"Μια Συντομη Εισαγωγη Στο Matlab Για Μαθηματικους"

Χρυσοβαλάντης Α. Σφυράκης

hammer@math.uoa.gr

http://math.uoa.gr/∼hammer

Αθήνα 2004

Περιεχόμενα

Κεφαλαιο 1

Προλογος

Οι σημειώσεις αυτές γράφτηκαν για τις ανάγκες των εργαστηρίων που γίνονται στα πλαίσια των μαθημάτων του Μαθηματικού Τμήματος του Πανεπιστημίου Αθηνών. Το MATLAB είναι στενά συνδεδεμένη με εφαρμογές στις Θετικές Επιστήμες γενικότερα, και αντικαθιστά τη συντριπτική πλειοψηφία του λογισμικού που χρησιμοποιείται μέχρι και σήμερα.

Η σύντομη αυτή εισαγωγή σε καμιά περίπτωση δεν μπορεί να θεωρηθεί πλήρης. Περιγράφονται μόνο κάποια από τα βασικά στοιχεία του προγράμματος, που όμως είναι αρκετά για την υλοποίηση των προγραμματιστικών αναγκών των μαθημάτων στα οποία απευθύνεται. Οι σημειώσεις αυτές απευθύνονται σε αναγνώστες που έχουν κάποια προγραμματιστική εμπειρία, όπως αυτή που παρέχεται από το μάθημα Πληροφορική Ι του Τμήματος, και παραπέμπουμε, όπου είναι αυτό δυνατόν, στις αντίστοιχες δομές της γλώσσας C.

Για όσους ενδιαφέρονται υπάρχει αρκετό υλικό (βιβλία, σημειώσεις κ.λ.π.) σχετικά με το MATLAB στο διαδίκτυο. Ενδεικτικά αναφέρουμε τα παρακάτω

1.1 Τι είναι το Matlab

Το MATLAB (MATrix LABoratory) δημιουργήθηκε τη δεκαετία του 1990, σχεδιάστηκε εξαρχής για μαθηματικούς σκοπούς, για να κάνει δυνατή την υπολογιστική επίλυση μαθηματικών προβλημάτων. Έτσι, εφόσον ένας αλγόριθμος διατυπωθεί σαφώς (π.χ. με διάγραμμα ροής), η υλοποίησή του σε MATLAB είναι απλή και άμεση. Επιπλέον, προγράμματα γραμμένα σε MATLAB είναι γενικά σαφή και εύκολα αντιληπτά (εξαρτάται και από τον προγραμματιστή!). Η γλώσσα είναι αυστηρά δομημένη, όσον αφορά τη σύνταξη των εντολών της, και έτσι πολλά από τα συντακτικά λάθη προγραμματισμού μπορούν να ανακαλυφθούν κατά την εκτέλεση του προγράμματος παρέχοντας σημαντική ασφάλεια στον προγραμματιστή.

Το Matlab είναι ένα μαθηματικό πρόγραμμα που κρύβει μέσα του μια γλώσσα προγραμματισμού που μπορεί να χρησιμοποιηθεί αποτελεσματικά για την υλοποίηση σύνθετων προβλημάτων. Παράλληλα η δυνατότητα να μπορεί να κάνει και συμβολικές πράξεις αλλά και ο μεγάλος αριθμός έτοιμων βιβλιοθηκών το κατατάσσει στα καλύτερα στην κατηγορία του.

Δημιουργήθηκε απο τον C. Moler, *αρχικά* σαν εργαλείο διαχείρισης των βιβλιοθηκών Fortran: LINPACK (γρ. άλγεβρα) και EISPACK (ιδιοτιμές και ιδιοδιανύσματα). *Εξελίκτικε* σε σύνθετο πακέτο (γραμμένο σε C, C++) που αναπτύσσεται συνεχώς. Η έκδοση που χρησιμοποιείται σήμερα είναι η 7 και πέρα από τις δυνατότητες που παρείχαν οι προηγούμενες αυτή έχει εμπλουτιστεί με νέες βιβλιοθήκες αλλά και με καινούριο πιο φιλικό-γραφικό περιβάλλον.

Κεφαλαιο 2

Βασικες Αρχες Προγραμματισμου

2.1 Σχεδιασμός Προγράμματος

2.1.1 Κατανομή του χρόνου μας

Ένα πρόγραμμα συνήθως θα ζητείτε να παραδοθεί μέσα σε συγκεκριμένο χρονικό διάστημα, αυτό βέβαια πάντα εξαρτάται από το πρόγραμμα, τη χρήση του και το πόσο καλό θέλουμε να είναι, δηλαδή πόσο κύκλο ζωή θα πρέπει να έχει. Επειδή όμως η πιστότητα του λογισμικού είναι αποτέλεσμα μεθοδικού σχεδιασμού θα πρέπει να κατανέμουμε το χρόνο μας κρατώντας κάποιες αναλογίες. Ενδεικτικά παραθέτουμε:

- 3 % Διατύπωση προβλήματος και ανάλυση απαιτήσεων: "Ποιο είναι το πρόβλημα", " Τι είναι μια αποδεκτή λύση".
- 5 % Προσδιορισμός λειτουργικότητας: Λεπτομερής προσδιορισμός του τι θα κάνει ο υπολογιστής, αλλά όχι πως θα το κάνει.
- 25 % Σχεδιασμός: Ανάπτυξη αλγορίθμων και δομών δεδομένων απαραίτητες για να υλοποιήσουν τα προσδιοριζόμενα. Χωρισμός σε modules. (Υποτίθεται καλή γνώση Θεωρίας αλγορίθμων και δομών δεδομένων.)

- 12 % Προγραμματισμός: μετατροπή του σχεδιασμού σε κώδικα.
- 15 % Testing: των modules και της ολοκλήρωσης τους
- 40 % Συντήρηση: διόρθωση bugs, επεκτάσεις, μεταφερσιμοτητα.

2.1.2 Σχεδιασμός Προγράμματος

Ακολουθώντας τα παραπάνω βήματα θα φτάσουμε τελικά στο επιθυμητό αποτέλεσμα. Όμως θα ήταν καλό πριν μπούμε σε πιο τεχνικά θέματα να δώσουμε κάποια εργαλεία για τον καθένα από τους παραπάνω 6 τομείς:

1. Ανάλυση απαιτήσεων και προσδιορισμός λειτουργικότητας: Φράσεις-περιγραφική ανάλυση.
2. Σχεδιασμός: γλώσσες προγραμματισμού - "ψευδό" γλώσσες, γραφικές τεχνικές, συμβάσεις, θεωρήματα, μεθοδολογίες.
3. Προγραμματισμός: γλώσσες προγραμματισμού.
4. Testing: Εργαλεία testing π.χ. συμβολικές εκτελέσεις, έλεγχος γνωστών αποτελεσμάτων, debuggers.
5. Συντήρηση: Εργαλεία συντήρησης π.χ. revision control systems, εργαλεία σύγκρισης αρχείων, debuggers.

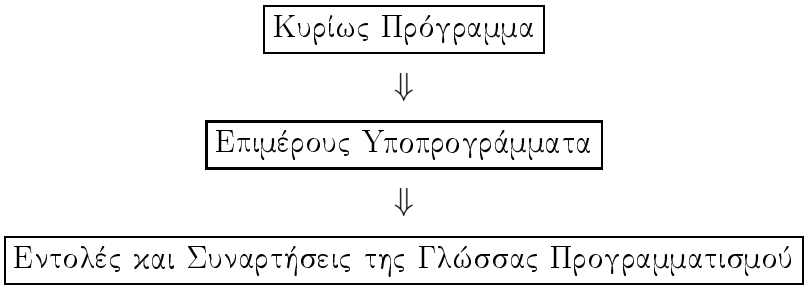
Για μικρά προγράμματα όλα τα στάδια μπορούν να γίνουν από ένα πρόσωπο. Κυρίως όμως σε μεγάλα προγράμματα υπάρχει μια κατανομή, που απαιτεί ο ένας να σκέφτεται τον άλλο και να λειτουργεί όσο πιο γενικά και πιο απλά μπορεί, ώστε και κάποιος άλλος στο μέλλον, να μπορεί να συνεχίσει ή να βελτιώσει αυτή τη δουλειά.

2.1.3 Μεθοδολογίες σχεδιασμού

Μετά την κατανομή του χρόνου, ένα βασικό μέρος ανήκει στη μεθοδολογία του σχεδιασμού ενός Προγράμματος. Ο κάθε προγραμματιστής θα πρέπει ν' ακολουθεί κάποιες μεθόδους αν θέλει στο τέλος να μην καταλήξει σε ένα χάος από άσκοπα κομμάτια που να μην δίνουν τίποτα το ουσιαστικό.
Οι δυο πιο διαδεδομένες μεθοδολογίες σχεδιασμού ενός Προγράμματος είναι:

- Top-down (από πάνω προς τα κάτω)
- Botton-up (από κάτω προς τα πάνω)

Στην πραγματικότητα και οι δυο προσεγγίσεις χρησιμοποιούνται, όμως εμείς θα αναλύσουμε από εδώ και κάτω μόνο την Top-down μεθοδολογία της οποία ένα σχεδιάγραμμα είναι το ακόλουθο:



Η μεθοδολογία Top-down ξεκινά με μια σύντομη λύση υψηλού επιπέδου και συνεχώς την ανασχηματίζει με διαδοχικούς μετασχηματισμούς σε λύσεις χαμηλότερου επιπέδου, ως τις εντολές γλώσσας προγραμματισμού (μπορούμε να φανταστούμε ότι έχουμε να κάνουμε με ένα δένδρο γυρισμένο ανάποδα και ξεκινώντας από την κορυφή ρίζα καταλήγουμε στα φύλλα).

Κάθε ανασχηματισμός πρέπει

- Να είναι μικρός,
- Να είναι σωστός,

- Να οδηγεί προς την τελική λύση.

Οι ανασχηματισμοί μπορούν να συνοδεύονται από δηλώσεις (- διαβεβαιώσεις) που θα βοηθήσουν την εξασφάλιση της ορθότητας.
Οι ανασχηματισμοί χρησιμοποιούν φράσεις και "ψευτο"- κώδικα, αλλά τελικά καταλήγουν σε κώδικα.

2.1.4 Προγραμματιστικό Στυλ

Πώς γράφω.

Πριν αρχίσουμε να μαθαίνουμε εντολές και συναρτήσεις από κάποια γλώσσα προγραμματισμού θα πρέπει πρώτα να μάθουμε το πώς θα είναι η εμφάνιση των προγραμμάτων μας ώστε να μπορούμε να πούμε ότι ο κώδικας που φτιάξαμε είναι "καλός". Γράφοντας "καλά" προγράμματα είναι σαν να γράφουμε καλά Ελληνικές προτάσεις: Ο σκοπός είναι η επικοινωνία.

Ο καλός κώδικας είναι ακριβής, ευθύς και δεν περιέχει άχρηστα μέρη, όπως ο καλός πεζός λόγος. Οι αρχές του καλού στυλ προγραμματισμού δεν εξαρτώνται από τη γλώσσα, απλός μερικές γλώσσες ενθαρρύνουν το καλό στυλ (όπως η C, η Pascal, κ.α.λ.), ενώ άλλες το αποθαρρύνουν, ανάλογα με τις λεπτομέρειες της γλώσσας. Όμως πάντα είναι στο χέρι μας να χρησιμοποιήσουμε τις καλές δυνατότητες μιας γλώσσας, αποφεύγοντας τις κακές.
Ο σκοπός μας είναι να γράφουμε με καθαρά και να μην θυσιάζουμε την ευκρίνεια για την ταχύτητα. Αφήστε τη μηχανή να κάνει τη βρώμικη δουλειά.
Επίσης:

- Μην βάζετε πολλές συμβολικές σταθερές.
- πρέπει να είναι φανερό από πού προέρχεται κάθε όρος στις πράξεις.
- Οι μεταβλητές να εκφράζουν μια ποσότητα (όχι σταθερά + ποσότητα).

Έλεγχος δομής.

Η ροή έλεγχου πρέπει να γράφεται για ανθρώπους. Γράφετε ευανάγνωστα !!! (όχι κατεβατά αλλά στοιχίζετε π.χ. τα while, if), χωριστέ το πρόγραμμα σε modules, χρησιμοποιείτε συναρτήσεις, κάθε modules πρέπει να κάνει ένα πράγμα καλά, αντικαταστήστε επαναλαμβανόμενες εκφράσεις με κλήσεις σε μια κοινή συνάρτηση. Γράφετε δομημένα (με loops, if-then- else,. . .), αποφεύγετε το goto.

Χρησιμοποιείτε for,while,do-while για τα loops και το if- else να υποδηλώσετε ότι μια μόνο περίπτωση θα γίνει. Απλοποιήστε τον κώδικα σας με διαδοχικά περάσματα και διαβάζοντας τον από την αρχή ως το τέλος. Χρησιμοποιείτε if-then-else για πολλαπλές αποβάσεις ή αν μπορείτε ακόμα καλύτερα Case. Καλό είναι να γράφουμε τους ελέγχους όσο πιο κοντά γίνεται στην δράση (π.χ. αποφεύγετε τα then-if χωρίς else, αφού if μέσα σε if όταν δεν ακολουθούνται από else μπορούν να συμπτυχθούν σε μια λογική πάξει δηλ. ένα if).

Κάνετε την **συνεργασία** των modules εμφανή. Χρησιμοποιείτε παραμέτρους, όχι global μεταβλητές για την επικοινωνία.

- Αφήστε τα δεδομένα να δομήσουν το πρόγραμμα.

```
while( there-is-more-input)
      if(we-are-at-the-top-of-a-page)
            write-header
      compute-info
      write-info-line
```

- Βεβαιωθείτε ότι κάθε module εκτελεί αυτοτελείς υπολογισμούς, τις λεπτομέρειες των οποίων δεν είναι απαραίτητο να γνωρίζει ο χρήστης (το βλέπουμε σαν μαύρο κουτί).

- βεβαιωθείτε πως ο κώδικας ελέγχει την καταλληλότητα των δεδομένων.

- Μην διορθώνετε κακό κώδικα, ξαναγράψτε τον.
- Απομονώνετε την είσοδο από την έξοδο σε υπορουτίνες.
- Προσοχή σε λάθη "παρά 1".

Προσπαθώντας να γράψουμε όλα τα παραπάνω εν περίληψη έχουμε:

- Το καλύτερο documentation περιλαμβάνει:
  - Καθαρή δομή
  - συνεπή μορφή
  - καλή επιλογή ονομάτων
  - διαφωτιστικά σχόλια
- Ο κώδικας πρέπει να γράφεται για να **διαβάζεται.** Αυτός είναι ο βασικός σκοπός!
- Τα σχόλια πρέπει να προσθέτουν **νέα** πληροφορία, όχι απλώς να περιγράφουν τον κώδικα. Κάθε σχόλιο πρέπει να είναι σημαντικό.
- Ο κώδικας και τα σχόλια πρέπει να συμφωνούν.
- Μην βάζετε σχόλια να δικαιολογήσετε άσκημο κώδικα, ξαναγράψτε τον.
- Εξοικειωθείτε με τη γλώσσα προγραμματισμού και κάνετε τον κώδικα να μιλά μόνος του.
- Να σχολιάζετε αλγορίθμους, όχι τις ιδιοσυγκρασίες της γλώσσας.

2.1.5 Πρόγραμμα και Ταχύτητα

Όπως είχαμε δει και στο κομμάτι της θεωρία αλγορίθμων δυο πράγματα κυρίως μας ενδιέφεραν η ταχύτητα- ο χρονος και η μνήμη-χώρος που χρειάζεται ένα πρόγραμμα. Εδώ θα ασχοληθούμε περισσότερο με το χρόνο και θα προσπαθήσουμε να δώσουμε τεχνικές που θα μας κάνουν το πρόγραμμα μας γρηγορότερο.

