Αθανάσιος Ε. Κουτσονικόλας

Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C

Ιούνιος 2021

Αθανάσιος Ε. Κουτσονικόλας

Η Γλώσσα Προγραμματισμού C

Ιούνιος 2021

ISBN: 978-618-00-3016-7

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ 1
2. ΤΑ ΓΕΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ C 3
   - 2.1 ΒΗΜΑΤΑ ΓΡΑΦΗΣ ΚΑΙ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ 3
   - 2.2 ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΤΗΣ C 6
   - 2.3 ΥΛΟΠΟΙΗΣΗ ΑΠΛΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ 7
3. ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ 11
   - 3.1 ΤΟ ΑΛΦΑΒΗΤΟ 11
   - 3.2 ΟΙ ΛΕΞΕΙΣ 12
   - 3.3 ΤΑ ΔΕΔΟΜΕΝΑ 13
     - 3.3.1 ΜΕΤΑΒΛΗΤΕΣ 14
     - 3.3.2 ΣΤΑΘΕΡΕΣ 14
   - 3.4 ΣΧΟΛΙΑ 17
4. ΤΥΠΟΙ - ΜΕΤΑΒΛΗΤΕΣ - ΤΕΛΕΣΤΕΣ 19
   - 4.1 ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ 19
     - 4.1.1 ΒΑΣΙΚΟΙ ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ 19
     - 4.1.2 ΤΡΟΠΟΠΟΙΗΤΕΣ ΤΥΠΩΝ 20
   - 4.2 ΔΗΛΩΣΕΙΣ ΜΕΤΑΒΛΗΤΩΝ 21
   - 4.3 ΤΕΛΕΣΤΕΣ 22
     - 4.3.1 ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ 22
     - 4.3.2 ΣΥΣΧΕΤΙΣΤΙΚΟΙ ΚΑΙ ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ 24
     - 4.3.3 ΤΕΛΕΣΤΕΣ BIT (BITWISE) 25
     - 4.3.4 ΤΕΛΕΣΤΕΣ ΑΠΟΔΟΣΗΣ ΤΙΜΗΣ (ΚΑΤΑΧΩΡΗΣΗΣ) 26
   - 4.4 ΠΑΡΑΣΤΑΣΕΙΣ 27
   - 4.5 ΜΕΤΑΤΡΟΠΕΣ ΤΥΠΩΝ 27
     - 4.5.1 ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΤΡΟΠΕΣ 27
     - 4.5.2 ΡΗΤΕΣ ΜΕΤΑΤΡΟΠΕΣ (CAST) 28
   - 4.6 ΠΡΟΤΕΡΑΙΟΤΗΤΑ ΤΕΛΕΣΤΩΝ 29
   - 4.7 ΠΑΡΑΔΕΙΓΜΑΤΑ 30
   - 4.8 ΑΣΚΗΣΕΙΣ 32
5. ΕΙΣΟΔΟΣ - ΕΞΟΔΟΣ 35
   - 5.1 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΧΑΡΑΚΤΗΡΩΝ 35
   - 5.2 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ 36
   - 5.3 ΜΟΡΦΟΠΟΙΗΜΕΝΗ ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ 37
     - 5.3.1 Η ΣΥΝΑΡΤΗΣΗ `printf()` 37
     - 5.3.2 Η ΣΥΝΑΡΤΗΣΗ `scanf()` 39
   - 5.4 ΑΣΚΗΣΕΙΣ 41
6. ΕΝΤΟΛΕΣ ΕΠΙΛΟΓΗΣ ΚΑΙ ΕΠΑΝΑΛΗΨΗΣ 43
   - 6.1 Η ΣΥΝΘΕΤΗ ΕΝΤΟΛΗ 44
   - 6.2 Η ΕΝΤΟΛΗ ΕΠΙΛΟΓΗΣ `if-else` 44
     - 6.2.1 ΕΝΘΕΤΕΣ ΕΝΤΟΛΕΣ `if` 46
     - 6.2.2 Ο ΤΕΛΕΣΤΗΣ ΣΥΝΘΗΚΗΣ `?:` 47
   - 6.3 Η ΕΝΤΟΛΗ ΕΠΙΛΟΓΗΣ `switch` 47
   - 6.4 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `while` 50
   - 6.5 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `do-while` 52
   - 6.6 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `for` 53
     - 6.6.1 Ο ΤΕΛΕΣΤΗΣ `comma` 55
   - 6.7 ΠΑΡΑΔΕΙΓΜΑΤΑ 56
   - 6.8 ΑΣΚΗΣΕΙΣ 60
7. ΣΥΝΑΡΤΗΣΕΙΣ 63
   - 7.1 ΟΡΙΣΜΟΣ ΣΥΝΑΡΤΗΣΗΣ 64
     - 7.1.1 ΠΡΟΤΥΠΟΠΟΙΗΣΗ ΣΥΝΑΡΤΗΣΗΣ 64
     - 7.1.2 ΑΠΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ 65
   - 7.2 Η ΕΝΤΟΛΗ `return` 66
   - 7.3 ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΗΣ "ΚΑΤ'ΑΞΙΑ" (by value) 67
     - 7.3.1 ΣΥΝΑΡΤΗΣΗ ΩΣ ΠΑΡΑΜΕΤΡΟΣ ΣΥΝΑΡΤΗΣΗΣ 69
   - 7.4 ΤΟΠΙΚΕΣ ΚΑΙ ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ 70
   - 7.5 ΚΑΘΟΡΙΣΤΕΣ ΚΑΤΗΓΟΡΙΑΣ ΑΠΟΘΗΚΕΥΣΗΣ 73
   - 7.6 ΠΡΟΣΔΙΟΡΙΣΤΕΣ ΤΥΠΩΝ 76

7.7 ΔΗΛΩΣΕΙΣ ΜΕΤΑΒΛΗΤΩΝ ... 77
7.8 ΑΝΑΔΡΟΜΗ ... 77
7.9 ΜΑΘΗΜΑΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΒΙΒΛΙΟΘΗΚΗΣ ... 81
7.10 ΠΑΡΑΔΕΙΓΜΑΤΑ ... 81
7.11 ΑΣΚΗΣΕΙΣ ... 88
8. ΠΙΝΑΚΕΣ ΚΑΙ ΑΛΦΑΡΙΘΜΗΤΙΚΑ ... 89
8.1 ΠΙΝΑΚΕΣ ... 89
8.1.1 ΔΗΛΩΣΗ ΠΙΝΑΚΑ ... 89
8.1.2 ΑΝΑΦΟΡΑ ΣΤΑ ΣΤΟΙΧΕΙΑ ΕΝΟΣ ΠΙΝΑΚΑ ... 89
8.1.3 ΠΙΝΑΚΕΣ ΠΟΛΛΩΝ ΔΙΑΣΤΑΣΕΩΝ ... 90
8.1.4 ΑΠΟΔΟΣΗ ΑΡΧΙΚΩΝ ΤΙΜΩΝ ΣΕ ΠΙΝΑΚΕΣ ... 92
8.1.5 ΠΙΝΑΚΕΣ ΩΣ ΠΑΡΑΜΕΤΡΟΙ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ ... 93
8.1.6 ΤΑΞΙΝΟΜΗΣΗ ΠΙΝΑΚΑ ... 96
8.2 ΑΛΦΑΡΙΘΜΗΤΙΚΑ ... 97
8.2.1 ΣΥΝΑΡΤΗΣΕΙΣ ΒΙΒΛΙΟΘΗΚΗΣ ΓΙΑ ΑΛΦΑΡΙΘΜΗΤΙΚΑ ... 97
8.3 ΠΑΡΑΔΕΙΓΜΑΤΑ ... 100
8.4 ΑΣΚΗΣΕΙΣ ... 106
9. ΑΠΑΡΙΘΜΗΣΕΙΣ, ΔΟΜΕΣ ΚΑΙ ΕΝΩΣΕΙΣ ... 109
9.1 ΑΠΑΡΙΘΜΗΣΕΙΣ ... 109
9.2 ΔΟΜΕΣ ... 111
9.2.1 ΔΗΛΩΣΗ ΔΟΜΗΣ ... 112
9.2.2 ΑΝΑΦΟΡΑ ΣΤΑ ΜΕΛΗ ΜΙΑΣ ΔΟΜΗΣ ... 113
9.2.3 ΕΝΘΕΤΕΣ ΔΟΜΕΣ ... 114
9.2.4 ΑΠΟΔΟΣΗ ΑΡΧΙΚΩΝ ΤΙΜΩΝ ΣΕ ΔΟΜΕΣ ... 115
9.2.5 ΔΟΜΕΣ ΩΣ ΠΑΡΑΜΕΤΡΟΙ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ ... 115
9.2.6 ΠΙΝΑΚΕΣ ΔΟΜΩΝ ... 117
9.3 ΕΝΩΣΕΙΣ ... 118
9.4 Η ΔΗΛΩΣΗ `typedef` ... 119
9.5 ΠΑΡΑΔΕΙΓΜΑΤΑ ... 120
9.6 ΑΣΚΗΣΕΙΣ ... 125
10. ΔΕΙΚΤΕΣ ... 127
10.1 ΔΗΛΩΣΗ ΜΕΤΑΒΛΗΤΗΣ ΔΕΙΚΤΗ ... 127
10.2 ΤΕΛΕΣΤΕΣ ΔΕΙΚΤΩΝ ... 128
10.3 ΠΑΡΑΣΤΑΣΕΙΣ - ΔΕΙΚΤΕΣ ... 129
10.3.1 ΑΠΟΔΟΣΗ ΤΙΜΩΝ ΣΕ ΔΕΙΚΤΕΣ ... 129
10.3.2 ΑΡΙΘΜΗΤΙΚΗ ΔΕΙΚΤΩΝ ... 130
10.3.3 ΣΥΓΚΡΙΣΗ ΔΕΙΚΤΩΝ ... 131
10.4 ΔΕΙΚΤΕΣ ΚΑΙ ΣΥΝΑΡΤΗΣΕΙΣ ... 132
10.5 ΔΕΙΚΤΕΣ ΚΑΙ ΠΙΝΑΚΕΣ ... 133
10.6 ΔΕΙΚΤΕΣ ΚΑΙ ΑΛΦΑΡΙΘΜΗΤΙΚΑ ... 134
10.7 ΠΙΝΑΚΕΣ ΔΕΙΚΤΩΝ ΚΑΙ ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ ... 136
10.8 ΟΡΙΣΜΑΤΑ ΓΡΑΜΜΗΣ ΔΙΑΤΑΓΩΝ ... 140
10.9 ΔΕΙΚΤΕΣ ΚΑΙ ΔΟΜΕΣ ... 141
10.10 ΠΑΡΑΔΕΙΓΜΑΤΑ ... 142
10.11 ΑΣΚΗΣΕΙΣ ... 145
11. ΑΡΧΕΙΑ ... 147
11.1 ΑΝΟΙΓΜΑ ΚΑΙ ΚΛΕΙΣΙΜΟ ΑΡΧΕΙΟΥ ... 147
11.2 ΠΡΟΣΠΕΛΑΣΗ ΑΡΧΕΙΟΥ ... 149
11.2.1 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΧΑΡΑΚΤΗΡΩΝ ... 149
11.2.2 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ ... 151
11.2.3 ΜΟΡΦΟΠΟΙΗΜΕΝΗ ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ... 152
11.2.4 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ ... 154
11.3 ΑΛΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΧΕΙΡΙΣΜΟΥ ΑΡΧΕΙΩΝ ... 156
11.4 ΠΑΡΑΔΕΙΓΜΑΤΑ ... 157
11.5 ΑΣΚΗΣΕΙΣ ... 165
12. Ο ΠΡΟΕΠΕΞΕΡΓΑΣΤΗΣ ... 167
12.1 `#define` και `#undef` ... 167
12.2 `#include` ... 169
12.3 `#if, #else, #endif, #elif, #ifdef` και `#ifndef` ... 170
12.4 ΑΣΚΗΣΕΙΣ ... 173
ΠΑΡΑΡΤΗΜΑ Α': ΤΟ ΣΥΝΟΛΟ ΧΑΡΑΚΤΗΡΩΝ ASCII ... 175
ΒΙΒΛΙΟΓΡΑΦΙΑ ... 179

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

Η C είναι μία γλώσσα προγραμματισμού, που σχεδιάστηκε και υλοποιήθηκε για πρώτη φορά από τον Dennis Ritchie, σ'έναν υπολογιστή DEC PDP-11, στα εργαστήρια της Bell στο New Jersey των Η.Π.Α. Η C σχεδιάστηκε αρχικά για το λειτουργικό σύστημα UNIX και εφαρμόστηκε σ'αυτό. Ο πυρήνας του UNIX, ο μεταγλωττιστής της C και σχεδόν όλα τα βοηθητικά προγράμματα του UNIX, είναι γραμμένα σε C.

Η γλώσσα C είναι το αποτέλεσμα μιας εργασίας που ξεκίνησε από μια παλαιότερη γλώσσα που ονομαζόταν BCL, η οποία σχεδιάστηκε από τον Martin Richards στη δεκαετία του 60. Η BCL επηρέασε μια άλλη γλώσσα που ονομάστηκε B και σχεδιάστηκε από τον Ken Thompson (συνεργάτη του Ritchie) το 1970 για το πρώτο σύστημα UNIX, σε έναν υπολογιστή DEC PDP-7. Η γλώσσα B στη συνέχεια, οδήγησε στην ανάπτυξη της C, γύρω στο 1972.

Από τα μέσα της δεκαετίας του 1970, το UNIX χρησιμοποιείται εκτεταμένα στα εργαστήρια της Bell και αργότερα επεκτείνεται η χρήση του σε διάφορα Πανεπιστήμια. Έτσι, η C άρχισε σιγά-σιγά να αντικαθιστά τις άλλες διαθέσιμες γλώσσες στο UNIX. Για πολλά χρόνια, το de facto πρότυπο της γλώσσας C ήταν η παρεχόμενη έκδοσή της με την έκδοση 5 του UNIX και περιγραφόταν στην πρώτη έκδοση του εγχειριδίου αναφοράς "The C Programming Language" των Brian Kernighan και Dennis Ritchie (συχνά αναφέρεται ως "πρότυπο K&R").

Με τη ραγδαία ανάπτυξη των μικροϋπολογιστών, δημιουργήθηκε ένας μεγάλος αριθμός υλοποιήσεων της C, οι οποίες παρουσίαζαν αποκλίσεις μεταξύ τους. Έτσι το 1983, το Αμερικανικό Εθνικό Ινστιτούτο Προτύπων (American National Standards Institute - ANSI) σύστησε μια επιτροπή για να δώσει έναν πλήρη και σύγχρονο ορισμό της C. Ο ορισμός που προέκυψε, το πρότυπο ANSI ή "ANSI C", εγκρίθηκε το 1988. Το πρότυπο αυτό βασίζεται στο αρχικό εγχειρίδιο αναφοράς.

