ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ

ΔΙΔΑΚΤΕΑ ΥΛΗ

Γενικά περί Πληροφορικής και Υπολογιστών. Τα μέρη του υπολογιστή (hardware = υλικό).

Τα προγράμματα του υπολογιστή (software = λογισμικό).

Συστήματα αρίθμησης και μετατροπές από το ένα σύστημα στο άλλο.

Η έννοια του αλγορίθμου. Αλγοριθμικές δομές. Λογικά διαγράμματα (flowcharts).

Προγραμματισμός σε C. Τύποι δεδομένων και μεγέθη.

Εντολές ελέγχου (if – else, switch).

Συσχετικοί και λογικοί τελεστές.

Εντολές επανάληψης (for, while, do - while).

Συναρτήσεις και δόμηση σε μπλοκ του προγράμματος.

Δείκτες (pointers) και Πίνακες. Αναδρομή και αναδρομικές συναρτήσεις.

Τεχνικές αναζήτησης (search). Τεχνικές ταξινόμησης (sort).

Προσπέλαση αρχείων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. ***«Εισαγωγή στην Πληροφορική. Προγραμματισμός με την Turbo Pascal»***, Χ. Τζόκας, Ν. Καρασαχινίδης, Εκδόσεις Δίαυλος, Αθήνα 1997. (μας ενδιαφέρει μόνο το πρώτο μέρος του βιβλίου που περιλαμβάνει μια σύντομη εισαγωγή στην πληροφορική, από σελίδα 19 έως 72)
2. ***«Η γλώσσα C σε βάθος»,*** Νίκος Χατζηγιαννάκης, Κλειδάριθμος, 2005.
3. *"**The C Programming Language**" (second edition),* Brian Kernighan, Dennis Ritchie, Prentice-Hall,1988, ή την μετάφρασή του *"Η γλώσσα Προγραμματισμού C",* από τον Θωμά Μωραΐτη, Κλειδάριθμος 1990.
4. ***«C για Αρχάριους»,*** Βασίλης Σεφερίδης, Κλειδάριθμος, 1995.
5. *"**C The Complete Reference**",* Herbert Schildt, Osborn/McGraw-Hill, 1987.
6. ***«Εγχειρίδιο εκμάθησης Turbo C»,*** Herbert Schildt.
7. ***«Οδηγός της C»,*** 3η εκδ., Herbert Schildt, απόδοση Ε. Γκαγκάτσιου, εκδότης: Μ.Γκιούρδας, Αθήνα 2000.
8. *"**C: How to program**" (second edition),* H. M. Deitel, P. J. Deitel, , Prentice-Hall, 1999.
9. *"**C Programming for Engineering & Computer Science**",* H.H.Tan, T.B. D'Orazio, McGraw-Hill, 2000, ή την μετάφρασή του ***"C για Μηχανικούς"***, από τους Δ. Μανωλάκη, Χ. Πολάτογλου, εκδόσεις Τζιόλα, 2000.

1. Εισαγωγή στους Ηλεκτρονικούς Υπολογιστές (Η/Υ)

1.1. Γενικά περί Πληροφορικής και Υπολογιστών

Ηλεκτρονικός Υπολογιστής (Η/Υ) είναι μια περίπλοκη ηλεκτρονική συσκευή, που έχει τη δυνατότητα να επεξεργάζεται πολύ γρήγορα και με ακρίβεια τεράστιο όγκο πληροφοριών (δεδομένων).

Η Πληροφορική είναι η επιστήμη που έχει ως αντικείμενο τη συλλογή, επεξεργασία, φύλαξη και μετάδοση των πληροφοριών, με τη βοήθεια του υπολογιστή.

Η ελάχιστη ποσότητα πληροφορίας στους ηλεκτρονικούς υπολογιστές είναι το ένα δυαδικό ψηφίο (**bit** = binary digit) που μπορεί να λάβει τιμή 0 ή 1. Μια οχτάδα τέτοιων ψηφίων αποτελούν μία ψηφιολέξη (**byte**) που είναι η μονάδα μέτρησης της χωρητικότητας των διαφόρων τμημάτων του Η/Υ.
Στη πράξη συνήθως χρησιμοποιούμε τα πολλαπλάσια αυτής της μονάδας που είναι:

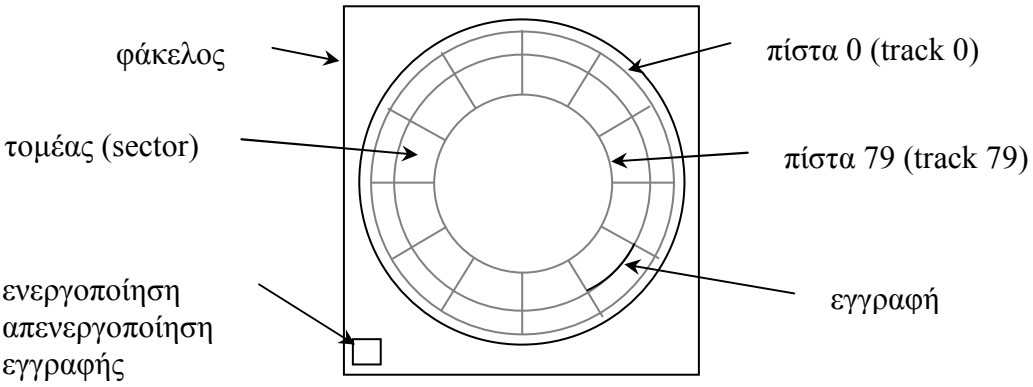
KiloByte	$1KB = 2^{10}$ Bytes = 1024 Bytes,
MegaByte	$1MB = 2^{20}$ Bytes = 2^{10} KB = 1024 KB = 1.048.576 Bytes,
GigaByte	$1GB = 2^{30}$ Bytes = 2^{10} MB = 1024 MB,
TeraByte	$1TB = 2^{40}$ Bytes = 2^{10} GB.

1.1.1. Τα μέρη του υπολογιστή (hardware = υλικό)

- κεντρική μονάδα
  - επεξεργαστής ή μικροεπεξεργαστής (microprocessor) π.χ. INTEL 8080, Zilog Z80, Motorola 6800 και οι πιο ισχυροί INTEL 80386, 80486, Motorola 68020, 68030 μαζί με τα κυκλώματα υποστήριξης. Λέγεται και κεντρική μονάδα επεξεργασίας (CPU ή Central Processing Unit) και αποτελείται ουσιαστικά από δύο μέρη: την αριθμητική και λογική μονάδα (ALU ή Arithmetic and Logic Unit) και την μονάδα ελέγχου (Control Unit).
  - κεντρική μνήμη ή κύρια μνήμη (main memory)
    (RAM ή Random Access Memory = μνήμη τυχαίας ή άμεσης προσπέλασης)
    (ROM ή Read Only Memory = μνήμη μόνο για ανάγνωση με μόνιμο περιεχόμενο)

- περιφερειακές μονάδες
  - μονάδα εύκαμπτου ή μαλακού μαγνητικού δίσκου (floppy disk drive),

	χωρητικότητα	
διάμετρος δισκέτας	DD ή Double Density	HD ή High Density
3,5 ίντσες	720KB (s=9, t=80)	1,44MB (s=18, t=80)
5¼ ίντσες	360KB (s=9, t=40)	1,2MB (s=15, t=80)



- μονάδα άκαμπτου ή σκληρού μαγνητικού δίσκου (hard disk drive), χωρητικότητες από 10 MB μέχρι 100 GB

  Σημείωση: Ο υπολογισμός της συνολικής χωρητικότητας (dc) ενός μαγνητικού δίσκου δίνεται από τον παρακάτω τύπο: $dc = s \cdot b \cdot t \cdot h$

  όπου s είναι ο αριθμός των τομέων (sectors) στον δίσκο,

  b ο αριθμός των Bytes ανά τομέα,

  t είναι ο αριθμός από τις πίστες (tracks) που έχει ο δίσκος, και

  h ο αριθμός των κεφαλών (heads) ή πλευρών του δίσκου.

  π.χ. για δισκέτες 3.5 ιντσών σε High-Density έχουμε s=18, b=512, t=80, h=2 οπότε

  $dc = 18 \cdot 512 \cdot 80 \cdot 2 = 1440$ KB ή 1.44MB

- μονάδα μαγνητικών δίσκων τύπου ZIP με χωρητικότητα 100MB ή και 250MB
- μονάδα μαγνητικής ταινίας (tape unit)
- μονάδα ανάγνωσης/εγγραφής ψηφιακών οπτικών δίσκων (Digital Optical Disks ή DOD) CD-ROM ή Compact Disk - Read Only Memory ή ακόμα καλύτερα CD-R δηλαδή Compact Disk - Recorder που είναι κατάλληλο τόσο για ανάγνωση όσο και για εγγραφή. Ένας τέτοιος δίσκος έχει χωρητικότητα 600 - 700MB.
- μονάδα ανάγνωσης/εγγραφής DVD (Digital Video Disk)

  Σημείωση: Όλες οι παραπάνω περιφερειακές μονάδες θεωρούνται περιφερειακή μνήμη και αποτελούν την προέκταση της κεντρικής μνήμης.

- πληκτρολόγιο (keyboard) και ποντίκι (mouse)
- οθόνη (monitor)
- εκτυπωτή (printer) ή και σχεδιογράφο (plotter)
- διάφορες "κάρτες" όπως:
  - κάρτα γραφικών - για την απεικόνιση στην οθόνη
  - κάρτα ήχου - για την εγγραφή, δημιουργία και απόδοση του ήχου
  - κάρτα δικτύου - για την επικοινωνία με άλλους υπολογιστές δια μέσου εξειδικευμένων γραμμών δικτύου
  - κάρτα Fax-Modem - για την επικοινωνία με άλλους υπολογιστές (ή άλλες συσκευές όπως το Fax) δια μέσου απλών τηλεφωνικών γραμμών

1.1.2. Τα λειτουργικά υποσυστήματα ενός Η/Υ

Ένας Η/Υ λοιπόν είναι ένα σύστημα αυτόματης επεξεργασίας δεδομένων, όπως είδαμε πιο πάνω. Στο παρακάτω γενικό διάγραμμα (Σχήμα 1.) διακρίνουμε τα λειτουργικά υποσυστήματα που απαρτίζουν έναν Η/Υ.

Συσκευές εισόδου - είναι ηλεκτρομαγνητικές συσκευές όπως: ο αναγνώστης διάτρητων δελτίων (cards) ή διάτρητων χαρτοταινιών (paper tapes) και το τερματικό (terminal), δηλαδή ένα πληκτρολόγιο και μια συσκευή απεικόνισης σε χαρτί ή σε οθόνη καθοδικού σωλήνα (Cathode Ray Tube, CRT). Σήμερα υπάρχουν κι' άλλες συσκευές εισόδου για εξειδικευμένες εργασίες όπως ο αναγνώστης ραβδωτών κωδίκων (bar code reader), ο σαρρωτής (scanner), διάφοροι αισθητήρες (sensors) κλπ.

Συσκευές εξόδου – μπορεί να είναι συσκευές όπως ο εκτυπωτής (printer) ή ο σχεδιογράφος (plotter) και το τερματικό (terminal).

Σημείωση:

Ταχύτητα ανάγνωσης δελτίων 300-1000 δελτία / λεπτό, ενώ 1 δελτίο περιέχει το πολύ 80 χαρακτήρες.
Ταχύτητα εκτυπωτή 800-1000-2500 γραμμές / λεπτό, ενώ 1 γραμμή περιέχει το πολύ 132 χαρακτήρες.

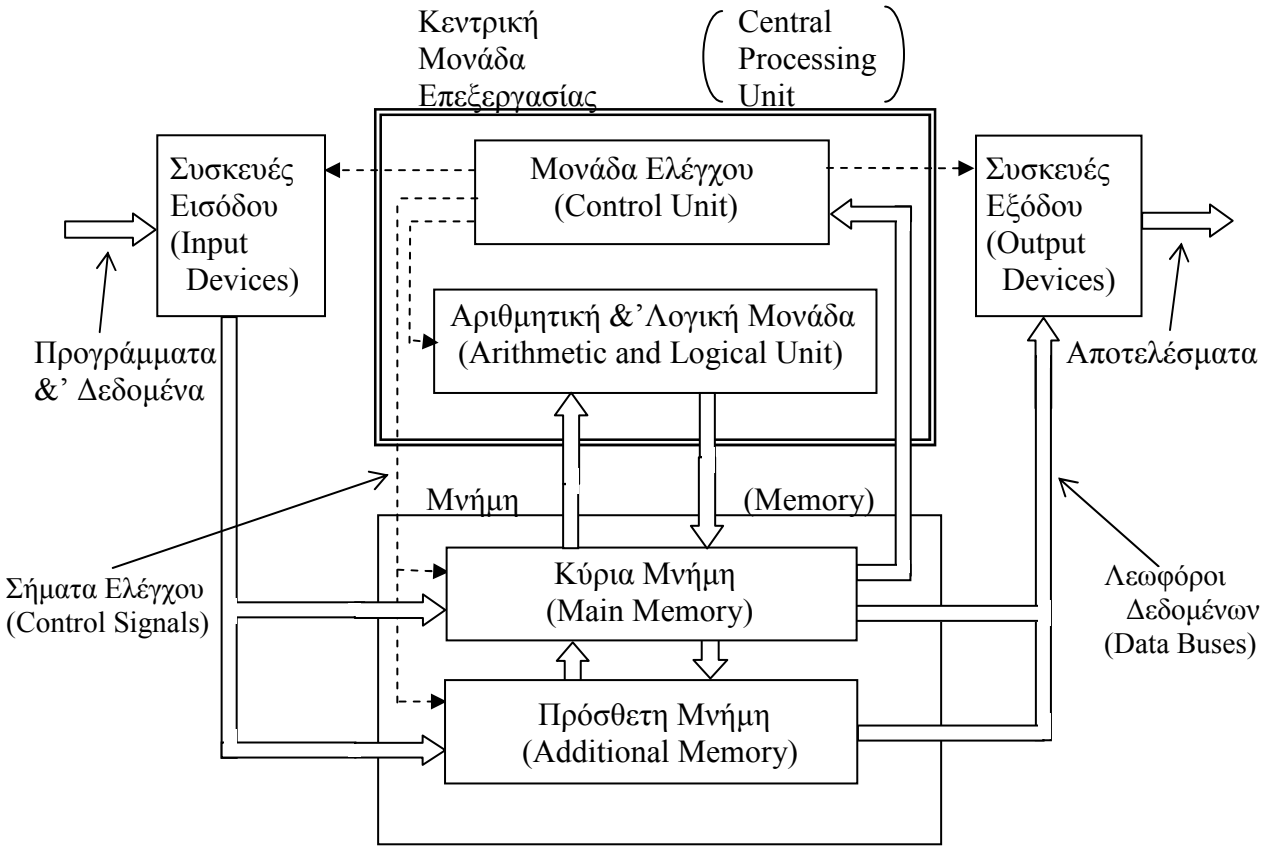
Κύρια μνήμη ή εσωτερική – συστήματα δυο καταστάσεων όπως τα flip-flops που παλαιότερα ήταν κατασκευασμένα από σιδηρομαγνητικά υλικά με σχήμα δακτυλίου ενώ τα τελευταία χρόνια έχουν επικρατήσει οι ημιαγωγοί με λογικά κυκλώματα κρυσταλλοδιόδους και κρυσταλλοτριόδους και ονομάζονται μνήμες στερεάς κατάστασης (solid state memory). Τα στοιχεία της μνήμης είναι συνήθως οργανωμένα σε ομάδες με μήκος που τις περισσότερες φορές είναι πολλαπλάσιο του 8.

1 byte = 8 δυαδικά ψηφία (bits) (δηλ. χρειάζονται 8 κύτταρα μνήμης ή στοιχεία μνήμης)

4 bytes = μια λέξη μνήμης (word)

Η χωρητικότητα της μνήμης είναι ένα από τα κύρια χαρακτηριστικά της και μετριέται συνήθως σε Kbytes ή Mbytes.

Η πρόσβαση στη κύρια μνήμη γίνεται σε χρόνο μικρότερο του ενός εκατομμυριοστού του δευτερολέπτου (access time << 1 *μsec)* (σήμερα 50÷100 *n*sec) και μεταφέρονται ταυτόχρονα 1 - 4 bytes.



Σχήμα 1. Τα λειτουργικά υποσυστήματα ενός Η/Υ

Πρόσθετη μνήμη ή εξωτερική – αποτελείται συνήθως από μαγνητικές ταινίες (tapes) ή μαγνητικούς δίσκους και χρησιμεύει στην αποθήκευση και στην επαναφορά των δεδομένων.

Σε γενικές γραμμές αυτές οι συσκευές έχουν πολύ μεγαλύτερο χρόνο πρόσβασης από τα συστήματα κύριας μνήμης, όμως έχουν μεγαλύτερες χωρητικότητες και πολύ καλύτερη σχέση κόστους/χωρητικότητας. Η ταχύτητα μεταφοράς για μια μαγνητική ταινία είναι 60 ÷ 100 Kbytes/sec ή ίσως λίγο μεγαλύτερη και πυκνότητα εγγραφής 800 ÷ 1600 bits/inch.

Για τους μαγνητικούς δίσκους έχουμε χρόνο πρόσβασης από ≈ 100 *m*sec στους παλαιότερους, μέχρι και κάτω των 10 ms στους πιο καινούργιους και ταχύτητα μεταφοράς δεδομένων από 100÷200 Kbytes/sec μέχρι μερικά Mbytes/sec. Για τα Zip disk drives έχουμε χρόνο προσπέλασης 29msec και ρυθμό μεταφοράς έως 20MB/min ≈ 300KB/sec.

Σημείωση:

Ο χρόνος προσπέλασης ή πρόσβασης (**access time**) είναι το απαραίτητο χρονικό διάστημα που χρειάζεται μια συσκευή μνήμης για να φτάσει στην αρχή των αναζητούμενων πληροφοριών, ενώ ο ρυθμός μεταφοράς ή ταχύτητα μεταφοράς (**transfer rate**) φανερώνει το πόσο γρήγορα περνάνε οι πληροφορίες από την μνήμη στον υπολογιστή.

Η χωρητικότητα των πρώτων σκληρών δίσκων (hard disc) ήταν από 256Kbytes μέχρι 300Mbytes σε συστοιχίες των 10 δίσκων. Σήμερα η χωρητικότητα αυτή τείνει να ξεπεράσει τα 100Gbytes.

Επίσης μια άλλη μοντέρνα συσκευή πρόσθετης μνήμης με ευρεία διάδοση σήμερα είναι η μονάδα ανάγνωσης/εγγραφής ψηφιακών οπτικών δίσκων (Digital Optical Discs ή DOD). Ένας τέτοιος δίσκος έχει χωρητικότητα περίπου 700MB.

Αριθμητική και λογική μονάδα – ηλεκτρονικά κυκλώματα που μπορούν να κάνουν διάφορες πράξεις με δυαδικές σειρές (συστοιχίες δυαδικών ψηφίων) όπως:

- απλές αριθμητικές πράξεις: πρόσθεση, αφαίρεση ίσως πολλαπλασιασμό και διαίρεση
- λογικές πράξεις: άρνηση, σύζευξη, διάζευξη.

Π.χ. πρόσημο: 0 = συν (+), 1 = μείον (-)

0	0 0 1 0 1 1 0	'+'	= 22	0	0 0 1 0 1 1 0	'AND'
0	0 1 0 1 1 0 1		= 45	0	0 1 0 1 1 0 1	
0	1 0 0 0 0 1 1		= 67	0	0 0 0 0 1 0 0	

Ταχύτητα εργασίας <<1μsec για μια απλή πράξη.

Μονάδα ελέγχου – είναι ένα ηλεκτρονικό τμήμα που περιέχει κυκλώματα τα οποία μπορούν να ερμηνεύουν σειρές (συστοιχίες) δυαδικών ψηφίων σαν διαταγές και να στέλνουν τα κατάλληλα σήματα ελέγχου στα υπόλοιπα τμήματα (συνθετικά μέρη) του συστήματος με σκοπό την πραγματοποίηση κάποιων πράξεων.

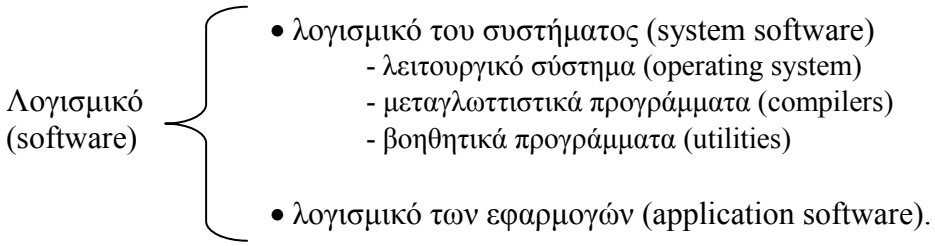
Οι λέξεις (οι αποτελούμενες από δυαδικά ψηφία) που ερμηνεύονται από την μονάδα ελέγχου λέγονται εντολές (instructions). Μαζί με αυτές στη κύρια μνήμη βρίσκονται και τα δεδομένα (data) ενός προβλήματος.

Μια ακολουθία από εντολές με την βοήθεια της οποίας επιδιώκεται η λύση ενός προβλήματος ονομάζεται πρόγραμμα.

1.1.3. Τα προγράμματα του υπολογιστή (software = λογισμικό)

Διακρίνουμε το λογισμικό των υπολογιστών σε δύο κατηγορίες:

(α) το λογισμικό του συστήματος (system software) που περιλαμβάνει το λειτουργικό σύστημα (operating system π.χ. CP/M, DOS, WINDOWS, UNIX), τα μεταγλωττιστικά προγράμματα (compilers) και τα βοηθητικά προγράμματα ή προγράμματα γενικής χρήσης (utilities) και **(β) το λογισμικό των εφαρμογών (application software).**



Τι είναι ένα λειτουργικό σύστημα;

Λειτουργικό σύστημα (operating system) είναι μια συλλογή από προγράμματα εξειδικευμένη για κάθε τύπο Η/Υ, που αυξάνει τις δυνατότητες την ευχρηστία και γενικά την αποτελεσματικότητα των συστημάτων υπολογισμού, κυνηγώντας δύο βασικούς στόχους:

(α) τη λογική χρήση (διαχείριση) των πόρων (αποθεμάτων) του συστήματος.
(β) τη διευκόλυνση των προγραμματιστών στην ανάπτυξη και εκτέλεση των προγραμμάτων.

Για να χρησιμοποιηθεί ένας υπολογιστής είναι απαραίτητη τόσο η γνώση μιας γλώσσας προγραμματισμού όσο και η γνώση της εξειδικευμένης γλώσσας ελέγχου ή επικοινωνίας με το λειτουργικό σύστημα του υπολογιστή.

Δια μέσου αυτής της γλώσσας επικοινωνίας, ο χρήστης ή ο προγραμματιστής εκφράζει τις υπηρεσίες που επιθυμεί να λάβουν χώρα από το λειτουργικό σύστημα. Ανάλογα με τον τρόπο πρόσβασης στο λειτουργικό σύστημα, τα συστήματα πληροφορικής χωρίζονται (κατατάσσονται) σε δύο κατηγορίες:

1. Συστήματα επεξεργασίας ανά δεσμίδες εντολών (ή αλλιώς ανά ομάδες ή τμήματα) (Batch Processing Systems). Ο χρήστης δεν βρίσκεται σε άμεση επαφή με τον Η/Υ, έτσι τα προγράμματα και οι εντολές της γλώσσας επικοινωνίας του λειτουργικού συστήματος παραδίδονται συνήθως υπό μορφή διάτρητων δελτίων (cards) σε ένα υπολογιστικό κέντρο ενώ τα αποτελέσματα παραλαμβάνονται μετά από ένα διάστημα μερικών ωρών ή ημερών υπό μορφή μιας εκτύπωσης (listing). Αυτή η μορφή επεξεργασίας εμφανίστηκε μαζί με τα πρώτα συστήματα πληροφορικής.
2. Συστήματα διαλογικά (Interactive Systems). Ο χρήστης, σε αντίθεση με την προηγούμενη περίπτωση, βρίσκεται σε άμεση επαφή με τον Η/Υ. Τα προγράμματα και τα δεδομένα εισάγονται από τα τερματικά ενώ τα αποτελέσματα μπορούν να αποκτηθούν πάλι από το τερματικό ή υπό μορφή μιας εκτύπωσης. Σήμερα όλα τα συστήματα ανήκουν σ' αυτήν την κατηγορία, ενώ δεν είναι λίγα αυτά που ενσωματώνουν και την προηγούμενη μορφή επεξεργασίας.

Η γενικότερη τάση που υπάρχει τα τελευταία χρόνια είναι να δημιουργηθούν όσο το δυνατόν φιλικότερα ως προς τον χρήστη (User Friendly) περιβάλλοντα ώστε η χρήση τουλάχιστον των υπολογιστών να απαιτεί όσο το δυνατόν λιγότερο εξειδικευμένα άτομα.

Οργάνωση της μνήμης

Μια από τις σημαντικές εργασίες που κάνει το λειτουργικό σύστημα σ' ένα ηλεκτρονικό υπολογιστή είναι η δυναμική διαχείριση της μνήμης. Δηλαδή, ανάλογα με τις ανάγκες του κάθε προγράμματος οριοθετεί αυτόματα ένα χώρο εργασίας στην μνήμη. Αυτός ο χώρος μπορεί να

χωρισθεί σε δύο βασικές περιοχές: στην περιοχή του προγράμματος, όπου αποθηκεύονται οι εντολές του προγράμματος και στην περιοχή των δεδομένων, όπου φυλάσσονται τα δεδομένα οι μεταβλητές και τα μερικά (ενδιάμεσα) ή τελικά αποτελέσματα της επεξεργασίας.

Η έννοια της μεταβλητής εδώ διαφέρει πολύ από αυτήν των μαθηματικών. Μια μεταβλητή στην Πληροφορική έχει ένα όνομα και ένα περιεχόμενο. Το περιεχόμενο αποτελεί την τιμή της και μπορεί να αλλάξει πολλές φορές κατά τη διάρκεια εκτέλεσης του προγράμματος. Τιμή μιας μεταβλητής σε μια χρονική στιγμή είναι η τελευταία τιμή που έχει καταχωρηθεί με το όνομα αυτό. Το μέγεθος μιας μεταβλητής (δηλ., πόσα bytes έχουν δεσμευτεί για να φιλοξενήσουν το περιεχόμενό της) εξαρτάται από τον τύπο της. Συνήθως, δίνουμε τέτοια ονόματα στις μεταβλητές ώστε να μας υπενθυμίζουν το ρόλο που διατελούν.

Το λειτουργικό σύστημα **CP/M**

Το λειτουργικό σύστημα **CP/M** (Console Program for Microcomputers) της εταιρείας Digital Research αποτελείται από τρία μέρη:

- ◊ BIOS (Basic Input /Output System)
- ◊ BDOS (Basic Disk Operating System) και CCP (Console Command Processor) που περιέχει τις εσωτερικές εντολές (dir, type, ren, era, save, user).
- ◊ και τις εξωτερικές εντολές (stat (status), pip (Peripheral Interchange Program), dip (Disk Interchange Program), dump, compare, unera (unerase), eraq (erase with query) κλπ.)

