επιπλέον, δεν επιτρέπεται η αλλαγή τιμής στη μεταβλητή ελέγχου μεταξύ **`DO`** και **`END DO`**.

Παράδειγμα

Έστω ότι θέλουμε να υπολογίσουμε το άθροισμα των περιττών ακέραιων αριθμών από το 1 έως και το 9 στην ακέραια μεταβλητή s. Προφανώς, μπορούμε να γράψουμε

```
INTEGER :: s
s = 1 + 3 + 5 + 7 + 9
```

Ας κάνουμε τον υπολογισμό με τη χρήση της εντολής επανάληψης. Μπορούμε, σε πρώτο στάδιο, να γράψουμε το ακόλουθο:

```
INTEGER :: s
s = 0
s = s + 1
s = s + 3
s = s + 5
s = s + 7
s = s + 9
```

Προσθέτουμε επομένως, σταδιακά, έναν-έναν, τους όρους στη μεταβλητή s, την οποία μηδενίζουμε αρχικά ώστε να έχει ουδέτερη τιμή στην πρώτη χρήση της σε άθροισμα.

Σε δεύτερο στάδιο, διαμορφώνουμε τον κώδικα ώστε να έχουμε μια εντολή που επαναλαμβάνεται αυτούσια ενώ αυξάνεται σταθερά μία ακέραια μεταβλητή:

```
INTEGER :: s
INTEGER :: i
s = 0

i = 1
s = s + i

i = i + 2
s = s + i

i = i + 2
s = s + i

i = i + 2
s = s + i

i = i + 2
s = s + i
```

Η χρήση της εντολής επανάληψης απλοποιεί τον τελικό κώδικά μας:

```
INTEGER :: s
INTEGER :: i
s = 0
DO i = 1, 9, 2
   s = s + i
END DO
```

4.2.2 Χρήση

Βασιζόμενοι στην παραπάνω ανάλυση, αν έχουμε εντολές που επαναλαμβάνονται, φροντίζουμε να τις φέρουμε στη μορφή

- «ακέραια μεταβλητή» = «ακέραια μεταβλητή» + «βήμα αύξησης»
- σύνολο εντολών, εξαρτώμενων ή μη από την «ακέραια μεταβλητή», αλλά με την ίδια μορφή ανεξάρτητα από την τιμή της μεταβλητής.

Αν το επιτύχουμε αυτό, τότε μπορούμε να γράψουμε την παραπάνω σειρά επαναλαμβανόμενων εντολών πιο συνοπτικά και κατανοητά με τη χρήση της εντολής `DO`.

Παραδείγματα

- Το άθροισμα των πρώτων 100 ακεραίων αριθμών, από το 1 ως το 100, υπολογίζεται με τον ακόλουθο κώδικα:

```
INTEGER :: sum, i
sum = 0
DO i = 1, 100
  sum = sum + i
END DO
```

- Το άθροισμα των άρτιων ακεραίων αριθμών στο διάστημα $[0, 1000]$ υπολογίζεται με τον ακόλουθο κώδικα:

```
INTEGER :: sum, i
sum = 0
DO i = 0, 1000, 2
  sum = sum + i
END DO
```

- Η εκτύπωση των αριθμών 99, 97, 95,..., 3, 1, με αυτή τη σειρά, μπορεί να γίνει με τον κώδικα

```
INTEGER :: i
DO i = 99, 1, -2
  WRITE (*,*) i
END DO
```

- Η εκτύπωση των αριθμών 0.0, 0.1, 0.2, ..., 99.9, 100.0 γίνεται ως εξής

```
INTEGER :: i
DO i = 0, 1000
  WRITE (*,*) 0.1d0 * i
END DO
```

Προσέξτε ότι καθώς δεν επιτρέπεται να έχουμε μη ακέραια μεταβλητή ελέγχου είναι λάθος να γράψουμε τον ακόλουθο κώδικα (που θα φαινόταν προφανής)

```
DOUBLE PRECISION :: x
DO x = 0.0d0, 100.0d0, 0.1d0     ! Λάθος
  WRITE (*,*) x
END DO
```

- Το πλήθος των άρτιων ακεραίων στο διάστημα $[5, 108]$ μπορεί να υπολογιστεί με τη χρήση εντολής επανάληψης ως εξής

```
INTEGER :: k, i
k = 0
DO i = 5, 108
   IF (MOD(i,2) == 0) k = k + 1
END DO
```

Στο παράδειγμα αυτό, το πλήθος υπολογίζεται στη μεταβλητή k που χρησιμοποιείται ως *μετρητής*. Ο μετρητής στον προγραμματισμό είναι μια ακέραια μεταβλητή που παίρνει αρχική τιμή 0 *ακριβώς πριν* την εντολή επανάληψης και αυξάνει κατά ένα όταν ικανοποιείται κάποια συνθήκη. Με αυτή την τεχνική μετράμε πόσες φορές στο διάστημα μεταβολής της μεταβλητής ελέγχου ήταν αληθής μία συνθήκη.

Στην περίπτωση που ενδιαφερόμαστε όχι για το *πόσες φορές* αληθεύει μία συνθήκη σε κάποιο διάστημα μεταβολής της μεταβλητής ελέγχου αλλά μόνο για το *αν* αληθεύει, ταιριάζει να χρησιμοποιήσουμε ως μετρητή μια μεταβλητή λογικού τύπου. Η μεταβλητή αυτή θα ξεκινά πριν την επανάληψη με κάποια τιμή και θα αλλάζει όταν ικανοποιηθεί κάποια συνθήκη.

- Η ακολουθία αριθμών 0,1,1,2,3,5,8,13,..., στην οποία κάθε μέλος της, από το τρίτο και μετά, είναι το άθροισμα των δύο προηγούμενων μελών, είναι η ακολουθία Fibonacci. Αν θέλουμε να τυπώσουμε στην οθόνη τους n πρώτους όρους, θα πρέπει να υπολογίσουμε ένα άθροισμα $n-2$ φορές. Ας δούμε πώς μπορούμε να υπολογίσουμε τους 4 πρώτους όρους:

```
INTEGER :: a,b,c,d

a = 0
PRINT *, a

b = 1
PRINT *, b

c = a + b
PRINT *, c
```

```
d = b + c
PRINT *, d
```

Παρατηρούμε ότι επαναλαμβάνεται η πρόσθεση αλλά η εντολή δεν έχει ακριβώς την ίδια μορφή κάθε φορά. Ας την τροποποιήσουμε ώστε να γίνει ίδια:

```
INTEGER :: a,b,c,d, x1, x2, x3

a = 0
PRINT *, a

b = 1
PRINT *, b

x1 = a
x2 = b

x3 = x1 + x2
c = x3
PRINT *, c
x1 = b
x2 = c

x3 = x1 + x2
d = x3
PRINT *, d
x1 = c
x2 = d
```

Η εντολή `x3 = x1 + x2` πλέον επαναλαμβάνεται αυτούσια.

Παρατηρήστε τη μεταβλητή `b`: αντιγράφουμε την τιμή της σε δύο μεταβλητές, `x2` και λίγο παρακάτω, `x1`. Το ίδιο κάνουμε και στη `c`. Ας απλοποιήσουμε κάπως τον κώδικα:

```
INTEGER :: a,b,c,d, x1, x2, x3

a = 0
PRINT *, a

b = 1
PRINT *, b

x1 = a
x2 = b
```

```
x3 = x1 + x2
c = x3
PRINT *, c
x1 = x2
x2 = c

x3 = x1 + x2
d = x3
PRINT *, d
x1 = x2
x2 = d
```

Παρατηρήστε τη μεταβλητή c: αποκτά τιμή από τη μεταβλητή x3, εκτυπώνεται και μεταφέρει την τιμή της σε άλλη μεταβλητή (x2). Μπορεί να παραληφθεί τελείως (και αυτή και η d που έχει τον ίδιο ρόλο). Όπου χρειαζόμαστε την τιμή της θα την πάρουμε από άλλη μεταβλητή με την ίδια τιμή:

```
INTEGER :: a,b, x1, x2, x3

a = 0
PRINT *, a

b = 1
PRINT *, b

x1 = a
x2 = b

x3 = x1 + x2
PRINT *, x3
x1 = x2
x2 = x3

x3 = x1 + x2
PRINT *, x3
x1 = x2
x2 = x3
```

Πλέον έχουμε μια ομάδα εντολών που επαναλαμβάνεται αυτούσια. Μπορούμε να χρησιμοποιήσουμε εντολή επανάληψης:

```
INTEGER :: a,b, x1, x2, x3, k

a = 0
```

```
PRINT *, a

b = 1
PRINT *, b

x1 = a
x2 = b

DO k=1,2
   x3 = x1 + x2
   PRINT *, x3
   x1 = x2
   x2 = x3
END DO
```

Αν απαλείψουμε τις μεταβλητές a,b και θέσουμε n το πλήθος των επιθυμητών όρων, καταλήγουμε στον κώδικα

```
INTEGER :: x1, x2, x3, k

x1 = 0
PRINT *, x1

x2 = 1
PRINT *, x2

DO k=1,n-2
   x3 = x1 + x2
   PRINT *, x3
   x1 = x2
   x2 = x3
END DO
```

- Το άθροισμα

$$\sum_{k=i}^{j} a_k,$$

όπου το k είναι ακέραιος αριθμός που παίρνει τιμές μεταξύ i και j και το a_k συμβολίζει ένα σύνθετο πραγματικό όρο που εξαρτάται από το k, μπορεί να υπολογιστεί, σύμφωνα με όσα αναφέραμε παραπάνω, προσθέτοντας σε μια πραγματική μεταβλητή έναν-έναν τους όρους. Ο σχετικός κώδικας είναι (για δεδομένα ακέραια i, j)

```
INTEGER :: k
DOUBLE PRECISION :: s
```

```
s = 0.0d0
DO k=i,j
   s = s + ak
END DO
```

Στη θέση του ak γράφουμε τον κώδικα που εκφράζει τον όρο a_k.

Προσέξτε ότι η μεταβλητή s που, με την ολοκλήρωση του **DO** έχει την επιδιωκώμενη τιμή, αποκτά αμέσως πριν το **DO** την αρχική τιμή 0 (το ουδέτερο στοιχείο της πρόσθεσης).

Αντίστοιχα, το γινόμενο

$$\prod_{k=i}^{j} a_k,$$

γράφεται

```
INTEGER :: k
DOUBLE PRECISION :: p

p = 1.0d0
DO k=i,j
   p = p * ak
END DO
```

Προσέξτε ότι η μεταβλητή p, που τελικά είναι το γινόμενο που θέλουμε να υπολογίσουμε, έχει αρχική τιμή το 1 (το ουδέτερο στοιχείο του πολλαπλασιασμού).

4.2.3 Υπονοούμενο DO

Το *υπονοούμενο DO* είναι δομή επανάληψης που παράγει μια ακολουθία αριθμών. Η χρήση της γίνεται όπως στο ακόλουθο παράδειγμα:

Έστω ότι θέλουμε να εκτυπώσουμε στην οθόνη, σε μία γραμμή, τους αριθμούς $1, 2, 3, 4, 5$. Προφανώς αυτό μπορεί να γίνει με την εντολή

```
WRITE (*,*) 1,2,3,4,5
```

Εναλλακτικά, μπορούμε να γράψουμε

```
WRITE (*,*) (i, i=1,5)
```

Το υπονοούμενο **DO** περιλαμβάνεται σε παρενθέσεις και αποτελείται από μία έκφραση, ακολουθούμενη από (γενικά) τριάδα των τιμών για τον καθορισμό μιας ακέραιας μεταβλητής. Η γενικευμένη σύνταξή του είναι η ακόλουθη

(έκφραση, μεταβλητή = αρχική τιμή, τελική τιμή, βήμα αύξησης)

Η ακέραια μεταβλητή «μεταβλητή» παίρνει τιμές που καθορίζονται από την τριάδα αριθμών που ακολουθούν το '=', με τους ίδιους κανόνες που ισχύουν για την εντολή **DO**. Αν παραλείπεται το «βήμα αύξησης» εννοείται το 1. Κάθε φορά που αλλάζει τιμή η «μεταβλητή», υπολογίζεται η «έκφραση». Έτσι, το υπονοούμενο **DO** αντικαθίσταται με μια σειρά τιμών.

Παράδειγμα
Η εκτύπωση των αριθμών $1, 3, 5, 7, 9$ *στην ίδια γραμμή* μπορεί να γίνει με την εντολή

```
WRITE (*,*) (i, i=1,19,2)
```

ή με την εντολή

```
WRITE (*,*) (2*i+1, i=0,9)
```

Ιδιαίτερη χρησιμότητα του υπονοούμενου **DO** εμφανίζεται κατά την εκχώρηση τιμών, ανάγνωση και εκτύπωση πινάκων, στο Κεφάλαιο 5.

4.3 DO για απροσδιόριστο αριθμό επαναλήψεων

Υπάρχει συχνά η ανάγκη να επαναλάβουμε ένα τμήμα εντολών χωρίς να γνωρίζουμε εκ των προτέρων το πλήθος των επαναλήψεων. Π.χ. κατά την είσοδο δεδομένων από το χρήστη, οι τιμές που εισάγονται μπορεί να απαιτείται να ικανοποιούν κάποια συνθήκη. Σε τέτοια περίπτωση χρειάζεται να επαναλάβουμε τις εντολές ανάγνωσης και ελέγχου των δεδομένων, για όσο τα δεδομένα είναι κατάλληλα. Δεν γνωρίζουμε όμως πόσες φορές θα χρειαστεί για να δώσει ο χρήστης αποδεκτές τιμές. Τέτοιες καταστάσεις αντιμετωπίζονται από τη Fortran 95 με την εντολή **DO** *χωρίς μεταβλητή ελέγχου*. Η σύνταξή της είναι απλή:

```
DO
....
....
END DO
```

Με τη συγκεκριμένη μορφή του **DO**, οι εντολές που περικλείονται στα **DO** ...**END DO** εκτελούνται *για πάντα*. Προφανώς, πρέπει με κάποιον τρόπο να διακόπτουμε την επανάληψη όταν επιθυμούμε. Η Fortran 95 παρέχει, μεταξύ άλλων, την εντολή **EXIT** (§4.4.1) που κάνει ακριβώς αυτό.

4.4 Βοηθητικές εντολές

4.4.1 EXIT

Η εντολή **EXIT** μπορεί να εμφανιστεί μόνο μέσα σε **DO**...**END DO** με ή χωρίς μεταβλητή ελέγχου. Όταν εκτελεστεί, προκαλεί την έξοδο από το βρόχο που την περικλείει. Η επόμενη εντολή που εκτελείται είναι αυτή που ακολουθεί το **END DO**.

Παράδειγμα χρήσης: αν επιθυμούμε να διαβάσουμε έναν πραγματικό αριθμό και να προχωρήσουμε στην εκτέλεση μόνο αν είναι θετικός, μπορούμε να γράψουμε

```
DOUBLE PRECISION :: x
DO
  READ (*,*) x
  IF (x > 0.0d0) EXIT
END DO
```

Όσο ο χρήστης δίνει μη θετικούς αριθμούς, οι εντολές θα επαναλαμβάνονται.

4.4.2 CYCLE

Η εντολή **CYCLE** μπορεί να εμφανιστεί μόνο μέσα σε **DO**...**END DO**. Όταν εκτελεστεί, προκαλεί τη μεταφορά της ροής του προγράμματος αμέσως μετά την τελευταία εντολή στο σώμα του **DO** που την περικλείει· παραλείπει, επομένως, τις εντολές που την ακολουθούν. Το αμέσως επόμενο βήμα της διαδικασίας επανάληψης που εκτελείται είναι η μεταβολή της ακέραιας μεταβλητής ελέγχου του **DO** κατά το βήμα αύξησης. Στην περίπτωση του **DO** χωρίς μεταβλητή ελέγχου, §4.3, ουσιαστικά ξαναρχίζει την επανάληψη.

Παράδειγμα χρήσης είναι το ακόλουθο: Έστω ότι θέλουμε να τυπώσουμε τις τετραγωνικές ρίζες των πρώτων 10 αριθμών εισόδου, αγνοώντας όμως τους αρνητικούς. Ο σχετικός κώδικας μπορεί να είναι

```
INTEGER :: i
DOUBLE PRECISION :: x
DO i=1,10
  READ (*,*) x
  IF (x < 0.0d0) CYCLE  ! απορρίπτουμε το x

  WRITE (*,*) "H tetragwnikh riza einai", SQRT(x)
END DO
```

4.4.3 Όνομα βρόχου DO

Έχουμε τη δυνατότητα να αποδώσουμε ένα όνομα σε κάθε βρόχο **DO**. Αυτό γίνεται δίνοντας το όνομα (ακολουθούμενο από ':') πριν το **DO**, στην ίδια γραμμή, και συμπληρώνοντας υποχρεωτικά το αντίστοιχο **END DO** με το ίδιο όνομα, όπως στο παρακάτω παράδειγμα:

```
onoma: DO i=1, 10
          ....
       END DO onoma
```

Το όνομα σχηματίζεται με τους κανόνες που είδαμε για τις μεταβλητές, §2.1.2.

Η ονοματοδοσία στο **DO** είναι χρήσιμη στις εντολές **EXIT** και **CYCLE**. Αν τις χρησιμοποιήσουμε με τον τρόπο που περιγράψαμε στις αντίστοιχες παραγράφους, εκτελούν τη λειτουργία τους για το πρώτο βρόχο που τις περικλείει, τον πιο «κοντινό». Αν, όμως, προσδιορίσουμε και το όνομα ενός βρόχου, δηλαδή,

```
CYCLE onoma
```

ή

```
EXIT onoma
```

τότε οι εντολές εκτελούνται για το βρόχο με το όνομα onoma. Παραδείγματος χάριν, ας εξετάσουμε την περίπτωση στην οποία έχουμε βρόχους που περικλείουν ο ένας τον άλλο. Έστω ότι επιθυμούμε να διακόψουμε τον «εξωτερικό» βρόχο με εντολή **EXIT** που βρίσκεται σε «εσωτερικό» βρόχο. Αυτό γίνεται ως εξής

```
outer: DO i=1,10
          DO j=1,10
            DO k=1,20
              ...
              ...
              IF (k > i+j) EXIT outer
            END DO
          END DO
       END DO outer
```

Στο παραπάνω παράδειγμα, η εντολή που θα εκτελεστεί μετά το **EXIT** (όταν αυτό εκτελεστεί) θα είναι η πρώτη εντολή μετά το **END DO** outer.

4.5 Ασκήσεις

1. Γράψτε πρόγραμμα που να τυπώνει στην οθόνη τους αριθμούς 0.0, 0.1, 0.2, 0.3, 0.4, ..., 2.5.

2. Τυπώστε τις πρώτες 20 δυνάμεις του 2 (2^0, 2^1, ...,2^{19}).

3. Τυπώστε το παραγοντικό αριθμού που θα δίνει ο χρήστης. Το $n!$ ορίζεται ως εξής:
$$n! = \begin{cases} 1 \times 2 \times 3 \cdots \times (n-1) \times n\ , & n > 0\ , \\ 1\ , & n = 0\ . \end{cases}$$
Φροντίστε ώστε το πρόγραμμά σας να μη δέχεται n μεγαλύτερο από 12.

4. Γράψτε πρόγραμμα που να υπολογίζει και να τυπώνει το διπλό παραγοντικό αριθμού που θα δίνει ο χρήστης. Το διπλό παραγοντικό ($n!!$) ορίζεται ως
$$n!! = \begin{cases} 1 \times 3 \times 5 \times \cdots \times (n-2) \times n & \text{αν το } n \text{ είναι περιττός,} \\ 2 \times 4 \times 6 \times \cdots \times (n-2) \times n & \text{αν το } n \text{ είναι άρτιος,} \\ 1 & \text{αν το } n \text{ είναι } 0. \end{cases}$$
Φροντίστε ώστε το πρόγραμμά σας να μη δέχεται n μεγαλύτερο από 19.

5. Τυπώστε στην οθόνη 53 ισαπέχοντα σημεία στο διάστημα $[-3.5, 6.5]$ (συμπεριλαμβανόμενων και των άκρων).

   Υπόδειξη: n ισαπέχοντα σημεία στο διάστημα $[a, b]$ έχουν άγνωστη απόσταση μεταξύ τους, έστω h. Πρέπει να ισχύει $x_1 = a$, $x_2 = a + h$, $x_3 = a + 2h$, ..., $x_n = a + (n-1)h$. Αλλά πρέπει ακόμα να ισχύει $x_n \equiv b$, άρα $a + (n-1)h = b \Rightarrow h = (b-a)/(n-1)$.

6. Γράψτε κώδικα που να τυπώνει στην οθόνη 30 τιμές της μαθηματικής συνάρτησης $f(x) = x(x^2 + 5\sin(x))$ σε ισαπέχοντα σημεία στο διάστημα $[-5, 5]$. Τα άκρα να περιλαμβάνονται σε αυτά.

7. Κάποιος καταθέτει 1000 ευρώ σε ένα απλό ταπεζικό λογαριασμό. Η τράπεζα δίνει τόκο που παραμένει στο λογαριασμό, με ετήσιο επιτόκιο 0.5%. Γράψτε πρόγραμμα που να υπολογίζει πόσα χρήματα θα υπάρχουν στο λογαριασμό αυτό μετά από 15 χρόνια.

   Απάντηση: 1077.68€

8. Σύμφωνα με την Εθνική Στατιστική Αρχή, ο πληθυσμός της Ελλάδας κατά την απογραφή του 2011 ήταν 10816286. Εάν αυξάνεται σταθερά κατά 0.53% το χρόνο, γράψτε πρόγραμμα που να υπολογίζει σε πόσα χρόνια θα ξεπεράσει τα 15000000.

   Απάντηση: 62 έτη

9. Βρείτε το μέγιστο κοινό διαιρέτη (ΜΚΔ) δύο ακέραιων αριθμών a, b. Χρησιμοποιήστε τον αλγόριθμο του Ευκλείδη[1]. Σύμφωνα με αυτόν, για δύο μη αρνητικούς ακέραιους αριθμούς a και b:

   - αν ισχύει $a < b$ εναλλάσσουμε τις τιμές τους.
   - αν ο b είναι 0 τότε ο a είναι ο ΜΚΔ.
   - αν ο b είναι θετικός, επαναλαμβάνουμε τη διαδικασία χρησιμοποιώντας ως νέους ακέραιους τον b και το υπόλοιπο της διαίρεσης του a με τον b.

   Χρησιμοποιήστε τον αλγόριθμο για να βρείτε το μέγιστο κοινό διαιρέτη των αριθμών 135 και 680.

   Απάντηση: 5

10. Γράψτε πρόγραμμα που να ελέγχει αν ένας ακέραιος αριθμός είναι τέλειος. Τέλειος είναι ο αριθμός, του οποίου το άθροισμα των διαιρετών του είναι ίσο με το διπλάσιο του. (Π.χ. το 6 είναι τέλειος αριθμός, γιατί διαιρείται ακριβώς με τους αριθμούς 1, 2, 3, 6 και το άθροισμά τους είναι το $1+2+3+6 = 12 = 2\times 6$).

11. Γράψτε πρόγραμμα που να δέχεται ένα θετικό ακέραιο αριθμό με οποιοδήποτε πλήθος ψηφίων και να εμφανίζει τα ψηφία του.

12. Πόσοι είναι οι θετικοί ακέραιοι αριθμοί με το πολύ 4 ψηφία, που έχουν την ιδιότητα το τετράγωνό τους να τελειώνει σε 444;

    Απάντηση: 40

13. Ποιος είναι ο μικρότερος ακέραιος που το τετράγωνό του έχει το 3 στο ψηφίο των εκατοντάδων;

    Απάντηση: 18

14. Πόσοι είναι οι θετικοί ακέραιοι με το πολύ 3 ψηφία, το τετράγωνο των οποίων έχει το 3 στο ψηφίο των εκατοντάδων;

    Απάντηση: 72

15. Γράψτε πρόγραμμα που να τυπώνει τους πρώτους N όρους της ακολουθίας Fibonacci[2]. Ο i-όρος της ακολουθίας, f_i, δίνεται από τις σχέσεις

$$f_i = \begin{cases} 0\,, & i = 0 \\ 1\,, & i = 1 \\ f_{i-1} + f_{i-2}\,, & i > 1 \end{cases}.$$

    Η ακολουθία επομένως είναι: $0, 1, 1, 2, 3, 5, 8, \ldots$. Το κάθε στοιχείο μετά το δεύτερο είναι το άθροισμα των δύο προηγούμενών του. Το πλήθος n να δίνεται από το χρήστη. Να φροντίσετε ώστε το πρόγραμμά σας να μην το δέχεται αν δεν ισχύει $0 \leq n < 31$.

[1] http://en.wikipedia.org/wiki/Euclidean_algorithm
[2] http://oeis.org/A000045

16. Γράψτε πρόγραμμα που να ελέγχει αν ένας ακέραιος αριθμός που θα τον δίνει ο χρήστης, είναι πρώτος.

    Υπόδειξη I: θα σας βοηθήσει ο ακόλουθος ορισμός για τους πρώτους αριθμούς:

    > Κάθε θετικός ακέραιος αριθμός είναι πρώτος εκτός αν διαιρείται ακριβώς με κάποιο αριθμό εκτός από το 1 και τον εαυτό του.

    Υπόδειξη II: Ψάξτε να βρείτε κάποιον θετικό ακέραιο που να διαιρεί ακριβώς (δηλαδή χωρίς υπόλοιπο) τον αριθμό εισόδου, εκτός από το 1 και τον εαυτό του. Μπορούμε να αποκλείσουμε όλους τους μεγαλύτερους του αριθμού εισόδου καθώς κανένας δεν θα τον διαιρεί ακριβώς.

17. Γράψτε πρόγραμμα που να ζητά από το χρήστη μια σειρά από θετικούς πραγματικούς αριθμούς. Το πλήθος τους δεν θα είναι γνωστό εκ των προτέρων αλλά το «διάβασμα» των τιμών θα σταματά όταν ο χρήστης δώσει αρνητικό αριθμό. Το πρόγραμμά σας να υπολογίζει το μέσο όρο αυτών των αριθμών.

18. Χρησιμοποιήστε τον αλγόριθμο για την ημερομηνία του ορθόδοξου Πάσχα από την άσκηση 14 στη σελίδα 29 για να βρείτε

    (α΄) ποια χρονιά το Πάσχα έπεσε πιο νωρίς,

    (β΄) ποια χρονιά έπεσε πιο αργά,

    (γ΄) πόσες φορές έπεσε το Μάιο,

    (δ΄) ποιες χρονιές έπεφτε στις 18 Απριλίου,

    μεταξύ των ετών 1930 έως και πέρυσι.

19. Χρησιμοποιήστε τις πληροφορίες της άσκησης 9 στη σελίδα 39 για να υπολογίσετε πόσες ημέρες έχουν περάσει από την ημερομηνία γέννησής σας (δηλαδή, την ηλικία σας σε ημέρες).

    Υπόδειξη: Μετακινήστε την αρχική ημερομηνία κατά μία ημέρα μπροστά πολλές φορές μέχρι να βρείτε την τελική ημερομηνία. Μετρήστε πόσες ημέρες πέρασαν.

20. Ο Μιχάλης γεννήθηκε στις 8/2/2013. Χρησιμοποιήστε τις πληροφορίες της άσκησης 9 της σελίδας 39 για να βρείτε την ημερομηνία που θα συμπληρώσει 17000 ημέρες ζωής.

    Υπόδειξη: Μετακινήστε την αρχική ημερομηνία κατά μία ημέρα μπροστά πολλές φορές μέχρι να εξαντλήσετε τις διαθέσιμες ημέρες.

    Απάντηση: 26/8/2059

21. Ο Γιάννης συμπλήρωσε 13000 ημέρες ζωής στις 2/7/2015. Πότε γεννήθηκε; Τι ημέρα ήταν; Θα σας χρειαστούν οι πληροφορίες της άσκησης 9 στη σελίδα 39 και ο αλγόριθμος Zeller της άσκησης 8 στη σελίδα 39.

Υπόδειξη: Μετακινήστε την αρχική ημερομηνία κατά μία ημέρα πίσω πολλές φορές μέχρι να συγκεντρώσετε το επιθυμητό πλήθος ημερών.

Απάντηση: Τετάρτη 28/11/1979

22. Το λειτουργικό σύστημα UNIX υπολογίζει το χρόνο με βάση τον αριθμό των δευτερολέπτων που πέρασαν από την 1/1/1970, στις 00:00:00. Ποια ημέρα και ώρα συμπληρώθηκαν 10^9 δευτερόλεπτα από τότε; Πότε θα συμπληρωθούν $2^{31} - 1$ δευτερόλεπτα (και πλέον θα πάψουν να τηρούν σωστά το χρόνο τα συστήματα UNIX που αποθηκεύουν το χρόνο σε εμπρόσημο ακέραιο 32bit);

    Θα σας χρειαστούν οι πληροφορίες της άσκησης 9 στη σελίδα 39. Αγνοήστε τα εμβόλιμα δευτερόλεπτα (leap seconds) που εισάγονται κατά καιρούς για τη διόρθωση της ώρας.

    Απάντηση: 9/9/2001 01:46:40, 19/1/2038 03:14:07

23. Χρησιμοποιήστε τον αλγόριθμο Zeller της άσκησης 8 στη σελίδα 39 για να βρείτε πόσες Πρωτοχρονιές από το 1950 έως φέτος έπεφταν Σαββατοκύριακο.

24. Γράψτε κώδικες που να υπολογίζουν τα e^x, $\sin x$, $\cos x$ από τις σχέσεις

$$\mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}\,, \qquad \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}\,, \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}\,.$$

    Για τη διευκόλυνσή σας παρατηρήστε ότι ο κάθε όρος στα αθροίσματα προκύπτει από τον αμέσως προηγούμενο αν αυτός πολλαπλασιαστεί με κατάλληλη ποσότητα.

    Στα αθροίσματα να σταματάτε τον υπολογισμό τους όταν ο όρος που πρόκειται να προστεθεί είναι κατ' απόλυτη τιμή μικρότερος από 10^{-10}.

25. Σύμφωνα με θεώρημα του Gauss, για κάθε θετικό ακέραιο a ισχύει ότι

$$2a = n(n+1) + m(m+1) + k(k+1)$$

    όπου n, m, k μη αρνητικοί και όχι απαραίτητα διαφορετικοί ακέραιοι. Να γράψετε ένα πρόγραμμα που να διαβάζει από τον χρήστη ένα ακέραιο a, να υπολογίζει όλες τις τριάδες n, m, k για αυτόν και να τις τυπώνει στην οθόνη. Οι τριάδες που προκύπτουν με εναλλαγή των n, m, k να παραλείπονται (δηλαδή τυπώστε αυτές για τις οποίες ισχύει $n \le m \le k$).

    Δοκιμάστε το για τους αριθμούς 16 ($[0, 1, 5]$ ή $[0, 3, 4]$ ή $[2, 2, 4]$), 104 ($[2, 4, 13]$ ή ...), και 111 ($[1, 10, 10]$ ή $[0, 9, 11]$ ή ...).

26. Γράψτε πρόγραμμα που να επαληθεύει το Θεώρημα των τεσσάρων τετραγώνων του Lagrange[3]. Σύμφωνα με αυτό, κάθε θετικός ακέραιος αριθμός μπορεί

[3] http://mathworld.wolfram.com/LagrangesFour-SquareTheorem.html

να γραφεί ως άθροισμα τεσσάρων (ή λιγότερων) τετραγώνων ακεραίων αριθμών.

Υπόδειξη: Δημιουργήστε τέσσερις βρόχους, ο ένας μέσα στον άλλο. Όταν το άθροισμα των τετραγώνων των μεταβλητών ελέγχου γίνει ίσο με το ζητούμενο αριθμό, τυπώστε τις μεταβλητές ελέγχου και συνεχίστε για την επόμενη τετράδα. Λάβετε υπόψη ότι κάποιες από τις μεταβλητές ελέγχου μπορεί να είναι 0 ή να είναι ίσες. Επιλέξτε κατάλληλα τα διαστήματα στα οποία αυτές παίρνουν τιμές.

27. Από τα Μαθηματικά γνωρίζουμε ότι

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{4n^2}{4n^2 - 1} \ .$$

Υπολογίστε το δεξί μέλος της εξίσωσης χωρίς φυσικά να πάρετε άπειρους όρους. Κρατήστε 10^4 όρους. Βρείτε πόσο διαφέρει το αποτέλεσμα από το $\pi/2$.

28. Να επαληθεύσετε ότι

$$\sum_{n=1}^{\infty} \left[\frac{12}{n^2} \cos\left(\frac{9}{n\pi + \sqrt{(n\pi+3)(n\pi-3)}} \right) \right] = -\frac{\pi^2}{e^3}$$

ως εξής: υπολογίσετε τα δύο μέλη της εξίσωσης χωριστά και τυπώστε αυτά καθώς και τη διαφορά τους (που θα πρέπει να πλησιάζει στο 0). Στον υπολογισμό του αθροίσματος δε θα πάρετε φυσικά άπειρους όρους· να σταματήσετε στον πρώτο όρο που είναι κατ᾽ απόλυτη τιμή μικρότερος από 10^{-7}.

29. Να επαληθεύσετε ότι

$$\ln(5/4) = \sum_{k=1}^{\infty} \frac{1}{k\, 5^k}$$

ως εξής: υπολογίστε τα δύο μέλη της εξίσωσης και βρείτε τη διαφορά τους (η οποία πρέπει να είναι πολύ «μικρή»).

Υπόδειξη: Στο άθροισμα δεν μπορούμε, φυσικά, να πάρουμε άπειρους όρους. Να σταματήσετε τον υπολογισμό του στον πρώτο όρο με τιμή μικρότερη από 10^{-11}.

30. Να υπολογίσετε το π από τη σχέση

$$\pi = 3 \sum_{k=0}^{\infty} \frac{(-1)^k}{(k + 1/2) 3^{k+1/2}} \ .$$

Στον υπολογισμό του αθροίσματος δεν μπορούμε να πάρουμε άπειρους όρους. Να σταματήσετε στον πρώτο όρο που έχει απόλυτη τιμή μικρότερη από 10^{-8}. Ποια τιμή βρίσκετε και πόσους όρους χρησιμοποιήσατε στον υπολογισμό της;

31. Από τα Μαθηματικά γνωρίζουμε ότι

$$\frac{\pi}{4} = \lim_{n\to\infty} f_n \ ,$$

όπου

$$f_n = \frac{1}{n}\sum_{k=1}^{n}\sqrt{1-\left(\frac{k}{n}\right)^2} \ .$$

Αυτό σημαίνει ότι για μεγάλες τιμές του n το f_n τείνει στο $\pi/4$. Υπολογίστε το f_n για $n = 10^6$ και βρείτε πόσο διαφέρει από το $\pi/4$.

32. Ο *δυαδικός αλγόριθμος* για τον πολλαπλασιασμό δύο ακεραίων έχει ως εξής: σχηματίζουμε δύο στήλες με επικεφαλής τους δύο αριθμούς. Κάθε αριθμός της *πρώτης* στήλης είναι το ακέραιο μέρος (πηλίκο) της διαίρεσης με το 2 του αμέσως προηγούμενού του στη στήλη. Κάθε αριθμός της *δεύτερης* στήλης είναι το διπλάσιο του αμέσως προηγούμενού του στη στήλη. Οι διαιρέσεις/πολλαπλασιασμοί στις στήλες σταματούν όταν στην πρώτη εμφανιστεί ο αριθμός 0. Το γινόμενο των δύο αρχικών αριθμών είναι το άθροισμα των αριθμών της δεύτερης στήλης που αντιστοιχούν σε περιττό αριθμό στην πρώτη στήλη.

Γράψτε κώδικα που θα δέχεται από το χρήστη δύο ακέραιους, θα υπολογίζει το γινόμενό τους με το συγκεκριμένο αλγόριθμο και θα το τυπώνει.

33. Γράψτε πρόγραμμα που να βρίσκει τις τριάδες διαδοχικών πρώτων αριθμών που διαφέρουν κατά έξι[4] (δηλαδή οι p, $p+6$, $p+12$ να είναι διαδοχικοί πρώτοι αριθμοί) και να τυπώνει το μικρότερο. Περιοριστείτε στους πρώτους που είναι μικρότεροι από 10000.

34. Γράψτε κώδικα που να παράγει 120 τυχαίους ακέραιους αριθμούς στο διάστημα $[-100, 100)$. Μετρήστε πόσοι από αυτούς είναι θετικοί και πόσοι αρνητικοί.

Υπόδειξη: Για την παραγωγή τυχαίων αριθμών χρησιμοποιήστε την υπορουτίνα `RANDOM_NUMBER()` (§2.9.1).

35. Από τα Μαθηματικά γνωρίζουμε ότι ισχύει

$$\pi = 3 + 2\sum_{k=1}^{\infty}\frac{k(5k+3)(2k-1)!k!}{2^{k-1}(3k+2)!} \ .$$

Χρησιμοποιήστε την παραπάνω σχέση για να υπολογίσετε το π με ακρίβεια 10^{-6}· αυτό σημαίνει ότι στον υπολογισμό του αθροίσματος θα σταματήσετε στον πρώτο όρο που είναι μικρότερος από 10^{-6}. Συγκρίνετε το αποτέλεσμά σας με τη «σωστή» τιμή.

Υπόδειξη: Στον υπολογισμό σας μπορείτε να βασιστείτε στο ότι ο κάθε όρος στο άθροισμα προκύπτει από τον προηγούμενο με πολλαπλασιασμό κατάλληλης ποσότητας.

[4] http://oeis.org/A047948

36. Ο θετικός ακέραιος 65728 μπορεί να γραφεί ως άθροισμα δύο κύβων (ακέραιων υψωμένων στην τρίτη) με μόνο δύο τρόπους:

$$65728 = 12^3 + 40^3 = 31^3 + 33^3 \ .$$

Το ίδιο ισχύει και για τον 64232:

$$64232 = 17^3 + 39^3 = 26^3 + 36^3 \ .$$

Βρείτε[5] το μικρότερο k που ικανοποιεί τη σχέση $k = i^3 + j^3$ με δύο και μόνο δύο διαφορετικά ζευγάρια i, j. Τα i, j, k είναι θετικοί ακέραιοι με $i \leq j < k$.

Απάντηση: $1729 = 1^3 + 12^3 = 9^3 + 10^3$

37. Πολλοί περιττοί ακέραιοι αριθμοί μπορούν να γραφτούν ως άθροισμα ενός πρώτου αριθμού (ή του 1) και του διπλάσιου κάποιου τετραγώνου μη μηδενικού αριθμού:

$$\begin{aligned} 3 &= 1 + 2 \times 1^2 \\ 5 &= 3 + 2 \times 1^2 \\ 9 &= 1 + 2 \times 2^2 = 7 + 2 \times 1^2 \\ 15 &= 7 + 2 \times 2^2 \\ 27 &= 19 + 2 \times 2^2 \\ \vdots &= \vdots \end{aligned}$$

Βρείτε τους πρώτους πέντε θετικούς περιττούς αριθμούς που ΔΕΝ είναι ίσοι με ένα τέτοιο άθροισμα.

Απάντηση: 17, 137, 227, 977, 1187

38. Ένας ακέραιος αριθμός με n ψηφία χαρακτηρίζεται ως παμψήφιος αν περιέχει όλα τα ψηφία από το 1 ως το n ακριβώς μία φορά. Π.χ. το 3142 είναι παμψήφιος τεσσάρων ψηφίων. Βρείτε τον μεγαλύτερο παμψήφιο τεσσάρων ψηφίων που είναι πρώτος.

Απάντηση: 4231

39. Αριθμός Goldbach λέγεται ένας άρτιος θετικός ακέραιος που μπορεί να γραφεί ως άθροισμα δύο περιττών αριθμών που είναι πρώτοι. Σύμφωνα με την υπόθεση του Goldbach, κάθε άρτιος αριθμός μεγαλύτερος του 4 είναι τέτοιος αριθμός. Δείξτε ότι ισχύει για τους άρτιους ακέραιους στο διάστημα $[6, 10000]$.

40. Ο ανάστροφος ενός θετικού ακέραιου είναι ένας άλλος αριθμός με τα ίδια ψηφία σε ανάστροφη σειρά. Π.χ. ο ανάστροφος του 529 είναι ο 925, ο ανάστροφος του 910 είναι ο 19.

[5] https://en.wikipedia.org/wiki/Taxicab_number

Κάποιοι θετικοί ακέραιοι αριθμοί έχουν την εξής ιδιότητα: το άθροισμα του αριθμού και του ανάστροφού του είναι αριθμός που τα ψηφία του είναι περιττοί αριθμοί. Π.χ. $409 + 904 = 1313$. Ας ονομάσουμε τους αριθμούς με αυτή την ιδιότητα *αναστρέψιμους*.

Γράψτε κώδικα που να τυπώνει στην οθόνη όλους τους αναστρέψιμους αριθμούς μέχρι το 100.

41. Έστω $f(x)$ μια συνάρτηση που είναι μη αρνητική στο διάστημα $[a, b]$. Ένας τρόπος για να υπολογίσουμε το ολοκλήρωμα

$$\int_a^b f(x)\,\mathrm{d}x\ ,$$

δηλαδή, το εμβαδόν κάτω από την καμπύλη της $f(x)$, είναι ο εξής: επιλέγουμε ένα μεγάλο αριθμό από τυχαία σημεία (x_i, y_i) (δηλαδή τυχαία x_i και y_i), ομοιόμορφα κατανεμημένα στο παραλληλόγραμμο που αποτελείται από τα σημεία (x, y) με $a \le x \le b$, $0 \le y \le \max\{f(x)\}$. Το $\max\{f(x)\}$ είναι η μέγιστη τιμή της $f(x)$ στο $[a, b]$. Μετρούμε όσα σημεία είναι κάτω από την καμπύλη $y = f(x)$ ή ακριβώς πάνω σε αυτή (δηλαδή αυτά για τα οποία ισχύει $y_i \le f(x_i)$). Το πλήθος αυτών προς το συνολικό αριθμό των σημείων είναι προσεγγιστικά ο λόγος του συγκεκριμένου ολοκληρώματος προς το εμβαδόν του παραλληλόγραμμου.

Γράψτε ένα πρόγραμμα που θα υπολογίζει με αυτό τον τρόπο το ολοκλήρωμα

$$\int_0^2 x^2\,\mathrm{d}x\ .$$

Υπόδειξη: Για την παραγωγή τυχαίων αριθμών χρησιμοποιήστε την υπορουτίνα `RANDOM_NUMBER( )` με κατάλληλο γραμμικό μετασχηματισμό (§2.9.1). Αγνοήστε το γεγονός ότι δεν μπορεί να προκύψει ως τυχαία τιμή το άνω όριο του διαστήματος.