Έτσι οι διάφορες υλοποιήσεις της C πρέπει να καλύπτουν το πρότυπο ANSI. Πράγματι διάφορες εταιρίες κυκλοφόρησαν τους δικούς τους μεταγλωττιστές για την C, οι οποίοι, προφανώς, ικανοποιούν το πρότυπο ANSI. Οι περισσότεροι όμως μεταγλωττιστές, υπερκαλύπτουν το πρότυπο αυτό. Για το λόγο αυτό, αν θέλουμε τα προγράμματα που γράφουμε να μπορούν να μεταφερθούν σε διάφορους ηλεκτρονικούς υπολογιστές (Η/Υ) με διαφορετικούς μεταγλωττιστές, τότε θα πρέπει να χρησιμοποιούμε *μόνο* τις δυνατότητες που ορίζονται στο πρότυπο ANSI. Έτσι θα είναι εγγυημένο ότι τα προγράμματά μας μπορούν να περάσουν από οποιονδήποτε μεταγλωττιστή που ικανοποιεί το πρότυπο ANSI. Στις σημειώσεις αυτές, όλα τα προγράμματα συμφωνούν με το πρότυπο ANSI.

Η C είναι γλώσσα γενικού σκοπού και σχετικά *μέσου επιπέδου*. Ο χαρακτηρισμός αυτός δεν είναι υποτιμητικός, απλώς σημαίνει ότι η C παρέχει στον προγραμματιστή δυνατότητες για προγραμματισμό σε *χαμηλό* (low level) αλλά και σε *υψηλό* (high level) επίπεδο. Ο προγραμματισμός σε χαμηλό επίπεδο,

επιτρέπει στον προγραμματιστή άμεση πρόσβαση στο *υλικό* (hardware) του Η/Υ και στις στοιχειώδεις λειτουργίες του. Έτσι η διαχείριση bits, bytes και διευθύνσεων μνήμης, γίνεται με μεγάλη ευκολία. Αυτές οι δυνατότητες κάνουν τη C κατάλληλη για προγραμματισμό σε επίπεδο συστήματος. Από την άλλη, ο προγραμματισμός σε υψηλό επίπεδο, "αποκρύπτει" από τον προγραμματιστή τις λεπτομέρειες του υλικού, κάνοντας έτσι τον κώδικα που παράγεται από τη C σε μεγάλο βαθμό *φορητό*. Αυτό σημαίνει ότι ένα πρόγραμμα που έχει γραφτεί για κάποιον τύπο Η/Υ, μπορεί να χρησιμοποιηθεί και σε Η/Υ άλλου τύπου.

Η γλώσσα C είναι επίσης γνωστή για την ελευθερία που προσφέρει στον χρήστη. Προφανώς, κατάχρηση αυτής της ελευθερίας δημιουργεί προβλήματα στα προγράμματα. Ως συνέπεια αυτής της ελευθερίας, οι ανακριβείς χρήστες της C σύντομα θα ανακαλύψουν ότι είναι πολύ εύκολο να γράψουν προγράμματα που εκτελούν ενέργειες διαφορετικές από αυτές που θα ήθελαν. Σε αντίθεση με τις άλλες γλώσσες, και αυτό είναι που εκπλήσσει τους άπειρους χρήστες, συχνά τέτοια προγράμματα είναι συντακτικά σωστά. Έτσι ο μεταγλωττιστής δε θα ανιχνεύσει λάθος, όμως τα αποτελέσματα που θα πάρουμε δε θα είναι τα αναμενόμενα. Για παράδειγμα, αν από λάθος γράψουμε ένα σημείο ισότητας (=, τελεστής καταχώρησης) εκεί που κανονικά έπρεπε να γράψουμε δύο (==, τελεστής σύγκρισης), μπορεί να είναι πολύ δύσκολο να βρούμε γιατί παίρνουμε λάθος αποτελέσματα. Αυτό δεν είναι σοβαρό μειονέκτημα της γλώσσας. Στον προγραμματισμό πρέπει να είμαστε προσεκτικοί και είναι λογικό, λάθος εντολές να οδηγούν σε λάθος αποτελέσματα.

Μια από τις πιο δημοφιλείς υλοποιήσεις της C στο χώρο των μικροϋπολογιστών, είναι η Turbo C (TC). Η TC είναι προϊόν (και σήμα κατατεθέν) της εταιρείας Borland International Inc. Η TC δεν είναι απλώς ένας μεταγλωττιστής της C, είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης προγραμμάτων σε C.

ΚΕΦΑΛΑΙΟ 2
ΤΑ ΓΕΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ C

2.1 ΒΗΜΑΤΑ ΓΡΑΦΗΣ ΚΑΙ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

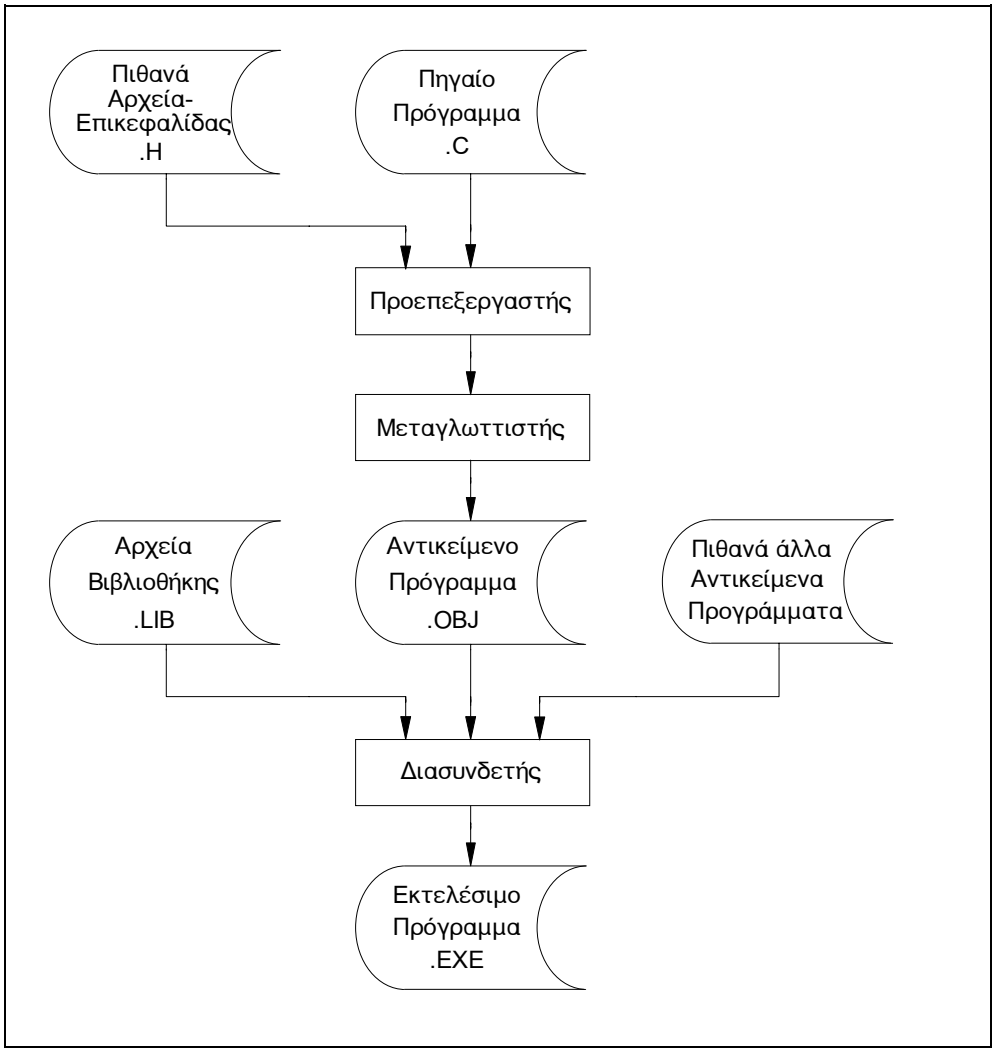
Τα προγράμματα που γράφονται σε C είναι μια *σειρά από εντολές* τα ονόματα των οποίων είναι, συνήθως, έτσι διαλεγμένα ώστε να φανερώνουν την ενέργεια της κάθε μιας. Για να γράψουμε τα προγράμματά μας, χρησιμοποιούμε κάποιο *συντάκτη κειμένου* (editor), έτσι παράγουμε ένα πρόγραμμα σε μορφή κειμένου αναγνωρίσιμη και κατανοητή από τον άνθρωπο. Σ'αυτή τη μορφή του, το πρόγραμμα λέγεται *πηγαίο πρόγραμμα* (source program). Από την άλλη, η κεντρική μονάδα επεξεργασίας του Η/Υ μπορεί να εκτελέσει εντολές σε δυαδική μορφή *(γλώσσα μηχανής)*. Αυτό σημαίνει ότι ένα πρόγραμμα γραμμένο σε C, χρειάζεται να μεταφραστεί από τη μορφή κειμένου που το συντάσσει ο προγραμματιστής, σε γλώσσα μηχανής. Την εργασία αυτή κάνει κατάλληλο μεταφραστικό πρόγραμμα που λέγεται *μεταγλωττιστής της C* (C compiler).

Ο μεταγλωττιστής λοιπόν είναι ένα μεταφραστικό πρόγραμμα η είσοδος του οποίου είναι το αρχείο πηγαίου προγράμματος της C, το όνομα του οποίου πρέπει να έχει προέκταση ".C", καθώς και πιθανά *αρχεία-επικεφαλίδας* (header files). Τα αρχεία-επικεφαλίδας (λέγονται επίσης και αρχεία συμπερίληψης: include files) είναι αρχεία κειμένου. Μπορούν να συνδυαστούν με το πηγαίο αρχείο πριν αρχίσει η μεταγλώττιση αυτού. Τη διαδικασία αυτή την επιτελεί ένα πρόγραμμα που λέγεται *προεπεξεργαστής της C* (C preprocessor), βλέπε Κεφάλαιο 12.

Η έξοδος του μεταγλωττιστή είναι ένα αρχείο που έχει το ίδιο όνομα με το αρχείο του πηγαίου προγράμματος, προέκταση ".OBJ" και λέγεται *αντικείμενο πρόγραμμα* (object program). Αυτό πρέπει να διασυνδεθεί (linking) με διάφορα αρχεία βιβλιοθήκης της γλώσσας και πιθανώς με άλλα αντικείμενα προγράμματα. Τη διαδικασία της διασύνδεσης την αναλαμβάνει κατάλληλο πρόγραμμα που λέγεται *διασυνδετής* (linker). Η έξοδος του διασυνδετή είναι ένα αρχείο με όνομα ίδιο με αυτό του πηγαίου προγράμματος και προέκταση ".EXE" που σημαίνει ότι είναι ένα *εκτελέσιμο πρόγραμμα* (executable program).

Τα αρχεία βιβλιοθήκης είναι ομάδες από προ-μεταγλωττισμένες ρουτίνες για την εκτέλεση συγκεκριμένων εργασιών. Για παράδειγμα, αν χρησιμοποιήσουμε στο πρόγραμμά μας μια συνάρτηση σαν την `printf()` (την οποία θα περιγράψουμε αργότερα) για να τυπώσουμε κάτι στην οθόνη, ο κώδικας γι'αυτή τη συνάρτηση περιλαμβάνεται σ'ένα αρχείο βιβλιοθήκης. Κάθε αρχείο βιβλιοθήκης έχει ένα βασικό χαρακτηριστικό: μόνο τα κομμάτια του αυτά που είναι απαραίτητα θα συνδεθούν με την εκτελέσιμη έκδοση του προγράμματος, όχι ολόκληρο το αρχείο.

Το Σχήμα 2.1 δείχνει τα βήματα που πρέπει να γίνουν και τις φάσεις από τις οποίες περνάει ένα αρχείο πηγαίου προγράμματος έως ότου παραχθεί το αντίστοιχο αρχείο εκτελέσιμου προγράμματος.

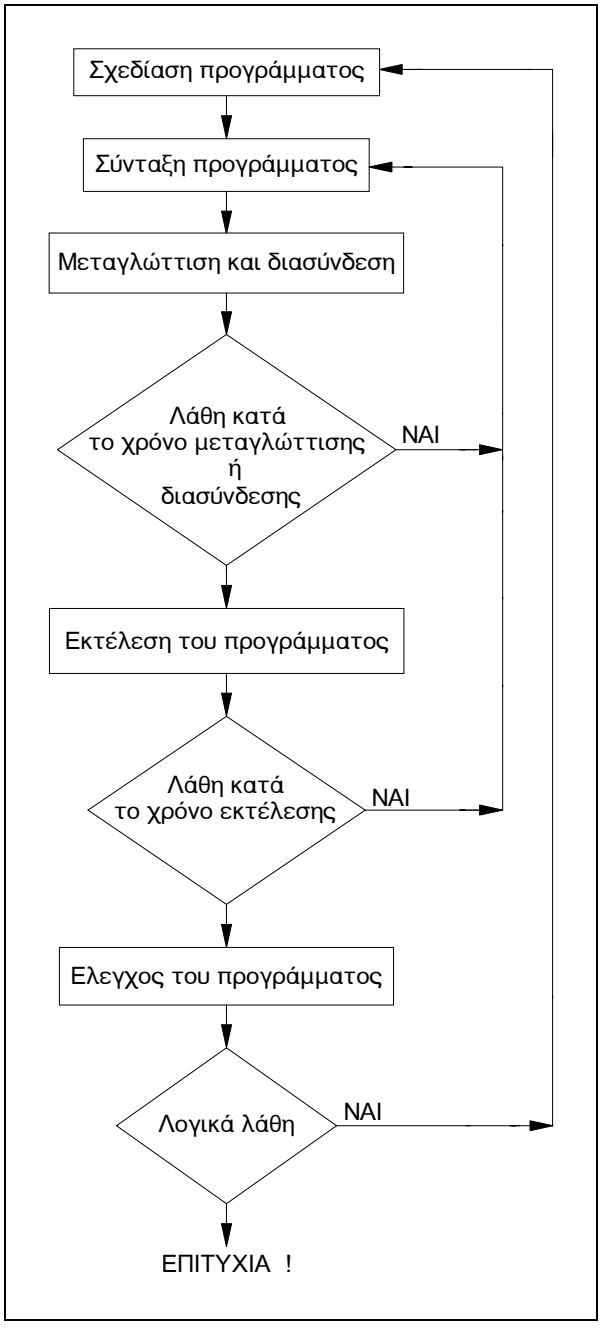


Σχήμα 2.1: *Γραφή - Μεταγλώττιση - Διασύνδεση προγράμματος*

Κατά τη διάρκεια της μεταγλώττισης ο μεταγλωττιστής ελέγχει (λεκτικά, συντακτικά και σημασιολογικά) το πηγαίο πρόγραμμα για πιθανά λάθη. Αυτά τα λάθη λέγονται *λάθη κατά το χρόνο της μεταγλώττισης* (compile-time errors) και είναι αποτέλεσμα παραβιάσεων του συντακτικού της γλώσσας. Για κάθε λάθος που ανιχνεύεται, ο μεταγλωττιστής παράγει κατάλληλο *διαγνωστικό μήνυμα λάθους*. Επίσης αντίστοιχα μηνύματα λάθους παράγονται από το διασυνδετή στην περίπτωση που συμβεί λάθος κατά τη διασύνδεση του προγράμματος με κάποιο από τα αρχεία βιβλιοθήκης.

Το επόμενο βήμα μετά την επιτυχημένη μεταγλώττιση και διασύνδεση του προγράμματος, είναι η *εκτέλεση* ή *τρέξιμο* του εκτελέσιμου προγράμματος. Κατά τη διάρκεια εκτέλεσης του προγράμματος είναι πιθανό το *λειτουργικό σύστημα* του Η/Υ να ανιχνεύσει λάθη. Αυτά λέγονται *λάθη κατά το χρόνο εκτέλεσης* (run-

time errors). Πιθανά τέτοια λάθη είναι η διαίρεση ενός αριθμού με το μηδέν, η απόπειρα εύρεσης της τετραγωνικής ρίζας ενός αρνητικού αριθμού κ.λ.π.