Δεν έχει όμως νόημα να προσπαθούμε να κάνουμε ένα πρόγραμμα γρηγορότερο όταν αυτό δεν δουλεύει. Γι αυτό είναι καλό να έχουμε πάντα στο μυαλό μας τους παρακάτω κανόνες:

- Να είναι σωστό πριν γίνει γρηγορότερο.
- Να είναι καθαρό πριν γίνει γρηγορότερο.
- Να παραμένει σωστό και καθαρό όσο γίνεται γρηγορότερο.
- Μην θυσιάζετε διαύγεια για μικρά κέρδη ταχύτητας.
- Αφήστε τον μεταφραστή να κάνει τις μικρές βελτιστοποιήσεις.
- Μην πιέζετε πολύ για να ξαναχρησιμοποιείτε τον κώδικα. Αντί για αυτό, ξαναοργανώστε τον.

Πολύς κακός κώδικας έχει προκύψει από λανθασμένες προσπάθειες να αυξηθεί η ταχύτητα. Το κέρδος είναι συχνά μικρό ή ανύπαρκτο. Βελτιώσεις στην ταχύτητα θα γίνονται **μόνο** αν υπάρχουν μετρήσεις που υποδεικνύουν τον αργό κώδικα. Η διαίσθηση των προγραμμάτων δεν είναι αρκετή!

Το βασικότερο όπως είπαμε είναι λοιπόν αρχικά το πρόγραμμα μας να δουλεύει σωστά και δεύτερο έρχεται η ταχύτητα. Συχνά όμως το πρόγραμμα μας είναι γραμμένο σε μια γλώσσα υψηλού επιπέδου και μετασχηματίζοντας το σε ισοδύναμο αλλά σε μια γλώσσα χαμηλότερου επιπέδου πετυχαίνουμε καλύτερους χρόνους. Η παραγωγή γρήγορου κώδικα σημαίνει εφαρμογή κάποιων τεχνικών με συστηματικό τρόπο,

όχι τυφλά να κάνουμε τεχνάσματα -hacking σ' ένα πρόγραμμα να το κάνουμε γρηγορότερο. Στον επόμενο πίνακα θα κάνουμε μια σύγκριση ανάμεσα στις τεχνικές και στα τεχνάσματα-hacks

Τεχνικές	Τεχνάσματα hacks
εφαρμόζονται μεθοδικά	αδιακρίτως
περιγράφονται επακριβώς	στο περίπου
βασίζονται σε	
επιστημονικές αρχές	αναλαμπές
καταγράφονται προσεκτικά	διαδίδονται από
	στόμα σε στόμα
περιγράφονται στο σωστό μέρος	οπουδήποτε
εφαρμόζονται με βεβαιότητα	
και δοκιμάζονται	χωρίς βεβαιότητα

Το γράψιμο γρήγορου κώδικα μπορεί να συνυπάρχει με το δομημένο σχεδιασμό και το καλό προγραμματιστικό στυλ. Ακόμα οι μετασχηματισμοί εφαρμόζονται σε τοπικά μέρη του προγράμματος μετά από στοιχεία για την ανάγκη τους και την αποτελεσματικότητα τους.

2.1.6 Παραγωγή γρήγορων προγραμμάτων

Ουσιαστικά στο μυαλό μας θα πρέπει να έχουμε πάντα ότι η ταχύτητα είναι δευτερεύουσα ανησυχία κατά την ανάπτυξη του προγράμματος: "Η βελτιστοποίηση που γίνεται πολύ νωρίς είναι η αρχή όλων των κακών". Πρώτα κάνουμε το σωστό, μετά κάνουμε το γρηγορότερο, αν χρειάζεται. Αν χρειάζεται βελτίωση, διαπιστώστε που το

πρόγραμμα ξοδεύει το χρόνο του και επικεντρώστε τις βελτιώσεις σε μερικά σημεία μόνο, "το 10 % του κώδικα παίρνει το 90 % του χρόνου".

Στην παράγραφο αυτή θα προσπαθήσουμε να δώσουμε τρόπους βελτιστοποιήσης της ταχύτητας ενός προγράμματος. Ακόμα θα αναλύσουμε κάποιους απ' αυτούς όσο δυνατό πιο σύντομα. Βασικοί παράγοντες που επηρεάζουν την ταχύτητα είναι:

1. Δομή συστήματος: Ολική δομή του συστήματος, όπως προσδιορίζεται κατά τη φάση του σχεδιασμού.
2. Δομή modules: Δομές δεδομένων και αλγόριθμοι που χρησιμοποιούνται σε module.
3. Οι κατάλληλοι μετασχηματισμοί του κώδικα μέσα σε κάποιο module να αυξηθεί η ταχύτητα.
4. Μετασχηματισμός σε assembly από γλώσσα υψηλού επιπέδου.
5. Λογισμικό συστήματος: Αλλαγές λειτουργικού συστήματος ή χρήση πληροφοριών για το συγκεκριμένο υπολογιστή.
6. Υλικό ειδικού σκοπού (Εξαρτάται από το μηχάνημα;)

Πού ξοδεύετε ο χρόνος;

Δεν είναι σωστό να προσπαθείτε να μαντέψετε το που ξοδεύεται ο χρονος, αφού το πιο πιθανό είναι συχνά να παραπλανούμαστε αλλά είναι και κάτι που χρειάζεται ιδιαίτερη πείρα. Χρησιμοποιείτε πάντα profilihg εργαλεία:

- Χρονος και μετρητές σε συναρτήσεις και που σχετίζονται με την προσπέλαση της κλήσης (prof, gprof στο Unix)
- Μετρητές εντολών (lcomp?).

- Εξοπλισμός του προγράμματος (μόνιμος με συναρτήσεις όπως η time στη C)
- Θεωρητικός υπολογισμός κόστους (διαφορετικό από μετρητές).

Στο Unix μεταφράζουμε προγράμματα με κώδικα για profiling με το -p κατά την κλήση του μεταφραστή (π.χ. gcc -p filename.c). Η εκτέλεση παράγει μετρήσεις στο mon.out το οποίο τυπώνει η εντολή prof. Μπορούμε να χρησιμοποιήσουμε και το gprof και προγράμματα που μετρούν εντολές.

Βασική είναι η υπόθεση σε αυτό το στάδιο προγραματισμού είναι ότι οι αλγόριθμοι που χρησιμοποιούμε είναι όσο το δυνατόν καλύτεροι τόσο από την πλευρά του χώρου, αλλά και του χρόνου και το μόνο που εμείς έχουμε να βελτιώσουμε είναι ο κώδικας μας, αφού εδώ ασχολούμαστε μόνο με το προγραμματιστικό μέρος. Σημαντικά εργαλεία για αλγοριθμικές βελτιώσεις όπως έχουμε προαναφέρει μπορούμε να βρούμε μέσα στη θεωρία αλγορίθμων π.χ. η αντικατάσταση της σειριακής αναζήτησης σε ταξινομημένους πίνακες με την δυαδική αναζήτηση.

Τροποποιώντας δομές δεδομένων

Ανταλλαγή χρόνου με χώρο (**packing**. Αν το πρόγραμμα χρησιμοποιεί πολύ χώρο, συμπυκνώστε δεδομένα σε μικρότερες μονάδες (κοστίζει περισσότερο η προσπέλαση) π.χ. λέξεις μεταβλητού μήκους.

Ανταλλαγή χώρου με χρόνο. Αντικατάσταση κώδικα με δεδομένα (π.χ. προϋπολογισμένα δεδομένα) άρα λιγότερο κώδικα, που ερμηνεύει τα δεδομένα. Ακόμα μπορείτε να μεγαλώσετε τις δομές και να βάλετε μέσα πληροφορίες που θα αυξήσουν την ταχύτητα. Αποθηκεύστε προϋπολογισμένα αποτελέσματα. Cachihg: τα δεδομένα που χρησιμοποιούνται περισσότερο πρέπει να γίνουν τα φθηνότερα στην προσπέλαση. Καθυστερείστε την αποτίμηση - τον υπολογισμό τιμών ώσπου να χρειαστούν (π.χ. έξοδο).

Προσοχή: αν δεν γίνει σωστά μπορεί να θέλει και πιο πολύ χώρο και πιο πολύ χρόνο.

Τροποποιώντας Loops

Ο περισσότερος χρονος κατά κοινή ομολογία δαπανάται κυρίως μέσα στα loops. Κάποιες βελτιώσεις που μπορείτε να κάνετε σε αυτά είναι το "ξετύλιγμα" του loop, μειώνει το κόστος της έξτρα αναφοράς. Δηλαδή να κάνετε αναφορές σε ομάδες στοιχείων (π.χ. i. . .i+5 σε ένα μεγάλο εσωτερικό γινόμενο) ή ανάλογα με το μετασχηματισμό (π.χ. μόνα ζυγά). Απάλειψη του σταθερού jump, με ανά σχηματισμό του loop. Σύμπτυξη πολλών loop σε ένα. Συνδυάζουμε πολλούς κανόνες. Συμπυκνώνουμε τον έλεγχο του loop π.χ
while($i < n$ && x[i]!=Null)
. . . . . .
Γίνεται με χρήση φρουρού
x[n]=Null;
while(x[i]!=Null)
. . . . . .

Τροποποιώντας λογικούς ελέγχους και συναρτήσεις

Εξερευνείσθε αλγεβρικές ταυτότητες για να ελαττώσετε το κόστος αποτίμησης. "Παραλείψετε" συναρτήσεις που μπορούν να γραφούν άμεσα στον κώδικα. Προϋπολογίστε συναρτήσεις και αντικαταστήσετε συναρτήσεις με μικρό πεδίο ορισμού με προσπέλαση σε πίνακα. Αναδιάταξε τους έλεγχους: βάλτε τον πιο συχνό και φθηνό έλεγχο πρώτα. Αντικαταστήστε λογικές μεταβλητές με λογική του προγράμματος. Εντοπίστε τετριμμένες κλήσεις (π.χ. a=b; c=a; αντίστοιχα c=b;)


ΓΡΑΦΟΥΜΕ ΚΑΘΑΡΟ ΚΩΔΙΚΑ
ΜΕ ΚΑΛΟ ΣΤΥΛ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ
ΜΕΤΡΑΜΕ ΤΗΝ ΤΑΧΥΤΗΤΑ
ΕΦΑΡΜΟΖΟΥΜΕ ΣΥΓΚΕΚΡΙΜΕΝΟΥΣ
ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥΣ
ΤΕΛΕΙΩΣΑΜΕ

Κεφαλαιο 3

Βασικες Εντολες Ενος Προγραμματος MATLAB

Τα προγράμματα στο MATLAB γράφονται μέσα στον κειμενογράφο του σύμφωνα με τις παρακάτω οδηγίες: Γράφουμε μία εντολή ανά γραμμή, για πάνω από μια (ανά γραμμή) θα πρέπει να τις χωρίζουμε μεταξύ τους με ελληνικό ερωτηματικό ";" ή κόμμα ",". Όταν βάζουμε ερωτηματικό στο τέλος μιας εντολής δεν εμφανίζεται το αποτέλεσμα της πράξεις στην οθόνη κατά την εκτέλεση αυτής. Σχόλια μπορούμε να γράφουμε σε οποιοδήποτε σημείο του προγράμματος "ακόμα μαζί εντολές και σχόλια".

3.1 Μεταβλητές-Σταθερές

3.1.1 Ονομασία μεταβλητών - υποπρογραμμάτων

Τα ονόματα των μεταβλητών πρέπει να δίνονται βάσει των ακόλουθων κανόνων: Επιτρέπονται τα γράμματα του λατινικού αλφαβήτου $A, \ldots, Z$ και τα ψηφία $0, \ldots, 9$. Ο πρώτος χαρακτήρας πρέπει να είναι πάντα γράμμα. Γίνεται διάκριση μεταξύ πεζών και κεφαλαίων. Πρέπει να προσέχουμε να μην χρησιμοποιούμε τυχόν δεσμευμένες λέξεις

(π.χ. εντολές MATLAB). Καλό είναι κάθε μεταβλητή (μια και δεν υπάρχει ειδικός τρόπος δήλωσης) πριν την χρησιμοποιήσουμε να την καθαρίζουμε με την εντολή clean π.χ. **clean** x.

Παράδειγμα:

Αποδεκτά ονόματα μεταβλητών:
GEORGE, A3, PROD, sum_1, SINUS, IWRITE.

Μη αποδεκτά ονόματα μεταβλητών:
A$3, sum-1, SIN, WRITE, 3a, 1WRITE.

Προσοχή δεν πρέπει να ορίζουμε ως μεταβλητή το ὅιὺ μια και χρησιμοποιείτε για να συμβολίζουμε τους μιγαδικούς αριθμούς.

Παρατήρηση: Πρέπει να είστε προσεκτικοί γιατί συνήθως γίνεται σύγχυση μεταξύ του ψηφίου 0 και του γράμματος O, καθώς και του ψηφίου 1 με το γράμμα I.

Παραδείγματα
clean SINUS,X1,X2
clean FLAG

3.1.2 Τα πάντα είναι πίνακες!

Βασικό αντικείμενο του Matlab είναι οι **πίνακες** (πραγματικοί ή μιγαδικοί).

Σε μερικές περιπτώσεις το Matlab ερμηνεύει:

- πίνακες 1×1 σαν βαθμωτά μεγέθη, και

- πίνακες με 1 γραμμή ή 1 στήλη σαν διανύσματα.

Η γλώσσα του Matlab είναι α-τυπη (δεν χρειάζεται δήλωση μεταβλητών).

Στο Matlab οι πράξεις κινητής υποδιαστολής γίνονται σύμφωνα με το standard της IEEE, συνήθως σε διπλή ακρίβεια.

3.2 Αριθμητικές παραστάσεις

Είναι φυσικό το MATLAB διατηρεί πολλά στοιχεία μια γλώσσα προγραμματισμού αλλά να μην ξεχνάμε ότι πάνω από όλα είναι ένα μαθηματικό εργαλείο. Έτσι λογικό είναι να υπάρχει διατηρήσει των συμβόλων και των ιδιοτήτων τους.

3.2.1 Προτεραιότητα τελεστών

Η προτεραιότητα των τελεστών όπως είπαμε και παραπάνω είναι όπως τα μαθηματικά. Έτσι αρκεί μια υπενθύμιση, παράλληλα με την παρουσίαση των συμβόλων που χρησιμοποιούμε για να γράψουμε τις μαθηματικές παραστάσεις στη γλώσσα **MATLAB.**

- Οι συνηθισμένοι δυαδικοί αριθμητικοί τελεστές : $+, -, *, /\hat{}$.
- Ο τελεστής modulus (υπόλοιπο) είναι %. Το x%y ορίζεται μόνο αν $x, y > 0$ (και x,y ακέραιοι) και συμβολίζει το υπόλοιπο όταν το x διαιρείται με το y (δηλαδή $x\%y \equiv x - x/y$).
- Προσετεριστικότητα από τα αριστερά προς τα δεξιά, δηλαδή το a-b-c θεωρείται ως ((a-b)-c).

Πίνακας Προτεραιοτήτων

( )	"παρενθέσεις"
^	Ύψωση σε δύναμη.
*, /, mod	"πολλαπλασιασμός","διαίρεση","modulo"
+, -	"πρόσθεση", "αφαίρεση"

Ο Πίνακας απεικονίζει τη σειρά των πράξεων, χωρισμένες σε επίπεδα προτεραιότητας, π.χ. πρώτα γίνονται οι παρενθέσεις και μετά οι *, /, mod, ενώ στο τέλος οι +,-.