42. Τα κέρματα του ευρώ έχουν αξία 1 λεπτό, 2 λεπτά, 5 λεπτά, 10 λεπτά, 20 λεπτά, 50 λεπτά, 100 λεπτά (= 1€) και 200 λεπτά (= 2€).

Ένα συγκεκριμένο ποσό μπορεί να σχηματιστεί με συνδυασμό διάφορων κερμάτων. Πόσοι είναι όλοι οι συνδυασμοί που έχουν αξία 300 λεπτών;

Υπόδειξη: Προφανώς, κάθε συνδυασμός θα έχει το πολύ 300 κέρματα του ενός λεπτού, 150 κέρματα των δύο λεπτών, 60 κέρματα των 5 λεπτών κλπ. Σχηματίστε όλους τους δυνατούς συνδυασμούς και μετρήστε όσους έχουν αξία 300 λεπτών.

Απάντηση: 471363

43. Πρώτος λέγεται ένας θετικός ακέραιος, μεγαλύτερος από 1, που διαιρείται ακριβώς μόνο από το 1 και τον εαυτό του: τέτοιοι είναι οι $3, 5, 7, 11, 13, \ldots$.

Δίδυμοι πρώτοι αριθμοί είναι τα ζεύγη των πρώτων αριθμών που διαφέρουν κατά 2: τέτοιοι είναι οι $(3,5),(5,7),(11,13),\ldots$.

Βρείτε το άθροισμα των αντίστροφων των δίδυμων πρώτων αριθμών:

$$\sum_p \left(\frac{1}{p}+\frac{1}{p+2}\right) = \left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\left(\frac{1}{11}+\frac{1}{13}\right)+\cdots .$$

Το p στο άθροισμα είναι το πρώτο μέλος κάθε ζεύγους δίδυμων πρώτων.

Το άθροισμα αυτό έχει πεπερασμένη τιμή, τη *σταθερά Brun* (≈ 1.902), αν υπολογιστούν όλα τα ζεύγη δίδυμων πρώτων. Εσείς, στον υπολογισμό του αθροίσματος λάβετε υπόψη μόνο τους δίδυμους πρώτους που είναι μικρότεροι από 100000.

44. Βρείτε το μοναδικό θετικό ακέραιο που το τετράγωνό του είναι δεκαψήφιος αριθμός της μορφής 1_2_3_4_5_ . Το _ συμβολίζει απλό ψηφίο (πιθανώς διαφορετικό σε κάθε θέση).

    Απάντηση: 34934

45. Τρεις θετικοί ακέραιοι αριθμοί a, b, c που ικανοποιούν τη σχέση $a^2+b^2=c^2$ αποτελούν μία Πυθαγόρεια τριάδα. Υπάρχει μία τέτοια τριάδα (με $a>b$) για την οποία ισχύει $a+b+c=1000$. Βρείτε τη.

    Υπόδειξη: Μπορούμε να παραγάγουμε μια Πυθαγόρεια τριάδα από δύο οποιουσδήποτε ακέραιους m, n, με $m>n$, σχηματίζοντας τους αριθμούς $a=m^2-n^2$, $b=2mn$, $c=m^2+n^2$.

    Απάντηση: $(375,200,425)$

46. Να γράψετε πρόγραμμα που να επαληθεύει τον τύπο

$$\frac{\pi}{24} = 3\sum_{n=1}^{\infty}\frac{1}{n(\mathrm{e}^{n\pi}-1)} - 4\sum_{n=1}^{\infty}\frac{1}{n(\mathrm{e}^{2n\pi}-1)} + \sum_{n=1}^{\infty}\frac{1}{n(\mathrm{e}^{4n\pi}-1)} .$$

    Να δείξετε, δηλαδή, ότι το δεξί μέλος έχει τιμή πολύ κοντά σε αυτή του αριστερού μέλους. Να σταματήσετε την πρόσθεση όρων σε κάθε άθροισμα μόλις συναντήσετε τον πρώτο που είναι μικρότερος κατ' απόλυτη τιμή από 10^{-12}.

47. Ένας θετικός ακέραιος αριθμός με n ψηφία που είναι ίσος με το άθροισμα των δυνάμεων τάξης n των ψηφίων του, λέγεται αριθμός Armstrong. Π.χ. ο αριθμός 6 είναι τέτοιος καθώς $6=6^1$. Ο αριθμός 371 είναι επίσης αριθμός Armstrong αφού $371=3^3+7^3+1^3$.

    Γράψτε πρόγραμμα που να τυπώνει όλους τους αριθμούς Armstrong μέχρι το 10000 στην οθόνη.

48. Ένας θετικός ακέραιος αριθμός που είναι ίσος με το άθροισμα των παραγοντικών των ψηφίων του, λέγεται *ισχυρός αριθμός*. Π.χ. ο αριθμός 145 είναι τέτοιος καθώς $145 = 1! + 4! + 5!$.

    Γράψτε πρόγραμμα που να τυπώνει στην οθόνη όλους τους ισχυρούς αριθμούς μέχρι το 100000.

49. Βρείτε και τυπώστε στην οθόνη ποιοι μήνες των ετών 2000–2020 είχαν 5 Κυριακές.

    Υπόδειξη: Εύκολα προκύπτει ότι ένας μήνας έχει 5 Κυριακές αν

    - έχει 29 ημέρες και η πρώτη του μηνός είναι Κυριακή.
    - έχει 30 ημέρες και η πρώτη ή η δεύτερη του μηνός είναι Κυριακή.
    - έχει 31 ημέρες και η πρώτη ή η δεύτερη ή η τρίτη του μηνός είναι Κυριακή.

    Θα σας χρειαστεί ο αλγόριθμος του Zeller (άσκηση 8 στη σελίδα 39) και ο αλγόριθμος της άσκησης 9 στη σελίδα 39).

50. Μέσα σε ένα δοχείο έχουμε 150 άσπρες και 75 μαύρες μπίλιες. Τραβούμε τυχαία ζεύγη μπιλιών από το δοχείο.

    - Αν οι δύο μπίλιες είναι άσπρες κρατούμε τη μία έξω και επιστρέφουμε την άλλη στο δοχείο.
    - Αν οι δύο μπίλιες είναι μαύρες τις κρατούμε και τις δύο έξω από το δοχείο αλλά προσθέτουμε μία άσπρη μπίλια στο δοχείο.
    - Αν είναι μία άσπρη και μία μαύρη κρατούμε την άσπρη έξω και επιστρέφουμε τη μαύρη στο δοχείο.

    Η διαδικασία αυτή συνεχίζεται μέχρι που απομένει μόνο μία μπίλια στο δοχείο. Τι χρώμα έχει αυτή η μπίλια;

    Απάντηση: Μαύρη

    Υπόδειξη: Έστω ότι σε κάθε επιλογή μπιλιών, W είναι το πλήθος των άσπρων και B το πλήθος των μαύρων μπιλιών. Επιλέξτε δύο τυχαίους, *διαφορετικούς* ακέραιους στο $[1, W + B]$. Αν ένας τυχαίος αριθμός είναι μικρότερος ή ίσος με W, έχετε επιλέξει άσπρη μπίλια, αλλιώς επιλέξατε μαύρη μπίλια.

    Υπόδειξη II: Θα σας χρειαστεί η παρακάτω συνάρτηση που επιστρέφει τυχαίο ακέραιο στο $[a, b]$:

```
FUNCTION rnd(a,b)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: a,b
  INTEGER :: rnd
  REAL :: u
  CALL RANDOM_NUMBER(u)
  rnd = a + FLOOR((b+1-a)*u)
END FUNCTION rnd
```

51. Τους θετικούς ακέραιους με το πολύ 7 ψηφία τους χωρίζουμε σε δύο σύνολα: στο ένα είναι οι ακέραιοι που στα ψηφία τους υπάρχει το 1 τουλάχιστον μία φορά και στο άλλο οι υπόλοιποι. Ποιο είναι το πλήθος των στοιχείων κάθε συνόλου;

    Απάντηση: 5217031, 4782968

Κεφάλαιο 5

Διανύσματα–Πίνακες

Χρησιμοποιώντας τους θεμελιώδεις, ενσωματωμένους τύπους που είδαμε ως τώρα, μπορούμε να ορίσουμε άλλους, σύνθετους, με κατάλληλο τρόπο ώστε να αναπαριστούν έννοιες του προβλήματός μας. Έτσι, για ομάδα σχετιζόμενων ποσοτήτων ίδιου τύπου χρησιμοποιούμε *διάνυσμα* (μονοδιάστατη συλλογή στοιχείων) ή *πίνακα* (πολυδιάστατη συλλογή στοιχείων), ενώ για σύνολο μεταβλητών διαφορετικού (ή και ίδιου) τύπου κατάλληλη είναι η δομή **TYPE** που θα περιγράψουμε στο Κεφάλαιο 6.

5.1 Διάνυσμα γνωστού πλήθους στοιχείων

5.1.1 Δήλωση

Συχνά υπάρχει η ανάγκη να χρησιμοποιήσουμε ένα σύνολο σχετιζόμενων ποσοτήτων με ίδιο τύπο. Για παράδειγμα, η θερμοκρασία ενός τόπου σε μια συγκεκριμένη ημέρα μπορεί να αναπαρασταθεί με μια πραγματική μεταβλητή· τι κάνουμε όμως αν επιθυμούμε να αποθηκεύσουμε τις θερμοκρασίες ενός μήνα ή ενός έτους; Θα μπορούσαμε να δηλώσουμε και να χρησιμοποιήσουμε 30 ή 365 μεταβλητές, αντιλαμβανόμαστε όμως ότι είναι επίπονη διαδικασία και υπάρχει μεγάλη πιθανότητα για λάθη. Η Fortran, όπως και όλες οι γλώσσες προγραμματισμού, μας δίνει τη δυνατότητα να τις ορίσουμε όλες μαζί, με μία εντολή. Έτσι, ένα σύνολο 30 πραγματικών μεταβλητών το αναπαριστούμε με ένα πραγματικό *διάνυσμα* 30 θέσεων. Η δήλωση τέτοιου διανύσματος με όνομα π.χ. temperature γίνεται ως εξής

```
DOUBLE PRECISION :: temperature(30)
```

ή, ισοδύναμα,

```
DOUBLE PRECISION, DIMENSION (30) :: temperature
```

Η δήλωση ενός διανύσματος μπορεί να γίνει ταυτόχρονα με άλλες, απλές, μεταβλητές ή και άλλα διανύσματα. Π.χ. η εντολή

```
DOUBLE PRECISION :: a, b(3), c(10)
```

δημιουργεί μια απλή μεταβλητή με όνομα a, ένα διάνυσμα 3 στοιχείων, τον b, και ένα άλλο διάνυσμα 10 θέσεων, το c, όλα πραγματικά. Προσέξτε όμως, στον ακόλουθο κώδικα, την εφαρμογή του δεύτερου τρόπου δήλωσης που είδαμε πιο πάνω:

```
INTEGER, DIMENSION (10) :: a, b, c, d(4)
```

Με την παραπάνω εντολή δημιουργούμε τέσσερα διανύσματα ακεραίων, τα τρία (a, b, c) με 10 στοιχεία και το τέταρτο (d) με 4.

Στη δηλώση διανύσματος το πλήθος των στοιχείων πρέπει να είναι *ακέραια σταθερή ποσότητα, γνωστή κατά τη μεταγλώττιση*. Αυτό σημαίνει ότι εκτός από ακέραιο αριθμό, για τη διάσταση μπορούμε να χρησιμοποιήσουμε

- ποσότητα που έχει την ιδιότητα **PARAMETER** (§2.4):

  ```
  INTEGER, PARAMETER :: n = 25
  DOUBLE PRECISION :: a(n)
  ```

- έκφραση με σταθερές ποσότητες:

  ```
  INTEGER, PARAMETER :: n = 25
  DOUBLE PRECISION :: a(2*n*n-n/2+13)
  ```

5.1.2 Πρόσβαση στα στοιχεία

Η πρόσβαση σε κάθε στοιχείο του διανύσματος γίνεται χρησιμοποιώντας το όνομα του διανύσματος ακολουθούμενο από ένα ακέραιο αριθμό (ή σταθερή έκφραση που έχει ακέραια τιμή) μέσα σε παρενθέσεις, από 1 έως το πλήθος των στοιχείων του διανύσματος[1]. Έτσι, το πρώτο στοιχείο είναι το `temperature(1)`, το δεύτερο είναι το `temperature(2)`, ενώ το τελευταίο είναι το `temperature(30)`. Με αυτά τα «ονόματα» συμμετέχουν σε εκφράσεις. Π.χ., ο μέσος όρος των 3 πρώτων στοιχείων του temperature είναι ο

```
DOUBLE PRECISION :: mo3

mo3 = (temperature(1) + temperature(2) + temperature(3)) / 3d0
```

5.1.3 Εκχώρηση τιμής

Απόδοση τιμής σε *στοιχείο διανύσματος* γίνεται με τους δύο γνωστούς τρόπους της εντολής εκχώρησης και της εντολής ανάγνωσης από το πληκτρολόγιο (ή αρχείο). Π.χ.

[1] στις περιπτώσεις που οι δηλώσεις γίνονται με τον τρόπο που είδαμε.

```
DOUBLE PRECISION :: a(5)

a(1) = 3.4d0
READ (*,*) a(2)
```

Ένα διάνυσμα μπορεί να πάρει τιμή *ως σύνολο* με τους ακόλουθους τρόπους:

- δίνοντας μεταξύ (/ και /)[2] ποσότητες κατάλληλου τύπου για όλα τα στοιχεία του. Έτσι, το πραγματικό διάνυσμα 5 στοιχείων με όνομα c, αποκτά τις τιμές 3.0, 4.5, 2.0, −100.0, 0.99 για κάθε στοιχείο του αντίστοιχα, με την ακόλουθη εντολή εκχώρησης

  ```
  DOUBLE PRECISION :: c(5)

  c = (/ 3.0d0, 4.5d0, 2.0d0, -1.0d2, 0.99d0 /)
  ```

- με εκχώρηση άλλου διανύσματος *της ίδιας διάστασης και πλήθους στοιχείων* (ή τμήματος πίνακα ή άλλου διανύσματος με αυτά τα χαρακτηριστικά, όπως θα δούμε παρακάτω). Π.χ.

  ```
  DOUBLE PRECISION, DIMENSION (5) :: c, d

  c = (/ 3.0d0, 4.5d0, 2.0d0, -1.0d2, 0.99d0 /)
  d = c
  ```

- με ανάγνωση από το πληκτρολόγιο ή από αρχείο, ποσοτήτων κατάλληλου πλήθους και τύπου όταν εκτελεστεί η εντολή **READ** για ολόκληρο το διάνυσμα:

  ```
  DOUBLE PRECISION :: c(5)

  READ (*,*) c   ! χρειάζεται 5 πραγματικούς
  ```

- με μετατροπή άλλου διανύσματος ή πίνακα χρησιμοποιώντας ενσωματωμένες συναρτήσεις (**RESHAPE**(), **SPREAD**(), **CSHIFT**(), **EOSHIFT**(), κλπ.). Δεν θα επεκταθούμε περισσότερο σε αυτές.

5.1.4 Όρια διανύσματος

Υπάρχουν περιπτώσεις που το σύνολο των ποσοτήτων που αναπαριστά το διάνυσμα έχει καταλληλότερη αρίθμηση, πιο «φυσική», που ξεκινά από άλλο αριθμό και όχι το 1. Π.χ. ένα διάνυσμα που αποθηκεύει τις πρώτες 6 δυνάμεις του 2 (2^0, 2^1, 2^2,...,2^5) είναι πιο βολικό να αριθμεί τα στοιχεία του από 0 έως 5. Η κατάλληλη δήλωση αυτού του διανύσματος περιλαμβάνει τους αριθμούς θέσης του πρώτου και του τελευταίου στοιχείου. Π.χ.

[2] ή [και] αν ο μεταγλωττιστής υποστηρίζει το πρότυπο Fortran 2003.

```
INTEGER :: powersOf2(0:5)

powersOf2 = (/1,2,4,8,16,32/)
```

Με την παραπάνω δήλωση, το διάνυσμα έχει πάλι 6 στοιχεία, τα `powersOf2(0)`, `powersOf2(1)`, ..., `powersOf2(5)`. Γενικότερα, η δήλωση

```
DOUBLE PRECISION :: a(M:N)
```

ή, ισοδύναμα,

```
DOUBLE PRECISION, DIMENSION (M:N) :: a
```

με M,N ακέραιες σταθερές και N≥M, δημιουργεί το πραγματικό διάνυσμα a με N−M+1 στοιχεία, τα `a(M)`, `a(M+1)`, ..., `a(N)`.

Με αφορμή τη δήλωση του powersOf2, ας δούμε έναν άλλο τρόπο να δημιουργήσουμε το σταθερό διάνυσμα [σύνολο τιμών χωριζόμενων με ',' μεταξύ `(/, /)`] αν αυτό έχει τιμές που μπορούν να παραχθούν επαναληπτικά:

```
INTEGER :: a(0:5)
INTEGER :: i

a = (/ (2**i,i=0,5) /)
```

Παρατηρήστε ότι έγινε χρήση του υπονοούμενου **DO** (§4.2.3) για να παραγάγουμε την επιθυμητή ακολουθία αριθμών. Τα διανύσματα powersOf2 και a αποκτούν τις ίδιες τιμές· ο δεύτερος τρόπος, του a, είναι πιο συνοπτικός, κατανοητός και επεκτάσιμος (πώς θα γράφαμε τις 20 πρώτες δυνάμεις;).

5.2 Πίνακας

Πολύ συχνά σε επιστημονικούς κώδικες, εμφανίζεται η ανάγκη να αναπαραστήσουμε ποσότητες σε 2 ή 3 διαστάσεις, π.χ. σε ένα καρτεσιανό πλέγμα. Η Fortran 95 παρέχει τη δυνατότητα να ορίσουμε διδιάστατους και, γενικότερα, πολυδιάστατους πίνακες (μέχρι 7 διαστάσεων). Η εντολή δήλωσης ενός πραγματικού διδιάστατου πίνακα 10×20 είναι η εξής

```
DOUBLE PRECISION :: a(10,20)
```

ή, ισοδύναμα,

```
DOUBLE PRECISION, DIMENSION (10,20) :: a
```

Το στοιχείο της γραμμής `i` και στήλης `j` είναι το `a(i,j)`. Ό,τι ισχύει για τα διανύσματα μπορεί να γενικευτεί και για τους πολυδιάστατους πίνακες[3].

[3] Για την ακρίβεια, ένας πολυδιάστατος πίνακας αποθηκεύεται εσωτερικά ως διάνυσμα, ξεκινώντας από τα στοιχεία για τα οποία ο πρώτος δείκτης αλλάζει πιο γρήγορα, κατόπιν ο δεύτερος, κλπ. Έτσι, ένας πίνακας `a`, 2×3, αποθηκεύεται ως διάνυσμα με διαδοχικά στοιχεία τα `a(1,1)`, `a(2,1)`, `a(1,2)`, `a(2,2)`, `a(3,1)`, `a(3,2)`.

Παράδειγμα
Έστω ότι έχουμε τις θερμοκρασίες ενός τόπου για κάθε ημέρα συγκεκριμένου έτους. Αυτές μπορεί να μας δίνονται με την παρακάτω μορφή

Ημέρα	Θερμοκρασία (°C)
1	5.0
2	7.5
3	6.4
⋮	⋮
157	19.1
158	21.4
⋮	⋮
364	4.5
365	7.0

Οι ημέρες αριθμούνται από το 1 έως το 365.

Παρατηρήστε ότι για να προσδιορίσουμε μια συγκεκριμένη θερμοκρασία πρέπει να γνωρίζουμε σε ποια γραμμή είναι, δηλαδή σε ποια ημέρα αναφερόμαστε. Με άλλα λόγια, η πληροφορία μας (οι θερμοκρασίες) παραμετροποιείται με *ένα* ακέραιο αριθμό. Είναι φυσιολογικό, σε πρόγραμμά μας, να αποθηκεύσουμε τις θερμοκρασίες σε ένα διάνυσμα:

```
DOUBLE PRECISION :: tempr(365)
```

Εναλλακτικά, οι θερμοκρασίες μπορεί να μας δίνονται στην ακόλουθη μορφή

	1	2	...	14	15	...	30	31
1	5.0	7.5	...	8.3	9.2	...	12.3	11.0
2	4.5	6.5	...	7.0	9.0	...		
⋮	⋮	⋮		⋮	⋮		⋮	⋮
6	25.0	27.5	...	28.3	29.2	...	27.3	
7	26.0	27.0	...	26.0	31.0	...	32.5	33.0
⋮	⋮	⋮		⋮	⋮		⋮	⋮
11	5.0	6.5	...	7.6	10.0	...	11.0	
12	5.0	7.5	...	8.5	9.5	...	12.0	12.5

Η πρώτη γραμμή παραθέτει τις ημέρες ενός μήνα ενώ η πρώτη στήλη παραθέτει τους μήνες.

Παρατηρήστε ότι για να προσδιορίσουμε μια συγκεκριμένη θερμοκρασία πρέπει να καθορίσουμε *δύο* ακέραιους αριθμούς: τον μήνα (γραμμή) και την ημέρα (στήλη). Επομένως, σε ένα προγραμμά μας είναι φυσιολογικό να χρησιμοποιήσουμε για τις θερμοκρασίες ένα διδιάστατο πίνακα 12×31,

```
DOUBLE PRECISION :: tempr(12,31)
```

που θα αποθηκεύει ουσιαστικά την ίδια πληροφορία με την προηγούμενη περίπτωση. Έτσι, η θερμοκρασία στις 14 Απριλίου είναι στη θέση `tempr(4,14)`[4].

5.3 Παρατηρήσεις

Σταθερό διάνυσμα ή πίνακας

Ένα διάνυσμα ή πίνακας μπορεί, όπως και κάθε άλλη ποσότητα, να οριστεί ως σταθερό περιλαμβάνοντας στη δήλωση την ιδιότητα **PARAMETER**, §2.4, με ταυτόχρονη εκχώρηση αρχικής (και μόνιμης) τιμής:

```
DOUBLE PRECISION, PARAMETER :: c(3) = (/2.0d0,2.5d0,4.0d0/)
```

Προσπέλαση όλων των στοιχείων διανύσματος ή πίνακα

Για να διατρέξουμε τα στοιχεία ενός διανύσματος ή πίνακα χρειαζόμαστε εντολή (ή εντολές) επανάληψης. Π.χ., η εκχώρηση τιμών στα στοιχεία ενός διανύσματος από το πληκτρολόγιο γίνεται ως εξής

```
DOUBLE PRECISION :: a(10)
INTEGER :: i

DO i = 1,10
   READ (*,*) a(i)
END DO
```

Προσέξτε ότι για να διατρέξουμε πίνακα δύο διαστάσεων χρειαζόμαστε δύο εντολές επανάληψης, με μεταβλητές ελέγχου που διατρέχουν η μία τις «γραμμές» και η άλλη τις «στήλες» του. Οι εντολές επανάληψης θα είναι η μία μέσα στην άλλη. Έτσι, η ανάγνωση τιμών σε πίνακα a με διάσταση 5×8 γίνεται ως εξής

```
DO i=1,5
   DO j=1,8
      READ (*,*) a(i,j)
   END DO
END DO
```

Η συγκεκριμένη επιλογή για τη σειρά των επαναλήψεων (αποκτά το i την πρώτη του τιμή και διατρέχουμε όλα τα j, μετά αλλάξει τιμή το i και ξαναδιατρέχουμε τα j, κοκ.) σημαίνει ότι όταν θα πληκτρολογούμε τις τιμές κατά τη διάρκεια εκτέλεσης του συγκεκριμένου κώδικα, πρέπει να δίνουμε τα στοιχεία του πίνακα κατά γραμμές.

Προσέξτε τον ακόλουθο κώδικα

[4] Θα πρέπει να προσέχουμε βέβαια να μην δώσουμε για δείκτες συνδυασμούς που δεν έχουν νόημα, π.χ. $(2, 30)$ ή $(6, 31)$.

```
IMPLICIT NONE
DOUBLE PRECISION :: b1, b2, b3
INTEGER :: i

DO i = 1, 3
   READ (*,*) bi   ! Λάθος, δεν έχει οριστεί η μεταβλητή bi
END DO
```

Οι μεταβλητές b1, b2, b3 είναι ανεξάρτητες, δεν είναι στοιχεία διανύσματος. *Δεν υπάρχει τρόπος να τις «παραγάγουμε» με εντολή επανάληψης*.

5.4 Διάνυσμα/Πίνακας άγνωστου πλήθους στοιχείων

Τονίσαμε στις δηλώσεις ότι το πλήθος των στοιχείων του διανύσματος πρέπει να είναι αριθμητική σταθερά, γνωστή εκ των προτέρων. Η Fortran 95 μας δίνει τη δυνατότητα να ορίσουμε διανύσματα με αρχικά άγνωστο πλήθος στοιχείων, που θα το μάθουμε, βέβαια, κατά την εκτέλεση του προγράμματος. Π.χ. αν διαβάζουμε ένα διάνυσμα από το πληκτρολόγιο ή από αρχείο, πιθανό είναι να μη γνωρίζουμε κατά τη μεταγλώττιση πόσα στοιχεία θα έχει. Αυτή την πληροφορία θα την πάρουμε από το χρήστη ή από το αρχείο πριν αρχίσει η εισαγωγή των στοιχείων, οπότε θα πρέπει να κατασκευάσουμε το διάνυσμα.

Η δήλωση

```
DOUBLE PRECISION, ALLOCATABLE :: a(:)
```

δηλώνει ένα πραγματικό διάνυσμα με όνομα a. Στοιχεία δεν έχει ακόμα. Αν θέλουμε να αποκτήσει N στοιχεία, όπου το N είναι ακέραια ποσότητα, σταθερή ή μεταβλητή, πρέπει να δεσμεύσουμε μνήμη δίνοντας την εντολή

```
ALLOCATE(a(N))
```

Όταν δε χρειαζόμαστε πλέον το συγκεκριμένο διάνυσμα καλό είναι να να «ελευθερώσουμε» τη μνήμη που κατέχει δίνοντας την εντολή

```
DEALLOCATE(a)
```

Αν δεν το κάνουμε, η αποδέσμευση γίνεται αυτόματα από το μεταγλωττιστή ή το λειτουργικό σύστημα[5].

Σε αντιστοιχία με τις δηλώσεις διανυσμάτων γνωστού πλήθους στοιχείων, το διάνυσμα με πρώτο στοιχείο το a(M) και τελευταίο το a(N) όπου M,N όχι απαραίτητα σταθερές ποσότητες, δημιουργείται με την εντολή

```
ALLOCATE(a(M:N))
```

Διδιάστατος (και, γενικεύοντας, πολυδιάστατος) πίνακας που θα δημιουργηθεί κατά την εκτέλεση δηλώνεται ως εξής

[5]όταν η ροή του προγράμματος επιστρέψει από το υποπρόγραμμα (Κεφάλαιο 8), στο οποίο έχει δηλωθεί το συγκεκριμένο (μη στατικό, §8.8) διάνυσμα ή τελειώσει το κυρίως πρόγραμμα.

```
DOUBLE PRECISION, ALLOCATABLE :: b(:,:)
```

Η δέσμευση μνήμης γίνεται με την εντολή

```
ALLOCATE(b(M,N))
```

όπου M,N ακέραιες ποσότητες. Η αποδέσμευση μπορεί να γίνει ρητά με την εντολή

```
DEALLOCATE(b)
```

ή αυτόματα από το μεταγλωττιστή ή το λειτουργικό σύστημα.

Οι εντολές **ALLOCATE/DEALLOCATE** μπορούν να χρησιμοποιηθούν για πολλά διανύσματα ή πίνακες ταυτόχρονα:

```
PROGRAM alloc
  IMPLICIT NONE
  DOUBLE PRECISION, DIMENSION (:), ALLOCATABLE :: a, b
  DOUBLE PRECISION, DIMENSION (:,:), ALLOCATABLE :: c
  INTEGER :: M,N, K, P

  READ (*,*) M, N, K, P

  ALLOCATE (a(M), b(N), c(K,P))

!  δίνουμε τιμές στους πίνακες a, b, c
!  χρησιμοποιούμε τους πίνακες a, b, c

  DEALLOCATE(a)
  DEALLOCATE(b, c)
END PROGRAM alloc
```

Προσέξτε ότι δεν μπορούμε να χρησιμοποιήσουμε στοιχεία διανύσματος ή πίνακα προτού αυτό δημιουργηθεί (με το **ALLOCATE**) ή αφού καταστραφεί (με το **DEALLOCATE**). Ένα διάνυσμα ή πίνακας που έχει αποδεσμεύσει τη μνήμη που κατείχε, μπορεί να ξαναδημιουργηθεί (με ίδιο ή διαφορετικό πλήθος στοιχείων).

Όταν θέλουμε να ελέγξουμε αν ένα διάνυσμα ή πίνακας που έχει δηλωθεί ως **ALLOCATABLE** έχει δημιουργηθεί, μπορούμε να χρησιμοποιήσουμε την ενσωματωμένη συνάρτηση **ALLOCATED**( ) που δέχεται ένα διάνυσμα ή πίνακα ως όρισμα. Επιστρέφει τιμή λογικού τύπου.

Παρατήρηση: Είναι λάθος η προσπάθεια να δημιουργήσουμε το διάνυσμα χωρίς να προσδιορίσουμε συγκεκριμένη τιμή για το πλήθος των στοιχείων ή να την προσδιορίσουμε αργότερα στο πρόγραμμά μας:

```
INTEGER :: n
DOUBLE PRECISION :: a(n)    ! Λάθος. Πόσο είναι το n;
READ (*,*) n
```

Επίσης, λάθος είναι να γράψουμε τη δήλωση του διανύσματος αφού διαβάσουμε ή υπολογίσουμε το πλήθος των στοιχείων σε μεταβλητή

```
INTEGER :: n
READ (*,*) n
DOUBLE PRECISION :: a(n)    ! Λάθος
```

Όπως γνωρίζουμε, όλες οι δηλώσεις πρέπει να γίνονται στην αρχή του προγράμματος, πριν αρχίσουν οι εντολές.

5.5 Σχετικές ενσωματωμένες συναρτήσεις

Η Fortran 95 παρέχει ένα πλήθος συναρτήσεων που δρουν σε διανύσματα ή πίνακες και εκτελούν πολλές συνήθεις πράξεις. Κάποιες από αυτές είναι:

SIZE():

Πολύ συχνά, ειδικά σε συναρτήσεις και υπορουτίνες που έχουν ως όρισμα ένα διάνυσμα ή πίνακα, είναι απαραίτητο να γνωρίζουμε το πλήθος των στοιχείων του ή το πλήθος θέσεων μιας διάστασής του. Η Fortran 95 παρέχει τη συνάρτηση **SIZE**() που μας δίνει αυτές τις πληροφορίες. Έτσι, έχοντας τον ακόλουθο πίνακα

```
DOUBLE PRECISION, DIMENSION(10,10,30) :: a
```

το πλήθος των στοιχείων του είναι **SIZE**(a), η πρώτη διάσταση έχει **SIZE**(a,1) θέσεις, η δεύτερη **SIZE**(a,2) και η τρίτη **SIZE**(a,3) θέσεις.

LBOUND()/**UBOUND**():

Κάποιες φορές χρειάζονται τα όρια των θέσεων σε κάθε διάσταση. Το κάτω όριο της πρώτης διάστασης στη δήλωση του πίνακα a είναι το **LBOUND**(a,1), της δεύτερης **LBOUND**(a,2), κλπ. Τα άνω όρια είναι αντίστοιχα **UBOUND**(a,1), **UBOUND**(a,2), κλπ.

SUM(): Η συνάρτηση **SUM**() με όρισμα διάνυσμα ή πίνακα μας επιστρέφει το άθροισμα όλων των στοιχείων του[6]. Π.χ.

```
INTEGER :: a(10), s

a = (/ 1, 5, 4, 2, 12, 9, 8, 2, -4, -7 /)
s = SUM(a)    ! s ← 32
```

το **SUM**() είναι μια πιο σύντομη εντολή, εναλλακτική της άθροισης με εντολή επανάληψης:

```
INTEGER :: a(10), s, i
```

[6] αρκεί τα στοιχεία να είναι ακέραιου, πραγματικού ή μιγαδικού τύπου ή, γενικότερα, αρκεί να ορίζεται η δράση του τελεστή '+' μεταξύ των στοιχείων.

```
a = (/ 1, 5, 4, 2, 12, 9, 8, 2, -4, -7 /)
s = 0
DO i = 1, SIZE(a)
   s = s + a(i)
END DO
```

PRODUCT(): Η συνάρτηση **PRODUCT**() με όρισμα διάνυσμα ή πίνακα μας επιστρέφει το γινόμενο όλων των στοιχείων του[7]. Π.χ.

```
INTEGER :: a(10), p

a = (/ 1, 5, 4, 2, 12, 9, 8, 2, -4, -7 /)
p = PRODUCT(a)     ! p ← 1935360
```

Η τελευταία εντολή ισοδυναμεί με

```
p = 1
DO i = LBOUND(a,1), UBOUND(a,1)
   p = p * a(i)
END DO
```

DOT_PRODUCT(): Η συνάρτηση **DOT_PRODUCT**() δέχεται για ορίσματα δύο διανύσματα με ίδιο πλήθος στοιχείων αλλά όχι απαραίτητα ίδιο τύπο. Η **DOT_PRODUCT**(a,b) επιστρέφει το

- $\sum_i a_i b_i$ αν το διάνυσμα a είναι ακέραιο ή πραγματικό.
- $\sum_i a_i^* b_i$ αν το διάνυσμα a είναι μιγαδικού τύπου.

Αν τα διανύσματα a,b είναι τύπου **LOGICAL**, το **DOT_PRODUCT**(a,b) επιστρέφει .**TRUE**. αν τουλάχιστο σε ένα ζεύγος αντίστοιχων στοιχείων των διανυσμάτων έχουν και τα δύο τιμή .**TRUE**., αλλιώς επιστρέφει .**FALSE**..

MATMUL(): Η συνάρτηση **MATMUL**() παίρνει δύο πίνακες ως ορίσματα και υπολογίζει το γινόμενό τους. Τουλάχιστον ένας πίνακας πρέπει να είναι διδιάστατος· ο άλλος μπορεί να είναι μονοδιάστατος (διάνυσμα). Οι διαστάσεις των πινάκων πρέπει να είναι συμβατές, σύμφωνα με όσα προβλέπονται από το μαθηματικό ορισμό του γινομένου πινάκων. Το αποτέλεσμα είναι διάνυσμα ή πίνακας με τις διαστάσεις που προκύπτουν από τη μαθηματική πράξη. Επομένως, η μαθηματική πράξη $C = A \cdot B$ με A, B, C πίνακες, γράφεται ως C=**MATMUL**(A,B).

Όπως γνωρίζουμε από τα μαθηματικά, το γινόμενο των πινάκων A με διαστάσεις $M \times N$ και B με διαστάσεις $N \times P$, είναι πίνακας C με διαστάσεις

[7] αρκεί τα στοιχεία να είναι ακέραιου, πραγματικού ή μιγαδικού τύπου ή, γενικότερα, αρκεί να ορίζεται η δράση του τελεστή '*' μεταξύ των στοιχείων

$M \times P$ και στοιχεία

$$C_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \ , \quad 1 \leq i \leq M \ , 1 \leq j \leq P \ .$$

Η εντολή C=**MATMUL**(A,B) με ορίσματα συμβατούς διδιάστατους πίνακες, με πραγματικά στοιχεία, διπλής ακρίβειας, ισοδυναμεί με τον ακόλουθο κώδικα

```
INTEGER :: i, j, k
DOUBLE PRECISION :: s

DO i = LBOUND(A,1), UBOUND(A,1)
   DO j = LBOUND(B,1), UBOUND(B,1)
      s = 0.0d0
      DO k = LBOUND(A,2), UBOUND(A,2)
         s = s + a(i,k) * b(k,j)
      END DO
      C(i,j) = s
   END DO
END DO
```

TRANSPOSE(): Η συνάρτηση **TRANSPOSE**() επιστρέφει τον ανάστροφο πίνακα του ορίσματός της. Έτσι μπορούμε να γράψουμε

```
DOUBLE PRECISION :: a(10,20), at(20,10)

at = TRANSPOSE(a)
```

Ο πίνακας at έχει στη θέση (i, j) την τιμή του στοιχείου (j, i) του πίνακα a.

COUNT(): Η συνάρτηση **COUNT**() με όρισμα διάνυσμα ή πίνακα λογικών ποσοτήτων ή λογική συνθήκη, απλή ή σύνθετη, στην οποία συμμετέχει πίνακας, υπολογίζει το πλήθος των στοιχείων που ικανοποιούν τη συγκεκριμένη συνθήκη. Π.χ. το **COUNT**(A > 10) μετρά το πλήθος των στοιχείων του (ακέραιου ή πραγματικού) πίνακα A που είναι μεγαλύτερα από 10.

MAXVAL()/**MINVAL**(): Οι συναρτήσεις **MAXVAL**() και **MINVAL**() επιστρέφουν τη μέγιστη και ελάχιστη, αντίστοιχα, τιμή που είναι αποθηκευμένη στο διάνυσμα ή πίνακα που δέχονται ως όρισμα.

Παράδειγμα
Έστω ότι δηλώθηκε το ακέραιο διάνυσμα a, με κάτω όριο θέσεων το 1, και *του έχουμε δώσει τιμές*. Οι παρακάτω κώδικες είναι ισοδύναμοι:

```
INTEGER :: m
INTEGER :: i
m = a(1)
```

```
DO i = 2,SIZE(a)
   IF (m > a(i)) m = a(i)
END DO
PRINT *, "Minimum ", m
```

και

```
INTEGER :: m
m = MINVAL(a)
PRINT *, "Minimum ", m
```

MAXLOC()/MINLOC(): Οι συναρτήσεις **MAXLOC()** και **MINLOC()** δέχονται διάνυσμα ή πίνακα ως όρισμα και επιστρέφουν σε διάνυσμα ακεραίων τους δείκτες που προσδιορίζουν τη θέση του μεγαλύτερου/μικρότερου στοιχείου. Π.χ. αν ο a είναι διδιάστατος πίνακας που έχει στη θέση $(2, 3)$ το ελάχιστο στοιχείο του, η κλήση της συνάρτησης **MINLOC**(a) επιστρέφει σε διάνυσμα δύο στοιχείων τις τιμές 2 (πρώτη θέση), 3 (δεύτερη θέση). Φυσικά, το διάνυσμα αυτό πρέπει να δηλωθεί κατάλληλα από το χρήστη.

Αν οι συγκεκριμένες συναρτήσεις κληθούν με πρώτο όρισμα ένα διάνυσμα και δεύτερο τον αριθμό 1, μας επιστρέφουν τη θέση του μέγιστου ή ελάχιστου στοιχείου ως βαθμωτό μέγεθος (ένα ακέραιο αριθμό).

Παράδειγμα
Έστω ότι δηλώθηκε το ακέραιο διάνυσμα a, με κάτω όριο θέσεων το 1, και *του έχουμε δώσει τιμές*. Οι παρακάτω κώδικες είναι ισοδύναμοι:

```
INTEGER :: j
INTEGER :: i
j = 1
DO i = 2,SIZE(a)
   IF (a(j) > a(i)) j = i
END DO
PRINT *, "index ", j
PRINT *, "minimum ", a(j)
```

και

```
INTEGER :: j
j = MINLOC(a,1)
PRINT *, "index ", j
PRINT *, "minimum ", a(j)
```

ALL(): Η συνάρτηση **ALL()** δέχεται ως όρισμα μια λογική συνθήκη, απλή ή σύνθετη, στην οποία συμμετέχει διάνυσμα ή πίνακας. Εάν για κάθε στοιχείο του πίνακα η συνθήκη είναι αληθής, η συνάρτηση επιστρέφει .**TRUE**.. Σε άλλη περίπτωση επιστρέφει .**FALSE**..

ANY(): Η συνάρτηση **ANY**() δέχεται ως όρισμα μια λογική συνθήκη, απλή ή σύνθετη, στην οποία συμμετέχει διάνυσμα ή πίνακας. Εάν έστω και για ένα στοιχείο του πίνακα η συνθήκη είναι αληθής, η συνάρτηση επιστρέφει .**TRUE**.. Στην αντίθετη περίπτωση επιστρέφει .**FALSE**..

Δε θα αναφερθούμε σε επιπλέον ορίσματα που δέχονται κάποιες από τις παραπάνω συναρτήσεις.

5.6 Πράξεις διανυσμάτων/πινάκων κατά στοιχείο

Ένα βασικό χαρακτηριστικό που εισήχθη με τη Fortran 90 για τους ενσωματωμένους τελεστές και υποπογράμματα και επεκτάθηκε με την Fortran 95 ώστε να ισχύει και για τελεστές και υποπογράμματα που ορίζει ο χρήστης, είναι η υποστήριξη για τις πράξεις *κατά στοιχείο* (elemental).

Είδαμε ότι οι τελεστές $=, +, -, *, /, **$ και οι ενσωματωμένες συναρτήσεις του Πίνακα 2.1 εκτελούν πράξεις σε απλές ποσότητες (μεταβλητές ή σταθερές). Στη Fortran 95 μπορούν να εκτελέσουν τις ίδιες πράξεις και σε διανύσματα ή πίνακες σαν να εκτελούσαμε τη συγκεκριμένη πράξη σε κάθε ένα στοιχείο χωριστά. Για παράδειγμα, ξέρουμε ότι η εκχώρηση τιμής σε μία απλή μεταβλητή μπορεί να γίνει ως εξής

```
DOUBLE PRECISION :: x

x = 3.4d0
```

Η εκχώρηση της ίδιας τιμής σε όλα τα στοιχεία ενός διανύσματος μπορεί βέβαια να γίνει με τον ακόλουθο κώδικα

```
DOUBLE PRECISION :: a(10)
INTEGER :: i

DO i=1,SIZE(a)
   a(i) = 3.4d0
END DO
```

Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε την εντολή εκχώρησης σε ολόκληρο διάνυσμα

```
DOUBLE PRECISION :: a(10)

a = 3.4d0
```

Ο τελεστής '=' σε αυτήν την περίπτωση αποδίδει τη σταθερή τιμή σε *κάθε ένα στοιχείο* του a.