Σχήμα 2.2: *Διάγραμμα ροής των διαδικασιών ανάπτυξης ενός προγράμματος*

Το πρόγραμμα όμως, μπορεί να μην παρουσιάζει λάθη κατά το χρόνο μεταγλώττισης ούτε κατά το χρόνο εκτέλεσης, αλλά παρ'όλα αυτά να μη δίνει τα αναμενόμενα αποτελέσματα, η να μη συμπεριφέρεται όπως αναμενόταν εξαιτίας λάθους στον επιλεγμένο αλγόριθμο ή στην έκφρασή του στη γλώσσα προγραμματισμού. Αυτές οι αποκλίσεις από τα αναμενόμενα, λέγονται *λογικά*

λάθη (logical errors) και είναι ευθύνη του προγραμματιστή να βρει και να διορθώσει αυτά τα λάθη.

Κάθε φορά που ανιχνεύεται κάποιο λάθος (οποιοδήποτε από τα προαναφερθέντα) πρέπει το πηγαίο πρόγραμμα να διορθωθεί. Η διόρθωση γίνεται χρησιμοποιώντας το συντάκτη κειμένου που χρησιμοποιείται και για τη γραφή του προγράμματος.

Το Σχήμα 2.2 δείχνει σε μορφή διαγράμματος ροής τις διαδικασίες: του *γραψίματος* του πηγαίου προγράμματος (σχεδίαση της μεθόδου επίλυσης, σύνταξη προγράμματος), της *μεταγλώττισης* και της *εκτέλεσης* του προγράμματος. Παρατηρούμε ότι οι παραπάνω διαδικασίες είναι δυνατό να γίνουν περισσότερες της μίας φορές σ'ένα πρόγραμμα, εξαιτίας διαφόρων ειδών λαθών.

2.2 ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ΤΗΣ C

Κάθε πρόγραμμα της C αποτελείται από μία ή περισσότερες συναρτήσεις (functions). Μία *συνάρτηση* της C είναι αντίστοιχη της υπορουτίνας της BASIC ή της συνάρτησης της Pascal, είναι δηλαδή μία ομάδα εντολών που επιτελεί συγκεκριμένη λειτουργία. Θα δούμε περισσότερα για τις συναρτήσεις στο Κεφάλαιο 7.

Κάθε πρόγραμμα περιέχει απαραιτήτως μία και μόνο μία συνάρτηση που καλείται `main()`. Ασχέτως με το πόσες συναρτήσεις υπάρχουν σ'ένα πρόγραμμα, η `main()` είναι εκείνη στην οποία περνάει ο έλεγχος από το λειτουργικό σύστημα όταν εκτελείται το πρόγραμμα, μ'άλλα λόγια είναι η πρώτη συνάρτηση που εκτελείται, οπουδήποτε κι αν βρίσκεται εντός του προγράμματος. Η `main()` μπορεί να καλεί άλλη συνάρτηση η οποία μπορεί να καλεί κάποια άλλη κ.ο.κ. Η μορφή ενός προγράμματος που περιέχει μόνο μία συνάρτηση, που προφανώς θα ονομάζεται `main()`, είναι:

```
main()          ← όνομα συνάρτησης (οι παρενθέσεις θα μπορούσαν να έχουν παραμέτρους)
{               ← οριοθέτης
   εντολή1
   εντολή2      ← σώμα της συνάρτησης
     ...
   εντολήn
}               ← οριοθέτης
```

Οι παρενθέσεις μετά το όνομα της συνάρτησης είναι απαραίτητες, έστω κι αν δεν περιέχουν ορίσματα, για λόγους που θα γίνουν κατανοητοί παρακάτω. Τον καθορισμό της συνάρτησης ακολουθούν τα άγκιστρα που *οριοθετούν* την αρχή και το τέλος του σώματος της συνάρτησης. Το άγκιστρο ανοίγματος (`{`) δείχνει ότι ένα κομμάτι κώδικα που αποτελεί μια καθορισμένη ομάδα, πρόκειται να ξεκινήσει. Το άγκιστρο κλεισίματος (`}`) τερματίζει ένα κομμάτι του κώδικα. Τα άγκιστρα στη C επιτελούν μία παρόμοια λειτουργία με τις εντολές `Begin` και `End` της Pascal.

Κάθε εντολή της C τερματίζεται με ένα ελληνικό ερωτηματικό (`;`). Ας σημειωθεί εδώ, ότι το ελληνικό ερωτηματικό στην C *δε διαχωρίζει εντολές,* όπως στην Pascal, αλλά *τερματίζει εντολές*. Με άλλα λόγια *το ελληνικό ερωτηματικό αποτελεί μέρος της εντολής*.

Μία ακόμα παρατήρηση που πρέπει να γίνει εδώ είναι ότι η C *κάνει διάκριση μεταξύ κεφαλαίων και πεζών γραμμάτων.* Μία σύμβαση που θα ακολουθήσουμε είναι ότι θα γράφουμε τα πάντα με πεζούς χαρακτήρες (εκτός από τις σταθερές όπως θα δούμε αργότερα), για ευκολία στην πληκτρολόγηση.

2.3 ΥΛΟΠΟΙΗΣΗ ΑΠΛΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ

Για να μπορούμε να γράφουμε απλά προγράμματα σε C, θα δούμε σ'αυτή την παράγραφο μερικά παραδείγματα, χρησιμοποιώντας κάποιες από τις συναρτήσεις βιβλιοθήκης της γλώσσας όπως η `printf()` (για εκτύπωση τιμών στην οθόνη) και η `scanf()` (για διάβασμα δεδομένων από το πληκτρολόγιο). Θα δούμε με περισσότερες λεπτομέρειες τους βασικούς τρόπους εισόδου-εξόδου στο Κεφάλαιο 5.

Ας γράψουμε λοιπόν το πρώτο μας πρόγραμμα σε C:

```
main()
{
  printf("Good morning World !");
}
```

Εκτελώντας αυτό το πρόγραμμα θα πάρουμε στην οθόνη:

```
Good morning World !
```

Αυτό είναι το αποτέλεσμα της μοναδικής εντολής του προγράμματός μας. Η λέξη "printf" είναι όνομα συνάρτησης όπως και η λέξη "main". Μιας και η "printf" είναι συνάρτηση ακολουθείται από παρενθέσεις. Σ'αυτή την περίπτωση, οι παρενθέσεις περιέχουν την προς εκτύπωση φράση, η οποία περικλείεται από διπλά εισαγωγικά (" "). Αυτή η φράση είναι ένα *όρισμα* συνάρτησης. Περισσότερα για τα ορίσματα θα δούμε στο Κεφάλαιο 7. Η φράση "Good morning World !" είναι ένα παράδειγμα *αλφαριθμητικού*. Στην C, οι σταθερές αλφαριθμητικού, όπως αυτό, περικλείονται από διπλά εισαγωγικά.

Χρησιμοποιήσαμε τη συνάρτηση `printf()` χωρίς να υπάρχει ο κώδικας αυτής της συνάρτησης στο πρόγραμμά μας. Ο κώδικας γι'αυτή τη συνάρτηση υπάρχει στο αρχείο βιβλιοθήκης της γλώσσας. Όταν μεταγλωττίζεται το πρόγραμμα, ο μεταγλωττιστής αντιλαμβάνεται ότι η `printf()` δεν είναι μια συνάρτηση που περιλαμβάνεται στο πηγαίο πρόγραμμα και έτσι αφήνει ένα μήνυμα στο διασυνδετή γι'αυτό το γεγονός. Ο διασυνδετής ψάχνει στο αρχείο βιβλιοθήκης της γλώσσας, βρίσκει το κομμάτι αυτού του αρχείου που περιλαμβάνει την `printf()` και προκαλεί τη διασύνδεση αυτού του κομματιού με το πηγαίο πρόγραμμα. Παρόμοια διαδικασία ακολουθείται για όλες τις συναρτήσεις βιβλιοθήκης της C.

Η συνάρτηση `printf()` μπορεί να χρησιμοποιηθεί για την εκτύπωση στην οθόνη με καθορισμένη μορφή τόσο σταθερών όσο και μεταβλητών.

Για παράδειγμα ας δούμε το παρακάτω πρόγραμμα:

```
main()
{
  printf("Αυτός είναι ο αριθμός %d",2);
}
```

Εκτελώντας αυτό το πρόγραμμα θα πάρουμε στην οθόνη:

```
Αυτός είναι ο αριθμός 2
```

Ας δούμε τώρα γιατί τυπώθηκε ο αριθμός 2 και τι προκαλεί το `%d`. Στην συνάρτηση `printf()` μπορούν να δοθούν παραπάνω από ένα ορίσματα. Στο προηγούμενο παράδειγμα της δώσαμε μόνο ένα: το αλφαριθμητικό "Good morning World !". Τώρα της δώσαμε δύο: ένα αλφαριθμητικό ("Αυτός είναι ο αριθμός %d") στα αριστερά και μία τιμή (τον ακέραιο αριθμό 2) στα δεξιά. Αυτά τα δύο ορίσματα χωρίζονται με κόμμα. Η συνάρτηση `printf()` παίρνει την τιμή δεξιά από το κόμμα και την τοποθετεί στο αλφαριθμητικό αριστερά. Η θέση του αλφαριθμητικού όπου τοποθετείται η τιμή, είναι εκείνη στην οποία υπάρχει ένας *καθοριστής μορφής* (format specifier).

Ο καθοριστής μορφής υποδεικνύει στην `printf()` πού να τοποθετήσει την τιμή στο αλφαριθμητικό και τί μορφή να χρησιμοποιήσει για να εκτυπώσει την τιμή. Στο παραπάνω παράδειγμα, το `%d` υποδεικνύει στην `printf()` να τυπώσει την τιμή 2 σαν δεκαδικό ακέραιο (στο δεκαδικό σύστημα αρίθμησης). Μπορούμε να χρησιμοποιήσουμε και άλλους καθοριστές για την εκτύπωση του αριθμού 2. Για παράδειγμα το `%f` θα έκανε το 2 να τυπωθεί σαν αριθμός κινητής υποδιαστολής και το `%x` θα το τύπωνε σαν δεκαεξαδικό ακέραιο (στο δεκαεξαδικό σύστημα αρίθμησης).

Χρησιμοποιώντας καθοριστές μορφής μπορούμε να εκτυπώσουμε το ίδιο καλά αλφαριθμητικά όπως και αριθμούς. Για παράδειγμα στο παρακάτω πρόγραμμα, ένα αλφαριθμητικό και ένας αριθμός τυπώνονται μαζί:

```
main()
{
  printf("Ο %s είναι %d ετών","Νίκος",30);
}
```

Εκτελώντας αυτό το πρόγραμμα θα πάρουμε στην οθόνη:

```
Ο Νίκος είναι 30 ετών
```

Χρησιμοποιήσαμε τον καθοριστή μορφής `%s` για να τυπώσουμε την αλφαριθμητική σταθερά "Νίκος" και το `%d` για την ακέραια σταθερά 30.

Ο τελευταίος καθοριστής μορφής που θα δούμε εδώ είναι ο `%c` που χρησιμοποιείται όταν θέλουμε να εκτυπώσουμε χαρακτήρες. Περισσότερα για τους καθοριστές μορφής θα δούμε στο Κεφάλαιο 5, όπου θα εξετάσουμε με περισσότερη λεπτομέρεια τις συναρτήσεις `printf()` και `scanf()`.

Ας θεωρήσουμε το παρακάτω πρόγραμμα:

```
main()
{
  printf("Το γράμμα %c είναι",'α');
  printf(" το αρχικό της λέξης %s","άλφα");
}
```

Η έξοδος αυτού του προγράμματος είναι:

```
Το γράμμα α είναι το αρχικό της λέξης άλφα
```

Σ'αυτό το πρόγραμμα το 'α' είναι χαρακτήρας και το "άλφα" αλφαριθμητικό. Ας προσέξουμε ότι ο χαρακτήρας περικλείεται από μονά εισαγωγικά, ενώ το αλφαριθμητικό από διπλά. Παρατηρούμε επίσης ότι παρόλο που το αποτέλεσμα εκτυπώνεται από δύο ξεχωριστές εντολές, στην οθόνη παίρνουμε μία γραμμή κειμένου. Αυτό γίνεται γιατί η `printf()` δεν εκτυπώνει αυτόματα χαρακτήρα καινούργιας γραμμής.

Θα γνωρίσουμε τώρα με συντομία τη συνάρτηση `scanf()` που χρησιμοποιείται για είσοδο δεδομένων στο πρόγραμμα, από το πληκτρολόγιο. Η μορφή της, που θα περιγραφεί αναλυτικά στο Κεφάλαιο 5, είναι παρόμοια με αυτή της `printf()`. Χρησιμοποιεί τους ίδιους καθοριστές μορφής, για την ανάγνωση τιμών, με αυτούς της `printf()`.

Στο παρακάτω πρόγραμμα φαίνεται μια τυπική χρήση της `scanf()`.

```
main()
{
  int i;

  printf("Δώσε την ηλικία σου σε χρόνια: ");
  scanf("%d",&i);
  printf("Είσαι %d ετών",i);
}
```

Στην τρίτη γραμμή του παραπάνω προγράμματος, γίνεται ορισμός μιας μεταβλητής ακεραίου τύπου, με όνομα `i`, που θα χρησιμοποιηθεί στο πρόγραμμα. Θα εξετάσουμε αναλυτικά τους δυνατούς τύπους των μεταβλητών και τις δηλώσεις τους, στο Κεφάλαιο 4. Ας παρατηρήσουμε το σύμβολο `&` που υπάρχει πριν από το όνομα της μεταβλητής στην συνάρτηση `scanf()`, θα δούμε τη χρησιμότητα αυτού του *τελεστή* αργότερα, στο Κεφάλαιο 10, προς το παρόν ας σημειώσουμε ότι η `scanf()` απαιτεί τη χρήση του `&` πριν το όνομα της μεταβλητής. Κατά τα άλλα, ένα δείγμα επικοινωνίας με το πρόγραμμα είναι:

```
Δώσε την ηλικία σου σε χρόνια: 30 ↵
Είσαι 30 ετών
```

Η συνάρτηση `scanf()` μπορεί να δεχτεί μονομιάς δεδομένα σε διάφορες μεταβλητές, όπως ακριβώς και η `printf()` μπορεί να εκτυπώσει τις τιμές περισσοτέρων του ενός ορισμάτων της. Π.χ:

```
scanf("%c %s %d", &set, &game, &time);
```

Σ'αυτή την περίπτωση η `scanf()` δέχεται από το πληκτρολόγιο τιμές για τρεις μεταβλητές (η πρώτη τύπου χαρακτήρα, η δεύτερη αλφαριθμητικού και η τρίτη ακεραίου τύπου). Όπως έχει συνταχθεί η `scanf()`, οι τιμές πρέπει (κατά την εισαγωγή τους από το πληκτρολόγιο) να χωρίζονται με ένα ή περισσότερα κενά

διαστήματα. Αυτό συμβαίνει διότι έχουμε χωρίσει με κενά τους καθοριστές μορφής στην `scanf()`.