Παρατηρήσεις:

Όταν δύο τελεστές έχουν την ίδια προτεραιότητα η έκφραση υπολογίζεται από αριστερά προς τα δεξιά. Σε περίπτωση αμφιβολίας ως προς τη σειρά υπολογισμού συνιστάται η χρήση παρενθέσεων. Οι πράξεις όλες γίνονται συμβολικά π.χ. 2/12 μας δίνει 1/6 και όχι 0.166667.

Παραδείγματα:

Η έκφραση A*B-C/D υπολογίζεται ως: (A*B)-(C/D) Η έκφραση A/B*C υπολογίζεται ως: (A/B)*C

Παράδειγμα υπολογισμός ριζών δευτεροβάθμιας εξίσωσης

Λάθος (Προσοχή στη σειρά...)

```
d = sqr(b) - 4*a*c;
root1= -b + sqrt(d) / 2*a;
root2= -(b -sqrt(d))/(2*a);
```

Σωστό

```
d= sqr(b) - 4*a*c;
```

```
root1= (-b + sqrt(d)) / (2*a);
root2= (-b -sqrt(d))/(2*a);
```

Άλλοι τελεστές του Octave[1]

Το Octave έχει και κάποιους άλλου δικούς της τελεστής όπως :
++ ή −− που αυξάνουν ή μειώνουν κατά ένα μια μεταβλητή π.χ. $i++$ είναι το i=i+1, ενώ $i--$ είναι το i=i-1. Ακόμα έχουμε και το +=,-=,*=,/= κ.α. που σημαίνουν π.χ. i+=a είναι το i=i+a, i-=a είναι το i=i-a, i*=a είναι το i=i*a, i/=a είναι το i=i/a. Προσοχή Το i*=a+2 είναι το i=i*(a+2) και όχι το i=i*a+2.

3.3 Μαθηματικές Συναρτήσεις

Αν και γενικότερα για συναρτήσεις θα μιλήσουμε παρακάτω είναι καλό να δείξουμε μερικές συναρτήσεις για άμεση χρήση. Μερικές από τις πιο συνηθισμένες μαθηματικές συναρτήσεις είναι:

sin(x)	ημίτονο του x
cos(x)	συνημίτονο τον x
tan(x)	εφαπτομένη του x
asin(x)	τόξο ημιτόνου x στο
	διάστημα [-π/2, π/2], x [-1, 1].
acos(x)	τόξο συνημιτόνου x στο
	διάστημα [0, π], x [-1, 1].
atan(x)	τόξο εφαπτομένης x στο
	διάστημα [-π/2, π/2].

[1] Οι παρακάτω τελεστές δεν δουλεύουν σε όλες τις εκδόσεις του MATLAB αλλά χρησιμοποιούνται στο Octave.

atan2(y,x)	τόξο εφαπτομένης y/x στο διάστημα [-π, π].
sinh(x)	υπερβολικό ημίτονο του x.
cosh(x)	υπερβολικό συνημίτονο του x.
tanh(x)	υπερβολική εφαπτομένη του x.
exp(x)	εκθετική συνάρτηση e^x.
log(x)	φυσικός λογάριθμος ln(x), $x > 0$.
log10(x)	δεκαδικός λογάριθμος log10(x),$x > 0$.
sqrt(x)	$\sqrt{x}, x >= 0$.
floor(x)	ο μεγαλύτερος ακέραιος που δεν υπερβαίνει το x, σαν double.
abs(x)	η απόλυτη τιμή $\|x\|$.
mod(x,y)	το υπόλοιπο της x/y, σε μορφή κινητής υποδιαστολής με πρόσημο ίδιο με του x. Αν το y είναι μηδέν, το αποτέλεσμα ορίζεται από την υλοποίηση.

3.4 Εντολές Ανάθεσης

Η απλούστερη εντολή στη MATLAB είναι η ανάθεση.

- Το Matlab ερμηνεύει κάθε γραμμή στην είσοδο. Οι εντολές του έχουν τη μορφή:

  μτβ=έκφραση
  ή απλά
  έκφραση

  η οποία αναθέτει την τιμή της έκφρασης εκφρ στη μεταβλητή μτβ.

- Οι εκφράσεις είναι σύνθεση τελεστών, μεταβλητών και συναρτήσεων. Ο υπολογισμός τους παράγει ένα πίνακα που μπορεί να εμφανισθεί στην έξοδο ή να αποθηκευτεί σε μεταβλητή.

- Οι εντολές τερματίζονται με το τέλος της γραμμής. Συνέχεια σε περισσότερες από μία γραμμές αν η προηγούμενη γραμμή τελειώνει σε ...

- Πολλές εντολές μεταξύ ,ή ; γράφονται σε μία γραμμή.

- Εντολή που τελειώνει σε ; δεν παράγει output στην οθόνη.

Για να μην υπάρχει καμία σύνχυση, διευκρινίζουμε ότι το σύμβολο '=' σημαίνει **ανάθεση,** δηλαδή πρώτα γίνονται στο δεξιό μέρος του οι πράξεις (γενικά αποτιμάται η παράσταση το δεξιού μέρους) και μετά γίνεται ανάθεση στη μεταβλητή που βρίσκεται στο αριστερό μέλος, σημειώνουμε ότι δεν μπορούμε να βάλουμε πάνω από μια μεταβλητή στο αριστερό μέλος, σε αντίθεση με το δεξιό, π.χ. i=a+b*k+2; Εκτός το σύμβολο της ανάθεσης '=', έχουμε και το σύμβολο της **ισότητας** που συμβολίζεται με '==' (δυο '='). 'Ομως το '==' αυτό που κάνει είναι να αποτιμά την παράσταση που βρίσκεται στο δεξιό και στο αριστερό μέρος αντίστοιχα (που μπορεί να είναι οτιδήποτε) και αν είναι ίσα επιστρέφει 1 σε αντίθετη περίπτωση 0, π.χ. 2*i+a+3==a*a+3*k αρκεί να βρούμε τις τιμές για τις οποίες γίνεται αληθείς εκεί θα παίρνει την τιμή 1, ενώ σε αντίθετη περίπτωση θα επιστρέφει 0.

Π.χ. RES := B + C*D^3 σημαίνει RES = B + C D3 .

Ο τύπος της εκφρ πρέπει εν γένει να συμφωνεί με τον τύπο της μτβ. Σε αντίθετη περίπτωση η τιμή της εκφρ μετατρέπεται αυτόματα στον τύπο της μτβ προτού γίνει ανάθεση. Π.χ. B = 9. Στην πραγματική μεταβλητή B ανατίθεται η τιμή 9.0. II = 1.9. Στην ακέραια μεταβλητή II ανατίθεται η τιμή 1.

3.5 Εντολές Εισόδου - Εξόδου

Στα περισσότερα προγράμματα απαιτείται η εισαγωγή δεδομένων από το χρήστη, συνήθως από το πληκτρολόγιο. Σχεδόν δε πάντα ένα χρήσιμο πρόγραμμα εκτυπώνει κάποια αποτελέσματα, π.χ. στην οθόνη. Για τους σκοπούς αυτούς χρησιμοποιούμε τις εντολές εισόδου - εξόδου.

Παρακάτω θα δούμε επίσης, πως μπορούμε να διαβαστούν δεδομένα από αρχείο, αλλά αντίστοιχα και να γραφούν σε αρχείο.

3.5.1 Εντολές Εισόδου

Στο MATLAB όμως δεν υπάρχουν στην ουσία εντολές διαβάσματος από το πληκτρολόγιο μια και το πρόγραμμα εκτελείται άμεσα μέσα από τον κειμενογράφο του, γι αυτό και δεν έχει ιδιαίτερο λόγο ύπαρξης της δυνατότητα να διαβάζουμε από το πληκτρολόγιο μια και μπορούμε να τα έχουμε καταχώρηση τα δεδομένα μας ώς σταθερές. Συνιστάται οι σταθερές να δηλώνονται στην αρχή του προγράμματος (για να μπορούμε να τις αλλάζουμε εύκολα) αφού πρώτα τις κάνουμε clean (για να μην έχουνε κρατήσει δεδομένα από προηγούμενες αναθέσεις). Παραπάνω συνιστούμε τον απλούστερο τρόπο διαβάσματος, αυτό όμως δεν συμμένει ότι το MATLAB δεν διαθέτει και άλλους. Η εντολή Input θα λέγαμε ότι εισάγει δεδομένα με ιδιαίτερα εντυπωσιακό τρόπο (ανοίγει ξεχωριστώ γραφικό περιβάλλον) αλλά παράλληλα και πολύ απλό.
Σύνταξη:

$$\mu\tau\beta = \mathbf{input}('\kappa\epsilon\iota\mu\epsilon\nu o')$$

3.5.2 Εντολές Εξόδου

Αντίστοιχα με τις εντολές διαβάσματος, πολύ εύκολα μπορούμε να τυπώσουμε και τα αποτελέσματα στην οθόνη, αρκεί απλά να μην βάλουμε ελληνικό ερωτηματικό στο τέλος την πράξεις ή της μεταβλητής που θέλουμε να εκτυπωθεί. Άλλες εντολές που μας δίνουν τη δυνατότητα να τυπώνουμε αποτελέσματα στην οθόνη είναι η **disp**, **sprintf**[2],**fprintf**,... Σύνταξη:

```
disp('εκφρ1')
disp(μτβ)
```

Ακόμα συναντάμε συχνά τη συνάρτηση sprintf, που δεν κάνει τίποτα άλλο από το να μας παρέχει μια φορμαρισμένη έξοδο. παρακάτω παρουσιάζουμε όλη την οικογένεια συναρτήσεων εξόδου.
int fprintf(FILE *strenm, const char *format, . . .)

Η fprintf μετατρέπει, και γράφει την έξοδο στο stream, κάτω από τον έλεγχο της format. Επιστρεφόμενη τιμή είναι ο αριθμός των χαρακτήρων που γράφονται, ή αρνητική τιμή σε περίπτωση λάθους.
Το αλφαριθμητικό φόρμας περιέχει δύο τύπους αντικειμένων: κοινούς χαρακτήρες, που αντιγράφονται στο ρεύμα εξόδου, και προδιαγραφές μετατροπών, που καθεμιά προκαλεί τη μετατροπή και εμφάνιση του επόμενου ορίσματος της fprintf. Κάθε προδιαγραφή μετατροπής αρχίζει με το χαρακτήρα % και τελειώνει με ένα χαρακτήρα μετατροπής. Ανάμεσα στο % και στο χαρακτήρα μετατροπής μπορεί να υπάρχουν, με τη σειρά:

- Σημαίες (με οποιαδήποτε σειρά) που τροποποιούν την προδιαγραφή:

[2]Δεν βάζουμε ελληνικό ερωτιματικό στο τέλος αν θέλουμε να δούμε το αποτελεσμα, ή αλλιώς τη γράφουμε μέσα σε disp π.χ. disp(sprintf('κείμενο-μεταβλητές',μεταβλητές));

- : που καθορίζει αριστερή στοίχιση του μετατρεπόμενου ορίσματος στο πεδίο του.

+ : που καθορίζει ότι ο αριθμός θα εμφανίζεται πάντα με πρόσημο.

␣ : αν ο πρώτος χαρακτήρας δεν είναι πρόσημο, θα μπαίνει μπροστά του ένα κενό διάστημα.

0 : για αριθμητικές μετατροπές, καθορίζει συμπλήρωση του πλάτους του πεδίον με μηδενικά στην αρχή.

: που καθαρίζει μια εναλλακτική μορφή εξόδου. Για το o, το πρώτο ψηφίο θα είναι μηδέν. Για x ή X, μπροστά σε μη-μηδενικό αποτέλεσμα θα μπαίνει 0x ή 0X. Για e, E, f, g, και G, η έξοδος θα έχει πάντα

– υποδιαστολή. Για g και G, τα μη-σημαντικά μηδενικά δεν αφαιρούνται από το τέλος.

- Ένας αριθμός που καθορίζει το ελάχιστο πλάτος πεδίου. Το μετατρεπόμενο όρισμα θα εμφανιστεί σε ένα πεδίο με τουλάχιστον αυτό το πλάτος. Αν το μετατρεπόμενο όρισμα έχει λιγότερους χαρακτήρες από το πλάτος του πεδίου, θα συμπληρωθεί με κενό στα αριστερά για να συμπληρωθεί το πλάτος του πεδίου. Ο χαρακτήρας συμπλήρωσης κανονικά είναι το κενό διάστημα, αν όμως υπάρχει σημαία συμπλήρωσης με μηδέν, είναι ο 0.

- Μια τελεία, που χωρίζει το πλάτος του πεδίου από την ακρίβεια.

- Ένας αριθμός, η ακρίβεια, που καθορίζει το μέγιστο αριθμό χαρακτήρων που θα εμφανιστούν από ένα αλφαριθμητικό, ή τον αριθμό των ψηφίων που θα εμφανιστούν μετά από την υποδιαστολή για μετατροπές e, E, f, ή τον αριθμό των σημαντικών ψηφίων για μετατροπές g ή G, ή τον ελάχιστο αριθμό ψηφίων που θα εμφανιστούν για έναν ακέραιο (στην αρχή του θα προστεθούν μηδενικά για να συμπληρώσουν το αναγκαίο πλάτος πεδίου).

- Ένα χαρακτήρα h,l (το γράμμα ελ) ή L. Το "h" υποδηλώνει ότι το αντίστοιχο όρισμα θα εμφανιστεί σαν short ή unsigned short. Το "l" υποδηλώνει ότι το όρισμα είναι long ή unsigned long. Το "L" υποδηλώνει ότι το όρισμα είναι long double.

Το πλάτος, η ακρίβεια, ή και τα δύο μπορούν να καθορίζονται σαν *, οπότε η τιμή υπολογίζεται με τη μετατροπή του επομένου ορίσματος (ή ορισμάτων), που πρέπει να είναι int.
Οι χαρακτήρες μετατροπής και η σημασία τους δίνονται στον παρακάτω πίνακα. Αν ο χαρακτήρας μετά το % δεν είναι χαρακτήρας μετατροπής, η συμπεριφορά είναι απροσδιόριστη.

Πίνακας Ορισμάτων της sprintf.

ΧΑΡΑΚΤΗΡΑΣ	ΤΥΠΟΣ ΟΡΙΣΜΑΤΟΣ ΕΜΦΑΝΙΖΕΤΑΙ ΣΑΝ
d, i	int προσημασμένος δεκαδικός συμβολισμός.
o	int απροσημός οκταδικός συμβολισμός (χωρίs μηδέν μπροστά).
x,X	int απρόσιμός δεκαεξαδικός συμβολισμός (χωρίς 0x ή0X μπροστά), που χρησιμοποιεί τα abcdef για 0x ή τα ABCDEF για 0X.
u	int απρόσιμος δεκαδικός συμβολισμός.
c	char ένας χαρακτήρας, μετά από

	μετατροπεί σε unsigned char.
s	char* -εμφανίζονται χαρακτήρες του αλφαριθμητικού μέχρι τον $'\backslash 0'$ ή τόσοι χαρακτήρες όσους ορίζει η ακρίβεια.
f	double δεκαδικός συμβολισμός της μορφής [-]m.dddddd όπου ο αριθμός των d καθορίζεται από την ακρίβεια η πρακαθορισμένη ακρίβεια είναι 6 αν οριστεί ακρίβεια 0, δεν εμφανίζεται υποδιαστολή.
e,E	double δεκαδεκός συμφολισμός της μορφής [-]m.ddddde ± xx ή [-]m.ddddddE ± xx, όπου ο αριθμός των d καθορίζεται από την ακρίβεια η προκαθορισμένη ακρίβεια είναι 6 αν οριστεί ακρίβεια 0, δεν εμφανίζεται υποδιαστολή.
g, G	double χρησιμοποιείται %e ή %E αν ο εκθέτης είναι μικρότερος από -4 ή μεγαλύτερος από ή ίσος με την ακρίβεια, διαφορετικά χρησιμοποιείται %f. Τα μηδενικά και η υποδιαστολή στο τέλος δεν εμφανίζονται.
p	void* εμφανίζει σαν δείκτη (η αναπαράσταση εξαρτάται από την υλοποίηση).
n	int* ο αριθμός των χαρακτήρων που έχουν γραφτεί μέχρι στιγμής απ' αυτήν την κλήση της printf γράφεται στο όρισμα. Δεν μετατρέπεται κανένα όρισμα..
%	δεν μετατρέπεται κανένα όρισμα

εμφανίζεται ένα %.

int sprintf(const char *format,. . .)