Αντίστοιχα ισχύουν και όταν έχουμε εκχώρηση διανύσματος σε διάνυσμα §5.1.3. Η εντολή `a = b`, όπου a, b διανύσματα ή πίνακες *ίδιας διάστασης και πλήθους στοιχείων*, κάνει αντιγραφή κάθε στοιχείου του b στο αντίστοιχο του a.

Άλλο παράδειγμα είναι η πρόσθεση διανυσμάτων ή πινάκων *ίδιας διάστασης και πλήθους στοιχείων* και γενικά οποιαδήποτε πράξη με πίνακες. Έτσι, το άθροισμα δύο διανυσμάτων μπορεί να γίνει είτε με το

```
DOUBLE PRECISION, DIMENSION (10) :: a, b, c
INTEGER :: i
..... ! εκχώρηση τιμών στα a,b

DO i=1,SIZE(a)
   c(i) = a(i) + b(i)
END DO
```

είτε πιο συνοπτικά με τον κώδικα

```
DOUBLE PRECISION, DIMENSION (10) :: a, b, c
..... ! εκχώρηση τιμών στα a,b

c = a + b
```

Οι γνωστοί τελεστές από τις πράξεις μεταξύ απλών ποσοτήτων, όταν δρουν μεταξύ διανυσμάτων ή πινάκων εκτελούν τις αντίστοιχες πράξεις στα ίδιας θέσης στοιχεία σαν να είχαμε κατάλληλο **DO**. Προσέξτε ότι ο τελεστής $*$ εκτελεί πολλαπλασιασμό κατά στοιχεία και όχι τον πολλαπλασιασμό διανυσμάτων ή πινάκων, δηλαδή οι παρακάτω κώδικες

```
DOUBLE PRECISION, DIMENSION (10) :: a, b, c
..... ! εκχώρηση τιμών στα a,b

c = a * b
```

και

```
DOUBLE PRECISION, DIMENSION (10) :: a, b, c
INTEGER :: i
..... ! εκχώρηση τιμών στα a,b

DO i=1,SIZE(a)
   c(i) = a(i) * b(i)
END DO
```

είναι ισοδύναμοι.

Απλή μεταβλητή, όταν εμφανίζεται σε πράξη με διάνυσμα πίνακα, «μετατρέπεται» σε πίνακα κατάλληλης διάστασης και πλήθους στοιχείων ώστε η πράξη να μπορεί να γίνει. Αυτό σημαίνει ότι με τις ακόλουθες δηλώσεις

```
DOUBLE PRECISION :: lambda, a(10), b(20)
```

οι κώδικες

```
a = lambda * a
```

και

```
b = lambda * b
```

είναι σωστοί: πολλαπλασιάζονται με lambda τα στοιχεία των a,b, όσα κι αν είναι αυτά. Αντίθετα, ο κώδικας

```
a = a * b
```

δεν έχει νόημα.

Εκτός από τους τελεστές, και οι ενσωματωμένες συναρτήσεις δρουν σε καθένα στοιχείο του ορίσματός τους αν αυτό είναι πίνακας και επιστρέφουν πίνακα αντίστοιχης διάστασης και πλήθους στοιχείων. Π.χ. ο κώδικας

```
DOUBLE PRECISION :: x, y
..... ! εκχώρηση τιμής στο x

y = SQRT(x)
```

υπολογίζει την τετραγωνική ρίζα του x και την εκχωρεί στο y. Η ίδια ακριβώς εντολή υπολογίζει τη ρίζα κάθε στοιχείου του διανύσματος a στον παρακάτω κώδικα και την εκχωρεί στα αντίστοιχα στοιχεία του όμοιου διανύσματος b:

```
DOUBLE PRECISION :: a(20), b(20)
..... ! εκχώρηση τιμής στο a

b = SQRT(a)
```

Ισοδύναμα, αλλά με πιο σύνθετο κώδικα, θα μπορούσαμε να γράψουμε

```
DOUBLE PRECISION :: a(20), b(20)
INTEGER :: i
..... ! εκχώρηση τιμής στο a

DO i=1,SIZE(a)
  b(i) = SQRT(a(i))
END DO
```

Ο μεταγλωττιστής έχει τη δυνατότητα να βελτιστοποιήσει ή να τις εκτελέσει παράλληλα, σε πολλούς επεξεργαστές ταυτόχρονα, τις πράξεις στις οποίες συμμετέχει ολόκληρος διάνυσμα ή πίνακας (ή τμήμα αυτού, §5.7), επιταχύνοντας έτσι το πρόγραμμα.

5.7 Τμήμα διανύσματος/πίνακα

Ένα μέρος των στοιχείων ενός πίνακα αποτελεί για τη Fortran έναν νέο πίνακα. Π.χ. αν a είναι διάνυσμα με 10 στοιχεία, τα στοιχεία $3, 4, 5, 6, 7$ αυτού συμβολίζονται με `a(3:7)` και αποτελούν ένα άλλο διάνυσμα (με 5 στοιχεία)· η συνάρτηση

SIZE(a(3:7)) επιστρέφει την τιμή 5. Το νέο αυτό διάνυσμα μπορεί να συμμετέχει σε οποιαδήποτε πράξη χρειάζεται διάνυσμα 5 στοιχείων. Έτσι, ο παρακάτω κώδικας

```
DOUBLE PRECISION :: a(10), b(20)

a = b(1:10)
```

είναι σωστός (τα 10 πρώτα στοιχεία του b εκχωρούνται στα αντίστοιχα του a), ενώ αντίθετα η εκχώρηση a=b είναι λάθος. Φυσικά ισχύει ότι a ≡ a(1:**SIZE**(a)) για κάθε πίνακα a (που έχει δηλωθεί ότι η αρίθμηση των στοιχείων του αρχίζει από το 1).

Επίσης, έχουμε τη δυνατότητα να ορίσουμε νέο πίνακα από ένα σύνολο στοιχείων κάποιου άλλου, χωρίς αυτά να είναι συνεχόμενα. Έτσι, το πρώτο, τρίτο, πέμπτο και έβδομο στοιχείο ενός πίνακα a αποτελούν πίνακα μονοδιάστατο με 4 στοιχεία. Ο νέος πίνακας είναι ο a(1:7:2). Ο τρίτος αριθμός στην παρένθεση καθορίζει το βήμα της αύξησης, το οποίο μπορεί να είναι και αρνητικό. Γενικά, από έναν πίνακα a δημιουργούμε άλλον προσδιορίζοντας αρχική θέση, τελική θέση και βήμα αύξησης ως εξής

a(αρχική θέση : τελική θέση : βήμα μεταβολής)

Αν το βήμα μεταβολής παραλείπεται, υπονοείται το 1. Αν παραλείπεται η αρχική ή η τελική θέση υπονοείται το πρώτο και το τελευταίο στοιχείο αντίστοιχα. Αν το βήμα είναι θετικό, πρέπει η αρχική θέση να μην είναι μετά την τελική· το αντίστροφο πρέπει να ισχύει αν το βήμα είναι αρνητικό. Έτσι

a(:10) ≡ a(1:10)
a(4:) ≡ a(4:SIZE(a))
a(:) ≡ a(1:SIZE(a)) ≡ a

Ενδιαφέρουσα εφαρμογή των παραπάνω είναι η ακόλουθη: έστω ότι θέλουμε να αντιστρέψουμε τη σειρά των στοιχείων ενός διανύσματος. Μπορούμε να το κάνουμε με τον ακόλουθο κώδικα

```
DOUBLE PRECISION :: a(100), temp
INTEGER :: i

DO i = 1, SIZE(a)/2
   temp = a(SIZE(a) - i + 1)
   a(SIZE(a) - i + 1) = a(i)
   a(i) = temp
END DO
```

Προσέξτε ότι πρέπει να χρησιμοποιήσουμε μια προσωρινή μεταβλητή καθώς δεν μπορούμε να κάνουμε απευθείας εναλλαγή των a(i), a(**SIZE**(a)−i+1). Ισοδύναμα και πολύ πιο απλά, μπορούμε να δώσουμε την ακόλουθη εντολή

```
a = a(SIZE(a):1:-1)
```

Ο νέος πίνακας στο δεξί μέλος της εκχώρησης έχει τα ίδια στοιχεία με τον a αλλά με αρίθμηση αντίθετης φοράς· προσέξτε ότι πρώτα υπολογίζεται και μετά αποδίδεται στον a.

Ανάλογα ισχύουν και για πίνακες. Έτσι αν ο a είναι πίνακας 10×20, με δήλωση

```
INTEGER :: a(10,20)
```

τότε ο `a(1:5,1:6)` είναι διδιάστατος πίνακας 5×6 και αποτελείται από τα στοιχεία των 5 πρώτων γραμμών και 6 πρώτων στηλών.

Παρατηρήστε ότι μπορούμε να επιλέξουμε μία γραμμή (π.χ. την τρίτη) ή στήλη (π.χ. την πέμπτη) του διδιάστατου πίνακα a με τους συμβολισμούς `a(3,:)` και `a(:,5)` αντίστοιχα. Οι νέοι αυτοί πίνακες είναι διανύσματα.

5.8 Εντολές που αφορούν διανύσματα/πίνακες

Η Fortran 95 παρέχει ορισμένες εντολές που δρουν σε διάνυσμα/πίνακες ή τμήματά τους. Ο μεταγλωττιστής έχει τη δυνατότητα να βελτιστοποιήσει ή να εκτελέσει παράλληλα, σε πολλούς επεξεργαστές ταυτόχρονα, τις διαδικασίες που καθορίζονται από αυτές. Η ταυτόχρονη εκτέλεση εντολών μικραίνει το συνολικό χρόνο εκτέλεσης του προγράμματος.

5.8.1 WHERE

Η εντολή **WHERE** μπορεί να θεωρηθεί ως ένα γενικευμένο **IF**, στη συνθήκη του οποίου συμμετέχει διάνυσμα ή πίνακας, και στο σώμα του έχουμε μόνο εκχωρήσεις σε όμοια διανύσματα ή πίνακες. Πιο συγκεκριμένα, η σύνταξη του **WHERE** είναι

```
WHERE (condition)
   ....
   ....
ELSEWHERE
   ....
   ....
END WHERE
```

Η συνθήκη condition, είναι λογική έκφραση, απλή ή σύνθετη, που περιλαμβάνει διάνυσμα ή πίνακα. Το σώμα του **WHERE** επιτρέπεται να έχει μόνο εντολές ανάθεσης σε ένα ή περισσότερους πίνακες *ίδιας διάστασης και πλήθους στοιχείων* με τον πίνακα της συνθήκης. Για τις θέσεις του συγκεκριμένου πίνακα στις οποίες τα στοιχεία ικανοποιούν τη συνθήκη, εκτελούνται οι αντίστοιχες εκχωρήσεις στο τμήμα πριν το **ELSEWHERE**. Για όσα στοιχεία δεν ικανοποιείται η συνθήκη, εκτελούνται οι αντίστοιχες εκχωρήσεις στο τμήμα μετά το **ELSEWHERE**. Π.χ., έστω ότι θέλουμε να αντιγράψουμε τα θετικά στοιχεία του πραγματικού πίνακα a στον όμοιο πίνακα b, ενώ θέλουμε να μηδενίσουμε τα στοιχεία του b αν το αντίστοιχο στοιχείο του a δεν είναι θετικό. Αυτό γίνεται με τον κώδικα

```
DO i = LBOUND(a,1),UBOUND(a,1)
   IF (a(i) > 0.0d0) THEN
      b(i) = a(i)
   ELSE
      b(i) = 0.0d0
   END IF
END DO
```

ή με το

```
WHERE (a > 0.0d0)
   b = a
ELSEWHERE
   b = 0.0d0
END WHERE
```

ή, βέβαια, με το

```
b = MAX(a, 0.0d0)
```

Στην εντολή **WHERE**, στην περίπτωση που δεν έχουμε εκχωρήσεις όταν η συνθήκη είναι ψευδής, μπορούμε να παραλείψουμε το **ELSEWHERE**. Εάν, επιπλέον, έχουμε *μια μόνο* εντολή εκχώρησης για αληθή συνθήκη, μπορούμε να τη γράψουμε στην ίδια γραμμή με το **WHERE**. Έτσι, μπορούμε να πάρουμε την τετραγωνική ρίζα του a στον b με τον ακόλουθο κώδικα

```
WHERE (a>=0.0d0) b = SQRT(a)
```

Τα στοιχεία του b που αντιστοιχούν σε αρνητικές τιμές του a δεν τροποποιούνται.

5.9 Ασκήσεις

1. Δημιουργήστε ένα διάνυσμα με 100 ακέραια στοιχεία. Στο στοιχείο του διανύσματος στη θέση i ($i = 1, \ldots, 100$) δώστε την τιμή $i^2 + 3i + 1$. Κατόπιν, υπολογίστε το μέσο όρο των στοιχείων του διανύσματος.

2. Δημιουργήστε διάνυσμα με πλήθος στοιχείων N που θα το προσδιορίζει ο χρήστης. Στο στοιχείο j ($j = 1, \ldots, N$) δώστε την τιμή $\sin(\pi j/N)$. Κατόπιν, υπολογίστε τη μέγιστη και την ελάχιστη τιμή σε αυτό το διάνυσμα καθώς και το πλήθος των στοιχείων που είναι κατ᾽ απόλυτη τιμή μεγαλύτερα από 0.4. Μπορείτε να τα βρείτε με χρήση ενσωματωμένων συναρτήσεων;

3. Γράψτε κώδικα που να δημιουργεί δύο πραγματικούς πίνακες A, B με διαστάσεις 20×30. Σε κάθε στοιχείο (i, j) ($i = 1, \ldots, 20$, $j = 1, \ldots, 30$) του A δώστε την τιμή $(i + j)/3$ ενώ στο $B(i, j)$ δώστε την τιμή $2i - j/3$.

   (α′) Υπολογίστε τον ανάστροφο πίνακα B^T του B με και χωρίς τη χρήση ενσωματωμένης συνάρτησης.

   (β′) Υπολογίστε το γινόμενο[8] των πινάκων A, B^T με και χωρίς τη χρήση ενσωματωμένης συνάρτησης.

   (γ′) Υπολογίστε το άθροισμα των στοιχείων της κύριας διαγωνίου (το ίχνος) του πίνακα $A \cdot B^T$.

4. Δημιουργήστε ένα διάνυσμα a, 100 πραγματικών στοιχείων. Στο στοιχείο j ($j = 1, \ldots, 100$) του διανύσματος δώστε την τιμή $\cos(\pi j/100)$. Κατόπιν, εναλλάξτε τα πρώτα 50 στοιχεία με τα 50 τελευταία, δηλαδή, $a(1) \leftrightarrow a(51)$, $a(2) \leftrightarrow a(52)$, ..., $a(50) \leftrightarrow a(100)$.

5. Γράψτε πρόγραμμα που να υπολογίζει και να αποθηκεύει σε διάνυσμα τα παραγοντικά των αριθμών από το 0 ως το 12. Κατόπιν, να υπολογίζει το e^x από το άθροισμα

$$\mathrm{e}^x \approx x^0/0! + x^1/1! + x^2/2! + \cdots + x^{12}/12! \, .$$

   Το x θα το δίνει ο χρήστης. Συγκρίνετε το αποτέλεσμα με αυτό που δίνει η ενσωματωμένη συνάρτηση `EXP()`.

6. Να υπολογίσετε το π από τον τύπο

$$\frac{1}{\pi} = \sum_{n=0}^{\infty} \frac{((2n)!)^3 (42n + 5)}{(n!)^6 16^{3n+1}} \, ,$$

[8] Το γινόμενο των πινάκων $\boldsymbol{A}_{M\times N}$, $\boldsymbol{B}_{N\times P}$ με στοιχεία τα A_{ij}, B_{ij}, είναι ο πίνακας $\boldsymbol{C}_{M\times P}$ με στοιχεία

$$C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj} \, .$$

κρατώντας τους πέντε πρώτους όρους στο άθροισμα. Το αποτέλεσμα με 15 ψηφία θα πρέπει να πλησιάζει την τιμή 3.1415926535898.

Υπόδειξη: Πρώτα υπολογίστε και αποθηκεύστε σε διάνυσμα τα παραγοντικά που θα χρειαστείτε.

7. Το «κόσκινο του Ερατοσθένη» είναι ένας αλγόριθμος που μπορεί να βρει τους θετικούς ακέραιους, μέχρι μια μέγιστη τιμή, που έχουν την ιδιότητα να είναι πρώτοι (δηλαδή να διαιρούνται ακριβώς μόνο από το 1 και τον εαυτό τους). Σύμφωνα με αυτόν, αποθηκεύουμε σε ένα διάνυσμα όλους τους ακέραιους από το 2 μέχρι τη μέγιστη τιμή και μετά διαγράφουμε (ή πιο απλά μηδενίζουμε) τα *πολλαπλάσια* κάθε αριθμού σε αυτό το διάνυσμα· *όχι τους ίδιους τους αριθμούς*. Όποιοι απομείνουν, δηλαδή δεν έχουν μηδενιστεί, είναι πρώτοι.

   Υπόδειξη: Προσέξτε ότι πρέπει να παραλείπουμε τους αριθμούς στο διάνυσμα που είναι 0.

   Γράψτε πρόγραμμα που να βρίσκει και να τυπώνει όλους τους πρώτους αριθμούς μέχρι το 1000, εφαρμόζοντας αυτό τον αλγόριθμο.

8. Ο Μανώλης, ο επιστάτης, είναι υπεύθυνος για να ανάβει και να σβήνει τα φώτα σε διάδρομο ενός κτηρίου. Έστω ότι ο διάδρομος έχει n λαμπτήρες στη σειρά. Καθένας έχει ένα χαρακτηριστικό αριθμό: $1, 2, 3, \ldots, n$.

   Κάθε λαμπτήρας έχει το δικό του διακόπτη. Το είδος του διακόπτη είναι τέτοιο ώστε πατώντας τον ανάβει ο λαμπτήρας (αν είναι σβηστός) ή σβήνει (αν είναι αναμμένος). Ο Μανώλης κάνει n διαδρομές πήγαινε–έλα (όσοι οι λαμπτήρες στο διάδρομο). Στη διαδρομή i διασχίζει το διάδρομο και πατάει το διακόπτη κάθε λαμπτήρα που ο χαρακτηριστικός αριθμός του είναι πολλαπλάσιος του i. Στην επιστροφή κάθε διαδρομής δεν πατά κανένα διακόπτη.

   Πόσοι είναι οι αναμμένοι λαμπτήρες μετά τη διαδρομή n, αν υποθέσουμε ότι αρχικά ήταν όλοι αναμμένοι;

   Υπόδειξη: Το πλήθος των λαμπτήρων θα το δίνει ο χρήστης κατά την εκτέλεση του προγράμματος.

9. Δημιουργήστε ένα πίνακα $M \times N$ με $M = 20$, $N = 60$, στον οποίο $K = 400$ στοιχεία θα έχουν την τιμή 1 και τα υπόλοιπα θα είναι 0. Τα στοιχεία με τιμή 1 θα είναι επιλεγμένα με τυχαίο τρόπο (§2.9.1). Τυπώστε στην οθόνη τον πίνακα αυτόν κατά σειρές, βάζοντας `'x'` για τα μη μηδενικά στοιχεία και `'o'` για τα μηδενικά.

10. Έστω ένα πλέγμα 9×9 πάνω στο οποίο κινείται ένα μυρμήγκι. Σε κάθε βήμα του, το μυρμήγκι κινείται τυχαία σε τετράγωνο που γειτονεύει με την τρέχουσα θέση του (δηλαδή πάνω, κάτω, δεξιά ή αριστερά· όχι διαγωνίως). Δεν μπορεί να φύγει από το πλέγμα.

    Σε κάθε τετράγωνο της πρώτης γραμμής του πλέγματος υπάρχει αρχικά ένας σπόρος. Όταν το μυρμήγκι, στην τυχαία του κίνηση, βρεθεί σε τετράγωνο

της πρώτης σειράς, «φορτώνεται» τον σπόρο και τον μεταφέρει έως ότου βρεθεί σε τετράγωνο της τελευταίας γραμμής του πλέγματος όπου και αφήνει τον σπόρο. Το μυρμήγκι μπορεί να μεταφέρει μόνο ένα σπόρο κάθε φορά· εάν βρεθεί σε τετράγωνο της πρώτης γραμμής που δεν έχει σπόρο (γιατί τον πήρε σε προηγούμενη επίσκεψη) προφανώς δεν παίρνει τίποτε. Επίσης, αν μεταφέρει σπόρο σε τετράγωνο της τελευταίας γραμμής που έχει ήδη σπόρο (από προηγούμενη επίσκεψη) δεν μπορεί να αφήσει το φορτίο του.

Η κίνηση του μυρμηγκιού τελειώνει όταν μεταφέρει όλους τους σπόρους στην τελική γραμμή.

Να γράψετε κώδικα που να προσομοιώνει την παραπάνω διαδικασία από την αρχική ως την τελική κατάσταση. Να τυπώνει το πλήθος των κινήσεων που έγιναν. Δώστε ως αρχική θέση του μυρμηγκιού το κεντρικό τετράγωνο.

11. Πρώτος χαρακτηρίζεται κάθε θετικός ακέραιος αριθμός μεγαλύτερος του 1 αν δεν διαιρείται ακριβώς με άλλο αριθμό εκτός από το 1 και τον εαυτό του.

    Γράψτε πρόγραμμα που

    (α΄) Βρίσκει και αποθηκεύει σε διάνυσμα όλους τους πρώτους αριθμούς μέχρι το 1000. Να το κάνετε ως εξής:

    - Μετρήστε πόσοι είναι οι πρώτοι ακέραιοι μέχρι το 1000 (υπολογίστε και αγνοήστε τους, απλά μετρήστε).
    - Δημιουργήστε ακέραιο διάνυσμα με πλήθος θέσεων όσοι είναι οι πρώτοι αριθμοί.
    - Υπολογίστε ξανά τους πρώτους. Αποθηκεύστε τους αυτή τη φορά στο διάνυσμα.

    (β΄) Υπολογίζει και τυπώνει στην οθόνη τους διαιρέτες του αριθμού 154938756 που είναι πρώτοι. Προσέξτε ότι μπορεί να επαναλαμβάνονται κάποιοι.

    Υπόδειξη: Για να ελέγξετε αν ένας αριθμός n είναι πρώτος, ψάξτε να βρείτε κάποιον θετικό ακέραιο από το 2 έως το $n-1$ που να τον διαιρεί ακριβώς (δηλαδή χωρίς υπόλοιπο).

    Απάντηση: $154938756 = 2 \times 2 \times 3 \times 7 \times 71 \times 83 \times 313$

12. Ο υπολογισμός της τιμής του πολυωνύμου $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ για κάποιο συγκεκριμένο x μπορεί να γίνει με το ελάχιστο πλήθος πράξεων ως εξής: $p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$. Αυτός ο τρόπος υπολογισμού, που φυσικά επεκτείνεται σε πολυώνυμο οποιασδήποτε τάξης, αποτελεί τον *κανόνα Horner*.

    Γράψτε κώδικα που να υπολογίζει την τιμή στο $x = 1.3$ του πολυωνύμου $p(x) = 3.0 + 2.5x - 1.4x^2 - 2.1x^3 + 5x^4$, εφαρμόζοντας τον κανόνα Horner.

    Υπόδειξη: Ξεκινήστε τον υπολογισμό από την πιο εσωτερική παρένθεση.

13. Ένα μυρμήγκι βρίσκεται σε ορθογώνιο πλέγμα από 1000×1000 τετράγωνα. Τα τετράγωνα μπορούν να είναι είτε άσπρα είτε μαύρα. Το μυρμήγκι μπορεί να κινηθεί κατά ένα τετράγωνο τη φορά, παράλληλα με τους άξονες (δηλαδή, μόνο πάνω, κάτω, δεξιά, αριστερά) σύμφωνα με τους κανόνες:

    - Αν είναι σε μαύρο τετράγωνο, αλλάξει το χρώμα του τετραγώνου σε άσπρο, στρέφει κατά 90° αριστερόστροφα και προχωρά κατά ένα τετράγωνο.
    - Αν είναι σε άσπρο τετράγωνο, αλλάξει το χρώμα του τετραγώνου σε μαύρο, στρέφει κατά 90° δεξιόστροφα και προχωρά κατά ένα τετράγωνο.

    Ας υποθέσουμε ότι το πλέγμα αρχικά έχει μόνο άσπρα τετράγωνα και ότι το μυρμήγκι βρίσκεται αρχικά στο κέντρο, στο τετράγωνο με συντεταγμένες $(500, 500)$, και με κατεύθυνση προς τα πάνω. Μετά από 10000 κινήσεις, σε ποια θέση (συντεταγμένες) βρίσκεται το μυρμήγκι; Ποια κατεύθυνση έχει (πάνω, κάτω, δεξιά, αριστερά); Πόσα τετράγωνα είναι μαύρα;

Κεφάλαιο 6

Παραγόμενοι Τύποι

Όπως αναφέραμε ήδη, οι σχετιζόμενες ποσότητες του προβλήματός μας, με ίδιο τύπο, είναι προτιμότερο να αναπαρίστανται στον κώδικά μας με διάνυσμα ή πίνακα παρά με ισάριθμες ανεξάρτητες μεταβλητές. Στην οργάνωση του κώδικα, αλλά και στη διαχείριση των μεταβλητών, τα διανύσματα και οι πίνακες παρουσιάζουν σημαντικά πλεονεκτήματα. Παρ' όλα αυτά, δεν είναι η καταλληλότερη δομή για την αναπαράσταση ορισμένων σύνθετων εννοιών.

Καταρχάς, δεν μπορούμε να εκφράσουμε με πίνακες τη σύνδεση που έχουν σχετιζόμενες ποσότητες που δεν είναι ίδιου τύπου. Π.χ. ας υποθέσουμε ότι επιθυμούμε να περιγράψουμε σε πρόγραμμα για υπολογιστή, τη δανειστική λειτουργία μιας πανεπιστημιακής Βιβλιοθήκης· σε πρώτη φάση, θέλουμε να καταγράψουμε τα βιβλία που έχουν δανειστεί οι φοιτητές, να γνωρίζουμε σε ποια υπάρχει καθυστέρηση στην επιστροφή, κλπ. Έχουμε, επομένως, ένα σύνολο φοιτητών (που χαρακτηρίζονται από το όνομά τους, το Τμήμα στο οποίο ανήκουν, τον Αριθμό Μητρώου στη Σχολή τους, το έτος εισαγωγής τους, κλπ.), και ένα πλήθος βιβλίων (που χαρακτηρίζονται από τον τίτλο τους, το συγγραφέα, τον εκδοτικό οίκο, το έτος έκδοσης, κλπ.), που μπορούν να δανειστούν. Η απλοϊκή υλοποίηση σε κώδικα περιλαμβάνει δηλώσεις διανυσμάτων για την ομαδοποίηση του καθενός από τα χαρακτηριστικά που αναφέρθηκαν. Έτσι, για 500 φοιτητές και 3000 βιβλία θα είχαμε

```
INTEGER, PARAMETER :: N = 500    ! αριθμός φοιτητών
INTEGER, PARAMETER :: M = 3000   ! αριθμός βιβλίων
CHARACTER(30) :: studentName(N)
CHARACTER(50) :: bookAuthor(M)
CHARACTER(60) :: bookTitle(M)
INTEGER :: studentAM(N), studentYear(N), bookYear(M)
```

Παρατηρήστε ότι τίποτε στους παραπάνω ορισμούς δεν εκφράζει την σχέση που έχουν (ή δεν έχουν) τα στοιχεία των διανυσμάτων μεταξύ τους· τίποτε, εκτός πιθανόν από το όνομά τους, δεν υποδηλώνει ότι κάποια περιγράφουν χαρακτηριστικά της έννοιας «φοιτητής» και κάποια άλλα της έννοιας «βιβλίο». Προσέξτε ότι ένα σωστά επιλεγμένο όνομα μεταβλητής διευκολύνει αρκετά τον προγραμματιστή ή τον αναγνώστη στην κατανόηση του κώδικα, αλλά για τον compiler δεν έχει κάποιο ιδιαίτερο νόημα. Όχι μόνο δε γίνεται η ομαδοποίηση των χαρακτηριστικών στο επίπεδο των δύο βασικών εννοιών του προβλήματός μας (όνομα φοιτητή, αριθμός μητρώου, κλπ. μαζί και, ξεχωριστά, αλλά πάλι συγκεντρωμένα, ο τίτλος του βιβλίου, το όνομα του συγγραφέα, κλπ.) αλλά, χειρότερα, συνδέονται όλα τα ονόματα μαζί, όλοι οι αριθμοί μητρώου, κλπ. Η συγκεκριμένη οργάνωση των δεδομένων μας, και συνεπώς, και του υπόλοιπου κώδικά μας, είναι αρκετά διαφορετική από αυτή που υπαγορεύει το πρόβλημα που προσομοιώνουμε.

Μια καλύτερη, πιο «φυσική» προσέγγιση είναι να ορίσουμε νέους τύπους ποσοτήτων που θα μπορούν να περιγράψουν *συνολικά* τα χαρακτηριστικά κάθε έννοιας. Είναι προφανές ότι οι ενσωματωμένοι τύποι δεν ανταποκρίνονται σε αυτήν την απαίτηση: η έννοια «φοιτητής» δεν είναι ακέραιος ούτε πραγματικός, για μεταβλητή τέτοιου τύπου δεν ορίζεται η πρόσθεση ή εξαγωγή τετραγωνικής ρίζας!

Η Fortran 95 μας δίνει τη δυνατότητα να ορίσουμε νέους, σύνθετους τύπους. Αυτό επιτυγχάνεται με τη δομή **TYPE**, η σύνταξη της οποίας είναι

```
TYPE onoma
     Δήλωση μεταβλητής
     Δήλωση μεταβλητής
     Δήλωση μεταβλητής
     ......
END TYPE onoma
```

Η **TYPE** μπορεί να εμφανιστεί μόνο στο τμήμα των δηλώσεων του προγράμματός μας ή σε **MODULE** (§8.11) και περιλαμβάνει μεταξύ των **TYPE**...**END TYPE** μόνο δηλώσεις μεταβλητών. Ο τύπος των μεταβλητών μπορεί να είναι από τους ενσωματωμένους ή από αυτούς που όρισε ο προγραμματιστής σε προηγούμενο ορισμό **TYPE**. Αν το επιθυμούμε, μπορούμε να έχουμε στις δηλώσεις και αποδόσεις αρχικών τιμών.

Δήλωση ποσότητας αυτού του τύπου γίνεται ως εξής:

```
TYPE (onoma) :: x
```

Προσέξτε ότι κατά τον *ορισμό* του νέου τύπου το όνομά του δεν περικλείεται σε παρενθέσεις. Αντίθετα, στη *δήλωση* ποσότητας το όνομά του είναι εντός παρενθέσεων.

Βασιζόμενοι στα παραπάνω, μπορούμε να ορίσουμε νέους τύπους που περιλαμβάνουν ως *μέλη* κατάλληλους θεμελιώδεις τύπους, και αποτελούν ένα πρώτο βήμα βελτίωσης του προγράμματος, καθώς ακολουθούν τον τρόπο οργάνωσης των δεδομένων του προβλήματός μας:

```
TYPE student
```

```
   CHARACTER(30) :: studentName
   INTEGER       :: studentAM, studentYear
END TYPE student

TYPE book
   CHARACTER(50) :: bookAuthor
   CHARACTER(60) :: bookTitle
   INTEGER       :: bookYear
END TYPE book
```

Οι μεταβλητές που βρίσκονται σε διαφορετικούς παραγόμενους τύπους μπορούν να έχουν το ίδιο όνομα. Επομένως, μπορούμε να συντομεύσουμε τα ονόματά τους ως εξής

```
TYPE student
   CHARACTER(30) :: name
   INTEGER       :: am, year
END TYPE student

TYPE book
   CHARACTER(50) :: author
   CHARACTER(60) :: title
   INTEGER       :: year
END TYPE book
```

χωρίς να προκαλείται «σύγκρουση» ονομάτων.

Δήλωση ποσοτήτων αυτών των νέων τύπων γίνεται με τον ακόλουθο κώδικα

```
TYPE (student) :: s
TYPE (book)    :: b
INTEGER, PARAMETER :: N = 500   ! αριθμός φοιτητών
INTEGER, PARAMETER :: M = 3000  ! αριθμός βιβλίων
TYPE (student) :: students(N)
TYPE (book)    :: books(M)
```

Με τον κώδικα αυτό ορίσαμε δύο σύνθετες μεταβλητές, τις s και b, και δύο διανύσματα από student και book.

Πρόσβαση στα μέλη μια ποσότητας σύνθετου, παραγόμενου τύπου έχουμε με τη χρήση του τελεστή %. Έτσι, εκχώρηση τιμής σε μια σύνθετη ποσότητα μπορεί να γίνει ως εξής

```
TYPE (book) :: b

b%title = "Fortran 95/2003 explained"
b%author = "Michael Metcalf, John Reid, Malcolm Cohen"
b%year = 2004
```

Τα μέλη μιας σύνθετης ποσότητας συμπεριφέρονται όπως ακριβώς αν είχαν δηλωθεί ανεξάρτητα, απλώς το όνομά τους περιλαμβάνει το όνομα της ποσότητας στην οποία ανήκουν. Έτσι π.χ., η εκτύπωση γίνεται με τις εντολές

```
PRINT *, b%title
PRINT *, b%author
```

ενώ η συμμετοχή σε μία συνθήκη είναι

```
IF (b%year > 2003) THEN
   .....
END IF
```

Εκχώρηση τιμής σε μια σύνθετη ποσότητα γίνεται σε κάθε μέλος χωριστά, όπως είδαμε. Εναλλακτικά, μπορεί να της δοθεί τιμή ως σύνολο, με εκχώρηση άλλης ποσότητας του ίδιου τύπου

```
TYPE (book) :: b, c
b%title = "Fortran 95/2003 explained"
b%author = "Michael Metcalf, John Reid, Malcolm Cohen"
b%year = 2004
c = b
```

ή με εκχώρηση σταθερής ποσότητας αυτού του τύπου που δημιουργείται με σταθερά μέλη ως εξής:

```
TYPE (book) :: b
b = book("Michael Metcalf, John Reid, Malcolm Cohen", &
         "Fortran 95/2003 explained", 2004)
```

Στο δεξί μέλος της εντολής εκχώρησης εμφανίζεται το όνομα του τύπου ακολουθούμενου από σταθερές ποσότητες εντός παρενθέσεων, χωριζόμενων με κόμμα. Αυτές αντιστοιχούν σε κάθε μέλος του τύπου με τη σειρά που αυτό εμφανίζεται στον ορισμό.

Οι δύο παραπάνω τρόποι για την εκχώρηση τιμής ως σύνολο, μπορούν να εμφανίζονται και σε δήλωση με ταυτόχρονη εκχώρηση τιμής.

Επιπλέον, όπως και για πίνακες, μπορεί να δοθεί τιμή με ανάγνωση από το πληκτρολόγιο ή από αρχείο, ποσοτήτων κατάλληλης σειράς, πλήθους και τύπου όταν εκτελεστεί η εντολή **READ** για ολόκληρη τη δομή:

```
TYPE (book) :: b
READ *, b
```

Εκτός από την περίπτωση που έχουμε διαφορετικού τύπου ποσότητες που συνδέονται και, όπως αναλύσαμε, δεν μπορούμε να χρησιμοποιήσουμε πίνακα για να εκφράσουμε τη συσχέτισή τους, υπάρχει περίπτωση να έχουμε σχετιζόμενες ποσότητες του ίδιου τύπου.

Έστω, π.χ., ότι έχουμε ένα σύνολο 70 ημερομηνιών (ημέρα, μήνας, έτος) με τις αντίστοιχες θερμοκρασίες ενός τόπου. Η απλοϊκή περιγραφή του σε κώδικα θα ήταν

```
INTEGER, PARAMETER :: N = 70
INTEGER, DIMENSION (N) :: day, month, year
DOUBLE PRECISION :: temperature(N)
```

Όπως παρατηρούμε, τα τέσσερα διανύσματα είναι για το μεταγλωττιστή ισοδύναμα παρόλο που σαν έννοιες δεν είναι. Μια βελτίωση (αμφιβόλου αξίας!) θα ήταν η ακόλουθη

```
INTEGER, PARAMETER :: N = 70
INTEGER, DIMENSION (3,N) :: date
DOUBLE PRECISION :: temperature(N)
```

Με την παραπάνω δήλωση του date θα πρέπει να έχουμε πάντα στο μυαλό μας ότι στη θέση `date(1,j)` είναι η ημέρα της j μέτρησης, στη θέση `date(2,j)` είναι ο μήνας και στη θέση `date(3,j)` είναι το έτος (αρκεί να μη δώσουμε τον κώδικα σε Αμερικανό προγραμματιστή για να το συμπληρώσει, καθώς θα δυσκολευτεί από την «αντίστροφη» σειρά ημέρας και μήνα!). Με αυτή τη διόρθωση έχουμε διαφοροποιήσει τη ημερομηνία (date) από τον πίνακα των θερμοκρασιών (temperature), όμως, επιπλέον, έχουμε εισαγάγει «αφύσικη» σύνδεση μεταξύ διαφορετικών ημερομηνιών: τα στοιχεία `date(1,5), date(2,5), date(3,5), date(1,8)` για το πρόγραμμά μας είναι ισοδύναμα παρόλο που τα τρία πρώτα αναφέρονται στην ίδια ημερομηνία ενώ το τέταρτο σε άλλη.

Ουσιαστική βελτίωση μπορούμε να έχουμε με το ορισμό νέου τύπου, της ημερομηνίας (date):

```
TYPE date
   INTEGER :: day, month, year
END TYPE date
```

και νέου τύπου, αυτού της μέτρησης (measurement):

```
TYPE measurement
   TYPE (date) :: d
   DOUBLE PRECISION :: temperature
END TYPE measurement
```

Με τα παραπάνω, η κατάλληλη δήλωση για την οργάνωση των δεδομένων του προγράμματός μας είναι η ακόλουθη

```
TYPE (measurement) :: meas(70)
```

Η εκχώρηση τιμών στο στοιχείο j αυτού του διανύσματος γίνεται ως εξής

```
meas(j)%d%day = 21
meas(j)%d%month = 3
meas(j)%d%year = 2014
meas(j)%temperature = 24.5D0
```

6.1 Ασκήσεις

1. Γράψτε κώδικα που να ζητά από το χρήστη δύο διανύσματα τριών διαστάσεων σε καρτεσιανές συντεταγμένες. Το πρόγραμμά σας να υπολογίζει και να τυπώνει τα μέτρα των διανυσμάτων, το άθροισμα, τη διαφορά τους, το εσωτερικό και το εξωτερικό γινόμενο αυτών, τα μοναδιαία διανύσματά τους, το μέτρο της προβολής του πρώτου στο δεύτερο, καθώς και τη γωνία *σε μοίρες* μεταξύ τους.

   Χρησιμοποιήστε μεταβλητές παραγόμενου τύπου για να αποθηκεύσετε τα διανύσματα.

Κεφάλαιο 7

Είσοδος/Έξοδος Δεδομένων

7.1 Εισαγωγή

Μέχρι τώρα, χρησιμοποιούσαμε τις εντολές **READ**/**WRITE** (ή **PRINT**) για την είσοδο και έξοδο δεδομένων, αφήνοντας τον μεταγλωττιστή να αποφασίσει πώς θα κάνει τις αντίστοιχες διαδικασίες, βασιζόμενος στον *τύπο* των ποσοτήτων που εμφανίζονται στις εντολές. Αυτό είναι συχνά αρκετό· κάποιες φορές όμως, επιθυμούμε να έχουμε μεγαλύτερο έλεγχο σε σημεία όπως στο πόσα δεκαδικά ψηφία ενός πραγματικού θα τυπώνονται, αν θα αλλάζει γραμμή μετά την εκτύπωση, κλπ. Για αυτό το σκοπό πρέπει να προσδιορίζουμε λεπτομερώς στις εντολές εισόδου/εξόδου τι και πώς θα μεταφερθεί. Η πληροφορία αυτή δίνεται είτε απευθείας στις εντολές εισόδου/εξόδου με μια σειρά χαρακτήρων, §2.6.1, είτε εμμέσως με τη δήλωση **FORMAT**. Το περιεχόμενο της σειράς χαρακτήρων ή του **FORMAT** θα το περιγράψουμε παρακάτω.

Ο απευθείας προσδιορισμός του **FORMAT** ως σειρά χαρακτήρων γίνεται ως εξής

```
READ (*, "(......)") a, b, i
WRITE (*, "(......)") f, g, hhhh
```

ή, ισοδύναμα,

```
READ "(......)", a, b, i
PRINT "(.......)", f, g, hhhh
```

Δηλαδή, το δεύτερο '*' στα **READ**(*,*)/**WRITE**(*,*) ή το μόνο '*' στην περίπτωση των **READ** */**PRINT** *, το οποίο άφηνε ελεύθερο το μεταγλωττιστή να αποφασίσει τις λεπτομέρειες της μεταφοράς των δεδομένων, έχει αντικατασταθεί από τη σειρά χαρακτήρων. Σημειώστε ότι ο πρώτος και ο τελευταίος χαρακτήρας της είναι παρενθέσεις. Τα εισαγωγικά που την περικλείουν μπορεί να είναι απλά ή διπλά, όπως σε κάθε σειρά χαρακτήρων.

Εναλλακτικά, μπορούμε να προσδιορίσουμε τη διαμόρφωση σε ξεχωριστή δήλωση με την εντολή **FORMAT**. Η συγκεκριμένη εντολή αριθμείται με αριθμό ενός έως πέντε ψηφίων πριν τη λέξη **FORMAT** και αυτός ο αριθμός μπαίνει αντί για το '*' στις εντολές εισόδου/εξόδου. Η εντολή **FORMAT** μπορεί να εμφανίζεται σε οποιοδήποτε σημείο του κυρίως προγράμματος, υποπρογράμματος ή module στο οποίο θα χρησιμοποιηθεί, *πριν* το **END** του κυρίως προγράμματος ή υποπρογράμματος ή *πριν* το **CONTAINS** του module. Συνήθως τοποθετείται είτε αμέσως μετά την εντολή μεταφοράς δεδομένων είτε ακριβώς πριν το **END** ή το **CONTAINS**. Μια αριθμημένη εντολή **FORMAT** μπορεί να χρησιμοποιηθεί σε καμία, μία ή και περισσότερες εντολές **READ**/**WRITE** μέσω του αριθμού της, ο οποίος είναι της επιλογής του προγραμματιστή αλλά μοναδικός στο τμήμα του προγράμματος (κυρίως πρόγραμμα, συνάρτηση, υπορουτίνα, module) στο οποίο ορίζεται. Οι εντολές μεταφοράς δεδομένων στον παραπάνω κώδικα μπορούν να γραφούν ισοδύναμα

```
READ (*, 12) a, b, i
12 FORMAT (......)

WRITE (*, 25) f, g, hhhh
25 FORMAT (.......)
```

και

```
READ 12, a, b, i
12 FORMAT (......)

PRINT 25, f, g, hhhh
25 FORMAT (.......)
```

Παρατηρείστε ότι μετά τη λέξη **FORMAT** ακολουθεί το περιεχόμενο της σειράς χαρακτήρων χωρίς τα εισαγωγικά.