Στην ουσία μπορούμε να χρησιμοποιήσουμε για το διαχωρισμό των τιμών, οποιοδήποτε *λευκό* χαρακτήρα (κενό, καινούργια γραμμή ή tab). Αν όμως θέλαμε κατά την εισαγωγή να χωρίζουμε τις τιμές με άλλον χαρακτήρα, (π.χ. κόμμα) θα έπρεπε να συμπεριλάβουμε αυτόν το χαρακτήρα μεταξύ των καθοριστών μορφής:

```
scanf("%c,%s,%d", &set, &game, &time);
```

ΚΕΦΑΛΑΙΟ 3
ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ

3.1 ΤΟ ΑΛΦΑΒΗΤΟ

Όλες οι γλώσσες, τόσο οι φυσικές όσο και οι γλώσσες προγραμματισμού, κάνουν χρήση ενός *αλφαβήτου*. Κάνοντας χρήση του αλφαβήτου, μπορούμε να δημιουργήσουμε τις λέξεις, τις παραστάσεις (ή εκφράσεις) και τις εντολές (ή προτάσεις) της γλώσσας, υπακούοντας στους γραμματικούς και συντακτικούς κανόνες της.

Το αλφάβητο της C αποτελείται από τους παρακάτω χαρακτήρες:

(i) Τα *γράμματα* του λατινικού αλφαβήτου, κεφαλαία και πεζά: A..Z, a..z.

(ii) Τα *ψηφία* 0..9. Στην περίπτωση που θέλουμε να παραστήσουμε αριθμούς στο *δεκαεξαδικό* σύστημα αρίθμησης, ψηφία θεωρούνται και τα γράμματα A έως F (ή ισοδύναμα a έως f).

(iii) Τους διάφορους *τελεστές*. Η C διαθέτει ένα πολύ πλούσιο σύνολο τελεστών. Ένας *τελεστής* (operator) είναι ένα σύμβολο, του οποίου η σημασία είναι καθορισμένη για το μεταγλωττιστή. Οι τελεστές που διαθέτει η C μπορούν να καταταχθούν στις επόμενες κατηγορίες: τους *αριθμητικούς* (arithmetic), τους *συσχετιστικούς* (relational), τους *λογικούς* (logical) τους *τελεστές bit* (bitwise), τους *τελεστές δεικτών*, τους *τελεστές απόδοσης τιμής* και *ειδικούς τελεστές* για ειδικές λειτουργίες. Στον Πίνακα 3.1 φαίνονται απλώς οι τελεστές κάθε κατηγορίας. Θα δούμε καθέναν από αυτούς, αναλυτικά σε επόμενα κεφάλαια.

(iv) Τα διάφορα *ειδικά σύμβολα*. Η C διαθέτει επίσης έναν αριθμό *ειδικών συμβόλων*, που χρησιμοποιούνται για διάφορους σκοπούς, όπως π.χ. για ιεράρχηση των πράξεων σε αριθμητικές παραστάσεις, οριοθέτηση αρχής και τέλους ενός κομματιού κώδικα (είδαμε αυτά τα σύμβολα στην § 2.2), τερματισμό εντολών κ.λ.π. Στον Πίνακα 3.1 φαίνονται αυτά τα ειδικά σύμβολα. Η εξήγηση της λειτουργίας καθενός συμβόλου θα γίνεται όταν αυτό συναντάται για πρώτη φορά (όπως έγινε για τα { } στην § 2.2 και τα " " στην § 2.3).

(v) Τους διάφορους χαρακτήρες *ειδικών λειτουργιών* ή ακολουθίες διαφυγής ή σταθερές ανάστροφης καθέτου. Αυτοί είτε είναι χαρακτήρες μη-εκτυπώσιμοι είτε χαρακτήρες ελέγχου είτε χαρακτήρες που χρησιμοποιούνται για ειδικές λειτουργίες. Καθένας απ'αυτούς συνίσταται από μία ανάστροφη κάθετο (\) ακολουθούμενη από έναν ή περισσότερους άλλους χαρακτήρες. Αυτές τις ακολουθίες ο μεταγλωττιστής τις αντιλαμβάνεται ως ένα χαρακτήρα στον οποίον και αποδίδει την κατάλληλη σημασία. Οι χαρακτήρες αυτοί αναφέρονται στην § 3.3.2 (Σταθερές Χαρακτήρα).

Κατηγορία	***Τελεστές***
Αριθμητικοί τελεστές	- + * / % -- ++
Συσχετιστικοί τελεστές	> >= < <= == !=
Λογικοί τελεστές	&& || !
Τελεστές bit	& | ^ ~ >> <<
Τελεστές δεικτών	* &
Τελεστές απόδοσης τιμής	= += -= *= /= %= &= ^= |= <<= >>=
Διάφοροι τελεστές	?: , -> .
Τα ειδικά Σύμβολα της C	
( ) [ ] /* */ { } ; " ' \	

Πίνακας 3.1: *Οι τελεστές και τα ειδικά σύμβολα της C*

3.2 ΟΙ ΛΕΞΕΙΣ

Οι λέξεις της C, χωρίζονται σε δύο κατηγορίες: στις δεσμευμένες λέξεις και στα αναγνωριστικά του προγραμματιστή.

Δεσμευμένες λέξεις (ή λέξεις-κλειδιά, keywords) είναι εκείνες οι λέξεις των οποίων η σημασία είναι ειδική και αμετάβλητη για το μεταγλωττιστή. Δεν επιτρέπεται να γίνεται χρήση μιας δεσμευμένης λέξης για άλλο σκοπό σ'ένα πρόγραμμα. Ο αριθμός των δεσμευμένων λέξεων που χρησιμοποιεί η C, είναι σχετικά μικρός, συγκρινόμενος με τον αντίστοιχο αριθμό άλλων γλωσσών προγραμματισμού. Στον Πίνακα 3.2 φαίνονται οι 32 δεσμευμένες λέξεις της C, όπως ορίζεται από το πρότυπο ANSI. Διάφορες υλοποιήσεις της C έχουν κάποιες επιπλέον δεσμευμένες λέξεις. Για παράδειγμα, στην Turbo C έχουν προστεθεί κάποιες δεσμευμένες λέξεις για καλύτερη διαχείριση της μνήμης, καθώς και για υποστήριξη διαγλωσσικού προγραμματισμού και διακοπών (interrupts). Στον Πίνακα 3.3 φαίνεται το σύνολο των επεκταμένων δεσμευμένων λέξεων της Turbo C.

Προσοχή: όλες οι δεσμευμένες λέξεις της C γράφονται με πεζά γράμματα. Όπως είναι γνωστό, η C διακρίνει τα πεζά από τα κεφαλαία γράμματα. Έτσι π.χ. το `while` είναι δεσμευμένη λέξη, ενώ το `WHILE` ή το `While` δεν είναι.

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Πίνακας 3.2: *Οι 32 δεσμευμένες λέξεις της C, όπως ορίζονται από το πρότυπο ANSI*

asm	_cs	_ds	_es
_ss	cdecl	far	huge
interrupt	near	pascal	

Πίνακας 3.3: *Το σύνολο των επεκταμένων δεσμευμένων λέξεων της Turbo C*

Αναγνωριστικά (identifiers) είναι τα συμβολικά ονόματα που δίνονται από τον προγραμματιστή στις *μεταβλητές*, τις *σταθερές*, τις *συναρτήσεις* και σε διάφορα άλλα αντικείμενα ενός προγράμματος.

Για το σχηματισμό των αναγνωριστικών πρέπει να ακολουθούνται οι παρακάτω κανόνες:

1. Ο πρώτος χαρακτήρας ενός αναγνωριστικού πρέπει να είναι γράμμα του λατινικού αλφαβήτου ή ο χαρακτήρας της υπογράμμισης (_).
2. Οι υπόλοιποι χαρακτήρες μπορούν να είναι γράμματα του λατινικού αλφαβήτου, ψηφία, ή ο χαρακτήρας της υπογράμμισης.
3. Το μέγιστο μήκος ενός αναγνωριστικού εξαρτάται από την υλοποίηση της C. Το πρότυπο ANSI C ορίζει ότι λαμβάνονται υπόψη από το μεταγλωττιστή οι 31 πρώτοι χαρακτήρες ενός αναγνωριστικού. Αυτό σημαίνει ότι αν δύο αναγνωριστικά έχουν τους 31 πρώτους χαρακτήρες τους ίδιους και διαφέρουν από τον τριακοστό δεύτερο και μετά, ο μεταγλωττιστής της C δε θα μπορεί να τα ξεχωρίσει.
4. Ένα αναγνωριστικό δε μπορεί να είναι ίδιο με μια δεσμευμένη λέξη της C.

Με βάση τους παραπάνω κανόνες, τα παρακάτω αναγνωριστικά είναι σωστά:

```
maxnumber
volts
test1
Sep_96
```

Ενώ τα παρακάτω είναι λάθος:

`version1.1`	περιέχει τελεία
`Km/h`	περιέχει /
`25_Jun_94`	αρχίζει με αριθμό
`$100`	αρχίζει με $
`case`	είναι δεσμευμένη λέξη

Προσοχή: καθόσον η C *διακρίνει τα κεφαλαία από τα πεζά γράμματα*, τα `number`, `Number` και `NUMBER`, είναι τρία διαφορετικά αναγνωριστικά.

3.3 ΤΑ ΔΕΔΟΜΕΝΑ

Τα δεδομένα είναι ποσότητες που καταλαμβάνουν κάποια ή κάποιες θέσεις μνήμης (ανάλογα με τον τύπο τους) και αποτελούν το πληροφοριακό υλικό το οποίο επεξεργάζεται το πρόγραμμα. Ανάλογα με τον αν είναι δυνατή ή όχι η

αλλαγή της τιμής ενός δεδομένου κατά τη διάρκεια της εκτέλεσης του προγράμματος, τα δεδομένα διακρίνονται σε μεταβλητές και σταθερές.

3.3.1 ΜΕΤΑΒΛΗΤΕΣ

Μεταβλητή είναι μία ονομασμένη περιοχή στη μνήμη του Η/Υ που κατακρατείται για συγκεκριμένου τύπου δεδομένο και τα περιεχόμενά της δύνανται να αλλάζουν κατά τη διάρκεια εκτέλεσης του προγράμματος. Στην περιοχή αυτή της μνήμης αποδίδεται, από τον προγραμματιστή, ένα αναγνωριστικό για εύκολη αναφορά σ'αυτήν.

Όλες οι μεταβλητές που υπεισέρχονται στο πρόγραμμα πρέπει να *δηλωθούν* (declared) ή *οριστούν* (defined), όσον αφορά το αναγνωριστικό τους και τον *τύπο* των τιμών που λαμβάνουν, για να γνωρίζει ο μεταγλωττιστής πόσο χώρο μνήμης θα δεσμεύσει για καθεμιά απ'αυτές. Οι δυνατοί τύποι τιμών που μπορούν να έχουν οι μεταβλητές, ο χώρος μνήμης που απαιτεί ο καθένας απ'αυτούς και το πώς γίνεται η δήλωση των μεταβλητών, θα καλυφθούν στο Κεφάλαιο 4.

3.3.2 ΣΤΑΘΕΡΕΣ

Σταθερά είναι μία τιμή που δε δύναται να τροποποιηθεί από το πρόγραμμα. Μία τυπική σταθερά είναι π.χ. η τιμή του π (3.14159). Μία σταθερά μπορεί να είναι *αριθμός* (ακέραιος ή πραγματικός, στο δεκαδικό, οκταδικό ή δεκαεξαδικό σύστημα), ένας *χαρακτήρας* ή ένα *αλφαριθμητικό*.

Μία σταθερά μπορεί να χρησιμοποιηθεί απ'ευθείας με την τιμή της μέσα σε μία εντολή του προγράμματος (οπότε λέγεται *κυριολεκτική σταθερά* - literal constant), ή να της αποδοθεί κάποιο αναγνωριστικό (οπότε λέγεται *ονομασμένη σταθερά* - named constant).

Ακέραιες Σταθερές

Οι *ακέραιες* δεκαδικές (στο δεκαδικό σύστημα) σταθερές γράφονται ως μία σειρά από ψηφία π.χ. 1 ή 255 ή 32767. Οι δεκαεξαδικές σταθερές ξεκινούν με 0x ή 0X (ένα μηδενικό ακολουθούμενο από x ή X), σ'αυτή την περίπτωση ως ψηφία μπορούν να χρησιμοποιηθούν τα γράμματα A έως F (ή ισοδύναμα a έως f) για τις τιμές που είναι μεγαλύτερες του 9. Για παράδειγμα ο δεκαδικός 255 γράφεται 0XFF ως δεκαεξαδικός. Οι οκταδικές σταθερές ξεκινούν με ένα μηδενικό, έτσι ο δεκαδικός 255 γράφεται 0377 ως οκταδικός.

Κάθε ακέραια σταθερά μπορεί να έχει ως *επίθεμα* (κατάληξη) ένα ή και τα δύο από τα παρακάτω γράμματα, τα οποία καθορίζουν τον *τύπο* ενός ακεραίου (δες § 4.1.2):

(i) Το γράμμα U (ή u), το οποίο καθορίζει ότι ο ακέραιος είναι *απρόσημος* (`unsigned`).

(ii) Το γράμμα L (ή l), το οποίο καθορίζει ότι πρόκειται για *μεγάλο* (`long`) ακέραιο.

Αυτοί οι τύποι διαφέρουν τόσο στο εύρος των τιμών που μπορούν να λάβουν όσο και στο πλήθος των bit που καταλαμβάνουν στη μνήμη (όπως φαίνεται στο Πίνακα 4.1). Για παράδειγμα η σταθερά 32768L είναι τύπου `long int` και καταλαμβάνει 32 bit, αλλά η 32768U είναι `unsigned int` και καταλαμβάνει 16 bit.

Σταθερές Κινητής Υποδιαστολής

Ο τύπος των σταθερών *κινητής υποδιαστολής* είναι `double` (δες § 4.1) εκτός και αν χρησιμοποιηθεί κάποιο κατάλληλο επίθεμα για να αλλάξει το εύρος τιμών και την ακρίβειά τους. Μπορεί να χρησιμοποιηθεί ένα από τα παρακάτω δύο επιθέματα:

(i) Το γράμμα F (ή f), το οποίο κάνει τη σταθερά τύπου `float`.

(ii) Το γράμμα L (ή l), το οποίο κάνει τη σταθερά τύπου `long double`.

Υπάρχουν δύο μορφές με τις οποίες μπορεί να γραφεί μία σταθερά κινητής υποδιαστολής:

(a) *Τυποποιημένη μορφή*: Ακέραιο μέρος, δεκαδικό σημείο, κλασματικό μέρος, με τη δυνατότητα να παραλειφθεί ένα από τα δύο μέρη. Για παράδειγμα:

0.99 1.0 ή ισοδύναμα .99 1.

(b) *Εκθετική μορφή*: Όπως παραπάνω, αλλά με την ακολουθία ενός `E` (ή `e`) το οποίο δηλώνει δύναμη του 10, και ενός ακέραιου αριθμού που είναι ο εκθέτης. Για παράδειγμα:

7.65E-12 είναι $7.65 \cdot 10^{-12}$

1.5E1 1.5e1 15E0 .15E2 150E-1 είναι όλα το ίδιο: 15

Σταθερές Χαρακτήρα

Μία σταθερά *χαρακτήρα* είναι ένας χαρακτήρας που περικλείεται από μονά εισαγωγικά (`'`) π.χ. `'a'`. Η τιμή μίας σταθεράς χαρακτήρα είναι η αριθμητική τιμή του χαρακτήρα στο σύνολο χαρακτήρων ASCII.