Η sprintf(. . .) είναι ισοδύναμη με την fprintf(stdout,. . .).

Η sprintf είναι ίδια με την printf με τη διαφορά ότι η έξοδος γράφεται στο αλφαριθμητικό s, που τερματίζεται με $'\backslash 0'$. Το s πρέπει να είναι αρκετά μεγάλο για να χωράει το αποτέλεσμα. Η επιστρεφόμενη μέτρηση δεν περιλαμβάνει το $'\backslash 0'$.

vprintf(const char *format, va_list arg)

vfprintf(FILE stream, const char *format, va_list arg)

vsprintf(char *s, const char *format, va_list arg)

Οι συναρτήσεις vprintf, vfprintf, και vsprintf είναι ισοδύναμες με τις αντίστοιχες συναρτήσεις printf, με τη διαφορά ότι η μεταβλητή λίστα ορισμάτων αντικαθίσταται από την arg, που παίρνει υπόσταση με τη μακροεντολή va_start και ίσως από κλήσεις της va_arg.

Παράδειγμα: Να γραφεί πρόγραμμα που θα διαβάζει την ακτίνα r ενός κύκλου και θα υπολογίζει το Εμβαδό και την Περίμετρο του.

```
%Πρόγραμμα Κύκλος

%Διαβάζουμε την ακτίνα
r:=input['Δώσε την ακτίνα του κύκλου'];
%Υπολογίζουμε το Εμβαδό
```

```
E=pi*r^2;
%Υπολογίζουμε την Περίμετρο
S=2*pi*r
%Τυπώνουμε τα αποτελέσματα
disp('Το Εμβαδό είναι:'),disp(E);
sprintf('Η Περίμετρο είναι: %f',S) %προσοχή όχι ερωτηματικό!!!!!
```

Κεφαλαιο 4

Εντολες Ελεγχου-Επαναληψης

Στα προγράμματα που είδαμε, οι εντολές εκτελούνται η μία μετά την άλλη. Σε πολλές περιπτώσεις όμως, η λύση του προβλήματος εξαρτάται από τις συνθήκες που επικρατούν κατά την εκτέλεση του προγράμματος εισαγωγή στοιχείων από το χρήστη τρέχουσες τιμές μεταβλητών. Τότε πρέπει να ληφθεί μία απόφαση, και από το αποτέλεσμά της θα εξαρτηθεί ποιοι υπολογισμοί θα γίνουν. Χαρακτηριστικό παράδειγμα είναι η λύση μιας δευτεροβάθμιας εξίσωσης $ax^2 + bx + c == 0$ που το είδος (π.χ. δυο πραγματικές, μια διπλή ή δυο μιγαδικές) της λύσης εξαρτάται άμεσα από την διακρίνουσα. Οι παράγοντες που επηρεάζουν την απόφαση σχηματίζουν μία λογική παράσταση, δηλ. μία παράσταση που μπορεί να είναι ή αληθής ή ψευδής.

4.0.3 Λογικές Εκφράσεις

Είναι εκφράσεις που παίρνουν τις τιμές: TRUE ή FALSE. Σχηματίζονται συνήθως από συγκρίσεις αριθμητικών δεδομένων (τελεστές συσχέτισης) και συνδυασμούς τους (λογικοί τελεστές).

Στον πίνακα που ακολουθεί δίνουμε την μορφή των τελεστών συσχέτισης στο

MATLAB και την αντιστοιχία τους στη C.

MATLAB	==	~ =	>	>=	<	<=
C	==	! =	>	>=	<	<=

Εκτός από τους τελεστές συσχέτισης έχουμε και σύμβολα που ορίζουν πράξεις μεταξύ δυο λογικών μεταβλητών όπως:

Σύμβολο	σημασία
~	άρνηση (NOT)
&&	το "και" (AND)
||	το διαζευκτικό "ή" (OR)

υπάρχουν και άλλες πιο σύνθετες πράξεις αλλά δεν είναι ο στόχος μας να φορτώσουμε τον αναγνώστη με περιττά, αντίθετα θα δώσουμε κάποια "τρικ" για να κάνουμε πιο εύκολη την εντριβή τις παραπάνω λογικές πράξεις, όπως για το && θα είναι πιο εύκολο να έχουμε στο μυαλό μας ότι είναι ένας πολλαπλασιασμός, ενώ για το || ότι είναι μια πρόσθεση. Δείτε με προσοχή τον παρακάτω πίνακα που υλοποιεί τις παραπάνω λογικές πράξεις μεταξύ δυο μεταβλητών a,b:

Στον ακόλουθο δίνουμε την μορφή των λογικών τελεστών στο MATLAB και την αντιστοιχία τους στη C.

MATLAB	~	&&	||
C	!	&&	||

Π.χ. η έκφραση $(A <= B)$ είναι αληθής αν η τιμή της μεταβλητής A είναι μεγαλύτερη ή ίση από την τιμή της μεταβλητής B.

Λογικές πράξεις

a	b	~a	a&&b	$a\|\|b$
A(1)[1]	A(1)	Ψ(0)[2]	A(1)	A(1)
A(1)	Ψ(0)	Ψ(0)	Ψ(0)	A(1)
Ψ(0)	A(1)	A(1)	Ψ(0)	A(1)
Ψ(0)	Ψ(0)	A(1)	Ψ(0)	Ψ (0)

Προτεραιότητα τελεστών:

- ( ) Παρενθέσεις
- Αριθμητικοί τελεστές
- Τελεστές συσχέτισης
- Λογικοί τελεστές

Προτεραιότητα ΛΟΓΙΚΩΝ τελεστών

()

!(NOT)

&&(AND), ||(OR)

==, ! =, <, <=, >, >=

Ιδιότητες των λογικών πράξεων

Αντιμεταθετική:

$a\&\&b == b\&\&a$

$a||b == b||a$

Προσεταιριστική:

$(a\&\&b)\&\&c == a\&\&(b\&\&c)$

$(a||b)||c == a||(b||c)$

Επιμεριστική:

$(a\&\&b)||c == (a||c)\&\&(b||c)$

$(a||b)\&\&c == (a\&\&c)||(b\&\&c)$

Νόμοι De Morgan:

$\tilde{}(a||b) == (\tilde{}a)\&\&(\tilde{}b)$

$\tilde{}(a\&\&b) == (\tilde{}a)||(\tilde{}b)$

Παράδειγμα:

Οι ακόλουθες δύο γραμμές είναι ισοδύναμες

$$A >= B\&\&B >= C$$

$$(A >= B)\&\&(B >= C)$$

4.0.4 Εντολές Ελέγχου

Η Εντολή Ελέγχου if

Μια εντολή που μπορεί να ελέγξει αν μια λογική παράσταση αν είναι ή αληθής ή ψευδής και ανάλογα να πάρει μια απόφαση είναι εντολή if.

```
"Αν"(λογική παράσταση) "τότε"
   εντολές (εκτελούνται αν λογική παράσταση αληθής)
"Τέλος Αν"

"Αν"(λογική παράσταση) "τότε"
   εντολές (εκτελούνται αν λογική παράσταση αληθής)
"Αλλιώς"
```

```
   εντολές (εκτελούνται αν λογική παράσταση ψευδής),
"Τέλος Αν"
```

Η σύνταξη της εντολής if είναι ανάλογη όπως στην γλώσσα (π.χ. ελληνικά, αγγλικά) που μιλάμε, αφού θα λέγαμε στα ελληνικά
Σύνταξη:

```
if(λογική παράσταση)
   εντολές (εκτελούνται αν λογική παράσταση αληθής)
end

if(λογική παράσταση)
   εντολές (εκτελούνται αν λογική παράσταση αληθής)
else
   εντολές (εκτελούνται αν λογική παράσταση ψευδής),
end

if(λογ. Παράστ. 1)
  εντολές (εκτελούνται αν η 1 είναι αληθής)
elseif (λογ. Παράστ. 2)
      εντολές (εκτελούνται αν η 2 είναι αληθής)
     .
     .
     .
      elseif(λογ. παράστ. n)
               εντολές (εκτελούνται αν η n είναι αληθής)
           else
               εντολές (εκτελούνται αν οι 1,2,...,n  είναι ψευδείς)
end
```

[3]

Παράδειγμα:
Για να βρούμε το πρόσημο SIGNUM ενός αριθμού NUMBER ($SIGNUM = 1$ για $NUMBER > 0$, $SIGNUM = -1$ για $NUMBER < 0$, $SIGNUM = 0$ για $NUMBER = 0$) μπορούμε να χρησιμοποιήσουμε την ακόλουθη δομή:

```
if(NUMBER < 0)
        SIGNUM = -1;
elseif(NUMBER > 0)
               SIGNUM = 1;
     else
               SIGNUM = 0;
end
```

ή

```
if(NUMBER < 0)
        SIGNUM = -1;
else if(NUMBER > 0)
               SIGNUM = 1;
     else
               SIGNUM = 0;
     end
end
```

Το αντίστοιχο πρόγραμμα C είναι:

```
if (number < 0){
```

[3]Προσοχή: Το elseif θα πρέπει να είναι μια λέξη τότε μόνο χρειάζεται να βάλουμε μόνο ένα end αλλιώς θα πρέπει να βάζουμε end με το τέλος του κάθε if.

```
    signum = -1;
}
else if (number > 0) {
        signum = 1;
     }
     else{
        signum = 0;
     }
```

Εντολή ελέγχου Πολλαπλή επιλογή - Switch

Χρησιμοποιείται όταν θέλουμε να επιλέξουμε από ένα σύνολο εντολών, μία για εκτέλεση, ελέγχει αν μια παράσταση ταυτίζεται με μια τιμή από το σύνολο των σταθερών παραστάσεων και διακλαδώνεται ανάλογα.

```
switch(παράσταση )
     case {σταθ. τιμή 1}
                  εντολές 1;
                  break;
     case {σταθ. τιμή 2}
                  εντολές 2;
                  break;

     case {σταθ. τιμή n}
                  εντολές n;
                  break;
     otherwise
                  εντολές
                  break;
```

```
end
```

ΠΑΡΑΔΕΙΓΜΑ:

```
input_num=input('DWSE ARI8MO:');
switch input_num
    case -1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
         disp('other value');
end
```

Παρατηρήσεις

- Η παράσταση ελέγχου πρέπει να παίρνει διακριτές τιμές (int, char), άρα δεν μπορεί να είναι αριθμός double ή float ή real
- Αν έχουμε λίστες τιμών οι οποίες εκτελούν όλες μια εντολή, τις γράφουμε case σταθ. τιμή 1 : case σταθ. τιμή 2 : . .. case σταθ. τιμή n: εντολή n; break;
- Μετά την εκτέλεση μιας case το πρόγραμμα συνεχίζει στην αμέσως επόμενη case που επαληθεύεται εκτός και βάλουμε διακοπή της switch με break (που μας βγάζει ένα έλεγχο ή ένα loop από εκεί που μπορεί να βρισκόμαστε) εδώ μπαίνει στο τέλος κάθε case για να μας βγάλει από την switch. Άλλες εντολές που προκαλούν διακοπεί είναι η return (που επιστέφει και τιμή π.χ. return τιμή), η goto (που μας πηγαίνει εκεί που ορίζει το label), η continue (που μας βγάζει

μέσα από όλους τους έλεγχου και τα loop's που μπορεί να βρισκόμαστε) και η exit(1) κάνει βίαιο τερματισμό του προγράμματος. Πάντως σαν γενική μας στρατηγική θα πρέπει να είναι να αποφεύγετε το goto που αλλάζει απότομα την ροή του προγράμματος.

- Η default είναι προαιρετική και εκτελείτε μόνο αν δεν ταιριάζει καμία από τις case. Ακόμα μπορεί να μπει πριν, ανάμεσα ή μετά τις case.

4.0.5 Εντολές Επανάληψης

Συχνά χρειάζεται κάποιοι υπολογισμοί να επαναληφθούν για διαφορετικές τιμές μεταβλητών ή όσο ισχύει κάποια συνθήκη. Τέτοιες επαναλήψεις υλοποιούνται με τη βοήθεια των εντολών επανάληψης.

Η Εντολή Επανάληψης for

Η εντολή επανάληψης **for** χρησιμοποιείται όταν γνωρίζουμε ακριβώς πόσες φορές πρέπει να επαναληφθεί μία εντολή ή τμήμα προγράμματος.
Η εντολή for είναι δομημένη εντολή και συμφωνεί με την έκφραση της καθημερινής μας διαλέκτου όταν λεμε:

```
"Για" μτβ="Αρχική τιμή" : "με βήμα" : "έως Τελική τιμή"
       "εντολές (αν είναι εντός ορίων η μτβ)"
"Τέλος Επαναλήψεων"
```

Σύνταξη:

```
for μτβ=εκφρ1:εκφρ3:εκφρ2
```

```
        εντολές (αν είναι εντός ορίων η μτβ)
end
```

Παρατηρήσεις

- Η μεταβλητή ελέγχου πρέπει να παίρνει διακριτές τιμές (όχι πραγματικές τιμές).
- Η αρχική και η τελική τιμή πρέπει να είναι του ίδιου τύπου με τη μεταβλητή ελέγχου.
- Η αρχική και η τελική τιμή υπολογίζονται μόλις συναντούμε η εντολή for και μια μόνο φορά.
- Η τιμή της μεταβλητής ελέγχου δεν πρέπει να μεταβάλλεται μέσα στις εντολές που επαναλαμβάνονται. Αυξάνεται ή μειώνεται αυτόματα σε κάθε βήμα κατά 1, αν θέλουμε να βάλουμε κάποιο άλλο βήμα.
- Η εντολή που επαναλαμβάνεται μπορεί να είναι απλή ή σύνθετη-πολλές μαζί, ή να περιλαμβάνει if και case, κ.λπ.

Συνήθως η εκφρ1 αντιστοιχεί σε κάποια αρχική τιμή, η εκφρ2 σε κάποια τελική τιμή, ενώ η εκφρ3 είναι το βήμα. Η εκφρ3 είναι προαιρετική. Όταν παραλείπεται το βήμα είναι 1.

Ο αριθμός των επαναλήψεων κ που θα εκτελεσθούν τελικά, δίνεται από τον τύπο:

$$\kappa = \max\left(int\left(\frac{\varepsilon\kappa\varphi\rho 2 - \varepsilon\kappa\varphi\rho 1 + \varepsilon\kappa\varphi\rho 3}{\varepsilon\kappa\varphi\rho 3}\right), 0\right)$$

Η τιμή της μτβ ελέγχεται αποκλειστικά από την εντολή for και δεν πρέπει να μεταβληθεί μέσα στην επανάληψη. Προτού αρχίσουν οι επαναλήψεις: Υπολογίζονται οι τιμές των εκφρ1, εκφρ2 και κ . Η μτβ αρχικοποιείται από εκφρ1 . Ελέγχεται αν $\kappa > 0$ οπότε και εκτελούνται οι εντολές του for. Όταν κάθε επανάληψη φτάνει στο

τέλος “end”: Η μτβ αυξάνει κατά το βήμα (εκφρ3). Η τιμή του κ ελαττώνεται κατά 1. Ελέγχεται αν $\kappa > 0$ οπότε και συνεχίζονται οι επαναλήψεις, διαφορετικά η επαναληπτική διαδικασία ολοκληρώνεται και εκτελείται η επόμενη εντολή.