7.2 Περιεχόμενο του FORMAT

Στα παρακάτω θα περιγράψουμε τι περιλαμβάνεται εντός των παρενθέσεων σε εντολή **FORMAT** ή εντός εισαγωγικών και παρενθέσεων στον άμεσο προσδιορισμό στο **READ**/**WRITE**.

Για κάθε ποσότητα που θα εκτυπωθεί ή θα διαβαστεί, πρέπει να προσδιορίσουμε στο **FORMAT** τον τύπο της και το μέγιστο πλήθος των ψηφίων της, πόσες θέσεις, δηλαδή, θα χρειαστούν για να γίνει η μεταφορά.

Επομένως, για

Ακέραιη ποσότητα χρησιμοποιούμε το χαρακτήρα 'I' ακολουθούμενο από το εύρος του πεδίου w, δηλαδή το πλήθος των θέσεων. Η γενική μορφή είναι Iw. Με την εντολή

```
WRITE (*, "(I5)") 12
```

τυπώνεται το

␣␣␣12

Η εκτύπωση γίνεται σε όσες θέσεις προσδιορίζει το εύρος και είναι στοιχισμένη στα δεξιά. Αν το εύρος είναι μεγαλύτερο από το πλήθος των ψηφίων, *συμπεριλαμβανόμενου και του προσήμου*, ο αριθμός συμπληρώνεται με κενά. Αν είναι μικρότερο, ο *ακέραιος δεν τυπώνεται* αλλά εμφανίζεται στο πεδίο w φορές ο χαρακτήρας *. Μπορούμε να προσδιορίσουμε ότι το εύρος είναι 0, δηλαδή να δώσουμε στο format `I0`· αυτό υποδηλώνει ότι ο μεταγλωττιστής θα επιλέξει το ελάχιστο εύρος του πεδίου ώστε ο αριθμός να εκτυπωθεί κανονικά.

Κατά την ανάγνωση ενός ακεραίου πρέπει να λαμβάνουμε υπόψη στον προσδιορισμό του εύρους και τα κενά που προηγούνται. Έτσι, αν οι χαρακτήρες

␣␣␣-345

δοθούν ως είσοδος σε εντολή ανάγνωσης, αυτή πρέπει να έχει format με εύρος τουλάχιστον 7, ώστε να διαβαστεί:

```
READ (*, "(I7)") i
```

Κατά την εκτύπωση μπορούμε να προσδιορίσουμε και το ελάχιστο πλήθος ψηφίων που θέλουμε εκτυπωθεί, n, με το format Iw.n. Αν ο αριθμός δεν έχει τουλάχιστον n ψηφία, συμπληρώνεται με 0. Επομένως, n εντολή

```
WRITE (*, "(I5.3)") 99
```

τυπώνει το

␣␣099

Πραγματική ποσότητα χρησιμοποιούμε έναν από τους χαρακτήρες (ή συνδυασμούς χαρακτήρων) 'E', 'EN', 'ES', 'F', 'D' ακολουθούμενο από το εύρος του πεδίου, w, και το πλήθος d των δεκαδικών ψηφίων (ψηφίων μετά την τελεία) που θα εμφανιστούν. Η μορφή είναι Ew.d, ENw.d, ESw.d, Fw.d, Dw.d. Το w υποδηλώνει το πλήθος των θέσεων στις οποίες θα τυπωθεί ο αριθμός, μετρώντας και το τυχόν πρόσημο και την τελεία. Στην ανάγνωση δεδομένων το d αγνοείται. Οι χαρακτήρες στο **FORMAT** είναι ισοδύναμοι κατά την ανάγνωση αλλά προκαλούν διαφορετική εκτύπωση.

Ειδικότερα:

Ο χαρακτήρας 'F' τυπώνει τον αριθμό σε δεκαδική μορφή. Έτσι, το

```
WRITE (*,"(F9.2)") -146.29d0
```

τυπώνει

␣␣-146.29

Για διευκόλυνση, στην εκτύπωση μπορούμε να προσδιορίσουμε ότι το εύρος είναι 0, δηλαδή να δώσουμε στο format για εύρος w το 0. Έτσι υποδηλώνουμε ότι ο μεταγλωττιστής θα πρέπει να επιλέξει το ελάχιστο εύρος του πεδίου ώστε ο αριθμός να εκτυπωθεί κανονικά.

Οι χαρακτήρες 'E', 'EN', 'ES' στο **FORMAT** προκαλούν την εκτύπωση του πραγματικού αριθμού στη μορφή που περιέχει το E ακολουθούμενου από τη δύναμη του 10. Πιο συγκεκριμένα, ο χαρακτήρας 'E' τυπώνει στη μορφή $\pm 0.xxxxE \pm yy$ ή $\pm 0.xxxxE \pm yyy$ ή ακόμα και $\pm 0.xxxx \pm yyy$ (αν ο εκθέτης yyy είναι μεγαλύτερος από 99). Η εντολή

```
WRITE (*, "(E10.2)") 123D5
```

τυπώνει

␣␣0.12E+08

Οι χαρακτήρες 'ES' εκτυπώνουν παρόμοια με το 'E' αλλά ο αριθμός πριν το E είναι κατ' απόλυτη τιμή μεταξύ του 1 και του 10. Η εντολή

```
WRITE (*, "(ES10.2)") 123D5
```

τυπώνει

␣␣1.23E+07

Οι χαρακτήρες 'EN' εκτυπώνουν παρόμοια με το 'E' αλλά ο εκθέτης είναι πολλαπλάσιο του 3 και ο αριθμός πριν το E είναι κατ' απόλυτη τιμή μεταξύ του 1 και του 1000. Η εντολή

```
WRITE (*, "(EN10.2)") 123D5
```

τυπώνει

␣␣12.30E+06

Ο χαρακτήρας D προκαλεί εκτύπωση όπως ακριβώς ο E αλλά τυπώνει το γράμμα D για να εισαγάγει το τμήμα του αριθμού με τη δύναμη του 10.

Εάν στην είσοδο του προγράμματος δοθεί ο αριθμός

␣9.3729

η ανάγνωσή του μπορεί να γίνει με το ακόλουθο **READ**:

```
READ (*, "(F8.2)") x    !   ή E8.2 ή EN8.2 ή ES8.2
```

Αν η είσοδος είναι

```
␣93.729E-1
```

το **READ** που χρειάζεται πρέπει να έχει εύρος τουλάχιστον 10 και είναι

```
READ (*, "(F10.1)") x !   ή E10.1 ή EN10.1 ή ES10.1
```

Όπως και στους ακεραίους, το εύρος πεδίου στην εκτύπωση με τον χαρακτήρα 'F' (μόνο), μπορεί να δοθεί 0 ώστε ο μεταγλωττιστής να επιλέξει το κατάλληλο για τη σωστή μεταφορά του αριθμού.

Μιγαδική ποσότητα χρησιμοποιούμε δύο format για πραγματικούς, χωριζόμενα με κόμμα. Π.χ. η εντολή

```
WRITE (*,"(F6.1,E8.1)") (0.1d0, 100.0d0)
```

τυπώνει

```
␣␣␣0.1␣0.1E+03
```

Όπως θα δούμε λίγο παρακάτω, μεταξύ των δύο format (που δεν είναι απαραιτήτως ίδια) μπορούμε να έχουμε σειρά χαρακτήρων (μέσα σε εισαγωγικά) που θα τυπωθεί αυτούσια: Π.χ. το

```
WRITE (*,"(F6.1,'+i',E8.1)") (0.1d0, 100.0d0)
```

τυπώνει

```
␣␣␣0.1+i␣0.1E+03
```

Λογική ποσότητα χρησιμοποιούμε τον χαρακτήρα 'L' ακολουθούμενο από το εύρος πεδίου, w. Η εκτύπωση των .**TRUE**. και .**FALSE**. μεταφέρει τους χαρακτήρες 'T' και 'F' αντίστοιχα. Οι συγκεκριμένοι χαρακτήρες αρκούν στην ανάγνωση για να αποδοθούν οι αντίστοιχες τιμές.

Ποσότητα Χαρακτήρα ή Σειρά Χαρακτήρων χρησιμοποιούμε το χαρακτήρα 'A', προαιρετικά ακολουθούμενο από το εύρος πεδίου. Αν αυτό δεν προσδιορίζεται ρητά, η εκτύπωση ή η ανάγνωση γίνεται με όσο εύρος χρειάζεται για να υπάρξει σωστή και πλήρης μεταφορά. Π.χ. η εντολή

```
WRITE (*,"(A)") "This is a statement"
```

τυπώνει

```
This␣is␣a␣statement
```

Για οποιαδήποτε ενσωματωμένη ποσότητα μπορεί να χρησιμοποιηθεί ο χαρακτήρας 'G' ακολουθούμενος από το εύρος του πεδίου, w, και το πλήθος των ψηφίων μετά την τελεία, d, στη μορφή Gw.d. Για ποσότητες ακέραιες, λογικές ή χαρακτήρα ισοδυναμεί με Iw, Lw, Aw ενώ για πραγματικό ισοδυναμεί με το Ew.d εκτός αν ο αριθμός είναι κατάλληλα μικρός και μπορεί να εκτυπωθεί με κάποιο αντίστοιχο 'F'.

Ποσότητα με τύπο που δημιουργήθηκε από τον προγραμματιστή (Κεφάλαιο 6), μπορεί να διαβαστεί ή να εκτυπωθεί συνολικά, αρκεί να παραθέσουμε χωριζόμενα με κόμματα, τα επιθυμητά format για τα μέλη της, με τη σειρά που εμφανίζονται οι δηλώσεις τους στο **TYPE**.

Αν η εκτύπωση ή ανάγνωση περιλαμβάνει πολλές ποσότητες, παραθέτουμε τα αντίστοιχα format χωριζόμενα με κόμμα. Μεταξύ δύο format μπορούμε να έχουμε σειρά χαρακτήρων που θα τυπωθεί αυτούσια. Πρέπει να χωρίζεται με κόμματα από τα format και βέβαια, να βρίσκεται εντός διαφορετικών εισαγωγικών από το συνολικό **FORMAT**. Συνεχόμενα ίδια format προσδιορισμού μπορούν να γραφούν με συντομία παραθέτοντας μόνο ένα από αυτά, πιθανώς εντός παρενθέσεων, αμέσως μετά από ένα αριθμό που προσδιορίζει το πλήθος τους. Έτσι, το

```
100 FORMAT (I4,I4,I4)
```

ισοδυναμεί με

```
100 FORMAT (3I4)
```

ενώ το

```
200 FORMAT (I4,F4.2,I4,F4.2)
```

μπορεί να γραφεί

```
200 FORMAT (2(I4,F4.2))
```

Παρατήρηση

Ο προσδιορισμός του format κατά την ανάγνωση δεδομένων θέλει ιδιαίτερη προσοχή ώστε να γίνει σωστά η μεταφορά των τιμών. Συνήθως είναι προτιμότερο να επιλέγεται η χρήση του '∗' αντί για **FORMAT** στην ανάγνωση ώστε να ελέγχεται η μεταφορά των τιμών από τον μεταγλωττιστή.

7.2.1 FORMAT ελέγχου

Εκτός από τους χαρακτήρες που είδαμε για κάθε τύπο ποσότητας, σε ένα **FORMAT** μπορούμε να συμπεριλάβουμε (μεταξύ άλλων)

- σειρές χαρακτήρων, οι οποίες τυπώνονται αυτούσιες,
- τον χαρακτήρα '/', που προκαλεί αλλαγή γραμμής,
- τους χαρακτήρες 'nX', όπου n ακέραιος αριθμός. Το format αυτό προκαλεί την εκτύπωση n κενών.

Επίσης, αν επιθυμούμε να εμφανίζεται το πρόσημο '+' κατά την εκτύπωση θετικών αριθμών, πρέπει στο **FORMAT** να περιλάβουμε τους χαρακτήρες 'sp', πριν το format των αριθμών. Η εκτύπωση θετικών αριθμών χωρίς πρόσημο επαναφέρεται με τους χαρακτήρες 'ss' ή όταν τελειώσει το **FORMAT**. Έτσι, οι εντολές

```
WRITE (*,"(F8.2,sp,F8.2)") 5.12d0, 2.3d0
WRITE (*,"(F8.2)") 9.16d0
WRITE (*,"(F8.2,sp,F8.2,F5.1,ss,F5.1)") 5.12d0, 2.3d0, 4.1d0, 3.5d0
```

τυπώνουν

```
␣␣␣␣5.12␣␣␣+2.30
␣␣␣␣9.16
␣␣␣␣5.12␣␣␣+2.30␣+4.1␣␣3.5
```

7.2.2 Αλλαγή γραμμής

Στην περίπτωση που δεν επιθυμούμε να αλλάξει η γραμμή μετά την εκτύπωση, συμπληρώνουμε την εντολή **WRITE** με το **advance**="no". Προσέξτε ότι μπορούμε να το κάνουμε μόνο αν προσδιορίζεται συγκεκριμένο **FORMAT** για την εκτύπωση (και όχι απλώς το '*'). Π.χ.

```
WRITE (*, "(A)", advance="no") "Give number:"
READ (*,*) x
```

Εξαρτάται από το μεταγλωττιστή πόσους χαρακτήρες μπορούμε να τυπώσουμε στην ίδια γραμμή. Το όριο είναι συνήθως πολύ μεγάλο και κανονικά δεν το φτάνουμε.

7.3 Αρχεία

Μέχρι τώρα, η είσοδος και η έξοδος δεδομένων γινόταν από το πληκτρολόγιο και την οθόνη αντίστοιχα. Μπορούμε, όμως, να χρησιμοποιήσουμε και εξωτερικά ή εσωτερικά αρχεία.

Παρακάτω θα αναφερθούμε μόνο στα αρχεία σειριακής προσπέλασης.

7.3.1 Εξωτερικά Αρχεία

Τα εξωτερικά αρχεία είναι αυτά που μας παρέχει το λειτουργικό σύστημα του υπολογιστή και αποθηκεύονται συνήθως στο σκληρό δίσκο. Για να μεταφέρουμε δεδομένα από και προς ένα τέτοιο αρχείο πρέπει *πρώτα να συνδεθούμε* με αυτό. Η εντολή **OPEN** κάνει αυτό ακριβώς. Η σύνταξή της είναι

```
OPEN(unit = u, file = fname, status = stat, position = pos, &
     action = act)
```

Το **unit**= είναι το μόνο υποχρεωτικό όρισμα. Σε αυτό δίνουμε έναν ακέραιο μη αρνητικό αριθμό u της επιλογής μας, μοναδικό για κάθε αρχείο, ο οποίος το προσδιορίζει μονοσήμαντα. Η μέγιστη τιμή αυτού του αριθμού καθορίζεται από τον μεταγλωττιστή· μια συνήθης τιμή για το άνω όριο είναι το 99. Όπως θα δούμε, το **file**= είναι επίσης απαραίτητο σε όλες σχεδόν τις περιπτώσεις. Σε αυτό δίνουμε το όνομα του αρχείου ως σειρά χαρακτήρων, σταθερή ή μεταβλητή. Έτσι, αν επιθυμούμε να ανοίξουμε το αρχείο με όνομα *input.dat* και να το αντιστοιχήσουμε στον αριθμό 12, αρκεί να εκτελέσουμε την εντολή

```
OPEN(unit = 12, file = "input.dat")
```

ή λίγο πιο απλά

```
OPEN(12, file = "input.dat")
```

χωρίς να είναι απαραίτητο να προσδιορίσουμε τίποτε επιπλέον.

Τα υπόλοιπα ορίσματα, αν υπάρχουν, μπορούν να πάρουν συγκεκριμένες τιμές. Αυτές είναι:

- Στο **status**=**stat**, το stat είναι ένα από τα "old", "new", "replace", "scratch", "unknown". Προσδιορίζουν αντίστοιχα ότι το αρχείο

  1. είναι παλαιό (υπάρχει ήδη),
  2. είναι καινούργιο (δεν υπάρχει και θα δημιουργηθεί),
  3. είναι είτε παλαιό και θα αντικατασταθεί, είτε καινούργιο και θα δημιουργηθεί,
  4. είναι για προσωρινή χρήση (και δεν επιτρέπεται να δίνεται όνομα με το **file**=),
  5. είναι άγνωστης κατάστασης.

  Αν δεν προσδιορίσουμε κάποιο από αυτά, ο μεταγλωττιστής θεωρεί ότι δώσαμε το **status**="unknown". Προσέξτε ότι αν δηλώσουμε ότι **status**="old" και το αρχείο δεν υπάρχει κατά την εκτέλεση του προγράμματος ή δηλώσουμε ότι **status**="new" και το αρχείο υπάρχει ήδη, θα εμφανιστεί σχετικό μήνυμα και διακοπή της εκτέλεσης του προγράμματος.

- Στο **position**=pos, το pos είναι ένα από τα

  1. "rewind", που τοποθετεί το δείκτη για είσοδο ή έξοδο δεδομένων στην αρχή του αρχείου,
  2. "append", που τοποθετεί το δείκτη στο τέλος του, ή
  3. "asis" που θέτει το δείκτη στην αρχή αν το αρχείο είναι νέο, ή εκεί που βρίσκεται αν έχει ήδη συνδεθεί με άλλο **OPEN**. Αυτή είναι η προκαθορισμένη τιμή αν παραληφθεί το **position**=.

- Στο **action**=act οι τιμές του act είναι ”read”, ”write” ή ”readwrite” αν θα χρησιμοποιηθεί μόνο για είσοδο δεδομένων, ή μόνο για έξοδο ή και για τα δυο, αντίστοιχα.

Π.χ. η σύνδεση στον αριθμό 34, του αρχείου με όνομα *data*, που υπάρχει ήδη και θα χρησιμοποιηθεί για είσοδο δεδομένων, γίνεται με την εντολή

```
OPEN (unit = 34, file ="data", status = "old", action = "read")
```

Αν επιθυμούμε να εκτελέσουμε ένα **READ** ή **WRITE** στο συγκεκριμένο αρχείο, βάζουμε τον αριθμό του σαν πρώτο όρισμα στην παρένθεση της εντολής αντί για το ‘∗’, δηλαδή

```
READ (unit = 34, fmt = "(F8.2)") x
```

ή, ισοδύναμα και πιο απλά,

```
READ (34, "(F8.2)") x
```

Αφού ολοκληρωθεί η χρήση του αρχείου πρέπει *να το αποσυνδέσουμε*. Αυτό επιτυγχάνεται με την εντολή

```
CLOSE (unit = u)
```

ή, ισοδύναμα και πιο απλά

```
CLOSE (u)
```

όπου u ο αριθμός του συγκεκριμένου αρχείου. Κατόπιν, ο αριθμός αυτός είναι ελεύθερος να ξαναχρησιμοποιηθεί για άλλο αρχείο.

Προσέξτε ότι αν προσδιορίσουμε στην εντολή **READ** λιγότερες μεταβλητές για ανάγνωση από όσα στοιχεία περιέχει το αρχείο στη γραμμή που πρόκειται να διαβαστεί, οι *επιπλέον τιμές αγνοούνται*. Η επόμενη εντολή **READ** διαβάζει από την επόμενη γραμμή. Αν προσδιορίσουμε περισσότερες μεταβλητές από όσα στοιχεία περιέχονται στη γραμμή, το διάβασμα συνεχίζει στην επόμενη γραμμή του αρχείου (και αγνοούνται όσες τιμές περισσεύουν).

Παρατηρήσεις

- Ο προσδιορισμός του ονόματος και της θέσης του αρχείου στο σύστημα αρχείων («filesystem») μπορεί να γίνει με τη χρήση σχετικής ή απόλυτης διαδρομής («path»), όπως την υπαγορεύει το λειτουργικό σύστημα.
- Υπάρχουν συγκεκριμένοι αριθμοί αρχείων που υποδηλώνουν την είσοδο από το πληκτρολόγιο και την έξοδο στην οθόνη. Αυτοί συνήθως είναι το 5 και το 6 αντίστοιχα. Το 0 αντιστοιχεί στο αρχείο που διοχετεύονται τα μηνύματα λάθους του προγράμματος (δηλαδή συνήθως την οθόνη). Καλό είναι να αποφεύγεται η χρήση των αριθμών αυτών για τη σύνδεση δικών μας αρχείων.
- Μπορούμε να έχουμε συνδεδεμένα ταυτόχρονα πολλά αρχεία. Εννοείται πως θα αντιστοιχούν σε διαφορετικούς αριθμούς.

7.3.2 Εσωτερικά Αρχεία

Τα εσωτερικά αρχεία είναι *σειρές χαρακτήρων* που δημιουργούνται από το πρόγραμμά μας με τη γνωστή δήλωση, §2.6.1, και έχουν διάρκεια ζωής το χρόνο εκτέλεσης του προγράμματος, όπως κάθε μεταβλητή. Μπορούμε να επιλέξουμε το μέγεθός τους ώστε να επαρκούν για οτιδήποτε θελήσουμε να γράψουμε σε αυτά. Δε χρειάζονται σύνδεση και αποσύνδεση, η δήλωσή τους είναι η μόνη αναγκαία πριν τη χρήση τους. Χρησιμοποιούνται απευθείας σε εντολές **READ**/**WRITE**, με το όνομά τους στη θέση του πρώτου '*'. Παράδειγμα χρήσης, και ταυτόχρονα μια σημαντική εφαρμογή των εσωτερικών αρχείων, είναι το ακόλουθο

```
CHARACTER(50) :: str
INTEGER :: i

i = 3
WRITE (str, "(A,I0,A)") "fname_", i, ".dat" ! Το str είναι "fname_3.dat"
```

Η εγγραφή των τριών δεδομένων με το προσδιοριζόμενο **FORMAT** γίνεται στη σειρά χαρακτήρων str. Αυτή κατόπιν μπορεί να χρησιμοποιηθεί για τον προσδιορισμό του ονόματος αρχείου σε επόμενο **OPEN**:

```
OPEN(unit = 43, file = str)
```

Με τις προηγούμενες εντολές, αντιστοιχούμε τον αριθμό 43 στο αρχείο *fname_3.dat*.

Παρατηρήστε ότι με τη χρήση εσωτερικών αρχείων μπορούμε να συνδέσουμε στο πρόγραμμά μας εξωτερικά αρχεία με όνομα που παράγεται από τιμές δεδομένων και μεταβλητών κατά την εκτέλεση του προγράμματος.

7.4 Ασκήσεις

1. Γράψτε πρόγραμμα που θα τυπώνει σε αρχείο με όνομα *prime.dat* όλους τους πενταψήφιους αριθμούς που είναι πρώτοι.

2. Να βρείτε 4 *διαδοχικούς* θετικούς ακέραιους αριθμούς n, $n+1$, $n+2$, $n+3$, τέτοιους ώστε ο πρώτος να είναι πολλαπλάσιο του 5, ο δεύτερος πολλαπλάσιο του 7, ο τρίτος πολλαπλάσιο του 9 και ο τέταρτος πολλαπλάσιο του 11.

   Γράψτε στο αρχείο με όνομα *data* όλες τις τετράδες τέτοιων αριθμών για $n \leq 100000$. Τυπώστε στην οθόνη το πλήθος τους.

3. - Να δημιουργήσετε με πρόγραμμα ένα αρχείο με όνομα *trig.dat*. Να τυπώσετε σε αυτό το ημίτονο, το συνημίτονο και την εφαπτομένη των γωνιών από $0°$ έως $359.9°$ ανά $0.7°$. Οι αριθμοί που εκτυπώνονται να έχουν 4 δεκαδικά ψηφία και να είναι στοιχισμένοι σε τέσσερις στήλες: γωνία, ημίτονο, συνημίτονο και εφαπτομένη. Στην πρώτη γραμμή του αρχείου να γράψετε τον αριθμό των γραμμών που ακολουθούν.
   - Διαβάστε το αρχείο *trig.dat* με άλλο πρόγραμμα και βρείτε σε ποια από τις γωνίες του αρχείου αντιστοιχεί το μικρότερο άθροισμα ημιτόνου και συνημιτόνου.

4. Γράψτε πρόγραμμα που θα δέχεται από το πληκτρολόγιο ένα πραγματικό αριθμό x_0. Να φροντίσετε ώστε το πρόγραμμα να μην τον κρατά αν δεν είναι στο διάστημα $(0, 1)$, αλλά να ξαναζητά αριθμό, όσες φορές χρειαστεί. Κατόπιν, υπολογίστε και τυπώστε στο αρχείο με όνομα *random.txt* τους αριθμούς x_1, x_2, ..., x_{100}, όπου

$$\begin{aligned} x_1 &= |(100 \ln(x_0)) \bmod 1| , \\ x_2 &= |(100 \ln(x_1)) \bmod 1| , \\ &\vdots \quad\quad \vdots \\ x_{100} &= |(100 \ln(x_{99})) \bmod 1| . \end{aligned}$$

   Η έκφραση $a \bmod 1$ σημαίνει το *δεκαδικό μέρος* του a. Το $\ln(x)$ είναι ο φυσικός λογάριθμος. Προσέξτε ότι στον υπολογισμό του x_1 χρειάζεται ο x_0, στον υπολογισμό του x_2 χρειάζεται ο x_1, κλπ.[1]

5. Αποθηκεύστε στον υπολογιστή σας το αρχείο στη διεύθυνση https://tinyurl.com/yjyjnywh. Περιέχει 126 βαθμούς εξέτασης φοιτητών σε κάποιο μάθημα, τον καθένα σε ξεχωριστή γραμμή. Οι βαθμοί είναι πραγματικοί αριθμοί μεταξύ 0 και 10. Βρείτε και τυπώστε στην οθόνη πόσοι φοιτητές πήραν 0 και πόσοι έχουν βαθμό στα διαστήματα $(0, 1]$, $(1, 2]$, ..., $(9, 10]$.

[1] Οι αριθμοί x_i που προκύπτουν με αυτή τη μέθοδο είναι ψευδοτυχαίοι στο διάστημα $[0, 1)$.

6. Το αρχείο στη διεύθυνση https://tinyurl.com/bp7yzkdr περιέχει 5288 ακέραιους αριθμούς, σε ξεχωριστή γραμμή ο καθένας. Αποθηκεύστε το στον υπολογιστή σας. Γράψτε πρόγραμμα που να διαβάζει τους ακέραιους από αυτό το αρχείο και να αποθηκεύει τους ζυγούς στο αρχείο *even.dat* και τους μονούς στο *odd.dat*.

7. Γράψτε πρόγραμμα που να διαβάζει μήνα και έτος από τον χρήστη και να τυπώνει στο αρχείο με όνομα *calendar* τις ημέρες του μήνα με τη μορφή (παράδειγμα για Μάρτιο του 2014):

			03/2014			
ΔΕΥ	ΤΡΙ	ΤΕΤ	ΠΕΜ	ΠΑΡ	ΣΑΒ	ΚΥΡ
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Θα χρειαστεί να βρείτε:

(α΄) Ποια ημέρα (Δευτέρα, Τρίτη, ...) πέφτει η πρώτη του μηνός. Θα σας βοηθήσει ο αλγόριθμος του Zeller· δείτε την άσκηση 8 στη σελίδα 39.

(β΄) Πόσες ημέρες έχει ο συγκεκριμένος μήνας. Δείτε την άσκηση 9 στη σελίδα 39.

8. Να υπολογίσετε τους δεκαδικούς λογαρίθμους των αριθμών από 1.0 έως το 9.9 με βήμα 0.1. Να τυπώσετε τα αποτελέσματα με 3 δεκαδικά ψηφία με τη μορφή διδιάστατου πίνακα, όπως παρακάτω:

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	0.000	0.041	0.079	0.114	0.146	0.176	0.204	0.230	0.255	0.279
2	0.301	0.322	0.342	0.362	0.380	0.398	0.415	0.431	0.447	0.462
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
9	0.954	0.959	0.964	0.968	0.973	0.978	0.982	0.987	0.991	0.996

Η πρώτη στήλη έχει το ακέραιο μέρος ενός αριθμού ενώ η πρώτη γραμμή έχει το δεκαδικό μέρος. Το άθροισμα του πρώτου στοιχείου στη γραμμή i και του πρώτου στη στήλη j έχει λογάριθμο που δίνεται στην τομή τους, δηλαδή στο στοιχείο (i, j).

9. Δημιουργήστε τον πίνακα του Pascal. Ο πίνακας αυτός είναι διδιάστατος, $n \times n$, και έχει στοιχεία που ορίζονται από τις σχέσεις

$$\begin{aligned} P(i,1) = P(1,j) &= 1 && \text{για κάθε } i,\ j \text{ και} \\ P(i,j) &= P(i-1,j) + P(i,j-1) && \text{για } i,j > 1. \end{aligned}$$

Το κάθε «εσωτερικό» στοιχείο επομένως είναι το άθροισμα των προηγούμενων στη στήλη του και στη γραμμή του.

- Γράψτε κώδικα που να τυπώνει στην οθόνη τον πίνακα του Pascal. Οι αριθμοί να είναι στοιχισμένοι κατά στήλες και κάθε γραμμή του πίνακα να τυπώνεται σε ξεχωριστή γραμμή. Εφαρμόστε τον για $n = 7$.
- Τροποποιήστε το πρόγραμμα ώστε να εκτυπώνει τον πίνακα στο αρχείο *pascal.txt*.

10. Έστω η ακόλουθη διαδικασία για ένα θετικό ακέραιο αριθμό («αριθμός εισόδου»):

    - αν ο αριθμός είναι άρτιος τον διαιρούμε με το 2.
    - αν ο αριθμός είναι περιττός τον πολλαπλασιάζουμε με το 3 και προσθέτουμε 1.

    Ξεκινώντας από ένα αριθμό n, επαναλαμβάνουμε τη διαδικασία θεωρώντας το αποτέλεσμα κάθε επανάληψης ως αριθμό είσοδου της επόμενης. Σύμφωνα με την υπόθεση του Collatz, η επανάληψη αυτής της διαδικασίας θα δώσει ως αποτέλεσμα το 1 μετά από πεπερασμένο αριθμό βημάτων. Ο αριθμός βημάτων που θα χρειαστεί για αυτό εξαρτάται από τον αρχικό μας αριθμό n.

    Να γράψετε πρόγραμμα που

    - να τυπώνει στο αρχείο *collatz.dat* κάθε θετικό ακέραιο εισόδου από το 2 μέχρι το 100000 (πρώτη στήλη) μαζί με το αντίστοιχο πλήθος των βημάτων μέχρι να βγει αποτέλεσμα το 1 (δεύτερη στήλη).
    - Να τυπώνει στην οθόνη τον αριθμό εισόδου που είχε το μεγαλύτερο αριθμό βημάτων, μαζί με τον αριθμό βημάτων.

11. (α′) Δημιουργήστε με πρόγραμμα το αρχείο *random.txt* με 10000 τυχαίους ακεραίους στο διάστημα $[-20, 20]$.

    (β′) Γράψτε πρόγραμμα που να διαβάζει το αρχείο *random.txt* και να τυπώνει στην οθόνη πόσους θετικούς, αρνητικούς και ίσους με το 0 αριθμούς περιέχει.

12. Το αρχείο `https://tinyurl.com/bp7yzkdr` περιέχει 5288 ακέραιους αριθμούς, σε ξεχωριστή γραμμή ο καθένας. Αποθηκεύστε το στον υπολογιστή σας. Δημιουργήστε με πρόγραμμα ένα αρχείο με όνομα *rev.txt* στο οποίο να αντιγράψετε τους αριθμούς του πρώτου αρχείου *αντίστροφα* (ο πρώτος να γραφτεί στο τέλος και ο τελευταίος στην αρχή).

13. Σύμφωνα με την υπόθεση Lemoine, κάθε περιττός θετικός ακέραιος αριθμός μεγαλύτερος του 5, μπορεί να γραφεί ως άθροισμα ενός πρώτου αριθμού και του διπλάσιου ενός άλλου πρώτου αριθμού (χωρίς να είναι απαραίτητα

διαφορετικοί οι δύο πρώτοι αριθμοί)[2]. Να ελέγξετε αυτή την υπόθεση για κάθε περιττό αριθμό m από το 7 μέχρι το 999999: βρείτε τους πρώτους αριθμούς p, q που ικανοποιούν τη σχέση $m = p + 2q$. Γράψτε τους αριθμούς m, p, q, με ένα κενό ανάμεσά τους, στο αρχείο *lemoine.dat*, ώστε να έχετε την κάθε τριάδα σε ξεχωριστή γραμμή.

14. Η καρδιοειδής καμπύλη σε πολικές συντεταγμένες δίνεται από την εξίσωση

$$r(\theta) = 4\cos^2(\theta/2) \ .$$

Επιλέξτε 70 ισαπέχουσες γωνίες θ_i ($i = 1, \ldots, 70$) στο διάστημα 0° έως 359°. Τα άκρα του διαστήματος να συμπεριλαμβάνονται σε αυτές.

Τυπώστε σε δύο στήλες σε αρχείο με όνομα *cardioid.txt* τις γωνίες θ_i και τις αντίστοιχες τιμές της απόστασης $r(\theta_i)$. Κάθε ζεύγος $(\theta_i, r(\theta_i))$ να είναι στην ίδια γραμμή του αρχείου με ένα κενό ανάμεσα. Οι τιμές που θα τυπώσετε να είναι στοιχισμένες και να έχουν 4 δεκαδικά ψηφία.

15. Γράψτε στο αρχείο με όνομα *numbers*, σε ξεχωριστή γραμμή τον καθένα, τους ακέραιους n, με $0 \leq n < 10^6$, που έχουν άθροισμα ψηφίων ίσο με το άθροισμα των ψηφίων του $137n$.

16. Στις διευθύνσεις `https://tinyurl.com/55dtyewa` και `https://tinyurl.com/bdzb6kp6` παρέχονται δύο αρχεία που περιέχουν 1200 και 1600 ακέραιους αριθμούς αντίστοιχα, ένα σε κάθε γραμμή τους. Αποθηκεύστε τα αρχεία στον υπολογιστή σας. Να γράψετε πρόγραμμα που να «διαβάζει» τους αριθμούς του πρώτου αρχείου στο διάνυσμα a και τους αριθμούς του δεύτερου στο διάνυσμα b, και να γράφει στο αρχείο *fileC.txt* τους αριθμούς του a που *δεν περιέχονται* στο b, σε ξεχωριστή γραμμή τον καθένα.

17. Το αρχείο στη διεύθυνση `https://tinyurl.com/4e6u7nxa` περιέχει 4996 θετικούς ακέραιους αριθμούς που επαναλαμβάνονται, ο καθένας σε ξεχωριστή γραμμή. Αποθηκεύστε το στον υπολογιστή σας. Γράψτε πρόγραμμα που

    (α′) Να βρίσκει πόσες φορές εμφανίζεται στο αρχείο ο αριθμός 42.

    (β′) Να βρίσκει πόσες φορές εμφανίζεται στο αρχείο ο αριθμός στη γραμμή 42.

    (γ′) Να βρίσκει πόσοι είναι οι διαφορετικοί αριθμοί του αρχείου.

    (δ′) Να γράφει στο αρχείο *single.dat* τους αριθμούς που εμφανίζονται στο αρχικό αρχείο, χωρίς τις επαναλήψεις τους.

    (ε′) Να βρίσκει πόσες φορές επαναλαμβάνεται κάθε αριθμός. Αυτή την πληροφορία να τη γράφει στο αρχείο *freq.dat* σε δύο στήλες: στην πρώτη να είναι οι αριθμοί και στη δεύτερη τα αντίστοιχα πλήθη.

[2] το 1 δεν θεωρείται πρώτος αριθμός.

18. Σε ένα αρχείο με όνομα *freq.dat*, περιέχονται δύο στήλες ακέραιων αριθμών: σε κάθε γραμμή, ο δεύτερος αριθμός είναι ένα «βάρος» που δείχνει πόσο «ισχυρή» είναι κάποια ιδιότητα του πρώτου αριθμού. Γράψτε στο αρχείο *sort.dat* τους αριθμούς της πρώτης στήλης του αρχικού αρχείου, με σειρά από αυτόν με το μεγαλύτερο βάρος προς αυτόν με το μικρότερο βάρος.

    Μπορείτε να χρησιμοποιήσετε για *freq.dat* το ομώνυμο αρχείο της άσκησης 17.

19. Μια διαμόρφωση ενός αρχείου κειμένου που μπορεί να χρησιμοποιηθεί για την αποθήκευση ασπρόμαυρης εικόνας είναι η ακόλουθη:

    (α′) Η πρώτη γραμμή του αρχείου πρέπει να γράφει: P1.

    (β′) Η δεύτερη να γράφει τις διαστάσεις της εικόνας: πλάτος ύψος (δηλαδή τους δύο αριθμούς με κενό μεταξύ τους).

    (γ′) Να ακολουθούν οι αριθμοί 0 ή 1· αυτοί αντιπροσωπεύουν τα pixels της εικόνας *κατά γραμμές*, ξεκινώντας από επάνω αριστερά: κάθε λευκό pixel αντιστοιχεί στο 0 και κάθε μαύρο στο 1. Οι αριθμοί μπορούν να διαχωρίζονται από κενά αλλά δεν είναι απαραίτητο. Σε κάθε σειρά του αρχείου μπορούμε να έχουμε έως 70 χαρακτήρες.

    Η διαμόρφωση αυτή αποτελεί ένα αρχείο τύπου plain pbm (portable bitmap) που μπορούμε να το δούμε με προγράμματα απεικόνισης.

    Δημιουργήστε ένα πίνακα 512×512 στον οποίο τα στοιχεία που βρίσκονται μία θέση κάτω και μία θέση πάνω από τις δύο διαγωνίους, κύρια και δευτερεύουσα (δηλαδή, σε τέσσερις συγκεκριμένες γραμμές), θα έχουν την τιμή 1 και τα υπόλοιπα θα είναι 0. Έτσι, έχουμε αποθηκεύσει μια διδιάστατη εικόνα στον πίνακα. Τυπώστε την σε αρχείο με κατάληξη .ppbm και διαμόρφωση plain pbm.

20. Μια διαμόρφωση ενός αρχείου κειμένου που μπορεί να χρησιμοποιηθεί για την αποθήκευση εικόνας με αποχρώσεις του γκρι είναι η ακόλουθη:

    (α′) η πρώτη γραμμή του αρχείου πρέπει να γράφει: P2.

    (β′) η δεύτερη πρέπει να γράφει τις διαστάσεις της εικόνας: πλάτος ύψος (δηλαδή τους δύο αριθμούς με κενό μεταξύ τους).

    (γ′) η τρίτη πρέπει να έχει ένα θετικό ακέραιο αριθμό K που αντιπροσωπεύει τη μέγιστη τιμή του γκρι. Πρέπει να είναι μικρότερη από 256. Τυπική τιμή για αυτή είναι το 255.

    (δ′) ακολουθούν τα pixels της εικόνας *κατά γραμμές*, ξεκινώντας από επάνω αριστερά. Κάθε pixel αντιπροσωπεύεται από ένα ακέραιο αριθμό από το 0 έως και το K. Μεταξύ των τιμών πρέπει να υπάρχει ένα τουλάχιστον κενό ή αλλαγή γραμμής. Οι γραμμές του αρχείου πρέπει να έχουν έως 70 χαρακτήρες. Διευκολύνει επομένως αν κάθε pixel είναι γραμμένο σε ξεχωριστή γραμμή.

Η διαμόρφωση αυτή αποτελεί ένα αρχείο τύπου plain pgm (portable graymap) που μπορούμε να το δούμε με προγράμματα απεικόνισης.

Γράψτε ένα πρόγραμμα που θα διαβάζει το αρχείο *input.ppgm* και θα δημιουργεί μία νέα εικόνα στο *output.ppgm* ως εξής: Το pixel (i, j) στη νέα εικόνα θα είναι ο μέσος όρος του pixel (i, j) της αρχικής και των γειτονικών του (μέχρι γείτονες τάξης p). Επομένως, αν το (i, j) είναι μακριά από τα άκρα, τα pixels που χρησιμοποιούμε στον υπολογισμό είναι αυτά που βρίσκονται στο τετράγωνο με κορυφές τα $(i \pm p, j \pm p)$. Αν το (i, j) είναι στα άκρα, οι γείτονες είναι λιγότεροι (αυτοί που περιέχονται στο πλέγμα).

Το πρόγραμμά σας θα ζητά από το χρήστη τον ακέραιο θετικό αριθμό p. Για να το δοκιμάσετε, χρησιμοποιήστε το αρχείο στη διεύθυνση `https://tinyurl.com/2tvnp8p2`.

Παρατήρηση: Καθώς το πλήθος των pixels που έχουν γραφτεί σε κάθε γραμμή ενός αρχείου με τη παραπάνω διαμόρφωση δεν είναι γνωστό, η ανάγνωση των pixels πρέπει να γίνει *συνολικά*, για όλες τις γραμμές μαζί, με ένα **`READ`**.

21. Μια διαμόρφωση ενός αρχείου που μπορεί να χρησιμοποιηθεί για την αποθήκευση έγχρωμης εικόνας είναι η ακόλουθη:

    (α′) Η πρώτη γραμμή του αρχείου πρέπει να γράφει: P3.

    (β′) Η δεύτερη πρέπει να γράφει τις διαστάσεις της εικόνας: πλάτος ύψος (δηλαδή τους δύο αριθμούς με κενό μεταξύ τους).

    (γ′) Η τρίτη πρέπει να έχει ένα θετικό ακέραιο αριθμό K, μέχρι το 255, που αντιπροσωπεύει τη μέγιστη τιμή του κάθε χρώματος. Τυπική τιμή είναι το 255.