Η χρήση μονών εισαγωγικών σε όλες τις σταθερές χαρακτήρα είναι αρκετή για τους περισσότερους χαρακτήρες, αλλά ορισμένοι ειδικοί χαρακτήρες, όπως η αλλαγή σελίδας (form feed), είναι αδύνατο να δοθούν από το πληκτρολόγιο. Για το λόγο αυτό η C παρέχει τις *ακολουθίες διαφυγής* (escape sequences) ή *σταθερές ανάστροφης καθέτου*, που φαίνονται παρακάτω:

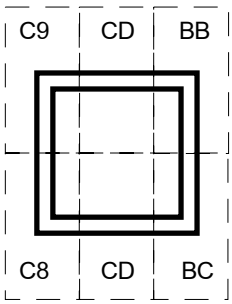
`\a`	ακουστική ειδοποίηση	BELL
`\b`	οπισθοδρόμηση	BS (backspace)
`\f`	αλλαγή σελίδας	FF (form feed)
`\n`	νέα γραμμή	NL(LF) (new line-line feed)
`\r`	επαναφορά κεφαλής	CR (carriage return)
`\t`	οριζόντιος στηλογνώμονας (tab)	HT (horizontal tab)
`\v`	κατακόρυφος στηλογνώμονας	VT (vertical tab)
`\\`	ανάστροφη κάθετος	`\`
`\'`	μονά εισαγωγικά	`'`
`\"`	διπλά εισαγωγικά	`"`
`\?`	λατινικό ερωτηματικό	`?`
`\ooo`	κωδικός ASCII σε οκταδική μορφή (κάθε 'o' αντιπροσωπεύει ένα ψηφίο)	
`\xhh`	κωδικός ASCII σε δεκαεξαδική μορφή (κάθε 'h' αντιπροσωπεύει ένα ψηφίο)	

Η ακολουθία διαφυγής `'\ooo'` αποτελείται από μία ανάστροφη κάθετο ακολουθούμενη από 1, 2 ή 3 οκταδικά ψηφία που καθορίζουν την τιμή του επιθυμητού χαρακτήρα στο σύνολο χαρακτήρων ASCII. Συνηθισμένο παράδειγμα είναι το `'\0'` που είναι ο *ανύπαρκτος χαρακτήρας* (NULL) και που πρέπει να ξεχωρίζεται από το χαρακτήρα `0` (μηδέν) και από το χαρακτήρα *κενό* (space). Η ακολουθία διαφυγής `'\xhh'` αποτελείται με τη σειρά της από μία ανάστροφη κάθετο, ένα `x` και δεκαεξαδικά ψηφία που καθορίζουν την τιμή του επιθυμητού χαρακτήρα.

Μία ακόμα εφαρμογή των δύο τελευταίων ακολουθιών διαφυγής, είναι για τυπώνουμε στην οθόνη χαρακτήρες γραφικών (όπως γωνίες, διπλές γραμμές κ.λ.π.) για τους οποίους δεν υπάρχει αντίστοιχο πλήκτρο στο πληκτρολόγιο. Όμως, όπως γνωρίζουμε, σε κάθε χαρακτήρα αντιστοιχεί ένας ακέραιος αριθμός (κωδικός), σύμφωνα με το σύνολο χαρακτήρων ASCII. Έτσι για να τυπώσουμε αυτούς τους χαρακτήρες, μπορούμε να χρησιμοποιήσουμε στην `printf()` τους κωδικούς τους σε οκταδική ή δεκαεξαδική μορφή. Για παράδειγμα, συμβουλευόμενοι τον πίνακα που δείχνει το σύνολο χαρακτήρων ASCII (Παράρτημα Α), μπορούμε να γράψουμε το παρακάτω πρόγραμμα:

```
main()
{
  printf("\XC9\XCD\XBB\n");
  printf("\XC8\XCD\XBC\n");
}
```

το οποίο, χρησιμοποιώντας δύο `printf()` και τους κωδικούς των χαρακτήρων στην δεκαεξαδική μορφή, εμφανίζει στην οθόνη ένα πλαίσιο με διπλό περίγραμμα:



Αλφαριθμητικές Σταθερές

Μία *αλφαριθμητική* σταθερά (λέγεται και *κυριολεκτικό αλφαριθμητικό*) είναι μία ακολουθία χαρακτήρων ASCII, που περικλείονται από διπλά εισαγωγικά (`"`). Πάλι η ανάστροφη κάθετος μπορεί να χρησιμοποιηθεί για ειδικούς χαρακτήρες που διαφορετικά θα μας δημιουργούσαν πρόβλημα. Παραδείγματα σταθερών αλφαριθμητικών είναι:

```
"ABC"
"A"
"%d\n\n"
"Μέση \"Ταχύτητα\": 50Km\\h"
```

Το τελευταίο παράδειγμα δηλώνει την ακολουθία 23 χαρακτήρων:

```
Μέση "Ταχύτητα": 50Km/h
```

Εσωτερικά, ο ανύπαρκτος χαρακτήρας (NULL) προστίθεται αμέσως μετά τον τελευταίο χαρακτήρα ενός αλφαριθμητικού. Για παράδειγμα η αλφαριθμητική σταθερά "ABC" αποθηκεύεται στη μνήμη του Η/Υ σε τέσσερα bytes, όπου το τέταρτο περιέχει το χαρακτήρα '\0'. Περισσότερα για τα αλφαριθμητικά, θα δούμε στο Κεφάλαιο 8.

Μία αλφαριθμητική σταθερά που περιέχει μόνο ένα χαρακτήρα, όπως η "A", δεν πρέπει να συγχέεται με την αντίστοιχη σταθερά χαρακτήρα 'A'.

Ονομασμένες Σταθερές

Μπορούμε να δώσουμε αναγνωριστικά σε σταθερές *χρησιμοποιώντας την οδηγία προς τον προεπεξεργαστή* (preprocessor directive) #define, ως εξής:

#define *αναγνωριστικό τιμή_σταθεράς*

Δίνοντας αναγνωριστικά σε σταθερές, τα προγράμματα γίνονται ευκολότερα κατανοητά και μπορούν να αλλαχθούν ευκολότερα. Δίνοντας π.χ. το αναγνωριστικό PI στη σταθερά 3.14 ως εξής:

```
#define PI 3.14
```

στη συνέχεια οπουδήποτε στον κώδικα του προγράμματος πρέπει να γράψουμε 3.14 γράφουμε απλώς PI. Αν θελήσουμε να αλλάξουμε την τιμή του π και να την κάνουμε 3.14159 αρκεί να αλλάξουμε την οδηγία #define:

```
#define PI 3.14159
```

Έτσι αν το π εμφανίζεται σε πολλά σημεία μέσα στο πρόγραμμά μας, δε χρειάζεται να αντικαταστήσουμε την καινούργια τιμή του παρά μόνο σε ένα σημείο, στην οδηγία #define.

Παρακάτω φαίνονται μερικές απλές αποδόσεις αναγνωριστικών σε σταθερές:

```
#define LINE 81
#define MASK 0X7F
#define RWMODE 0666
#define PI 3.14159265358979323846
#define HELP '?'
#define PAGE "Σελίδα"
#define MINSINDAY 24*60
#define EPSILON 1E-5
```

Θα ακολουθήσουμε τη σύμβαση να γράφουμε τα αναγνωριστικά των σταθερών με κεφαλαία γράμματα και αυτά των μεταβλητών με πεζά, για να ξεχωρίζουν εύκολα (με ένα απλό κοίταγμα του προγράμματος) οι σταθερές από τις μεταβλητές.

3.4 ΣΧΟΛΙΑ

Σχόλια (comments) είναι τα μέρη εκείνα του πηγαίου προγράμματος τα οποία αγνοούνται από το μεταγλωττιστή. Χρησιμοποιούνται για να κάνουν το πρόγραμμα πιο ευκολονόητο στον αναγνώστη.

Τα σχόλια μπορούν να εμφανίζονται παντού όπου μπορεί να εμφανιστεί ένα κενό διάστημα ή ένας χαρακτήρας στηλογνώμονα ή νέας γραμμής. Τα σχόλια αρχίζουν με το ειδικό σύμβολο `/*` και τελειώνουν με το `*/` και μπορούν να περιέχουν οποιουσδήποτε χαρακτήρες ASCII.

Στη C τα σχόλια δε μπορούν να είναι ένθετα σε άλλα σχόλια. Δηλαδή δε μπορούμε να έχουμε:

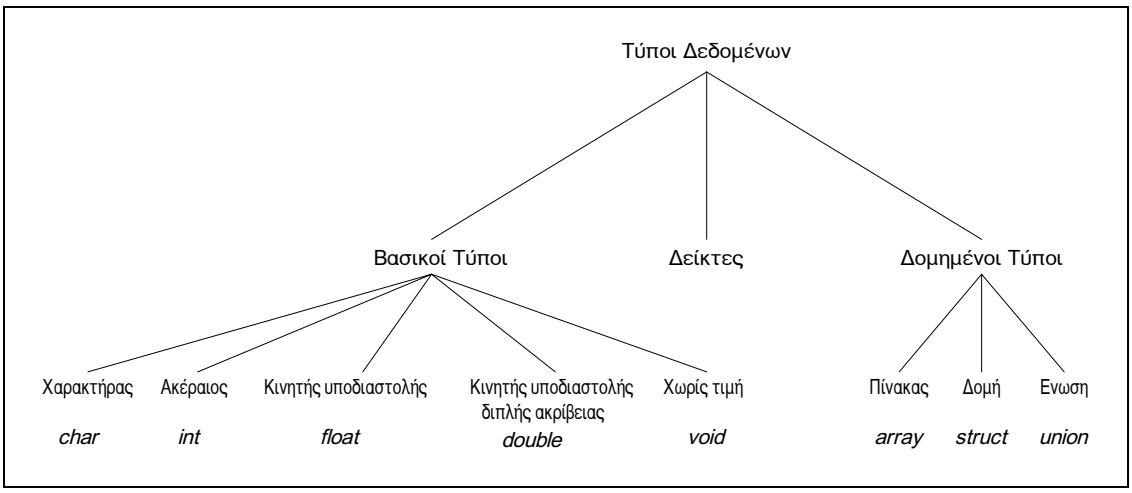
```
/* Σχόλιο1  /* Σχόλιο2 */  συνέχεια σχολίου1 */
```

Στο εξής στα προγράμματά μας, θα τοποθετούμε σχόλια τουλάχιστον στην αρχή τους, που θα αναφέρουν το όνομα του αρχείου πηγαίου κώδικα και τη λειτουργία του προγράμματος.

ΚΕΦΑΛΑΙΟ 4
ΤΥΠΟΙ - ΜΕΤΑΒΛΗΤΕΣ - ΤΕΛΕΣΤΕΣ

4.1 ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Το Σχήμα 4.1 δείχνει τους τύπους δεδομένων που χρησιμοποιεί η C. Σ'αυτό το Κεφάλαιο θα ασχοληθούμε μόνο με τους βασικούς τύπους δεδομένων. Οι άλλοι τύποι δεδομένων αποτελούν αντικείμενο άλλων Κεφαλαίων.



Σχήμα 4.1: *Οι τύποι δεδομένων της C*

4.1.1 ΒΑΣΙΚΟΙ ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Όπως είδαμε στην § 3.3.1, όλες οι μεταβλητές στη C πρέπει να δηλώνονται πριν χρησιμοποιηθούν, για να γνωρίζει ο μεταγλωττιστής πόσο χώρο μνήμης θα δεσμεύσει. Στη C υπάρχουν πέντε βασικοί τύποι δεδομένων: *χαρακτήρας, ακέραιος, πραγματικός κινητής υποδιαστολής, πραγματικός κινητής υποδιαστολής διπλής ακρίβειας* και *χωρίς τιμή*. Οι δεσμευμένες λέξεις που χρησιμοποιούνται, για να δηλώνουν μεταβλητές αυτών των τύπων είναι οι `char`, `int`, `float`, `double` και `void` αντίστοιχα και λέγονται *καθοριστές τύπου* (type specifiers).

Οι μεταβλητές τύπου `char` χρησιμοποιούνται για να αποθηκεύουν 8-μπιτους χαρακτήρες ASCII, ή οποιαδήποτε άλλη 8-μπιτη ποσότητα (μικρούς ακεραίους). Οι μεταβλητές τύπου `int` μπορούν να περιέχουν ακέραιες ποσότητες και μπορούν να χρησιμοποιούνται για να ελέγχουν βρόχους και εντολές με συνθήκη. Οι μεταβλητές τύπου `float` και `double` χρησιμοποιούνται στα προγράμματα είτε όταν απαιτούνται κλασματικοί αριθμοί είτε όταν η εφαρμογή χρειάζεται πολύ

μεγάλους αριθμούς. Η διαφορά ανάμεσα σε μία μεταβλητή `float` και σε μία μεταβλητή `double` βρίσκεται στο μέγεθος του μεγαλύτερου (και του μικρότερου) αριθμού που μπορούν να κρατάνε (δες Πίνακα 4.1). Ο τύπος `void` καθορίζει ένα κενό σύνολο τιμών. Χρησιμοποιείται σαν τύπος που επιστρέφουν οι συναρτήσεις που δεν παράγουν τιμή (δες Κεφάλαιο 7 όπου θα γίνει κατανοητός ο τύπος `void`).

4.1.2 ΤΡΟΠΟΠΟΙΗΤΕΣ ΤΥΠΩΝ

Εκτός του τύπου `void` , στους βασικούς τύπους δεδομένων μπορούν να εφαρμοστούν διάφοροι *τροποποιητές τύπων* (type modifiers). Σκοπός των τροποποιητών είναι να τροποποιούν τους βασικούς τύπους ώστε να ικανοποιούνται καλύτερα οι ανάγκες διαφόρων προγραμμάτων. Οι τροποποιητές είναι:

`signed`	(προσημασμένος)
`unsigned`	(απρόσημος)
`long`	(μεγάλος)
`short`	(μικρός)

Οι τροποποιητές `signed`, `unsigned`, `long` και `short` μπορούν να εφαρμοστούν στους βασικούς τύπους `char` και `int`. Ο τροποποιητής `long` μπορεί να εφαρμοστεί στον βασικό τύπο `double`. Ο Πίνακας 4.1 παρουσιάζει όλους τους επιτρεπόμενους συνδυασμούς των βασικών τύπων με τους τροποποιητές τύπων. Παρόλο που επιτρέπεται, η χρήση του `signed` στους ακέραιους και τους χαρακτήρες είναι περιττή, γιατί εξ'ορισμού η δήλωση ενός ακέραιου ή χαρακτήρα αφορά έναν προσημασμένο αριθμό.