Το λειτουργικό σύστημα **MS-DOS**

Το λειτουργικό σύστημα **MS-DOS** (MicroSoft Disk Operating System) αποτελείται από τρία μέρη:

- ◊ BIOS (Basic Input /Output System) που βρίσκεται στη ROM
- ◊ IO.SYS, MSDOS.SYS και COMMAND.COM που βρίσκονται στη δισκέτα συστήματος ή δισκέτα εκκίνησης (system disk). Το αρχείο COMMAND.COM περιέχει τις εσωτερικές εντολές (dir, type, ren, del (ή erase), copy, cls (clear screen)).
- ◊ και τις εξωτερικές εντολές που βρίσκονται συνήθως στον υποκατάλογο C:\DOS> (π.χ. fdisk, format, print, xcopy, diskcopy, label κλπ)

Οι εντολές του DOS είναι λέξεις ή συντομογραφίες που πληκτρολογούνται μετά το σύμβολο ετοιμότητας για το περιβάλλον DOS (prompt) π.χ. C:\>

Εσωτερικές εντολές του MS-DOS

Εντολές με ευρετήρια (directories)

md	(ή **mkdir**) *<όνομα ευρετηρίου>*	Δημιουργία ευρετηρίου
	π.χ.	
	md dokimi	Δημιουργία ευρετηρίου με όνομα dokimi
cd	(ή **chdir**) *<όνομα ευρετηρίου>*	Αλλαγή τρέχοντος ευρετηρίου
	π.χ.	
	cd dokimi	Μεταφορά στο ευρετήριο dokimi
	cd ..	Μεταφορά στο ευρετήριο "γονέας" (parent)
	cd\	Μεταφορά στο ευρετήριο "ρίζα" (root)

rd (ή **rmdir**) *<όνομα ευρετηρίου>* Διαγραφή ευρετηρίου εφόσον είναι άδειο

π.χ.
rd dokimi Διαγραφή του ευρετηρίου dokimi

dir *<όνομα ευρετηρίου>* Εμφάνιση των περιεχομένων ενός ευρετηρίου

π.χ.
dir Εμφάνιση των περιεχομένων του τρέχοντος ευρετηρίου

Εντολές με αρχεία (files)

copy <αρχείο πηγή> <αρχείο προορισμός> Αντιγραφή αρχείων

π.χ.
copy con keimeno.txt Δημιουργία του αρχείου keimeno.txt
copy keimeno.txt gramma.doc Αντιγραφή του αρχείου keimeno.txt στο gramma.doc

type <όνομα αρχείου> | more Εμφάνιση των περιεχόμενων ενός αρχείου

π.χ.
type keimeno.txt Εμφάνιση περιεχόμενων του αρχείου keimeno.txt
type gramma.doc Εμφάνιση περιεχόμενων του αρχείου gramma.doc

del (ή **erase**) <όνομα αρχείου> Διαγραφή αρχείου

π.χ.
del keimeno.txt Διαγραφή του αρχείου keimeno.txt

ren <παλαιό όνομα αρχείου> <νέο όνομα αρχείου> Μετονομασία αρχείου

π.χ.
ren gramma.doc epistoli.txt Μετονομασία του αρχείου gramma.doc σε epistoli.txt

Διάφορες άλλες εσωτερικές εντολές

prompt Καθορισμός του συμβόλου ετοιμότητας

time Εμφάνιση ή καθορισμός της ώρας του συστήματος

date Εμφάνιση ή καθορισμός της ημερομηνίας του συστήματος

ver Εμφάνιση της έκδοσης του λειτουργικού συστήματος (MS-DOS)

cls Καθαρισμός της οθόνης και επαναφορά του δρομέα πάνω αριστερά

sort Εντολή ταξινόμησης

tree Εμφάνιση της δενδρικής δομής των ευρετηρίων

path Καθορισμός του μονοπατιού αναζήτησης αρχείων

Εξωτερικές εντολές του MS-DOS

fdisk Εντολή για τον χωρισμό σε μέρη του σκληρού δίσκου (partitioning)

format Εντολή για την διαμόρφωση (προετοιμασία) των δισκετών και των σκληρών δίσκων

diskcopy Εντολή για την δημιουργία ενός αντίγραφου μιας δισκέτας

diskcomp	Εντολή για την σύγκριση των περιεχομένων δύο δισκετών
chkdsk	Εντολή αναζήτησης βλαβών σε δίσκο
backup	Εντολή για δημιουργία αντιγράφων ασφαλείας
restore	Εντολή για επαναφορά αρχείων από αντίγραφα ασφαλείας
xcopy	Εντολή για την αντιγραφή αρχείων

Στο DOS το όνομα (filename) ενός αρχείου μπορεί να λάβει μέχρι 8 χαρακτήρες και μια επέκταση (filename extension) μέχρι 3 χαρακτήρες. Πολλές φορές η επέκταση παίζει καθοριστικό ρόλο στον τύπο του αρχείου:

π.χ. .BAT για αρχεία που περιέχουν μια ομάδα εντολών (batch files),
.COM για μικρά εκτελέσιμα αρχεία (μέχρι 64KB),
.EXE για εκτελέσιμα αρχεία,
.SYS για αρχεία συστήματος,
.BAK για αντίγραφα ασφαλείας,
.BAS για αρχεία που περιέχουν προγράμματα σε γλώσσα BASIC,
.PAS για αρχεία σε γλώσσα PASCAL,
.C για αρχεία σε γλώσσα C,
.CPP για αρχεία σε γλώσσα C++, κλπ.

Τα μόνα εκτελέσιμα αρχεία στο λειτουργικό σύστημα MS-DOS είναι αυτά που έχουν επέκταση ονόματος BAT, COM και EXE. Αυτή είναι και η σειρά προτεραιότητας με την οποία το DOS αναζητεί ένα εκτελέσιμο αρχείο. Αυτό μπορεί να γίνει αντιληπτό στην περίπτωση που υπάρχουν στο ίδιο ευρετήριο δύο ή τρία εκτελέσιμα αρχεία με το ίδιο όνομα (αλλά με διαφορετική επέκταση ονόματος) εφόσον καθοριστεί μόνο το όνομα του αρχείου προς εκτέλεση.

Η διαδικασία εκκίνησης (boot) ενός Η/Υ πολύ συνοπτικά έχει ως εξής: αρχικά εκτελούνται κάποιες ρουτίνες αυτοελέγχου του υλικού (π.χ. έλεγχος μνήμης) και υποτυπώδους προγραμματισμού των συσκευών (π.χ. timers, controllers). Οι ρουτίνες αυτές βρίσκονται στην μνήμη ROM. Κατόπιν, διαβάζεται ο τομέας εκκίνησης (boot sector) από τον δίσκο ή την δισκέτα εκκίνησης (boot disk) ο οποίος περιέχει πληροφορίες για την διαμόρφωση του δίσκου και έναν προφορτωτή (preloader). Ο προφορτωτής μεταφέρεται στην μνήμη RAM και αναλαμβάνει να μεταφέρει τον κύριο φορτωτή (loader) από τον δίσκο εκκίνησης στην RAM. Στη συνέχεια, ο loader αρχίζει το «φόρτωμα» του λειτουργικού συστήματος στη μνήμη του υπολογιστή. Μόλις τελειώσει αυτή η διαδικασία αναλαμβάνει τον έλεγχο του μηχανήματος ο διερμηνευτής εντολών (command interpreter), δηλαδή, το γνωστό μας αρχείο COMMAND.COM και εμφανίζεται το σύμβολο ετοιμότητας (prompter) του λειτουργικού συστήματος MS-DOS. Αν υπάρχουν στο κύριο ευρετήριο (root) του δίσκου εκκίνησης τα αρχεία AUTOEXEC.BAT και CONFIG.SYS τα οποία περιέχουν κάποιες επιπλέον ρυθμίσεις για τις συσκευές του συστήματος τότε αυτές οι ρυθμίσεις θα λάβουν χώρα λίγο πριν την εμφάνιση του συμβόλου ετοιμότητας.

Βέβαια, αν λόγω κάποιας βλάβης κάτι δεν πάει καλά, εμφανίζονται τα ανάλογα μηνύματα ή στην χειρότερη περίπτωση το μηχάνημα μας ειδοποιεί με μια ακολουθία από ήχους (beep) ανάλογα με το πρόβλημα. Το εγχειρίδιο του κατασκευαστή της κάθε μητρικής πλακέτας περιέχει τις σχετικές λεπτομέρειες.

1.1.4. Γλώσσες προγραμματισμού

Η πραγματοποίηση (δημιουργία) ενός προγράμματος με τη μορφή σειρών (συστοιχιών) δυαδικών ψηφίων σύμφωνα με τη συμβατότητα ενός συγκεκριμένου υπολογιστή λέγεται προγραμματισμός σε **κώδικα μηχανής** (machine language).

Η εισαγωγή αυτών των προγραμμάτων στη μνήμη του Η/Υ αρχικά γινόταν με τη βοήθεια κάποιων διακοπτών με 2 θέσεις (μια για το '0' και η άλλη για το '1'). Για διευκόλυνση και για μείωση του κόπου δημιουργίας και εισαγωγής των προγραμμάτων εφευρέθηκαν συμφωνίες που προσδιορίζουν συμβολικά τις πράξεις που εκτελεί ο υπολογιστής. Αυτές οι συμφωνίες λέγονται γλώσσες προγραμματισμού.

Οι πρώτες γλώσσες προγραμματισμού ανήκουν στην κατηγορία των **συμβολικών γλωσσών** (assembly language). Η κάθε πράξη που εκφράζεται δια μέσου μιας τέτοιας γλώσσας αντιστοιχεί σε μια εντολή του Η/Υ. Αυτός είναι ο λόγος που αυτές οι γλώσσες ονομάζονται γλώσσες προγραμματισμού χαμηλού επιπέδου.

Π.χ. για την πράξη C ← A + B έχουμε:

κώδικας μηχανής ή γλώσσα μηχανής (machine language)	συμβολική γλώσσα (assembly language).
0 0 1 | 0 0 0 1 0	LDA A
0 1 0 | 0 0 0 1 1	ADD B
1 0 0 | 0 0 1 0 0	STO C

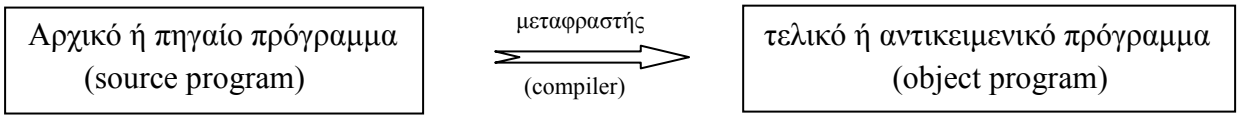
Για να είναι δυνατή η εκτέλεση των προγραμμάτων που γράφτηκαν σε συμβολική γλώσσα είναι απαραίτητο αυτά να μεταφραστούν σε κώδικα μηχανής. Γι' αυτό το σκοπό χρησιμοποιείται ένα πρόγραμμα που λέγεται συμβολομεταφραστής (assembler). Αυτό το πρόγραμμα χρησιμοποιεί σαν δεδομένα τα προγράμματα σε συμβολική γλώσσα και παράγει σαν αποτέλεσμα προγράμματα σε κώδικα μηχανής.

Από το γεγονός ότι μια συμβολική γλώσσα είναι άρρηκτα συνδεδεμένη με το σύνολο των εντολών (instruction set) ενός επεξεργαστή πηγάζουν δύο σημαντικά χαρακτηριστικά αυτών των γλωσσών. Πρώτον, έχουν το μεγάλο προτέρημα να προσφέρουν στον προγραμματιστή την δυνατότητα να γράψει πολύ ταχύ και περιεκτικό κώδικα. Και δεύτερον, έχουν, εξ' ορισμού, το σημαντικό μειονέκτημα ότι αυτός ο κώδικας δεν είναι κατανοητός παρά μόνο από τον συγκεκριμένο τύπο Η/Υ. Αν, δηλαδή, θέλουμε μια εφαρμογή να τρέχει σε περισσότερες πλατφόρμες τότε θα πρέπει να μπούμε στον κόπο να γράψουμε τον αντίστοιχο κώδικα σε γλώσσα χαμηλού επιπέδου προγραμματισμού για κάθε μια από τις διαφορετικές πλατφόρμες.

Το επόμενο βήμα στην εξελικτική πορεία του προγραμματισμού το αποτελεί η εκπόνηση (δημιουργία) γλωσσών ανεξάρτητων από τη μηχανή (Η/Υ), προσανατολισμένων προς κάποια συγκεκριμένη κατηγορία προβλημάτων (γλώσσες προγραμματισμού υψηλού επιπέδου – high level programming languages)

BASIC	LET	C = A + B
FORTRAN		C = A + B
(ADA) PASCAL		c := a + b;
C		c = a + b;

Η μετάφραση των προγραμμάτων από τις γλώσσες υψηλού επιπέδου σε γλώσσα μηχανής ή κώδικα μηχανής πραγματοποιείται από προγράμματα που λέγονται **μεταφραστές** (compilers) ή **μεταγλωττιστές**.



Ένας άλλος τύπος μεταφραστή είναι ο **διερμηνευτής** (ή αλλιώς διερμηνέας ή ερμηνευτής) (interpreter) που δεν παράγει αντικειμενικό πρόγραμμα αλλά μεταφράζει μία προς μία τις εντολές και τις δίνει για εκτέλεση. Οι γλώσσες προγραμματισμού υψηλού επιπέδου εμφανίστηκαν το 1955. Πρώτη ήταν η FORTRAN (FORmula TRANslator) που είχε εμπνευστή τον John Backus.

1.2. Η παράσταση της πληροφορίας (των τύπων των δεδομένων) στον Η/Υ

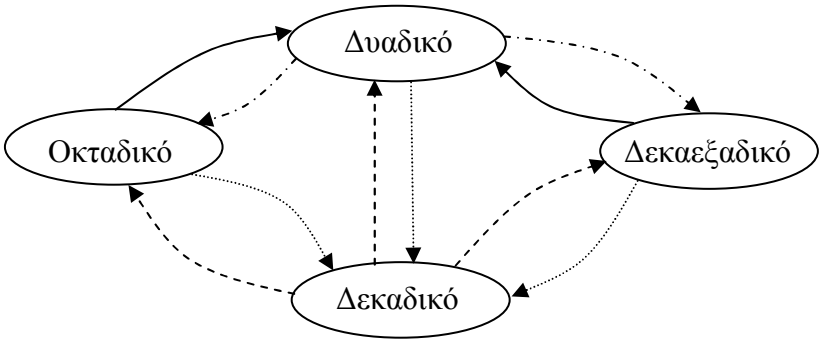
1.2.1. Συστήματα αρίθμησης και μετατροπές από το ένα σύστημα στο άλλο

Πίνακας που εμφανίζει την αντιστοιχία στην παράσταση των 21 πρώτων αριθμών στα διάφορα αριθμητικά συστήματα.

Δεκαδικό	Δυαδικό	Οκταδικό	Δεκαεξαδικό
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
…	…	…	…

Η μετατροπή ενός αριθμού από το δυαδικό στο οκταδικό ή το δεκαεξαδικό σύστημα αρίθμησης (και αντίστροφα) γίνεται απευθείας, χωρίς να χρειαστεί ενδιάμεσα το πέρασμα από το δεκαδικό σύστημα.

Για να μετατρέψουμε έναν αριθμό από το οκταδικό ή το δεκαεξαδικό σύστημα στο δυαδικό, αρκεί να αντικαταστήσουμε κάθε ψηφίο τού αριθμού τού πρώτου συστήματος με την ακολουθία δυαδικών ψηφίων που αντιστοιχούν στο ψηφίο αυτό. Πάντα αντικαθιστούμε ένα ψηφίο οκταδικού αριθμού με τρία δυαδικά ψηφία ενώ ένα ψηφίο δεκαεξαδικού αριθμού με τέσσερα δυαδικά ψηφία.



Σχήμα 2. Διακρίνουμε ότι δεν μπορούμε να κάνουμε απ' ευθείας μετατροπή από το οκταδικό στο δεκαεξαδικό σύστημα αρίθμησης.

π.χ. Μετατροπή από οκταδικό σε δυαδικό

2701_8

$010 \quad 111 \quad 000 \quad 001_2$

π.χ. Μετατροπή από δεκαεξαδικό σε δυαδικό

$3F10A9_H$

$0011 \quad 1111 \quad 0001 \quad 0000 \quad 1010 \quad 1001_2$

Με παρόμοιο τρόπο στην αντίστροφη περίπτωση, δηλαδή, για να μετατρέψουμε έναν τού αριθμού σε τριάδες (ή τετράδες) ξεκινώντας από τα λιγότερο σημαντικά ψηφία και αντικαθιστούμε την κάθε ομάδα ψηφίων με το αντίστοιχο ψηφίο τού νέου συστήματος.

π.χ. Μετατροπή από δυαδικό σε οκταδικό

$110 \quad 101 \quad 001 \quad 011_2$

$6 \; 5 \; 1 \; 3 \; _8$

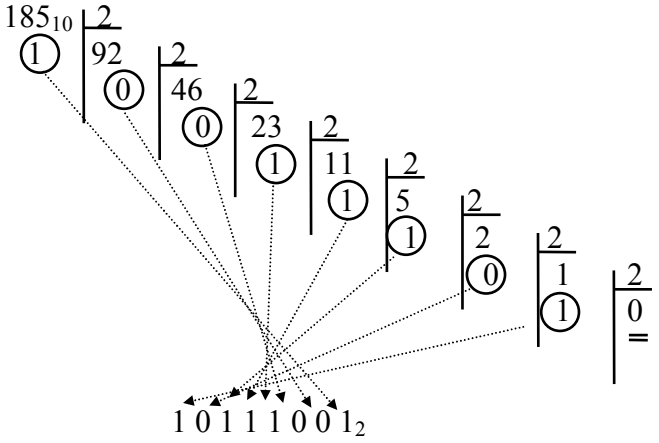
π.χ. Μετατροπή από δυαδικό σε δεκαεξαδικό

$1110 \quad 0100 \quad 0000 \quad 1100 \quad 0010_2$

$E \; 4 \; 0 \; C \; 2 \; _H$

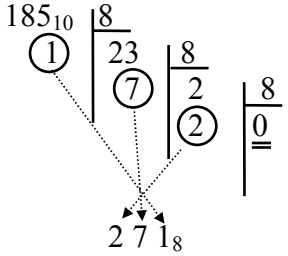
Για να μετατρέψουμε ένα αριθμό από τη δεκαδική μορφή στη δυαδική, εκτελούμε διαδοχικές διαιρέσεις του αριθμού με το 2 μέχρι να λάβουμε πηλίκο μηδέν. Η ακολουθία των υπολοίπων των διαιρέσεων με την αντίστροφη σειρά αποτελεί την παράσταση του αριθμού αυτού στο δυαδικό σύστημα αρίθμησης.

π.χ. Μετατροπή από δεκαδικό σε δυαδικό

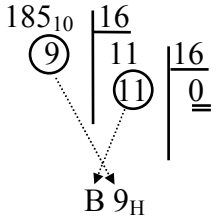


Με ανάλογο τρόπο για να μετατρέψουμε έναν αριθμό από το δεκαδικό σύστημα στο οκταδικό ή δεκαεξαδικό διαιρούμε δια της αντίστοιχης βάσης 8 ή 16 μέχρι να λάβουμε πηλίκο μηδέν. Ομοίως η ακολουθία των υπολοίπων των διαιρέσεων με την αντίστροφη σειρά αποτελεί την παράσταση του αριθμού αυτού στο ζητούμενο σύστημα αρίθμησης.

π.χ. Μετατροπή από δεκαδικό σε οκταδικό



π.χ. Μετατροπή από δεκαδικό σε δεκαεξαδικό



Για να μετατρέψουμε (υπολογίσουμε) ένα αριθμό εκφρασμένο στο δυαδικό ή οκταδικό ή δεκαεξαδικό σύστημα υπολογίζουμε το άθροισμα των γινομένων του κάθε ψηφίου του αριθμού με την αξία που έχει στη θέση που βρίσκεται.

π.χ. Μετατροπή από δυαδικό σε δεκαδικό

$2^5\,2^4\,2^3\,2^2\,2^1\,2^0$

$1\;\;0\;\;1\;\;0\;\;1\;\;1_2 = 1\cdot2^0 + 1\cdot2^1 + 0\cdot2^2 + 1\cdot2^3 + 0\cdot2^4 + 1\cdot2^5 = 1 + 2 + 8 + 32 = 41_{10}$

π.χ. Μετατροπή από οκταδικό σε δεκαδικό

$8^2\,8^1\,8^0$

$2\;\;5\;\;3_8 = 3\cdot8^0 + 5\cdot8^1 + 2\cdot8^2 = 3 + 40 + 128 = 171_{10}$

π.χ. Μετατροπή από δεκαεξαδικό σε δεκαδικό

$16^2\,16^1\,16^0$

$2\;\;\;A\;\;\;F_H = 15\cdot16^0 + 10\cdot16^1 + 2\cdot16^2 = 15 + 160 + 512 = 687_{10}$

Εκτός όμως από τους αριθμούς είναι αναγκαίο να μπορούμε να παραστήσουμε σε δυαδική μορφή και τα γράμματα του αλφάβητου καθώς και τα διάφορα σημεία στίξης και άλλα σύμβολα (εκτυπώσιμα ή μη). Έτσι οι επιστήμονες για λόγους συμβατότητας κατάληξαν σε κάποια στάνταρτ που έχουν ευρεία εφαρμογή. Οι πιο διαδεδομένοι είναι: ο κώδικας ASCII (American Standards Committee on Information Interchange) που για το κάθε σύμβολο χρησιμοποιεί επτά δυαδικά ψηφία και ο επεκταμένος κώδικας ASCII που χρησιμοποιεί οκτώ όπως και ο κώδικας EBCDIC (Extended Binary Coded Decimal Interchange Code).

Σε κάποιες περιπτώσεις για την παράσταση αριθμών χρησιμοποιείται και ο κώδικας BCD (Binary Coded Decimal) που χρησιμοποιεί τέσσερα δυαδικά ψηφία για το κάθε δεκαδικό ψηφίο.

Τύπος δεδομένου είναι ο συνδυασμός από:

(α) ένα σύνολο τιμών που παίρνει το δεδομένο, και

(β) το σύνολο των ιδιοτήτων αυτών των τιμών.

1.2.2. Απλοί ή στοιχειώδεις τύποι δεδομένων:

α) ακέραιος τύπος (integer)
β) πραγματικός τύπος (real)
γ) λογικός τύπος (boolean)
δ) χαρακτήρες (characters)

που χρησιμοποιούνται στις περισσότερες γλώσσες προγραμματισμού.

Σύνθετοι τύποι είναι αυτοί που ορίζονται από άλλους τύπους απλούς ή σύνθετους.

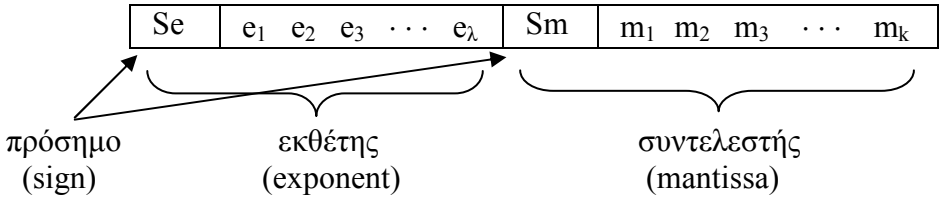
Οι ακέραιοι που παριστάνονται με ν δυαδικά ψηφία θα είναι στο διάστημα $[-2^{\nu-1}, 2^{\nu-1}-1]$ (π.χ. για $\nu = 16$ έχουμε από –32768 μέχρι 32767).

$$19{,}5 \longrightarrow \quad 10011{,}1 = 0{,}100111 \times 2^5 = 0{,}100111 \times 2^{101}$$

Η τεχνική αυτή είναι η βάση της παράστασης των αριθμών με κινητή υποδιαστολή (floating point) διότι κάθε αριθμός μπορεί να πάρει τη μορφή:

$$0, m_1 m_2 m_3 \cdots m_\kappa \times R^{e_1 e_2 e_3 \cdots e_\lambda}$$

Ο παραπάνω είναι δηλαδή ο γενικός τύπος ενός πραγματικού αριθμού εκφρασμένου με τη μορφή της κινητής υποδιαστολής.



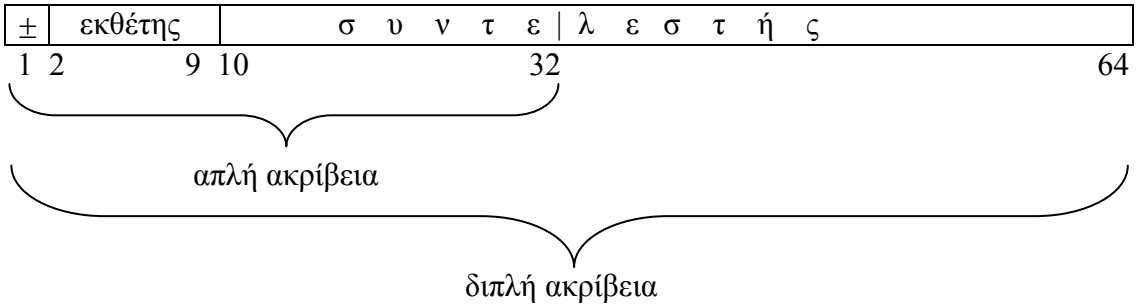
Η διαφορά μεταξύ της προσέγγισης και του ακριβούς αριθμού λέγεται σφάλμα προσέγγισης (round-off error)

Στην πράξη έχουμε: (α) απλής ακρίβειας (single precision)

(β) διπλής ακρίβειας (double precision)

και επιλέγονται ανάλογα με τις ανάγκες.