    (δ′) Να ακολουθούν τα pixels της εικόνας *κατά γραμμές*, ξεκινώντας από επάνω αριστερά. Στο αρχείο θα γράφονται οι τιμές των χρωμάτων «κόκκινο» (R), «πράσινο» (G), «μπλε» (B) για το κάθε pixel, με ένα κενό μεταξύ τους. Ένα pixel που έχει χρώμα κόκκινο θα αναπαρίσταται από την τριάδα $K\ 0\ 0$ (αν το K είναι 255 θα γράφουμε $255\ 0\ 0$). Το pixel με «πράσινο» χρώμα θα αντιστοιχεί στη γραμμή $0\ K\ 0$. Το μαύρο χρώμα είναι το $0\ 0\ 0$ ενώ το λευκό $K\ K\ K$. Το κίτρινο είναι $K\ K\ 0$. Σε κάθε συνιστώσα RGB μπορούμε γενικά να έχουμε οποιαδήποτε τιμή μεταξύ 0 και K ώστε να παράγουμε όλα τα χρώματα.

    Οι γραμμές του αρχείου πρέπει να έχουν έως 70 χαρακτήρες.

    Η διαμόρφωση αυτή αποτελεί ένα αρχείο τύπου plain ppm (portable pixmap) που μπορούμε να το δούμε με προγράμματα απεικόνισης.

    Δημιουργήστε ένα αρχείο με όνομα *france.pppm* με τη σημαία της Γαλλίας[3], σε 512×768 pixels, χρησιμοποιώντας την παραπάνω διαμόρφωση.

[3] τρεις κατακόρυφες λωρίδες ίσου πλάτους: μπλε, λευκή, κόκκινη.

22. Γράψτε πρόγραμμα που θα διαβάζει την εικόνα που είναι αποθηκευμένη με διαμόρφωση ppgm (δείτε την περιγραφή της διαμόρφωσης στην άσκηση 20), στο αρχείο *input.ppgm*, και θα την γράφει στο αρχείο *output.ppgm*, με ίδια διαμόρφωση, αφού την περιστρέψει κατά τη φορά των δεικτών του ρολογιού κατά $90°$.

    Υπόδειξη: Σε μια εικόνα με M γραμμές και N στήλες, η δεξιόστροφη περιστροφή κατά $90°$ δημιουργεί εικόνα με N γραμμές και M στήλες και μετακινεί το pixel (i, j) στο pixel $(j, M - 1 - i)$, με $0 \leq i < M$, $0 \leq j < N$.

    Για να δοκιμάσετε το πρόγραμμά σας χρησιμοποιήστε το αρχείο στη διεύθυνση `https://tinyurl.com/2tvnp8p2`.

23. Γράψτε ένα πρόγραμμα που να υλοποιεί το Game of Life[4] του Dr. J. Conway. Αυτό προσομοιώνει την εξέλιξη ζωντανών οργανισμών βασιζόμενο σε συγκεκριμένους κανόνες.

    Σε ένα πλέγμα $M \times N$, κάθε τετράγωνο έχει οκτώ πρώτους γείτονες (λιγότερους αν βρίσκεται στα άκρα). Τοποθετούμε σε τυχαίες θέσεις K οργανισμούς. Σε κάθε βήμα της εξέλιξης (νέα γενιά):

    (α΄) Ένα κενό τετράγωνο με ακριβώς τρεις «ζωντανούς» γείτονες γίνεται «ζωντανό» (γέννηση).

    (β΄) Ένα «ζωντανό» τετράγωνο με δύο ή τρεις «ζωντανούς» γείτονες παραμένει ζωντανό (επιβίωση).

    (γ΄) Σε κάθε άλλη περίπτωση ένα τετράγωνο γίνεται ή παραμένει κενό δηλαδή «πεθαίνει» ή παραμένει «νεκρό» (από υπερπληθυσμό ή μοναξιά!).

    Η «αποθήκευση» της επόμενης γενιάς γίνεται αφού ολοκληρωθεί ο υπολογισμός της για όλα τα τετράγωνα.

    Να τυπώνετε την κάθε γενιά σε αρχεία τύπου plain pbm (δείτε την περιγραφή της διαμόρφωσης στην άσκηση 19), ώστε να μπορείτε να τις δείτε όλες μαζί διαδοχικά[5].

    Σχηματίστε τετραγωνικό πλέγμα 512×512 και υπολογίστε 1000 γενιές. Δοκιμάστε να τοποθετήσετε αρχικά τους οργανισμούς όχι σε τυχαίες θέσεις αλλά σε μία θέση κάτω και μία θέση πάνω από τις δύο διαγωνίους, κύρια και δευτερεύουσα (δηλαδή, σε τέσσερις συγκεκριμένες γραμμές).

24. Ένα μυρμήγκι (Langton's ant[6]) βρίσκεται σε ορθογώνιο πλέγμα από 512×512 τετράγωνα. Τα τετράγωνα μπορούν να είναι είτε άσπρα είτε μαύρα. Αρχικά είναι όλα άσπρα. Το μυρμήγκι έχει αρχική θέση το κέντρο του πλέγματος (το

[4] `http://www.math.com/students/wonders/life/life.html`

[5] Σε συστήματα UNIX, με εγκατεστημένο το πρόγραμμα imagemagick, η εντολή είναι `animate *.ppbm`

[6] `http://mathworld.wolfram.com/LangtonsAnt.html`

σημείο $(256, 256)$), κατεύθυνση προς τα επάνω και κινείται σε κάθε βήμα του σύμφωνα με τους ακόλουθους κανόνες:

- Αν βρίσκεται σε μαύρο τετράγωνο, αλλάξει το χρώμα του τετραγώνου σε άσπρο, στρέφει *αριστερά* κατά 90° και προχωρά κατά ένα τετράγωνο.
- Αν βρίσκεται σε άσπρο τετράγωνο, αλλάξει το χρώμα του τετραγώνου σε μαύρο, στρέφει *δεξιά* κατά 90° και προχωρά κατά ένα τετράγωνο.

Γράψτε πρόγραμμα που να προσομοιώνει την κίνηση του μυρμηγκιού για 12000 βήματα. Κάθε 100 βήματα να αποθηκεύετε την εικόνα του πλέγματος σε αρχείο με τη διαμόρφωση plain pbm (δείτε την περιγραφή της διαμόρφωσης στην άσκηση 19). Δείτε όλες τις εικόνες· τι παρατηρείτε;

25. Ο αριθμός 12 μπορεί να γραφεί ως γινόμενο ακεραίων με τις μορφές 2×6, 3×4, $2 \times 2 \times 3$. Οι αριθμοί σε κάθε γινόμενο αποτελούν τους διαιρέτες του αρχικού αριθμού. Στην τελευταία μορφή, οι διαιρέτες είναι πρώτοι αριθμοί (διαιρούνται ακριβώς μόνο από το 1 και τον εαυτό τους).

    Γράψτε πρόγραμμα που θα δέχεται ένα ακέραιο αριθμό από το χρήστη και θα τον αναλύει σε γινόμενο πρώτων διαιρετών. Το πρόγραμμα θα τυπώνει τους διαιρέτες *σε μία γραμμή* στην οθόνη, με ένα κενό μεταξύ τους. Έτσι, αν δώσουμε 12 θα πρέπει να τυπώσει: 2 2 3, ενώ αν δώσουμε πρώτο αριθμό, π.χ. 13, θα τυπώσει μόνο ένα διαιρέτη: 13.

26. Γράψτε πρόγραμμα που:

    (α΄) Θα δέχεται από το πληκτρολόγιο ένα ακέραιο αριθμό. Να φροντίσετε ώστε το πρόγραμμα να μην τον κρατά αν είναι αρνητικός αλλά να ξαναζητά αριθμό, όσες φορές χρειαστεί.

    (β΄) Θα αναλύει τον αριθμό στα ψηφία του και θα τα αποθηκεύει σε διάνυσμα 10 θέσεων.

    (γ΄) Θα τυπώνει τον αριθμό στην οθόνη *αντίστροφα*, δηλαδή στα αριστερά θα είναι το ψηφίο των μονάδων, δεξιά του των δεκάδων κλπ., *χωρίς κενά μεταξύ τους*. Αν τυχόν εμφανίζονται μηδενικά στην αρχή του «αντίστροφου» αριθμού, δεν πρέπει να τυπώνονται.

    Παράδειγμα: το 1023 θα γίνεται 3201 ενώ το 100 θα γίνεται 1.

27. Οι αριθμοί 2, 3, 5, 7, 11, 13, 17, 19, ... είναι πρώτοι. Κάθε ακέραιο θετικό αριθμό μπορούμε να τον γράψουμε με μοναδικό τρόπο ως γινόμενο πρώτων αριθμών. Η ανάλυση αυτή λέγεται *παραγοντοποίηση σε πρώτους αριθμούς*. Έτσι

$$\begin{aligned} 15 &= 3 \times 5\,, \\ 364 &= 2 \times 2 \times 7 \times 13\,. \end{aligned}$$

    Προσέξτε ότι στην παραγοντοποίηση κάποιοι πρώτοι αριθμοί μπορεί να επαναλαμβάνονται.

Αποθηκεύστε στον υπολογιστή σας το αρχείο στη διεύθυνση https://tinyurl.com/yb9ubdjj. Περιέχει τους θετικούς ακέραιους που είναι πρώτοι και μικρότεροι από το 1000. Η πρώτη γραμμή του αρχείου έχει το πλήθος των αριθμών που ακολουθούν.

Γράψτε πρόγραμμα που να χρησιμοποιεί τους αριθμούς του αρχείου για να αναλύσει σε γινόμενο πρώτων αριθμών το $n = 76378260$. Βρείτε δηλαδή ποιοι από τους πρώτους αριθμούς διαιρούν το n ακριβώς (χωρίς υπόλοιπο) αλλά και πόσες φορές εμφανίζονται ως παράγοντές του. Τυπώστε τους διαιρέτες του n στην *ίδια γραμμή* στην οθόνη, με ένα κενό ανάμεσά τους.

Απάντηση: Το 76378260 *είναι το γινόμενο των πρώτων ακεραίων* 2, 2, 3, 5, 7, 7, 83, 313.

28. Το αρχείο στη διεύθυνση https://tinyurl.com/2h8wfweh περιέχει βαθμούς πτυχίου των αποφοίτων ενός πανεπιστημιακού τμήματος. Στην πρώτη γραμμή έχει το πλήθος των πτυχιούχων και μετά ακολουθούν σε ξεχωριστές γραμμές οι βαθμοί πτυχίου. Βρείτε

    - το μεγαλύτερο και το μικρότερο βαθμό,
    - το πλήθος των πτυχιούχων με «Άριστα» (βαθμός ≥ 8.5), «Πολύ καλά» ($6.5 \leq$ βαθμός < 8.5), «Καλά» (βαθμός< 6.5),

    Τα παραπάνω αποτελέσματα τυπώστε τα στην οθόνη.

    Κατόπιν, υπολογίστε πόσοι πτυχιούχοι έχουν βαθμό στα διαστήματα $[5, 5.25)$, $[5.25, 5.5)$, ..., $[9.75, 10)$. Τυπώστε στο αρχείο *graduates.txt* σε διαδοχικές γραμμές, τα όρια κάθε διαστήματος και το αντίστοιχο πλήθος πτυχιούχων. Τα όρια κάθε διαστήματος να τυπωθούν με 2 δεκαδικά.

29. Το αρχείο στη διεύθυνση https://tinyurl.com/293zdr6v περιέχει 3590 ακέραιους με το πολύ 7 ψηφία, σε ξεχωριστή γραμμή ο καθένας. Αποθηκεύστε το στον υπολογιστή σας. Βρείτε τους αριθμούς σε αυτό που έχουν άθροισμα ψηφίων ίσο με 27. Γράψτε τους στο αρχείο *s27.txt*, ένα αριθμό σε κάθε σειρά, και τυπώστε στην οθόνη το πλήθος τους.

30. **Αριθμός Mersenne** λέγεται ο ακέραιος που μπορεί να γραφτεί στη μορφή $2^k - 1$, για κάποιο ακέραιο k. **Πρώτος αριθμός** λέγεται κάθε ακέραιος μεγαλύτερος του 1 που διαιρείται ακριβώς μόνο από το 1 και τον εαυτό του.

    Να βρείτε και να τυπώσετε στο αρχείο *mersenne_prime.txt* όλους τους ακεραίους αριθμούς μέχρι το 1000000 που είναι ταυτόχρονα Mersenne και πρώτοι. Η εκτύπωση κάθε αριθμού θα γίνεται σε ξεχωριστή γραμμή.

    Απάντηση: 3, 7, 31, 127, 8191, 131071, 524287.

Κεφάλαιο 8

Συναρτήσεις–Υπορουτίνες

8.1 Εισαγωγή

Στα προηγούμενα κεφάλαια έχουν παρουσιαστεί κάποιες από τις βασικές εντολές και δομές της Fortran, αρκετές ώστε να μπορούμε να γράψουμε σχετικά πολύπλοκους κώδικες. Η συγκέντρωση, όμως, όλου του κώδικα στο κυρίως πρόγραμμα, ειδικά όταν είναι μεγάλος, καθιστά δύσκολη την κατανόησή του και, κυρίως, τη διόρθωση λαθών. Σχεδόν πάντα ο κώδικας αποτελείται από τμήματα που είναι, σε μεγάλο βαθμό, ανεξάρτητα μεταξύ τους. Αυτά μπορούν να απομονωθούν σε αυτόνομες *συναρτήσεις ή υπορουτίνες*, να αποτελούν, δηλαδή, ομάδες εντολών με συγκεκριμένο όνομα, οι οποίες θα καλούνται όπου και όσες φορές χρειάζεται από το κυρίως πρόγραμμα ή άλλες συναρτήσεις και υπορουτίνες, χρησιμοποιώντας μόνο αυτό το όνομα. Οι συγκεκριμένες ομάδες εντολών θα παραμετροποιούνται συνήθως από μία ή περισσότερες ποσότητες, τα *ορίσματα* της συνάρτησης ή υπορουτίνας. Καθώς η συνάρτηση δεν έχει ουσιαστική διαφορά από την υπορουτίνα, θα αναφερόμαστε σε αυτές συνολικά ως υποπρογράμματα.

Η οργάνωση του προγράμματός μας σε υποπρογράμματα είναι ένα πρώτο βήμα στην απλοποίηση του κώδικα και μας επιτρέπει να επικεντρωνόμαστε σε συγκεκριμένες, κατά το δυνατόν απλές, εργασίες κατά την ανάπτυξη ή διόρθωση του προγράμματος. Έτσι π.χ., ένας αλγόριθμος μπορεί να υλοποιηθεί, να διορθωθεί και να βελτιστοποιηθεί αυτόνομα, ανεξάρτητα από τον υπόλοιπο κώδικα και, επομένως, να μπορεί να χρησιμοποιείται από εμάς ή άλλους σε διαφορετικά προγράμματα. Από τη στιγμή που θα υπάρξει απομόνωση του κώδικα σε αυτόνομο υποπρόγραμμα, η χρήση του απλοποιείται σημαντικά καθώς μας απασχολεί μόνο το πώς τον καλούμε και τι ορίσματα πρέπει να «περάσουμε» στο υποπρόγραμμα και όχι το ποιους ακριβώς υπολογισμούς εκτελεί.

Η οργάνωση του κώδικα σε δεδομένα και σε διαδικασίες που επιδρούν σε αυτά

περιγράφεται ως *δομημένος (structured)* ή *διαδικαστικός (procedural)* προγραμματισμός.

Ένα πλήρες πρόγραμμα Fortran πρέπει να περιλαμβάνει ένα, και μόνο ένα, κυρίως πρόγραμμα, δηλαδή κώδικα μεταξύ **`PROGRAM`** και **`END PROGRAM`**. Οι εντολές στον κώδικα είναι αυτές που περιγράψαμε στα προηγούμενα κεφάλαια· επιπλέον, μπορούμε να έχουμε *κλήσεις* συναρτήσεων ή υπορουτινών που έχουν οριστεί πριν ή μετά το κυρίως πρόγραμμα και έχουν δηλωθεί σε αυτό κατάλληλα. Για ειδικές περιπτώσεις μπορούμε να δηλώσουμε υποπρογράμματα μέσα σε άλλο υποπρόγραμμα.

8.2 Η έννοια του υποπρογράμματος

Ας προσπαθήσουμε να κατανοήσουμε την έννοια της συνάρτησης στον προγραμματισμό με βάση τη γνωστή έννοια της μαθηματικής συνάρτησης. Στα μαθηματικά μπορούμε να ορίσουμε ότι

$$f(x) = x^2 + 5x - 2 \; .$$

Αυτό σημαίνει ότι οι συγκεκριμένες πράξεις, $x^2 + 5x - 2$, έχουν αποκτήσει ένα *όνομα*, f, μέσω του οποίου θα τις χρησιμοποιούμε όποτε χρειαζόμαστε το αποτέλεσμά τους. Παρατηρούμε ότι εξαρτώνται από ένα *σύμβολο*, το x, και επομένως, δεν μπορούν να εκτελεστούν και να μας δώσουν αποτέλεσμα. Ο συμβολισμός $f(x)$ υποδηλώνει ότι η συνάρτηση f έχει ως παράμετρο, ως *όρισμα* όπως λέμε, την ποσότητα x. Για τις συναρτήσεις πραγματικής μεταβλητής το x συμβολίζει έναν πραγματικό αριθμό. Όταν επιθυμούμε να εκτελέσουμε τις πράξεις x^2+5x-2 για κάποια τιμή του x μπορούμε να χρησιμοποιήσουμε (να *καλέσουμε*) τη συνάρτηση f με ταυτόχρονο προσδιορισμό της τιμής του ορίσματος, του συμβόλου x. Γι᾽ αυτό γράφουμε, π.χ., $y = f(2.5)$ αντί για το ισοδύναμο αλλά πιο εκτεταμένο $y = 2.5^2 + 5 \times 2.5 - 2$. Οι πράξεις μπορούν να εκτελεστούν αφού δώσουμε τιμή στο σύμβολο x, θέσουμε, δηλαδή, τη δεδομένη τιμή όπου εμφανίζεται το x. Το αποτέλεσμά τους για τη συγκεκριμένη τιμή εκχωρείται (*επιστρέφεται*) στο όνομα της συνάρτησης και μπορούμε να το κρατήσουμε σε κάποια κατάλληλη ποσότητα. Μια μαθηματική συνάρτηση μπορεί να έχει περισσότερα από ένα ορίσματα (παραμέτρους). Αφού τα προσδιορίσουμε όλα, μπορούν να εκτελεστούν οι πράξεις τις οποίες αντιπροσωπεύει. Προφανώς, μια συνάρτηση μπορεί να κληθεί όσες φορές επιθυμούμε.

Στη Fortran, κατά πλήρη αντιστοιχία: ένα τμήμα κώδικα μπορούμε να το ξεχωρίσουμε από το κύριο σώμα των εντολών του προγράμματός μας και να του δώσουμε ένα όνομα. Αυτό το τμήμα κώδικα μπορεί να εξαρτάται από καμία, μία ή περισσότερες ποσότητες. Στον ορισμό του υποπρογράμματος τα ορίσματα δεν είναι τίποτε άλλο παρά σύμβολα που αντιστοιχούν σε ποσότητες συγκεκριμένων τύπων. Όταν το καλέσουμε με το όνομά του, πρέπει να προσδιορίσουμε ταυτόχρονα και τα ορίσματά του, δίνοντας τιμές των αντίστοιχων τύπων σε καθένα από αυτά. Τότε μόνο μπορούν να εκτελεστούν οι εντολές που αυτό αντιπροσωπεύει.

Όταν ολοκληρωθεί η εκτέλεση του υποπρογράμματος μπορεί να επιστρέφεται τιμή μέσω του ονόματός του (στην περίπτωση που το υποπρόγραμμα είναι συνάρτηση). Η επιστρεφόμενη τιμή πρέπει να χρησιμοποιηθεί· έτσι μπορούμε να την αποθηκεύσουμε σε κατάλληλη μεταβλητή, ή να την τυπώσουμε σε αρχείο, ή να την συμπεριλάβουμε σε σύνθετη έκφραση. Ένα τμήμα κώδικα που θέλουμε να επιστρέφει τιμή το υλοποιούμε ως συνάρτηση ενώ αν δεν επιστρέφει τίποτε στον κώδικά μας αλλά απλώς εκτελεί κάποιες εντολές, το υλοποιούμε ως υπορουτίνα. Στην περίπτωση που θέλουμε να λάβουμε περισσότερα του ενός αποτελέσματα μπορούμε να υποκαταστήσουμε το τμήμα του κώδικα με συνάρτηση στην οποία θα χρησιμοποιήσουμε ως τύπο επιστρεφόμενης ποσότητας ένα κατάλληλο παραγόμενο τύπο, Κεφάλαιο 6. Εναλλακτικά, μπορούμε να το υποκαταστήσουμε είτε με συνάρτηση είτε με υπορουτίνα, στις οποίες έχουμε επιπλέον ορίσματα που αποκτούν τιμή μετά την ολοκλήρωση της εκτέλεσης του υποπρογράμματος.

Η γενική μορφή ορισμού μιας συνάρτησης είναι

```
επιστρεφόμενος_τύπος FUNCTION όνομα(όρισμαΑ, όρισμαΒ,...)
τύπος_ορίσματος_Α :: όρισμαΑ
τύπος_ορίσματος_Β :: όρισμαΒ


τύπος_Α :: τοπική_μεταβλητή_Α, ...
τύπος_Β :: τοπική_μεταβλητή_Β, ...


..... ! κώδικας
όνομα = .....
..... ! κώδικας
END FUNCTION όνομα
```

Ισοδύναμα, μπορούμε να περιλάβουμε στο σώμα της συνάρτησης τη δήλωση του ονόματός της (που, μην ξεχνάμε, αποκτά τιμή στο σώμα της συνάρτησης):

```
FUNCTION όνομα(όρισμαΑ, όρισμαΒ,...)
τύπος_ορίσματος_Α :: όρισμαΑ
τύπος_ορίσματος_Β :: όρισμαΒ
επιστρεφόμενος_τύπος :: όνομα


τύπος_Α :: τοπική_μεταβλητή_Α, ...
τύπος_Β :: τοπική_μεταβλητή_Β, ...


..... ! κώδικας
όνομα = .....
..... ! κώδικας
END FUNCTION όνομα
```

Έναν τρίτο τρόπο ορισμού, ισοδύναμο με τους παραπάνω, θα τον αναφέρουμε παρακάτω.

Ο τύπος του ονόματος της συνάρτησης μπορεί να είναι οποιοσδήποτε από τους ενσωματωμένους ή όσους ορίζει ο χρήστης (Κεφάλαιο 6). Δεν είναι απαραίτητο να είναι απλός τύπος· μπορεί ακόμα να είναι και διάνυσμα ή πίνακας. Επίσης, τα ορίσματα μπορεί να μην υπάρχουν, στη σπάνια περίπτωση που δεν επιθυμούμε να δώσουμε τιμές «εξωτερικά», οι παρενθέσεις, όμως, πρέπει να υπάρχουν (με κενή λίστα παραμέτρων). Το σώμα της συνάρτησης μπορεί να περιέχει όλες τις εντολές και δηλώσεις που έχουμε ήδη συναντήσει.

Για την υπορουτίνα—που δεν επιστρέφει τιμή—η γενική μορφή ορισμού είναι

SUBROUTINE όνομα(όρισμαΑ, όρισμαΒ,...)
τύπος_ορίσματος_Α :: όρισμαΑ
τύπος_ορίσματος_Β :: όρισμαΒ

τύπος :: μεταβλητή, ...

! κώδικας
.....
END SUBROUTINE όνομα

Στην περίπτωση που η υπορουτίνα δεν χρειάζεται ορίσματα μπορούν να παραληφθούν οι παρενθέσεις μετά το όνομά της.

8.3 Παραδείγματα

Τα παραπάνω θα γίνουν κατανοητά με κάποια παραδείγματα:

Έστω ότι στο πρόγραμμά μας θέλουμε να υπολογίσουμε την τιμή της ποσότητας x^2+x-9 για $x = 1.3$, $x = 2.7$, $x = -0.5$. Μπορούμε να έχουμε τον ακόλουθο κώδικα:

```
PROGRAM fun
  IMPLICIT NONE

  DOUBLE PRECISION :: x1, x2, x3, y1, y2, y3

  x1 = 1.3d0
  y1 = x1**2 + x1 - 9.0d0

  x2 = 2.7d0
  y2 = x2**2 + x2 - 9.0d0

  x3 = -0.5d0
  y3 = x3**2 + x3 - 9.0d0
```

```
! ......
END PROGRAM fun
```

Παρατηρούμε ότι επαναλαμβάνουμε κώδικα, εδώ μία γραμμή, ουσιαστικά χωρίς αλλαγές για κάθε υπολογισμό. Έχουμε τη δυνατότητα να τον «απομονώσουμε» σε μια πιο εύχρηστη συνάρτηση, απλοποιώντας το πρόγραμμά μας· σκεφτείτε πόσο δυσνόητο θα ήταν αν ο επαναλαμβανόμενος κώδικας αποτελείται από μια πολύπλοκη έκφραση ή εκτείνεται σε πολλές γραμμές κειμένου. Μπορούμε να ορίσουμε το εξής υποπρόγραμμα

```
FUNCTION polynomial(x)
  IMPLICIT NONE

  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: polynomial

  polynomial = x*x + x - 9.0d0
END FUNCTION polynomial
```

Το όνομα της συνάρτησης (εδώ polynomial) είναι της επιλογής του προγραμματιστή και σχηματίζεται σύμφωνα με τους κανόνες που ισχύουν για τα ονόματα των μεταβλητών, §2.1.2. Η μία παράμετρος που πρέπει να προσδιοριστεί στη συγκεκριμένη συνάρτηση, το μοναδικό όρισμά της, το x, παρατίθεται μετά το όνομα μέσα σε παρενθέσεις. Η συγκεκριμένη συνάρτηση περιέχει όλο το σχετικό κώδικα για τον υπολογισμό του πολυωνύμου. Τα υποπρογράμματα και το κυρίως πρόγραμμα είναι ανεξάρτητα τμήματα κώδικα μεταξύ τους και, συνεπώς, όποια ποσότητα χρησιμοποιηθεί σε ένα από αυτά πρέπει να δηλωθεί στο αντίστοιχο τμήμα. Το όνομα της συνάρτησης πρέπει να δηλωθεί, είτε στο τμήμα των δηλώσεων, όπως στο παράδειγμα, είτε στην πρώτη γραμμή του ορισμού της συνάρτησης, ως εξής

```
DOUBLE PRECISION FUNCTION polynomial(x)
.....
.....
END FUNCTION polynomial
```

Παρατηρήστε ότι στον κώδικα της συνάρτησης γίνεται εκχώρηση του τελικού αποτελέσματος, αυτού που θέλουμε να επιστρέψουμε, στο όνομά της. Το όνομα της συνάρτησης δηλώνεται και συμπεριφέρεται όπως μια οποιαδήποτε μεταβλητή. Βέβαια, δεν έχει αρχική τιμή και πρέπει να αποκτήσει κάποια προτού τελειώσει η εκτέλεση της συνάρτησης.

Απομένει να εξηγήσουμε την ιδιαίτερη δήλωση της παραμέτρου. Προφανώς, ως ποσότητα που χρησιμοποιείται στη συνάρτηση, πρέπει να δηλωθεί, να ενημερωθεί δηλαδή ο μεταγλωττιστής ότι εδώ το σύμβολο x αντιστοιχεί σε πραγματικό αριθμό. Όπως βλέπουμε, η δήλωση της παραμέτρου συμπληρώνεται με το `INTENT (in)`. Το συγκεκριμένο υποδηλώνει την πρόθεσή μας να χρησιμοποιήσουμε τη μεταβλητή

μόνο για ανάγνωση. Παρατηρούμε ότι δεν την τροποποιούμε στον κώδικα και συνεπώς είναι καλό να ενημερώνουμε τον μεταγλωττιστή και τον χρήστη του υποπρογράμματος γι' αυτό. Στην περίπτωση που δε μας ενδιαφέρει η τιμή που θα έχει το σύμβολο αυτό όταν κληθεί η συνάρτηση αλλά θα του αποδώσουμε τιμή, μόνο, η δήλωση (καλό είναι να) συμπληρώνεται με το **INTENT** (**out**). Στην περίπτωση που χρησιμοποιούμε και τροποποιούμε την τιμή της παραμέτρου στη συνάρτηση, συμπληρώνουμε τη δήλωση με το **INTENT** (**inout**). Πάντως, ένας από τους τρεις προσδιορισμούς της χρήσης καλό είναι να υπάρχει σε κάθε δήλωση ορίσματος. Να τονίσουμε ότι τέτοιος προσδιορισμός έχει νόημα μόνο για τα ορίσματα και όχι για το όνομα της συνάρτησης ή άλλες μεταβλητές.

Τον κώδικα που ορίζει τη συνάρτηση τον γράφουμε πριν ή μετά το κυρίως πρόγραμμα (ή άλλα υποπρογράμματα), στο ίδιο αρχείο. Εναλλακτικά, μπορούμε να τον έχουμε σε ξεχωριστό αρχείο και να το συμπεριλάβουμε κατάλληλα κατά τη μεταγλώττιση.

Προτού εξηγήσουμε πώς χρησιμοποιούμε το υποπρόγραμμα, ας δούμε ακόμα ένα, πιο σύνθετο παράδειγμα:

Γνωρίζουμε από τα μαθηματικά ότι

$$\ln(1+x) = -\sum_{k=1}^{\infty} \frac{(-x)^k}{k} \,, \qquad -1 < x \leq 1 \,.$$

Έστω ότι θέλουμε να υπολογίσουμε την τιμή του $\ln(1.05)$ με το συγκεκριμένο τύπο και να το συγκρίνουμε με το αποτέλεσμα της σχετικής ενσωματωμένης συνάρτησης (**LOG**()). Φυσικά, δεν μπορούμε να συγκεντρώσουμε άπειρους όρους και έτσι σταματούμε τη σειρά σε κάποιον όρο που είναι αρκετά μικρός κατ' απόλυτη τιμή και δεν συνεισφέρει στην τελική τιμή· όλοι οι υπόλοιποι είναι μικρότεροί του. Ο κώδικας θα είναι

```
PROGRAM logarithm
  IMPLICIT NONE
  DOUBLE PRECISION :: sum, y, x, term
  INTEGER :: k

! PART A: READ
  DO
     READ *, y
     x = y - 1.0d0
     IF ( (-1.0d0 < x) .AND. ( x <= 1.0d0 ) ) EXIT
  END DO

! PART B: COMPUTE
  k = 1
  sum = 0.0d0
  DO
```

```
    term = -(-x)**k / k
    IF (ABS(term) < 1d-6) EXIT
    sum = sum + term
    k = k + 1
  END DO

! PART C: PRINT
  WRITE (*,*) "ln(", y, ") = ", sum
  WRITE (*,*) "Timi apo enswmatomeni synartisi", LOG(y)
END PROGRAM logarithm
```

Παρατηρούμε ότι ο κώδικας αποτελείται από τρία τμήματα:

1. την ανάγνωση και αποδοχή της τιμής,
2. τον υπολογισμό της ποσότητας που θέλουμε και
3. την εκτύπωση του αποτελέσματος.

Αντί να έχουμε όλες τις εντολές συγκεντρωμένες και να τις ξεχωρίζουμε μόνο με σχόλια μπορούμε να απομονώσουμε τμήματα του προγράμματος σε υποπρογράμματα. Ας δούμε πώς:

Παρατηρούμε ότι ο υπολογισμός του sum είναι το (μόνο χρήσιμο) αποτέλεσμα του κώδικα μεταξύ των !PART B και !PART C. Μπορούμε να ορίσουμε μια συνάρτηση που να περιέχει μόνο τις σχετικές εντολές και να επιστρέφει την τιμή του sum:

```
FUNCTION sumseries(x)
  IMPLICIT NONE

  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: sumseries

  DOUBLE PRECISION :: term, sum
  INTEGER :: k

  k = 1
  sum = 0.0d0
  DO
     term = -(-x)**k / k
     IF (ABS(term) < 1d-6) EXIT
     sum = sum + term
     k = k + 1
  END DO
  sumseries = sum
END FUNCTION sumseries
```

Παρατηρήστε ότι συμπληρώσαμε τον κώδικα με την εκχώρηση του τελικού αποτελέσματος στο όνομα της συνάρτησης. Επίσης, προσέξτε ότι δηλώσαμε στο σώμα της συνάρτησης τις μεταβλητές που χρειάζονται σε αυτή.

Ένα άλλο, αρκετά ανεξάρτητο τμήμα του προγράμματος που μπορούμε να απομονώσουμε, είναι ο κώδικας για την ανάγνωση και αποδοχή τιμής. Βλέπουμε ότι δίνει τιμή σε δύο μεταβλητές που χρησιμοποιούνται στο υπόλοιπο πρόγραμμα. Μπορούμε να τον γράψουμε σε ξεχωριστή συνάρτηση που θα επιστρέφει τη μία τιμή (θα εκχωρείται, δηλαδή, στο όνομα της συνάρτησης) ενώ η άλλη ποσότητα θα αποδίδεται σε όρισμα. Έτσι θα έχουμε

```
FUNCTION readxy(y)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (out) :: y
  DOUBLE PRECISION :: readxy

  DOUBLE PRECISION :: x

  DO
     READ *, y
     x = y - 1.0d0
     IF ( (-1.0d0 < x) .AND. ( x <= 1.0d0 ) ) EXIT
  END DO

  readxy = x
END FUNCTION readxy
```

Προσέξτε ότι η δήλωση του ορίσματος y περιλαμβάνει το **INTENT** (**out**). Ο προσδιοριστής αυτός υποδηλώνει ότι η ποσότητα y παίρνει τιμή από τη συνάρτηση κατά την εκτέλεσή της και ότι η τυχόν αρχική τιμή της δεν χρησιμοποιείται. Παρατηρήστε, ακόμα, την εκχώρηση τιμής στο όνομα της συνάρτησης.

Η συγκεκριμένη επιλογή για τη δημιουργία του υποπρογράμματος δεν είναι λάθος· όμως, αντιμετωπίζει διαφορετικά δύο όμοιες, ισοδύναμες ποσότητες. Ας δούμε μια άλλη προσέγγιση που δεν παρουσιάζει αυτήν την ανομοιομορφία, την υλοποίηση με υπορουτίνα:

```
SUBROUTINE readxy(x,y)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (out) :: x, y

  DO
     READ *, y
     x = y - 1.0d0
     IF ( (-1.0d0 < x) .AND. ( x <= 1.0d0 ) ) EXIT
  END DO
END SUBROUTINE readxy
```

8.3.1 Δήλωση υποπρογράμματος

Όπως έχουμε ήδη αναφέρει και θα τονίσουμε για άλλη μια φορά, οποιαδήποτε ποσότητα στο πρόγραμμά μας, μεταβλητή, σταθερή, πίνακας ή υποπρόγραμμα, προτού χρησιμοποιηθεί πρέπει να δηλωθεί, ώστε να ενημερωθεί ο μεταγλωττιστής για το είδος της, τον τύπο της ή, πιθανόν, την τιμή της. Η απαραίτητη δήλωση μιας συνάρτησης ή υπορουτίνας γίνεται ως εξής: στο κυρίως πρόγραμμα ή στο υποπρόγραμμα που θέλουμε να τη χρησιμοποιήσουμε, συμπληρώνουμε το τμήμα των δηλώσεων με το *interface* (διασύνδεση) του υποπρογράμματος που θα κληθεί. Το interface είναι ό,τι απομένει από τον ορισμό του υποπρογράμματος αν παραλείψουμε τις εντολές που αποτελούν το σώμα του και τις δηλώσεις των ποσοτήτων που χρησιμοποιούνται αποκλειστικά σε αυτές. Επομένως, αποτελείται από την πρώτη γραμμή του ορισμού (που εισαγάγει το υποπρόγραμμα), τις δηλώσεις των ορισμάτων και τυχόν επιστρεφόμενης ποσότητας καθώς και την τελευταία γραμμή (το **END**). Π.χ. το **interface** της συνάρτησης polynomial() που ορίσαμε παραπάνω είναι:

```
INTERFACE
  FUNCTION polynomial(x)
    IMPLICIT NONE
    DOUBLE PRECISION, INTENT (in) :: x
    DOUBLE PRECISION :: polynomial
  END FUNCTION polynomial
END INTERFACE
```

ενώ της υπορουτίνας readxy() είναι:

```
INTERFACE
  SUBROUTINE readxy(x,y)
    IMPLICIT NONE
    DOUBLE PRECISION, INTENT (out) :: x, y
  END SUBROUTINE readxy
END INTERFACE
```

Οι δηλώσεις των υποπρογραμμάτων που θα κληθούν στο κυρίως πρόγραμμα ή σε υποπρόγραμμα μπορούν να περιλαμβάνονται σε ένα κοινό **INTERFACE** αντί να είναι σε ξεχωριστά.

8.4 Κλήση υποπρογράμματος

Η κλήση μιας συνάρτησης ή υπορουτίνας μπορεί να γίνει σε οποιοδήποτε σημείο του κώδικά μας, είτε στο κυρίως πρόγραμμα, είτε σε άλλο υποπρόγραμμα (όχι, όμως, στον εαυτό τους[1]).

[1] θα δούμε παρακάτω τα *αναδρομικά υποπρογράμματα* που δεν έχουν αυτόν τον περιορισμό.

Για να χρησιμοποιήσουμε μια συνάρτηση παραθέτουμε το όνομά της ακολουθούμενο από κατάλληλες τιμές εντός παρενθέσεων, αντίστοιχες σε σειρά, πλήθος και τύπο με τα ορίσματά της[2]. Καθώς επιστρέφει τιμή, μπορεί να εμφανίζεται μόνο σε εκφράσεις που συμμετέχουν και άλλες ποσότητες, σε εντολή εκχώρησης ή, γενικότερα, σε εντολή που «περιμένει» ποσότητα του τύπου που επιστρέφει η συνάρτηση. Έτσι, π.χ. η συνάρτηση polynomial() που ορίσαμε παραπάνω μπορεί να χρησιμοποιηθεί σε κώδικα ως εξής:

```
y = polynomial(1.3d0)
```

ή

```
y = polynomial(2.4d0) + 2.0d0 * polynomial(3.5d0) + 4.0d0
```

ή, ακόμα,

```
WRITE (*,*) polynomial(4.6d0)
```

αλλά, βέβαια, όχι σε μια εντολή από μόνη της ή σε εντολή εκχώρησης σε αυτή, ή χωρίς να ακολουθείται από τα ορίσματά της[3]:

```
polynomial(3.4d0)          ! Λάθος
polynomial(3.4d0) = 4.0    ! Λάθος
y = polynomial             ! Λάθος
```

Κλήση μιας υπορουτίνας γίνεται με αρκετά διαφορετικό τρόπο. Και σε αυτή την περίπτωση χρησιμοποιούμε το όνομά της ακολουθούμενο από κατάλληλες τιμές, εντός παρενθέσεων, αντίστοιχες με τα ορίσματά της. Όμως, προηγείται η δεσμευμένη λέξη **CALL** και η κλήση αποτελεί ξεχωριστή εντολή. Η υπορουτίνα readxy() που ορίσαμε παραπάνω μπορεί να κληθεί ως εξής (μόνο!)

```
DOUBLE PRECISION :: x, y

CALL readxy(x,y)
```

Παρατηρήστε ότι ως ορίσματα δίνουμε δύο *μεταβλητές*. Καθώς η συγκεκριμένη υπορουτίνα τροποποιεί τις παραμέτρους της (έχουν δηλωθεί ως **INTENT** (**out**)), δεν μπορεί να έχει ως ορίσματα σταθερές ποσότητες. Μετά την κλήση, τα x, y έχουν πάρει κατάλληλες τιμές και μπορούμε να τα χρησιμοποιήσουμε.

8.5 Παράδειγμα

Τα παραπάνω θα γίνουν κατανοητά τροποποιώντας το κυρίως πρόγραμμα (logarithm), στο §8.3 στη σελίδα 118. Πρώτα, αν και δεν έχει σημασία η σειρά, παραθέτουμε ξανά τη συνάρτηση `sumseries()` και το υποπρόγραμμα `readxy()`. Κατόπιν, γράφουμε τροποποιημένο το κυρίως πρόγραμμα:

[2] εκτός αν έχει προαιρετικά ορίσματα, §8.7.3. ή χρησιμοποιήσουμε keywords για να αλλάξουμε σειρά παράθεσης.

[3] εκτός και αν είναι η ίδια όρισμα.

```
FUNCTION sumseries(x)
  IMPLICIT NONE

  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: sumseries

  DOUBLE PRECISION :: term, sum
  INTEGER :: k

  k = 1
  sum = 0.0d0
  DO
     term = -(-x)**k / k
     IF (ABS(term) < 1d-6) EXIT
     sum = sum + term
     k = k + 1
  END DO
  sumseries = sum
END FUNCTION sumseries

SUBROUTINE readxy(x,y)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (out) :: x, y

  DO
     READ *, y
     x = y - 1.0d0
     IF ( (-1.0d0 < x) .AND. ( x <= 1.0d0 ) ) EXIT
  END DO
END SUBROUTINE readxy

PROGRAM logarithm
  IMPLICIT NONE
  DOUBLE PRECISION :: sum, y, x, term
  INTEGER :: k

  INTERFACE
     FUNCTION sumseries(x)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT (in) :: x
       DOUBLE PRECISION :: sumseries
     END FUNCTION sumseries
```

```
    SUBROUTINE readxy(x,y)
      IMPLICIT NONE
      DOUBLE PRECISION, INTENT (out) :: x, y
    END SUBROUTINE readxy
  END INTERFACE

  CALL readxy(x,y)     ! READ

  sum = sumseries(x)   ! COMPUTE

  WRITE (*,*) "ln(", y, ") = ", sum     ! PRINT
  WRITE (*,*) "Timi apo enswmatomeni synartisi", LOG(y)
END PROGRAM logarithm
```

Παρατηρούμε ότι το κυρίως σώμα του προγράμματός μας απλοποιήθηκε σημαντικά. Εκτός από αυτή τη βελτίωση, μπορούμε να επεκτείνουμε το πρόγραμμα επαναλαμβάνοντας τις κλήσεις των δυο υποπρογραμμάτων όσες φορές επιθυμούμε χωρίς να επαναλαμβάνουμε τον κώδικά τους, όπως θα ήμασταν αναγκασμένοι να κάνουμε χωρίς τη χρήση των υποπρογραμμάτων.