Τύπος	***Εύρος σε bytes***	***Εύρος τιμών***
`char`	1	-128 έως 127
`unsigned char`	1	0 έως 255
`signed char`	1	-128 έως 127
`int`	2	-32768 έως 32767
`unsigned int`	2	0 έως 65535
`signed int`	2	-32768 έως 32767
`short int`	2	-32768 έως 32767
`unsigned short int`	2	0 έως 65535
`signed short int`	2	-32768 έως 32767
`long int`	4	-2147483648 έως 2147483647
`signed long int`	4	-2147483648 έως 2147483647
`unsigned long int`	4	0 έως 4294967295
`float`	4	3.4E-38 έως 3.4E+38
`double`	8	1.7E-308 έως 1.7E+308
`long double`	10	3.4E-4932 έως 1.1E+4932

Πίνακας 4.1: *Οι δυνατοί συνδυασμοί βασικών τύπων και τροποποιητών της C*

4.2 ΔΗΛΩΣΕΙΣ ΜΕΤΑΒΛΗΤΩΝ

Όλες οι μεταβλητές πρέπει να δηλώνονται πριν χρησιμοποιηθούν. Μία δήλωση καθορίζει τον τύπο και περιέχει μία λίστα μεταβλητών αυτού του τύπου, όπως φαίνεται παρακάτω:

τύπος λίστα_μεταβλητών;

όπου *τύπος* πρέπει να είναι ένας αποδεκτός τύπος δεδομένου της C, ενώ η *λίστα μεταβλητών* μπορεί να αποτελείται από ένα ή περισσότερα αναγνωριστικά χωρισμένα με κόμμα. Παραδείγματα δηλώσεων μεταβλητών είναι:

```
int i,j,k;
char ch;
short int si;
unsigned long int ui;
double lower,upper;
```

Η *θέση* όπου δηλώνεται μία μεταβλητή επηρεάζει σε μεγάλο βαθμό τον τρόπο με τον οποίο μπορούν να χρησιμοποιήσουν τη μεταβλητή τα υπόλοιπα τμήματα του προγράμματος. Οι κανόνες που καθορίζουν την ορατότητα μίας μεταβλητής με βάση τη θέση όπου δηλώθηκε αυτή, λέγονται *κανόνες εμβέλειας*. Θα δούμε αναλυτικά αυτούς τους κανόνες στο Κεφάλαιο 7. Εδώ απλώς θα αναφέρουμε τα βασικά στοιχεία αυτών των κανόνων:

Υπάρχουν τρεις θέσεις σε ένα πρόγραμμα της C όπου μπορούν να δηλωθούν μεταβλητές. Η πρώτη θέση είναι έξω από όλες τις συναρτήσεις, συμπεριλαμβανομένης της συνάρτησης `main()`. Οι μεταβλητές που δηλώνονται κατ'αυτόν τον τρόπο καλούνται *εξωτερικές* (external) και έχουν *καθολική* (global) εμβέλεια. Έτσι μπορούν να χρησιμοποιούνται από οποιοδήποτε τμήμα του προγράμματος. Η δεύτερη θέση όπου μπορούν να δηλωθούν μεταβλητές είναι μέσα σε μία συνάρτηση. Οι μεταβλητές που δηλώνονται κατ'αυτόν τον τρόπο έχουν *τοπική* (local) εμβέλεια και μπορούν να χρησιμοποιούνται μόνο από αυτή τη συνάρτηση. Η τελευταία θέση όπου μπορούν να δηλωθούν μεταβλητές είναι στη δήλωση των τ*υπικών παραμέτρων* μίας συνάρτησης. Εκτός του ότι διαβιβάζουν τιμές στη συνάρτηση, αυτές οι παράμετροι ενεργούν όπως οι υπόλοιπες τοπικές μεταβλητές.

Μία μεταβλητή μπορεί να λάβει *αρχική τιμή* κατά τη δήλωσή της, χρησιμοποιώντας το σύμβολο = και μία σταθερά μετά το όνομα της μεταβλητής. Η γενική μορφή της απόδοσης αρχικής τιμής κατά τη δήλωση, είναι:

τύπος αναγνωριστικό_μεταβλητής **=** *σταθερά;*

Μερικά παραδείγματα φαίνονται παρακάτω:

```
char ch='C';
int i=0;
float eps=1.0E-5;
```

Οι εξωτερικές μεταβλητές μπορούν να παίρνουν αρχική τιμή μόνο στην αρχή του προγράμματος. Αν δεν τους δοθεί κάποια αρχική τιμή, τότε παίρνουν αυτόματα αρχική τιμή μηδέν. Οι τοπικές μεταβλητές όμως, παίρνουν αρχική τιμή κάθε φορά που χρησιμοποιείται η συνάρτηση στην οποία είναι δηλωμένες. Αν στις τοπικές μεταβλητές δε δώσουμε αρχική τιμή, έχουν α*προσδιόριστη τιμή* (σκουπίδια).

Στη δήλωση οποιασδήποτε μεταβλητής με αρχική τιμή, μπορεί να εφαρμόζεται ο *προσδιοριστής τύπου* (type qualifier) `const` που εξασφαλίζει ότι η τιμή της δε θα αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος, φέροντας έτσι το ίδιο σχεδόν αποτέλεσμα με την `#define`. Π.χ:

```
const double e=2.71828182845905;
```

Αν γίνει απόπειρα αλλαγής μίας const, ο μεταγλωττιστής δίνει μήνυμα λάθους. Περισσότερα για τις δηλώσεις μεταβλητών, θα δούμε στο Κεφάλαιο 7.

Παρατηρήσεις:

1. Όπως γίνεται φανερό από τα παραπάνω, η C δε διαθέτει το λογικό τύπο (boolean). Αντί αυτού, χρησιμοποιεί τον τύπο `int` και ερμηνεύει την τιμή 0 ως false και οποιαδήποτε μη-μηδενική τιμή ως true.
2. Μία μεταβλητή τύπου αλφαριθμητικού (string), ορίζεται ως *πίνακας χαρακτήρων*, π.χ:

   ```
   char str[20]; /* Αλφαριθμητικό με όνομα str και
                    μήκος 20 */
   ```

   Περισσότερα για τα αλφαριθμητικά θα δούμε στο Κεφάλαιο 8.

4.3 ΤΕΛΕΣΤΕΣ

Όπως είδαμε στην § 3.1, η C έχει ένα πολύ πλούσιο σύνολο τελεστών τους οποίους παραθέσαμε στον Πίνακα 3.1. Σε τούτη την παράγραφο θα καλύψουμε ορισμένες από τις κατηγορίες των τελεστών, ξεκινώντας από τους αριθμητικούς τελεστές.

4.3.1 ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

Στον Πίνακα 4.2 παρουσιάζονται οι επτά αριθμητικοί τελεστές της C.

Αριθμητικός Τελεστής	***Πράξη***
`-`	Αφαίρεση, μοναδικό πλην
`+`	Πρόσθεση
`*`	Πολλαπλασιασμός
`/`	Διαίρεση
`%`	Υπόλοιπο διαίρεσης ακεραίων
`--`	Μοναδιαία μείωση
`++`	Μοναδιαία αύξηση

Πίνακας 4.2: *Οι αριθμητικοί τελεστές της C*

Οι αριθμητικοί τελεστές με δύο τελεστέους (δυαδικοί) `+`, `-`, `*` και `/` λειτουργούν στη C με τον ίδιο τρόπο που λειτουργούν και στις άλλες γλώσσες προγραμματισμού.

Ο τελεστής / της διαίρεσης χρειάζεται ειδική προσοχή. Στην πραγματικότητα υπάρχουν δύο διαφορετικές πράξεις διαίρεσης και η κάθετος (/) χρησιμοποιείται και για τις δύο. Αν διαιρέσουμε δύο ακεραίους, παίρνουμε το *ακέραιο πηλίκο* της διαίρεσης, π.χ:

```
10/3=3
```

Αν στη διαίρεση x/y τουλάχιστον ένας από τους τελεστέους είναι τύπου κινητής υποδιαστολής, το αποτέλεσμα θα είναι τύπου κινητής υποδιαστολής. Αυτό ισχύει επίσης στην πρόσθεση, αφαίρεση και πολλαπλασιασμό. Αλλά γι'αυτές τις πράξεις η τιμή του αποτελέσματος δεν επηρεάζεται συνήθως σοβαρά από τον τύπο των τελεστέων. Για παράδειγμα:

```
4+3=7                (ακέραιος)
4.0+3=7.0            (κινητής υποδιαστολής)
```

Με το / όμως, οι τύποι δεδομένων των τελεστέων έχουν δραστική επίδραση στο αποτέλεσμα:

```
4/3=1                (ακέραιος)
4.0/3=1.333333...    (κινητής υποδιαστολής)
```

Ο τελεστής υπολοίπου % παράγει το υπόλοιπο μίας διαίρεσης ακεραίων. Δε μπορούμε να χρησιμοποιήσουμε το % με τους τύπους `float` ή `double`. Έτσι για παράδειγμα:

```
39%5=4               (εφόσον 39=7*5+4)
```

Ο *μοναδικός* (unary) τελεστής − πολλαπλασιάζει τον τελεστέο του με -1. Έτσι αν τοποθετηθεί το σύμβολο πλην μπροστά από κάποιον αριθμό, θα προκληθεί αλλαγή του προσήμου του.

Η C διαθέτει δύο ασυνήθιστους τελεστές για την αύξηση και μείωση των τιμών των *μεταβλητών* κατά 1. Ο *τελεστής μοναδιαίας αύξησης* ++ προσθέτει 1 στον τελεστέο του, ενώ ο *τελεστής μοναδιαίας μείωσης* −− αφαιρεί 1. Έτσι οι πράξεις:

```
x++
x--
```

είναι ίδιες με τις:

```
x=x+1
x=x-1
```

Το ασυνήθιστο είναι ότι οι ++ και −− μπορούν να χρησιμοποιούνται είτε σαν *προθεματικοί* τελεστές (πριν από τη μεταβλητή, όπως ++x) είτε σαν *μεταθεματικοί* (μετά τη μεταβλητή, όπως x++). Και στις δύο περιπτώσεις το αποτέλεσμα είναι η μοναδιαία αύξηση του τελεστέου. Αλλά η προθεματική παράσταση αυξάνει την τιμή του τελεστέου *πριν* χρησιμοποιηθεί η τιμή του, ενώ η μεταθεματική παράσταση αυξάνει την τιμή του τελεστέου *αφού* έχει χρησιμοποιηθεί η τιμή του. Αυτό σημαίνει ότι στις περιπτώσεις που χρησιμοποιείται η τιμή και όχι απλώς το αποτέλεσμα, οι x++ και ++x είναι διαφορετικές. Αν για παράδειγμα το x είναι 5, τότε η παράσταση:

```
y=x++
```

δίνει στο y τιμή 5, ενώ η παράσταση:

```
y=++x
```

του δίνει τιμή 6. Και στις δύο περιπτώσεις, το x γίνεται 6.

4.3.2 ΣΥΣΧΕΤΙΣΤΙΚΟΙ ΚΑΙ ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ

Στους όρους *συσχετιστικός τελεστής* και *λογικός τελεστής*, η λέξη *συσχετιστικός* αναφέρεται στις σχέσεις που μπορούν να έχουν οι τελεστέοι, ενώ η λέξη *λογικός* αναφέρεται στους τρόπους με τους οποίους μπορούν να συνδεθούν μεταξύ τους αυτές οι σχέσεις.

Το κλειδί στις έννοιες των συσχετιστικών και λογικών τελεστών, είναι οι ιδέες της *αλήθειας* (true) και του *ψεύδους* (false). Στη C, αληθής είναι οποιαδήποτε παράσταση που έχει τιμή διάφορη του μηδενός, ενώ ψευδής οποιαδήποτε παράσταση που έχει τιμή ίση με μηδέν. Οι παραστάσεις που χρησιμοποιούν συσχετιστικούς ή λογικούς τελεστές επιστρέφουν το μηδέν σαν ψευδή τιμή και το ένα σαν αληθή τιμή. Στον Πίνακα 4.3 παρουσιάζονται οι συσχετιστικοί και λογικοί τελεστές.

Οι συσχετιστικοί τελεστές χρησιμοποιούνται για να καθορίζουν τη σχέση της μίας παράστασης με την άλλη. Επιστρέφουν πάντοτε 1 ή 0, ανάλογα με το αποτέλεσμα του ελέγχου. Το παρακάτω πρόγραμμα δείχνει το αποτέλεσμα κάθε πράξης εμφανίζοντάς το με 0 ή 1:

```
/* relation.c */
/* Χρήση των συσχετιστικών τελεστών */
main()
{
  int i,j;

  printf("Δώσε δύο ακέραιους αριθμούς: ");
  scanf("%d %d",&i,&j);
  printf("Η σχέση %d==%d είναι %d\n",i,j,i==j);
  printf("Η σχέση %d!=%d είναι %d\n",i,j,i!=j);
  printf("Η σχέση %d<=%d είναι %d\n",i,j,i<=j);
  printf("Η σχέση %d>=%d είναι %d\n",i,j,i>=j);
  printf("Η σχέση %d<%d είναι %d\n",i,j,i<j);
  printf("Η σχέση %d>%d είναι %d\n",i,j,i>j);
}
```

Το παραπάνω πρόγραμμα περιλαμβάνει το σύμβολο, το '\n' το οποίο, όπως έχουμε δει στην § 3.3.2, είναι ένας μόνο χαρακτήρας που, όταν παρεμβάλλεται σε ένα αλφαριθμητικό, προκαλεί μία αλλαγή γραμμής, δηλαδή μετά το χαρακτήρα '\n' η εκτύπωση αρχίζει απ'την αρχή μίας νέας γραμμής.

Οι λογικοί τελεστές χρησιμοποιούνται για να υποστηρίζουν τις βασικές λογικές πράξεις ΚΑΙ (AND), Η (OR), και ΟΧΙ (NOT) με βάση τον πίνακα αλήθειας που ακολουθεί, ο οποίος χρησιμοποιεί το 1 για το αληθές και το 0 για το ψευδές:

p	q	p AND q	p OR q	NOT p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Συσχετιστικός Τελεστής	*Σημασία*
>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο
<	Μικρότερο
<=	Μικρότερο ή ίσο
==	Ίσο
!=	Διάφορο

Λογικός Τελεστής	*Πράξη*
&&	AND (λογική σύζευξη)
||	OR (λογική διάζευξη)
!	NOT (λογική άρνηση)

Πίνακας 4.3: *Οι συσχετιστικοί και οι λογικοί τελεστές της C*

Τόσο οι συσχετιστικοί όσο και οι λογικοί τελεστές, έχουν χαμηλότερη προτεραιότητα από τους αριθμητικούς τελεστές. Αυτό σημαίνει ότι η παράσταση:

```
15>1+20
```

θα υπολογιστεί σαν να ήταν γραμμένη:

```
15>(1+20)
```

Το παρακάτω πρόγραμμα δείχνει πώς λειτουργούν οι λογικοί τελεστές:

```
/* logical.c */
/* Χρήση λογικών τελεστών */
main()
{
  int i,j;

  printf("Δώσε δύο αριθμούς (ο καθένας να είναι 0 ή 1): ");
  scanf("%d %d",&i,&j);
  printf("%d AND %d μας κάνει %d\n",i,j,i&&j);
  printf("%d OR %d μας κάνει %d\n",i,j,i||j);
  printf("NOT %d μας κάνει %d\n",i,!i);
}
```

4.3.3 ΤΕΛΕΣΤΕΣ BIT (BITWISE)

Η C διαθέτει έξι τελεστές για χειρισμό bit (bitwise). Αυτοί μπορούν να εφαρμόζονται μόνο σε *ακέραιους* τελεστέους, δηλαδή σε `char`, `short`, `int` και `long`, προσημασμένους ή απρόσημους και δρουν πάνω στα bits των ακεραίων. Στον Πίνακα 4.4 παρουσιάζονται οι τελεστές bit.