Ο τρόπος αποθήκευσης πραγματικών αριθμών στους μικροϋπολογιστές είναι:



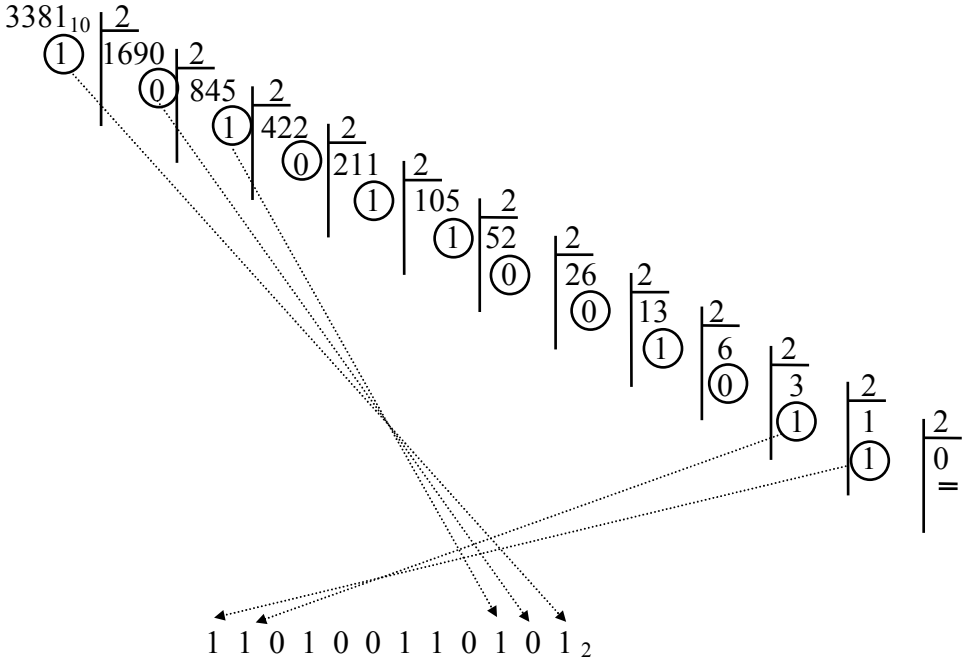
2^{23}-1 = 8.388.607 δηλ. 6-7 σημαντικά ψηφία, από το $10^{-38} \div 10^{38}$ επειδή $2^{128-1} \approx 10^{38}$.

1.2.3. Μετατροπή ενός δεκαδικού αριθμού με ψηφία και δεξιά της υποδιαστολής σε δυαδικό με ψηφία δεξιά της υποδιαστολής

Ένας τρόπος, για τη μετατροπή ενός αριθμού εκφρασμένου στο δεκαδικό σύστημα αρίθμησης με ψηφία και δεξιά της υποδιαστολής στον αντίστοιχό του στο δυαδικό σύστημα αρίθμησης, με ψηφία δεξιά της υποδιαστολής, είναι να πολλαπλασιάσουμε ολόκληρο τον πρώτο αριθμό (μαζί με το τμήμα δεξιά της υποδιαστολής) με τη βάση του δευτέρου συστήματος αρίθμησης τόσες φορές, όσα ψηφία (ακρίβεια) επιθυμούμε να έχουμε μετά την υποδιαστολή. Μετά εκτελούμε τη μετατροπή κανονικά μόνον του ακέραιου τμήματος του γινομένου που βρήκαμε σαν αποτέλεσμα. Μετά την μετατροπή δεν πρέπει να ξεχάσουμε να βάλουμε την υποδιαστολή στη σωστή θέση, μετρώντας τα ψηφία ακρίβειας του τμήματος δεξιά της υποδιαστολής.

π.χ. Μετατροπή από δεκαδικό (που δεν είναι ακέραιος) σε δυαδικό

Έστω ότι έχουμε το αριθμό $211{,}352_{10}$ και επιθυμούμε να τον εκφράσουμε στο δυαδικό σύστημα αρίθμησης με τέσσερα ψηφία ακρίβεια μετά την υποδιαστολή. Άρα λοιπόν σαν πρώτο βήμα θα κάνουμε τον πολλαπλασιασμό $211{,}352_{10} \times 2^4 = 3381{,}632$ και μετά μετατρέπουμε μόνο το ακέραιο τμήμα όπως είδαμε στα προηγούμενα.

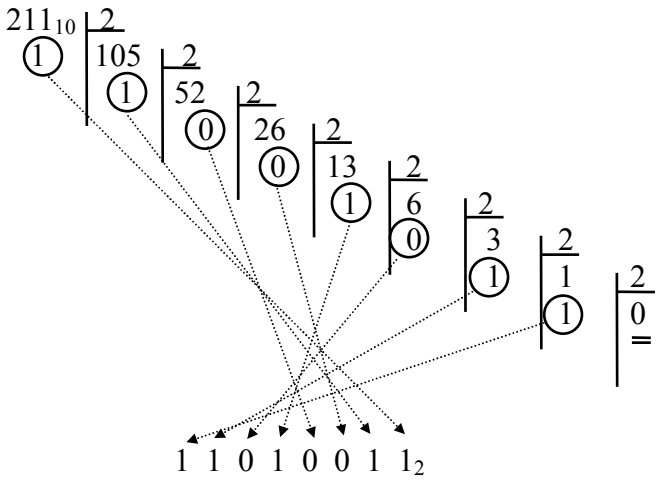


οπότε μένει να τοποθετήσουμε την υποδιαστολή μετρώντας τέσσερα ψηφία από τα δεξιά του δυαδικού αριθμού, δηλαδή έχουμε:

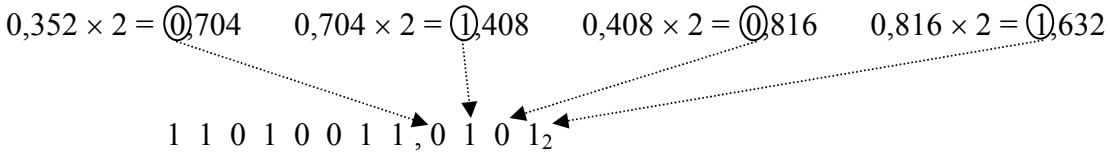
$$1\ 1\ 0\ 1\ 0\ 0\ 1\ 1{,}0\ 1\ 0\ 1_2$$

Ένας άλλος τρόπος είναι να μετατρέψουμε χωριστά το ακέραιο τμήμα όπως ήδη γνωρίζουμε και χωριστά το δεκαδικό τμήμα του αριθμού. Για τη μετατροπή του δεκαδικού τμήματος το πολλαπλασιάζομε με τη βάση (δηλαδή με το 2) διαδοχικά, παίρνοντας κάθε φορά το ακέραιο τμήμα του κάθε γινομένου (είτε 0, είτε 1) δημιουργώντας τα δυαδικά ψηφία του αριθμού μετά την υποδιαστολή.

Για το παραπάνω παράδειγμα με τον αριθμό $211{,}352_{10}$ θα είχαμε:



Αυτό είναι το ακέραιο τμήμα, ενώ το τμήμα δεξιά της υποδιαστολής υπολογίζεται ως εξής:



1.3. Η έννοια του αλγόριθμου

Αλγόριθμος είναι μια ακολουθία από αυστηρά καθορισμένα βήματα πού έχει σκοπό την καθοδήγηση προκειμένου να διεκπεραιωθεί μια εργασία. Αποτελεί δηλαδή τις οδηγίες (εντολές) που πρέπει να εκτελεστούν και υποδεικνύει την ακριβή σειρά.

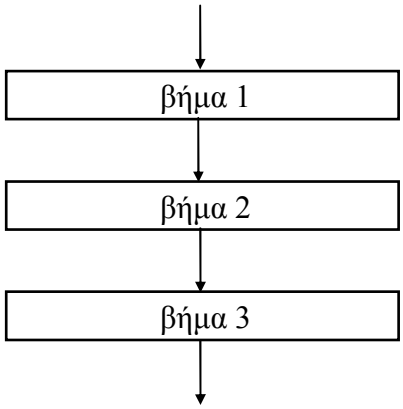
Στην διατύπωση ενός αλγόριθμου θα πρέπει να ορίζεται με σαφήνεια η αρχή και το τέλος της ακολουθίας των βημάτων που αφορούν την διεκπεραίωση της συγκεκριμένης εργασίας, που η ίδια μπορεί να αποτελεί μέρος μιας ευρύτερης διαδικασίας.

Επιπλέον, στη σύνταξη ενός αλγόριθμου θα πρέπει να τηρούνται κάποιοι κανόνες όπως:

* κάθε βήμα να είναι σαφώς διαχωρισμένο από τα υπόλοιπα
* ένα βήμα να αποτελείται από μία ή από περισσότερες στοιχειώδεις ενέργειες, οι οποίες αναφέρονται με τη σειρά που πρόκειται να πραγματοποιηθούν
* μια στοιχειώδης ενέργεια εκφράζεται με ένα ρήμα και ένα αντικείμενο
* τα βήματα πρέπει να προβλέπουν κάθε ενδεχόμενο
* το σύνολο των βημάτων να ολοκληρώνει την εργασία

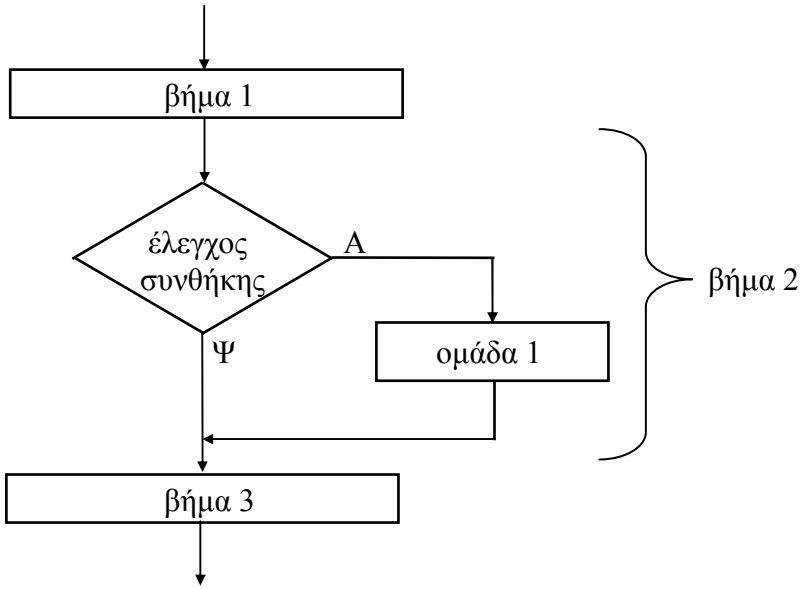
1.3.1. Δομές αλγορίθμων

Ακολουθιακή δομή (απλή διαδοχή βημάτων) λέγεται η αλγοριθμική δομή, στην οποία τα βήματα εκτελούνται με τη σειρά που περιγράφονται. Η ροή εκτέλεσης μιας ακολουθιακής δομής αποδίδεται παραστατικά με ένα διάγραμμα, όπως αυτό που ακολουθεί:

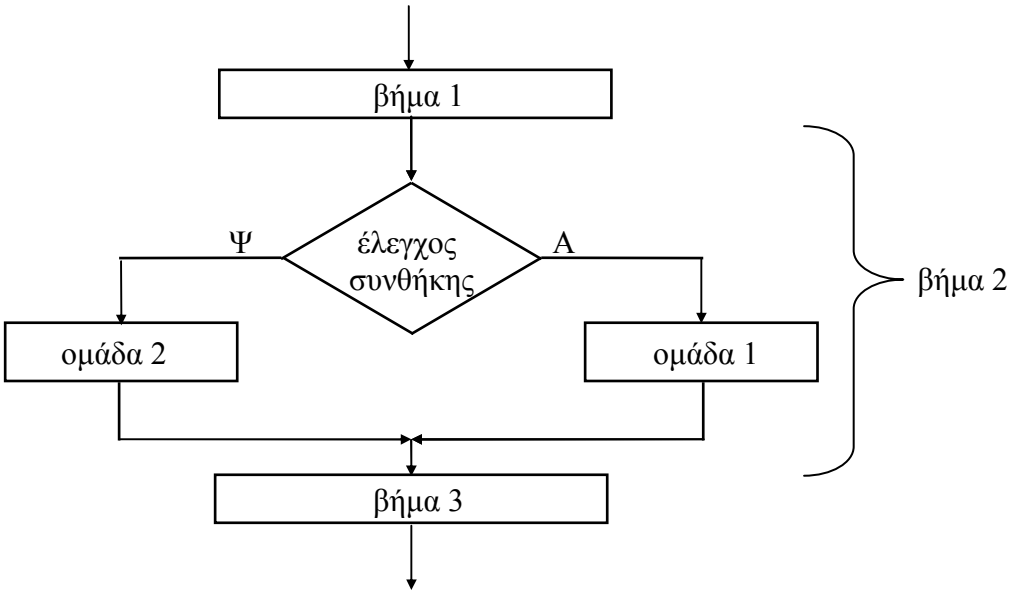


Σχήμα 3. Ένα τέτοιο διάγραμμα λέγεται διάγραμμα ροής (flowchart) ή λογικό διάγραμμα.

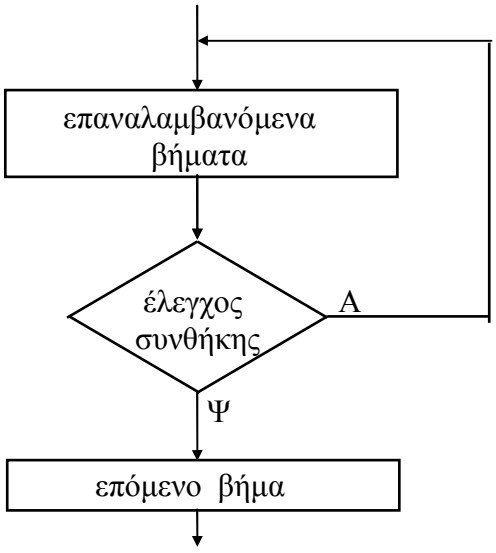
Δομές ελέγχου (ΑΝ … ΤΟΤΕ … / ΑΝ … ΤΟΤΕ … ΔΙΑΦΟΡΕΤΙΚΑ …) λέγονται οι αλγοριθμικές δομές, στις οποίες ελέγχεται η κατάσταση κάποιας συνθήκης (αληθής ή ψευδής) και ανάλογα εκτελείται ή όχι μια ομάδα εντολών. Η ροή εκτέλεσης μιας δομής ελέγχου ΑΝ … ΤΟΤΕ … αποδίδεται παραστατικά με ένα διάγραμμα, όπως αυτό που ακολουθεί:



Ενώ για τη δομή ΑΝ … ΤΟΤΕ … ΔΙΑΦΟΡΕΤΙΚΑ … (με εναλλακτική ομάδα) το λογικό διάγραμμα είναι:



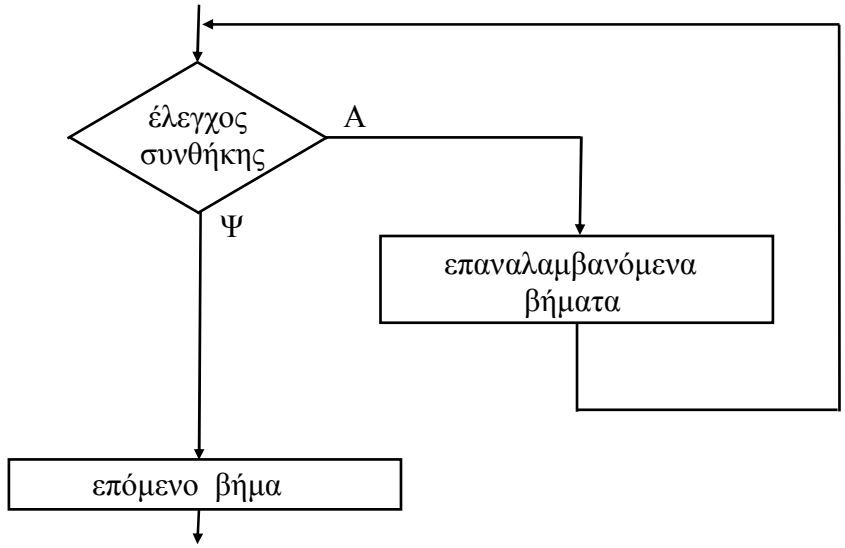
Δομή επανάληψης με συνθήκη λέγεται η αλγοριθμική δομή, στην οποία τα επαναλαμβανόμενα βήματα εκτελούνται ξανά αν η συνθήκη είναι αληθής.



Σε μια δομή επανάληψης λέμε ότι η ροή εκτέλεσης του αλγόριθμου δημιουργεί ένα βρόχο (loop), που στο εσωτερικό του περιλαμβάνει την ομάδα των βημάτων που επαναλαμβάνονται.

Ο έλεγχος της συνθήκης μιας δομής επανάληψης μπορεί να προηγείται των εντολών του βρόχου ή να έπεται. Στο προηγούμενο σχήμα έχουμε το διάγραμμα ροής μιας δομής επανάληψης όπου ο έλεγχος της συνθήκης γίνεται μετά από κάθε εκτέλεση των επαναλαμβανόμενων εντολών. Αυτό σημαίνει ότι οι εντολές που απαρτίζουν τον βρόχο θα εκτελεστούν τουλάχιστον μία φορά, ακόμη και στην περίπτωση που η συνθήκη είναι εξαρχής ψευδής. Αυτό δεν ισχύει στις δομές επανάληψης στις οποίες πρώτα ελέγχεται η συνθήκη και

ανάλογα με το αποτέλεσμα εκτελούνται οι εντολές του βρόχου ή όχι. Μια τέτοια δομή παρουσιάζεται στο επόμενο σχήμα.



Σε έναν αλγόριθμο που περιλαμβάνει δομές ελέγχου και επανάληψης, η ροή εκτέλεσης δεν ακολουθεί μια μοναδική και σταθερή πορεία, όπως συμβαίνει στην ακολουθιακή δομή, αλλά μπορεί να διακλαδώνεται μεταπηδώντας σε άλλο βήμα του αλγόριθμου. Διακλάδωση μπορεί να έχουμε και σε άλλες περιπτώσεις, όπως η διακλάδωση υπό συνθήκη, όταν η αλλαγή της ροής εκτέλεσης εξαρτάται από τον έλεγχο κάποιας συνθήκης.

1.4. Η εξέλιξη των υπολογιστών

συνεχίζεται μέχρι σήμερα με ασύλληπτη ταχύτητα. Ανάλογα με την τεχνολογία που χρησιμοποιήθηκε τη χωρίζουμε σε 5 περιόδους που τις λέμε γενιές.

1 γενιά 1950 ÷ 1959
- ηλεκτρονικές λυχνίες
- τεράστιος όγκος, αναξιοπιστία
- μεγάλη κατανάλωση
- 50.000 ÷ 200.000 πράξεις / sec

2 γενιά 1959 ÷ 1963
- χρήση ημιαγωγών στην κατασκευή λογικών κυκλωμάτων
- transistors - σιδηρομαγνητικές μνήμες
- πρωτόγονα λειτουργικά συστήματα
- 50.000 ÷ 200.000 πράξεις / sec

3 γενιά 1963 ÷ 1970
- λειτουργικά συστήματα
- αύξηση υπολογιστικής ισχύος
- ολοκληρωμένα κυκλώματα SSI, MSI, LSI
- μείωση όγκου
- μείωση κόστους

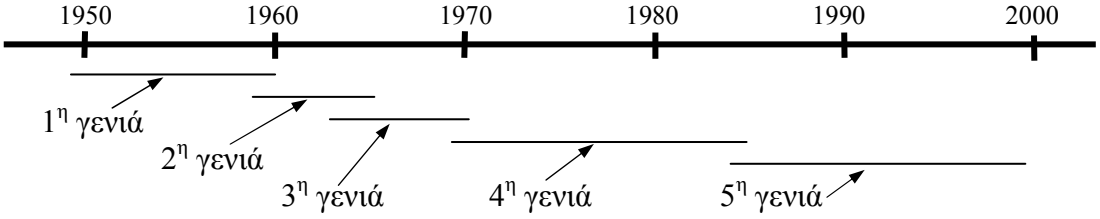
4 γενιά 1971 ÷ 1985

εμφάνιση Basic, Pascal
πολλοί επεξεργαστές (multiprocessor)
IC 200.000 transistors σε 10 mm^2

5 γενιά μετά το 1985

ολοκληρωμένα κυκλώματα VLSI, WSI
ταχύτητα μεγαλύτερη 10 ÷ 20 εκατομμύρια εντολές / sec
προσανατολισμένα στη τεχνητή νοημοσύνη (artificial intelligence) και
στα έμπειρα συστήματα (expert systems)

Στην υπάρχουσα βιβλιογραφία υπάρχουν σημαντικές διαφορές σχετικά με το πότε αρχίζει και πότε τελειώνει η κάθε γενιά. Αυτό είναι λογικό, διότι, μετά την εμφάνιση μιας τεχνολογίας δεν καταργούταν ακαριαία η προηγούμενη. Συνυπήρχαν για κάποια περίοδο και έτσι υπάρχουν χρονικές επικαλύψεις όπως φαίνεται στο σχήμα που ακολουθεί.



1.5. Τα στάδια της δημιουργίας ενός προγράμματος

Ο προγραμματισμός είναι μια συστηματική δραστηριότητα, η οποία είναι οργανωμένη συνήθως στα εξής βήματα:

1. Ανάλυση του προβλήματος – καθορισμός του αλγόριθμου επίλυσης
2. Σχεδίαση του προγράμματος – καθορισμός της δομής του προγράμματος και των δεσμών μεταξύ των συστατικών τμημάτων του στο λογικό επίπεδο
3. Κωδικοποίηση του προγράμματος – μεταφορά (διασκευή) του λογικού σχεδίου σε μια γλώσσα προγραμματισμού
4. Ολοκλήρωση και έλεγχος του προγράμματος
5. Χρήση και συντήρηση του προγράμματος (νέες βελτιωμένες εκδόσεις)

2. Η γλώσσα προγραμματισμού C

2.1. Εισαγωγή

2.1.1. Γενικά περί της γλώσσας προγραμματισμού C

Η C είναι μια γλώσσα προγραμματισμού γενικής χρήσης που χαρακτηρίζεται από οικονομία στην έκφραση, σύγχρονη ροή ελέγχου και δομές δεδομένων, και ένα πλούσιο σύνολο τελεστών.

Η C σχεδιάστηκε αρχικά για το λειτουργικό σύστημα UNIX, και εφαρμόστηκε σ' αυτό σ' έναν υπολογιστή DEC PDP-11, από τον Dennis Ritchie.

Το 1978 εμφανίστηκε το πρώτο βιβλίο που καθόριζε τη γλώσσα C με τίτλο *'The C Programming Language"* των Brian Kernighan και Dennis Ritchie. Η δημοτικότητα που γνώριζε η νέα αυτή γλώσσα ήταν ολοένα και μεγαλύτερη και έκανε αισθητή την ανάγκη για έναν πιο ακριβή και ολοκληρωμένο ορισμό της γλώσσας.

Το 1983, το Αμερικανικό Εθνικό Ινστιτούτο Προτύπων (American National Standards Institute - ANSI) σύστησε μια επιτροπή με σκοπό να παράγει έναν σύγχρονο, ακριβή και πλήρη ορισμό της γλώσσας C, ανεξάρτητο από μηχανήματα, διατηρώντας όμως το πνεύμα της γλώσσας. Ο ορισμός που προτάθηκε από την επιτροπή, εγκρίθηκε το 1988, και προέκυψε το πρότυπο ANSI C.

Η C είναι ευχάριστη, εκφραστική, και ευέλικτη γλώσσα για μεγάλη ποικιλία προγραμμάτων. Είναι εύκολη στην εκμάθηση, και σας ταιριάζει καλύτερα όσο αυξάνεται η εμπειρία σας μ' αυτήν. Η C έχει αποκαλεστεί «γλώσσα προγραμματισμού συστημάτων», διότι είναι χρήσιμη στην σύνταξη μεταγλωττιστών και λειτουργικών συστημάτων. Πολλές σημαντικές ιδέες της C προέρχονται από την γλώσσα BCPL, που αναπτύχθηκε από τον Martin Richards. Η επιρροή της BCPL πάνω στην C πηγάζει έμμεσα από την γλώσσα B, οποία γράφτηκε από τον Ken Thompson το 1970 για το πρώτο σύστημα UNIX στον DEC PDP-7. Η C όμως διαθέτει ποικιλία τύπων δεδομένων σε αντίθεση με τις προγενέστερες της BCPL και B που είναι γλώσσες άτυπες «χωρίς τύπους».

Η C δεν παρέχει τελεστές για τον άμεσο χειρισμό σύνθετων αντικειμένων όπως τα αλφαριθμητικά, τα σύνολα, οι λίστες, ή οι πίνακες. Δεν υπάρχουν τελεστές για τον χειρισμό ενός ολόκληρου πίνακα ή αλφαριθμητικού, αν και οι δομές μπορούν να αντιγράφονται σαν ενότητα.

Στα μειονεκτήματα της γλώσσας θα μπορούσαμε να συγκαταλέξουμε ότι δεν διαθέτει καμιά ευκολία κατανομής μνήμης, καθώς και ότι δεν προσφέρει ευκολίες εισόδου / εξόδου (δεν έχει π.χ. READ, WRITE), ούτε ενσωματωμένες μεθόδους προσπέλασης αρχείων (παρέχονται όμως από ρητά καλούμενες συναρτήσεις). Επίσης, η C προσφέρει μόνο απλή ροή ελέγχου (single thread) … όχι πολυπρογραμματισμό, όχι παράλληλες λειτουργίες.

Αν και όλες αυτές οι ελλείψεις μοιάζουν με σοβαρή ανεπάρκεια, το μέτριο μέγεθος της γλώσσας έχει πραγματικά οφέλη.

Το μάθημά μας θα βασιστεί στην δεύτερη έκδοση του βιβλίου *"The C Programming Language"* των Kernighan και Ritchie, που δημοσιεύθηκε το 1988, και περιγράφει την C όπως ορίζεται από το πρότυπο ANSI. Βέβαια η μελέτη του παραπάνω βιβλίου προϋποθέτει κάποια εξοικείωση με τις βασικές έννοιες του προγραμματισμού όπως μεταβλητές, εντολές απόδοσης τιμής, βρόχους και συναρτήσεις.

Ο βασικός στόχος του μαθήματος δεν είναι τόσο η εκμάθηση της γλώσσας προγραμματισμού C, όσο η κατανόηση και η αφομοίωση των αρχών που διέπουν τον δομημένο προγραμματισμό των ηλεκτρονικών υπολογιστών. Έτσι, όπου είναι δυνατόν θα δίνουμε παραδείγματα χρήσιμων αλγόριθμων και τις αρχές για καλό στυλ και ορθό προγραμματισμό.

2.1.2. Μερικές εξηγήσεις τώρα για το ίδιο το πρόγραμμα

Ένα πρόγραμμα σε C, ανεξαρτήτως μεγέθους, αποτελείται από συναρτήσεις (functions) και μεταβλητές (variables). Μια συνάρτηση περιέχει εντολές (statements).
Η main() είναι μια ειδική περίπτωση συνάρτησης που πρέπει να υπάρχει σε όλα τα προγράμματα. Από αυτήν ξεκινάει το εκτελέσιμο πρόγραμμα και μέσα σε αυτήν βρίσκονται οι κλήσεις για όλες τις άλλες συναρτήσεις.
Ένας τρόπος μεταφοράς δεδομένων ανάμεσα στις συναρτήσεις είναι με τη χρήση παραμέτρων κατά την κλήση των συναρτήσεων, οι οποίες παράμετροι ονομάζονται ορίσματα (arguments).