8.6 Εκτέλεση υποπρογράμματος–RETURN

Ένα υποπρόγραμμα εκτελείται αν, και μόνο αν, κληθεί από άλλο τμήμα του προγράμματός μας. Στο σημείο της κλήσης, η ροή της εκτέλεσης των εντολών συνεχίζει στο σώμα του καλούμενου υποπρογράμματος μέχρι να συναντήσει το καταληκτικό **END FUNCTION**/**END SUBROUTINE** ή την εντολή **RETURN**. Τότε, η ροή επιστρέφει στο σημείο που έγινε η κλήση και συνεχίζει με την επόμενη εντολή.

Ένα παράδειγμα χρήσης του **RETURN** είναι το εξής: έστω ότι χρειαζόμαστε μια υπορουτίνα που να τυπώνει με κατάλληλο μήνυμα την τετραγωνική ρίζα ενός πραγματικού αριθμού που θα δέχεται ως όρισμα. Θα πρέπει, βέβαια, να ελέγχει αν ο αριθμός είναι μη αρνητικός και να μας ενημερώνει αν δεν επιτρέπεται να υπολογίσουμε τη ρίζα του. Μπορούμε να έχουμε τον ακόλουθο κώδικα

```
SUBROUTINE riza(x)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: x

  IF (x < 0.0d0) THEN
     WRITE (*,*) "Wrong argument"
     RETURN
  END IF

  WRITE (*,*) "The root is", SQRT(x)
END SUBROUTINE riza
```

Οι ποσότητες που ορίζονται σε ένα υποπρόγραμμα, δημιουργούνται όταν η ροή εκτέλεσης συναντήσει τη δήλωσή τους[4]· *καταστρέφονται όταν η ροή εγκαταλείψει το υποπρόγραμμα.*

8.7 Όρισμα υποπρογράμματος

Όπως αναφέραμε, το όνομα ενός υποπρογράμματος ακολουθείται κατά τον ορισμό του από καμία, μία ή περισσότερες συμβολικές ποσότητες εντός παρενθέσεων. Αυτές είναι τα ορίσματα, οι παράμετροι του κώδικα που αντιπροσωπεύει το υποπρόγραμμα· ο κώδικας δεν μπορεί να εκτελεστεί αν δεν δοθούν τιμές σε αυτά. Τιμές αποδίδουμε στα ορίσματα κατά την κλήση του υποπρογράμματος, με πλήθος, σειρά και τύπο όπως ακριβώς προσδιορίζεται στον ορισμό του υποπρογράμματος[5].

Ειδικές περιπτώσεις που χρειάζονται εξήγηση είναι οι ακόλουθες:

8.7.1 Διάνυσμα/Πίνακας ως όρισμα

Όρισμα ενός υποπρογράμματος μπορεί φυσικά να είναι ένα διάνυσμα ή πίνακας. Ας δούμε τους τρόπους δήλωσής του:

Όταν το υποπρόγραμμα πρόκειται να χρησιμοποιηθεί για πίνακα με διαστάσεις σταθερές και γνωστές κατά τη μεταγλώττιση, (ή πίνακες με ίδιο σχήμα και διαστάσεις) η δήλωση του αντίστοιχου ορίσματος είναι η αναμενόμενη. Π.χ. η υπορουτίνα

```
SUBROUTINE athr(a)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: a(3)

  WRITE (*,*) a(1) + a(2) + a(3)
END SUBROUTINE athr
```

δέχεται ως όρισμα ένα διάνυσμα 3 πραγματικών στοιχείων.

Αν επιθυμούμε να χρησιμοποιήσουμε το υποπρόγραμμά μας για πίνακες με οποιεσδήποτε διαστάσεις, αλλά πάντα με ίδιο σχήμα (μονοδιάστατους, διδιάστατους, κλπ.), μπορούμε να δηλώσουμε το αντίστοιχο όρισμα χωρίς να προσδιορίσουμε αριθμητικά τις διαστάσεις, αλλά μόνο το σχήμα χρησιμοποιώντας κατάλληλο πλήθος των χαρακτήρων (:). Έτσι, το άθροισμα οποιουδήποτε διδιάστατου πίνακα μπορεί να εκτυπωθεί με την ακόλουθη υπορουτίνα

```
SUBROUTINE def(a)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: a(:,:)
```

[4] αν δεν είναι *στατικές* ποσότητες, §8.8.

[5] εκτός αν έχει προαιρετικά ορίσματα, §8.7.3, ή χρησιμοποιήσουμε keywords και αλλάξουμε τη σειρά προσδιορισμού.

```
  WRITE (*,*) SUM(a)
END SUBROUTINE def
```

Η κλήση της συγκεκριμένης χρειάζεται μόνο το όνομα ενός διδιάστατου πίνακα για όρισμα. Όποτε χρειαζόμαστε τις διαστάσεις του συγκεκριμένου ορίσματος μπορούμε να τις βρούμε με τη συνάρτηση **SIZE**(). Έτσι, η δημιουργία ενός τοπικού πραγματικού πίνακα με ίδιες διαστάσεις με το εκάστοτε όρισμα γίνεται με την ακόλουθη δήλωση (στο υποπρόγραμμα)

```
DOUBLE PRECISION :: b(SIZE(a,1), SIZE(a,2))
```

Εναλλακτικά (χωρίς να υπάρχει κανένα πλεονέκτημα), μπορούμε να περάσουμε και τις διαστάσεις του πίνακα ως ορίσματα και να τις χρησιμοποιήσουμε στη δήλωση. Έτσι, αν το όρισμα είναι ένας διδιάστατος πίνακας a(m,n) ο ορισμός γίνεται

```
SUBROUTINE def(a, m, n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: m,n
  DOUBLE PRECISION, INTENT (in) :: a(m,n)

  WRITE (*,*) SUM(a)
END SUBROUTINE def
```

Φυσικά, η κλήση του υποπρογράμματος αλλάζει αντίστοιχα.

Παρατηρήσεις

- Σε όρισμα που είναι πίνακας μπορούμε να συμπληρώσουμε τη δήλωσή του σε υποπρόγραμμα με το κατάλληλο **INTENT**.
- Όρισμα που είναι διάνυσμα ή πίνακας μπορεί να δηλωθεί ως **ALLOCATABLE**[6]. Τότε, μέσα στο σχετικό υποπρόγραμμα μπορεί να γίνει δέσμευση ή αποδέσμευση μνήμης για αυτό (**ALLOCATE**/**DEALLOCATE**). Όταν γίνει η κλήση του υποπρογράμματος πρέπει υποχρεωτικά να περάσουμε στο συγκεκριμένο όρισμα έναν όμοιο (σε σχήμα και τύπο) πίνακα που να έχει δηλωθεί ως **ALLOCATABLE**.

8.7.2 Υποπρόγραμμα ως όρισμα

Εκτός από απλές μεταβλητές ενσωματωμένου ή παραγόμενου τύπου και πίνακες μπορούμε να έχουμε ως όρισμα *άλλο υποπρόγραμμα*. Ο τρόπος που γίνεται αυτό είναι απλός: στη λίστα των ορισμάτων παραθέτουμε μόνο ένα συμβολικό όνομα που θα το δηλώσουμε ως συνάρτηση ή υπορουτίνα· στις δηλώσεις των ορισμάτων πρέπει να περιλάβουμε τη «δήλωση» του υποπρογράμματος, δηλαδή το **INTERFACE** του.

[6] ISO TR 15581 και Fortran 2003.

Φυσικά, όπως ισχύει και για όλα τα ορίσματα, το όνομα του υποπρογράμματος που είναι όρισμα, είναι ένα τοπικής χρήσης σύμβολο. Το ποια συνάρτηση ή υπορουτίνα θα χρησιμοποιηθεί καθορίζεται κατά την κλήση. Τότε πρέπει στη θέση του αντίστοιχου ορίσματος να δώσουμε το *όνομα* ενός υποπρογράμματος ίδιου είδους (συνάρτηση ή υπορουτίνα) με το όρισμα, και με ίδιο πλήθος και τύπο ορισμάτων και τυχόν επιστρεφόμενης τιμής. Παράδειγμα χρήσης είναι το ακόλουθο:

```
SUBROUTINE plot(a, b, f)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: a, b

  INTERFACE
     FUNCTION f(x)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT (in) :: x
       DOUBLE PRECISION :: f
     END FUNCTION f
  END INTERFACE

  INTEGER :: i
  DOUBLE PRECISION :: x, step

  step = (b-a) / 100.0d0

  DO i=0,100
     x = a + i * step
     WRITE (*,*) x, f(x)
  END DO
END SUBROUTINE plot

FUNCTION g(x)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: g

  g = x*x - 2.0d0
END FUNCTION g

FUNCTION h(x)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: h

  h = 1.0d0 - 2.0d0 * x
```

```
END FUNCTION h

PROGRAM test
  IMPLICIT NONE
  INTERFACE
     SUBROUTINE plot(a, b, f)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT (in) :: a, b

       INTERFACE
          FUNCTION f(x)
            IMPLICIT NONE
            DOUBLE PRECISION, INTENT (in) :: x
            DOUBLE PRECISION :: f
          END FUNCTION f
       END INTERFACE
     END SUBROUTINE plot

     FUNCTION g(x)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT (in) :: x
       DOUBLE PRECISION :: g
     END FUNCTION g

     FUNCTION h(x)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT (in) :: x
       DOUBLE PRECISION :: h
     END FUNCTION h
  END INTERFACE

  CALL plot(0.0d0, 1.0d0, g)
  CALL plot(0.0d0, 1.0d0, h)

END PROGRAM test
```

Στον παραπάνω κώδικα έχουμε τη συνάρτηση plot που τυπώνει στην οθόνη τις τιμές της συνάρτησης που δέχεται ως όρισμα, f, σε 101 ισαπέχουσες τιμές μεταξύ των a, b που και αυτά είναι ορίσματα. Τη χρησιμοποιούμε για να τυπώσουμε τιμές για δύο συναρτήσεις g(x), h(x) μεταξύ των 0.0, 1.0.

Παρατήρηση Αν επιθυμούμε να δώσουμε ως όρισμα υποπρογράμματος μία ενσωματωμένη συνάρτηση, πρέπει να χρησιμοποιήσουμε ειδικό όνομα για αυτή, ανάλογα με τον τύπο των ορισμάτων της. Τούτο αποτελεί κατάλοιπο της Fortran 66 που έχει

κληρονομηθεί στις επόμενες εκδόσεις· δεν έχουμε αναφερθεί στα ειδικά ονόματα των ενσωματωμένων συναρτήσεων.

8.7.3 Προαιρετικό όρισμα

Η Fortran μας δίνει τη δυνατότητα να προσδιορίσουμε ότι κάποια από τα ορίσματα είναι *προαιρετικά*, συμπληρώνοντας τη δήλωσή τους με τη δεσμευμένη λέξη **OPTIONAL**. Τα συγκεκριμένα ορίσματα πρέπει να βρίσκονται στο τέλος της λίστας ορισμάτων. Κατά την κλήση μπορούμε να δώσουμε τιμή σε αυτά κανονικά, ή να παραλείψουμε να το κάνουμε. Στην τελευταία περίπτωση δεν μπορούμε να τα χρησιμοποιήσουμε στο σώμα του υποπρογράμματος, προτού ελέγξουμε την ύπαρξή τους.

Η συνάρτηση **PRESENT**() παρέχεται από την Fortran 95 για να ελέγχει αν το όρισμά της, που πρέπει να είναι ένα από τα προαιρετικά ορίσματα τού υποπρογράμματος που την καλεί, έχει τιμή. Αν κατά την κλήση του υποπρογράμματος το προαιρετικό όρισμα, π.χ. a, αποκτά τιμή, η τιμή του **PRESENT**(a) είναι .**TRUE**.. Αντίστοιχα, το **PRESENT**(a) είναι .**FALSE**. αν το όρισμα δεν προσδιορίστηκε στην κλήση.

Παράδειγμα της δυνατότητας που περιγράψαμε είναι το ακόλουθο:

```
FUNCTION f(x,a)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT(in) :: x
  DOUBLE PRECISION, INTENT (in), OPTIONAL :: a
  DOUBLE PRECISION :: f

  IF (PRESENT(a)) THEN
     f = x + a
  ELSE
     f = x + 10.0d0
  END IF

END FUNCTION f
```

Στον παραπάνω ορισμό, το δεύτερο όρισμα της f() είναι προαιρετικό. Αν υπάρχει, χρησιμοποιείται για να υπολογιστεί η επιστρεφόμενη τιμή· στην αντίθετη περίπτωση, ο υπολογισμός γίνεται χωρίς αυτό.

Η κλήση της f() μπορεί να γίνει με δύο τρόπους, όπως στον παρακάτω κώδικα

```
PROGRAM add
  IMPLICIT NONE
  INTERFACE
     FUNCTION f(x,a)
       IMPLICIT NONE
       DOUBLE PRECISION, INTENT(in) :: x
```

```
      DOUBLE PRECISION, INTENT (in), OPTIONAL :: a
      DOUBLE PRECISION :: f
    END FUNCTION f
  END INTERFACE

  WRITE (*,*) f(3.0d0)
  WRITE (*,*) f(3.0d0, 20d0)
END PROGRAM add
```

8.8 Στατικές ποσότητες

Αναφέραμε ότι οι μεταβλητές που ορίζονται στο σώμα ενός υποπρογράμματος έχουν διάρκεια ζωής όση και η διάρκεια εκτέλεσης του υποπρογράμματος. Δημιουργούνται κάθε φορά που η ροή εκτέλεσης συναντά τη δήλωσή τους και καταστρέφονται όταν τελειώσει το υποπρόγραμμα στο οποίο ανήκουν.

Μπορούμε όμως να ορίσουμε κατάλληλα κάποια μεταβλητή έτσι ώστε να δημιουργηθεί μία φορά και να μην καταστραφεί κατά την έξοδο της ροής από το υποπρόγραμμα (και να διατηρήσει, επομένως, την τιμή της). Αυτό γίνεται προσθέτοντας στον ορισμό της μεταβλητής την προκαθορισμένη λέξη SAVE ή, ισοδύναμα, δίνοντας αρχική τιμή κατά τη δήλωση. Έτσι στην υπορουτίνα

```
SUBROUTINE f(a)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: a

  INTEGER, SAVE :: k = 0

  k = k + 1

! ...
END SUBROUTINE f
```

η μεταβλητή k ουσιαστικά μετρά πόσες φορές κλήθηκε το υποπρόγραμμα. Εννοείται ότι «φαίνεται» μόνο μέσα στην f().

Η δημιουργία μια μεταβλητής με SAVE και η τυχόν εκχώρηση αρχικής τιμής γίνεται κατά τη διάρκεια της μεταγλώττισης. Η αρχική τιμή πρέπει να είναι σταθερή ποσότητα που μπορεί να υπολογιστεί σε αυτό το στάδιο. Κατά την εκτέλεση του προγράμματος, η δήλωση και η εκχώρηση τιμής αγνοούνται.

8.9 Αναδρομική (recursive) κλήση συνάρτησης

Πολύ συχνά συναντούμε προβλήματα που ορίζονται αναδρομικά. Π.χ. ο ορισμός του παραγοντικού ενός ακεραίου είναι

$$n! = \begin{cases} 1 \times 2 \times \cdots \times (n-1) \times n = (n-1)! \times n\ , & n > 0\ , \\ 1\ , & n = 0\ . \end{cases}$$

Παρατηρούμε ότι για να υπολογίσουμε το παραγοντικό ενός αριθμού n με τον συγκεκριμένο τύπο, πρέπει να υπολογίσουμε το παραγοντικό ενός άλλου αριθμού (του $n-1$). Η υλοποίηση αυτού του τύπου σε μια γλώσσα προγραμματισμού απαιτεί μια συνάρτηση που, προτού ολοκληρωθεί, καλεί τον εαυτό της· για να βγάλει, δηλαδή, αποτέλεσμα πρέπει να χρησιμοποιήσει την ίδια. Φυσικά, η κλήση αυτή πρέπει να γίνεται υπό κάποια συνθήκη, αλλιώς δε θα επιστραφεί ποτέ τιμή.

Στη Fortran 95 επιτρέπεται σε μια συνάρτηση να καλεί τον εαυτό της. Μια συνάρτηση που καλεί τον εαυτό της απλοποιεί πολύ οποιοδήποτε πρόβλημα, αρκεί ο αλγόριθμος επίλυσής του να μπορεί να γραφεί ώστε:

- ο υπολογισμός του αποτελέσματος να χρειάζεται την εφαρμογή του ίδιου αλγόριθμου αλλά σε διαφορετικές «τιμές» για τα δεδομένα εισόδου από αυτές που δέχτηκε αρχικά,
- ο αλγόριθμος να μπορεί να υπολογίσει το αποτέλεσμα για ένα συγκεκριμένο σύνολο «τιμών» με άλλο τρόπο και όχι με εφαρμογή του εαυτού του. Το συγκεκριμένο σύνολο πρέπει να μπορεί να το «φτάσει» σε κάποια από τις διαδοχικές εφαρμογές του εαυτού του.

Παράδειγμα

Ας δούμε πώς μπορούμε να γράψουμε μια συνάρτηση για το παραγοντικό ενός ακεραίου με *αναδρομικό (recursive)* τρόπο στην Fortran 95. Η πρώτη απόπειρα θα μπορούσε να είναι ως εξής

```
FUNCTION par(n)            ! όχι σωστό
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n == 0) THEN
     par = 1
  ELSE
     par = par(n-1) * n
  END IF
END FUNCTION par
```

Στη συνάρτηση αυτή έχουμε αγνοήσει τους ελέγχους που κανονικά πρέπει να γίνονται (το n να μην είναι αρνητικό και το αποτέλεσμα να μπορεί να αναπαρασταθεί στον τύπο της επιστρεφόμενης ποσότητας). Παρατηρήστε ότι το σώμα της συνάρτησης είναι πιστή μεταγραφή του μαθηματικού τύπου σε κώδικα Fortran, και αρκετά πιο απλή από την εναλλακτική υλοποίηση που έχουμε δει ως τώρα. Προσέξτε ότι η κλήση της `par()` στο σώμα της δεν είναι ανεξέλεγκτη· η ακολουθία `par(n)`→`par(n-1)` →`par(n-2)`→ ... σταματά (και επιστρέφεται τιμή που υπολογίζεται χωρίς την κλήση της) όταν το όρισμα γίνει 0.

Δυστυχώς, ο ορισμός αναδρομικής συνάρτησης στην Fortran 95 δεν μπορεί να γίνει με το γνωστό τρόπο που περιγράψαμε στο παρόν κεφάλαιο (χωρίς να υπάρχει ουσιαστικός λόγος). Ο σωστός ορισμός για τη συνάρτηση του παραγοντικού γράφεται ως εξής

```
RECURSIVE FUNCTION par(n) RESULT (p)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: p

  IF (n == 0) THEN
     p = 1
  ELSE
     p = par(n-1) * n
  END IF
END FUNCTION par
```

Προσέξτε ότι περιλάβαμε τη δεσμευμένη λέξη **RECURSIVE** (αναδρομικός) στον ορισμό και διαχωρήσαμε (με το **RESULT**) το *αποτέλεσμα* της συνάρτησης (μια νέα μεταβλητή με όνομα της επιλογής μας) από το *όνομα* της συνάρτησης. Πλέον, γίνεται δήλωση της μεταβλητής που αντιστοιχεί στο αποτέλεσμα, και σε αυτή τη μεταβλητή γίνεται εκχώρηση τιμής. Το όνομα της συνάρτησης δε δηλώνεται· με αυτό όμως γίνεται η κλήση της.

Αν εξαιρέσουμε τον λίγο πολύπλοκο τρόπο δήλωσης (που μπορεί να χρησιμοποιηθεί και για τις μη αναδρομικές συναρτήσεις, παραλείποντας φυσικά το **RECURSIVE**), η υποστήριξη από την Fortran 95 του αναδρομικού μηχανισμού κλήσης αποτελεί μια σημαντική προσθήκη στις δυνατότητες της γλώσσας σε σχέση με τη Fortran 77.

Ας δούμε ένα άλλο παράδειγμα χρήσης της αναδρομικής συνάρτησης:

Στη Μαθηματική Φυσική υπάρχουν οικογένειες πολυωνύμων που έχουν κάποιες ειδικές ιδιότητες. Μία από αυτές τις οικογένειες, τα πολυώνυμα Hermite, εμφανίζεται στην κβαντομηχανική αντιμετώπιση του αρμονικού ταλαντωτή. Κάποια από αυτά είναι

$$\begin{aligned} H_0(x) &= 1\,, \\ H_1(x) &= 2x\,, \end{aligned}$$

$$\begin{aligned} H_2(x) &= 4x^2 - 2 \,, \\ H_3(x) &= 8x^3 - 12x \,, \\ H_4(x) &= 16x^4 - 48x^2 + 12 \,, \\ &\vdots \end{aligned}$$

Τα πολυώνυμα $H_n(x)$ ικανοποιούν την αναδρομική σχέση:

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \,, \qquad n \geq 2 \,.$$

Παρατηρήστε ότι χρειαζόμαστε τα πολυώνυμα μηδενικού βαθμού ($H_0(x) = 1$) και πρώτου βαθμού ($H_1(x) = 2x$) για να υπολογίσουμε από την αναδρομική σχέση το πολυώνυμο δεύτερου βαθμού. Ανάλογα, χρειαζόμαστε τα $H_1(x)$ και $H_2(x)$ για να υπολογίσουμε το $H_3(x)$, κοκ.

Αν θελήσουμε να γράψουμε συνάρτηση που να υπολογίζει την τιμή των πολυωνύμων Hermite, $H_n(x)$, για κάποια τιμή του x, μπορούμε να μεταγράψουμε τον προηγούμενο μαθηματικό τύπο στην ακόλουθη αναδρομική συνάρτηση:

```
RECURSIVE FUNCTION hermite(n,x) RESULT (h)
  IMPLICIT NONE

  INTEGER,          INTENT (in) :: n
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION              :: h

  IF (n==0) h = 1.0d0
  IF (n==1) h = 2.0d0 * x
  IF (n>=2) h = 2.0d0 * (x * hermite(n-1,x) - (n-1) * hermite(n-2,x))
END FUNCTION hermite
```

Παρατήρηση: Μπορούμε να έχουμε αναδρομικές υπορουτίνες (που καλούν, δηλαδή, τον εαυτό τους) αν συμπληρώσουμε τη δήλωση με τη δεσμευμένη λέξη **RECURSIVE**· *δεν συμπληρώνουμε τον ορισμό με το RESULT* καθώς η υπορουτίνα δεν επιστρέφει τιμή.

8.10 Υποπρογράμματα κατά στοιχείο (ELEMENTAL)

Όπως έχουμε περιγράψει, τα υποπρογράμματα κατά την κλήση τους απαιτούν να δίνονται τιμές στα ορίσματά τους κατάλληλου τύπου. Δεν είναι δυνατόν σε υποπρόγραμμα που περιμένει π.χ. ακέραια μεταβλητή να δώσουμε πραγματική, ή πίνακα ή σύνθετο τύπο. Όμως, οι ενσωματωμένες συναρτήσεις και υπορουτίνες της Fortran έχουν την ιδιότητα να δέχονται και απλές μεταβλητές και πίνακες. Έτσι, μπορούμε να έχουμε

```
DOUBLE PRECISION :: x, y
```

```
x = 5.3d0
y = SQRT(x)
```

αλλά και

```
DOUBLE PRECISION, DIMENSION (100) :: x, y
x(1) = ....
x(2) = ....
...
x(100) = ....

y = SQRT(x)
```

Η τελευταία εντολή ισοδυναμεί με

```
y(1) = SQRT(x(1))
y(2) = SQRT(x(2))
...
y(100) = SQRT(x(100))
```

Αυτή την ιδιότητα μπορούμε να την επεκτείνουμε και για τα υποπρογράμματα που ορίζει ο χρήστης αν συμπληρώσουμε τη δήλωση μιας υπορουτίνας ή συνάρτησης που έχει μόνο απλά ορίσματα με τη δεσμευμένη λέξη **ELEMENTAL**. Π.χ. η συνάρτηση

```
ELEMENTAL FUNCTION myexp(x)
  IMPLICIT NONE
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION :: myexp

  INTEGER :: i
  DOUBLE PRECISION :: term

  myexp = 0.0d0
  term = 1.0d0
  DO i = 0, 10
    myexp = myexp + term
    term = term * x / (i+1)
  END DO
END FUNCTION myexp
```

μπορεί να δεχτεί ως όρισμα είτε ένα πραγματικό αριθμό (ή μεταβλητή), είτε ένα πραγματικό διάνυσμα ή πίνακα. Το αποτέλεσμα θα είναι αντίστοιχα ένας πραγματικός αριθμός ή ένας πίνακας πραγματικών αριθμών με την ίδια διάσταση και σχήμα όπως ο πίνακας στο όρισμα.

Ένα υποπρόγραμμα μπορεί να οριστεί ως **ELEMENTAL** αν δεν έχει στατικές μεταβλητές (§8.8), δεν περιέχει εντολή **STOP** (§2.10.1) και δεν γράφει σε ή διαβάζει από εξωτερικό αρχείο. Επιπλέον, όλα τα ορίσματά του πρέπει να είναι απλές μεταβλητές.

Μια συνάρτηση που είναι **ELEMENTAL** πρέπει να έχει όλα τα ορίσματά της προσδιορισμένα με το **INTENT** (**in**).

Μία υπορουτίνα μπορεί να είναι **ELEMENTAL** αν στα ορίσματά της έχουμε προσδιορίσει κάποιο **INTENT**, όχι απαραίτητα **INTENT** (**in**).

8.11 MODULE

Ας τονίσουμε σε αυτό το σημείο ότι ποσότητες με το ίδιο όνομα, δηλωμένες σε διαφορετικά υποπρογράμματα (ή στο κυρίως πρόγραμμα), είναι τελείως ανεξάρτητες, δεν έχουν καμία σχέση μεταξύ τους. Επιπλέον, δεν υπάρχει τρόπος ώστε ένα υποπρόγραμμα να χρησιμοποιήσει ποσότητα που ορίζεται σε άλλο τμήμα του κώδικά μας. Αν επιθυμούμε κάτι τέτοιο θα πρέπει να περάσουμε τη συγκεκριμένη ποσότητα ως όρισμα στο υποπρόγραμμα.

Συχνά υπάρχει η ανάγκη να χρησιμοποιήσουμε μια ποσότητα σε πολλά υποπρογράμματα (ή και στο κυρίως πρόγραμμα). Οι τιμές μαθηματικών ή φυσικών σταθερών (π.χ. η τιμή του π), οι παράμετροι του προβλήματός μας (πλήθος αντικειμένων που περιγράφονται από τον κώδικά μας, διαστάσεις πινάκων κλπ.), οι δηλώσεις παραγόμενων τύπων κλπ., είναι απαραίτητο, σύμφωνα με όσα είπαμε, να επαναλαμβάνονται, οριζόμενες σε κάθε υποπρόγραμμα που θα τις προσπελάσει. Οποιαδήποτε αλλαγή στην τιμή τους ή στον τύπο τους (π.χ. για αύξηση της ακρίβειας με την οποία ορίζονται οι σταθερές) πρέπει φυσικά να γίνει σε όλα τα σημεία στα οποία αυτές ορίζονται. Σε μια τέτοια διαδικασία είναι πολύ εύκολο να γίνει λάθος ή να μην γίνει παντού η τροποποίηση.

Το τέταρτο στοιχείο ομαδοποίησης κώδικα (μετά το κυρίως πρόγραμμα, τη συνάρτηση και την υπορουτίνα), το module, διευκολύνει πολύ όταν θέλουμε να ορίσουμε μια ποσότητα *μία φορά* και να τη χρησιμοποιήσουμε σε πολλά σημεία. Συντάσσεται ως εξής: στην αρχή του κώδικά μας, έξω από το κυρίως πρόγραμμα ή τα υποπρογράμματα και πριν την πρώτη χρήση του, γράφουμε

```
MODULE onoma
```

```
END MODULE onoma
```

Το όνομα του **MODULE** επιλέγεται σύμφωνα με τους γνωστούς κανόνες που ισχύουν για τα ονόματα των μεταβλητών, §2.1.2. Στο εσωτερικό του μπορούμε να περιλάβουμε δηλώσεις σταθερών και μεταβλητών ποσοτήτων, ορισμούς παραγόμενων τύπων (Κεφάλαιο 6), δηλώσεις υποπρογραμμάτων (**INTERFACE**), ακόμα και ορισμούς υποπρογραμμάτων (μετά τη διαχωριστική λέξη **CONTAINS**), αλλά όχι κώδικα εκτός υποπρογράμματος.

Σε κάθε υποπρόγραμμα ή στο κυρίως πρόγραμμα ή σε άλλο **MODULE** στο οποίο χρειαζόμαστε τις ποσότητες που έχουν περιληφθεί σε **MODULE**, περιλαμβάνουμε την εντολή

```
USE onoma
```

πριν από τις δηλώσεις οποιασδήποτε ποσότητας (και πριν το **IMPLICIT NONE**). Με άλλα λόγια, η συγκεκριμένη εντολή ακολουθεί το **PROGRAM** ... ή το **FUNCTION** ... ή το **SUBROUTINE** ... ή το **MODULE** .... Το τμήμα του προγράμματός μας που κάνει «χρήση» ενός **MODULE**, αποκτά πρόσβαση σε όλες τις ποσότητες που ορίζονται και σε όλα τα υποπρογράμματα που δηλώνονται ή ορίζονται στο σχετικό **MODULE**[7].

Παράδειγμα χρήσης είναι το ακόλουθο:

```
MODULE constants
  IMPLICIT NONE

  DOUBLE PRECISION, PARAMETER :: pi = 3.14159265358979323846d0
END MODULE constants

PROGRAM mainprog
  USE constants

  IMPLICIT NONE
  !.....

  ! use pi
END PROGRAM mainprog

FUNCTION f(x)
  USE constants

  IMPLICIT NONE
  !.....

  ! use pi
END FUNCTION f
```

Όπως αναφέραμε, ένα **MODULE** μπορεί να περιέχει και δηλώσεις ποσοτήτων που δεν είναι σταθερές. Αποτελεί, επομένως, ένα μηχανισμό για να μεταφέρουμε τιμές από ένα υποπρόγραμμα σε κάποιο άλλο. Καθώς οι μεταβλητές ενός module είναι προσβάσιμες από όσα υποπρογράμματα κάνουν «χρήση» αυτού, υπάρχει η δυνατότητα κάποιο από αυτά να αποδίδει τιμή σε μια «κοινή» μεταβλητή και κάποιο άλλο να τη χρησιμοποιεί.

Συμπερασματικά, υπάρχουν δύο τρόποι για μεταφορά πληροφορίας μεταξύ διαφορετικών τμημάτων του κώδικά μας: είτε μέσω ορισμάτων είτε με τη χρήση **MODULE**. Προσέξτε όμως ότι το **INTERFACE** ενός υποπρογράμματος, όταν περιλαμβάνει στις δηλώσεις των ορισμάτων κατάλληλους προσδιορισμούς για το **INTENT**, αρκεί για να προσδιορίσουμε αν η μεταβλητή που του «περνάμε» ως όρισμα τροποποιείται. Αντίθετα, μόνο με την προσεκτική εξέταση *των εντολών* του υποπρογράμματος (που

[7] αρκεί να μην έχουν δηλωθεί ως **PRIVATE**.

μπορεί να βρίσκεται σε άλλο αρχείο) πληροφορούμαστε για το ποιες μεταβλητές από αυτές που «περνούν» μέσω **MODULE** αλλάζουν. Ο εντοπισμός σφαλμάτων σε ένα μεγάλο πρόγραμμα γίνεται πιο απλός όσο πιο εύκολα μπορούμε να προσδιορίσουμε ποιες μεταβλητές αλλάζουν και πού. Επομένως, *η χρήση των modules προτείνεται για τη «διάδοση» σταθερών ποσοτήτων, υποπρογραμμάτων και τύπων. Οι μεταβλητές ποσότητες είναι προτιμότερο να μεταφέρονται με ορίσματα.*

Η εισαγωγή του module στη Fortran 90 επιτρέπει την υποστήριξη του «προγραμματισμού βασιζόμενου σε αντικείμενα» (object based) από τη γλώσσα.

8.12 Ασκήσεις

1. Να γράψετε συνάρτηση που να δέχεται ως όρισμα έναν πραγματικό αριθμό x και να επιστρέφει την τιμή της ποσότητας $\mathrm{e}^{-x^2/2}$. Να τη χρησιμοποιήσετε στο πρόγραμμά σας για να υπολογίσετε και να τυπώσετε την τιμή της για $x = 0.3,\ 1.4,\ 5.6$.

2. Να γράψετε συνάρτηση που να δέχεται ως όρισμα την ακτίνα ενός κύκλου και να υπολογίζει το εμβαδόν του. Καλέστε την για ακτίνα $R = 2.3$.

3. Να γράψετε συνάρτηση που να δέχεται ως όρισμα έναν (μικρό) ακέραιο αριθμό k και να επιστρέφει το παραγοντικό του, $k!$. Να την καλέσετε για να υπολογίσετε και να τυπώσετε τα $3!$, $5!$, $7!$.

4. Να γράψετε συνάρτηση που να δέχεται ως ορίσματα τρεις πραγματικούς αριθμούς και να υπολογίζει το άθροισμα των τετραγώνων τους. Να τη χρησιμοποιήσετε για την τριάδα $(3.2, 5.6, 8.1)$.

5. Να γράψετε συνάρτηση που να ελέγχει αν το όρισμά της, ένας ακέραιος αριθμός, είναι πρώτος ή όχι (η ποσότητα που θα επιστρέφει ποιου τύπου είναι;). Να τη χρησιμοποιήσετε για να ελέγξετε τους αριθμούς 89, 261, 1511.

6. Να γράψετε συνάρτηση που να υπολογίζει και να επιστρέφει το μέσο όρο των στοιχείων ενός διανύσματος ακεραίων που θα δέχεται ως όρισμα.

7. Να γράψετε συνάρτηση που να υπολογίζει και να επιστρέφει το μικρότερο στοιχείο ενός διανύσματος ακεραίων που θα δέχεται ως όρισμα. (Ουσιαστικά, δηλαδή, υλοποιήστε την ενσωματωμένη συνάρτηση **`MINVAL`**`()`.)

8. Να γράψετε συνάρτηση που να υπολογίζει και να επιστρέφει τη *θέση* του μέγιστου στοιχείου ενός διανύσματος ακεραίων που θα δέχεται ως όρισμα. (Ουσιαστικά, δηλαδή, υλοποιήστε την ενσωματωμένη συνάρτηση **`MAXLOC`**`()`.)

9. Να γράψετε υποπρόγραμμα που να εναλλάσσει τις τιμές δύο πραγματικών μεταβλητών. Κατόπιν, να γράψετε πρόγραμμα το οποίο να χρησιμοποιεί το υποπρόγραμμα αυτό.

10. Να γράψετε συνάρτηση που να υπολογίζει και να επιστρέφει το εσωτερικό γινόμενο δύο διανυσμάτων πραγματικών αριθμών με ίδιο πλήθος στοιχείων, δηλαδή υπολογίστε το $\sum_i a_i b_i$ αν a, b είναι τα διανύσματα. (Ουσιαστικά, υλοποιήστε την ενσωματωμένη συνάρτηση **`DOT_PRODUCT`**`()`.)

11. Να γράψετε συνάρτηση που να επιστρέφει ένα τυχαίο ακέραιο σε διάστημα $[a, b]$ που θα προσδιορίζεται από τα ορίσματά της.

12. Γράψτε συνάρτηση που να δέχεται ως όρισμα ένα ακέραιο αριθμό n. Θα επιλέγει n τυχαία σημεία (x_i, y_i) στο τετράγωνο $1 \le x_i \le 1$, $1 \le y_i \le 1$ και

θα επιστρέφει το πηλίκο όσων βρίσκονται εντός ενός κύκλου με ακτίνα 1 ($x^2 + y^2 = 1^2$) προς τα συνολικά.

Καλέστε τη συνάρτηση για διάφορες μεγάλες τιμές του n: $n = 10^3$, $n = 10^4$, ..., $n = 10^9$. Παρατηρήστε ότι ο λόγος αυτός για πολύ μεγάλα n προσεγγίζει το λόγο του εμβαδού του κύκλου με διάμετρο 2 προς το εμβαδόν του τετραγώνου με πλευρά 2.

13. Στο αρχείο στη διεύθυνση https://tinyurl.com/bp7yzkdr περιέχονται ακέραιοι αριθμοί, ένας σε κάθε γραμμή. Η πρώτη γραμμή του αρχείου περιέχει το πλήθος των αριθμών που ακολουθούν. Γράψτε συνάρτηση που να μετρά πόσες φορές εμφανίζεται το όρισμά της, ένας ακέραιος αριθμός, στο αρχείο. Εφαρμόστε τη για τους αριθμούς 5744, 6789, 2774.

14. Η μετατροπή από καρτεσιανές συντεταγμένες, (x, y), σε πολικές συντεταγμένες, (r, θ), γίνεται με τις ακόλουθες σχέσεις

$$r = \sqrt{x^2 + y^2}\ , \qquad \theta = \tan^{-1}(y/x)\ .$$

Η αντίστροφη μετατροπή γίνεται με τις σχέσεις:

$$x = r\cos\theta\ , \qquad y = r\sin\theta\ .$$

Γράψτε δύο υποπρογράμματα που θα δέχονται από 4 ορίσματα, x, y, r, θ, και θα υλοποιούν αυτές τις μετατροπές. Το πρώτο θα δέχεται τιμές στα x, y και θα εξάγει τιμές στα r, θ, και το δεύτερο αντίστροφα.

Υπόδειξη: Προσέξτε την επιλογή που θα κάνετε για τη συνάρτηση της αντίστροφης εφαπτομένης.

15. Η μετατροπή από καρτεσιανές συντεταγμένες, (x, y, z), σε σφαιρικές συντεταγμένες, (r, θ, ϕ), γίνεται με τις ακόλουθες σχέσεις

$$r = \sqrt{x^2 + y^2 + z^2}\ , \qquad \theta = \cos^{-1}\left(z/\sqrt{x^2 + y^2 + z^2}\right)\ , \qquad \phi = \tan^{-1}(y/x)\ .$$

Γράψτε ένα υποπρόγραμμα που να κάνει αυτή τη μετατροπή. Θα δέχεται έξι ορίσματα: τρεις πραγματικούς αριθμούς για τις καρτεσιανές συντεταγμένες και τρεις για τις σφαιρικές. Χρησιμοποιήστε το σε πρόγραμμά σας για να τυπώσετε στην οθόνη τις σφαιρικές συντεταγμένες που αντιστοιχούν στα σημεία $(3.5, 2.5, -1.0)$ και $(0.0, 1.5, -2.0)$.

Υπόδειξη: Προσέξτε την επιλογή που θα κάνετε για τη συνάρτηση της αντίστροφης εφαπτομένης.

16. Δίνεται η καμπύλη

$$r(\theta) = \mathrm{e}^{\sin\theta} - 2\cos(4\theta) + \sin^5[(2\theta - \pi)/24]$$

σε πολικές συντεταγμένες. Υπολογίστε τις τιμές $r_i = r(\theta_i)$ για τις τιμές του θ $\theta_i = \{0°,2°,4°,...,358°\}$. Γράψτε πρόγραμμα που να τυπώνει στο αρχείο με όνομα *butterfly.txt* τα σημεία (x_i, y_i) που αντιστοιχούν στις πολικές συντεταγμένες (r_i, θ_i). Χρησιμοποιήστε ένα από τα υποπρογράμματα που γράψατε στην άσκηση 14. Η καμπύλη που σχηματίζεται είναι η «καμπύλη πεταλούδας».

17. Η ακόλουθη συνάρτηση δίνει προσεγγιστικά την τιμή του π για οποιαδήποτε τιμή του ακέραιου N:

$$p(N) = \frac{4}{N} \sum_{k=1}^{N} \frac{1}{1 + \left(\frac{k-\frac{1}{2}}{N}\right)^2} .$$

Μπορεί να δειχθεί ότι $\lim_{N\to\infty} p(N) = \pi$.

- Γράψτε πρόγραμμα που να υπολογίζει την προσεγγιστική τιμή του π με τη χρήση της συγκεκριμένης συνάρτησης για $N = 1, 2, 10, 50, 100, 500$. Για κάθε τιμή του N τυπώστε στην οθόνη την προσεγγιστική τιμή και την απόκλιση αυτής από την ακριβή τιμή του π, δηλαδή το $e(N) = |p(N) - \pi|$.
- Βρείτε τη μικρότερη τιμή $N_{\min}$ που ικανοποιεί τη σχέση $e(N_{\min}) < 10^{-6}$.

18. Ένας μη αρνητικός ακέραιος αριθμός K μικρότερος του 1024 ($= 2^{10}$) μπορεί να αναλυθεί σε άθροισμα δυνάμεων του 2:

$$K = d_9 2^9 + d_8 2^8 + \cdots d_1 2^1 + d_0 2^0 .$$

Οι συντελεστές d_9, d_8,..., d_1, d_0 αποτελούν τα ψηφία της αναπαράστασης του K στο δυαδικό σύστημα.

Γράψτε υποπρόγραμμα που να δέχεται ως πρώτο όρισμα ένα ακέραιο και ως δεύτερο ένα διάνυσμα 10 θέσεων. Το υποπρόγραμμα θα υπολογίζει τα δυαδικά ψηφία d_0, ..., d_9 για τον ακέραιο και θα τα αποθηκεύει στο διάνυσμα. Κατόπιν, χρησιμοποιήστε το για να βρείτε και να τυπώσετε στην οθόνη τα δυαδικά ψηφία των αριθμών 81, 833, 173.

Υπόδειξη: Αν το K είναι ακέραιος γραμμένος στη δυαδική αναπαράσταση, πόσο κάνει `MOD(K,2)`; Πόσο κάνει `K/2`;

19. Να γράψετε συνάρτηση που να ελέγχει αν το όρισμά της, ένας θετικός ακέραιος αριθμός, είναι αριθμός Mersenne. Ένας ακέραιος αριθμός k είναι αριθμός Mersenne αν το $k+1$ είναι δύναμη του 2. Βρείτε τους αριθμούς Mersenne μέχρι το 10000.

20. Να γράψετε συνάρτηση που να υπολογίζει το e^x από τον τύπο

$$\mathrm{e}^x \approx x^0/0! + x^1/1! + x^2/2! + \cdots + x^{12}/12! .$$

Να βρείτε με αυτή τη συνάρτηση τις τιμές του e^x για $x = 0.5$, 1.2, 4.1.