Παραδείγματα:

Βρίσκει το άθροισμα 0+1+2, ξεκινώντας αρχικοποίει το sum=0 και το i=0, και εκτελεί την εντολή sum=sum+i, στη αυξάνει το i κατά 1 και κάνει τον έλεγχο $i <= 2$ που αν τον βρει αληθή εκτελεί την εντολή...

```
sum=0;
for i=0:2
 sum=sum+i;
end
sum
```

Τυπώνει στην οθόνη τους αριθμούς -1,1,-1

```
for KAPPA=3:3:9
    RES = cos(KAPPA*pi);
    disp(RES) %ή sprintf("%f\n",RES);
end
```

Το αντίστοιχο πρόγραμμα στη C είναι:

```
pi = 3.141593;
for (kappa = 3; kappa <= 9; kappa + = 3) {
   res = cos(kappa*pi);
   printf("%f\n",res);
}
```

Λίγα παραδείγματα ακόμα....

```
x = [];
for i = 1:n
   x=[x,i^2]
end
```

Στην πιο γενική της μορφή:

```
s = 0;              % Gia 2D pinaka A:
for c = A           % Diatrexei tic sthles tou A
   s = s + sum(c)
end
```

Η Εντολή Επανάληψης "while"

Άλλη εντολή επανάληψης εκτός από την for είναι η **while** που χρησιμοποιούνται όταν δεν γνωρίζουμε εκ των προτέρων τον ακριβή αριθμό των επαναλήψεων (σε αντίθεση με την for). Ακόμα η εντολές αυτές έχουν τη δυνατότητα να παίρνουν οι μεταβλητές ελέγχου πραγματικές τιμές (βέβαια τα βήματα- επανάληψης που κάνουν είναι πάντα διακριτές) αφού ο τερματισμός τους εξαρτάτε από την ικανοποίηση μιας συνθήκης. Αυτό είναι ένα πλεονέκτημα αφού συχνά η συνέχιση της επανάληψης εξαρτάται από την ικανοποίηση κάποιων συνθηκών.

Το συντακτικός τους και αυτόν τον εντολών, δεν είναι ιδιαίτερα δύσκολο αφού όπως έχουμε αναφέρει επανειλημμένα έχουμε να κάνουμε με μια δομημένη γλώσσα, δηλαδή έχουμε:

```
"Οσο ισχύει" ("Λογική Συνθήκη")
             "εντολές (αληθής)"
"Τέλος Επαναλήψεων"
```

Στο MATLAB χρησιμοποιείται και η εντολή **while**:

```
while(λογική έκφραση)
      εντολές (αληθής)
end
```

Παρατηρήσεις

- Στο while η λογική συνθήκη εξετάζεται στην αρχή και η (σύνθετη) εντολή εκτελείται μόνο αν ικανοποιείται η συνθήκη.
- Μέσα στις εντολές που επαναλαμβάνονται, θα πρέπει να μεταβάλλονται οι τιμές των μεταβλητών που συμμετέχουν στη λογική παράσταση, ώστε να εξασφαλιστεί ο τερματισμός της επανάληψης.

Παράδειγμα 1:

Να γραφεί πρόγραμμα MATLAB που να υπολογίζει το άθροισμα και το πλήθος των όρων της σειράς $1 + 1/2 + 1/3 + 1/4 + ...$ μέχρι τον όρο που είναι μικρότερος από $\varepsilon = 0.005$

```
EPSLON = 5.E-3;
OROS = 1.0;
COUNT = 1;
SUM = 0.0;
while(OROS >= EPSLON)
      SUM = SUM + OROS;
      COUNT = COUNT + 1;
      OROS = 1./COUNT;
end
disp(΄Αθροισμα = '),disp(SUM)
disp('Πλήθος όρων ='),disp(COUNT-1)
```

Κεφαλαιο 5

Πινακεσ - Διανυσματα

Ένας πίνακας (array) είναι μία δομημένη μεταβλητή που χρησιμεύει για την καταχώρηση δεδομένων. Οι πίνακες δηλώνονται όπως και οι συνήθεις μεταβλητές. Δεν πρέπει να ξεχνάμε ότι το MATLAB όλες τις μεταβλητές τις αντιστοιχεί σε πίνακες, ακόμα και αυτές που είναι διάστασης 1 τις αντιστοιχεί σε πίνακα 1×1.
Δηλαδή γράφουμε

```
πιν=[a1,a2,...,an;
     b1,b2,...,bn;
...
     z1,z2,...,zn]
```

για να δηλώσουμε ένα διάνυσμα γράφουμε αντίστοιχα

```
διαν=[a1,a2,...,an]
```

όπου πιν είναι το όνομα του πίνακα και διαν είναι το όνομα του διανύσματος αντίστοιχα.

ΠΡΟΣΠΕΛΑΣΗ ΣΤΟΙΧΕΙΩΝ:
Για να καλέσουμε το στοιχείο i ενός διανύσματος γράφουμε διαν(i). Αντίστοιχα για

να καλέσουμε το στοιχείο που βρίσκεται στη θέση (i,j) ενός πίνακα γράφουμε πιν(i,j). Οι δείκτες είναι θετικές ακέραιες σταθερές ή μεταβλητές.
Αν θέλουμε να πάρουμε ολόκληρη τη γραμμή i γράφουμε πιν(i,:), ακόμα μπορούμε και ζητήσουμε και μόνο κάποια στοιχεία μιας γραμμής π.χ.πιν(i,[1,3]) ή πιν(:,[1,3]), ενώ αν θέλουμε τη στήλη j γράφουμε πιν(:,j), ακόμα μπορούμε και ζητήσουμε και μόνο κάποια στοιχεία μιας στήλης π.χ.πιν([1,3],j) ή πιν([1,3],j).
Προφανώς αν δεν θέλουμε να υπάρξει πρόβλημα τα i,j θα πρέπει να είναι μέσα στα όρια του πίνακα-διανύσματος. Αν γράψουμε πιν(1,2) θα μας δώσει το a2, όμοια το διαν(1) θα μας δώσει το a1. Αν θέλουμε να αντικαταστήσουμε a2 με το k1 γράφουμε πιν(1,2)=k1 όμοια και για διανύσματα. Αν θέλουμε να δημιουργήσουμε ένα κενό διάνυσμα ή πίνακα γράφουμε:

```
b=zeros(n,m)
```

Τέλος για να διαγράψουμε ένα διάνυσμα ή πίνακα χρησιμοποιούμε την εντολή **clear** π.χ. clear πιν

Παραδείγματα-Εισαγωγή πινάκων

1. Αμεσα από το χρήστη:

   ```
   A = [1 2 3; 4 5 6; 7 8 9]
   ```

   ή ισοδύναμα

   ```
   A = [ 1 2 3
           4 5 6
         7 8 9 ]
   ```

2. Από συναρτήσεις του Matlab:

   ```
   b = rand(1,5)
   ```

δημιουργεί τυχαίο πίνακα 1×5 (διάνυσμα) με στοιχεία $\in [0,1]$.

3. Πίνακές πάνω από δύο διαστάσεις

```
A(:,:,1,2) = [1 2 3; 4 5 6; 7 8 9];
A(:,:,2,2) = [9 8 7; 6 5 4; 3 2 1];
A(:,:,3,2) = [1 0 1; 1 1 0; 0 1 1];

B = randn(4,3,2)

A = cat(3,[9 2; 6 5],[7 1; 8 4])
```

5.0.6 Πράξεις πινάκων

Οι σημαντικότεροι αριθμητικοί τελεστές είναι η πρόσθεση +, η αφαίρεση -, ο πολλαπλασιασμός *, η διαίρεση /, και η ύψωση σε δύναμη ^.

+	Πρόσθεση	`help arith`
-	Αφαίρεση	`help arith`
*	Πολ/σμός	`help arith`
^	Υψωση σε δύναμη	`help arith`
'	Ανάστροφος πίνακας	`help punct`
\	**Αριστερή** διαίρεση	`help slash`
/	**Δεξιά** διαίρεση	`help slash`
.*	Πολ/σμός στοιχείο με στοιχείο	`help arith`
.^	Υψωση σε δύναμη στοιχείο με στοιχείο	`help arith`
.\	**Αριστερή** διαίρεση στοιχείο με στοιχείο	`help slash`
./	**Δεξιά** διαίρεση στοιχείο με στοιχείο	`help slash`

Π.χ. A+B-C, A*(-B), A*B/C, $Z\hat{X}$,

Η προτεραιότητα είναι:

1. Υπολογισμός παρενθέσεων (από τις εσωτερικές προς τις εξωτερικές).

2. Ύψωση σε δύναμη.

3. Πολλαπλασιασμός και διαίρεση.

4. Πρόσθεση και αφαίρεση.

Παράδειγμα :
Να γραφεί ένα πρόγραμμα MATLAB που να διαβάζει δύο διανύσματα του $\mathbf{IR^3}$ και να υπολογίζει το εσωτερικό τους γινόμενο.

```
sum([a1,a2,a3].*[b1,b2,b3])
```

ή

```
[a1,a2,a3]*[b1,b2,b3]'
```

Παρατηρήσεις

- Ισχύουν και για βαθμωτά μεγέθη (= πίνακες 1×1)
- Ασυμβατότητα διαστάσεων $\Longrightarrow \mathit{ΛΑΘΟΣ}$.

  ΕΞΑΙΡΕΣΗ: πράξεις μεταξύ πινάκων και αριθμών, οπότε η πράξη εκτελείται μεταξύ του αριθμού και *κάθε* στοιχείου του πίνακα.

- Για τις διαιρέσεις: Αν ο πίνακας A είναι αντιστρέψιμος τότε:
  - x = A\b είναι η λύση του συστήματος: A*x = b.
  - x = b/A είναι η λύση του συστήματος: x*A = b.
- Οι πράξεις: .* .^ ./ .\\ εκτελούνται μεταξύ των **στοιχείων** των πινάκων. Π.χ. [1 2; 3 4].^2 δίνει [1 4; 9 16].

Λογικές Πράξεις-Παρατηρήσεις

Ακόμα μεταξύ των πινάκων μπορούμε να έχουμε και λογικές εκφράσεις που παίρνουν τις τιμές: TRUE ή FALSE Σχηματίζονται συνήθως από συγκρίσεις (τελεστές συσχέτισης) και συνδυασμούς τους (λογικοί τελεστές).

Στον πίνακα που ακολουθεί δίνουμε την μορφή των τελεστών συσχέτισης στο MATLAB

MATLAB	==	∼=	>	>=	<	<=

- Τιμές: Αληθής $\longrightarrow$ 1, Ψευδής $\longrightarrow$ 0.

- Οι λογικές σχέσεις μεταξύ πινάκων, εκτελούνται μεταξύ των στοιχείων τους και δίνουν πίνακα με 1 ή 0 στις αντίστοιχες θέσεις. Π.χ. L = [1 2; 3 4] > [1 0; 10 0] δίνει την τιμή [0 1; 0 1] στον L

- Οι while και if ερμηνεύουν μια σχέση μεταξύ πινάκων σαν αληθή όταν ο παραγόμενος πίνακας έχει όλα τα στοιχεία του = 1. Π.χ. για το προηγούμενο L, η

```
if L,
    disp('MATLAB');
end
```

ΔΕΝ ΕΚΤΕΛΕΙΤΑΙ

5.0.7 Κατασκευή πινάκων

Μερικές συναρτήσεις (για σύνταξη: help *onoma*)

eye(n)	δημιουργία $n \times n$ μοναδιαίου πίνακα.
zeros	Μηδενικός πίνακας
ones	Πίνακας με στοιχεία μονάδες
triu, **tril**	Άνω, κάτω τριγωνικός πίνακας
rand	Πίνακας με "τυχαία" στοιχεία
magic	Μαγικά τετράγωνα.
hilb,**invhilb**	Δίνει τον πίνακα Hilbert και τον αντιστροφότου.
diag(list)	δίνει το διαγώνιο πίνακα με στοιχεία με διαγωνίου αυτά που περιέχει η list και όλα τα άλλα είναι 0.
size(πιν)	δίνει τη λίστα των διαστάσεων.
length(expr)	δίνει τον αριθμό των στοιχείων της έκφρασης.
rank(πιν)	δίνει το βαθμός του πίνακα.
norm(expr, p)	βρίσκει το αποτέλεσμα της p-norm όπου expr μπορεί να είναι οποιαδήποτε έκφραση όπως πίνακας-λίστα-μιγαδικός... και το p μπορεί να είναι 1,2,...,inf, αν δεν δώσουμε τιμή στο p τότε είναι 2.

Παρατηρήσεις, παραδείγματα

- zeros(m,n), m×n μηδενικός πίνακας, αλλά zeros(n) τετραγωνικός μηδενικός πίνακας.
- Για x διάνυσμα, diag(x) πίνακας με x στη διαγώνιο. Για πίνακα A, diag(A) διάνυσμα με τα διαγώνια στοιχεία του A.
  ΕΡΩΤΗΣΗ: Τι υπολογίζει η diag(diag(A))?
- Παραγωγή πινάκων από πίνακες: αν A είναι 3×3 τότε:

  ```
  B = [A, zeros(3,2); ones(2,3), eye(2,2)]
  ```

  δίνει πίνακα 5×5.

Υποπίνακες

- Οι εκφράσεις: 1:5 και 0.2:0.2:1.2 είναι στην ουσία τα διανύσματα: [1 2 3 4 5] και [0.2 0.4 0.6 0.8 1.0 1.2]
- A(1:4,3) διάνυσμα με τα 4 πρώτα στοιχεία της 3ης στήλης του A.
- A(:,3) είναι ή 3η στήλη του A.
- A(:,[2,4]) Οι στήλες 2 και 4 του A.
- A(:,[2 4 5]) = B(:,1:3) Αντικαθιστά τις στήλες 2, 4 και 5 του A με τις στήλες 1, 2, 3 του B.