Όπως είπαμε παραπάνω η γλώσσα δεν προσφέρει ευκολίες εισόδου / εξόδου. Λίγα είναι όμως τα προγράμματα που δεν χρειάζονται να πάρουν δεδομένα από το πληκτρολόγιο (στάνταρ είσοδο) ή να στείλουν πληροφορίες στην οθόνη (στάνταρ έξοδο). Γι' αυτό λοιπόν στα περισσότερα προγράμματα συμπεριλαμβάνουμε στον κώδικά μας ένα αρχείο επικεφαλίδα (header), το «stdio.h», το οποίο περιέχει όλες τις συμβολικές σταθερές και τις συναρτήσεις εισόδου / εξόδου σύμφωνα με το πρότυπο ANSI. Αυτή η πρότυπη βιβλιοθήκη εισόδου / εξόδου ενεργοποιείται όταν εισάγουμε στην αρχή του προγράμματος την παρακάτω εντολή - οδηγία (directive) του προεπεξεργαστή:

```
#include <stdio.h>
```

Αν θέλουμε να εισάγουμε κι άλλες πρότυπες βιβλιοθήκες όπως την «string.h» ή την «math.h» δεν έχουμε παρά να εισάγουμε ακολούθως και τις αντίστοιχες γραμμές:

```
#include <string.h>
#include <math.h>
```

Ακόμα, αν συντάξουμε εμείς κάποιο αρχείο επικεφαλίδα, για παράδειγμα το myheader.h, μπορούμε να το συμπεριλάβουμε στα προγράμματά μας ως εξής:

```
#include "myheader.h"
```

2.1.3. Οι τύποι δεδομένων και προσδιοριστές που υποστηρίζει η γλώσσα C

Οι απλοί ή στοιχειώδεις τύποι δεδομένων που υποστηρίζει η γλώσσα είναι ο χαρακτήρας (char), ο ακέραιος (int), ο πραγματικός απλής ακρίβειας (float) και ο πραγματικός διπλής ακρίβειας (double). Επίσης, υπάρχουν μια σειρά από **προσδιοριστές (qualifiers)** που προσδίδουν στους παραπάνω τύπους κάποια ιδιαίτερα χαρακτηριστικά. Για παράδειγμα, έχουμε τα short και long τα οποία είναι προσδιοριστές για ακεραίους και πραγματικούς (double) με την βοήθεια των οποίων η γλώσσα μπορεί να υποστηρίξει μέχρι τρία διαφορετικά μεγέθη για τον κάθε τύπο από τους παραπάνω εφόσον βέβαια υποστηρίζονται και από το υλικό (hardware). Επειδή τα μεγέθη διαφέρουν από υλοποίηση σε υλοποίηση εμείς θα δίνουμε τις ενδεικτικές τιμές για τους μικροϋπολογιστές (IBM compatible). Έτσι, για τους ακεραίους έχουμε:

short int ή short	16 bits	μικρός ακέραιος
int	16 ή 32 bits	μεσαίος ακέραιος
long int ή long	32 bits	μεγάλος ακέραιος

Ομοίως για τους πραγματικούς οι παρακάτω τύποι θα μπορούσαν να αντιπροσωπεύουν ένα, δύο ή τρία διαφορετικά μεγέθη:

float	32 bits	κινητής υποδιαστολής απλής ακρίβειας
double	64 bits	κινητής υποδιαστολής διπλής ακρίβειας
long double	80 bits	κινητής υποδιαστολής εκτεταμένης ακρίβειας

Οι προσδιοριστές signed ή unsigned μπορούν να εφαρμοστούν μόνο σε χαρακτήρες (char) και σε ακέραιους και καθορίζουν αν ο τύπος δεδομένων που προσδιορίζουν υποστηρίζει πρόσημο ή όχι. Παραδείγματος χάρη:

```
unsigned char     ή σκέτο   char   με τιμές στο διάστημα  [0, 255]
signed char                                - >> -         [-128, 127]

signed short int     ή   signed short   ή   short    [-32.768, 32.767]
unsigned short int   ή   unsigned short                [0, 65.535]

signed long int    ή  long           ή  int  [-2.147.483.648,  2.147.483.647]
unsigned long int  ή  unsigned long  ή  unsigned int      [0, 4.294.967.295]
```

Οι πρότυπες επικεφαλίδες (headers) <limits.h> και <float.h> περιέχουν συμβολικές σταθερές για όλα τα μεγέθη καθώς και για άλλα χαρακτηριστικά του μηχανήματος και του μεταγλωττιστή.

Αναπαράσταση Σταθερών στη C

Ακέραια σταθερά

Μια σταθερά τύπου ακεραίου σε ένα πρόγραμμα C εκλαμβάνεται ως ακέραιος μεσαίου μεγέθους δηλαδή τύπου int. Π.χ.
1234 → int
εκτός αν ακολουθείται από το γράμμα l ή L.
1234l ή 1234L → long (int)
επίσης ένας πολύ μεγάλος ακέραιος που δεν χωράει σε int εκλαμβάνεται σαν long.

1234u ή 1234U → unsigned (απρόσημη) σταθερά
1234ul ή 1234UL → unsigned long σταθερά

Η τιμή ενός ακέραιου μπορεί να καθορίζεται στο οκταδικό ή στο δεκαεξαδικό αντί για το δεκαδικό.
01234 → οκταδικό (αν αρχίζει με μηδέν 0)
0x1234 ή 0X1234 → δεκαεξαδικό (αν αρχίζει με 0x ή 0X)

31 = 037 = 0x1f ή 0X1F
π.χ. 0XFUL → σταθερά unsigned long =15

Πραγματική σταθερά

Μια σταθερά πραγματικού τύπου εκλαμβάνεται ως πραγματικός (μεσαίου μεγέθους) κινητής υποδιαστολής διπλής ακρίβειας δηλαδή τύπου double.
Π.χ .
123.45 ή 12345e-2 → σταθερά κινητής υποδιαστολής double

εκτός αν ακολουθείται από τους παρακάτω χαρακτήρες που προσδιορίζουν ότι πρέπει να εκληφθεί ως απλής ακρίβειας ή επεκταμένης ακρίβειας.
f ή F → float
l ή L → long double

Σταθερά χαρακτήρα

Μια σταθερά τύπου χαρακτήρα οριοθετείται από μονούς αποστρόφους, π.χ.
′A′ ′b′ ′x′ τιμή ASCII
′0′ → 48
Οι χαρακτήρες μπορούν να συμμετέχουν σε αριθμητικές πράξεις διότι η γλώσσα C τους αντιμετωπίζει ως πολύ μικρούς ακεραίους του ενός byte.
′\0ooo′ οκταδικά ψηφία που καθορίζουν διάταξη bit μεγέθους ενός byte
ή ′\xhh′ για δεκαεξαδικά

Αλφαριθμητική σταθερά ή αλφαριθμητική κυριολεξία (string)

″I am a string″ , ″Καλημέρα″ , ″PRESS ANY KEY TO CONTINUE″ , ″Κώστας″
Τα ″ ″ εξυπηρετούν την οριοθέτηση της αλφαριθμητικής σταθεράς, που δεν είναι τίποτα άλλο από μια ακολουθία (ένας ορμαθός) από χαρακτήρες.
Από τεχνική άποψη, μια αλφαριθμητική σταθερά είναι ένας πίνακας χαρακτήρων στον οποίο μετά τον τελευταίο χαρακτήρα πρέπει να υπάρχει ένα ειδικό σύμβολο, το ′\0′, που ονομάζεται μηδενικός χαρακτήρας και δηλώνει το τέλος του. **Μήκος** μιας αλφαριθμητικής ακολουθίας λέγεται ο αριθμός των χαρακτήρων που την αποτελούν χωρίς τον μηδενικό χαρακτήρα. Στην γλώσσα C, λόγω της εσωτερικής αναπαράστασης των αλφαριθμητικών ακολουθιών (strings), δεν υπάρχει περιορισμός στο μήκος τους.

Σταθερά απαρίθμησης (enumeration constant)

Σύνταξη:
enum [<type_tag>] { <constant_name> [= <value>], … }[var_list];

Παραδείγματα:
enum boolean {NO, YES} ;
enum months {JAN=1, FEB, MAR,…} ;
(αλλιώς θα ήταν JAN=0)

Παρέχουν ένα βολικό τρόπο για την σύνδεση συνεχόμενων σταθερών τιμών με ονόματα, εναλλακτικά για τις συμβολικές σταθερές (#define …)

Δηλώσεις μεταβλητών

Στην γλώσσα C για να είναι δυνατή η χρήση μιας μεταβλητής θα πρέπει να έχει δηλωθεί νωρίτερα στο πρόγραμμα σε κατάλληλο σημείο, δηλαδή να έχει καθοριστεί το όνομά της και ο τύπος της. Πρώτα γράφεται ο τύπος και μετά το όνομα ή τα ονόματα των μεταβλητών.
Π.χ.

int a;	(δήλωση μιας μεταβλητής τύπου ακεραίου με όνομα a)
long int i,k,l;	(δήλωση τριών μεταβλητών τύπου μεγάλου ακεραίου)
char c, line[100];	(δήλωση μια μεταβλητής τύπου χαρακτήρα με όνομα c και ενός πίνακα χαρακτήρων, 100 θέσεων, με όνομα line)
float periferia;	(δήλωση μια μεταβλητής τύπου κινητής υποδιαστολής απλής ακρίβειας με όνομα periferia)

Μια μεταβλητή μπορεί να πάρει αρχική τιμή (να αρχικοποιηθεί) και μέσα στη δήλωσή της.

Π.χ.

```
char cp='a';          char esc='\\';          int k=0;        int m=MAX+5;
float eps=1.0e-5;     float inch=2.54;   ή   float inch=254e-2;
long double pi=314159265359e-11;
```

Μια αυτόματη μεταβλητή παίρνει ρητά αρχική τιμή. Οι εξωτερικές και οι στατικές (static) μεταβλητές παίρνουν αυτόματα αρχική τιμή μηδέν.

Στην δήλωση οποιασδήποτε μεταβλητής μπορεί να εφαρμόζεται ο προσδιορισμός const που εξασφαλίζει ότι η τιμή της μεταβλητής δεν θα αλλάξει.

Π.χ.

```
const double e=2.718281;     const char msg[]="Προσοχή";     int strlen (const char []);
```

Τελεστές της γλώσσας προγραμματισμού C

Αριθμητικοί τελεστές

Με δύο τελεστέους (λεγόμενοι δυαδικοί) είναι: +, -, *, /, και ο τελεστής υπολοίπου % .
Οι μοναδικοί τελεστές (unary) + και – έχουν μεγαλύτερη προτεραιότητα όχι μόνο από τους αντίστοιχους δυαδικούς αλλά και από τους *, /, % .

Συσχετιστικοί και Λογικοί τελεστές

Συσχετιστικοί είναι:

>	>=	<	<=	==	!=
(μεγαλύτερο)	(μεγαλύτερο ή ίσο)	(μικρότερο)	(μικρότερο ή ίσο)	(ίσο)	(διάφορο)

και έχουν χαμηλότερη προτεραιότητα από τους αριθμητικούς.

Έτσι, π.χ., η λογική έκφραση k<lim-1 είναι ισοδύναμη με την k<(lim-1)

Οι λογικοί τελεστές && (=AND) και || (=OR) υπολογίζονται από τα αριστερά προς τα δεξιά και ο υπολογισμός σταματάει μόλις εξακριβωθεί το αληθές ή ψευδές του αποτελέσματος. Η προτεραιότητα του && είναι υψηλότερη από αυτήν του ||, ενώ και των δυο είναι χαμηλότερη από αυτήν των συσχετιστικών τελεστών και των τελεστών ισότητας.
Ο μοναδικός τελεστής άρνησης είναι ! (=NOT).
π.χ.

```
if (valid==0) …    →    if (!valid) …
if (valid==1) …    →    if (valid) …
```

Σημείωση: Στο τμήμα ελέγχου των if, while, for κτλ. «αληθής» σημαίνει μη-μηδενικός, κι έτσι δεν υπάρχει διαφορά.

Τελεστές Πράξεων με Bit

Υπάρχουν 6 τελεστές για χειρισμό bit, που εφαρμόζονται μόνο σε ακεραίους τελεστέους.

```
&     →AND
|     →OR
^     →XOR (αποκλειστικό OR)
<<    →ολίσθηση αριστερά (shift left)
>>    →ολίσθηση δεξιά (shift right)
~     →συμπλήρωμα ως προς ένα (μοναδικός τελεστής)
```

Παραδείγματα :
n = n & 0177; // εφαρμογή μάσκας

n = n | 03; // ενεργοποίηση επιπλέον των n
x = x | SET_ON // ενεργοποίηση

x=1; y=2;
x & y → 0 (μηδενική τιμή, δηλαδή FALSE)
x && y → 1 (μη-μηδενική τιμή, δηλ. TRUE)

x & ~077 // είναι ανεξάρτητη από το μήκος της λέξης.

x << 2 ⎧ ←0 ⎫ // γεμίζει με '0' τα κενά
x >> 2 ⎩ 0→ ⎭ // αριθμητική ολίσθηση γεμίζει με bit πρόσημου (ενώ η λογική βάζει '0')

Τελεστές Αντικατάστασης (assignment operator) &΄ Παραστάσεις

Η εκτέλεση μιας εντολής αντικατάστασης (καταχώρησης ή εκχώρησης), που στην γλώσσα C παριστάνεται με το σύμβολο =, προκαλεί τον υπολογισμό της παράστασης που βρίσκεται στο δεξιό μέλος και την καταχώρηση του αποτελέσματος στην μεταβλητή που βρίσκεται στο αριστερό μέλος.

Π.χ. x = a + b; x = a*5 + b/c + (d-8);

Στην εξειδικευμένη περίπτωση κατά την οποία η μεταβλητή που αποτελεί το αριστερό μέλος της εντολής αντικατάστασης συμμετέχει και στην παράσταση του δεξιού μέλους, η γλώσσα C μας προσφέρει έναν επιπλέον τρόπο, πιο βολικό, γραφής της εντολής. Στην γενική της μορφή μια εντολή αντικατάστασης όπως η παρακάτω:

παρ1 = (παρ1) <u>τελ</u> (παρ2)

μπορεί για λόγους ευκολίας να γραφεί:

παρ1 <u>τελ</u>= παρ2

όπου τελεστής «<u>τελ</u>» ένας από τους παρακάτω : + , - , * , / , % , << , >> , & , ^ , | .
Π.χ.

j = j + 2 ; ←→ j += 2 ; (τελεστής αντικατάστασης αθροίσματος)

x = x * (y + 1); ←→ x *= y + 1; (τελεστής αντικατάστασης γινομένου)

Η εντολή αντικατάστασης έχει ως τιμή, την τιμή του αριστερού της μέλους μετά την αντικατάσταση, και μπορεί να εμφανίζεται σε παραστάσεις. Σπανιότερα εμφανίζονται και οι τελεστές (/=, -= κτλ.)

Παραστάσεις υπό συνθήκη και τριαδικός τελεστής "? :"

Ο μόνος τελεστής που δέχεται τρεις τελεστέους (γι αυτόν τον λόγο αποκαλείται και τριαδικός τελεστής) συντάσσεται ως εξής:

παρ1 ? παρ2 : παρ3

όπου παρ1, παρ2 και παρ3 είναι παραστάσεις. Κατά την εκτέλεση υπολογίζεται πρώτα η παράσταση παρ1 και αν είναι αληθής (μη μηδενική) τότε υπολογίζεται η παράσταση παρ2 αλλιώς η παρ3. Ουσιαστικά η τιμή ολόκληρης της παράστασης είναι το αποτέλεσμα της παρ2 ή παρ3 ανάλογα με την παράσταση παρ1. Έτσι αιτιολογείται το γεγονός ότι τέτοιου είδους παραστάσεις λέγονται παραστάσεις υπό συνθήκη.

Π.χ. z = a>b ? a : b ; //ισοδύναμο με: `if (a>b) z=a; else z=b;` ή `z = max (a, b);`

Βέβαια, όπως είναι φυσικό, μια παράσταση υπό συνθήκη μπορεί να αποτελεί τμήμα μιας άλλης μεγαλύτερης παράστασης.

Π.χ. z = a - 3*b + (c-d ? 2*c : 13) - d;

Γράφοντας ένα απλό πρόγραμμα σε γλώσσα προγραμματισμού C

Το πρώτο πρόγραμμα στην C που προτείνουμε είναι πολύ απλό. Το μόνο που κάνει είναι να στέλνει ένα μήνυμα στην πρότυπη έξοδο, χαιρετώντας τον κόσμο. Ο κώδικάς του έχει ως εξής:

```
/* my first C program */

#include <stdio.h>

main()
{
  printf ("Hello World");
}
```

Η πρώτη γραμμή του προγράμματος είναι ένα σχόλιο το οποίο ο μεταγλωττιστής δεν το λαμβάνει υπόψη. Με τους συνδυασμούς χαρακτήρων /* και */ οριοθετούμε τα σχόλια τα οποία μπορούμε να εισάγουμε σε οποιοδήποτε σημείο του προγράμματος για δική μας διευκόλυνση.

Η επόμενη γραμμή αποτελεί μια εντολή συμπερίληψης (include) και αφορά τον προ-επεξεργαστή του μεταγλωττιστή της γλώσσας C. Στην συγκεκριμένη περίπτωση ο προ-επεξεργαστής θα συμπεριλάβει τον κώδικα του αρχείου-επικεφαλίδα (header) με το όνομα *stdio.h*, το οποίο περιέχει τις πρότυπες συναρτήσεις εισόδου /εξόδου. Αυτό το κάναμε διότι, όπως ήδη γνωρίζουμε, η γλώσσα δεν διαθέτει ενσωματωμένες εντολές εισόδου /εξόδου.

Η μοναδική εκτελέσιμη εντολή του παραπάνω παραδείγματος είναι η printf, η οποία εμφανίζει στην οθόνη ότι έχουμε περικλείσει μέσα στους διπλούς αποστρόφους (quotation marks ή quotes). Η εντολή αυτή βρίσκεται μέσα στην συνάρτηση main(), που, όπως έχουμε αναφέρει ήδη, πρέπει να υπάρχει υποχρεωτικά σε όλα τα προγράμματα C, διότι από αυτήν ξεκινάει η εκτέλεση του εκάστοτε προγράμματος.

Τα άγκιστρα στην C χρησιμοποιούνται για να οριοθετούν ομάδες εντολών, όπου είναι αναγκαίο, ενώ η χρήση τους είναι υποχρεωτική στις συναρτήσεις για να δηλώνουν την αρχή και το τέλος της συνάρτησης.

Χρήσιμοι συνδυασμοί πλήκτρων

Μετά το γράψιμο του κάθε προγράμματος πρέπει να το μεταγλωττίσουμε για να μπορέσουμε να το δώσουμε προς εκτέλεση. Στο περιβάλλον της Dev-C που δουλεύουμε, είναι πολύ χρήσιμοι οι παρακάτω συνδυασμοί πλήκτρων:

Ctrl	+	F9	→	Compile (Μεταγλώττιση)
Ctrl	+	F10	→	Run (Εκτέλεση)
F9			→	Compile & Run (Μεταγλώττιση και ύστερα Εκτέλεση)
Ctrl	+	F11	→	Rebuild All (Μεταγλώττιση και Σύνδεση όλων)

Βέβαια μπορούμε να καθοδηγηθούμε στις επιλογές μας από το μενού που μας προσφέρει το περιβάλλον της Dev-C. Η επιλογή σ' αυτήν την περίπτωση μπορεί να γίνει πατώντας το πλήκτρο Alt και το πλήκτρο του γράμματος που εμφανίζεται υπογραμμισμένο στην επιλογή που επιθυμούμε. Επίσης, μπορούμε να χρησιμοποιήσουμε και τα πλήκτρα ελέγχου κατεύθυνσης του δρομέα (τα γνωστά βελάκια) ή κατευθείαν τα αντίστοιχα εικονίδια.
Πρώτα απ' όλα πρέπει να πούμε ότι η γλώσσα C κάνει διάκριση κεφαλαίων-πεζών (είναι δηλαδή **case sensitive**). Αυτό σημαίνει ότι θεωρεί διαφορετικό χαρακτήρα τον κεφαλαίο από τον αντίστοιχο πεζό. Αυτό δεν ισχύει σε πολλές άλλες γλώσσες π.χ. Pascal, Basic.

Μορφοποίηση ή διαμόρφωση εισόδου / εξόδου

Εκτός από την printf, πολύ χρήσιμη είναι και η συνάρτηση scanf, με τη χρήση της οποίας εισάγουμε δεδομένα από το πληκτρολόγιο. Και οι δυο υπάρχουν, μαζί με άλλες, στην πρότυπη βιβλιοθήκη και χρησιμοποιούνται κυρίως όταν μας ενδιαφέρει να καθορίζουμε ρητά τον τύπο ή την μορφή των δεδομένων που εισάγουμε ή εξάγουμε. Αυτό ονομάζεται μορφοποίηση εισόδου /εξόδου. Η συμπεριφορά τους ορίζεται στο πρότυπο ANSI. Παρακάτω παραθέτουμε μερικούς χαρακτήρες μετατροπής και την σημασία τους.

%d	→	εμφάνιση σαν δεκαδικός ακέραιος
%6d	→	εμφάνιση σαν δεκαδικός ακέραιος, με πλάτος πεδίου τουλάχιστον έξι ψηφίων και στοίχιση δεξιά
%ld	→	εμφάνιση σαν μεγάλος δεκαδικός ακέραιος (long)
%f	→	εμφάνιση σαν δεκαδικός κινητής υποδιαστολής
%6.2f	→	εμφάνιση σαν δεκαδικός κινητής υποδιαστολής, με πλάτος πεδίου τουλάχιστον έξι ψηφίων και δύο ψηφία δεξιά της υποδιαστολής
%.3f	→	εμφάνιση σαν δεκαδικός κινητής υποδιαστολής, με τρία ψηφία δεξιά της υποδιαστολής και χωρίς καθορισμό ελάχιστου πλάτους πεδίου
%o	→	εμφάνιση σαν οκταδικός ακέραιος
%x	→	εμφάνιση σαν δεκαεξαδικός ακέραιος
%c	→	εμφάνιση σαν χαρακτήρας
%s	→	εμφάνιση σαν αλφαριθμητική ακολουθία (string)
%%	→	%

Σημείωση: Όταν χρησιμοποιούμε την scanf γράφουμε μόνο τα %d, %f, %o, %x, %c, %s, διότι δεν επιτρέπεται η χρήση άλλων στοιχείων για τη μορφοποίηση των δεδομένων εισόδου, όπως πχ το ελάχιστο πλάτος πεδίου, ή τον αριθμό των ψηφίων δεξιά της υποδιαστολής.

Παραδείγματα:

```
/* my second C program */
#include <stdio.h>

main()
{
  int age;
  age = 28;
  printf ("Είμαι %d χρονών\n",age);
}
```

```
/* my third C program */
#include <stdio.h>

main()
{
  float poso;
  poso = 95.68;
  printf ("Η αξία του βιβλίου είναι %.2f Ευρώ.\n",poso);
}
```

```
/* my forth C program */
#include <stdio.h>

main()
{
  int a=3,b,p;
  printf ("Δώσε ακέραιο : ");
  scanf("%d",&b);
  p=a*b;
  printf ("Το %d επί το %d κάνει %d",a,b,p);
}
```

Εντολή ελέγχου if

Συντάσσεται ως εξής:

if (παράσταση ή συνθήκη)
εντολή1;
else
εντολή2;

Κατά την εκτέλεση της εντολής, επιλύεται η παράσταση ή συνθήκη που ακολουθεί την δεσμευμένη λέξη if, και αν είναι αληθής (μη μηδενική τιμή) τότε εκτελείται η εντολή1, διαφορετικά (μηδενική τιμή) εκτελείται η εντολή2. Αν στην θέση της εντολής1 ή εντολής2 υπάρχουν περισσότερες εντολές, τότε πρέπει να τις περικλείσουμε μέσα σε άγκιστρα ({ }) και να δημιουργήσουμε έτσι μια σύνθετη εντολή.

Το τμήμα **else εντολή2;** είναι προαιρετικό. Στην περίπτωση που δεν υπάρχει, τότε η τιμή της παράστασης καθορίζει αν θα εκτελεστεί ή όχι η εντολή1.

```
/* παράδειγμα προγράμματος για την χρήση της εντολής if  */
#include <stdio.h>

main()
{
  int vathmos;
  printf ("Δώσε βαθμό : ");
  scanf("%d",&vathmos);
  if (vathmos >= 5)
      printf("προβιβάστηκες\n");
  else
      printf("απορρίφθηκες\n");
}
```

Σημείωση: Στην C το ελληνικό ερωτηματικό (semicolon → **;**) είναι σημείο τερματισμού εντολών και όχι διαχωριστικό εντολών όπως είναι σε άλλες γλώσσες σαν την Pascal. Για τον λόγο αυτό πρέπει να υπάρχει στο τέλος κάθε εντολής ακόμα και αν ακολουθείται από το else.

Εντολή επανάληψης while

Συντάσσεται ως εξής: **while (παράσταση ή συνθήκη)**
εντολή;

Κατά την εκτέλεση της εντολής, επιλύεται η παράσταση ή συνθήκη που ακολουθεί την δεσμευμένη λέξη while, και εφόσον είναι αληθής (μη μηδενική τιμή) τότε εκτελείται η εντολή που ακολουθεί και ξαναεπιλύεται η παράσταση. Αυτό επαναλαμβάνεται μέχρι η παράσταση να γίνει ψευδής (μηδενική τιμή), οπότε και τερματίζεται η εκτέλεση της εντολής while.

Είναι υποχρέωση του προγραμματιστή να μεριμνά για τις αρχικές τιμές των μεταβλητών που συμμετέχουν στην συνθήκη. Επίσης, πρέπει να περιλαμβάνει τέτοιες εντολές στο σώμα του βρόχου, που να μεταβάλλουν την τιμή της παράστασης έτσι ώστε μετά από έναν πεπερασμένο αριθμό επαναλήψεων να γίνεται μηδενική και να τερματίζεται η εκτέλεση του βρόχου. Σε αντίθετη περίπτωση θα έχουμε έναν ατέρμονα βρόχο.

Αυτός είναι ο λόγος για τον οποίο, συνήθως, το σώμα του βρόχου το αποτελούν περισσότερες από μια εντολές και έτσι είμαστε υποχρεωμένοι να τις περικλείσουμε μέσα σε άγκιστρα.

```
/* παράδειγμα για την χρήση της while:
   υπολογισμός του αθροίσματος των εκατό πρώτων φυσικών αριθμών
               1 + 2 + 3 +...+ 100   */

#include <stdio.h>

main()
{
   int sum, i;
   sum = 0 ;
   i = 1;
   while (i<=100)
   { sum = sum + i;
     i=i+1;
   }
   printf("Το αποτέλεσμα είναι %d", sum);
}
```

```
/* δημιουργία και εμφάνιση πίνακα αντιστοιχίας θερμοκρασιών Fahrenheit-
Celsius από το 0 μέχρι το 300, ανά 20, γνωρίζοντας ότι C= (5/9) (F-32)  */

#include <stdio.h>

main()
{
   int fahr,celsius;
   int lower,upper,step;
   lower=0;
   upper=300;
   step=20;
   fahr=lower;
   printf("\n  FAHRENHEIT        CELSIUS  ");
   printf("\n-----------------------------");
   while(fahr<=upper)
      {
         celsius=5*(fahr-32)/9;
         printf("\n%d\t%d",fahr,celsius);
         fahr=fahr+step;
      }
}
```

Χρειάζεται μεγάλη προσοχή στη χρήση των αριθμητικών τελεστών σε σχέση με τον τύπο των τελεστέων. Στο παραπάνω πρόγραμμα, για παράδειγμα αν γράφαμε :

```
celsius=(5/9)*(fahr-32);
```

αντί της γραμμής που έχουμε χρησιμοποιήσει, θα παίρναμε σαν αποτέλεσμα πάντα μηδέν.
Αυτό συμβαίνει διότι ο τελεστής «/» συμβολίζει τόσο την ακέραια διαίρεση όσο και την πραγματική (σε κινητή υποδιαστολή). Έτσι, η ακέραια διαίρεση 5 / 9 δίνει αποτέλεσμα 0, ενώ η πραγματική διαίρεση 5.0 / 9.0 δίνει αποτέλεσμα 0.5555 κλπ.