21. Να γράψετε συνάρτηση που να υπολογίζει το ημίτονο από τον τύπο

$$\sin x \approx x^1/1! - x^3/3! + x^5/5! - x^7/7! + x^9/9! - x^{11}/11! \,.$$

Βασιστείτε στο ότι ο κάθε όρος στο άθροισμα προκύπτει από τον προηγούμενό του με πολλαπλασιασμό κατάλληλης ποσότητας. Να χρησιμοποιήσετε τη συνάρτησή σας για να υπολογίσετε το ημίτονο των $35°$.

22. Γράψτε συναρτήσεις που να υπολογίζουν και να επιστρέφουν τα e^x, $\sin x$, $\cos x$ από τις σχέσεις

$$\mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \,, \qquad \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \,, \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} \,.$$

Για τη διευκόλυνσή σας παρατηρήστε ότι ο κάθε όρος στα αθροίσματα προκύπτει από τον αμέσως προηγούμενο αν αυτός πολλαπλασιαστεί με κατάλληλη ποσότητα.

Στα αθροίσματα να σταματάτε τον υπολογισμό τους όταν ο όρος που πρόκειται να προστεθεί είναι κατ' απόλυτη τιμή μικρότερος από 10^{-10}.

Χρησιμοποιήστε τις συγκεκριμένες συναρτήσεις για να βρείτε και να τυπώσετε τις τιμές $\mathrm{e}^{-0.2}$, $\sin(37°)$, $\cos(\pi/3)$. Συγκρίνετε τις τιμές που υπολογίζετε εσείς με τις τιμές που προκύπτουν από τις αντίστοιχες ενσωματωμένες συναρτήσεις.

23. Γράψτε συνάρτηση που να δέχεται δύο διανύσματα πραγματικών αριθμών, με οποιοδήποτε πλήθος στοιχείων. Η συνάρτηση να ελέγχει αν όλα τα στοιχεία του δεύτερου διανύσματος περιέχονται στο πρώτο και να επιστρέφει το αποτέλεσμα σε τιμή λογικού τύπου.

Αποθηκεύστε στον υπολογιστή σας το αρχείο στη διεύθυνση `https://tinyurl.com/yjyjnywh`. Περιέχει 126 πραγματικούς αριθμούς, τον καθένα σε ξεχωριστή σειρά. Γράψτε πρόγραμμα που να χρησιμοποιεί τη συνάρτηση που γράψατε για να ελέγξετε αν οι αριθμοί {7.6, 3.2, 9.1} περιέχονται στους αριθμούς του αρχείου.

24. Γράψτε συνάρτηση που να *ελέγχει* αν το όρισμά της, ένα διάνυσμα ακεραίων, είναι ταξινομημένο κατά αύξουσα σειρά (από το μικρότερο στο μεγαλύτερο). Να επιστρέφει μία ποσότητα λογικού τύπου.

Χρησιμοποιήστε τη συνάρτηση που γράψατε σε πρόγραμμά σας για να ελέγξετε αν τα 200 πρώτα στοιχεία του αρχείου στο `https://tinyurl.com/4e6u7nxa` είναι ταξινομημένα. Το πρόγραμμά σας να γράφει το σχετικό μήνυμα στην οθόνη.

25. Γράψτε συνάρτηση που να δέχεται ως ορίσματα δύο μιγαδικούς αριθμούς και ένα χαρακτήρα. Ο χαρακτήρας θα είναι ένας από τους '+', '-', '*', '/'. Οποιοσδήποτε άλλος δεν είναι αποδεκτός και θα προκαλεί την εκτύπωση ενός μηνύματος που θα ενημερώνει τον χρήστη για το λάθος του και θα διακόπτεται η

εκτέλεση της συνάρτησης (με την εντολή **RETURN**). Ανάλογα με το χαρακτήρα, η συνάρτηση θα υπολογίζει την αντίστοιχη πράξη μεταξύ των μιγαδικών ορισμάτων και θα επιστρέφει το αποτέλεσμα. Το πρόγραμμά σας να την καλεί και να τυπώνει το αποτέλεσμα του πολλαπλασιασμού των αριθμών $2 + 3\mathrm{i}$, $5.7 - 9\mathrm{i}$.

26. Να γράψετε δύο υποπρογράμματα που μετατρέπουν θερμοκρασία από βαθμούς Κελσίου σε βαθμούς Φαρενάιτ και αντίστροφα. Η σχέση των κλιμάκων Κελσίου (C) και Φαρενάιτ (F) είναι γραμμική. Ο τύπος μετατροπής είναι $F = 9/5\,C + 32$.

    Να τα χρησιμοποιήσετε στο πρόγραμμά σας για να υπολογίσετε τη θερμοκρασία σε βαθμούς Φαρενάιτ για:

    - τη θερμοκρασία $22\,^{\circ}\mathrm{C}$,
    - τη θερμοκρασία του απόλυτου 0 ($-273.15\,^{\circ}\mathrm{C}$),
    - τη μέση θερμοκρασία της επιφάνειας του Ήλιου ($6000\,^{\circ}\mathrm{C}$).

    και τη θερμοκρασία σε βαθμούς Κελσίου για τους $100\,^{\circ}\mathrm{F}$.

27. Να γράψετε υποπρόγραμμα που να επιλύει τη δευτεροβάθμια εξίσωση $ax^2 + bx + c = 0$ και να μας επιστρέφει τις λύσεις. Προσέξτε να κάνετε διερεύνηση ανάλογα με τις τιμές των a, b, c, που θα δέχεται ως ορίσματα. Το υποπρόγραμμά σας δε θα τυπώνει τις τιμές των ριζών αλλά θα τις επιστρέφει σε ορίσματα.

28. Δύο διδιάστατοι πραγματικοί πίνακες περιέχονται στα αρχεία στις διευθύνσεις https://tinyurl.com/yradd65z και https://tinyurl.com/54r8z9y3. Η πρώτη γραμμή σε κάθε αρχείο είναι ο αριθμός των γραμμών και η δεύτερη ο αριθμός των στηλών. Οι επόμενες γραμμές περιέχουν τα στοιχεία των πινάκων κατά γραμμή, από αριστερά προς τα δεξιά, δηλαδή διαδοχικά τα στοιχεία $(1,1)$, $(1,2)$, ..., $(2,1)$, $(2,2)$, ... για κάθε πίνακα.

    - Να γράψετε υποπρόγραμμα που να διαβάζει τα στοιχεία από ένα αρχείο και να τα εκχωρεί σε πίνακα. Ως ορίσματα θα δέχεται τον πίνακα και το όνομα του αρχείου. Χρησιμοποιήστε το για να δώσετε τιμές σε δύο πίνακες A,B.
    - Να γράψετε άλλο υποπρόγραμμα που να υπολογίζει το γινόμενο των δύο πινάκων (χωρίς τη χρήση ενσωματωμένης συνάρτησης).
    - Να γράψετε υποπρόγραμμα που να τυπώνει το όρισμά του, ένα πραγματικό πίνακα, στοιχισμένο κατά στήλες, με 4 δεκαδικά ψηφία σε κάθε στοιχείο.

    Χρησιμοποιώντας τα παραπάνω υποπρογράμματα, γράψτε ένα πρόγραμμα που να διαβάζει τους δύο πίνακες και να τυπώνει στην οθόνη το γινόμενό τους.

29. Γράψτε υποπρόγραμμα που να υπολογίζει τον ερμιτιανό συζυγή ενός τετραγωνικού πίνακα μιγαδικών αριθμών. Ο συζυγής να αποθηκεύεται στον αρχικό πίνακα. Εφαρμόστε το για τον πίνακα

$$\begin{pmatrix} 2.3-\mathrm{i} & 1-7\mathrm{i} & 5.8 & -2.9-3.7\mathrm{i} \\ -4.9\mathrm{i} & \mathrm{i} & 9-0.3\mathrm{i} & -2+0.72\mathrm{i} \\ 8.2+4\mathrm{i} & -0.8+\mathrm{i} & 0.2+5\mathrm{i} & 9-3\mathrm{i} \\ 2.3\mathrm{i} & -7.1+9\mathrm{i} & 0.9 & -4\mathrm{i} \end{pmatrix}$$

Γίνεται με χρήση ενσωματωμένων συναρτήσεων;

30. Η απομάκρυνση από τη θέση ισορροπίας μιας μπάλας στην άκρη ενός ελατηρίου περιγράφεται χρονικά από την εξίσωση $x(t) = A\cos(\omega t) + B\sin(\omega t)$, με $A = 3\,\mathrm{cm}$, $B = 2\,\mathrm{cm}$, $\omega = 12\,\mathrm{Hz}$.

    (α΄) Να γράψετε συνάρτηση που να δέχεται το χρόνο t και να επιστρέφει την αντίστοιχη απομάκρυνση $x(t)$.

    (β΄) Να γράψετε πρόγραμμα που να χρησιμοποιεί τη συνάρτηση για να τυπώσει στο αρχείο *data* τις τιμές των t και $x(t)$ με 4 δεκαδικά ψηφία, για $t = 0.0,\ 0.5,\ 1.0,\ ...,100.0\,\mathrm{s}$. Κάθε ζεύγος τιμών να είναι σε ξεχωριστή γραμμή.

31. Να γράψετε αναδρομικό υποπρόγραμμα που να υπολογίζει το παραγοντικό ενός ακέραιου αριθμού βασιζόμενοι στο ότι

$$n! = \begin{cases} (n-1)! \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

32. Να γράψετε αναδρομική συνάρτηση που να δέχεται ένα ακέραιο n και να επιστρέφει τον n-οστό αριθμό της ακολουθίας Fibonacci. Χρησιμοποιήστε τη για να τυπώσετε τους 15 πρώτους όρους της ακολουθίας.

33. Να γράψετε αναδρομική συνάρτηση που να ελέγχει αν το όρισμά της, μια ακέραιη ποσότητα, είναι δύναμη του 2. Να επιστρέφει τιμή λογικού τύπου. Να τη χρησιμοποιήσετε για να ελέγξετε αν οι αριθμοί 4096, 65534, 1855932 είναι δυνάμεις του 2.

34. (α΄) Γράψτε υποπρόγραμμα που θα δέχεται ως πρώτο όρισμα ένα ακέραιο αριθμό, θα τον αναλύει στα ψηφία του και θα τα αποθηκεύει στο δεύτερο όρισμά του, ένα διάνυσμα τουλάχιστον 10 θέσεων.

    (β΄) Ένας θετικός ακέραιος αριθμός χαρακτηρίζεται ως παλίνδρομος αν «διαβάζεται» το ίδιο από αριστερά προς τα δεξιά και αντίστροφα. Ο παλίνδρομος αριθμός δηλαδή έχει ίδια το πρώτο (μη μηδενικό) και το τελευταίο ψηφίο, το δεύτερο και το προτελευταίο κλπ. Π.χ. οι ακέραιοι 19791, 4774 είναι παλίνδρομοι.

    Γράψτε υποπρόγραμμα που θα δέχεται ένα ακέραιο αριθμό και θα ελέγχει αν είναι παλίνδρομος.

(γ΄) Γράψτε σε αρχείο με όνομα *palindrome.dat* όλους τους παλίνδρομους ακέραιους που είναι γινόμενο δύο τριψήφιων αριθμών. Τον μεγαλύτερο από αυτούς γράψτε τον στην οθόνη μαζί με τους τριψήφιους αριθμούς που τον παρήγαγαν.

35. Γράψτε κώδικα που να τυπώνει στο αρχείο *palindrome.dat* όλους τους παλίνδρομους αριθμούς (δείτε τον ορισμό τους στην άσκηση 34) μέχρι το 1000000. Στην οθόνη να τυπώνει το πλήθος τους.

36. Γράψτε συνάρτηση που θα δέχεται τρία διανύσματα πραγματικών αριθμών, έστω a,b,c. Τα διανύσματα a,b θα έχουν οποιοδήποτε πλήθος στοιχείων ενώ το c θα θεωρούμε ότι έχει τουλάχιστον όσες θέσεις έχει το «μικρότερο» διάνυσμα από τα a,b (δηλαδή, αυτό που έχει τα λιγότερα στοιχεία). Η συνάρτησή σας θα αντιγράφει τα κοινά στοιχεία των a,b στο διάνυσμα c και θα επιστρέφει το πλήθος τους.

    Χρησιμοποιήστε τη συνάρτηση αυτή για να βάλετε σε ένα διάνυσμα τα κοινά στοιχεία των συνόλων $\{1.2, 3.4, 5.7, 8.8, 9.2, 6.5\}$ και $\{3.4, 4.3, 6.5, 5.7\}$. Τυπώστε στην οθόνη τα στοιχεία του διανύσματος που προκύπτει. Ο κάθε αριθμός που θα τυπώνετε να έχει 1 δεκαδικό ψηφίο.

    Παρατήρηση: Προφανώς, τα κοινά στοιχεία των a,b μπορούν να είναι κανένα, κάποια ή όλα τα στοιχεία του «μικρότερου» διανύσματος.

37. Γράψτε μια συνάρτηση με όνομα digit που να δέχεται δύο ακέραια ορίσματα, N και d. Η συνάρτηση θα επιστρέφει το ψηφίο στη θέση d του αριθμού N. Προσέξτε ότι το N μπορεί να είναι αρνητικός. Θεωρούμε ότι στην πρώτη θέση είναι το ψηφίο των μονάδων. Για παράδειγμα, η κλήση της digit με N=57960, d=2 πρέπει να επιστρέφει τον αριθμό 6. Αν το d είναι μεγαλύτερο από το πλήθος των ψηφίων του N, η συνάρτηση θα επιστρέφει 0.

    Το αρχείο στη διεύθυνση `https://tinyurl.com/293zdr6v` περιέχει 3590 ακέραιους, σε ξεχωριστή γραμμή ο καθένας. Αποθηκεύστε στον υπολογιστή σας. Χρησιμοποιήστε τη συνάρτηση που γράψατε για να βρείτε το τρίτο ψηφίο των αριθμών του αρχείου. Τα ψηφία που θα βρείτε να τα γράψετε στο αρχείο *digit.txt*, ένα σε κάθε σειρά.

38. Γράψτε συνάρτηση που να δέχεται ένα διάνυσμα με ακέραια στοιχεία και να εντοπίζει και να επιστρέφει το στοιχείο που εμφανίζεται τις περισσότερες φορές συνεχόμενα (ή το τελευταίο από όσα εμφανίζονται με ίδιο πλήθος). Ελέγξτε την για τη σειρά στοιχείων $\{2, 8, 8, 3, 5, 5, 5, 8, 8, 1, 6, 7, 7, 7\}$· θα πρέπει να βρει το 7.

39. Η στροφή ενός τριδιάστατου διανύσματος $\vec{r} = (x, y, z)$ κατά γωνία θ γύρω από τον άξονα $\hat{x}$, μπορεί να αναπαρασταθεί με τον πολλαπλασιασμό του

διανύσματος $\vec{r}$ με τον πίνακα

$$R_x(\theta) = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{vmatrix} ,$$

δηλαδή, το στραμμένο διάνυσμα έχει συνιστώσες

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{vmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} .$$

Οι πίνακες στροφής γύρω από τους άξονες $\hat{y}$, $\hat{z}$ είναι αντίστοιχα οι

$$R_y(\theta) = \begin{vmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{vmatrix}$$

και

$$R_z(\theta) = \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} .$$

(α′) Να γράψετε τρεις συναρτήσεις· η κάθε μια από αυτές θα εκτελεί τη στροφή γύρω από έναν άξονα. Κάθε συνάρτηση θα δέχεται ως ορίσματα

- τη γωνία στροφής θ και
- ένα διάνυσμα, οι συνιστώσες του οποίου θα τροποποιούνται.

(β′) Να γράψετε συναρτήσεις που θα υπολογίζουν το μέτρο ενός διανύσματος και τη γωνία μεταξύ δύο διανυσμάτων.

(γ′) Να γράψετε πρόγραμμα που θα χρησιμοποιεί τα παραπάνω για να κάνετε τα εξής:

i. Δημιουργήστε ένα διάνυσμα με συνιστώσες $x = 0.5$, $y = -0.3$, $z = 1.2$. Να το στρέψετε διαδοχικά κατά γωνία 30° γύρω από τον άξονα $\hat{y}$, κατόπιν κατά γωνία 35° γύρω από τον άξονα $\hat{x}$ και τέλος κατά γωνία 58° γύρω από τον άξονα $\hat{z}$. Τυπώστε στην οθόνη τις τελικές συνιστώσες.

ii. Υπολογίστε και τυπώστε στην οθόνη τα μέτρα του αρχικού και του τελικού (μετά τις στροφές) διανύσματος καθώς και τη μεταξύ τους γωνία.

40. Στη Μαθηματική Φυσική χρησιμοποιείται η οικογένεια πολυωνύμων Hermite, $H_n(x)$. Ο βαθμός n του πολυωνύμου είναι ακέραιος, 0, 1,.... Τα πρώτα πολυώνυμα Hermite είναι

$$H_0(x) = 1$$

$$\begin{aligned} H_1(x) &= 2x \\ H_2(x) &= 4x^2 - 2 \\ \vdots &= \vdots \end{aligned}$$

Για τα πολυώνυμα Hermite ισχύει η αναδρομική σχέση

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \ , \qquad n \geq 2 \ .$$

Γράψτε υποπρόγραμμα που να υπολογίζει την τιμή ενός πολυωνύμου Hermite. Αυτό θα δέχεται ως ορίσματα έναν ακέραιο αριθμό n που θα αντιπροσωπεύει το βαθμό του πολυωνύμου και ένα πραγματικό x που θα είναι το σημείο υπολογισμού. Θα επιστρέφει την τιμή του $H_n(x)$.

41. Στη Μαθηματική Φυσική εμφανίζεται η οικογένεια πολυωνύμων Bessel, $y_n(x)$. Η τάξη n του πολυωνύμου είναι ακέραια, $0, 1, \ldots$. Τα πρώτα πολυώνυμα Bessel είναι

$$\begin{aligned} y_0(x) &= 1 \\ y_1(x) &= x + 1 \\ y_2(x) &= 3x^2 + 3x + 1 \\ \vdots &= \vdots \end{aligned}$$

Για τα πολυώνυμα Bessel ισχύουν οι εξής σχέσεις:

$$\begin{aligned} y_n(x) &= (2n-1)xy_{n-1}(x) + y_{n-2}(x) \qquad n \geq 2 \ , \\ x^2 y_n'(x) &= (nx-1)y_n(x) + y_{n-1}(x) \qquad n \geq 1 \ , \\ y_0'(x) &= 0 \ . \end{aligned}$$

Χρησιμοποιώντας τις παραπάνω σχέσεις,

(α΄) γράψτε συνάρτηση που να υπολογίζει την τιμή ενός πολυωνύμου Bessel. Αυτή θα δέχεται ως ορίσματα έναν ακέραιο αριθμό n, που θα αντιπροσωπεύει την τάξη του πολυωνύμου, και ένα πραγματικό x που θα είναι το σημείο υπολογισμού. Θα επιστρέφει την τιμή του $y_n(x)$. Χρησιμοποιήστε τη για να υπολογίσετε τα $y_3(1.2)$, $y_6(4.1)$.

(β΄) γράψτε συνάρτηση που να υπολογίζει την πρώτη παράγωγο του $y_n(x)$ (για $x \neq 0$). Χρησιμοποιήστε τη για να υπολογίσετε τα $y_3'(1.2)$, $y_6'(4.1)$.

42. Η κβαντομηχανική αντιμετώπιση του μονοδιάστατου αρμονικού ταλαντωτή (μάζα m σε δυναμικό $V = kx^2/2$) καταλήγει στις ιδιοσυναρτήσεις με χωρικό τμήμα

$$\psi_n(y) = \sqrt{\frac{1}{2^n n! \sqrt{\pi}}} \left(\frac{\sqrt{km}}{\hbar} \right)^{1/4} H_n(y) \mathrm{e}^{-y^2/2} \ , \tag{8.1}$$

όπου $y = x\sqrt{\sqrt{km}/\hbar}$ και $H_n(x)$ το πολυώνυμο Hermite βαθμού n.

Χρησιμοποιήστε τη συνάρτηση που γράψατε για τα πολυώνυμα Hermite στην άσκηση 40 για να υπολογίσετε την πυκνότητα πιθανότητας ($\psi\psi^*$) της κυματοσυνάρτησης (8.1). Θα γράψετε μια νέα συνάρτηση γι' αυτή που θα δέχεται ως ορίσματα τα n,x. Θεωρήστε ότι $m = k = \hbar = 1$.

Να τυπώσετε στο αρχείο *harmonic.dat* τις τιμές της πυκνότητας πιθανότητας για $n = 5$ σε 60 ισαπέχοντα σημεία x στο διάστημα $[-6, 6]$, μαζί με τα αντίστοιχα σημεία x (δηλαδή το αρχείο θα περιέχει δύο στήλες, x και $\psi\psi^*$).

43. Στη Μαθηματική Φυσική χρησιμοποιείται η οικογένεια πολυωνύμων Legendre, $P_\ell(x)$, με $x \in [-1, 1]$. Ο βαθμός ℓ του πολυωνύμου είναι ακέραιος, 0, 1,.... Τα δύο πρώτα πολυώνυμα Legendre είναι $P_0(x) = 1$ και $P_1(x) = x$, ενώ για μεγαλύτερες τιμές του ℓ υπολογίζονται από την αναδρομική σχέση:

$$\ell P_\ell(x) = (2\ell - 1)xP_{\ell-1}(x) - (\ell - 1)P_{\ell-2}(x) \; .$$

Γράψτε υποπρόγραμμα που να υπολογίζει την τιμή ενός πολυωνύμου Legendre. Αυτό θα δέχεται ως ορίσματα έναν ακέραιο αριθμό ℓ που θα αντιπροσωπεύει το βαθμό του πολυωνύμου και ένα πραγματικό x που θα είναι το σημείο υπολογισμού. Θα επιστρέφει την τιμή του $P_\ell(x)$.

44. Το αρχείο στη διεύθυνση https://tinyurl.com/bp7yzkdr περιέχει ακέραιους αριθμούς, σε ξεχωριστή σειρά ο καθένας. Η πρώτη γραμμή περιέχει το πλήθος των αριθμών που ακολουθούν. Βρείτε πόσοι από αυτούς δεν είναι πρώτοι αριθμοί και τυπώστε στην οθόνη το πλήθος τους. Για τον έλεγχο του αν είναι πρώτος ένας αριθμός χρησιμοποιήστε το υποπρόγραμμα που γράψατε στην άσκηση 5.

45. Γράψτε υποπρόγραμμα που να υπολογίζει όλους τους πρώτους αριθμούς μέχρι έναν ακέραιο N εφαρμόζοντας το «κόσκινο του Ερατοσθένη». Να το χρησιμοποιήσετε για να βρείτε και να τυπώσετε στην οθόνη τους πρώτους αριθμούς μέχρι το 1000.

46. Υλοποιήστε τη γεννήτρια ψευδοτυχαίων αριθμών του Cliff Pickover[8]:

$$x_{i+1} = |(100 \ln(x_i)) \bmod 1| \; ,$$

με $x_0 = 0.1$. Η έκφραση $a \bmod 1$ σημαίνει το δεκαδικό μέρος του a. Οι πραγματικοί αριθμοί x_i προκύπτουν τυχαίοι στο $[0, 1)$. Προσέξτε να γράψετε έτσι τη συνάρτηση ώστε να καλείται χωρίς ορίσματα[9] (ή με ένα όρισμα `INTENT (out)` αν επιλέξατε να την υλοποιήσετε με υπορουτίνα).

[8] http://mathworld.wolfram.com/CliffRandomNumberGenerator.html
[9] Δείτε την §8.8.

47. Γράψτε αναδρομική συνάρτηση που να υπολογίζει την ορίζουσα ενός τετραγωνικού πίνακα A διάστασης $N \times N$ εφαρμόζοντας τον ακόλουθο τύπο[10]

$$\det A = \sum_{i=1}^{N} (-1)^{i+j} a_{ij} \det \widetilde{A}_{ij} \ ,$$

για σταθερό j, π.χ. 1. Το στοιχείο του A στην i γραμμή και j στήλη συμβολίζεται με a_{ij}, ενώ $\widetilde{A}_{ij}$ είναι ο πίνακας που προκύπτει από τον A με διαγραφή της i γραμμής και της j στήλης.

48. Γράψτε πρόγραμμα που να χρησιμοποιεί τη συνάρτηση για την ορίζουσα που γράψατε στην άσκηση 47 ώστε να προσδιορίζει τη λύση γραμμικού συστήματος εφαρμόζοντας τη μέθοδο του Cramer[11].

49. Η κβαντομηχανική αντιμετώπιση του ατόμου του Υδρογόνου καταλήγει στις ιδιοσυναρτήσεις (σε σφαιρικές συντεταγμένες)

$$\psi_{n\ell m}(r, \theta, \phi) = R_{n\ell}(r) Y_{\ell m}(\theta, \phi) \ .$$

Το γωνιακό τμήμα τους είναι οι *σφαιρικές αρμονικές*,

$$Y_{\ell m}(\theta, \phi) = \sqrt{\frac{2\ell + 1}{4\pi} \frac{(\ell - m)!}{(\ell + m)!}} \, P_\ell^m(\cos\theta) \mathrm{e}^{\mathrm{i}m\phi} \ .$$

Τα *συναφή πολυώνυμα Legendre*, $P_\ell^m(x)$, ικανοποιούν τις σχέσεις (με ακέραια ℓ, m για τα οποία ισχύει $-\ell \le m \le \ell$)

- αν $\ell = m$

$$P_\ell^m(x) = (-1)^m \, 1 \times 3 \times 5 \times \cdots \times (2m - 1) \, (1 - x^2)^{m/2} \ ,$$

- αν $\ell = m + 1$

$$P_\ell^m(x) = x(2m + 1) P_m^m(x) \ ,$$

- ενώ σε άλλη περίπτωση δίνονται από την αναδρομική σχέση

$$(\ell - m) P_\ell^m(x) = x(2\ell - 1) P_{\ell-1}^m(x) - (l + m - 1) P_{\ell-2}^m(x) \ .$$

Οι γωνίες θ και ϕ μεταβάλλονται στα διαστήματα $[0, \pi]$ και $[0, 2\pi)$ αντίστοιχα.

- Γράψτε συνάρτηση που να υπολογίζει το παραγοντικό ενός μικρού ακεραίου.
- Γράψτε συνάρτηση που να υπολογίζει το συναφές πολυώνυμο Legendre, $P_\ell^m(x)$.

[10] http://mathworld.wolfram.com/DeterminantExpansionbyMinors.html
[11] http://mathworld.wolfram.com/CramersRule.html

- Γράψτε συνάρτηση που να υπολογίζει τη σφαιρική αρμονική, $Y_{\ell m}(\theta, \phi)$.
- Δημιουργήστε ένα καρτεσιανό πλέγμα 50×100 σημείων στο επίπεδο $\theta - \phi$ και υπολογίστε σε καθένα από αυτά τις τιμές των $Y_{\ell m}(\theta, \phi)$. Οι συντεταγμένες των σημείων θα ισαπέχουν στους άξονες θ και ϕ, στα διαστήματα $[0, \pi]$ και $[0, 2\pi)$ αντίστοιχα. Τυπώστε στο αρχείο με όνομα *ylm_data* τις τιμές των εκφράσεων $\sin\theta\cos\phi$, $\sin\theta\sin\phi$, $\cos\theta$, $Y_{\ell m}(\theta, \phi)Y^*_{\ell m}(\theta, \phi)$ (δηλαδή, ουσιαστικά τα x, y, z, $\psi\psi^*$) για κάθε σημείο, με $\ell = 2$, $m = 0$ (δηλαδή, ένα από τα d-τροχιακά).

50. Ένας αλγόριθμος για να βρούμε τη ρίζα μιας συνάρτησης $f(x)$, δηλαδή, την πραγματική ή μιγαδική τιμή $\bar{x}$ στην οποία η $f(x)$ μηδενίζεται ($f(\bar{x}) = 0$), είναι ο αλγόριθμος Müller. Σύμφωνα με αυτόν

(α΄) επιλέγουμε τρεις διαφορετικές τιμές x_0, x_1, x_2 στην περιοχή της αναζητούμενης ρίζας.

(β΄) Ορίζουμε τις ποσότητες

$$\begin{aligned} w_0 &= \frac{f(x_2) - f(x_0)}{x_2 - x_0} \\ w_1 &= \frac{f(x_2) - f(x_1)}{x_2 - x_1} \\ a &= \frac{w_1 - w_0}{x_1 - x_0} \,, \\ b &= w_0 + a(x_2 - x_0) \,, \\ c &= f(x_2) \,. \end{aligned}$$

(γ΄) Η επόμενη προσέγγιση της ρίζας δίνεται από τη σχέση

$$x_3 = x_2 - \frac{2c}{d} \,,$$

όπου d ο, εν γένει *μιγαδικός*, αριθμός που έχει το μεγαλύτερο μέτρο μεταξύ των $b + \sqrt{b^2 - 4ac}$, $b - \sqrt{b^2 - 4ac}$.

(δ΄) Αν η νέα προσέγγιση είναι ικανοποιητική πηγαίνουμε στο βήμα 50στ΄.

(ε΄) Θέτουμε $x_0 \leftarrow x_1$, $x_1 \leftarrow x_2$, $x_2 \leftarrow x_3$. Επαναλαμβάνουμε τη διαδικασία από το βήμα 50β΄.

(στ΄) Τέλος.

Προσέξτε ότι οι διαδοχικές προσεγγίσεις της ρίζας μπορεί να είναι *μιγαδικές* λόγω της τετραγωνικής ρίζας, οπότε οι ποσότητες x_n, q, A, B, C, D είναι γενικά *μιγαδικές*.

Βρείτε μια ρίζα της συνάρτησης $f(x) = x^3 - x + 1$ χρησιμοποιώντας τον αλγόριθμο Müller.

51. Η μαθηματική συνάρτηση $\Gamma(z)$ μπορεί να οριστεί από την έκφραση

$$\Gamma(z) = \frac{1}{z}\prod_{n=1}^{\infty}\frac{n\left(1+\frac{1}{n}\right)^z}{n+z}\ .$$

Γράψτε συνάρτηση που να υπολογίζει την $\Gamma(z)$. Χρησιμοποιήστε τη για να δείξετε ότι

$$\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{5}{2}\right) = \frac{3}{4}\pi\ .$$

Υπόδειξη I: υπολογίστε τα δύο μέλη της εξίσωσης· θα πρέπει να διαφέρουν ελάχιστα.

Υπόδειξη II: Στο γινόμενο δεν μπορούμε, φυσικά, να πάρουμε άπειρους όρους. Να σταματήσετε τον υπολογισμό του στον πρώτο όρο που διαφέρει από το 1 κατ᾽ απόλυτη τιμή λιγότερο 10^{-12}.

52. Η συνάρτηση Bessel πρώτου είδους, ακέραιας τάξης n, $J_n(x)$, μπορεί να οριστεί ως εξής

$$J_n(x) = \sum_{m=0}^{\infty}\frac{(-1)^m}{m!(m+n)!}\left(\frac{x}{2}\right)^{2m+n}\ .$$

Να τυπώσετε στο αρχείο *bessel.dat* τις τιμές των συναρτήσεων $J_0(x)$, $J_1(x)$, $J_2(x)$ σε 150 ισαπέχοντα σημεία x_i στο διάστημα $[0, 20]$. Το αρχείο θα έχει σε κάθε γραμμή τις τιμές

$$x_i \qquad J_0(x_i) \qquad J_1(x_i) \qquad J_2(x_i)$$

Υπόδειξη I: Στο άθροισμα δεν μπορούμε, φυσικά, να πάρουμε άπειρους όρους. Να σταματήσετε τον υπολογισμό του στον πρώτο όρο που κατ᾽ απόλυτη τιμή είναι μικρότερος από 10^{-12}.

Υπόδειξη II: Παρατηρήστε ότι ο κάθε όρος στο άθροισμα προκύπτει από τον προηγούμενό του με πολλαπλασιασμό κατάλληλης ποσότητας. Μπορεί να σας βοηθήσει.

53. Η κυβική ρίζα ενός πραγματικού αριθμού a μπορεί να υπολογιστεί προσεγγιστικά ως εξής: Επιλέγουμε μια οποιαδήποτε μη μηδενική τιμή, x_0. Έστω $x_0 = 1$. Εφαρμόζουμε τον τύπο

$$x_{i+1} = x_i\frac{x_i^3+2a}{2x_i^3+\ a} \qquad i = 0, 1, 2, \ldots$$

για να παραγάγουμε διαδοχικά τις τιμές $x_1, x_2, \ldots$. Δηλαδή,

$$\begin{aligned} x_1 &= x_0\frac{x_0^3+2a}{2x_0^3+\ a}\ , \\ x_2 &= x_1\frac{x_1^3+2a}{2x_1^3+\ a}\ , \quad \text{κλπ.} \end{aligned}$$

Κάθε τιμή από τις $x_1, x_2, \ldots$ προσεγγίζει όλο και καλύτερα το $\sqrt[3]{a}$. Μπορούμε να σταματήσουμε την επανάληψη σε κάποια τιμή x_k που ικανοποιεί τη σχέση $|x_k^3 - a| \leq \varepsilon$, όπου ε μια αρκετά μικρή θετική τιμή, π.χ. 10^{-12}.

Γράψτε συνάρτηση που να δέχεται ως όρισμα ένα πραγματικό αριθμό και να επιστρέφει την προσεγγιστική τιμή για την κυβική ρίζα του. Χρησιμοποιήστε τη για να υπολογίσετε τις κυβικές ρίζες των αριθμών 20.0, 20.1, 20.2,..., 30.0. Να τυπώσετε σε αρχείο με όνομα *cbrt* δύο στήλες αριθμών: η πρώτη θα αποτελείται από τους αριθμούς 20.0, 20.1, 20.2,..., 30.0 και η δεύτερη από τις κυβικές ρίζες τους. Να κρατήσετε 12 δεκαδικά ψηφία στις ρίζες.

54. Γράψτε ένα πρόγραμμα που να παίζει τρίλιζα με αντίπαλο το χρήστη. Σε αυτό το παιχνίδι, οι δύο παίκτες τοποθετούν διαδοχικά σε θέσεις πλέγματος 3×3 ή, γενικότερα, $N \times N$, το σύμβολό τους (π.χ. 'X' ή 'O') με σκοπό να επιτύχουν το σχηματισμό τριάδας (ή, γενικότερα, N-άδας) ίδιων συμβόλων σε οριζόντια, κάθετη, ή διαγώνια γραμμή. Στην περίπτωση που δε σχηματιστεί τέτοια γραμμή, υπάρχει ισοπαλία.

Φροντίστε στον κώδικά σας να υπάρχει δυνατότητα επιλογής του ποιος παίζει πρώτος. Το πρόγραμμα θα πρέπει να δίνει επαρκείς οδηγίες στο χρήστη για το πώς επιλέγει θέση πλέγματος. Προφανώς, πρέπει ο υπολογιστής να επιδιώκει τη νίκη, καταρχάς, και, όσο είναι δυνατό, να αποφεύγει την ήττα. Το πρόγραμμα να τυπώνει σε στοιχειώδη μορφή το πλέγμα μετά από κάθε κίνηση· ας εμφανίζεται κάτι σαν

```
X|O|X
-----
 |  |O
-----
O|X|
```

Φροντίστε, επιπλέον, να περιγράφετε επαρκώς με σχόλια (τι κάνουν) τις ομάδες εντολών που χρησιμοποιείτε.

55. Γράψτε ένα πρόγραμμα που να παίζει four-in-a-row με αντίπαλο εσάς. Σε αυτό το παιχνίδι, δύο παίκτες τοποθετούν διαδοχικά τις «μάρκες» τους σε ένα κατακόρυφο πλέγμα $M \times N$ (εφαρμόστε το για 7 στήλες επί 6 γραμμές). Κάθε μάρκα τοποθετείται στην κορυφή μιας στήλης και πέφτει έως ότου συναντήσει άλλη μάρκα ή το άκρο του πλέγματος. Νικητής είναι ο παίκτης που σχηματίζει τέσσερις συνεχόμενες μάρκες οριζοντίως, καθέτως ή διαγωνίως. Εάν το πλέγμα γεμίσει χωρίς να έχει σχηματιστεί τέτοια γραμμή, έχουμε ισοπαλία.

Να φροντίσετε ο υπολογιστής να μην επιλέγει τυχαία τη στήλη στην οποία θα ρίξει τη «μάρκα» του. Θα πρέπει προφανώς να την επιλέγει ώστε να προσπαθεί να σχηματίσει τετράδα. Αν δεν γίνεται αυτό, θα πρέπει να εμποδίζει τον αντίπαλο να σχηματίσει τετράδα (μόλις έρθει η σειρά του). Αλλιώς, μπορεί να επιλέγει μια τυχαία στήλη.

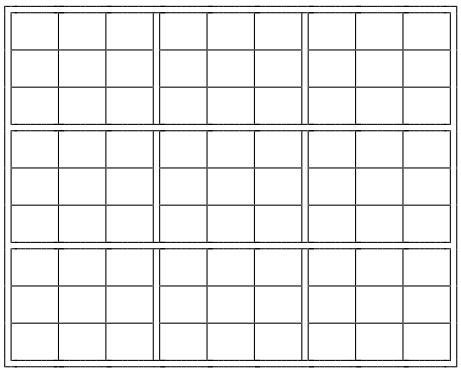
56. **Το πρόβλημα των N βασιλισσών**. Σε μια σκακιέρα $N \times N$, με $N > 3$, θέλουμε να τοποθετήσουμε N βασίλισσες σε τέτοιες θέσεις ώστε να μη βρίσκονται ανά δύο στην ίδια γραμμή, στήλη ή διαγώνιο. Γράψτε πρόγραμμα που να υπολογίζει και να τυπώνει στην οθόνη μια τέτοια τοποθέτηση.

    Κάθε γραμμή της σκακιέρας θα έχει προφανώς μια μόνο βασίλισσα. Το πρόγραμμά σας θα είναι πιο απλό αν «γεμίζετε» διαδοχικά τις γραμμές επιλέγοντας μόνο τη στήλη στην οποία θα τοποθετηθεί το κομμάτι.

    Ακολουθήστε τον εξής αλγόριθμο:

    - Δημιουργήστε ένα πίνακα ακεραίων με διαστάσεις $N \times N$. Έστω ότι ονομάζεται board. Τα στοιχεία με τιμή 0 θα αντιπροσωπεύουν επιτρεπτές θέσεις.
    - Δημιουργήστε ένα διάνυσμα ακεραίων με N στοιχεία και όνομα π.χ. column. Θα αποθηκεύει τις στήλες των βασιλισσών. Η γραμμή i θα έχει βασίλισσα στη θέση `column(i)`.
    - Γράψτε μια συνάρτηση που θα δέχεται συγκεκριμένη γραμμή και στήλη, θα εντοπίζει τις «απαγορευμένες» θέσεις (γραμμή, στήλη, διαγώνιους) και θα αυξάνει την τιμή των αντίστοιχων στοιχείων του board. Έτσι, αν κάποιο στοιχείο είναι επιτρεπτό (έχει τιμή 0) θα γίνεται μη επιτρεπτό (με τιμή 1). Αν είναι ήδη απαγορευμένο θα γίνεται πιο «έντονη» η απαγόρευση.
    - Γράψτε μια συνάρτηση που θα δέχεται συγκεκριμένη γραμμή και στήλη και θα «ακυρώνει» τη διαδικασία που έκανε η προηγούμενη. Αν κάποιο στοιχείο είναι απαγορευμένο με τιμή 1 θα γίνεται επιτρεπτό, αν είναι απαγορευμένο με μεγαλύτερη τιμή θα γίνεται λιγότερο απαγορευμένο.
      Οι δύο συναρτήσεις μπορούν εύκολα να συγχωνευθούν σε μία.
    - Ξεκινήστε από την πρώτη γραμμή. Αν υπάρχουν διαθέσιμες θέσεις σε αυτή, επιλέξτε μία, κάντε κατάλληλες τροποποιήσεις στο διάνυσμα και στον πίνακα και συνεχίστε στην επόμενη γραμμή. Αν δεν υπάρχουν διαθέσιμες θέσεις σημαίνει ότι κάποια προηγούμενη επιλογή κενής στήλης δεν οδηγεί σε λύση. Αναιρέστε την τυχόν αποθηκευμένη στήλη για την τρέχουσα γραμμή, πηγαίνετε στην προηγούμενη, ακυρώστε τις αλλαγές που έγιναν στην προηγούμενη επιλογή στήλης. Επιλέξτε άλλη στήλη. Αν εξαντληθούν οι επιτρεπτές στήλες σε μια γραμμή, μετακινηθείτε στην προηγούμενή της και ακολουθήστε την ίδια διαδικασία.
    - Όταν υπολογιστούν οι θέσεις όλων των βασιλισσών, τυπώστε στην οθόνη τη σκακιέρα (N σύμβολα σε N γραμμές, όπου υπάρχει βασίλισσα να εμφανίζεται ο χαρακτήρας 'Q' αλλιώς να εμφανίζεται ο χαρακτήρας '_').

57. **Sudoku**. Γράψτε ένα πρόγραμμα που να λύνει sudoku. Σε αυτή τη δραστηριότητα ο σκοπός είναι να γεμίσει το παρακάτω πλέγμα 9×9 με αριθμητικά ψηφία ώστε κάθε γραμμή, στήλη ή κουτί 3×3 να περιέχει όλα τα ψηφία $1-9$, από μία φορά το καθένα (χωρίς επανάληψη).



Το πρόγραμμα θα δέχεται ένα μερικώς συμπληρωμένο πλέγμα, θα προσδιορίζει τα ψηφία στα κενά τετράγωνα και θα το τυπώνει συμπληρωμένο.

Ο αλγόριθμος που μπορείτε να ακολουθήσετε είναι ο εξής:

(α΄) Ξεκινάμε από το πρώτο κενό τετράγωνο και τοποθετούμε εκεί το ψηφίο 1.

(β΄) Ελέγχουμε αν είναι αποδεκτό σύμφωνα με τους κανόνες που αναφέρθηκαν. Αν όχι, το αντικαθιστούμε με το 2, 3, κλπ. έως ότου βρούμε αποδεκτό ψηφίο. Αν εξαντλήσουμε τα ψηφία χωρίς να αποδεχθούμε κανένα, το πλέγμα δεν έχει λύση.