5.0.8 Συναρτήσεις

Βαθμωτές Συναρτήσεις

Ενεργούν σε βαθμωτά μεγέθη (δηλ. για πίνακα: σε κάθε στοιχείο του και παράγουν πίνακα με τα αποτελέσματα, ίδιας διαστασης με τον αρχικό)

sin	asin	exp	abs	round
cos	acos	log	sqrt	floor
tan	atan	rem	sign	ceil

ΠΑΡΑΔΕΙΓΜΑ
sin([pi pi/2; 0 pi/4]) $\longrightarrow$ [0.0 1.0; 0.0 0.707]

Διανυσματικές συναρτήσεις

Ενεργούν σε διανύσματα (δηλ. για πίνακα: σε κάθε στήλη του και παράγουν πίνακα γραμμή με τα αποτελέσματα κάθε στήλης).

max	sum	median	any
min	prod	mean	all
sort		std	

ΠΑΡΑΔΕΙΓΜΑΤΑ

max([1 2 3]) $\longrightarrow$ 3

max([1; 2; 3]) $\longrightarrow$ 3

max([1 2 3; 3 2 1]) $\longrightarrow$ [3 2 3]

max(max([1 2 3; 3 2 1])) $\longrightarrow$ 3

```
x = []; for i = 1:n, x=[x,i^2], end
```

ή

```
x = []; for i = 1:n
   x=[x,i^2]
end
```

Στην πιο γενική της μορφή:

```
s = 0;              % Gia 2D pinaka A:
for c = A           % Diatrexei tic sthles tou A
   s = s + sum(c)
end
```

Συναρτήσεις πινάκων

eig	Ιδιοδιανύσματα, ιδιοτιμές
chol	Παραγοντοποίηση **Choleski**
lu	Παραγοντοποίηση **Gauss**
qr	Παραγοντοποίηση **QR**
expm	e^A (συγκρ. exp)
sqrtm	Τετραγωνική ρίζα πίνακα (συγκρ. sqrt)
det	Ορίζουσα

Παραδείγματα-Πινάκων

Για: A=rand(3,5), B οποιοσδήποτε 3×5 πίνακας, x=rand(5,1), y ένα διάνυσμα 5×1 και C=[1 2 3; 4 5 6; 7 8 9]:

1. Πραξεις:

   ```
   A+B,  A+2,  C^2,  C.^2,  C.^C
   ```

2. Δημιουργία πινάκων:

   ```
   diag(C), diag(y), diag(diag(C)),
   triu(C) eye(3)-2, -eye(3)
   ```

3. Σύνθετοι πίνακες:

   ```
   D = [A; zeros(2,3) ones(2)]
   ```

4. For:

   ```
   s = 0; for i = 1:5, s = s + x(i); end; s, sum(x)
   ```

5. Λογικές σχέσεις:

   ```
   D = triu(C),   C == D
   ```

6. Συναρτήσεις:

   ```
   sin(B), max(y), rank(B), det(C), size(x) help eig eig(C)
   [V,L]=eig(C)
   ```

7. Υποπίνακες:

```
B(1:2,3), B(:,2), B(2,:),
B(:,[1 5]) M = B, M(:,1:3)=eye(3)
```

Παράδειγμα: δημιουργία τριδιαγώνιου πίνακα

Η εντολή:

```
10*eye(3) + diag(ones(2,1),1) + diag(ones(2,1),-1)
```

δίνει τον τριδιαγώνιο πίνακα:

$$\begin{pmatrix} 10 & 1 & 0 \\ 1 & 10 & 1 \\ 0 & 1 & 10 \end{pmatrix} = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

5.0.9 Επίλυση γραμμικών συστημάτων

Αν A πίνακας n×n και b, x διανύσματα στήλες, τότε:

```
x = A\b
```

υπολογίζει τη λύση του συστήματος $Ax = b$ με κάποια μορφή απαλοιφής Gauss.

Αλγόριθμοι

1. A συμμετρικός: παραγοντοποίηση Choleski.

2. A μη συμμετρικός: απαλοιφή Gauss με μερική οδήγηση.

Στη γενικευμένη μορφή του ο τελεστής
επιλύει συστήματα πινάκων $AX = B$ με A διαστάσεων $m \times n$. (Οταν $m \neq n$ γίνεται επίλυση ελαχίστων τετραγώνων).

Αντιστροφή πίνακα

Αντίστροφος του *τετραγωνικού* πίνακα X:

```
Y = inv(X)
```

- Στην πράξη η αντιστροφή χρησιμοποιείται *πολύ σπάνια*.
- Η συνηθέστερη κατάχρηση της για επίλυση γραμμικών συστημάτων. Η απαλοιφή Gauss (τελεστής \) υπερέχει σε *ακρίβεια* και *υπολογιστικό χρόνο*.

5.0.10 Νόρμες και δείκτης κατάστασης

norm(X), norm(X,2) υπολογίζει: $\|\cdot\|_2$

norm(X,1) υπολογίζει: $\|\cdot\|_1$

norm(X,inf), norm(X,Inf) υπολογίζει: $\|\cdot\|_\infty$

Για το δείκτη κατάστασης του πίνακα X στην *Ευκλείδεια* νόρμα:

```
kappa = cond(X)
```

5.0.11 Πίνακες Hilbert

Πίνακας Hilbert τάξης n:

```
H = hilb(n)
```

Κλασσικό παράδειγμα πινάκων με *υψηλό* δείκτη κατάστασης.

Ο αντίστροφος πίνακα Hilbert υπολογίζεται με:

```
H = invhilb(n)
```

ακριβώς (χωρίς σφάλματα περικοπής) για n$\leq$ 13, και *προσεγγιστικά* για μεγαλύτερα n.

Η invhilb(n) σε σχέση με inv(hilb(n)) αντιμετωπίζει καλύτερα τα σφάλματα περικοπής:

- στην παράσταση των hilb(n) και invhilb(n)
- στη διαδικασία αντιστροφής.

Παράδειγμα :
Να γραφεί πρόγραμμα που να διαβάζει ένα πραγματικό πίνακα 4×4 και να τυπώνει τα αθροίσματα στηλών, γραμμών και όλων των στοιχείων.

Κεφαλαιο 6

Συναρτησεις - Υποπρογραμματα

Το MATLAB έχει μια βιβλιοθήκη από "εσωτερικές" συναρτήσεις που είναι δυνατόν να χρησιμοποιηθούν σε κάθε πρόγραμμα. Παράλληλα με τις "εσωτερικές" συναρτήσεις μπορούμε να φτιάξουμε και εμείς τις δικές μας συναρτήσεις και υποπρογράμματα να τα χρησιμοποιήσουμε μια και τα προβλήματα που θα έχουμε να λύσουμε στη συνέχεια θα είναι πιο δύσκολα.
Πριν ξεκινίσουμε να παρουσιαζουμε τις Συναρτήσεις και Υποπρογράμματα θα πρέπει να τονίσουμε ότι:

- Τα ονόματα τους ακολουθούν του κανόνες που έχουμε πει και για τις μεταβλητές... προσοχή θυμίζουμε ότι στο MATLAB γίνεται διαχωρισμός πεζών-κεφαλαίων.

- Σώζουμε κάθε Συνάρτηση - Υποπρόγραμμα σε ξεχωριστώ αρχείο με επέκταση **.m**,.

- **Scripts** Αρχεία που περιέχουν αλληλουχία εντολών του Matlab. Π.χ. εντολές στο αρχείο mycommand.m εκτελείται με mycommand.
  Παράδειγμα

```
% Σώζουμε με το όνομα kyklos.m
r=input('Δώσε την ακτίνα του κύκλου:')
S=2*pi*r;
E=pi*r^2;
disp( Ή περίμετρος του κύκλου είναι:'),disp(S);
sprintf('Το εμβαδό του κύκλου είναι: %f',E) %όχι ερωτηματικό
```

Για να εκτελέσουμε το παραπάνω Scripts γράφουμε στη γραμμή εντολών
$>> kyklos$

6.0.12 Συναρτήσεις

- Αρχικά θα επαναλάβουμε τον τρόπο δήλωσης μια συνάρτησης και θα κάνουμε το διαχωρισμό ανάμεσα στις συναρτήσεις και τις διαδικασίες. Θα μπορούσαμε να δώσουμε δυο πρόχειρους ορισμούς παρόλο που η διαχωριστική γραμμή δεν είναι και τόσο ευδιάκριτη γι αυτό θα επικεντρωθούμε σε ένα ορισμό που τονίζει περισσότερο τη διαφορά τους: συναρτήσεις θα λεμε τα υποπρογράμματα που παίρνουν μεταβλητές με κάποια δεδομένα και μας επιστρέφουν αποτέλεσμα.
  Για τις συνάρτησης ο τρόπος που τις δηλώνουμε μοιάζει με των μεταβλητών, δηλαδή

$$\textbf{function}\ [\mu\tau\beta_1, ..., \mu\tau\beta_n] = \ ''ονομα\ συναρτησης''(\mu\tau\beta_1, ..., \mu\tau\beta_\kappa);$$

  Χαρακτηριστικό γνώρισμα των συναρτήσεων (αν και δεν είναι απαραίτητη στο MATLAB) είναι η εντολή

$$\boxed{\text{return}(\text{“τιμή που επιστρέφεται”});}$$

  μπαίνει συνήθως εκεί που θέλουμε να επιστρέψουμε το αποτέλεσμα και προκαλεί εκτός της επιστροφής του αποτελέσματα και τερματισμό της συναρτήσεις (δεν

εκτελούνται τυχόν εντολές που ακολουθούν).

Παράδειγμα

Συναρτήσεις Δυνατότητα δημιουργίας νέων συναρτήσεων. Π.χ. η νέα εντολή randint ορίζεται στο αρχείο randint.m:

```
function a = randint(m, n)
%RANDINT Randomly generated integral matrix.
%  RANDINT(M,N) M-by-N matrix.
%  Elements between 0 and 9
a = floor(10*rand(m,n));
```

Τι εμφανίζει η: help randint?

Παράδειγμα

```
% Σώσουμε στο αρχείο με όνομα avg.m
function mean = avg(x,n)
  mean = sum(x)/n;

% Σώσουμε στο αρχείο με όνομα f
function res= f(x)
  res= x^2 + 3.0*x ὐ 2.0;

% Σώσουμε στο αρχείο με όνομα DD.m
function res= DD(x,y)
  res=abs(x-y);
```

6.0.13 Υποπρογράμματα

Με τον όρο υποπρογράμματα στο MATLAB εννοούμε "μικρά" ανεξάρτητα προγράμματα που λειτουργούν βοηθητικά στο κυρίως πρόγραμμα.

Παράδειγμα:

Να γραφεί πρόγραμμα που να υπολογίζει την τιμή του γινομένου δυο δεδομένων συναρτήσεων F(x), G(x), στα σημεία $X_i,\ i = 0, 1, \ldots, 100$, ενός ομοιομόρφου διαμερισμού του διαστήματος [0,2]. Να τυπώνει σε ένα αρχείο με το όνομα result.dat τα αποτελέσματα.

```
% Σώσουμε στο αρχείο με όνομα F.m
function res=F(x)
  res=cos(x)+sin(x);

% Σώσουμε στο αρχείο με όνομα G.m
function res=G(x)
  res=x^2-3.0*x+2.;

% Σώσουμε στο αρχείο με όνομα File_res.m
function File_res(a,b,n)
  h=(b-a)/n;
  strm=fopen('c:\\result.txt','w');
  for i=1:n
      x=h*i;
      tmp=F(x)*G(x);
      fprintf(strm,'%f %f \n',x,tmp);
   end
   fclose(strm);
```

```
% Σώσουμε στο αρχείο με όνομα Plot_res.m
function Plot_res(a,b,n)
  h=(b-a)/n;
  x=a:h:b;
  plot(x,F(x).*G(x));
```

Παράδειγμα :

Να γραφεί πρόγραμμα που να Παράδειγμα :

Το παρακάτω πρόγραμμα διαβάζει τις πολικές συντεταγμένες ενός σημείου τις μετατρέπει σε καρτεσιανές και τις εκτυπώνει στην οθόνη. Για το σκοπό αυτό φτιαχτεί-χρησιμοποιηθεί η υπορουτίνα CONVER.

Κεφαλαιο 7

Εισοδος - Εξοδος Απο Αρχειο

Πολλές φορές ο όγκος των δεδομένων που πρέπει να διαβαστούν από ένα πρόγραμμα είναι μεγάλος, επομένως η πληκτρολόγηση τους κάθε φορά που εκτελείται το πρόγραμμα είναι κουραστική ή και πρακτικά αδύνατη, και φυσικά υπάρχει μεγάλη πιθανότητα να γίνει κάποιο λάθος. Επίσης συχνά θέλουμε να διατηρήσουμε τα αποτελέσματα ενός προγράμματος είτε για μελέτη, είτε για πιθανή χρήση τους ως δεδομένα εισόδου από κάποιο άλλο πρόγραμμα Στις περιπτώσεις αυτές χρησιμοποιούμε αρχεία για την είσοδο ή έξοδο των δεδομένων. Τα αρχεία εισόδου/εξόδου που θα χρησιμοποιήσουμε θα πρέπει να τα συνδέσουμε μέσα στο πρόγραμμα με κάποια μονάδα εισόδου/εξόδου ρεύμα-**stream** (βλ. **fopen**,**fscanf**, **fprintf**, **fclose**).

Ρεύμα είναι μια πηγή ή ένας προορισμός δεδομένων τα οποία μπορεί να σχετίζονται με ένα δίσκο ή άλλο περιφερειακό. Η βιβλιοθήκη υποστηρίζει ρεύματα κειμένου και δυαδικά, αν και σε μερικά συστήματα, ειδικά στο UNIX, αυτά είναι πανομοιότυπα. Ρεύμα κειμένου είναι μια ακολουθία γραμμών. Κάθε γραμμή έχει μηδέν ή περισσότερους χαρακτήρες, και τερματίζεται με $'\backslash n'$. Ένα περιβάλλον μπορεί να χρειαστεί να μετατρέψει ένα ρεύμα κειμένου προς ή από κάποια άλλη αναπαράσταση (όπως αυτήν που απεικονίζει το $'\backslash n'$ σαν επαναφορά κεφαλής και αλλαγή γραμμής). Δυαδικό ρεύμα είναι μια ακολουθία ανεπεξέργαστων byte που καταγράφουν εσωτερικά δεδομένα, κι

έχει την ιδιότητα ότι αν γραφτεί και μετά ξαναδιαβαστεί από το ίδιο σύστημα, θα είναι το ίδιο.

Ένα ρεύμα συνδέεται με ένα αρχείο ή συσκευή ανοίγοντας την. Η σύνδεση διακόπτεται με το κλείσιμο του ρεύματος. Το άνοιγμα ενός αρχείου επιστρέφει ένα δείκτη σε αντικείμενο τύπου FILE που καταγράφει οποιεσδήποτε πληροφορίες είναι απαραίτητες για τον έλεγχο του ρεύματος. Θα χρησιμοποιήσουμε τους όρους "δείκτης αρχείου" και "ρεύμα" εναλλακτικά, όταν δεν υπάρχει ασάφεια.
Όταν ένα πρόγραμμα αρχίζει να εκτελείται, τα ρεύματα stdin, stdout, και stderr είναι ήδη ανοικτά.

Άνοιγμα-Κλείσιμο Αρχείου, fopen-fclose

Η σύνδεση ενός αρχείου γίνεται με χρήση της εντολής **fopen**.
Σύνταξη:

```
FILE *fopen(const char *filename, const chαr *mode)
```

Η fopen ανοίγει το κατονομαζόμενο αρχείο, κι επιστρέφει ένα ρεύμα, ή NULL αν η προσπάθεια αποτύχει Ανάμεσα στις έγκυρες τιμές του mode είναι και οι

- "r" άνοιγμα αρχείου κειμένου για διάβασμα
- "w" δημιουργία αρχείου κειμένου για γράψιμο, και απόρριψη τυχόν προϋπαρχόντων περιεχομένων
- "a" προσθήκη' άνοιγμα ή δημιουργία αρχείου κειμένου για γράψιμο στο τέλος του αρχείου
- "r+" άνοιγμα αρχείου κειμένου για ενημέρωση (δηλ. γράψιμο και διάβασμα)

- "w+" δημιουργία αρχείου κειμένου για ενημέρωση' απόρριψη τυχόν προϋπαρχόντων περιεχομένων
- "a+" προσθήκη άνοιγμα ή δημιουργία αρχείου κειμένου για ενημέρωση, με γράψιμο στο τέλος

Η κατάσταση ενημέρωσης επιτρέπει γράψιμο και διάβασμα του ίδιου αρχείου. Αν η mode περιλαμβάνει ένα b μετά το πρώτο γράμμα, όπως "rb" ή "w+b", σημαίνει ότι πρόκειται για δυαδικό αρχείο. Τα ονόματα των αρχείων περιορίζονται σε FILENAME_MAX χαρακτήρες. Το πολύ FOPEN_MAX αρχεία μπορούν να είναι ανοικτά ταυτόχρονα.