Σε οποιαδήποτε παράσταση μπορούν να επιβληθούν ρητές μετατροπές τύπου που λέγονται προσαρμογές (**cast**) με ένα μοναδικό (ή εναδικό - unary) τελεστή, ως εξής:

```
(όνομα_τύπου) παράσταση
```

δηλαδή η σωστότερη λύση για το παραπάνω παράδειγμα θα ήταν:

```
celsius = (float)5/9 *(fahr-32);
```

Συμβολικές Σταθερές

Η εντολή :

#define *όνομα κείμενο_αντικατάστασης*

ορίζει ένα συμβολικό όνομα ή αλλιώς μια συμβολική σταθερά. Το όνομα έχει την ίδια φόρμα με ένα όνομα μεταβλητής (ακολουθία γραμμάτων και ψηφίων που αρχίζει με γράμμα)
π.χ.

#define	INCH	2.54
#define	EPSILON	2.718281

Προσέξτε ότι δεν χρειάζεται το σύμβολο της καταχώρησης (=) ανάμεσα και ούτε το ελληνικό ερωτηματικό στο τέλος της εντολής.

Συνήθως οι προγραμματιστές τηρούν έναν άγραφο κανόνα και χρησιμοποιούν κεφαλαία γράμματα στο συμβολικό όνομα. Έτσι, κοιτάζοντας το πρόγραμμα είναι εύκολο να διακρίνει κάποιος τις συμβολικές σταθερές από τις μεταβλητές.

```
/*Υπολογισμός περιφέρειας κύκλου δίνοντας την ακτίνα*/

#include <stdio.h>

#define     PI    3.14159265359

main()
{
  int r;
  double perif;                    /*       float perif,PI=3.1415;     */
  printf("dwse aktina : ");
  scanf("%d",&r);
  perif=2*PI*r;
  printf("H periferia einai %f\n",perif);
  system("PAUSE");           // ή getchar(); Αναμονή μέχρι το πάτημα ενός πλήκτρου.
}
```

Εντολή επανάληψης for

Συντάσσεται ως εξής:

for (απόδοση αρχικής τιμής ; συνθήκη ; βήμα αύξησης ή γενικότερα μεταβολής)
εντολή;

Το τμήμα ελέγχου της εντολής for, (δηλαδή αυτό που βρίσκεται μέσα στις παρενθέσεις) αποτελείται από τρία μέρη που χωρίζονται μεταξύ τους με δυο ελληνικά ερωτηματικά (semicolons). Στο πρώτο μέρος, αρχικοποιούμε την μεταβλητή που χρησιμοποιούμε σαν μετρητή στην εντολή επανάληψης. Στο δεύτερο γράφουμε την συνθήκη που θα καθορίζει την συνέχιση ή τον τερματισμό της εντολής επανάληψης, ενώ στο τρίτο μέρος γράφουμε κατά πόσο θα αυξάνεται ή θα μειώνεται ο μετρητής μετά από κάθε εκτέλεση του βρόχου.

Τα άγκιστρα που ακολουθούν μια εντολή επανάληψης, για να περικλείσουν το σώμα του βρόχου, μπορούν να παραληφθούν όταν το σώμα του βρόχου αποτελείται μόνο από μια εντολή.

Σημείωση : Η εντολή for της C, δουλεύει και με μετρητή τύπου κινητής υποδιαστολής (float) (όχι όπως την Pascal που απαιτεί οπωσδήποτε μετρητή ακέραιου τύπου)

```
/* δημιουργία και εμφάνιση πίνακα αντιστοιχίας θερμοκρασιών Fahrenheit-
Celsius από το 0 μέχρι το 300, ανά 20, γνωρίζοντας ότι C= (5/9) (F-32)
χρησιμοποιώντας αυτήν τη φορά συμβολικές σταθερές και την εντολή for */

#include <stdio.h>

#define LOWER     0
#define UPPER     300
#define STEP      20

main()
{
   int fahr; float celsius;
   printf("\n  FAHRENHEIT        CELSIUS  ");
   printf("\n-----------------------------");
   for(farh=LOWER; fahr<=UPPER; farh+=STEP)
      {
         celsius=(float)5/9 *(fahr-32);
         printf("\n %4d\t%8.2f ",fahr,celsius);
      }
}
```

Για περισσότερη εξάσκηση να γίνουν κι άλλα παραδείγματα με την for, όπως, π.χ. ο υπολογισμός του αθροίσματος και του μέσου όρου των χιλίων πρώτων φυσικών αριθμών.

```
/* παράδειγμα για την χρήση της for:    υπολογισμός του αθροίσματος και του
   μέσου όρου των χιλίων πρώτων φυσικών αριθμών       1, 2, 3, ..., 1000   */

#include <stdio.h>
#define N   1000
main()
{ int i; long ath=0; float mo;
  for (i=1; i<=N; i=i+1)
    ath+=i;
  mo=(float)ath/N;
  printf("Το άθροισμα είναι %ld και ο μέσος όρος %.2f", ath, mo);
}
```

Τι αλλαγές πρέπει να γίνουν στο παραπάνω πρόγραμμα αν μας ενδιέφερε να υπολογίσουμε το γινόμενο αντί του αθροίσματος και ποιες αν μας ενδιέφεραν το άθροισμα και το γινόμενο.

Είσοδος και έξοδος χαρακτήρων

Η στάνταρτ είσοδος και η στάνταρτ έξοδος αντιμετωπίζονται από την C, όπως και τα αρχεία, σαν ακολουθίες χαρακτήρων, που τις ονομάζουμε ρεύματα κειμένου (text streams). Ένα ρεύμα κειμένου χωρίζεται σε γραμμές από τους χαρακτήρες «νέας γραμμής» (′\n′). Στα αποθηκευτικά μέσα (μνήμες, δίσκους) οι χαρακτήρες που αποτελούν ένα ρεύμα κειμένου βρίσκονται ο ένας μετά τον άλλο, σε διαδοχικές θέσεις.

Αν έχω δηλώσει μια μεταβλητή τύπου χαρακτήρα, έστω την c, τότε για να πραγματοποιήσω την ανάγνωση από την στάνταρτ είσοδο ή το γράψιμο στην στάνταρτ έξοδο ενός μόνο χαρακτήρα τη φορά, μπορώ να χρησιμοποιήσω τις παρακάτω μακροεντολές, οι οποίες έχουν δηλωθεί στο αρχείο stdio.h:

```
char c;

c = getchar();

putchar(c) ;
```

```
/* function - macro prototypes */
int getc (FILE *stream);
int getchar(void);

int putc(int c,FILE *stream);
int putchar(int c);
```

Η μακροεντολή getchar() επιστρέφει έναν χαρακτήρα από το input-stream stdin. Έχει δηλωθεί ως getc(stdin) όπου η συνάρτηση getc επιστρέφει έναν χαρακτήρα από ένα stream. Πρέπει να πούμε ότι η μακροεντολή είναι line-buffered που σημαίνει ότι δεν εκτελείται η επιστροφή εάν δεν πατηθεί το ENTER.

Η μακροεντολή putchar() στέλνει τον χαρακτήρα που έχει σαν όρισμα, στο output-stream stdout. Έχει δηλωθεί ως putc(c, stdout) όπου η συνάρτηση putc στέλνει τον χαρακτήρα c σε συγκεκριμένο stream.

Οι κλήσεις στις putchar και printf μπορούν να εναλλάσσονται.

Σημείωση: **macro – function** (speed versus size)

Το κυριότερο πλεονέκτημα της χρησιμοποίησης μακροεντολών είναι η ταχύτητα εκτέλεσης (faster execution time). Κατά την διάρκεια της προεπεξεργασίας κάθε macro αναπτύσσεται (αντικαθίσταται από τον ορισμό της) όπου χρησιμοποιείται.
Οι macros αυξάνουν το μέγεθος του κώδικα, αλλά δεν έχουν την υπερφόρτωση που απορρέει από τις κλήσεις των συναρτήσεων.
You can declare a pointer to a function, but not a pointer to a macro.
Macro side effects: A macro may treat arguments incorrectly when the macro evaluates its arguments more than once.

Αντιγραφή αρχείων

Μόνο με την χρήση των δύο παραπάνω απλών εντολών εισόδου / εξόδου μπορεί να γραφεί εκπληκτικής έκτασης χρήσιμος κώδικας χωρίς την οποιαδήποτε άλλη γνώση για είσοδο και έξοδο.

Έστω ότι θέλουμε να γράψουμε ένα πρόγραμμα το οποίο να πραγματοποιεί την αντιγραφή ενός αρχείου σε ένα άλλο.

Γνωρίζουμε ότι μόλις ξεκινήσει η εκτέλεση ενός προγράμματος στη C αυτόματα ανοίγονται και συνδέονται μ' αυτό τρία αρχεία (text streams), το **stdin**, το **stdout** και το **stderr** που έχουν δηλωθεί σαν pointers τύπου αρχείου ως εξής: FILE *stdin, *stdout, *stderr;

Μπορούμε να απλοποιήσουμε το πρόγραμμα, θεωρώντας σαν αρχείο εισόδου την στάνταρτ είσοδο (stdin), και σαν αρχείο εξόδου την στάνταρτ έξοδο (stdout), χωρίς να χάσουμε από την γενικότητα του προβλήματος.

Έτσι καταφέρνουμε να επικεντρώσουμε την προσοχή μας πάνω στο αντικείμενο που μας ενδιαφέρει και να μην ασχοληθούμε με την διαδικασία σύνδεσης του προγράμματός μας με «φυσικά» αρχεία που βρίσκονται αποθηκευμένα σε μαγνητικά μέσα αποθήκευσης.

Το περίγραμμα ενός τέτοιου προγράμματος θα μπορούσε να έχει ως εξής:

διάβασμα ενός χαρακτήρα
ενόσω (ο χαρακτήρας δεν είναι ένδειξη τέλους αρχείου)
έξοδος του χαρακτήρα που μόλις διαβάστηκε
διάβασμα ενός χαρακτήρα

Με βάση το παραπάνω περίγραμμα μπορούμε να γράψουμε το πρόγραμμα σαν μια πρώτη προσέγγιση:

```
#include <stdio.h>

/* αντιγραφή της εισόδου στην έξοδο (α' προσέγγιση) */

main()
{
  char c;
  c = getchar();
  while ( c != EOF)
  {
     putchar(c) ;
     c = getchar();
  }
}
```

Στη C, οποιαδήποτε αντικατάσταση όπως η `c = getchar();` είναι παράσταση και έχει μια τιμή, την τιμή του αριστερού της μέλους μετά την αντικατάσταση. Αυτό σημαίνει ότι μια αντικατάσταση μπορεί να εμφανίζεται σαν τμήμα μιας μεγαλύτερης παράστασης. Αν η απόδοση ενός χαρακτήρα στην C τοποθετηθεί μέσα στο τμήμα ελέγχου ενός βρόχου while, το πρόγραμμα αντιγραφής μπορεί να γραφεί ως εξής:

```
#include <stdio.h>

/* αντιγραφή της εισόδου στην έξοδο (β' προσέγγιση) */

main()
{
  char c;
  printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
  while ((c = getchar() ) != EOF)
    putchar(c) ;
}
```

Πάντως, είναι δυνατόν να παρασυρθεί κανείς και να δημιουργήσει ακατανόητο κώδικα, μια τάση που θα προσπαθήσουμε να αποτρέψουμε.

Οι παρενθέσεις γύρω από την απόδοση τιμής μέσα στην συνθήκη είναι απαραίτητες, διότι η προτεραιότητα της πράξης ελέγχου (!=) είναι υψηλότερη από αυτήν της αντικατάστασης (=). Αυτό σημαίνει ότι αν έλειπαν οι παρενθέσεις ο έλεγχος συσχέτισης (!=) θα γίνονταν πριν από την απόδοση τιμής (=). Έτσι, η εντολή

```
c =  getchar() != EOF
```

που είναι ισοδύναμη με την

```
c = (getchar() != EOF)
```

δεν θα έχει το επιθυμητό αποτέλεσμα (δηλαδή το c θα πάρει τιμή σύμφωνα με το αποτέλεσμα της σύγκρισης).

Μέτρηση Χαρακτήρων

Έστω τώρα, ότι θέλουμε να καταμετρήσουμε τους χαρακτήρες που περιέχει ένα αρχείο. Θεωρούμε, και πάλι, για ευκολία σαν αρχείο την στάνταρτ είσοδο.
Το πρόγραμμα αυτό, με μετρητή χαρακτήρων μια μεταβλητή τύπου (long int) «μεγάλου ακεραίου» και χρησιμοποιώντας την εντολή επανάληψης while, μπορεί να γραφεί ως εξής:

```
#include <stdio.h>                  /* μέτρηση χαρακτήρων */

main()
{
  long nc;
  nc=0;
  printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
  while (getchar() != EOF)
        ++nc;            /* ή αλλιώς  nc = nc + 1;  ή  nc += 1;  ή και  nc++  */
  printf( "%ld\n",nc) ;
}
```

++ → τελεστής αύξησης κατά ένα
-- → τελεστής μείωσης κατά ένα

Οι τελεστές αυτοί μπορεί να είναι είτε προθεματικοί (++nc) είτε μεταθεματικοί (nc++).

Π.χ.
```
n = 5;
x = n++;          /* μεταθεματικός τελεστής, άρα μετά την εκτέλεση
                     της πράξης θα έχουμε x = 5 και n = 6  */
```

Π.χ.
```
n = 5;
x = ++n;          /* προθεματικός τελεστής, άρα μετά την εκτέλεση της
                     πράξης θα έχουμε  x = 6 και  n = 6  */
```

Το ίδιο πρόγραμμα, με μετρητή χαρακτήρων μια μεταβλητή τύπου (double) «κινητής υποδιαστολής διπλής ακρίβειας» και χρησιμοποιώντας την εντολή επανάληψης for, μπορεί να γραφεί ως εξής:

```
#include <stdio.h>                  /* μέτρηση χαρακτήρων */

main()
{ double nc;
  printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
  for (nc = 0; getchar() != EOF; ++nc)
          ;       /* κενό σώμα βρόχου  ; μηδενική εντολή (null statement) */
  printf( "%.0f\n",nc) ;
}
```

Παρατηρήστε ότι, για την διαμόρφωση της εξόδου, στην εντολή printf, χρησιμοποιούμε το `%.0f` το οποίο εμποδίζει την εμφάνιση δεκαδικών ψηφίων και υποδιαστολής και ταιριάζει τόσο στο τύπο float όσο και στον τύπο double.

Επίσης, επαληθεύεται το γεγονός ότι στην γλώσσα C για την εντολή for μπορούμε να χρησιμοποιούμε σαν μετρητή μια μεταβλητή πραγματικού τύπου.

Μέτρηση γραμμών

Για να μετρήσουμε τον αριθμό των γραμμών ενός αρχείου κειμένου δεν έχουμε παρά να μετρήσουμε τον αριθμό των χαρακτήρων «νέας γραμμής» (`'\n'`) που υπάρχουν σ' αυτό.

```
#include <stdio.h>

/* για τη μέτρηση γραμμών  αρκεί η μέτρηση των χαρακτήρων νέας γραμμής */

main()
{
   char car; int nl;
   nl=0;
   printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
   while(( car=getchar())!=EOF)
      if (car=='\n')
            ++nl;
   printf("Υπάρχουν %d new-lines",nl);
}
```

Ένας χαρακτήρας γραμμένος μέσα σε μονά εισαγωγικά αντιπροσωπεύει μια ακέραιη τιμή που είναι ίση με την αριθμητική τιμή του χαρακτήρα στο σύνολο χαρακτήρων του μηχανήματος και λέγεται σταθερά χαρακτήρα (character constant).

Προσοχή, το '\n' είναι ένας μόνο χαρακτήρας (στις παραστάσεις είναι ένας απλός ακέραιος) ενώ το "\n" είναι μια αλφαριθμητική σταθερά που τυχαίνει να έχει έναν μόνο χαρακτήρα.

Μέτρηση λέξεων

Για να μετρήσω τις λέξεις ενός αρχείου κειμένου αυξάνω τον μετρητή των λέξεων, κάθε φορά που συναντάω τον πρώτο χαρακτήρα μιας λέξης. Σαν διαχωριστικά λέξεων θεωρούμε μόνο το κενό διάστημα (`' '`), τον στηλογνώμονα (`'\t'`) και τον χαρακτήρα νέας γραμμής (`'\n'`) που ονομάζονται λευκά διαστήματα.

```
#include <stdio.h>

/* μέτρηση λέξεων, γραμμών, χαρακτήρων */

#define       IN   1
#define       OUT  0

main()
{ char car;              // στην Turbo C έπρεπε να είναι ακέραιος [int car;]
  int nc, nw, nl, status;
  nc=nw=nl=0;
  status=OUT;
  printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
  while(( car=getchar())!=EOF)
  { ++nc;
    if (car=='\n')
      ++nl;
    if (car==' '||car=='\t'||car=='\n')
      status=OUT;
    else
      if (status==OUT)
        { ++nw;
          status=IN;
        }
  }
  printf("Υπάρχουν %d λέξεις, %d γραμμές και %d χαρακτήρες.\n",nw,nl,nc);
}
```

Φωλιασμένα if (nested if)

Στο προηγούμενο παράδειγμα για να επιτύχουμε την μέτρηση λέξεων παρατηρούμε την χρήση μιας ακόμη νέας δομής η οποία αποτελείται από δυο εντολές ελέγχου if, η μια εξαρτώμενη από την άλλη. Αυτό γίνεται και με περισσότερα if τα οποία ονομάζουμε φωλιασμένα if (ή περικλειόμενα if), ως εξής:

```
if  (συνθήκη-1)
        εντολή-1;
else if  (συνθήκη-2)
        εντολή-2;
else if  (συνθήκη-3)
        εντολή-3;
        …
else if  (συνθήκη-N)
        εντολή-N;
else
        εναλλακτική-εντολή;
```

Κατά την εκτέλεση ελέγχονται οι συνθήκες μία-μία και μόλις βρεθεί μια αληθής, εκτελείται η εντολή που την ακολουθεί και κατόπιν παρακάμπτεται όλη η υπόλοιπη δομή των φωλιασμένων if. Αν δεν βρεθεί καμιά συνθήκη αληθής, τότε εκτελείται η εναλλακτική εντολή, η οποία βρίσκεται μετά το τελευταίο else. Η ύπαρξη του τελευταίου else και της εναλλακτικής εντολής είναι προαιρετική.

Σημείωση: Ο όρος εντολή καλύπτει και τις τρεις περιπτώσεις
- κενή εντολή (empty statement)
- απλή εντολή
- ομάδα εντολών (δηλαδή σύνθετη εντολή)

Γενικοί κανόνες για την χρήση της εντολής if:

Το else συνδέεται με το πιο κοντινό if στο οποίο δεν έχει ήδη αντιστοιχηθεί μια εντολή else.

Τόσο το if όσο και το else που του αντιστοιχεί πρέπει να βρίσκονται μέσα στην ίδια ομάδα κώδικα.

Μια δομή με *ν* εντολές απλών ή ανεξάρτητων if, των οποίων οι συνθήκες είναι τέτοιες ώστε σε οποιαδήποτε χρονική στιγμή το πολύ η μια από αυτές να είναι αληθής, μπορεί να αντικατασταθεί από μια δομή με *ν*-1 φωλιασμένα if.

Προσοχή: Μια ομάδα εντολών ανεξάρτητων if των οποίων οι συνθήκες δεν αλληλοαποκλείονται (δηλαδή, η ισχύς τής μιας συνθήκης να αποκλείει την ισχύ όλων των υπολοίπων) δεν μπορεί να μετατραπεί σε δομή φωλιασμένων if.

Εντολή επανάληψης do – while

Συντάσσεται ως εξής:

```
do {
        εντολές ;
} while (παράσταση ή συνθήκη);
```

Κατά την εκτέλεση της εντολής, εκτελούνται οι εντολές που βρίσκονται ανάμεσα στις λέξεις κλειδιά do και while και κατόπιν επιλύεται η παράσταση ή συνθήκη που ακολουθεί την δεσμευμένη λέξη while. Εφόσον είναι αληθής (μη μηδενική τιμή) τότε εκτελούνται και πάλι οι εντολές που αποτελούν το σώμα του βρόχου και ξαναεπιλύεται η παράσταση. Αυτό επαναλαμβάνεται μέχρι η παράσταση να γίνει ψευδής (μηδενική τιμή), οπότε και τερματίζεται η εκτέλεση της εντολής do - while.

Όπως και στη while έτσι και στην do - while είναι υποχρέωση του προγραμματιστή να μεριμνά για τις μεταβλητές που συμμετέχουν στην συνθήκη.

Παράδειγμα:

```
#include <stdio.h>

/* υπολογισμός αθροίσματος και μέσου όρου ακεραίων αριθμών μέχρι να δοθεί η
τιμή μηδεν (0) ως ένδειξη τέλους εισαγωγής δεδομένων */

main()
{ int a=0, pl=-1;    float avg;   long sum=0;
  do{
    sum +=a;
    pl++;
    printf ("Δώστε το %do ακέραιο : ",pl+1);
    scanf("%d",&a);
  }while (a!=0);                          /*    while(a);   */
  if (pl)
    avg = (float)sum/pl;
  printf("\nΔόθηκαν %d ακέραιοι με άθροισμα %ld και μ.όρο %8.2f",pl,sum,avg);
}
```

Παρατήρηση: Η εντολή **do-while** εκτελεί τις εντολές του βρόχου όση ώρα η συνθήκη είναι αληθής, σε αντίθεση με την repeat-until της Pascal, η οποία εκτελεί τις εντολές του βρόχου μέχρι η συνθήκη να γίνει αληθής, δηλαδή όση ώρα η συνθήκη είναι ψευδής.

Ασκήσεις :

Υπολογισμός αθροίσματος και μέσου όρου αρτίων ή περιττών αριθμών οι οποίοι βρίσκονται σε κλειστό διάστημα, όπως επίσης και ακεραίων μέχρι να δοθεί η τιμή μηδέν (0) χωριστά για θετικούς και αρνητικούς.

Κατηγοριοποίηση μαθητών με βαθμό ακέραιο ή πραγματικό (με φωλιασμένα if)

Πίνακες

Είναι μια δομή που αποτελείται από ένα προκαθορισμένο αριθμό στοιχείων όλα ίδιου τύπου. Οι δείκτες των πινάκων στην C αρχίζουν πάντα από το μηδέν. Δείκτης μπορεί να είναι οποιαδήποτε παράσταση (ακέραιου τύπου) που περιέχει ακέραιες μεταβλητές και σταθερές.
Αυτή η τελευταία πρόταση είναι μια περίπτωση ενός γενικού κανόνα: Οπουδήποτε επιτρέπεται να χρησιμοποιήσετε μια μεταβλητή ενός τύπου, μπορείτε να χρησιμοποιήσετε και μια πιο πολύπλοκη παράσταση του ίδιου τύπου.

Χρειάζεται μεγάλη προσοχή στην χρήση πινάκων διότι δεν γίνεται έλεγχος ορίων του πίνακα.

-δήλωση πίνακα:

```
int a[20];      // δήλωση πίνακα είκοσι στοιχείων [0..19] που το καθένα είναι ακέραιος.
char s[50];     // δήλωση πίνακα χαρακτήρων (string) με 50 στοιχεία.
```

-αρχικοποίηση πίνακα:

```
for (i=0; i<20; ++i)
  a[i]=0;
```

Παράδειγμα:

```
#include <stdio.h>

/* μέτρηση ψηφίων, λευκών διαστημάτων και των λοιπών χαρακτήρων της εισόδου*/

main()
{ int i, car, nwhite, nother;
   int ndigit[10];
   nwhite=nother=0;
   for (i=0;i<10;++i)
      ndigit[i]=0;
   printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
   while(( car=getchar())!=EOF)
      if (car>='0'&&car<='9')
        ++ndigit[car-'0'];
      else  if (car==' '||car=='\t'||car=='\n')
        ++nwhite;
      else
        ++nother;
   printf("\n Ψηφία =");
   for (i=0; i<10; ++i)
      printf("  %d",ndigit[i]);
   printf(", λευκά διαστήματα = %d, άλλοι χαρακτήρες = %d \n",nwhite,nother);
}
```

Οι συναρτήσεις-εντολές srand() και rand().

Αρκετές φορές σε προγράμματα μας είναι απαραίτητη η χρήση τυχαίων αριθμών. Η γλώσσα C μας προσφέρει αυτήν την δυνατότητα με την rand() η οποία είναι ορισμένη στο αρχείο-επικεφαλίδα stdlib.h με το πρωτότυπο:

```
int rand(void);
```

και η οποία επιστρέφει έναν ακέραιο μεταξύ του μηδενός και του 32767 (του RAND_MAX) μείον ένα, δηλαδή μια τιμή από το κλειστό διάστημα [0 , 32767].

Για να μη παίρνουμε τις ίδιες ψευδοτυχαίες τιμές σε κάθε εκτέλεση του προγράμματός μας από την rand(), καλούμε μια άλλη συνάρτηση-εντολή που είναι επίσης ορισμένη στο αρχείο-επικεφαλίδα stdlib.h με το πρωτότυπο:

```
void srand (unsigned int seed);
```

και η οποία χρησιμοποιεί το seed σαν φύτρο για νέα ακολουθία ψευδοτυχαίων αριθμών. Το αρχικό φύτρο είναι το 1.

Σημείωση:

Σε κάποιες εκδόσεις υποστηρίζονται δύο μακροεντολές που ορίζονται στο αρχείο-επικεφαλίδα stdlib.h με την βοήθεια των προηγούμενων συναρτήσεων. Αυτές είναι η:

`int random (int num);` // ορισμένη ως: **`rand() % num;`**

και η οποία επιστρέφει έναν ακέραιο μεταξύ του μηδενός και του ορίσματος που της δώσαμε μείον ένα, δηλαδή μια τιμή από το κλειστό διάστημα [0 , num-1].