Για παράδειγμα, η τιμή του 23&26 είναι 18. Για να το καταλάβουμε αυτό πρέπει να είμαστε εξοικειωμένοι με το δυαδικό σύστημα αριθμών, όπου οι αριθμοί γράφονται ως ακολουθίες δυαδικών ψηφίων, στη συνέχεια η πράξη AND εφαρμόζεται για τα αντίστοιχα bits των τελεστέων:

```
23 = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1
26 = 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0   &
-----------------------------------------
18 = 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
```

Τελεστής bit	*Πράξη*
&	AND (σύζευξη)
|	OR (διάζευξη)
^	XOR (αποκλειστική διάζευξη)
~	Συμπλήρωμα ως προς ένα
>>	Ολίσθηση δεξιά
<<	Ολίσθηση αριστερά

Πίνακας 4.4: *Οι τελεστές bit της C*

Η γενική μορφή μίας παράστασης *ολίσθησης* είναι:

μεταβλητή **>>** *(ή* **<<***) αριθμός_θέσεων*

Σημειώστε ότι ολίσθηση ενός ακεραίου κατά μία θέση δεξιά ισοδυναμεί με διαίρεσή του με το 2, ενώ ολίσθηση κατά μία θέση αριστερά ισοδυναμεί με πολλαπλασιασμό του με το 2. Για παράδειγμα έστω:

```
x=18                    (=0000000000010010)
```

τότε με δεξιά ολίσθηση κατά ένα bit (`x>>1`) προκύπτει:

```
0000000000001001   (=9)
```

ενώ με αριστερή ολίσθηση κατά ένα bit (`x<<1`) προκύπτει:

```
0000000000100100   (=36)
```

Είναι προφανές ότι ολίσθηση κατά περισσότερες της μίας θέσης δεξιά ή αριστερά, ισοδυναμεί με διαδοχικές διαιρέσεις ή πολ/μούς του αριθμού με το 2.

4.3.4 ΤΕΛΕΣΤΕΣ ΑΠΟΔΟΣΗΣ ΤΙΜΗΣ (ΚΑΤΑΧΩΡΗΣΗΣ)

Στη C, ο τελεστής απόδοσης τιμής (ή τελεστής καταχώρησης) είναι το σημείο ίσον (=). Ωστόσο, η C επιτρέπει μία πολύ βολική στενογραφική μορφή για αποδόσεις τιμών του γενικού τύπου:

μεταβλητή **=** *μεταβλητή τελεστής παράσταση*

ως εξής:

μεταβλητή τελεστής **=** *παράσταση*

Έτσι π.χ. οι παραστάσεις:

```
x=x+10
y=y/2
```

μπορούν να γραφούν:

```
x+=10
y/=2
```

4.4 ΠΑΡΑΣΤΑΣΕΙΣ

Οι τελεστές, οι σταθερές και οι μεταβλητές είναι τα συστατικά των παραστάσεων. Μία *παράσταση* στη C είναι οποιοσδήποτε έγκυρος συνδυασμός αυτών των στοιχείων. Ένα παράδειγμα παράστασης είναι:

```
x=a+b
```

που είναι μία παράσταση απόδοσης τιμής η οποία καταχωρεί το άθροισμα των `a` και `b` στο `x`.

Σε αντίθεση με άλλες γλώσσες προγραμματισμού, η C μας επιτρέπει να χρησιμοποιούμε τους τελεστές απόδοσης τιμής σε παραστάσεις που περιέχουν και άλλους τελεστές απόδοσης τιμής. Αυτό διότι ο τελεστής απόδοσης τιμής κάνει δύο πράγματα: Πρώτον αποδίδει την τιμή της δεξιάς πλευράς στη μεταβλητή της αριστερής πλευράς και δεύτερον *η όλη παράσταση παίρνει αυτή την τιμή*. Έτσι, για παράδειγμα, η παρακάτω παράσταση είναι σωστή:

```
y=3*(x=5+(u=7))+1
```

μετά την εκτέλεση της παραπάνω παράστασης θα έχουμε `u=7`, `x=12`, `y=37` και η συνολική τιμή της παράστασης θα είναι 37. Έστω επίσης η παράσταση:

```
r=(p=x*y)<0
```

στην αρχή υπολογίζεται το γινόμενο `x*y` και η τιμή του αποδίδεται στο `p`. Στη συνέχεια ελέγχεται η τιμή της παράστασης που έχει τοποθετηθεί στις παρενθέσεις, σε σχέση με το μηδέν. Αν είναι μικρότερη του μηδενός τότε στο `r` αποδίδεται η τιμή 1 διαφορετικά αποδίδεται η τιμή 0. Έτσι αν `x=1` και `y=3` τότε το `r` παίρνει τιμή 0.

Αν θέσουμε το σημείο `;` (ελληνικό ερωτηματικό, που λέγεται και semicolon) στο τέλος μίας παράστασης, το αποτέλεσμα είναι μία *εντολή*. Έτσι:

`t=5` είναι *παράσταση* απόδοσης τιμής
`t=5;` είναι *εντολή* απόδοσης τιμής

4.5 ΜΕΤΑΤΡΟΠΕΣ ΤΥΠΩΝ

4.5.1 ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΤΡΟΠΕΣ

Όταν ένας τελεστής δρα σε τελεστέους διαφορετικών τύπων, αυτοί πριν γίνει η πράξη, μετατρέπονται αυτόματα σε ενιαίο τύπο με βάση μερικούς κανόνες. Γενικά οι μόνες *αυτόματες* μετατροπές είναι αυτές που μετατρέπουν έναν τελεστέο "κατώτερου" τύπου σε έναν τελεστέο "ανώτερου" τύπου. Το αποτέλεσμα της πράξης, φυσικά ανήκει στον ανώτερο τύπο. Η διαβάθμιση των τύπων αντιστοιχεί στον αριθμό των bytes που καταλαμβάνει αυτός ο τύπος στη μνήμη.

Οι κανόνες μετατροπής είναι οι παρακάτω:

1. Όλοι οι τύποι `char` και `short int` μετατρέπονται σε `int`. Όλοι οι `float` μετατρέπονται σε `double`.

2. Για όλα τα ζεύγη τελεστέων, η σειρά μετατροπής είναι η παρακάτω. Αν ο ένας τελεστέος είναι `long double` και ο άλλος τελεστέος μετατρέπεται σε `long double`. Αν ο ένας από τους τελεστέους είναι `double` και ο άλλος μετατρέπεται σε `double`. Αν ο ένας τελεστέος είναι `long` και ο άλλος μετατρέπεται σε `long`. Αν ο ένας είναι `unsigned` και ο άλλος μετατρέπεται σε `unsigned`.

Αφού ο μεταγλωττιστής εφαρμόσει αυτούς τους κανόνες μετατροπής, το κάθε ζευγάρι τελεστέων θα γίνει του αυτού τύπου, οπότε το αποτέλεσμα της κάθε πράξης θα είναι του ίδιου τύπου με τον τύπο των τελεστέων. Σημειώνουμε ότι ο δεύτερος κανόνας έχει αρκετές συνθήκες που πρέπει *να εφαρμοστούν διαδοχικά*.

Οι παραπάνω κανόνες ισχύουν προφανώς για το αποτέλεσμα του δεξιού μέρους μίας παράστασης καταχώρησης. Στη συνέχεια, το αποτέλεσμα αυτό καταχωρούμενο στη μεταβλητή του αριστερού μέρους, μετατρέπεται στον τύπο της μεταβλητής αυτής. Έτσι χρειάζεται προσοχή σε παραστάσεις που περιέχουν δεδομένα διαφόρων τύπων!.

Για παράδειγμα, ας δούμε το παρακάτω πρόγραμμα:

```
/* type_con.c */
/* Μετατροπές τύπων */
main()
{
  int i=16;
  char ch='C';

  printf("%d\n",ch);
  i=i+(2*ch);
  printf("%d",i);
}
```

Αυτό το πρόγραμμα τυπώνει στην οθόνη:

```
67
150
```

Ο πρώτος αριθμός (67) είναι το αποτέλεσμα της πρώτης `printf()`, στην οποία ζητάμε να τυπώσουμε τη μεταβλητή `ch` τύπου χαρακτήρα ως ακέραιο. Έτσι τυπώνεται ο κωδικός ASCII του C, που είναι 67. Στη συνέχεια χρησιμοποιούμε τη μεταβλητή `ch` στην πράξη `i+(2*ch)` το αποτέλεσμα της οποίας (150) καταχωρείται στη μεταβλητή `i` η οποία και τυπώνεται.

4.5.2 ΡΗΤΕΣ ΜΕΤΑΤΡΟΠΕΣ (CAST)

Σε οποιαδήποτε παράσταση μπορούν να επιβληθούν ρητές μετατροπές τύπου, με ένα μοναδικό τελεστή που λέγεται *προσαρμογή* (cast). Η γενική μορφή της δομής προσαρμογής είναι:

(τύπος) *παράσταση*

όπου *τύπος* είναι ένας από τους καθιερωμένους τύπους της C και η τιμή της *παράστασης* μετατρέπεται στον κατονομαζόμενο τύπο. Ας προσέξουμε ότι η προσαρμογή παράγει μία κατάλληλου τύπου *τιμή* της παράστασης, η ίδια όμως η παράσταση *δεν αλλάζει*.

Για παράδειγμα, αν ο `x` είναι τύπου `int` (με τιμή 7) και ο `y` τύπου `float`, η παρακάτω παράσταση:

```
y=(float)x/2
```

δίνει στο `y` την τιμή 3.50000. Εδώ η προσαρμογή `(float)` συνδυάζεται με το `x`, πράγμα που κάνει την τιμή του να μετατραπεί σε `float` οπότε και το αποτέλεσμα θα είναι τύπου `float`. Προσοχή, αν γράψουμε την παράσταση με τον τρόπο:

```
y=(float)(x/2)
```

το `y` θα πάρει την τιμή 3.000000. Αυτό συμβαίνει διότι πρώτα εκτελείται μία ακέραια διαίρεση και *το αποτέλεσμα* (που είναι το ακέραιο πηλίκο) μετατρέπεται σε `float`.

4.6 ΠΡΟΤΕΡΑΙΟΤΗΤΑ ΤΕΛΕΣΤΩΝ

Ο Πίνακας 4.5 συνοψίζει τους κανόνες προτεραιότητας και προσεταιριστικότητας για όλους τους τελεστές, συμπεριλαμβανομένων και αυτών που δεν έχουμε ακόμα εξετάσει.

Οι τελεστές που είναι στην ίδια γραμμή του πίνακα έχουν την ίδια προτεραιότητα, ενώ οι γραμμές είναι με αύξουσα σειρά προτεραιότητας. Αν θέλουμε να αλλάξουμε τη σειρά των πράξεων χρησιμοποιούμε παρενθέσεις.

Όταν λέμε *προσεταιριστικότητα* εννοούμε τη *φορά υπολογισμού*. Οι περισσότεροι τελεστές προσεταιρίζουν από αριστερά προς τα δεξιά. Αυτό σημαίνει ότι, για παράδειγμα, η παράσταση:

```
100-20-1+3
```

πρέπει να ερμηνευθεί ως:

```
((100-20)-1)+3
```

Αν ο υπολογισμός γινόταν σαν

```
100-(20-(1+3))
```

(που δε συμβαίνει !) θα λέγαμε ότι οι τελεστές προσεταιρίζουν από δεξιά προς τα αριστερά.

Έτσι, σύμφωνα με τα περιεχόμενα του Πίνακα 4.5, η παράσταση:

$$x = \frac{-c \cdot (a + b)}{(d + e) \cdot (f + g)} \cdot k^2$$

κωδικοποιείται στην C, ως εξής:

```
x=((-c*(a+b))/((d+e)*(f+g)))*k*k
```

Τελεστές	*Προσεταιριστικότητα*
() [] -> .	→
! ~ ++ -- -(1) *(2) &(3) (τύπος) sizeof	←
*(4) / %	→
+ -(5)	→
<< >>	→
< <= > >=	→
== !=	→
&(6)	→
^	→
|	→
&&	→
||	→
?:	←
= += -= *= /= %= &= ^= |= <<= >>=	←
,	→

Υπόμνημα
(1) Μοναδικό πλην
(2) Περιεχόμενο διεύθυνσης μνήμης
(3) Διεύθυνση μνήμης μεταβλητής
(4) Πολλαπλασιασμός
(5) Αφαίρεση
(6) Bitwise AND

Πίνακας 4.5: *Προτεραιότητα και προσεταιριστικότητα τελεστών*

4.7 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί πρόγραμμα που να εμφανίζει στην οθόνη το όνομά σας μέσα σε διπλό περίγραμμα.

```
/* name.c */
/* Εκτύπωση ονόματος μέσα σε διπλό περίγραμμα */
#define DOUBLE_LINE "\xCD\xCD\xCD\xCD\xCD\xCD\xCD"
main()
{
  printf("\n\xC9\%s\xBB",DOUBLE_LINE);
  printf("\n\xBA ONOMA \xBA");
  printf("\n\xC8\%s\xBC",DOUBLE_LINE);
}
```

2. Να γραφεί πρόγραμμα το οποίο να δέχεται από το πληκτρολόγιο ένα χαρακτήρα και να τυπώνει τον κωδικό ASCII αυτού.

```
/* ascii.c */
/* Εύρεση του κωδικού ASCII ενός χαρακτήρα */
main()
{
  char ch;

  printf("\nΔώσε ένα χαρακτήρα: ");
  scanf("%c",&ch);
  printf("\nΟ κωδικός ASCII του χαρακτήρα \"%c\" είναι %d\n",ch,ch);
}
```

3. Να γραφεί πρόγραμμα το οποίο να δέχεται έναν ακέραιο αριθμό και να βρίσκει αν αυτός είναι άρτιος (ζυγός), αν είναι άρτιος να εμφανίζει στην οθόνη το μήνυμα: "Αρτιος=1", ενώ αν δεν είναι άρτιος να εμφανίζει "Αρτιος=0". Υπενθυμίζεται ότι ένας ακέραιος είναι άρτιος όταν το υπόλοιπο της διαίρεσής του με το 2 είναι 0.

```
/* even.c */
/* Ελεγχος αρτιότητας ακεραίου */
main()
{
  int i,x;

  printf("Δώσε έναν ακέραιο: ");
  scanf("%d",&i);
  x=!(i%2);
  printf("Άρτιος=%d",x);
}
```

4. Να γράψετε πρόγραμμα που να δέχεται το μισθό ενός υπαλλήλου (σε ακέραιο αριθμό δραχμών έως του ποσού 32767) και να τυπώνει στην οθόνη τον ελάχιστο αριθμό χαρτονομισμάτων και κερμάτων που χρειάζονται για να πληρωθεί. Δηλαδή να τυπώνει πόσα δεκαχίλιαρα, πεντοχίλιαρα, χιλιάρικα, πεντακοσάρικα, κ.λ.π. πρέπει να δοθούν στον υπάλληλο ώστε αυτά να είναι ελάχιστα στον αριθμό.

```
/* salary.c */
/* Μετατροπή μισθού σε χαρτονομίσματα και κέρματα */
main()
{
  int salary;

  printf("Δώσε μισθό (μέχρι 32767): ");
  scanf("%d",&salary);
  printf("\nΔεκαχίλιαρα  : %d",salary/10000);
  salary%=10000;
  printf("\nΠεντοχίλιαρα : %d",salary/5000);
  salary%=5000;
  printf("\nΧιλιάρικα    : %d",salary/1000);
  salary%=1000;
  printf("\nΠεντακοσάρικα: %d",salary/500);
  salary%=500;
  printf("\nΔιακοσάδραχμα: %d",salary/200);
  salary%=200;
  printf("\nΚατοστάρικα  : %d",salary/100);
```

```
  salary%=100;
  printf("\nΠενηντάρικα  : %d",salary/50);
  salary%=50;
  printf("\nΕικοσάρικα   : %d",salary/20);
  salary%=20;
  printf("\nΔεκάρικα     : %d",salary/10);
  salary%=10;
  printf("\nΤάληρα       : %d",salary/5);
  salary%=5;
  printf("\nΔίδραχμα     : %d",salary/2);
  salary%=2;
  printf("\nΔραχμές      : %d",salary);
}
```