(γ΄) Προχωράμε στο επόμενο κενό τετράγωνο και ακολουθούμε την ίδια διαδικασία. Στην περίπτωση που εξαντλήσουμε τα ψηφία 1–9, το αφήνουμε κενό το συγκεκριμένο και μετακινούμαστε στο προηγούμενο τετράγωνο που έχουμε συμπληρώσει. Αυξάνουμε τον αριθμό του διαδοχικά, ελέγχοντας κάθε φορά τις συνθήκες. Αν αποδεχθούμε ψηφίο, προχωράμε στο επόμενο τετράγωνο, αν τα εξαντλήσουμε, μετακινούμαστε πιο πίσω κ.ο.κ.

Δοκιμάστε το για το πλέγμα

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

58. Ένας τρόπος να σχεδιάσουμε ένα διδιάστατο fractal είναι ο εξής: ξεκινάμε από ένα σημείο του επιπέδου, έστω το $(x = 0, y = 0)$, και το μετακινούμε στη

θέση (x', y') όπου

$$\begin{aligned} x' &= a \cdot x + b \cdot y + e \\ y' &= c \cdot x + d \cdot y + f \end{aligned}$$

και a, b, c, d, e, f σταθερές.

Το νέο σημείο το μεταφέρουμε με τον ίδιο μετασχηματισμό στο επόμενο σημείο του fractal (δηλαδή, θέτουμε $x' \to x$ και $y' \to y$ και παράγουμε το νέο (x', y')). Τη διαδικασία αυτή την επαναλαμβάνουμε επ' άπειρο. Η ακολουθία των σημείων (x, y) που παράγονται, αποτελεί το fractal.

Γράψτε ένα πρόγραμμα το οποίο:

(α΄) θα διαβάζει από το αρχείο *in.dat* 4 γραμμές. Σε κάθε γραμμή θα υπάρχουν 7 πραγματικοί αριθμοί: οι 6 πρώτοι αντιστοιχούν στους συντελεστές a, b, c, d, e, f και ο τελευταίος στην πιθανότητα p να γίνει ο συγκεκριμένος μετασχηματισμός. Κάθε γραμμή αντιστοιχεί σε άλλο μετασχηματισμό. Το άθροισμα των πιθανοτήτων, $\sum p_i$, όλων των μετασχηματισμών είναι 1.

(β΄) Θα επιλέγει ένα τυχαίο πραγματικό αριθμό r στο διάστημα $[0, 1)$. Ανάλογα με την τιμή του θα εφαρμόζεται διαφορετικός μετασχηματισμός. Δηλαδή, αν ισχύει $0 \leq r < p_1$ θα εκτελείται ο πρώτος μετασχηματισμός, αν ισχύει $p_1 \leq r < p_1 + p_2$ θα εκτελείται ο δεύτερος κλπ.

(γ΄) θα επαναλαμβάνει το προηγούμενο βήμα 1000 φορές σώζοντας κάθε φορά το σημείο που προκύπτει στο αρχείο *fractal.dat*.

Δοκιμάστε το πρόγραμμά σας με τις εξής παραμέτρους στο *in.dat*

$$\begin{array}{rrrrrrr} 0 & 0 & 0 & 0.16 & 0 & 0 & 0.01 \\ 0.85 & 0.04 & -0.04 & 0.85 & 0 & 1.6 & 0.85 \\ 0.2 & -0.26 & 0.23 & 0.22 & 0 & 1.6 & 0.07 \\ -0.15 & 0.28 & 0.26 & 0.24 & 0 & 0.44 & 0.07 \end{array}$$

και

$$\begin{array}{rrrrrrr} 0 & 0 & 0 & 0.25 & 0 & -0.4 & 0.02 \\ 0.95 & 0.005 & -0.005 & 0.93 & -0.002 & 0.5 & 0.84 \\ 0.035 & -0.2 & 0.16 & 0.04 & -0.09 & 0.02 & 0.07 \\ -0.04 & 0.2 & 0.16 & 0.04 & 0.083 & 0.12 & 0.07 \end{array}$$

Αν θέλετε, μπορείτε να σχεδιάσετε τα *fractal.dat* που προκύπτουν.

59. Γράψτε ένα πρόγραμμα που να παρέχει την υποδομή για να παίξουν «Ναυμαχία» δύο παίκτες. Ο ένας μπορεί να είναι ο ίδιος ο υπολογιστής. Σε αυτό το παιχνίδι κάθε παίκτης έχει ένα διδιάστατο πλέγμα 10×10 στο οποίο τοποθετεί τα πλοία του και ένα όμοιο πλέγμα για τις βολές του εναντίον του αντίπαλου παίκτη. Ο κάθε παίκτης τοποθετεί στο πλέγμα του, είτε οριζόντια είτε κάθετα, τα εξής πλοία:

(α΄) 1 Μεταγωγικό (5 θέσεις),

(β΄) 1 Θωρηκτό (4 θέσεις),

(γ΄) 1 Αντιτορπιλικό (3 θέσεις),

(δ΄) 1 Υποβρύχιο (3 θέσεις),

(ε΄) 1 Ναρκαλιευτικό (2 θέσεις).

Τα πλοία προφανώς δεν μπορούν να επικαλύπτονται στο πλέγμα και οι θέσεις τους δεν είναι γνωστές στον αντίπαλο.

Κάθε παίκτης, διαδοχικά, επιλέγει μια θέση στο πλέγμα του αντιπάλου. Αν χτυπήσει πλοίο, θα ενημερωθεί από τον αντίπαλο. Ένα πλοίο βυθίζεται όταν χτυπηθεί σε όλες τις θέσεις που καταλαμβάνει. Σκοπός κάθε παίκτη είναι να βυθίσει όλα τα πλοία του αντιπάλου. Νικητής είναι αυτός που θα το επιτύχει.

Το πρόγραμμά σας να τυπώνει στην οθόνη τα δύο πλέγματα (πλοίων και βολών) του παίκτη που είναι η σειρά του να παίξει. Οι βολές να σημειώνονται με X αν είναι επιτυχείς και με O αν δεν έχουν βρει το στόχο. Ο υπολογιστής θα ζητά από τον παίκτη να προσδιορίσει τη βολή του. Αν ο παίκτης είναι ο ίδιος ο υπολογιστής δεν θα τυπώνετε το πλέγμα του στην οθόνη αλλά θα γίνεται η επιλογή στόχου και θα έρχεται η σειρά σας.

60. Έστω η μιγαδική συνάρτηση μιγαδικής μεταβλητής $p(z)$. Μια οποιαδήποτε αρχική τιμή z_0 (για την οποία ισχύει $p'(z_0) \neq 0$) θα συγκλίνει σε μία από τις ρίζες της $p(z)$, στα σημεία δηλαδή που μηδενίζεται η $p(z)$, αν την μεταβάλλουμε ως εξής:

$$z_{i+1} = z_i - \frac{p(z_i)}{p'(z_i)}\,, \quad i = 0, 1, 2, \ldots\ .$$

Δηλαδή, αν ξεκινήσουμε από μια τιμή z_0, η εφαρμογή του τύπου θα μας δώσει τη z_1. Με νέα εφαρμογή του τύπου θα υπολογίσουμε τη z_2, κλπ., έως ότου πλησιάσουμε όσο κοντά θέλουμε σε μια από τις ρίζες της $p(z)$, όταν δηλαδή $|p(z_i)| \leq \epsilon$ με ϵ ένα πολύ μικρό θετικό αριθμό. Αυτή η επαναληπτική διαδικασία αποτελεί τον *αλγόριθμο Newton–Raphson* για εύρεση ρίζας, εφαρμοσμένο σε μιγαδικές συναρτήσεις.

Η μιγαδική συνάρτηση μιγαδικής μεταβλητής $p(z) = z^3 - 1$ έχει ρίζες τα $a = 1$, $b = \mathrm{e}^{\mathrm{i}2\pi/3}$, $c = \mathrm{e}^{-\mathrm{i}2\pi/3}$. Οποιαδήποτε αρχική τιμή στο μιγαδικό επίπεδο, εκτός από την $z = 0 + \mathrm{i}0$, θα συγκλίνει σε μια από τις ρίζες. Μπορούμε να δημιουργήσουμε μια έγχρωμη εικόνα αν σε κάθε σημείο στο μιγαδικό επίπεδο αντιστοιχήσουμε ένα χρώμα ανάλογα με το σε ποια ρίζα καταλήγει. Έτσι, π.χ., όσα σημεία καταλήγουν στην a τα χρωματίζουμε κόκκινα (RGB = $(255, 0, 0)$). Όσα καταλήγουν στην b τα χρωματίζουμε πράσινα (RGB = $(0, 255, 0)$) και όσα καταλήγουν στη c τα χρωματίζουμε μπλε (RGB = $(0, 0, 255)$). Το σημείο $0 + \mathrm{i}0$ το χρωματίζουμε λευκό (RGB = $(255, 255, 255)$).

(α΄) Επιλέξτε στον άξονα των πραγματικών $N = 512$ ισαπέχουσες τιμές στο διάστημα $[-1, 1]$ (τα άκρα περιλαμβάνονται): x_i, $i = 0, \ldots, N - 1$.

(β΄) Επιλέξτε στον άξονα των φανταστικών $M = 512$ ισαπέχουσες τιμές στο διάστημα $[-1, 1]$ (τα άκρα περιλαμβάνονται): y_j, $j = 0, \ldots, M-1$.

(γ΄) Σχηματίστε τον μιγαδικό αριθμό $z = x_i + \mathrm{i}y_j$ και βρείτε το «χρώμα» του με τη διαδικασία που περιγράφηκε παραπάνω.

(δ΄) Αποθηκεύστε τα pixels (i, j) με το αντίστοιχο χρώμα τους σε αρχείο με όνομα *newton.pppm*. Χρησιμοποιήστε τη διαμόρφωση plain ppm (δείτε την άσκηση 21 στη σελίδα 108). Η εικόνα στο *newton.pppm* είναι ένα Newton fractal.

Παρατήρηση: Να θεωρήσετε ότι δύο πραγματικοί είναι ίσοι όταν η διαφορά τους είναι κατ᾽ απόλυτη τιμή μικρότερη από 10^{-8}.

61. Να γράψετε συνάρτηση που να υπολογίζει τους αριθμούς Bernoulli, B_n, $n = 0, 1, 2, \ldots$. Ο αριθμός B_n υπολογίζεται από τον εξής αλγόριθμο

```
for m ← 0, n do
    a[m] ← 1/(m + 1)
    for j ← m, 1, −1 do
        a[j − 1] ← j(a[j − 1] − a[j])
    end for
end for
return a[0]                                   ▷ είναι το B_n
```

Επαληθεύστε τη σχέση

$$(m+1)\sum_{k=1}^{n} k^m = \sum_{k=0}^{m}\left(B_k \prod_{j=k+1}^{m+1} \frac{nj}{j-k}\right)$$

για $m = 2, 3, 4, 5, 6$ και $n = 7, 8, 9, 10$ ως εξής: υπολογίστε και τυπώστε στην οθόνη τα δύο μέλη της εξίσωσης για τις διάφορες τιμές των m, n· θα πρέπει να είναι ίσα (για τα ίδια m, n).

62. Γράψτε υποπρόγραμμα που να δέχεται τρία ορίσματα: ένα διάνυσμα πραγματικών αριθμών με οποιοδήποτε πλήθος θέσεων και δύο ακέραιους. Το υποπρόγραμμα θα βρίσκει τις θέσεις στο διάνυσμα των δύο μικρότερων τιμών του κατ᾽ απόλυτη τιμή και θα τις αποθηκεύει στα δύο ακέραια ορίσματα. Κατόπιν, γράψτε πρόγραμμα που να αποθηκεύει σε διάνυσμα τις πέντε τιμές $\{2.1, 3.2, -5.1, -6.3, -1.4\}$ και χρησιμοποιήστε το υποπρόγραμμα που γράψατε για να βρείτε πού είναι οι δύο μικρότερες τιμές του, κατ᾽ απόλυτη τιμή.

Υπόδειξη: Ένας από τους πολλούς τρόπους για να βρείτε τις δύο μικρότερες τιμές είναι ο εξής: βρείτε αρχικά τη θέση του μικρότερου στοιχείου στο διάνυσμα. Κατόπιν, ψάξτε ξανά για το μικρότερο στοιχείο, παραλείποντας αυτή τη φορά τη θέση που βρήκατε στο προηγούμενο στάδιο.

63. Το πολυώνυμο Chebyshev πρώτου είδους, (ακέραιας) τάξης n, μπορεί να οριστεί ως εξής:

$$T_n(x) = 2^{n-1} \prod_{k=1}^{n} \left\{ x - \cos\left[\frac{(2k-1)\pi}{2n} \right] \right\} ,$$

για $|x| \leq 1$.

Γράψτε συνάρτηση που να υπολογίζει την τιμή του πολυωνύμου $T_n(x)$. Θα δέχεται ως ορίσματα τα n, x.

Κατόπιν, γράψτε πρόγραμμα που να υπολογίζει τις τιμές του πολυωνύμου $T_4(x)$ στα σημεία $x = -1.0, -0.9, -0.8, \ldots, 0.8, 0.9, 1.0$. Να τυπώσετε στο αρχείο *cheb.dat* τις παραπάνω τιμές των x μαζί με τις αντίστοιχες τιμές του πολυωνύμου, κρατώντας 4 δεκαδικά ψηφία. Θα δημιουργήσετε δύο στήλες στο αρχείο· σε κάθε γραμμή του θα υπάρχει ένα ζεύγος τιμών x, $T_4(x)$, με κενό ανάμεσά τους.

64. Γράψτε συνάρτηση που να δέχεται ως ορίσματα τρεις ακέραιους αριθμούς που θα αντιπροσωπεύουν ημερομηνία: ημέρα, μήνας, έτος. Η συνάρτηση να ελέγχει αν η δεδομένη ημερομηνία είναι έγκυρη (δηλαδή υπαρκτή) ή όχι. Να επιστρέφει αυτή την πληροφορία. Χρησιμοποιήστε τη για να ελέγξετε αν είναι έγκυρες οι ημερομηνίες 5/12/2016, 31/11/2010, 29/2/2016, 29/2/1900. Τυπώστε τη σχετική πληροφορία στην οθόνη.

Θα σας χρειαστούν οι πληροφορίες της άσκησης 9 στη σελίδα 39.

65. Γράψτε υποπρόγραμμα που να δέχεται ως ορίσματα τρεις ακέραιους αριθμούς (d, m, y) που αναπαριστούν μια ημερομηνία: ημέρα, μήνας, έτος. Το υποπρόγραμμα θα τροποποιεί τα ορίσματά του ώστε να αντιπροσωπεύουν την ημερομηνία της **επόμενης** ημέρας. Έτσι π.χ.

- τα $(d, m, y) = (3, 12, 2003)$ το υποπρόγραμμα θα τα αλλάξει σε $(d, m, y) = (4, 12, 2003)$,
- τα $(d, m, y) = (30, 11, 2003)$ θα τα κάνει $(d, m, y) = (1, 12, 2003)$,
- τα $(d, m, y) = (31, 12, 2003)$ θα τα κάνει $(d, m, y) = (1, 1, 2004)$.

Χρησιμοποιήστε το υποπρόγραμμα σε πρόγραμμα που θα διαβάζει μια ημερομηνία από το χρήστη και θα τυπώνει στην οθόνη την ημερομηνία της επόμενης ημέρας.

Θα σας χρειαστούν οι πληροφορίες της άσκησης 9 στη σελίδα 39.

66. Γράψτε υποπρόγραμμα που να δέχεται ένα διάνυσμα 4 ακεραίων ως όρισμα. Θα αποθηκεύει στο διάνυσμα αυτό τυχαίους ακέραιους στο διάστημα $[1, 8]$. Ο κώδικάς σας να εξασφαλίζει ότι οι τυχαίοι αριθμοί είναι μεταξύ τους *διαφορετικοί*.

Χρησιμοποιήστε το υποπρόγραμμα σε πρόγραμμα που θα τυπώνει στην οθόνη 10 τετράδες διαφορετικών τυχαίων ακεραίων στο $[1, 8]$.

67. Γράψτε υποπρόγραμμα που να δέχεται τρία διανύσματα 4 ακεραίων, έστω a, b, c, ως ορίσματα.

    Ο κώδικας θα αποθηκεύει στις θέσεις του c τις τιμές 0, 1, 2 ως εξής:

    - Αν τα a,b στην ίδια θέση έχουν τον ίδιο αριθμό, τότε στην αντίστοιχη θέση του c θα αποθηκεύεται το 2.
    - Αν στη θέση i του b υπάρχει αριθμός που εμφανίζεται στο a αλλά όχι στη θέση i, τότε στη θέση i του c θα αποθηκεύεται το 1.
    - Σε άλλη περίπτωση, δηλαδή ο αριθμός στη θέση i του b δεν υπάρχει στο a, τότε στη θέση i του c θα αποθηκεύεται το 0.

    Γράψτε πρόγραμμα που να χρησιμοποιεί το υποπρόγραμμα.

Παράρτημα Α΄

Αναζήτηση–Ταξινόμηση

Α΄.1 Αναζήτηση στοιχείου

Α΄.1.1 Γραμμική αναζήτηση

Η αναζήτηση συγκεκριμένης τιμής σε ένα διάνυσμα χωρίς συγκεκριμένη οργάνωση (ταξινόμηση, κατηγοριοποίηση, κλπ.) των στοιχείων του, απαιτεί να γίνεται σύγκρισή της με κάθε στοιχείο του διανύσματος. Κατά μέσο όρο χρειάζονται $(1 + N)/2$ συγκρίσεις, όπου N το πλήθος των στοιχείων του διανύσματος.

Α΄.1.2 Δυαδική αναζήτηση

Για ένα *ταξινομημένο* διάνυσμα με N στοιχεία, το πλήθος των απαιτούμενων συγκρίσεων στην αναζήτηση μπορεί να γίνει $1 + \log_2 N$, πολύ μικρότερο (για μεγάλα N) από όσο είναι στο γραμμικό αλγόριθμο, αν εφαρμόσουμε τη δυαδική αναζήτηση. Βέβαια, στις απαιτούμενες πράξεις δεν υπολογίζουμε αυτές που χρειάζονται για να ταξινομηθεί το διάνυσμα.

Ο συγκεκριμένος αλγόριθμος σε ένα ταξινομημένο, με αύξουσα σειρά, διάνυσμα έχει ως εξής:

Συγκρίνουμε το μεσαίο στοιχείο του διανύσματος (ή ένα από τα δύο πλησιέστερα στη μέση αν το διάνυσμα έχει άρτιο πλήθος στοιχείων) με τη ζητούμενη τιμή:

- Αν η ζητούμενη τιμή είναι μικρότερη, σημαίνει ότι, αν υπάρχει, βρίσκεται στο πρώτο μισό του διανύσματος.
- Αν η ζητούμενη τιμή δεν είναι μικρότερη, σημαίνει ότι, αν υπάρχει, βρίσκεται στο δεύτερο μισό του διανύσματος.

Επομένως, με την πρώτη σύγκριση περιορίζουμε στο μισό τον χώρο αναζήτησης. Επαναλαμβάνουμε τη διαδικασία για το τμήμα του διανύσματος που επιλέξαμε στο προηγούμενο βήμα έως ότου, με διαδοχικές διχοτομήσεις, περιοριστούμε σε ένα στοιχείο. Τότε, αν είναι ίσο με τη ζητούμενη τιμή επιστρέφουμε τη θέση του, αλλιώς η ζητούμενη τιμή δεν περιέχεται στο διάνυσμα.

Μπορούμε να υλοποιήσουμε τη συγκεκριμένη μέθοδο αναζήτησης και με συνάρτηση που καλεί τον εαυτό της:

1. αν το πλήθος των στοιχείων είναι 0, το ζητούμενο στοιχείο δεν υπάρχει.
2. αν το πλήθος των στοιχείων είναι 1, συγκρίνουμε το μοναδικό στοιχείο με το ζητούμενο. Αν είναι ίσα, επιστρέφουμε την *απόλυτη* θέση του (δηλαδή, από την αρχή του διανύσματος), αλλιώς το ζητούμενο στοιχείο δεν υπάρχει.
3. αν το πλήθος των στοιχείων είναι μεγαλύτερο από 1, συγκρίνουμε το μεσαίο στοιχείο (ή ένα από τα δύο στοιχεία που είναι πλησιέστερα στο μέσο του διανύσματος) με το ζητούμενο. Αν το ζητούμενο στοιχείο είναι μικρότερο, το αναζητούμε στο ίδιο διάνυσμα, στο «πρώτο μισό». Αλλιώς, το αναζητούμε στο «δεύτερο μισό».

Α΄.1.3 Αναζήτηση με hash

Τα στοιχεία στα οποία επιθυμούμε να αναζητήσουμε μια συγκεκριμένη τιμή, μπορεί να είναι οργανωμένα σε *πίνακα κατακερματισμού* (hash table), δηλαδή σε πολλές ομάδες λίγων στοιχείων με ίδια χαρακτηριστική τιμή. Η αναζήτηση σε τέτοιο πίνακα γίνεται σε δύο στάδια: Πρώτα υπολογίζουμε τη χαρακτηριστική τιμή της αναζητούμενης ποσότητας και κατόπιν συγκρίνουμε την ποσότητα μόνο με τα στοιχεία που έχουν την ίδια χαρακτηριστική τιμή. Η αναζήτηση με αυτό τον τρόπο χρειάζεται πράξεις ανεξάρτητες από το πλήθος των αποθηκευμένων τιμών, και υπό κατάλληλες συνθήκες, είναι πολύ πιο γρήγορη από την δυαδική αναζήτηση.

Για την κατάλληλη οργάνωση των στοιχείων ώστε να είναι εφικτή τέτοια αναζήτηση, είναι βασική η *συνάρτηση κατακερματισμού*, (hash function). Τέτοια συνάρτηση αντιστοιχεί καθένα από τα στοιχεία σε μία χαρακτηριστική τιμή (hash). Το hash μπορεί να πάρει τιμή σε ένα πεπερασμένο σύνολο. Έτσι, π.χ.

- Οι λέξεις ενός λεξικού ή τα λήμματα μιας εγκυκλοπαίδειας μπορούν να οργανωθούν σε διαφορετικούς τόμους (ομάδες), ανάλογα με το αρχικό γράμμα τους.
- ένας οποιοσδήποτε απρόσημος ακέραιος μήκους $16\,\mathrm{bit}$, δηλαδή στο διάστημα $[0, 65535]$, μπορεί να αντιστοιχηθεί στους ακέραιους στο διάστημα $[0, 255]$, υπολογίζοντας το υπόλοιπο της διαίρεσής του με το 256. Η συνάρτηση hash σε αυτή την περίπτωση είναι η $h(k) = k \bmod 256$.
- Ένα κείμενο, δηλαδή μια σειρά χαρακτήρων οποιουδήποτε μήκους, μπορεί να αντιστοιχηθεί σε ακέραιο στο $[0, 255]$ αν κάθε χαρακτήρας του συνδυαστεί σε ένα byte.

Σε μία συνάρτηση hash, γενικά, το πεδίο ορισμού της έχει πολύ περισσότερες τιμές από τις δυνατές τιμές του hash, δηλαδή του αποτελέσματος. Αυτό έχει ως συνέπεια να υπάρχει πιθανότητα περισσότερες από μία τιμές να έχουν το ίδιο hash και επομένως να κατανέμονται στην ίδια ομάδα στοιχείων· σε τέτοια περίπτωση λέμε ότι έχουμε *σύγκρουση* (collision). Μια καλή συνάρτηση hash (και συνεπώς, καλή επιλογή του πλήθους των ομάδων) μπορεί να οργανώσει ένα πλήθος στοιχείων σε ομάδες με ελάχιστες συγκρούσεις, οπότε η αναζήτηση είναι ταχύτατη. Προφανώς, δεν λαμβάνουμε υπόψη το χρόνο οργάνωσης των στοιχείων σε πίνακα κατακερματισμού και θεωρούμε ότι ο υπολογισμός του hash είναι γρήγορος.

Α΄.2 Ταξινόμηση στοιχείων

Α΄.2.1 Bubble sort

Ένας αλγόριθμος ταξινόμησης είναι ο bubble sort (αλγόριθμος φυσαλλίδας). Είναι πολύ απλός αλλά πολύ αργός καθώς το πλήθος των απαιτούμενων πράξεων για να γίνει η ταξινόμηση με αυτόν είναι ανάλογο του τετραγώνου του πλήθους των στοιχείων, N, προς ταξινόμηση. Γι᾽ αυτό χαρακτηρίζεται ως τάξης $O(N^2)$.

Έστω ότι επιθυμούμε να ταξινομήσουμε N αντικείμενα από το μικρότερο στο μεγαλύτερο. Ο αλγόριθμος έχει ως εξής:

1. Συγκρίνουμε το πρώτο με το δεύτερο στοιχείο. Αν χρειάζεται, τα εναλλάσσουμε ώστε το μικρότερο να είναι πρώτο. Επαναλαμβάνουμε διαδοχικά τη σύγκριση και πιθανή εναλλαγή για τα ζεύγη στοιχείων $(2, 3)$, $(3, 4)$, ..., $(N-1, N)$. Στο τέλος της συγκεκριμένης διαδικασίας, το μεγαλύτερο στοιχείο από όλα θα βρεθεί στη θέση N, δηλαδή στη σωστή του θέση.

2. Επαναλαμβάνουμε το προηγούμενο βήμα, διαδοχικά: μία φορά μέχρι το ζεύγος $(N-2, N-1)$, την επόμενη μέχρι το $(N-2, N-1)$, ..., την τελευταία φορά μέχρι το ζεύγος $(1, 2)$. Σε κάθε βήμα, το στοιχείο με το κατάλληλο μέγεθος έρχεται στις θέσεις $N-1$, $N-2$, ..., 1.

Με το τέλος του αλγορίθμου η λίστα των στοιχείων είναι ταξινομημένη από το μικρότερο προς το μεγαλύτερο.

Όπως θα καταλάβετε από τις συγκρίσεις, είναι εξαιρετικά αργός αλγόριθμος και δε θα πρέπει να τον χρησιμοποιείτε για οτιδήποτε σοβαρό!

Α΄.2.2 Insertion sort

Είναι παρόμοιος αλγόριθμος με τον αλγόριθμο bubble sort. Είναι τάξης $O(N^2)$ έως (στην καλύτερη περίπτωση της ήδη ταξινομημένης λίστας) $O(N)$. Έχει ως εξής:

1. Ξεκινώντας από το δεύτερο στοιχείο της λίστας, το «ταξινομούμε» σε σχέση με το πρώτο.

2. Επιλέγουμε διαδοχικά το τρίτο, τέταρτο,... στοιχείο και το τοποθετούμε στη σωστή σειρά σε σχέση με τα προηγούμενα (που έχουν ήδη ταξινομηθεί), κάνοντας και όποιες μετακινήσεις στοιχείων είναι απαραίτητες.

Α΄.2.3 Quick sort

Ο αλγόριθμος quick sort είναι (υπό προϋποθέσεις) από τους πιο γρήγορους αλγόριθμους ταξινόμησης. Είναι τάξης $O(N^2)$ (στη χειρότερη περίπτωση της ήδη ταξινομημένης λίστας) έως (συνήθως) $O(N \log N)$, όπου N το πλήθος των στοιχείων.

Έστω ότι έχουμε λίστα στοιχείων (σε διάνυσμα) που επιθυμούμε να τα ταξινομήσουμε από το μικρότερο στο μεγαλύτερο. Ο αλγόριθμος quick sort υλοποιείται πολύ εύκολα με αναδρομικό υποπρόγραμμα.

Ο αλγόριθμος είναι ο ακόλουθος:

1. Αν η λίστα στοιχείων δεν έχει κανένα στοιχείο ή έχει μόνο ένα, επιστρέφουμε καθώς δεν χρειάζεται ταξινόμηση.

2. Επιλέγουμε ένα οποιοδήποτε στοιχείο της αρχικής λίστας.

3. Επιδιώκουμε να μεταφέρουμε στην αρχή της λίστας τα στοιχεία που είναι μικρότερα ή ίσα με το επιλεγμένο και στο τέλος της λίστας όσα είναι μεγαλύτερα από το επιλεγμένο. Στη μοναδική θέση που απομένει, μεταφέρουμε το επιλεγμένο στοιχείο.

4. Εφαρμόζουμε την ίδια διαδικασία στις υπο-λίστες πριν και μετά το επιλεγμένο στοιχείο (στη νέα του θέση), χωρίς να το περιλαμβάνουμε.

Ο διαχωρισμός των στοιχείων της λίστας μπορεί να γίνει ως εξής:

- διατρέχουμε τη λίστα με δύο δείκτες· ο ένας ξεκινά από την αρχή και ο άλλος από το τέλος. Ο πρώτος θα αυξάνει όσο δείχνει σε στοιχεία μικρότερα ή ίσα με το επιλεγμένο ενώ ο δεύτερος θα μειώνεται όσο δείχνει σε στοιχεία μεγαλύτερα από το επιλεγμένο. Στις μετακινήσεις των δεικτών παραλείπουμε τη θέση του επιλεγμένου στοιχείου και προσέχουμε να μην φύγουν έξω από τα όρια της λίστας.

- Όσο ο πρώτος δείκτης δεν έχει ξεπεράσει το δεύτερο, εναλλάσσουμε τα στοιχεία στα οποία δείχνουν οι δύο δείκτες.

- Όταν ο πρώτος δείκτης ξεπεράσει το δεύτερο, εναλλάσσουμε το επιλεγμένο στοιχείο με αυτό που δείχνει ο δεύτερος δείκτης.

Α΄.2.4 Merge sort

Είναι από τους πιο γρήγορους αλγόριθμους ταξινόμησης, με πλήθος πράξεων ανάλογο του $N \log N$, όπου N το πλήθος των στοιχείων. Σύμφωνα με τον αλγόριθμο merge sort:

1. Αν η λίστα στοιχείων δεν έχει κανένα ή έχει μόνο ένα στοιχείο, επιστρέφουμε καθώς δεν χρειάζεται ταξινόμηση.
2. Χωρίζουμε τη λίστα σε δύο περίπου ίσα μέρη.
3. Ταξινομούμε κάθε νέο τμήμα με ξεχωριστή εφαρμογή της τρέχουσας διαδικασίας (επομένως καλούμε το υποπρόγραμμα που γράφουμε και που υλοποιεί τη merge sort).
4. Συγχωνεύουμε τις δύο ταξινομημένες λίστες με τέτοιο τρόπο ώστε η τελική να είναι επίσης ταξινομημένη.

Α΄.3 Ασκήσεις

1. Να γράψετε δύο υποπρογράμματα που να δέχονται από δύο ορίσματα: το πρώτο θα είναι ένα διάνυσμα ακεραίων και το δεύτερο ένας ακέραιος (η ζητούμενη τιμή). Να επιστρέφουν σε όρισμα τη θέση του διανύσματος στην οποία βρίσκεται η ζητούμενη τιμή ή -1 αν αυτή δεν βρέθηκε. Το ένα υποπρόγραμμα θα εφαρμόζει τη γραμμική αναζήτηση και το άλλο τη δυαδική (το μη αναδρομικό αλγόριθμο).

2. Γράψτε συναρτήση που να υλοποιεί τον αναδρομικό αλγόριθμο (η συνάρτηση καλεί τον εαυτό της) για τη δυαδική αναζήτηση. Θα δέχεται ως ορίσματα ένα ταξινομημένο διάνυσμα ακεραίων, δύο ακέραιους που θα προσδιορίζουν το πρώτο και το τελευταίο στοιχείο του διανύσματος που θα ληφθούν υπόψη στην αναζήτηση, καθώς και τη ζητούμενη τιμή. Η συνάρτηση θα επιστρέφει τη θέση που βρήκε τη ζητούμενη τιμή ή -1 αν δεν τη βρει.

3. Γράψτε κατάλληλο υποπρόγραμμα που να εφαρμόζει τον αλγόριθμο φυσαλλίδας. Αυτό θα δέχεται ως όρισμα ένα διάνυσμα ακεραίων και θα το τροποποιεί ώστε να είναι ταξινομημένο από το μικρότερο στο μεγαλύτερο στοιχείο.

   Χρησιμοποιήστε το υποπρόγραμμα για να ταξινομήσετε τους 1254 ακέραιους αριθμούς που δίνονται σε ξεχωριστή γραμμή ο καθένας στο αρχείο στη διεύθυνση `https://tinyurl.com/2f7dckfd`. Το πρόγραμμά σας να τυπώνει τους αριθμούς αυτούς, ταξινομημένους, στο αρχείο *sorted_data*.

   Υπόδειξη: Να γράψετε και να χρησιμοποιήσετε υποπρόγραμμα για την εναλλαγή τιμών δύο μεταβλητών.

4. Υλοποιήστε σε αναδρομικό υποπρόγραμμα τον αλγόριθμο ταξινόμησης quick sort. Το υποπρόγραμμα θα δέχεται ως όρισμα ένα διάνυσμα πραγματικών αριθμών διπλής ακρίβειας, το οποίο θα ταξινομεί από το μικρότερο στο μεγαλύτερο στοιχείο. Χρησιμοποιήστε το υποπρόγραμμά σας για να ταξινομήσετε τα στοιχεία του αρχείου στη διεύθυνση `https://tinyurl.com/bp7yzkdr`. Η πρώτη γραμμή του αρχείου περιέχει το πλήθος των αριθμών που ακολουθούν.

5. Γράψτε υποπρόγραμμα που θα δέχεται τρία διανύσματα πραγματικών αριθμών, έστω a,b,c. Τα διανύσματα a,b θα έχουν οποιοδήποτε πλήθος στοιχείων ενώ το c θα θεωρούμε ότι έχει όσες θέσεις στοιχείων έχουν τα a,b μαζί. Τα διανύσματα a,b θα θεωρούμε ότι έχουν ταξινομημένους από το μικρότερο στο μεγαλύτερο τους αριθμούς που αποθηκεύουν. Το υποπρόγραμμά σας θα αντιγράφει τα στοιχεία των a,b στο διάνυσμα c με τέτοια σειρά ώστε να αποθηκεύονται στο c ταξινομημένα από το μικρότερο στο μεγαλύτερο.

   Χρησιμοποιήστε το υποπρόγραμμα αυτό για να συγχωνεύσετε σε ένα διάνυσμα τα σύνολα στοιχείων $1.2, 3.4, 5.6, 8.8, 9.2, 14.9$ και $2.1, 4.3, 6.5, 7.7$. Τυπώστε στην οθόνη τα στοιχεία του διανύσματος που προκύπτει. Ο κάθε αριθμός που θα τυπώνετε να έχει 1 δεκαδικό ψηφίο.

Υπόδειξη: Προσέξτε ότι τα a,b μπορεί να έχουν διαφορετικό πλήθος θέσεων. Κατά την αντιγραφή των στοιχείων κάποιο μπορεί να εξαντληθεί πρώτο.

6. Χρησιμοποιήστε το υποπρόγραμμα που γράψατε στην άσκηση 5 για να υλοποιήσετε τον αλγόριθμο merge sort.

7. Στο αρχείο στη διεύθυνση https://tinyurl.com/2f7dckfd περιέχονται 1254 ακέραιοι αριθμοί σε ξεχωριστή γραμμή ο καθένας. Να γράψετε πρόγραμμα που να τους διαβάζει σε διάνυσμα και να χρησιμοποιεί τη γραμμική αναζήτηση για να εντοπίσει τους αριθμούς 316001 και 499160. Κατόπιν, να ταξινομεί το διάνυσμα είτε με quick sort (άσκηση 4) είτε με merge sort (άσκηση 6) και να επαναλαμβάνει την αναζήτηση των αριθμών με το δυαδικό αλγόριθμο.

Παράρτημα Β΄

Σύνοψη

Σε ένα πρόγραμμα Fortran 95:

Β΄.1 Σειρά εκτέλεσης

Οι εντολές εκτελούνται με τη σειρά που εμφανίζονται στο αρχείο, από επάνω προς τα κάτω.

Β΄.2 Δηλώσεις

- Στην αρχή του κυρίως προγράμματος ή του κάθε υποπρογράμματος συγκεντρώνονται οι δηλώσεις των μεταβλητών και σταθερών ποσοτήτων (§2.4), μεταβλητών και σταθερών πινάκων (§5), υποπρογραμμάτων (§8.3.1), που θα χρησιμοποιηθούν. Πρώτη εντολή στο τμήμα των δηλώσεων πρέπει να είναι η `IMPLICIT NONE`. Κατόπιν, ακολουθούν οι εκτελέσιμες εντολές. *Δεν επιτρέπεται να αναμίξουμε δηλώσεις και εντολές.*

- Οποιαδήποτε ποσότητα σταθερή ή μεταβλητή, απλή ή πίνακας, προτού χρησιμοποιηθεί πρέπει *να έχει δηλωθεί και να έχει αποκτήσει τιμή.* Κάθε συνάρτηση ή υπορουτίνα προτού χρησιμοποιηθεί πρέπει να έχει δηλωθεί· να έχουμε περιλάβει, δηλαδή, το `INTERFACE` της, §8.3.1, στις δηλώσεις του τμήματος (υποπρόγραμμα ή κυρίως πρόγραμμα) στο οποίο χρησιμοποιείται.

- Οι δηλώσεις τοπικών ποσοτήτων που εμφανίζονται σε ένα τμήμα του κώδικά μας (υποπρόγραμμά ή κυρίως πρόγραμμα), *είναι τελείως ανεξάρτητες* από τις δηλώσεις σε άλλο τμήμα. Οι τοπικές ποσότητες επιτρέπεται να χρησιμοποιηθούν *μόνο* στο τμήμα κώδικα στο οποίο δηλώθηκαν.

Β΄.3 Πίνακες

Όταν επιθυμούμε να χειριστούμε πολλές, σχετιζόμενες ποσότητες καλό είναι να χρησιμοποιούμε πίνακα (§5) κατάλληλου πλήθους στοιχείων, μονοδιάστατου (διάνυσμα) ή—αν ταιριάζει στο πρόβλημά μας—πολυδιάστατου.

- Όταν γνωρίζουμε *την ώρα που γράφουμε τον κώδικά μας*, το πλήθος των στοιχείων ενός διανύσματος (ή πλήθος γραμμών, στηλών, κλπ. πίνακα), το περιλαμβάνουμε στη δήλωσή του.
- Όταν *δεν έχουμε αυτήν την πληροφορία*, αλλά θα τη μάθουμε με ανάγνωση από αρχείο (ή από το πληκτρολόγιο), ή θα την υπολογίσουμε κατά την εκτέλεση του προγράμματος, δηλώνουμε το διάνυσμα ή πίνακα ως `ALLOCATABLE`, §5.4, προσδιορίζοντας μόνο το πλήθος των διαστάσεών του.

  Αφού διαβάσουμε ή υπολογίζουμε το πλήθος των στοιχείων καλούμε την εντολή `ALLOCATE`. *Μετά την τελευταία (χρονικά) χρήση του πίνακα* καλό είναι να καλέσουμε την εντολή `DEALLOCATE`.

Β΄.4 Εκχώρηση τιμής

Μια απλή μεταβλητή, ή μεταβλητός πίνακας μπορεί να πάρει τιμή με δύο τρόπους:

Εντολή εκχώρησης, §2.2 Είναι της μορφής

> μεταβλητή = [έκφραση με σταθερές και μεταβλητές, κλήση συνάρτησης, ...]

Στο αριστερό της μέλος επιτρέπεται να εμφανίζεται *μόνο* απλή μεταβλητή, στοιχείο μεταβλητού διανύσματος ή πίνακα ή μεταβλητός πίνακας ως σύνολο. Στην τελευταία περίπτωση, η εντολή εκτελείται για κάθε στοιχείο του πίνακα.

Εντολή ανάγνωσης από αρχείο (ή το πληκτρολόγιο) Η σχετική εντολή είναι η `READ`, §7. Προσέξτε ότι, αν η ανάγνωση γίνει από αρχείο, αυτό πρέπει να έχει συνδεθεί πιο πριν με το πρόγραμμά μας (`OPEN`, §7.3).

Προφανώς, δεν έχει νόημα να χρησιμοποιούνται και οι δύο τρόποι ταυτόχρονα για εκχώρηση τιμής στην ίδια ποσότητα.

Β΄.5 Εκτέλεση υπό συνθήκη

Αν επιθυμούμε να εκτελέσουμε κάποιες εντολές όταν ικανοποιείται μια συνθήκη (απλή ή σύνθετη, §3.1) χρησιμοποιούμε την εντολή `IF`, §3.2.1. Σε ειδικές περιπτώσεις, η εντολή `SELECT CASE` (§3.2.2), ίσως είναι πιο κατάλληλη.

Β΄.6 Εντολή επανάληψης

Όταν θέλουμε να εκτελέσουμε κάποιες εντολές πολλές φορές χρησιμοποιούμε την εντολή `DO`, §4.1.

Στην περίπτωση που *γνωρίζουμε το πλήθος των επαναλήψεων* χρησιμοποιούμε το `DO` με μεταβλητή ελέγχου, §4.2. *Η μεταβλητή ελέγχου είναι υποχρεωτικά ακέραιου τύπου.*

Στην περίπτωση που *δεν γνωρίζουμε το ακριβές πλήθος των επαναλήψεων*, αλλά θέλουμε να εκτελέσουμε τις εντολές *έως ότου ικανοποιηθεί κάποια συνθήκη*, χρησιμοποιούμε το `DO` *χωρίς μεταβλητή ελέγχου*, §4.3. Μεταξύ των `DO` και `END DO` πρέπει *υποχρεωτικά* να εμφανίζεται η εντολή `EXIT`, και μάλιστα, μέσα σε εντολή `IF` που ελέγχει την επιθυμητή συνθήκη.

Β΄.7 Υποπρόγραμμα

Εντολές που σχετίζονται μεταξύ τους και πρόκειται να χρησιμοποιηθούν πολλές φορές σε ένα ή περισσότερα προγράμματά μας, καλό είναι να απομονώνονται σε υποπρόγραμμα, §8.1. Ένα υποπρόγραμμα, συνάρτηση ή υπορουτίνα, παραμετροποιείται από κάποιες ποσότητες, τα ορίσματά του. *Η «επικοινωνία» του με άλλα τμήματα κώδικα γίνεται μέσω των ορισμάτων του (ή μέσω MODULE).* Αν ο κώδικας που θα απομονώσουμε, θέλουμε να επιστρέφει *μία* τιμή, επιλέγουμε να τον γράψουμε ως `FUNCTION`. Αν δεν επιστρέφει τιμή ή θέλουμε να υπολογίσει και να επιστρέψει περισσότερες από μία τιμές, επιλέγουμε να τον υλοποιήσουμε σε `SUBROUTINE`.

Κατάλογος πινάκων

Ευρετήριο