Ομοίως, για να μη παίρνουμε τις ίδιες ψευδοτυχαίες τιμές σε κάθε εκτέλεση του προγράμματός μας από την random(), καλούμε μια άλλη μακροεντολή με πρωτότυπο:

`void randomize (void);` // ορισμένη ως: **`srand((unsigned) time(NULL));`**

και η οποία χρησιμοποιεί μια χρονική συνάρτηση για να αρχικοποιήσει την γεννήτρια ψευδοτυχαίων αριθμών.

Εμείς με την Dev-C θα χρησιμοποιούμε την srand() και την rand() καθώς και τις επικεφαλίδες stdlib.h και time.h η οποία περιέχει την συνάρτηση time().

Παράδειγμα:

```
#include <stdio.h>
#include <stdlib.h>

/* γέμισμα μονοδιάστατου πίνακα 50 ακεραίων με (ψευδο)τυχαίες τιμές
                                                  από το 0 μέχρι το 200 */
#define N   50
#define K   200
main()
{ int i,p[N];
  srand((unsigned)time(NULL));                          //randomize();
  for (i=0;i<N;++i)
     p[i]=rand()%(K+1);                                 // p[i]=random(K+1);
}
```

Εντολή ελέγχου switch

Η εντολή switch παρέχει έναν άλλο τρόπο γραφής μιας πολλαπλής διακλάδωσης, ο οποίος είναι ιδιαίτερα κατάλληλος όταν η συνθήκη ελέγχει αν κάποια ακέραιη παράσταση ή παράσταση χαρακτήρων ταυτίζεται με μία από τις σταθερές ενός συνόλου.
Η switch διαφέρει από την if στο ότι η πρώτη ελέγχει μόνο την ισότητα, ενώ η παράσταση με συνθήκη της if μπορεί να είναι οποιουδήποτε τύπου.
Παρόλα αυτά όμως, όπου μπορεί να χρησιμοποιηθεί η εντολή switch την προτιμάμε διότι είναι πιο αποδοτική από τα φωλιασμένα if.

Συντάσσεται ως εξής:

```
switch (παράσταση)
{
        case  σταθερά1 : εντολέςA  ; break ;
        case  σταθερά2 : εντολέςB  ; break ;
        case  σταθερά3 :
        case  σταθερά4 : case  σταθερά5 : εντολέςC  ; break ;
        …
        case  σταθεράN : εντολέςD ; break ;
        default : εντολέςE ; break ;
}
```

Κάθε περίπτωση (case) επιγράφεται με μια ή περισσότερες σταθερές ακέραιης τιμής ή σταθερές παραστάσεις. Αν μια περίπτωση ταυτίζεται με την τιμή της παράστασης, η εκτέλεση αρχίζει από αυτήν. Όλες οι παραστάσεις case πρέπει να είναι διαφορετικές, δηλαδή δεν μπορούμε έχουμε δυο σταθερές case με ίδιες τιμές στην ίδια switch. Η περίπτωση που επιγράφεται default εκτελείται όταν δεν ικανοποιείται καμιά από τις άλλες περιπτώσεις. Η default είναι προαιρετική. Οι default και case μπορούν να εμφανίζονται με οποιαδήποτε σειρά. Η εντολή break προκαλεί την άμεση έξοδο από την switch όπως και από τους βρόχους while, for και do.

Έτσι λοιπόν το παραπάνω πρόγραμμα θα μπορούσε να γραφεί ως εξής:

```
#include <stdio.h>

/* μέτρηση ψηφίων, λευκών διαστημάτων και των λοιπών χαρακτήρων της εισόδου
   με switch */

main()
{
   int i, car, nwhite, nother;
   int ndigit[10];
   nwhite=nother=0;
   for (i=0;i<10;++i)
      ndigit[i]=0;
   printf("\nΠληκτρολόγησε ό,τι θέλεις και ENTER(ή CTRL+Z για τέλος):\n");
   while(( car=getchar())!=EOF)
      switch (car)
            {     case '0': case '1': case '2': case '3': case '4':
                  case '5': case '6': case '7': case '8': case '9':
                        ++ndigit[car-'0'];
                        break;
                  case ' ':
                  case '\n':
                  case '\t':
                        nwhite++;
                        break;
                  default:
                        nother++;
                        break;
            }
   printf("\n Ψηφία =");
   for (i=0;i<10;++i)
      printf("  %d",ndigit[i]);
   printf(", λευκά διαστήματα = %d, άλλοι χαρακήρες = %d\n",nwhite,nother);
}
```

Σημείωση: Αν είχαμε να επιλύσουμε το πρόβλημα κατηγοριοποίησης των μαθητών χρησιμοποιώντας βαθμούς εκφραζόμενους δια μέσου πραγματικών αριθμών δεν θα μπορούσαμε να χρησιμοποιήσουμε την switch.

Δημιουργία *Μενού Επιλογής*

Με τις εντολές do – while και switch μπορούμε να φτιάξουμε πολύ εύκολα μενού επιλογής με τον τρόπο που ακολουθεί:

```
 .
 .
 .
do{    clrscr();        /* καθαρισμός οθόνης δεν ανήκει στο πρότυπο ANSI C
                                              αλλά ορίζεται στην conio.h */
  printf("\n   ΜΕΝΟΥ ΕΠΙΛΟΓΗΣ");
  printf("\n 1. ααα  . . .          ");
  printf("\n 2. βββ  . . .          ");
  printf("\n 3. γγγ  . . .          ");
    .
    .
    .
  printf("\n 0. ΕΞΟΔΟΣ               ");
  printf("\n\n ΔΩΣΤΕ ΕΠΙΛΟΓΗ : ");
  scanf ("%d",&choice);
}while(choice<0||choice>3);
```

```
switch (choice)
{ case 1:   . . .      break;
  case 2:   . . .      break;
  case 3:   . . .      break;

  case 0:  exit(1);    break;
}
 .
 .
 .
```

Αρκεί να συμπληρωθούν οι διαθέσιμες επιλογές μέσα στην do-while (εδώ είναι τρεις και μία η έξοδος τέσσερις), και να εισαχθεί ο κατάλληλος κώδικας μετά από κάθε case της switch.

Άσκηση: Έχοντας σαν κορμό τα παραπάνω φτιάξτε για εξάσκηση ένα πρόγραμμα που θα υπολογίζει κατ' επιλογή το εμβαδόν ενός τριγώνου, ενός τετραγώνου, ενός κύκλου ή ενός ορθογωνίου παραλληλογράμμου.

Οι εντολές break, continue και goto

Οι εντολές break και continue χρησιμοποιούνται σπάνια. Μερικές φορές εξυπηρετεί να μπορούμε να βγούμε από ένα βρόχο χωρίς να εκτελέσουμε τον έλεγχο της αρχής ή του τέλους. Η εντολή **break** προσφέρει πρόωρη έξοδο από τους for, while και do-while ακριβώς όπως και στην switch. Προκαλεί δηλαδή άμεση έξοδο από τον πιο εσωτερικό βρόχο ή switch. Η εντολή **continue** σχετίζεται με την break, αλλά χρησιμοποιείται σπανιότερα. Προκαλεί την έναρξη του επόμενου βρόχου for, while, ή do-while που την περιέχει. Στους while και do-while αυτό σημαίνει ότι εκτελείται αμέσως το τμήμα ελέγχου, ενώ στην for ο έλεγχος περνάει στο βήμα αύξησης. Η εντολή continue εφαρμόζεται μόνο σε βρόχους, και όχι στην switch. Η continue χρησιμοποιείται συχνά όταν το τμήμα του βρόχου που ακολουθεί είναι περίπλοκο, με αποτέλεσμα η αντιστροφή ενός ελέγχου και η δημιουργία ακόμα μιας εσοχής να βάζει το πρόγραμμα σε επαλληλίες με πολύ μεγάλο βάθος.

Μια άλλη εντολή που χρησιμοποιείται σπάνια είναι η **goto**. Τυπικά, η goto δεν χρειάζεται ποτέ, και όντως είναι σχεδόν πάντα εύκολο να γραφεί κώδικας χωρίς αυτήν. Μια ίσως δικαιολογημένη περίπτωση χρήσης της είναι για να προκαλέσει την έξοδο από μια βαθιά φωλιασμένη δομή, όπως από δυο ή περισσότερους βρόχους. Η εντολή break δεν μπορεί να χρησιμοποιηθεί άμεσα, διότι προκαλεί την έξοδο μόνο από τον πιο εσωτερικό βρόχο. Έτσι:

```
for ( . . . )
      for ( . . . )
            for ( . . . )
            {
                  . . .
                  if  (disaster)
                        goto error;

            }


. . .
error:                                    /* ονομασία ετικέτας */
      . . .
```

Συναρτήσεις

Όπως στην Pascal είχαμε διαδικασίες (procedures) και συναρτήσεις (functions) για να γράφουμε τα υποπρογράμματά μας έτσι και στην C έχουμε τις συναρτήσεις (functions). Μια συνάρτηση είναι ένας βολικός και ενδεδειγμένος τρόπος για την απομόνωση μέρους ενός υπολογισμού, που μπορεί να χρησιμοποιείται μετά, χωρίς να μας ενδιαφέρει ο τρόπος υλοποίησής του. Με κατάλληλα σχεδιασμένες συναρτήσεις, μπορούμε να αγνοούμε πως γίνεται μια εργασία, αφού είναι αρκετό να γνωρίζουμε τι γίνεται. Η C κάνει τη χρήση συναρτήσεων εύκολη, άνετη και αποτελεσματική.

Επειδή η C δεν έχει τελεστή ύψωσης σε δύναμη (βέβαια, η πρότυπη βιβλιοθήκη math.h περιέχει μια συνάρτηση την pow(x,y) για τον υπολογισμό x^y), ας παρουσιάσουμε τον μηχανισμό του ορισμού συναρτήσεων γράφοντας μια συνάρτηση, έστω την *power(m,n)* για την ύψωση ενός ακέραιου *m* στην ακέραιη και θετική δύναμη *n*.

```
  #include <stdio.h>

/* πρόγραμμα που εμφανίζει έναν πίνακα με τα αποτελέσματα της ύψωσης
            των βάσεων 2 και −3 στις δυνάμεις από το 0 μέχρι το 9 */

/* συνάρτηση υπολογισμού ύψωσης δύναμης */

int power (int basi, int ekthetis)
{
   int i,p;
   p=1;
   for (i=1;i<=ekthetis;++i)
      p=p*basi;
   return p;
}

/* κυρίως πρόγραμμα */

main()
{
   int i;
   printf("Εκθέτης ι  2 εις την ι  -3 εις την ι \n");
   printf("------------------------------------ \n");
   for (i=0;i<10;++i)
      printf("%5d %12d %12d \n",i,power(2,i),power(-3,i));
   return 0;
}
```

Ο ορισμός μιας συνάρτησης ξεκινάει με τον τύπο της συνάρτησης δηλαδή με τον τύπο των δεδομένων που επιστρέφει η συνάρτηση. Κατόπιν, ακολουθεί το όνομα της συνάρτησης και μετά από αυτό ακολουθεί η λίστα των παραμέτρων (ορισμάτων) της συνάρτησης, εάν υπάρχουν μέσα σε παρενθέσεις ( ). Αυτές οι παράμετροι ονομάζονται τυπικές (formal) παράμετροι (ή τυπικά ορίσματα), ενώ κατά την κλήση της συνάρτησης ονομάζονται πραγματικές (actual) παράμετροι (ή πραγματικά ορίσματα).

Μετά ακολουθεί μέσα σε άγκιστρα ({ , }) το σώμα της συνάρτησης, δηλαδή οι δηλώσεις των τοπικών μεταβλητών και οι εντολές που πρόκειται να εκτελεστούν.

Η τιμή που υπολογίζει η power επιστρέφεται στην main με την εντολή return. Η return μπορεί να ακολουθείται από οποιαδήποτε παράσταση.

Μια συνάρτηση δεν είναι υποχρεωτικό να επιστρέφει τιμή. Η εντολή return χωρίς παράσταση αναγκάζει την επιστροφή του ελέγχου, όχι όμως και κάποιας χρήσιμης τιμής, στην

καλούσα συνάρτηση. Το ίδιο γίνεται και με την υπέρβαση του τέλους μιας συνάρτησης, που συμβαίνει όταν ο έλεγχος φτάσει στο τελικό δεξί άγκιστρο. Αντίστοιχα, η καλούσα συνάρτηση μπορεί να αγνοήσει την τιμή που επιστρέφεται από μια συνάρτηση.

Στο τέλος της συνάρτησης main υπάρχει μια εντολή return, η οποία επιστέφει μια τιμή στο περιβάλλον που την κάλεσε. Τα προγράμματα, κατά κανόνα, για να υποδηλώσουν κανονικό τερματισμό, επιστρέφουν μια μηδενική τιμή προς το περιβάλλον στο οποίο εκτελέστηκαν. Από εδώ και μετά θα περιλαμβάνουμε στις συναρτήσεις main τις εντολές return, τις οποίες μέχρι τώρα παραλείπαμε για λόγους απλότητας.

Θα μπορούσαμε να γράψουμε πρώτα τη συνάρτηση main και κατόπιν την power. Σ' αυτή την περίπτωση πριν από την main θα έπρεπε να είχαμε γράψει επιπλέον την παρακάτω δήλωση:

```
int power (int m, int n);
```

Η δήλωση αυτή, που λέει ότι η power είναι συνάρτηση η οποία περιμένει δυο ορίσματα ακέραιου τύπου και επιστρέφει έναν ακέραιο, λέγεται ***πρωτότυπο συνάρτησης*** (function prototype) και πρέπει να συμφωνεί τόσο με τον ορισμό όσο και με την χρήση της power.

Τα ονόματα των παραμέτρων δεν χρειάζεται να συμφωνούν. Μάλιστα, τα ονόματα των παραμέτρων είναι προαιρετικά στο πρωτότυπο μιας συνάρτησης. Θα μπορούσαμε, δηλαδή, να είχαμε γράψει:

```
int power (int, int);
```

Σημείωση : Στην C++ πρέπει πάντοτε να γίνεται χρήση των πρωτότυπων των συναρτήσεων. Προτείνεται επίσης η χρήση τους και στην απλή C.

Η ουσιαστική διαφορά ανάμεσα σ' έναν ορισμό (definition) και μια δήλωση (declaration) είναι ότι ο ορισμός περιέχει και το σώμα της συνάρτησης.

Ορίσματα - Κλήση κατ' αξία

Όλα τα ορίσματα των συναρτήσεων στη C μεταβιβάζονται «κατ' αξία» (pass by value), γεγονός που οδηγεί σε κάποιες ιδιότητες διαφορετικές από αυτές που βλέπουμε σε γλώσσες με κλήση «κατ' αναφορά» (pass by reference), όπου η καλούμενη ρουτίνα έχει πρόσβαση στο ίδιο το όρισμα και όχι σε ένα τοπικό αντίγραφό του.

Η βασική διάκριση είναι ότι στην C η καλούμενη συνάρτηση δεν μπορεί να αλλάξει άμεσα μια μεταβλητή της καλούσας συνάρτησης. Όταν όμως είναι απαραίτητο, μπορούμε να κανονίσουμε ώστε μια συνάρτηση να τροποποιεί μια μεταβλητή της καλούσας ρουτίνας. Η καλούσα ρουτίνα πρέπει να δίνει τη διεύθυνση της μεταβλητής που θα πάρει τιμή - πρακτικά να δίνει δείκτη διεύθυνσης ή εν συντομία δείκτη (pointer) στην μεταβλητή - και η καλούμενη ρουτίνα πρέπει να δηλώνει την παράμετρο σαν δείκτη, και έτσι να προσπελάζει έμμεσα τη μεταβλητή χρησιμοποιώντας αυτό τον δείκτη.

Δείκτης είναι μια μεταβλητή που περιέχει τη διεύθυνση μιας θέσης μνήμης (πχ μιας μεταβλητής). Οι δείκτες χρησιμοποιούνται πολύ στη C, τόσο επειδή μερικές φορές είναι ο μόνος τρόπος για να διατυπωθεί ένας υπολογισμός, όσο και επειδή συνήθως οδηγούν σε κώδικα πιο περιεκτικό και πιο αποτελεσματικό απ' αυτόν που θα προέκυπτε με άλλους τρόπους. Ο τύπος void * (δείκτης σε void) αντικαθιστά τον char * σαν ο κατάλληλος τύπος γενικού δείκτη.

Όπως είδαμε πιο πάνω, λοιπόν, δείκτης είναι μια ομάδα κελιών (συνήθως δύο ή τέσσερα) που μπορούν να κρατήσουν μια διεύθυνση. Έτσι αν η μεταβλητή c είναι τύπου char και η p είναι δείκτης (pointer) που θέλουμε να δείχνει σ' αυτήν τότε με τον τελεστή & μπορούμε να αποδώσουμε (καταχωρήσουμε) τη διεύθυνση της μεταβλητής c στην p ως εξής:

```
char c,  *p;      /* δήλωση χαρακτήρα και δείκτη σε χαρακτήρα */
p  =  &c ;        /* καταχώρηση της διεύθυνσης του c στον δείκτη p */
*p = 'a';         /* καταχώρηση του χαρακτήρα 'a' στη μεταβλητή c μέσω του δείκτη p */
```

Ο τελεστής διεύθυνσης & εφαρμόζεται μόνο σε αντικείμενα που είναι στη μνήμη, δηλαδή σε μεταβλητές και στοιχεία πινάκων και όχι σε παραστάσεις, σταθερές ή μεταβλητές register.

Ο τελεστής * είναι ο τελεστής έμμεσης αναφοράς (indirection ή dereference). Όταν εφαρμόζεται σε δείκτη, προσπελάζει το αντικείμενο που δείχνει ο δείκτης. Έστω ότι οι x και y είναι ακέραιοι, και ότι ο ip είναι δείκτης τύπου ακεραίου. Η επόμενη ακολουθία δείχνει πώς δηλώνεται ένας δείκτης και πώς χρησιμοποιούνται οι τελεστές & και * :

```
int x=1, y=2, z[10];      /* δηλώσεις
int *ip;                        μεταβλητών */
ip=&x;                    /* καταχώρηση της διεύθυνσης του x στον δείκτη ip */
y=*ip;                    /* καταχώρηση των περιεχομένων του x στο y μέσω του δείκτη ip */
*ip=0;                    /* καταχώρηση της τιμής 0 στο x μέσω του δείκτη ip */
ip=&z[0];                 /* καταχώρηση της διεύθυνσης του πρώτου στοιχείου του πίνακα z
                               στον δείκτη ip */
```

Στους πίνακες η υπόθεση είναι διαφορετική. Όταν χρησιμοποιείται το όνομα ενός πίνακα σαν όρισμα, η τιμή που μεταβιβάζεται στην συνάρτηση είναι η θέση (ή διεύθυνση) της αρχής του πίνακα (δεν γίνεται αντιγραφή των στοιχείων του πίνακα). Δεικτοδοτώντας την τιμή αυτή, η συνάρτηση μπορεί να προσπελάζει και να αλλάζει οποιοδήποτε στοιχείο του πίνακα.

Παράδειγμα: Να γίνει ένα πρόγραμμα το οποίο να γεμίζει ένα μονοδιάστατο πίνακα 20 ακεραίων με τυχαίες τιμές από 150 μέχρι το 250 και να τον εμφανίζει σε μια γραμμή. Το γέμισμα του πίνακα και η εμφάνιση να γίνουν σε ανεξάρτητες συναρτήσεις όσο το δυνατόν πιο γενικές

```
#include <stdio.h>
#include <stdlib.h>
/* γέμισμα μονοδιάστατου πίνακα 20 ακεραίων με (ψευδο)τυχαίες τιμές
                              από το 150 μέχρι το 250 με συναρτήσεις  */
#define N   20
#define L   150
#define H   250
void rapin1(int a[], int arst, int low, int high);
void emfpin1(int a[], int arst);

main()
{ int  pin[N];
  srand((unsigned)time(NULL));                    //randomize();
  rapin1(pin, N, L, H);
  emfpin1(pin, N);
  return 0;
}

void rapin1(int a[], int arst, int low, int high)
{ int i;
  for (i=0 ; i<arst ; ++i)
     a[i] = low + rand()%(high - low + 1);     //a[i]=low+random(high-low+1);
  return;
}
```

```
void emfpin1(int a[], int arst)
{ int i;
  printf("\n\n Ο πίνακας είναι :\n");
  for (i=0 ; i<arst ; ++i)
     printf("%4d", a[i] );
  return;
}
```

Πίνακες Χαρακτήρων

Ο συνηθέστερος τύπος πίνακα στη C είναι ο πίνακας χαρακτήρων. Κάθε στοιχείο του αποτελείται από έναν χαρακτήρα. Με αυτόν τον τρόπο αποθηκεύονται στην C οι αλφαριθμητικές ακολουθίες (strings). Μάλιστα, στο τέλος κάθε ακολουθίας αποθηκεύεται ένας ακόμα χαρακτήρας ο ′\0′, ο οποίος ονομάζεται μηδενικός χαρακτήρας, και σηματοδοτεί το τέλος του string.

Π.χ.

'Κ'	'α'	'λ'	'η'	'μ'	'έ'	'ρ'	'α'	'\0'					

Ας γράψουμε ένα πρόγραμμα που διαβάζει ένα σύνολο γραμμών κειμένου και εμφανίζει τη μεγαλύτερη και το μήκος της. Το περίγραμμα του προγράμματος έχει ως εξής:

ενόσω (υπάρχει άλλη γραμμή διάβασέ την)
εάν (είναι πιο μεγάλη από την προηγούμενη μεγαλύτερη)
αποθήκευσέ την
αποθήκευσε το μήκος της
εμφάνισε τη μεγαλύτερη γραμμή και το μήκος της

Ας γράψουμε λοιπόν μια συνάρτηση με όνομα getline που θα παίρνει την επόμενη γραμμή εισόδου και θα επιστρέφει το μήκος της ή 0 εάν έχει φτάσει στο τέλος του αρχείου. Ξέρουμε ότι ο ελάχιστος αριθμός χαρακτήρων μιας γραμμής είναι το 1, δηλαδή υπάρχει μόνον ο χαρακτήρας new_line (′\n′).

Μια άλλη συνάρτηση που πρέπει να φτιάξουμε είναι η copy η οποία θα αποθηκεύει την κάθε μεγαλύτερη γραμμή που συναντάμε στον πίνακα χαρακτήρων που έχουμε ορίσει για την μεγαλύτερη γραμμή.

Επίσης, χρειαζόμαστε το κυρίως πρόγραμμα για να ελέγχει τις παραπάνω συναρτήσεις. Ιδού το αποτέλεσμα:

```
#include <stdio.h>

#define MAXLINE   1000              /* μέγιστο μήκος γραμμής εισόδου */

int getline(char line[], int maxline);
void copy(char from[], char to[]);

main()
{
   int len;                         /* μήκος τρέχουσας γραμμής */
   int max;                         /* μεγαλύτερο μήκος μέχρι στιγμής */
   char line[MAXLINE];              /* τρέχουσα γραμμή */
   char longest[MAXLINE];           /* μεγαλύτερη γραμμή μέχρι στιγμής */

   max=0;
```

```
   while ((len = getline(line, MAXLINE))>0)
     if (len >max)
        {
          max=len;
          copy (line, longest);
        }
   if (max>0)                          /* υπήρξε μια γραμμή */
      printf("the line :\n%s\n has %d characters ",longest,max);
   return 0;
}

/* συνάρτηση που διαβάζει μια γραμμή στο s και επιστρέφει το μήκος */

int getline (char s[], int lim)
{
   int i,car;

   for (i=0; i<lim-1 && (car=getchar())!=EOF && car!='\n'; ++i)
      s[i]=car;
   if (car=='\n')
   {
      s[i]=car;
      ++i;
   }
   s[i]='\0';     /* προσθήκη του μηδενικού χαρακτήρα που σηματοδοτεί */
   return i;                   /* το τέλος της ακολουθίας των χαρακτήρων */
}

/* συνάρτηση που αντιγράφει το from στο to */

void copy (char from[], char to[])
{
   int i=0;

   while ((to[i]=from[i])!='\0')
      ++i;
   return;
}
```

Στην getline στο τέλος κάθε πίνακα χαρακτήρων τοποθετούμε τον μηδενικό χαρακτήρα '\0'. Η μετατροπή αυτή χρησιμοποιείται και από την γλώσσα C, στις αλφαριθμητικές σταθερές. Παραδείγματος χάριν το : "Hello World!\n" θα αποθηκευτεί σε ένα πρόγραμμα C σαν πίνακας χαρακτήρων όπως αυτός που ακολουθεί:

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'!'	'\n'	'\0'

Η προδιαγραφή φόρμας %s στην printf περιμένει ότι το αντίστοιχο όρισμα θα είναι αλφαριθμητικό αυτής της μορφής. Ομοίως, η copy βασίζεται στο γεγονός ότι το όρισμα εισόδου τερματίζεται με έναν χαρακτήρα '\0' , αντιγράφει και αυτόν τον χαρακτήρα στο όρισμα εξόδου.

Στο παραπάνω παράδειγμα χάριν συντομίας έχει αγνοηθεί η περίπτωση κατά την οποία μια γραμμή εισόδου ξεπερνάει το μέγιστο όριο χαρακτήρων.

Εσωτερικές - Εξωτερικές Μεταβλητές και Εμβέλεια

Εμβέλεια ενός ονόματος είναι το μέρος του προγράμματος μέσα στο οποίο μπορεί να χρησιμοποιείται το όνομα. Όσες μεταβλητές δηλώνονται μέσα σε μια συνάρτηση λέγονται

ιδιωτικές ή τοπικές και καμιά άλλη συνάρτηση δεν έχει άμεση πρόσβαση σε αυτές. Κάθε τοπική μεταβλητή σε μια συνάρτηση αποκτά υπόσταση μόνον όταν καλείται η συνάρτηση, και εξαφανίζεται με την έξοδο από την συνάρτηση. Γι' αυτό και οι μεταβλητές αυτού του είδους είναι γνωστές και σαν αυτόματες μεταβλητές (ακολουθώντας την ορολογία άλλων γλωσσών).

Υπάρχει και η κατηγορία αποθήκευσης εσωτερικές στατικές (static), στην οποία οι μεταβλητές διατηρούν την τιμή τους ανάμεσα στις κλήσεις.

Επειδή οι αυτόματες μεταβλητές έρχονται και παρέρχονται με την κλήση της συνάρτησης, και δεν διατηρούν την τιμή τους από την μια κλήση στην άλλη, πρέπει να καθορίζεται ρητά η αρχική τους τιμή κάθε φορά που καλούμε την συνάρτηση.