Ένα αρχείο που συνδέθηκε στο πρόγραμμα με την εντολή fopen θα πρέπει να αποσυνδεθεί από την μονάδα με την εντολή fclose. Σύνταξη:

```
int fclose(FILE *stream);
```

Η fclose γράφει τυχόν δεδομένα που είναι στην περιοχή ενδιάμεσης αποθήκευσης και δεν έχουν γραφτεί στο ρεύμα stream, απορρίπτει τυχόν είσοδο που βρίσκεται στην περιοχή ενδιάμεσης αποθήκευσης της εισόδου που όμως δεν έχει διαβαστεί, ελευθερώνει τυχόν αυτόματα κατανεμημένη περιοχή ενδιάμεσης αποθήκευσης, και μετά κλείνει το ρεύμα. Επιστρέφει EOF αν έχουν συμβεί λάθη, διαφορετικά επιστρέφει μηδέν.

Παρατήρηση:
Δεν μπορείτε να συνδέσετε το ίδιο αρχείο σε δύο μονάδες, αν δεν το έχετε πρώτα αποσυνδέσει με την εντολή fclose. Επίσης, δεν μπορείτε να συνδέσετε δύο αρχεία στην ίδια μονάδα.

Φορμαρισμένη Έξοδος

Οι συναρτήσεις **fprintf** παρέχουν μετατροπές φόρμας της εξόδου.

```
int fprintf(FILE *strenm, const char *format, . . .)
```

Η fprintf μετατρέπει, και γράφει την έξοδο στο stream, κάτω από τον έλεγχο της format. Επιστρεφόμενη τιμή είναι ο αριθμός των χαρακτήρων που γράφονται, ή αρνητική τιμή σε περίπτωση λάθους.
Το αλφαριθμητικό φόρμας περιέχει δύο τύπους αντικειμένων: κοινούς χαρακτήρες, που αντιγράφονται στο ρεύμα εξόδου, και προδιαγραφές μετατροπών, που καθεμιά προκαλεί τη μετατροπή και εμφάνιση του επόμενου ορίσματος της fprintf. Κάθε προδιαγραφή μετατροπής αρχίζει με το χαρακτήρα % και τελειώνει με ένα χαρακτήρα μετατροπής. Ανάμεσα στο % και στο χαρακτήρα μετατροπής μπορεί να υπάρχουν, με τη σειρά:

- Σημαίες (με οποιαδήποτε σειρά) που τροποποιούν την προδιαγραφή:

  κενό διάστημα: αν ο πρώτος χαρακτήρας δεν είναι πρόσημο, θα μπαίνει μπροστά του ένα κενό διάστημα.

  - : που καθορίζει αριστερή στοίχιση του μετατρεπόμενου ορίσματος στο πεδίο του.

  + : που καθορίζει ότι ο αριθμός θα εμφανίζεται πάντα με πρόσημο.

  0 : για αριθμητικές μετατροπές, καθορίζει συμπλήρωση του πλάτους του πεδίον με μηδενικά στην αρχή.

  # : που καθαρίζει μια εναλλακτική μορφή εξόδου. Για το o, το πρώτο ψηφίο θα είναι μηδέν. Για x ή X, μπροστά σε μη-μηδενικό αποτέλεσμα θα μπαίνει 0x ή 0X. Για e, E, f, g, και G, η έξοδος θα έχει πάντα υποδιαστολή. Για g και G, τα μη-σημαντικά μηδενικά δεν αφαιρούνται από το τέλος.

- Ένας αριθμός που καθορίζει το ελάχιστο πλάτος πεδίου. Το μετατρεπόμενο

όρισμα θα εμφανιστεί σε ένα πεδίο με τουλάχιστον αυτό το πλάτος. Αν το μετατρεπόμενο όρισμα έχει λιγότερους χαρακτήρες από το πλάτος του πεδίου, θα συμπληρωθεί με κενό στα αριστερά (η στα δεξιά, αν έχει οριστεί αριστερή στίχοι) για να συμπληρωθεί το πλάτος του πεδίου. Ο χαρακτήρας συμπλήρωσης κανονικά είναι το κενό διάστημα, αν όμως υπάρχει σημαία συμπλήρωσης με μηδέν, είναι ο 0.

- Μια τελεία, που χωρίζει το πλάτος του πεδίου από την ακρίβεια.

- Ένας αριθμός, η ακρίβεια, που καθορίζει το μέγιστο αριθμό χαρακτήρων που θα εμφανιστούν από ένα αλφαριθμητικό, ή τον αριθμό των ψηφίων που θα εμφανιστούν μετά από την υποδιαστολή για μετατροπές e, E, f, ή τον αριθμό των σημαντικών ψηφίων για μετατροπές g ή G, ή τον ελάχιστο αριθμό ψηφίων που θα εμφανιστούν για έναν ακέραιο (στην αρχή του θα προστεθούν μηδενικά για να συμπληρώσουν το αναγκαίο πλάτος πεδίου).

- Ένα χαρακτήρα h,l (το γραμμή ελ) ή L. Το "h" υποδηλώνει ότι το αντίστοιχο όρισμα θα εμφανιστεί σαν short ή unsigned short. Το "l" υποδηλώνει ότι το όρισμα είναι long ή unsigned long. Το "L" υποδηλώνει ότι το όρισμα είναι long double.

Το πλάτος, η ακρίβεια, ή και τα δύο μπορούν να καθορίζονται σαν *, οπότε η τιμή υπολογίζεται με τη μετατροπή του επομένου ορίσματος (ή ορισμάτων), που πρέπει να είναι int.
Οι χαρακτήρες μετατροπής και η σημασία τους δίνονται στον παρακάτω πίνακα. Αν ο χαρακτήρας μετά το % δεν είναι χαρακτήρας μετατροπής, η συμπεριφορά είναι απροσδιόριστη.

Πίνακας Ορισμάτων της fprintf.

ΧΑΡΑΚΤΗ-ΡΑΣ	ΤΥΠΟΣ ΟΡΙΣΜΑΤΟΣ ΕΜΦΑΝΙΖΕΤΑΙ ΣΑΝ
d, i	int προσημασμένος δεκαδικός συμβολισμός.
o	int απρόσημός οκταδικός συμβολισμός (χωρίs μηδέν μπροστά).
x,X	int απρόσιμός δεκαεξαδικός συμβολισμός (χωρίς 0x ή0X μπροστά), που χρησιμοποιεί τα abcdef για 0x ή τα ABCDEF για 0X.
u	int απρόσιμος δεκαδικός συμβολισμός.
c	char ένας χαρακτήρας, μετά από μετατροπεί σε unsigned chαr.
s	char* -εμφανίζονται χαρακτήρες του αλφαριθμητικού μέχρι τον $'\backslash 0'$ ή τόσοι χαρακτήρες όσους ορίζει η ακρίβεια.
f	double δεκαδικός συμβολισμός της μορφής [-]m.ddddddd όπου ο αριθμός των d καθορίζεται από την ακρίβεια η πρακαθορι-σμένη ακρίβεια είναι 6 αν οριστεί ακρίβεια 0, δεν εμφανίζεται υποδιαστολή.
e,E	double δεκαδεκός συμβολισμός της μορφής [-]m.dddddde ± xx ή [-]m.ddddddE ± xx, όπου ο αριθμός των d καθορίζεται από την ακρίβεια

	η προκαθορισμένη ακρίβεια είναι 6 αν οριστεί
	ακρίβεια 0, δεν εμφανίζεται υποδιαστολή.
g, G	double χρησιμοποιείται %e ή %E
	αν ο εκθέτης είναι μικρότερος από -4
	ή μεγαλύτερος από ή ίσος με την ακρίβεια,
	διαφορετικά χρησιμοποιείται %f. Τα μηδενικά
	και η υποδιαστολή στο τέλος δεν εμφανίζονται.
p	void* εμφανίζει σαν δείκτη (η
	αναπαράσταση εξαρτάται από την υλοποίηση).
n	int* ο αριθμός των χαρακτήρων που
	έχουν γραφτεί μέχρι στιγμής απ' αυτήν την
	κλήση της printf γράφεται στο όρισμα.
	Δεν μετατρέπεται κανένα όρισμα..
%	δεν μετατρέπεται κανένα όρισμα
	εμφανίζεται ένα %.

int **sprintf**(char *s, const char *format, . . .)

Η sprintf είναι ισοδύναμη με την fprintf(stdout,. . .). με τη διαφορά ότι η έξοδος γράφεται στο αλφαριθμητικό s, που τερματίζεται με $'\backslash 0'$. Το s πρέπει να είναι αρκετά μεγάλο για να χωράει το αποτέλεσμα. Η επιστρεφόμενη μέτρηση δεν περιλαμβάνει το $'\backslash 0'$.

Φορμαρισμένη Είσοδος

Οι συναρτήσεις **fscanf** ασχολούνται με μετατροπές φορμαρισμένης εισόδου.

```
int fscanf(FILE *stream, const char *format, . . .)
```

Η scanf διαβάζει από το stream κάτω από τον έλεγχο της format, και αποδίδει μέσω των υπόλοιπων ορισμάτων τις τιμές που έχουν μετατραπεί. Καθένα από τα υπόλοιπα ορίσματα πρέπει να είναι δείκτης. Επιστρέφει όταν εξαντληθεί το format. Η fscanf επιστρέφει EOF στο τέλος του αρχείου ή αν συμβεί λάθος στη μετατροπή, αλλιώς επιστρέφει τον αριθμό των αντικειμένων της εισόδου που μετατράπηκαν και αποδόθηκαν σε μεταβλητές.

Το αλφαριθμητικό φόρμας συνήθως περιέχει προδιαγραφές μετατροπής, που χρησιμοποιούνται για να κατευθύνουν τον έλεγχο της ερμηνείας της εισόδου. Το αλφαριθμητικό φόρμας μπορεί να περιέχει:

- Κενά ή στηλογνώμονες, που αγνοούνται.
- Κοινούς χαρακτήρες (όχι τον %), που αναμένεται να ταιριάζουν με τον επόμενο μη-λευκό χαρακτήρα του ρεύματος εισόδου.
- Προδιαγραφές μετατροπής, που αποτελούνται από το χαρακτήρα %, έναν προαιρετικό χαρακτήρα * για την παρεμπόδιση της αντικατάστασης, έναν προαιρετικό αριθμό που καθορίζει το μέγιστο πλάτος με πεδίου, ένα προαιρετικό h, l ή L που δείχνει το πλάτος του προορισμού, κι ένα χαρακτήρα μετατροπής.

Μια προδιαγραφή μετατροπής καθορίζει τη μετατροπή του επόμενου πεδίον της εισόδου. Κανονικά το αποτέλεσμα τοποθετείται στη μεταβλητή που δείχνει το αντίστοιχο όρισμα. Ωστόσο, αν υποδηλώνεται παρεμπόδιση της αντικατάστασης με το χαρακτήρα *, όπως στο %*s, τότε το πεδίο εισόδου παρακάμπτεται και δεν γίνεται αντικατάσταση. Ένα πεδίο εισόδου ορίζεται σαν αλφαριθμητικό με μη-λευκούς χαρακτήρες. Εκτείνεται μέχρι τον επόμενο χαρακτήρα "λευκού" διαστήματος ή μέχρι να εξαντληθεί το πλάτος του πεδίον, εφόσον έχει καθοριστεί. Αυτό συνεπάγεται ότι η scanf θα διαβάζει πέρα από τα όρια της γραμμής για να βρει την είσοδό της, αφού οι αλλαγές γραμμής είναι λευκά διαστήματα. (Χαρακτήρες λευκών διαστημάτων είναι το κενό,

ο στηλογνώμονας, η αλλαγή γραμμής, η επαναφορά κεφαλής, ο κατακόρυφος στηλογνώμονας, και η αλλαγή σελίδας).

Ο χαρακτήρας μετατροπής υποδηλώνει την ερμηνεία του πεδίου εισόδου. Το αντίστοιχο όρισμα πρέπει να είναι δείκτης, Οι χαρακτήρες μετατροπής παρουσιάζονται στον παρακάτω πίνακα.

Οι χαρακτήρες μετατροπής d, i, o, u, και x μπορούν να έχουν μπροστά τους ένα h αν το όρισμα είναι δείκτης σε short αντί για int, ή ένα l (το λατινικό γράμμα ελ) αν το όρισμα είναι δείκτης σε long. Μπροστά από τους χαρακτήρες μετατροπής e, f, και g μπορεί να είναι ένα l, εφόσον στη λίστα ορισμάτων υπάρχει ένας δείκτης σε double κι όχι σε float, ή ένα L αν υπάρχει δείκτης σε long double.

ΠΙΝΑΚΑΣ ΜΕΤΑΤΡΟΠΕΣ ΤΗΣ fscanf

ΧΑΡΑΚΤΗ- ΡΑΣ	ΔΕΔΟΜΕΝΑ ΕΙΣΟΔΟΥ ΤΥΠΟΣ ΟΡΙΣΜΑΤΟΣ
d	δεκαδικός ακέραιος int *.
i	ακέραιος int*. Ο ακέραιος μπορεί να είναι στο οκταδικό (με 0 μπροστά) ή στο δεκαεξακό (με 0x ή 0X μπροστά).
o	οκταδικός ακέραιος (με ή χωρίς μηδέν μπροστά) int *.
u	απρόσημος ακέραιος του δεκαδικού unsigned int *.
x	δεκαεξαδικός ακέραιος (με ή χωρίς 0x ή 0X μπροστά) int *.
c	χαρακτήρες char *.Οι επόμενοι χαρακτήρες εισόδου(εξ ορισμού ένας) τοποθετούνται στον υποδεικνυόμενο πίνακα, μέχρι τον αριθμό που καθορίζει το πεδίο πλάτους (εξ ορισμού 1). Δεν προστίθεται $'\backslash 0'$. Η κανονική αγνόηση των λευκών κενών αναστέλλεται.

	Για να διαβάσετε τον επόμενο μη-λευκό χαρακτήρα, χρησιμοποιήστε το %s.
s	αλφαριθμητικό (χωρίς εισαγωγικά) -char *, που δείχνει σ' έναν πίνακα αρκετά μεγάλο για να χωράει το αλφαριθμητικό κι ένα τερματικό '\0' που θα προστεθεί.
e, f, g	αριθμός κινητής υποδιαστολής float *. Η τυπική φόρμα εισόδου για τους float είναι προαιρετικό πρόσημο, μια σειρά αριθμών που μπορεί να περιέχει και υποδιαστολή, κι ένα προαιρετικό πεδίο εκθέτη που περιέχει ένα E ή ένα e, ακολουθούμενο από έναν πιθανά προσημασμένο ακέραιο.
p	τιμή δείκτη που εμφανίζεται από την printf("%p")-void *.
n	γράφει στο όρισμα τον αριθμό των χαρακτήρων που έχουν διαβαστεί μέχρι εκείνη τη στιγμή από αυτήν την κλήση int *. Δεν διαβάζεται είσοδος. Ο μετρητής αντικειμένων που έχουν μετατραπεί δεν αυξάνεται.
[...]	ταυτίζεται με τη μεγαλύτερη μη κενή σειρά χαρακτήρων εισόδου από το σύνολο που περιέχεται ανάμεσα στις αγκύλες char *. Προστίθεται ένα '\0'. Με το []...] περιλαμβάνεται και το] στο σύνολο.
[^...]	ταυτίζεται με τη μεγαλύτερη μη κενή

σειρά χαρακτήρων εισόδου που δεν
ανήκουν στο σύνολο που περιέχεται
ανάμεσα στις αγκύλες char*. Προστίθεται
ένα $'\backslash 0'$. Με το [^...]
περιλαμβάνεται και το] στο σύνολο.