Εναλλακτικά με τις αυτόματες μεταβλητές είναι δυνατόν να ορίζονται μεταβλητές εξωτερικές για όλες τις συναρτήσεις, δηλαδή μεταβλητές που μπορούν να προσπελαστούν με το όνομά τους από όλες τις συναρτήσεις. Μια εξωτερική μεταβλητή πρέπει να **ορίζεται** μια και μόνο φορά, έξω από οποιαδήποτε συνάρτηση. Έτσι, δεσμεύεται γι' αυτήν χώρος αποθήκευσης στη μνήμη. Η μεταβλητή πρέπει επίσης να δηλώνεται σε κάθε συνάρτηση που θέλει να την προσπελάσει, έτσι **δηλώνεται** ο τύπος της μεταβλητής. Η δήλωση μπορεί να είναι μια ρητή εντολή extern ή μπορεί να υπονοείται από τα συμφραζόμενα.

```
#include <stdio.h>     /* το προηγούμενο παράδειγμα με αλλαγές που όμως δεν τις συνιστάμε */

#define MAXLINE       1000          /* μέγιστο μήκος γραμμής εισόδου */

int max;                            /* μεγαλύτερο μήκος μέχρι στιγμής */
char line[MAXLINE];                 /* τρέχουσα γραμμή */
char longest[MAXLINE];              /* μεγαλύτερη γραμμή μέχρι στιγμής */

int getline(void);
void copy(void);

main()                      /* κυρίως πρόγραμμα */
{  int len;                         /* μήκος τρέχουσας γραμμής */
   extern int max;
   extern char longest[];

   max=0;

   while ((len = getline())>0)
      if (len >max)
        {
          max=len;
          copy ();
        }
   if (max>0)                       /* υπήρξε μια γραμμή */
       printf("the line :\n%s\n has %d characters ",longest,max);
   return 0;
}

int getline (void)    /*συνάρτηση που διαβάζει μια γραμμή στο line και επιστρέφει το μήκος της */
{  int i,car;
   extern char line[];

   for (i=0; i<MAXLINE-1 && (car=getchar())!= EOF && car!='\n'; ++i)
       line[i]=car;
   if (car=='\n')
   {
     line[i]=car;
     ++i;
   }
   line[i]='\0';             /* προσθήκη του μηδενικού χαρακτήρα που σηματοδοτεί */
   return i;                              /* το τέλος της ακολουθίας των χαρακτήρων */
}

void copy (void)                    /* συνάρτηση που αντιγράφει το line στο longest */
{  int i=0;
   extern char line[], longest[];
   while ((longest[i]=line[i])!='\0')
       ++i;
   return;
}
```

Οι δηλώσεις extern στις συναρτήσεις main, getline και copy είναι περιττές, επειδή ο ορισμός των εξωτερικών μεταβλητών υπάρχει στην αρχή του ίδιου αρχείου πηγαίου κώδικα. Μάλιστα είναι κοινή πρακτική να τοποθετούνται όλοι οι ορισμοί όλων των εξωτερικών μεταβλητών στην αρχή του αρχείου πηγαίου κώδικα, και μετά να παραλείπονται όλες οι δηλώσεις extern. Αν υπάρχουν περισσότερα αρχεία πηγαίου κώδικα τότε οι δηλώσεις extern για μεταβλητές και συναρτήσεις συγκεντρώνονται σε ένα χωριστό αρχείο που λέγεται επικεφαλίδα (header) και περιλαμβάνεται στην αρχή κάθε αρχείου πηγαίου κώδικα με την βοήθεια της εντολής #include "abc.h". [*Ο προ-επεξεργαστής της C έχει #include (συμπερίληψη) και #define (ορισμός)*]

Η υπερβολική χρήση εξωτερικών μεταβλητών εγκυμονεί κινδύνους.
Τόσο γι' αυτόν τον λόγο, όσο και επειδή καταστρέφει την γενικότητα των δυο χρήσιμων συναρτήσεων (ενσωματώνοντας στον κώδικά τους τα ονόματα των μεταβλητών) η τελευταία έκδοση του προγράμματος θεωρείται υποδεέστερη της πρώτης.

Εκτός από τις στατικές εσωτερικές μεταβλητές που είδαμε πιο πάνω υπάρχουν στατικές εξωτερικές μεταβλητές καθώς και στατικές συναρτήσεις.

Π.χ.
```
static int buf=0;           /* δηλώνεται έξω από κάθε συνάρτηση */
```

Η δήλωση static που εφαρμόζεται σε εξωτερικές μεταβλητές και συναρτήσεις περιορίζει την εμβέλεια του αντικειμένου της μόνο στο υπόλοιπο μεταγλωττιζόμενο αρχείο του πηγαίου κώδικα.

Επίσης η C μας δίνει τη δυνατότητα να δηλώσουμε κάποιες μεταβλητές τύπου register. Μια δήλωση register υποδεικνύει στο μεταγλωττιστή ότι οι μεταβλητές θα χρησιμοποιηθούν πάρα πολύ συχνά, με στόχο την τοποθέτησή τους σε καταχωρητές για πιο γρήγορη πρόσβαση. Βέβαια, αν δεν υπάρχει διαθέσιμος χώρος η υπόδειξη αυτή θα αγνοηθεί.

Σημειώσεις :

- Αν θέλουμε μια συνάρτηση να μην επιστρέφει τιμή, τότε χρησιμοποιούμε τον τύπο void.
- Αν παραληφθεί ο επιστρεφόμενος τύπος μιας συνάρτησης, χρησιμοποιείται ο int.
- Η εντολή return (η οποία μπορεί να ακολουθείται από οποιαδήποτε παράσταση) είναι ο μηχανισμός για την επιστροφή μιας τιμής από την καλούμενη συνάρτηση στην καλούσα.
- Ο έλεγχος επιστρέφει στην καλούσα συνάρτηση χωρίς τιμή, όταν η εκτέλεση ξεπεράσει το τέλος της συνάρτησης φτάνοντας το τελευταίο δεξιό άγκιστρο.
- Δεν είναι παράτυπο για μια συνάρτηση να επιστρέφει τιμή από ένα σημείο και να μην επιστρέφει από ένα άλλο, όμως πιθανότατα αποτελεί ένδειξη ότι υπάρχουν προβλήματα.
- Αν δεν υπάρχει πρωτότυπο για μια συνάρτηση, η συνάρτηση δηλώνεται σιωπηλά από την πρώτη της εμφάνιση μέσα σε παράσταση.

Για να κατανοήσουμε τις συναρτήσεις ας φτιάξουμε ακόμα ένα πρόγραμμα που θα εμφανίζει κάθε γραμμή της εισόδου του εφόσον αυτή περιέχει ένα συγκεκριμένο συνδυασμό χαρακτήρων και πόσες γραμμές ήταν συνολικά. Πρόκειται για μια ειδική περίπτωση του προγράμματος grep του UNIX.
Με λόγια η εργασία χωρίζεται καθαρά σε τρία κομμάτια:

Ενόσω (υπάρχει κι άλλη γραμμή)
εάν (η γραμμή περιέχει το συνδυασμό)
εμφάνισέ την
εμφάνισε τον συνολικό αριθμό γραμμών που βρέθηκαν να περιέχουν τον συνδυασμό

Ο κώδικας του παραπάνω προγράμματος σε γλώσσα C είναι ο εξής:

```
#include <stdio.h>
#define     MAXLINE     1000

int getline (char line[], int max);
int strindex(char source[], char searchfor[]);

char pattern[]="tei";

main ()
{
   char line[MAXLINE];
   int found=0;
   while (getline(line,MAXLINE)>0)
      if (strindex(line,pattern)>=0)
        {   printf("%s",line);
            found++;
        }
   printf("%d γραμμές περιείχαν τον συνδυασμό %s", found, pattern);
   return 0;
}

int getline (char s[], int lim)
{
   int i,car;
   i=0;
   while (--lim>0 && (car=getchar())!=EOF && car!='\n')
           s[i++]=car;
   if (car=='\n')
           s[i++]=car;
  s[i]='\0';
  return i;
}

int strindex (char s[], char t[])
{
   int i,j,k;
   for (i=0 ; s[i]!='\0' ; i++)
      {
          for (j=i,k=0; t[k]!='\0' && s[j]==t[k] ; j++, k++)
            ;
          if (k>0 && t[k]=='\0')
            return i;
      }
   return -1;
}
```

Δείκτες και ορίσματα συναρτήσεων

Η παρακάτω συνάρτηση πραγματοποιεί την αντιμετάθεση των περιεχομένων δύο μεταβλητών ακέραιου τύπου.

```
void  swap  (int  *px,  int
*py)
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

Η εντολή κλήσης της συνάρτησης είναι η εξής:

```
swap (&a, &b);
```

όπου *a* και *b* είναι μεταβλητές ακέραιου τύπου της καλούσας συνάρτησης.

Δείκτες και πίνακες

Στην C υπάρχει τόσο ισχυρός δεσμός ανάμεσα στους δείκτες διεύθυνσης (pointers) και στους πίνακες, που οι δείκτες διευθύνσεων και οι πίνακες θα πρέπει να εξεταστούν μαζί. Οποιαδήποτε λειτουργία μπορεί να γίνει με δείκτες πινάκων, μπορεί να γίνει και με δείκτες διευθύνσεων.

```
int x;          // δήλωση μιας μεταβλητής τύπου ακεραίου
int a[10];      // δήλωση ενός πίνακα ακεραίων a μεγέθους 10
int *pa;        // δήλωση ενός δείκτη σε ακέραιο

pa = &a[0];     // καταχώρηση της διεύθυνσης του a[0] στο δείκτη

x = *pa;        // αντιγραφή των περιεχόμενων του a[0] στο x δια μέσου του δείκτη διεύθυνσης pa
```

Αν ο pa δείχνει ένα συγκεκριμένο στοιχείο ενός πίνακα, τότε εξ ορισμού ο pa+1 δείχνει το επόμενο στοιχείο, ο pa + i δείχνει i στοιχεία μετά τον pa, και ο pa - i δείχνει i στοιχεία πριν.
Έτσι, αν ο pa δείχνει το a[0], η *(pa+1) αναφέρεται στα περιεχόμενα του a[1].
Οι παρατηρήσεις αυτές ισχύουν ανεξάρτητα από τον τύπο ή το μέγεθος των μεταβλητών του πίνακα a. Η σημασία του "προσθέτω 1 σε κάποιο δείκτη" και, κατ' επέκταση, ολόκληρης της αριθμητικής δεικτών, είναι ότι ο pa+1 δείχνει το επόμενο αντικείμενο.

Η αντιστοιχία ανάμεσα στη δεικτοδότηση και την αριθμητική των δεικτών είναι πολύ στενή. Εξ ορισμού, η τιμή μιας μεταβλητής ή παράστασης τύπου πίνακα είναι η διεύθυνση του πρώτου στοιχείου του πίνακα (με δείκτη 0). Έτσι μετά την αντικατάσταση pa = &a[0] οι pa και a έχουν τις ίδιες τιμές. Αφού το όνομα ενός πίνακα είναι συνώνυμο με τη θέση του πρώτου του στοιχείου, η πιο πάνω αντικατάσταση μπορεί να γραφεί σαν pa = a;

Ακόμη πιο εκπληκτικό είναι το γεγονός ότι η αναφορά στο a[i] μπορεί να γραφτεί και σαν *(a+i). Υπολογίζοντας το a[i], η C το μετατρέπει αμέσως σε *(a+i). Οι δυο μορφές είναι ισοδύναμες. Αν εφαρμόσουμε τον τελεστή & και στα δυο ισοδύναμα μέλη, προκύπτει ότι οι &a[i] και a+i είναι επίσης ταυτόσημες. Από την άλλη πλευρά, αν ο pa είναι δείκτης διεύθυνσης, οι παραστάσεις μπορούν να τον χρησιμοποιούν με δείκτη στοιχείου πίνακα, δηλαδή το pa[i] είναι ταυτόσημο με το *(pa+i). Με λίγα λόγια, μια παράσταση με πίνακα και δείκτη στοιχείου πίνακα είναι ισοδύναμη με μια παράσταση γραμμένη σαν δείκτης διεύθυνσης και απόσταση.

Υπάρχει μια διαφορά μεταξύ ονόματος πίνακα και δείκτη, που πρέπει να την έχετε υπόψη. Ο δείκτης είναι μεταβλητή, κι έτσι οι pa = a και pa++ είναι έγκυρες εκφράσεις. Όμως ένα όνομα πίνακα δεν είναι μεταβλητή, και έτσι εκφράσεις όπως οι a = pa και a++ δεν είναι έγκυρες.

Όταν ένα όνομα πίνακα μεταβιβάζεται σε μια συνάρτηση, αυτό που μεταβιβάζεται είναι η θέση του αρχικού στοιχείου. Μέσα στην καλούμενη συνάρτηση το όρισμα αυτό είναι τοπική μεταβλητή, και έτσι μια παράμετρος που είναι όνομα πίνακα είναι δείκτης, δηλαδή που περιέχει μια διεύθυνση. Χρησιμοποιώντας αυτό το γεγονός μπορούμε να φτιάξουμε μια άλλη εκδοχή της συνάρτησης που επιστρέφει το μήκος ενός αλφαριθμητικού.

```
/* νέα εκδοχή με pointers */
int strlen (char *s)
{  int n;
   for (n=0; *s!='\0' ; n++)
      s++;
   return n;
}
```

```
/* εκδοχή με πίνακες */
int strlen (char s[])
{   int n=0;
   while( s[n]!='\0')
     n++;
   return n;
}
```

Σαν τυπικές παράμετροι σε ορισμό συνάρτησης οι char s[] και char *s είναι ισοδύναμες, όμως προτιμάμε την δεύτερη, γιατί αναφέρει ρητά ότι η παράμετρος είναι δείκτης. Όταν μεταβιβάζεται ένα όνομα πίνακα σε μια συνάρτηση, τότε αυτή, ανάλογα με το τι την εξυπηρετεί, μπορεί να θεωρήσει ότι της δόθηκε είτε ένας πίνακας είτε ένας δείκτης, και να τον χειριστεί αντίστοιχα. Επίσης, μπορεί να χρησιμοποιήσει και τους δύο συμβολισμούς, εφόσον αυτό φαίνεται να είναι σκόπιμο και σαφές.

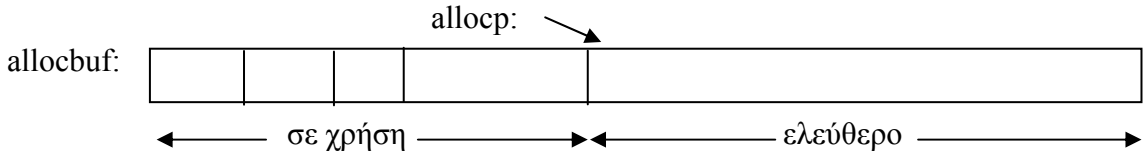
Είναι δυνατόν να μεταβιβάσετε σε μια συνάρτηση μόνο ένα μέρος πίνακα, μεταβιβάζοντας ένα δείκτη στην αρχή του υποπίνακα.

Π.χ. αν ο a είναι πίνακας τότε οι f (&a[2]) και f (a+2) μεταβιβάζουν και οι δύο στην συνάρτηση f την διεύθυνση του υποπίνακα που αρχίζει από το σημείο a[2]. Μέσα στην f, η δήλωση της παραμέτρου μπορεί να γραφεί

```
f (int arr[])                    ή και    f (int *arr)
{                                         {
   …                                         …
}                                         }
```

Παράδειγμα πάνω στην Αριθμητική Διευθύνσεων

Στοιχειώδης κατανεμητής μνήμης με ρουτίνες για δέσμευση *alloc(n)* και απελευθέρωση *afree(p)* μνήμης.




```
#define ALLOCSIZE 1000
static char allocbuf[ALLOCSIZE];
static char *allocp = allocbuf;

char *alloc(int n)
{
     if (allocbuf + ALLOCSIZE - allocp >= n)
        {
            allocp += n;
            return allocp-n;
        }
     else
      return 0;                              //NULL or 0 or expression
}

void afree(char *p)
{
     if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
      allocp=p;
     return;
}
```

Σημείωση: Έγκυρες πράξεις με δείκτες είναι η απόδοση τιμής με δείκτη ίδιου τύπου, η πρόσθεση ή αφαίρεση δείκτη και ακεραίου, η αφαίρεση ή σύγκριση δυο δεικτών για μέλη του ίδιου πίνακα, και η αντικατάσταση η σύγκριση με το μηδέν. Κάθε άλλη αριθμητική πράξη με δείκτες δεν είναι έγκυρη.

Δείκτες χαρακτήρα και συναρτήσεις

Μια αλφαριθμητική σταθερά, που γράφεται με τη μορφή : "Καλημέρα"
είναι ένας πίνακας χαρακτήρων. Στην εσωτερική αναπαράσταση ο πίνακας τερματίζεται με τον μηδενικό χαρακτήρα '\0', για να μπορούν τα προγράμματα να βρίσκουν το τέλος του. Επομένως, το μήκος που αποθηκεύεται είναι κατά ένα μεγαλύτερο από τον αριθμό των χαρακτήρων που βρίσκονται μέσα στα διπλά εισαγωγικά.

Όταν σε κάποιο πρόγραμμα ή συνάρτηση εμφανίζεται ένα παρόμοιο αλφαριθμητικό, η προσπέλασή του γίνεται με δείκτη χαρακτήρα που δείχνει την αρχή του πίνακα χαρακτήρων.

Οι αλφαριθμητικές σταθερές δεν χρειάζεται να είναι ορίσματα συναρτήσεων. Αν η pmessage δηλωθεί σαν

```
char     *pmessage;
```

η εντολή

```
pmessage = "Please Wait!!";
```

αποδίδει στην pmessage ένα δείκτη για τον πίνακα χαρακτήρων. Δεν πρόκειται για αντιγραφή αλφαριθμητικών, αφού χρησιμοποιούνται μόνο δείκτες. Η C δεν διαθέτει τελεστές για το χειρισμό ενός ολόκληρου αλφαριθμητικού μονομιάς.

Υπάρχει μια σημαντική διαφορά ανάμεσα στους ορισμούς:

```
char amessage[] = "ΚΑΛΗΜΕΡΑ";     /* πίνακας με το αλφαριθμητικό και το '\0' */

char *pmessage  = "ΚΑΛΗΜΕΡΑ";     /* δείκτης */
```

Παρακάτω θα δούμε και άλλες πλευρές των δεικτών και των πινάκων, μελετώντας την συνάρτηση strcopy(d,s) που αντιγράφει το αλφαριθμητικό s στο d. Καλό θα ήταν να μπορούσαμε να γράφαμε απλώς d = s, αλλά έτσι αντιγράφεται ο δείκτης και όχι οι χαρακτήρες. Για να αντιγράψουμε τους χαρακτήρες, χρειαζόμαστε ένα βρόχο.

```
/* strcopy :  αντιγραφή του s στο d , εκδοχή με πίνακες */

void strcopy(char *d, char *s)
{
       int i=0;
       while ((d[i]=s[i])!='\0')
              i++;
       return;
}
```

```
/* strcopy :  αντιγραφή του s στο d , εκδοχή με δείκτες διεύθυνσης (pointers) */

void strcopy(char *d, char *s)
{
     while ((*d=*s)!='\0')
     {
              d++;
             s++;
     }
     return;
}
```

ή ακόμα καλύτερα:

```
void strcopy(char *d, char *s)
{
     while ((*d++=*s++)!='\0')    /* ή και while (*d++=*s++){παράσταση =0} */
       ;
     return;
}
```

Η συνάρτηση που κάνει αυτή τη δουλειά, της πρότυπης βιβλιοθήκης <string.h> είναι η :

```
char *strcpy(char *dest, const char *src);
```

επιστρέφει σαν τιμή της συνάρτησης το αλφαριθμητικό προορισμού.

Μια άλλη ρουτίνα που θα εξετάσουμε είναι η strcomp(s1,s2) η οποία συγκρίνει τα αλφαριθμητικά s1 και s2, και επιστρέφει τιμή αρνητική, μηδενική, ή θετική, ανάλογα με την λεξικογραφική σχέση τους. Η τιμή προκύπτει από την αφαίρεση των χαρακτήρων στην πρώτη θέση που δεν συμφωνούν τα δυο αλφαριθμητικά.

```
/* strcomp : επιστρέφει <0 αν s1<s2,  0 αν s1==s2, και >0 αν s1>s2
                                                    εκδοχή με πίνακες */

int strcomp(char *s1, char *s2)
{           int i;
            for (i=0; s1[i]==s2[i]; i++)
                  if (s1[i]=='\0')
                        return 0;

            return s1[i]-s2[i];
}
```

```
/* strcomp : επιστρέφει <0 αν s1<s2,  0 αν s1==s2, και >0 αν s1>s2
                              εκδοχή με δείκτες διεύθυνσης (pointers) */

int strcomp(char *s1, char *s2)
{
             for (     ; *s1==*s2 ;  s1++, s2++)
                  if (*s1=='\0')
                        return 0;

            return *s1-*s2;
}
```

Επίσης, στη <string.h> υπάρχει η συνάρτηση

```
int strcmp(const char *s1, const char *s2);
```

που συγκρίνει το ένα αλφαριθμητικό με το άλλο ξεκινώντας από τον πρώτο χαρακτήρα του καθενός και συνεχίζοντας με τους επόμενους μέχρι οι αντίστοιχοι χαρακτήρες να διαφέρουν ή μέχρι να φθάσει το τέλος των αλφαριθμητικών.
Αν το s1 είναι μικρότερο (δηλαδή προηγείται αλφαβητικά) του s2 τότε επιστρέφει τιμή μικρότερη του μηδενός, αν το s1 είναι μεγαλύτερο (δηλαδή έπεται αλφαβητικά) του s2 τότε επιστρέφει τιμή μεγαλύτερη του μηδενός, ενώ αν τα s1 και s2 ταυτίζονται επιστρέφει την τιμή μηδέν.

Δομές Δεδομένων με την C

Δομή (data structure ή structure) είναι η συλλογή από μια ή περισσότερες μεταβλητές, πιθανώς διαφορετικών τύπων, που ομαδοποιούνται με ένα μόνο όνομα για ευκολία στον χειρισμό τους. Οι μεταβλητές αυτές αποτελούν τα μέλη (members) της δομής.

Σε άλλες γλώσσες προγραμματισμού συναντάμε τις ίδιες έννοιες με διαφορετική ορολογία, όπως για παράδειγμα στην Pascal που μιλάμε για εγγραφές (records) και για τα πεδία (fields) που τις συνιστούν.
Στη C μπορούμε να δηλώσουμε μεταβλητές τύπου δομής με τρεις τρόπους. Παρακάτω παραθέτουμε αυτούς τους τρόπους με την βοήθεια ενός παραδείγματος.

Έστω ότι θέλουμε μεταβλητές τύπου δομής με τρία μέλη για την ενοποίηση του ονόματος, του αρχικού πατρώνυμου και του επωνύμου.

α) άμεση δήλωση μεταβλητών τύπου δομής

```
struct {
        char first[10];          /*  μικρό όνομα  */
        char midinit;            /*  αρχικό πατρώνυμο  */
        char last[15];           /*  επώνυμο  */
} aname, bname;                          // ← structure variables
```

β) δήλωση δομής με ετικέτα (tag) και κατόπιν δήλωση μεταβλητών τύπου δομής χρησιμοποιώντας την ετικέτα οπουδήποτε είναι αναγκαίο.

```
struct  Namet {                          // ← tag
        char first[10];
        char midinit;
        char last[15];
};

 struct  Namet  aname, bname;
```

γ) ορισμός νέου τύπου δεδομένων και κατόπιν δήλωση μεταβλητών τύπου δομής χρησιμοποιώντας τον νέο τύπο δεδομένων.

```
typedef struct {
        char first[10];
        char midinit;
        char last[15];
}NameType;

 NameType  aname, bname;
```

Αυτός ο τελευταίος τρόπος είναι ο βολικότερος στην περίπτωση που είναι απαραίτητη η μεταφορά τέτοιων δομών δια μέσου των ορισμάτων των συναρτήσεων.
Σε όλες τις περιπτώσεις για να προσπελαστεί ένα μέλος μιας μεταβλητής τύπου δομής χρησιμοποιείται ο τελεστής τελεία (.), π.χ. aname.midinit = 'Σ' ;

Άσκηση με δομές δεδομένων: Να γραφεί ένα πρόγραμμα, το οποίο να ορίζει τον τύπο μιας δομής με τρία μέλη: α) το όνομα ενός προϊόντος, β) την ποσότητα που βρίσκεται στην αποθήκη και γ) το κόστος του ανά τεμάχιο. Επίσης, να ορισθεί ένας γραμμικός πίνακας με εκατό στοιχεία τύπου όπως της παραπάνω δομής το καθένα. Το πρόγραμμα θα πρέπει να γεμίζει τον πίνακα με στοιχεία από την κονσόλα και να τον εμφανίζει στην οθόνη υπό μορφή κατάστασης.

```
#include <stdio.h>
#define N   100                 /* το μέγεθος του πίνακα */

typedef struct {                /* ορισμός νέου τύπου με όνομα Proion */
            char on[20];        /* ονομασία  προϊόντος  */
            float tt;           /* τιμή ανά τεμάχιο     */
            int  po;            /* ποσότητα σε τεμάχια  */
}Proion;

void eisag (Proion x[]);
void emfan (Proion x[]);
```

```
main()
{ Proion ap[N];
  eisag(ap);
  emfan(ap);
  return 0;
}

void eisag( Proion x[])       /* συνάρτηση για την εισαγωγή των στοιχείων */
{ int i;
  for (i=0; i<N;i++)
  { printf ("Δώστε για το %do προϊόν την ονομασία : ", i+1);
    scanf ("%s", x[i].on);
    printf ("Δώστε τιμή ανά τεμάχιο : ");
    scanf ("%f", &x[i].tt);
    printf ("Δώστε ποσότητα : ");
    scanf ("%d", &x[i].po);
  }
  return;
}

void emfan (Proion x[]);      /* συνάρτηση για την εμφάνιση των στοιχείων */
{ int i;
  printf ("\n\nA/A  ONOMΑΣΙΑ ΤΙΜΗ/ΤΕΜΑΧΙΟ ΠΟΣΟΤΗΤΑ");
  for (i=0; i<N;i++)
    printf ("\n%3d %10s %12.2f %8d", i+1, x[i].on,  x[i].tt, x[i].po);
  return;
}
```

Αναδρομικότητα (Recursion)

Έχουμε δύο είδη αναδρομής την **άμεση** (όταν μια συνάρτηση καλεί τον εαυτό της) και την **έμμεση** (όταν μια συνάρτηση καλεί μια άλλη συνάρτηση η οποία κατόπιν καλεί την πρώτη συνάρτηση).

Μια συνάρτηση που καλεί τον εαυτόν της ονομάζεται αναδρομική ή λέμε ότι έχει αναδρομική μορφή.

Για παράδειγμα ας φτιάξουμε την συνάρτηση του παραγοντικού σε αναδρομική μορφή.

```
long int parag(int n)
{ long p;
  if (n==0)
    p=1;
  else
    p=parag(n-1)*n;                /* εντολή κλήσης του εαυτού της */
  return p;
}
```

Παρατηρείστε ότι δεν υπάρχει εντολή επανάληψης ούτε αρχικοποίηση της τοπικής μεταβλητής p. Επίσης, προσέξτε ότι υπάρχει μια εντολή ελέγχου η οποία δίνει το αποτέλεσμα για μια γνωστή τιμή του n, ενώ για όλες τις άλλες χρησιμοποιεί έναν επαγωγικό τύπο.

Οι συναρτήσεις παραγοντικού, ύψωσης σε δύναμη και Fibonacci σε μη αναδρομική μορφή και σε αναδρομική μορφή. Προσέξτε ότι ο τρόπος κλήσης της συνάρτησης είναι ίδιος και στις δύο μορφές.

```
  /* ΜΗ ΑΝΑΔΡΟΜΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ */

#include <stdio.h>

long int parag(int);
float power(float,int);
int fib(int);

main()
{ int k; float bash;
  long int par;
  printf("\nΔώστε αριθμό : ");
  scanf("%d",&k);
  par=parag(k);
  printf("\nΤο %d! είναι %ld\n",k,par);

  bash=2.5;
  printf("\n%f\n",power(bash,3));

  printf("\n%d\n",fib(8));

  return 0;
}


long int parag(int n)
{ int i;
  long p=1;
  for(i=1;i<=n;i++)
       p=p*i;
  return p;
}



float power(float b, int e)
{ int i;
  float p=1.0;
  for(i=0;i<e;i++)
       p=p*b;
  return p;
}



int fib(int n)
{ int i, f=0, a=0, b=1;
  if (n==2)
    f=1;
  for(i=3;i<=n;i++)
    { f=a+b;
      a=b;
      b=f;
    }
  return f;
}
```

```
  /* ΑΝΑΔΡΟΜΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ */

#include <stdio.h>

long int parag(int);
float power(float,int);
int fib(int);

main()
{ int k; float bash;
  long int par;
  printf("\nΔώστε αριθμό : ");
  scanf("%d",&k);
  par=parag(k);
  printf("\nΤο %d! είναι %ld\n",k,par);

  bash=2.5;
  printf("\n%f\n",power(bash,3));

  printf("\n%d\n",fib(8));

  return 0;
}


long int parag(int n)
{ long p;
  if (n==0)
    p=1;
  else
    p=parag(n-1)*n;
  return p;
}


float power(float b, int e)
{ float p;
  if (e==0)
    p=1;
  else
    p=power(b,e-1)*b;
  return p;
}


int fib(int n)
{ int f;
  if (n==1)
    f=0;
  else
    if (n==2)
      f=1;
    else
      f=fib(n-1)+fib(n-2);
  return f;
}
```

Μέθοδοι Ταξινόμησης (Sorting)

Συχνά χρειάζεται να διατάξουμε σύνολα αντικειμένων σε μια καθορισμένη σειρά. Η διαδικασία της τοποθέτησης των αντικειμένων σε μια καθορισμένη σειρά ονομάζεται ταξινόμηση. Σχεδόν πάντα τα αντικείμενα είναι δομές (εγγραφές) που αποτελούνται από περισσότερα μέλη (πεδία). Το πεδίο (ή τα πεδία) που χρησιμοποιείται για να γίνει η ταξινόμηση των αντικειμένων λέγεται κλειδί (key) της ταξινόμησης. Έτσι λοιπόν, ταξινομούμε τις εγγραφές με τέτοιο τρόπο ώστε τα κλειδιά τους να είναι σε αύξουσα διάταξη, ή σε φθίνουσα διάταξη.

Εμείς, χωρίς να χάνουμε από την γενικότητα, όσο θα μελετάμε τις μεθόδους ταξινόμησης και όχι τις εφαρμογές τους, μπορούμε να χρησιμοποιούμε πολύ απλουστευμένα αντικείμενα, όπως αριθμούς, που είναι κλειδιά του εαυτού τους.

Απ' ευθείας μέθοδοι ταξινόμησης (straight)

Απ' ευθείας επιλογή (selection)

Για να εξηγήσουμε την μέθοδο «απ' ευθείας επιλογή», ας υποθέσουμε, ότι θέλουμε να ταξινομήσουμε σε αύξουσα διάταξη μια ακολουθία από ακέραιους, έστω αυτήν που ακολουθεί:

14 20 3 8 17 11 5 16

Αρχικά, ανταλλάσσουμε τον μικρότερο από του ακέραιους με τον πρώτο, και έτσι καταφέραμε να τοποθετήσουμε ήδη το πρώτο στοιχείο στην σωστή θέση.

3 20 14 8 17 11 5 16

Στην συνέχεια εφαρμόζουμε την ίδια διαδικασία και στην ακολουθία που αρχίζει από τον δεύτερο μέχρι τον τελευταίο. Επιλέγουμε λοιπόν το 5 και το ανταλλάζουμε με το 20. Μ' αυτόν τον τρόπο έχουμε τοποθετήσει τους δύο πρώτους αριθμούς στην σωστή θέση.

3 5 14 8 17 11 20 16

Συνεχίζουμε ομοίως μέχρι να τοποθετήσουμε και τον προτελευταίο. Ο τελευταίος θα είναι στην θέση του ούτως ή άλλως.

Η συνάρτηση sortbyselection() που ακολουθεί ταξινομεί σε αύξουσα διάταξη έναν πίνακα ακεραίων υλοποιώντας την μέθοδο που μόλις περιγράψαμε.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define M 15

void sortbyselection(int a[], int n);

main()
{ int pin[M],i;

  srand((unsigned)time(NULL));                // randomize();

  for (i=0; i<M;i++)
    pin[i]=rand()%100;                        // pin[i]=random(100);

  printf("\n\n Αρχικά ο πίνακας είναι: \n");
  for (i=0; i<M; i++)
    printf("%4d",pin[i]);

  sortbyselection(pin, MAX);

  printf("\n\n Ο πίνακας ταξινομημένος είναι: \n");
  for (i=0; i<M;i++)
    printf("%4d",pin[i]);
  printf("\n\n");

  return 0;
}
```

```
void sortbyselection(int a[], int n)
{
  int i,j,k,min;
  for(i=0;i<n-1;i++)
  {
         k=i; min=a[k];
         for (j=i+1;j<n;j++)
                if (a[j]<min)
                {
                       k=j;
                       min=a[k];
                }
         a[k]=a[i];
         a[i]=min;
  }
  return;
}
```

Παρατηρήστε ότι ο χρόνος εκτέλεσης (running time) του αλγόριθμου είναι $O(n^2)$. Υπάρχουν δυο επάλληλοι βρόχοι στον κώδικα. Αν διπλασιάσουμε το n το πεδίο ορισμού της μεταβλητής ελέγχου i στον εξωτερικό βρόχο περίπου θα διπλασιαστεί. Χρησιμοποιούμε την λέξη περίπου διότι το πεδίο ορισμού θα πολλαπλασιαστεί με το συντελεστή (2n-1)/(n-1) αντί του 2. Ομοίως, στον εσωτερικό βρόχο το μέσο πεδίο ορισμού της μεταβλητής ελέγχου j θα διπλασιαστεί. Έτσι, παρατηρούμε λοιπόν ότι η αύξηση του n κατά ένα συντελεστή 2 συνεπάγεται αύξηση του συνολικού χρόνου λειτουργίας κατά έναν συντελεστή 4. Γενικότερα η αύξηση του n κατά ένα συντελεστή k συνεπάγεται αύξηση του συνολικού χρόνου λειτουργίας κατά έναν συντελεστή k^2. Αυτό εκφράζεται με συντομία λέγοντας ότι ο χρόνος λειτουργίας του αλγόριθμου είναι $O(n^2)$ ή ισοδύναμα, ότι ο αλγόριθμος έχει πολυπλοκότητα (χρονική συμπεριφορά, time complexity) n^2.

Για πολλά προγράμματα ο χρόνος λειτουργίας $T(n)$ είναι μια συνάρτηση του πλήθους n των δεδομένων εισαγωγής (επεξεργασίας). Για παράδειγμα, ένα πρόγραμμα ταξινόμησης μιας ακολουθίας n αριθμών μπορεί να έχει χρόνο λειτουργίας

$$T(n) = cn^2$$

όπου c είναι μια σταθερά. Αν και η τιμή c της σταθεράς δεν μας είναι αδιάφορη, θεωρείται πολύ πιο σημαντικό το γεγονός ότι ο χρόνος λειτουργίας είναι συνάρτηση δευτέρου βαθμού του n. Γι' αυτό χρησιμοποιούμε τον συμβολισμό του λεγόμενου «Όμικρον» ("big-oh") και, όσον αφορά το παράδειγμά μας, λέμε ότι ο χρόνος λειτουργίας $T(n)$ είναι $O(n^2)$.

Επανερχόμενοι στις μεθόδους ταξινόμησης θα μπορούσαμε να γράψουμε τον παραπάνω αλγόριθμο λίγο διαφορετικά, με μικρά πλεονεκτήματα και μειονεκτήματα, ως εξής:

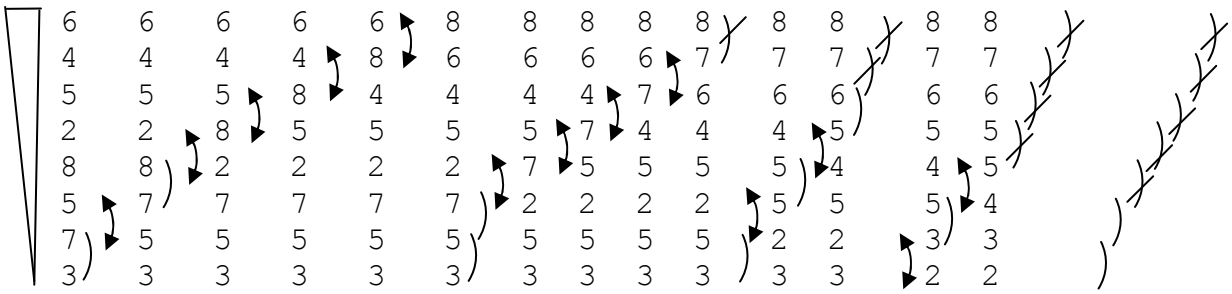
```
void sortbyselection2(int a[], int n)
{
  int i,j,k,min;
  for(i=0;i<n-1;i++)
  {
         k=i;
         for (j=i+1;j<n;j++)
                if (a[j]<a[k])
                       k=j;
         min=a[k];
         a[k]=a[i];
         a[i]=min;
  }
  return;
}
```

Σημείωση: Για καλύτερη εξοικείωση με την μέθοδο, χρησιμοποιήστε την για ταξινόμηση πιο σύνθετων αντικειμένων (δομές) με κλειδί αρχικά κάποιο αριθμητικό πεδίο και στην συνέχεια αλφαριθμητικό.

Μέθοδος Ταξινόμησης της φυσαλίδας (Bubble Sort)

Αυτή η μέθοδος ταξινόμησης σαρώνει κατ' επανάληψη τον πίνακα από την αρχή προς το τέλος συγκρίνοντας ανά δύο όλα τα στοιχεία που βρίσκονται σε διαδοχικές θέσεις και αν δεν είναι σωστά τοποθετημένα το ένα σε σχέση με το άλλο, τα αντιμεταθέτει. Εάν σε ένα ολόκληρο σάρωμα από την αρχή ως το τέλος δεν χρειαστεί να γίνει καμιά αντιμετάθεση, τερματίζεται η ταξινόμηση διότι έχει ολοκληρωθεί.

Θα κατανοήσουμε την μέθοδο καλύτερα μέσω του παρακάτω παραδείγματος. Έστω ότι έχουμε έναν πίνακα ακεραίων που πρέπει να ταξινομήσουμε:



Μπορούμε να συμπληρώσουμε το πρόγραμμα της προηγουμένης άσκησης με τις δομές δεδομένων (σελ.56) με τον πίνακα των προϊόντων γράφοντας μια συνάρτηση για την ταξινόμηση του πίνακα ως προς το πεδίο της ποσότητας του προϊόντος σε φθίνουσα διάταξη.

```
void taxin(Proin x[])
{ int i, top, top1, tax_ok;
  Proion h;
  top= N-1;
  do { tax_ok = TRUE;
       for (i=0; i<top ; i++)
          if (x[i].po<x[i+1].po)
             { h=x[i]; x[i]=x[i+1]; x[i+1]=h;
               tax_ok=FALSE;
               top1=i;
             }
        top=top1;
  }while (tax_ok==FALSE);
  return;
}
```

Μέθοδοι Αναζήτησης (Searching)

Στόχος των αλγόριθμων αναζήτησης είναι η πιο αποτελεσματική συλλογή πληροφοριών που είναι αποθηκευμένες στην μνήμη ενός Η/Υ.

Το πρόβλημα της αναζήτησης, στην πιο γενική του μορφή, αναφέρεται στην αναζήτηση κάποιου στοιχείου ενός συνόλου, που ικανοποιεί μια συνθήκη ή ένα σύνολο προδιαγραφών και πιο συγκεκριμένα έχει ορισμένα χαρακτηριστικά ή ιδιότητες.

Σειριακή αναζήτηση (sequential search)

Η πιο απλή μέθοδος αναζήτησης είναι αυτή της σειριακής αναζήτησης όπου συγκρίνονται διαδοχικά, «με την σειρά», τα κλειδιά των αντικειμένων του συνόλου στο οποίο διενεργείται η αναζήτηση με το ζητούμενο κλειδί. Η διαδικασία τερματίζεται όταν διαπιστωθεί ότι κάποιο από τα κλειδιά είναι ίσο με το ζητούμενο ή όταν εξετασθούν όλα τα κλειδιά. Η συνάρτηση *seqsearch* που ακολουθεί υλοποιεί τον παραπάνω αλγόριθμο.

```
int seqsearch (int *a, int n, int key);
#define MAX 20

main()
{ int i,thesh;
  int pin[MAX+1];             //Μια παραπάνω θέση για τις ανάγκες της σειριακής αναζήτησης
  for (i=0; i<MAX;i++)
         pin[i]= …
  …
  thesh=seqsearch(pin, MAX, 30);
  if (thesh>-1)
         printf("Found at position %d\n\n",thesh);
  else
         printf("NOT FOUND.\n\n");
  …
  return 0;
}

int seqsearch (int *a, int n, int key)
{ int d=0;
  a[n]=key;
  while (a[d]!=key)
         d++;
  if (d==n)
         return -1;           //Δεν υπάρχει
  else
         return d;            //Υπάρχει στην θέση d
}
```

Προσέξτε ότι αρχικά καταχωρεί μια θέση μετά το τέλος του πίνακα το κλειδί αναζήτησης. Αυτό γίνεται για να απλοποιηθεί η συνθήκη ελέγχου της εντολής επανάληψης.

Ένα μειονέκτημα του αλγόριθμου είναι ότι σε περιπτώσεις ανεπιτυχών αναζητήσεων ελέγχεται κάθε κλειδί. Όταν ανεπιτυχείς αναζητήσεις συμβαίνουν συχνά είναι προτιμότερο να ταξινομούμε τα κλειδιά σε αύξουσα διάταξη και να εφαρμόζουμε τον τροποποιημένο σειριακό αλγόριθμο αναζήτησης, όπως έχει υλοποιηθεί στην συνάρτηση *seqsearch2* που ακολουθεί:

```
int seqsearch2(int *a, int n, int key)
{ int d=0;
  a[n]=INT_MAX;               // 2147483647 Maximum (signed) int value
  while (a[d]<key)
         d++;
  if (d<n && a[d]==key)
         return d;            //Υπάρχει στην θέση d
  else
         return -1;   //Δεν υπάρχει
}
```

Και για τις δύο σειριακές τεχνικές αναζήτησης ισχύει ότι αν έχουμε *n* κλειδιά (στοιχεία) στην χειρότερη περίπτωση θα εκτελεστούν *(n+1)* συγκρίσεις, στην μέση περίπτωση έχουμε *(n+1)/2*, και στην καλύτερη έχουμε μια σύγκριση. Με άλλα λόγια η πολυπλοκότητα της μέσης και της χειρότερης περίπτωσης είναι *O(n)*.

Υπάρχουν άλλοι αλγόριθμοι, όπως θα δούμε παρακάτω, που προτιμώνται επειδή είναι ταχύτεροι (ειδικά για μεγάλο αριθμό στοιχείων), οι οποίοι όμως υποθέτουν ότι το σύνολο των κλειδιών είναι γραμμικώς διατεταγμένο.

Το γεγονός όμως ότι υπάρχουν άλλοι αλγόριθμοι αναζήτησης με καλύτερη πολυπλοκότητα δεν σημαίνει αναγκαστικά ότι δεν υπάρχουν περιπτώσεις στις οποίες η σειριακή αναζήτηση είναι η πιο συμφέρουσα λύση. Ενδεικτικά αναφέρουμε τις περιπτώσεις όπου ο αριθμός των αναζητήσεων είναι μικρός για να απορροφήσει το σχετικά υψηλό κόστος που συνεπάγεται η ταξινόμηση των κλειδιών, ή ακόμη κι όταν τα κλειδιά δεν μπορούν να διαταχθούν γραμμικώς.

Δυαδική αναζήτηση (binary search)

Η μέθοδος αυτή, η οποία μπορεί να εφαρμοστεί μόνο σε ταξινομημένα κλειδιά, διχοτομεί σε κάθε επανάληψη το μέρος του πίνακα που δεν έχει διερευνηθεί και περιορίζει την αναζήτηση στον μισό πίνακα. Έτσι μετά από *lgn* στην χειρότερη περίπτωση ή θα έχουμε εντοπίσει το ζητούμενο κλειδί ή έχουμε καταλήξει ότι το κλειδί αυτό δεν υπάρχει. Επειδή, ο μέγιστος αριθμός συγκρίσεων που απαιτούνται για την αναζήτηση ενός κλειδιού είναι ο λογάριθμος με βάση το 2 του συνολικού αριθμού *n* των κλειδιών, η διαδικασία αυτή ονομάζεται και λογαριθμική αναζήτηση (logarithmic search).

Η συνάρτηση *binsearch* που ακολουθεί αποτελεί την υλοποίηση του παραπάνω αλγόριθμου σε μη αναδρομική μορφή.

```
int binsearch(int *a, int n, int key)
{ int bot,top,mid;
  bot=0; top=n-1;
  while (bot<=top)
  { mid =(bot + top)/2;
    if (key<a[mid])
      top=mid-1;
    else if (key>a[mid])
      bot=mid+1;
    else
      return mid;
  }
  return -1;
}
```

Άσκηση: Να γίνει ένα πρόγραμμα το οποίο να γεμίζει ένα πίνακα 15 ακεραίων με τυχαίες τιμές από το μηδέν μέχρι το εκατό να τον εμφανίζει να τον ταξινομεί σε αύξουσα διάταξη να τον εμφανίζει εκ νέου και να αναζητά σε ποια θέση υπάρχει το στειχείο 33 αν υπάρχει.

```
/* Ολοκληρωμένο παράδειγμα με τις BUBBLE SORT and BINARY SEARCH*/

#include <stdio.h>
#include <stdlib.h>
#define FALSE  0
#define TRUE   1
#define N      15

void bubble_sort(int a[], int max);
void swap(int *, int *);
int binsearch(int *a, int key, int max);
void emf(int *a, int max);
```

```
main()
{ int pin[N],i;
  randomize();
  for(i=0;i<N;i++)
    pin[i]=random(101);
  emf(pin,N);
  bubble_sort(pin,N);
  emf(pin,N);

  printf("Υπάρχει στη θέση %d \n",binsearch(pin,33,n));

  return 0;
}


void bubble_sort(int a[],int k)
{ int i,top,top1,tax_ok;
  top =k-1;
  do {
    tax_ok=TRUE;
    for(i=0;i<top;i++)
      if (a[i]>a[i+1])
        { swap(&a[i],&a[i+1]);
          tax_ok=FALSE;
          top1=i;
        }
    top=top1;
  } while (tax_ok==FALSE);
}

void swap(int *a, int *b)
{ int h;
  h=*a;
  *a=*b;
  *b=h;
}


int binsearch(int *a, int key, int max)
{ int bot,top,mid;
  bot=0; top=max-1;
  while (bot<=top)
  { mid =(bot + top)/2;
    if (key<a[mid])
      top=mid-1;
    else if (key>a[mid])
      bot=mid+1;
    else
      return mid;
  }
  return -1;
}

void emf(int *a, int max);
{ int i;
  printf("\n Ο πίνακας είναι : \n");
  for(i=0;i<max;i++)
    printf("%4d",a[i]);
  printf("\n");
  return;
}

/*-----Αυτό που εμφανίζεται στην οθόνη ----------------------
npet@dilos:~/test_c >a.out
 Ο πίνακας είναι :
  49   2  19  50  51  17  18   3   3  18  19  35  33  48  35

Ο πίνακας είναι :
   2   3   3  17  18  18  19  19  33  35  35  48  49  50  51

Υπάρχει στη θέση 8
npet@dilos:~/test_c >
-------------------------------------------------------------*/
```

Προσπέλαση Αρχείων

Μέχρι τώρα όλα τα παραδείγματα διάβαζαν από την πρότυπη (στάνταρτ) είσοδο και έγραφαν στην πρότυπη έξοδο, που για ένα πρόγραμμα ορίζονται αυτόματα από το τοπικό λειτουργικό σύστημα.

Το επόμενο βήμα είναι να γράψουμε ένα πρόγραμμα που να προσπελάζει ένα αρχείο το οποίο δεν είναι ήδη συνδεδεμένο με το πρόγραμμα.

Π.χ. cat x.c y.c //συνενώνει ένα σύνολο κατονομαζόμενων αρχείων στην οθόνη

Το πρόβλημα είναι πως να οργανώσουμε το διάβασμα των αναφερόμενων αρχείων - δηλαδή, πως να συνδέσουμε τα εξωτερικά ονόματα που έχει στο νου του ο χρήστης, με τις εντολές που διαβάζουν τα δεδομένα.

ΚΑΝΟΝΕΣ

Για να μπορεί να γραφτεί ή να διαβαστεί ένα αρχείο, πρέπει να ανοιχτεί με την συνάρτηση βιβλιοθήκης fopen. Η fopen παίρνει ένα εξωτερικό όνομα (όπως το x.c ή y.c) κάνει μερικές τακτοποιήσεις και διαπραγματεύσεις με το λειτουργικό σύστημα (που οι λεπτομέρειές τους δεν μας αφορούν), και επιστρέφει ένα δείκτη για να χρησιμοποιηθεί στις επόμενες αναγνώσεις ή καταγραφές του αρχείου.

Ο δείκτης αυτός, που λέγεται δείκτης αρχείου, δείχνει μια δομή που περιέχει πληροφορίες για το αρχείο, όπως η θέση μιας περιοχής ενδιάμεσης αποθήκευσης (buffer), η τρέχουσα θέση χαρακτήρα στην περιοχή ενδιάμεσης αποθήκευσης, το αν γράφεται ή διαβάζεται το αρχείο, κι ’αν έχουν συμβεί λάθη ή έχει συναντηθεί το τέλος του αρχείου.

Οι χρήστες δεν χρειάζεται να γνωρίζουν τις λεπτομέρειες, γιατί στους ορισμούς που παίρνουμε από το <stdio.h> περιλαμβάνεται και μια δήλωση δομής με όνομα FILE. Η μόνη δήλωση που χρειάζεται να κάνουμε εμείς για δείκτη αρχείου φαίνεται εδώ:

```
FILE *fp;
```

που λέει ότι η fp είναι δείκτης αρχείου.

Το πρωτότυπο της fopen είναι:

```
FILE *fopen(char *name, char *mode);
```

Η κλήση της fopen σε ένα πρόγραμμα είναι:

```
fp = fopen (name , mode)

"x.dat"  "r"   // ανάγνωση
         "w"   // γράψιμο
         "a"   // προσθήκη
         "b"   // δυαδικό (όχι κειμένου)
```

Το επόμενο που χρειάζεται είναι ένας τρόπος για την ανάγνωση ή την καταγραφή στο αρχείο όταν αυτό είναι ανοιχτό.

Υπάρχουν διάφορες δυνατότητες, από τις απλούστερες είναι οι

```
getc    // int getc(FILE *fp);
putc    // int putc(int c, FILE *fp);

                //EOF για τέλος αρχείου ή λάθος και για τις δυό.
```

Για φορμαρισμένη είσοδο και έξοδο αρχείων, μπορούν να χρησιμοποιηθούν οι συναρτήσεις:

fscanf // int fscanf(FILE *fp, char *format ……..);
fprintf // int fprintf(FILE *fp, char *format ……..);

Η συνάρτηση :

int fclose(FILE *fp);

είναι η αντίστροφη της fopen.

Διακόπτει τη σύνδεση ανάμεσα στο δείκτη αρχείου και το εξωτερικό όνομα που έχει καθοριστεί με την fopen, ελευθερώνοντας το δείκτη αρχείου για άλλο αρχείο.

Τα λειτουργικά έχουν όριο στον αριθμό των ταυτόχρονα ανοικτών αρχείων.

Σε αρχεία εξόδου αδειάζει την περιοχή ενδιάμεσης αποθήκευσης στην οποία η putc συλλέγει την έξοδο.

Κατά τον κανονικό τερματισμό του προγράμματος, η fclose καλείται, αυτόματα για όλα τα ανοικτά αρχεία.

Άσκηση: Να γίνει ένα πρόγραμμα που να αντιγράφει ένα αρχείο με κώδικα γραμμένο σε γλώσσα C, σε ένα άλλο αφαιρώντας όλα τα σχόλια (/* … */). Θεωρήστε ότι τα σχόλια δεν είναι επάλληλα, δηλαδή φωλιασμένα το ένα μέσα στο άλλο. Συγκρίνετέ το με το πρόγραμμα που κάνει την ίδια δουλειά αντιγράφοντας, όμως, από την στάνταρτ είσοδο (stdin) στην στάνταρτ έξοδο (stdout)

```
/* program nocomment    stdin → stdout */

#include <stdio.h>
#define NO     0
#define YES    1
#define NC     -3

main()
{ int car, cp=NC, comment=NO;


  printf("\n Δώστε κείμενο (^z για τέλος) :\n");









  while ((car=getchar())!=EOF)
  { if (comment==NO)
      { if ((cp=='/')&&(car=='*'))
          { comment=YES;
            car=NC;
          }
        else
          if (cp!=NC)
            putchar(cp);
      }
    else
      if ((cp=='*')&&(car=='/'))
        { comment=NO;
          car=NC;
        }
    cp = car;
  }
  if ((comment==NO)&&(cp!=NC))
    putchar(cp);


  return 0;
}
```

```
/* program nocomment  test.c → testnc.c */

#include <stdio.h>
#define NO     0
#define YES    1
#define NC     -3

main()
{ int car, cp=NC, comment=NO;
  FILE *fps, *fpd;
  printf("\n ***Copy Files***\n");
  if ((fps=fopen("test.c","r"))==NULL)
    { printf("\n File not found\n\n");
      exit(1);
    }
  if ((fpd=fopen("testnc.c","w"))==NULL)
    { printf("\n File not accessible\n\n");
      exit (1);
    }
  while ((car=getc(fps))!=EOF)
  { if (comment==NO)
      { if ((cp=='/')&&(car=='*'))
          { comment=YES;
            car=NC;
          }
        else
          if (cp!=NC)
            putc(cp,fpd);
      }
    else
      if ((cp=='*')&&(car=='/'))
        { comment=NO;
          car=NC;
        }
    cp = car;
  }
  if ((comment==NO)&&(cp!=NC))
    putc(cp,fpd);
  fclose(fps);
  fclose(fpd);
  return 0;
}
```