% κυριολεκτικό % -δεν γίνεται αντικατάσταση.

Άλλες Συναρτήσεις Εισόδου και Εξόδου

char ***fgetl**(FILE *stream)

Η fgetl διαβάζει ως το χαρακτήρα νέας γραμμής. Ο χαρακτήρας νέας γραμμής περιλαμβάνεται στον πίνακα, που τερματίζεται με $'\backslash 0'$. Η fgetl επιστρέφει s, ή NULL αν συναντηθεί τέλος αρχείου ή λάθος.

char ***fgets**(char *s, int n, FILE *stream)

Η fgets διαβάζει το πολύ τους επόμενους n-1 χαρακτήρες για τον πίνακα s, σταματώντας αν συναντήσει χαρακτήρα νέας γραμμής. Ο χαρακτήρας νέας γραμμής περιλαμβάνεται στον πίνακα, που τερματίζεται με $'\backslash 0'$. Η fgets επιστρέφει s, ή NULL αν συναντηθεί τέλος αρχείου ή λάθος.

Συναρτήσεις Άμεσης Εισόδου και Εξόδου

size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream)

Η fread διαβάζει από το ρεύμα stream για τον πίνακα ptr, το πολύ nobj αντικείμενα μεγέθους size. Η fread επιστρέφει τον αριθμό των αντικειμένων που διαβάστηκαν,

που μπορεί να είναι μικρότερος από τον αριθμό που ζητήθηκε. Για τον προσδιορισμό της κατάστασης πρέπει να χρησιμοποιούνται,οι feof και ferror.

size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)

Η fwrite γράφει στο stream nobj το πλήθος αντικείμενα μεγέθους size, από τον πίνακα ptr. Επιστρέφει τον αριθμό των αντικειμένων που γράφτηκαν, που σε περίπτωση λάθους είναι λιγότερα από το nobj.

Συναρτήσεις Τοποθέτησης μέσα ατο Αρχείο

int fseek(FILE *stream, long offset, int origin)

Η fseek ορίζει για το stream τη θέση μέσα στο αρχείο. Το επόμενο διάβασμα ή γράψιμο θα προσπελάσει δεδομένα αρχίζοντας απ' αυτήν τη θέση. Για δυαδικό αρχείο, η θέση ορίζεται σε απόσταση offset χαρακτήρων από το origin, που μπορεί να είναι SEEK_SET (αρχή), SEEK_CUR (τρέχουσα θέση), ή SEEK_END (τέλος του αρχείου). Για ένα ρεύμα κειμένου, το offset πρέπει να είναι μηδέν, ή μια τιμή που επιστρέφεται από την ftell (οπότε το origin πρέπει να είναι SEEK_SET). Η fseek επιστρέφει μη-μηδενική τιμή σε περίπτωση λάθους.
long ftell(FILE *stream)

Η ftell επιστρέφει την τρέχουσα θέση στο αρχείο για το ρεύμα stream, ή -1L σε περίπτωση λάθους.

void rewind(FILE *stream)

int fgetpos(FILE *stream, fpos_t *ptr)

int fsetpos(FILE *stream, const fpos_t *ptr)

Παράδειγμα

Να γραφεί ένα πρόγραμμα MATLAB που να διαβάζει από το αρχείο input.dat τη διάσταση n, ένα $(n \times n)$ πίνακα A και ένα $(n \times 1)$ διάνυσμα X, να υπολογίζει το διάνυσμα AX και να το αποθηκεύει σ ένα αρχείο με όνομα output.dat.

Το αρχείο input.dat πρέπει να είναι π.χ. της μορφής:

3
1 0 1
0 2 −1
−2 1 0

Στην περίπτωση αυτή n=3, και, ενώ το αρχείο output.dat που θα προκύψει θα είναι το

−0.5
1
0.5
0
1.5
2

Διαβάζοντας από Αρχείο με τη load

Σύνταξη:

```
load(h, 'filename')
```

ή

```
load('filename')
```

Να γραφεί ένα πρόγραμμα MATLAB που να διαβάζει από το αρχείο input.dat

```
Το αρχείο input.dat πρέπει να είναι π.χ. της μορφής:
---------------------------------------
    1.0000    1.0000    1.0000
    2.0000    2.0000    2.0000
    1.0000    2.2000    3.0000
---------------------------------------


A=load('c:\tmp\input.dat')
```

Κεφαλαιο 8

Γραφικα

cftool Open the Curve Fitting Tool

Η συνάρτηση **plot**(x,y) σχεδιάζει 2-διάστατες γραφικές παραστάσεις του διανύσματος y ως προς x.

```
plot Linear 2-D plot Syntaxplot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyValue,...)
h = plot(...)
```

Παραδειγμα:

```
x=-4:.01:4; y = sin(x);
plot(x,y);
title('MATLAB plot');
xlabel('x');
ylabel('sin');

t = 0:pi/100:2*pi;
```

```
y = sin(t);
plot(t,y)
grid on

y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,t,y2,t,y3)
```

See Alsoaxis, bar, grid, hold, legend, line, LineSpec, loglog, plotyy, semilogx, semilogy, subplot, title, xlabel, xlim, ylabel, ylim, zlabel, zlim, stem See the text String property for a list of symbols

Συνάρτηση **plot3**

```
plot3(X1,Y1,Z1,...)
plot3(X1,Y1,Z1,LineSpec,...)
plot3(...,'PropertyName',PropertyValue,...)
```

Παραδειγμα:

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
grid on
axis square
```

Για λογαριθμική κλίμακα στον άξονα των y: **semilogy**

Παράδειγμα:

```
semilogy(x,y);
```

Συνάρτηση **surf**

```
surf(Z)
surf(X,Y,Z)
surf(X,Y,Z,C)
surf(...,'PropertyName',PropertyValue)
surfc(...)
h = surf(...)
h = surfc(...)
```

Παράδειγμα:

```
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv
axis([-3 3 -3 3 -10 5])
```

Παράδειγμα:

```
[x,y] = meshgrid([-2:.2:2]);
z = x.*exp(-x.^2 - y.^2);
surf(x,y,z)
```

Συνάρτηση **subplot**

Παράδειγμα:

```
upper_plot = subplot(211);
surf(x,y,z)
lower_plot = subplot(212);
surf(x,y,z)
pbaspect(upper_plot,'manual')
```

Συνάρτηση **surface**

```
surface(Z)
surface(Z,C)
surface(X,Y,Z)
surface(X,Y,Z,C)
surface(...'PropertyName',PropertyValue,...)
h = surface(...)
```

Συνάρτηση **imagesc**

```
imagesc(C)
imagesc(x,y,C)
imagesc(...,clims)
h = imagesc(...)
```

Συνάρτηση **polar**

```
polar(theta,rho)
polar(theta,rho,LineSpec)
```

Παράδειγμα:

```
t = 0:.01:2*pi;
polar(t,sin(2*t).*cos(2*t),'--r')

plotmatrix(X,Y)
plotmatrix(...,'LineSpec')
[H,AX,BigAx,P] = plotmatrix(...)


x = randn(50,3); y = x*[-1 2 1;2 0 1;1 -2 3;]';
plotmatrix(y,'*r')


contour(Z)
```

```
contour(Z,n)
contour(Z,v)
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
contour(...,LineSpec)
[C,h] = contour(...)

[x,y,z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
hold off

[X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
[C,h] = contour(X,Y,Z);
clabel(C,h)
colormap cool

[x,y,z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
hold off
```

```
contour3(Z)
contour3(Z,n)
contour3(Z,v)
contour3(X,Y,Z)
contour3(X,Y,Z,n)
contour3(X,Y,Z,v)
contour3(...,LineSpec)

[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2-Y.^2);
contour3(X,Y,Z,30)
surface(X,Y,Z,'EdgeColor',[.8 .8 .8],'FaceColor','none')
grid off
view(-15,25)
colormap cool


mesh(X,Y,Z)
mesh(Z)
mesh(...,C)
mesh(...,'PropertyName',PropertyValue,...)
meshc(...)
meshz(...)
h = mesh(...)
h = meshc(...)
h = meshz(...)
```

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])

[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshz(X,Y,Z)



slice(V,sx,sy,sz)
slice(X,Y,Z,V,sx,sy,sz)
slice(V,XI,YI,ZI)
slice(X,Y,Z,V,XI,YI,ZI)

[x,y,z] = meshgrid(-2:.2:2,-2:.25:2,-2:.16:2);
v = x.*exp(-x.^2-y.^2-z.^2);
xslice = [-1.2,.8,2]; yslice = 2; zslice = [-2,0];
slice(x,y,z,v,xslice,yslice,zslice)
colormap hsv



Y = [5 1 2
     8 3 7
```

```
     9 6 8
     5 5 5
     4 2 3];
area(Y)

errorbar(Y,E)
errorbar(X,Y,E)
errorbar(X,Y,L,U)
errorbar(...,LineSpec)
h = errorbar(...)

X = 0:pi/10:pi;
Y = sin(X);
E = std(Y)*ones(size(X));
errorbar(X,Y,E)


alpha = 0.01;
beta = 0.5;
t = 0:10;
f = exp(-alpha*t).*sin(beta*t);

stairs(t,f)
hold on
plot(t,f,'--*')
hold off
```

```
bar(Y)
bar(x,Y)
bar(...,width)
bar(...,'style')
bar(...,LineSpec)

x = -2.9:0.2:2.9;
bar(x,exp(-x.*x))
colormap hsv



Y = round(rand(5,3)*10);
subplot(2,2,1)
bar(Y,'group')
title 'Group'

subplot(2,2,2)
bar(Y,'stack')
title 'Stack'

subplot(2,2,3)
barh(Y,'stack')
title 'Stack'

subplot(2,2,4)
bar(Y,1.5)
title 'Width = 1.5'
```

```
randn('state', 0)             % Start from a known state.
x = randn(1000, 1);           % 1000 Gaussian deviates ~ N(0,1).
y = filter([1 -1 1], 1, x);   % Create an MA(2) process.
[ACF, Lags, Bounds] = autocorr(y, [], 2);  % Compute the ACF
                                           % with 95 percent
                                           % confidence.
[Lags, ACF]

autocorr(y, [], 2)      % Use the same example, but plot the ACF
                        % sequence with confidence bounds.
```

```
fill(X,Y,C)
fill(X,Y,ColorSpec)
fill(X1,Y1,C1,X2,Y2,C2,...)
fill(...,'PropertyName',PropertyValue)
h = fill(...)
fill3(X,Y,Z,C)
fill3(X,Y,Z,ColorSpec)
fill3(X1,Y1,Z1,C1,X2,Y2,Z2,C2,...)
fill3(...,'PropertyName',PropertyValue)
h = fill3(...)
```

```
t = (1/16:1/8:1)'*2*pi;
x = sin(t);
y = cos(t);
fill(x,y,'r')
axis square

X = [0 1 1 2;1 1 2 2;0 0 1 1];
Y = [1 1 1 1;1 0 1 0;0 0 0 0];
Z = [1 1 1 1;1 0 1 0;0 0 0 0];
C = [0.5000 1.0000 1.0000 0.5000;
     1.0000 0.5000 0.5000 0.1667;
     0.3330 0.3330 0.5000 0.5000];
fill3(X,Y,Z,C)
```

```
Log-log scale plot Syntaxloglog(Y)
loglog(X1,Y1,...)
loglog(X1,Y1,LineSpec,...)
loglog(...,'PropertyName',PropertyValue,...)
h = loglog(...)

x = logspace(-1,2);
```

```
loglog(x,exp(x),'-s')
grid on
```

Κεφαλαιο 9

Συμβολοσειρες

Μέχρι τώρα περιστραφήκαμε γύρω από τους πίνακες, αριθμών (είτε int είτε double), όμως αντίστοιχα ορίζονται και οι πίνακες χαρακτήρων δηλαδή

```
char pin[7]; char *pin;
```

Ένα String (συμβολοσειρά) είναι ένας πίνακας χαρακτήρων, που στη C τελειώνει με $'\backslash 0'$.
Παράδειγμα: η λέξη "$hello\backslash n$" θα καταχωρούνταν στον πίνακα ως

'h'	'e'	'l'	'l'	'o'	$'\backslash n'$	$'\backslash 0'$

Η C μέσα από μια σειρά συναρτήσεων μας παρέχει τη δυνατότητα να κάνουμε πράξεις και συγκρίσεις μεταξύ αλφαριθμητικών. Παρακάτω παρουσιάζουμε τις πιο σημαντικές απ' αυτές τις συναρτήσεις, όπου οι μεταβλητές s και t είναι τύπου char *, οι cs και ct είναι τύπον const char *, το n είναι τύπου size_t, και το c είναι ένας int που μετατρέπεται σε char.

char *strcpy(s, ct)

αντιγράφει το αλφαριθμητικό ct στο αλφαριθμητικό s, περιλαμβάνοντας και το $'\backslash 0'$. Επιστρέφει το s.

char* strncpy(s,ct,n)

αντιγράφει το πολύ η χαρακτήρες του αλφαριθμητικού ct στο s επιστρέφει s. Αν το ct έχει λιγότερους από n χαρακτήρες, συμπληρώνεται με χαρακτήρες , $'\backslash 0'$.

char *strcat (s, ct)

συνενώνει το αλφαριθμητικό ct στο τέλος του αλφαριθμητικού s επιστρέφει το s.

char *strncat(s, ct, n)

συνενώνει το πολύ η χαρακτήρες του αλφαριθμητικού ct με το αλφαριθμητικό s, και τερματίζει το s με , $'\backslash 0'$ επιστρέφει το s.

int strcmp (cs, ct)

συγκρίνει το αλφαριθμητικό cs με το αλφαριθμητικό ct επιστρέφει < 0 αν $cs < ct$, 0 αν cs==ct, και > 0 αν $cs > ct$.

int strncmp(cs, ct, n)

συγκρίνει το πολύ n χαρακτήρες του αλφαριθμητικού cs με το αλφαριθμητικό ct επιστρέφει < 0 αν $cs < ct$, 0 αν cs==ct, και > 0 αν $cs > ct$.
int *strchr(cs, c)

επιστρέφει δείκτη στην πρώτη εμφάνιση του c μέσα στο cs, ή NULL αν δεν υπάρ-

χει.

char *strrchr(cs, c)

επιστρέφει δείκτη στην τελευταία εμφάνιση του c μέσα στο cs, ή NULL αν δεν υπάρχει.

char *strpbrk (cs, ct)

επιστρέφει δείκτη στην πρώτη εμφάνιση μέσα στο αλφαριθμητικό cs, οποιουδήποτε χαρακτήρα του αλφαριθμητικού ct, ή NULL αν δεν υπάρχει κανένας.

char *strdup (cs)

κάνει malloc την μεταβλητή (του δεξειούτου μέλους) με θέσεις ίσες με το μήκος του cs και 1 και της αναθέτη το αλφαριθμητικό cs, με $'\backslash 0'$ στο τέλος.

char *strstr(cs, ct)

επιστρέφει δείκτη στην πρώτη εμφάνιση του αλφαριθμητικού ct μέσα στο cs, ή NULL αν δεν υπάρχει
size-t strlen(cs)

επιστρέφει το μήκος του cs.

Βιβλιογραφία

[1] Μ. Δρακόπουλος, Διδακτικές σημειώσεις για το MATLAB,Μαθηματικό Αθηνών, 1999.

[2] M. Abell and J. Braselton, Mathematica by Example, Second Edition, Academic Press, 1997.

Ευρετήριο