5. Ας υποθέσουμε ότι ο μισθός ενός εργαζομένου υπολογίζεται ως εξής: για κάθε χρόνο υπηρεσίας προσαυξάνεται κατά 4% επί του αρχικού μισθού και για κάθε παιδί προσαυξάνεται κατά 5000 δρχ. Γράψτε πρόγραμμα που να δέχεται τον αρχικό μισθό (έως 32767), τα χρόνια υπηρεσίας και τον αριθμό των παιδιών ενός υπαλλήλου και να υπολογίζει και τυπώνει το συνολικό μισθό του.

```
/* saqlary1.c */
/* Υπολογισμός μισθού υπαλλήλου */
main()
{
  int salary,years,childs;
  float total;

  printf("Δώσε αρχικό μισθό (έως 32767) έτη υπηρ. αρ. παιδιών: ");
  scanf("%d %d %d",&salary,&years,&childs);
  total=salary+salary*0.04*years+5000*childs;
  printf("Συνολικός μισθός: %f Δρχ.",total);
}
```

4.8 ΑΣΚΗΣΕΙΣ

1. Να υπολογιστεί η τιμή της παράστασης:

   `x=i*j>k&&i+j<k`

   στις εξής περιπτώσεις:

   α) `i=2, j=8, k=12` β) `i=2, j=8, k=10` γ) `i=3, j=1, k=7`

2. Θεωρήστε γνωστό ότι η συνάρτηση `sqrt()` επιστρέφει την τετραγωνική ρίζα του ορίσματός της. Κωδικοποιήστε στην C τις παρακάτω αριθμητικές παραστάσεις:

   α) $2\pi \cdot \sqrt{\frac{m}{D}}$ β) $K + \frac{X \cdot K \cdot E}{360}$ γ) $K_0(1 + r)^2 (1 + \frac{r \cdot n}{360})$

   δ) $\frac{Q}{4\pi\varepsilon\varepsilon_0} (\frac{1}{r_1^2} - \frac{1}{r_2^2})$ ε) $\frac{\pi}{8r} \cdot \frac{r_1 - r_2}{L} \cdot R^4$ στ) $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$

3. Αν `a=4`, `b=1`, και `c=7`, να υπολογιστεί η τιμή για καθεμιά από τις επόμενες παραστάσεις:
   α) `x=a<b && b<c`
   β) `x=b<c && (a<b || a>c)`
   γ) `x=(b>c || a>c) || c>a`
   δ) `x=! (a>b && (! b>c))`
   ε) `x=b>c || (a<c || b<c) && a<b`
4. Να κωδικοποιηθούν στη γλώσσα C οι παρακάτω προτάσεις:
   α) Το `x` μικρότερο του μηδενός **και** μεγαλύτερο του `y` **ή** το `x` μικρότερο του μηδενός **και** το `z` ίσο με το `r`.
   β) Το βάρος να μην είναι μικρότερο των 50 Kgr και το ύψος να μην είναι μεγαλύτερο των 1,90 m.
   γ) Ο `x` είναι θετικός ακέραιος μικρότερος του 152.
   δ) Η ηλικία να είναι μεταξύ 20 και 40 ετών.
5. Γράψτε ένα πρόγραμμα που να δέχεται το μήκος και το πλάτος ενός ορθογωνίου και να τυπώνει στην οθόνη το εμβαδόν και την περίμετρο αυτού.
6. Γράψτε πρόγραμμα που να δέχεται το μήκος , το πλάτος, το ύψος και τη μάζα ενός τούβλου και να τυπώνει στην οθόνη τον όγκο του και την πυκνότητά του.
7. Να γραφεί πρόγραμμα που να δέχεται την ηλικία ενός ατόμου σε έτη και να τη μετατρέπει σε ημέρες, ώρες και λεπτά.
8. Γράψτε πρόγραμμα που να δέχεται τη θερμοκρασία σε βαθμούς Fahrenheit και να τη μετατρέπει σε βαθμούς Celsius. Ο τύπος μετατροπής είναι:

$$C = (F - 32) \cdot \frac{5}{9}$$

9. Να γραφεί πρόγραμμα που να δέχεται τιμές για δύο ακέραιες μεταβλητές `x` και `y`, στη συνέχεια να εναλλάσσει τις τιμές των δύο μεταβλητών και να τυπώνει στην οθόνη τις τιμές των `x` και `y` μετά την εναλλαγή.
10. Να γραφεί πρόγραμμα που να δέχεται τιμές για τρεις ακέραιες μεταβλητές `x`, `y` και `z` και να εναλλάσσει κυκλικά τις τιμές τους. Τελικά να τυπώνει στην οθόνη τις τιμές των `x` ,`y` και `z` μετά την εναλλαγή.
11. Να γραφεί πρόγραμμα που να δέχεται την ακτίνα ενός κύκλου και να εκτυπώνει στην οθόνη το εμβαδόν του κύκλου καθώς και το εμβαδόν του τετραγώνου που είναι περιγεγραμμένο στον κύκλο.
12. Γράψτε πρόγραμμα που να δέχεται πέντε ακέραιες τιμές και να υπολογίζει και τυπώνει τη μέση τιμή τους.
13. Να γραφεί πρόγραμμα που να δέχεται έναν ακέραιο αριθμό και να βρίσκει αν είναι διαιρετός με το δύο το τρία και το τέσσερα ταυτόχρονα.
14. Γράψτε ένα πρόγραμμα που να δέχεται από το πληκτρολόγιο την τιμή κάποιου αντικειμένου, χωρίς ΦΠΑ και το συντελεστή ΦΠΑ. Κατόπιν να υπολογίζει και τυπώνει την τιμή με ΦΠΑ.

15. Ας υποθέσουμε ότι η ΔΕΗ χρεώνει 17.15 δρχ/kWh για το ημερήσιο ρεύμα και 10.61 δρχ/kWh για το νυχτερινό και 2605 δρχ πάγιο. Πάνω στο σύνολο των παραπάνω ο καταναλωτής πληρώνει ΦΠΑ 18%. Γράψτε πρόγραμμα που να δέχεται την ημερήσια και τη νυκτερινή κατανάλωση ρεύματος σε kWh και να βγάζει το λογαριασμό.

16. Αν συνδέσουμε δύο αντιστάσεις R_1 και R_2 σε σειρά, η συνολική αντίσταση είναι:

    $$R = R_1 + R_2$$

    Γράψτε πρόγραμμα που να δέχεται τις τιμές των R_1 και R_2 και να υπολογίζει και τυπώνει την R.

17. Αν συνδέσουμε δύο αντιστάσεις R_1 και R_2 παράλληλα, η συνολική αντίσταση είναι:

    $$R = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

    Γράψτε πρόγραμμα που να δέχεται τις τιμές των R_1 και R_2 και να υπολογίζει και τυπώνει την R.

ΚΕΦΑΛΑΙΟ 5
ΕΙΣΟΔΟΣ - ΕΞΟΔΟΣ

Η γλώσσα C διαθέτει αρκετές συναρτήσεις για διάβασμα και γράψιμο δεδομένων από το πρόγραμμα. Εδώ πρέπει να ξεκαθαρίσουμε ένα πράγμα: Όταν λέμε ότι το πρόγραμμα *διαβάζει* εννοούμε πάντα ότι *δέχεται* δεδομένα από κάποια περιφερειακή μονάδα εισόδου (πληκτρολόγιο, ποντίκι, δίσκο) και τα αποθηκεύει στην κεντρική μνήμη (RAM) του Η/Υ. Όταν λέμε ότι το πρόγραμμα *γράφει* εννοούμε ότι *στέλνει* δεδομένα από την κεντρική μνήμη προς κάποια περιφερειακή μονάδα εξόδου (οθόνη, εκτυπωτή, δίσκο).

Η επικοινωνία του προγράμματος με τις περιφερειακές συσκευές, γίνεται με τη χρήση *καναλιών ροής* (ή ρευμάτων-streams) τα οποία πρέπει να *ανοίξουν* πριν την επικοινωνία και να *κλείσουν* μετά από αυτή. Επειδή ένα μεγάλο ποσοστό επικοινωνίας των προγραμμάτων γίνεται με το πληκτρολόγιο και την οθόνη, η C με την έναρξη της εκτέλεσης οποιουδήποτε προγράμματος ανοίγει αυτόματα τα δύο κανάλια: προς την standard έξοδο (standard output ή stdout) δηλαδή την οθόνη και την standard είσοδο (standard input ή stdin) δηλαδή το πληκτρολόγιο. Τα δύο αυτά κανάλια κλείνουν επίσης αυτόματα με την έξοδο από το πρόγραμμα. Σ'αυτό το Κεφάλαιο θα εξετάσουμε τις συναρτήσεις εισόδου δεδομένων από το πληκτρολόγιο και εξόδου προς την οθόνη. Η επικοινωνία με το δίσκο (αρχεία) θα εξεταστεί στο Κεφάλαιο 11.

Οι κύριες συναρτήσεις εισόδου από το πληκτρολόγιο και εξόδου στην οθόνη είναι:

`getchar()` (`getche`,`getch`)	και	`putchar()`
`gets()`	και	`puts()`
`scanf()`	και	`printf()`

Το πρώτο ζευγάρι χρησιμεύει για είσοδο και έξοδο χαρακτήρων, το δεύτερο για αλφαριθμητικά και το τρίτο για *μορφοποιημένη* (formatted) είσοδο και έξοδο.

5.1 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΧΑΡΑΚΤΗΡΩΝ

Η συνάρτηση `getchar()` δεν παίρνει καμία παράμετρο και επιστρέφει μία ακέραια τιμή που είναι ο κωδικός ASCII του χαρακτήρα που διαβάζει από το πληκτρολόγιο. Μετά την εκτέλεση της εντολής:

```
c=getchar();
```

η μεταβλητή θα `c` περιέχει το χαρακτήρα που θα πληκτρολογήσουμε.

Η συνάρτηση `putchar()` εμφανίζει στην οθόνη το χαρακτήρα που έχει ως όρισμα. Έτσι μετά την εκτέλεση της εντολής:

```
putchar(c);
```

θα τυπωθεί στην οθόνη η τιμή της μεταβλητής χαρακτήρα `c`.

Σύμφωνα με τα προαναφερθέντα, το παρακάτω μικρό πρόγραμμα, διαβάζει ένα χαρακτήρα από το πληκτρολόγιο και στη συνέχεια τον εμφανίζει στην οθόνη:

```
#include <stdio.h>
main()
{
  char c;

  c=getchar();
  putchar(c);
}
```

Για να διαβάσει ένα χαρακτήρα η `getchar()` πρέπει μετά το χαρακτήρα να δώσουμε μία επαναφορά κεφαλής (ENTER). Αυτό πολλές φορές είναι ενοχλητικό. Έτσι μπορούμε να χρησιμοποιήσουμε τη συνάρτηση `getche()` η οποία διαβάζει ένα χαρακτήρα από το πληκτρολόγιο χωρίς να χρειάζεται να δώσουμε επαναφορά κεφαλής. Ο χαρακτήρας που διαβάζεται από την `getche()` εμφανίζεται (αντηχείται) στην οθόνη. Αν θέλουμε ο χαρακτήρας που διαβάζεται να μην εμφανίζεται στην οθόνη, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση `getch()` η οποία λειτουργεί όπως ακριβώς και η `getche()` με τη διαφορά ότι δεν εμφανίζεται στην οθόνη ο χαρακτήρας που πληκτρολογούμε.

5.2 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ

Η συνάρτηση `gets()` διαβάζει από το πληκτρολόγιο ένα αλφαριθμητικό και το τοποθετεί στη μεταβλητή που δέχεται ως όρισμα. Το όρισμα της συνάρτησης πρέπει να είναι τύπου *πίνακα χαρακτήρων*. Π.χ. μετά την εκτέλεση της εντολής:

```
gets(str);
```

η μεταβλητή `str` θα περιέχει το αλφαριθμητικό που θα πληκτρολογήσουμε.

Η συνάρτηση `puts()` γράφει στην οθόνη το αλφαριθμητικό που έχει ως όρισμά της και στη συνέχεια αλλάζει γραμμή. Έτσι π.χ. η εντολή:

```
puts("Καλημέρα");
```

γράφει στην οθόνη τη λέξη "Καλημέρα" και στη συνέχεια αλλάζει γραμμή.

Σύμφωνα με τα προαναφερθέντα, το παρακάτω μικρό πρόγραμμα, διαβάζει ένα αλφαριθμητικό από το πληκτρολόγιο και στη συνέχεια το εμφανίζει στην οθόνη:

```
main()
{
  char str[80];
  gets(str);
  puts(str);
}
```

5.3 ΜΟΡΦΟΠΟΙΗΜΕΝΗ ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ

Εκτός από τις απλές συναρτήσεις εισόδου-εξόδου που περιγράφηκαν προηγουμένως, η καθιερωμένη βιβλιοθήκη της C περιλαμβάνει δύο συναρτήσεις που πραγματοποιούν *μορφοποιημένη* είσοδο και έξοδο των βασικών τύπων δεδομένων. Ο όρος *μορφοποιημένη* αναφέρεται στο γεγονός ότι αυτές οι συναρτήσεις μπορούν να διαβάζουν και να γράφουν δεδομένα σε διάφορες μορφές τις οποίες μπορούμε εμείς να καθορίζουμε.

5.3.1 Η ΣΥΝΑΡΤΗΣΗ `printf()`

Η συνάρτηση `printf()` μετατρέπει τις τιμές των μεταβλητών σε χαρακτήρες και τους τυπώνει στη οθόνη. Η γενική μορφή της είναι:

printf(*αλφαριθμητικό_ελέγχου , λίστα_ορισμάτων****)***

Το *αλφαριθμητικό ελέγχου* αποτελείται από στοιχεία δύο τύπων:

(α) Κανονικούς χαρακτήρες οι οποίοι θα τυπωθούν από τη συνάρτηση στην οθόνη.

(β) *Καθοριστές μορφής* (format specifiers) που καθορίζουν τη *μορφή* με την οποία θα εμφανιστούν οι τιμές των ορισμάτων στην οθόνη.

Κάθε καθοριστής μορφής αρχίζει με το σύμβολο `%` και στη συνέχεια περιέχει ένα χαρακτήρα ο οποίος καθορίζει με ποια μορφή θα τυπωθεί στην οθόνη η τιμή του ορίσματος που αντιστοιχεί σ'αυτόν τον καθοριστή μορφής. Στον Πίνακα 5.1 φαίνονται οι καθοριστές μορφής της `printf()`. Ο αριθμός των ορισμάτων πρέπει να είναι ίσος με τον αριθμό των καθοριστών μορφής και μάλιστα πρέπει να έχουν την ίδια σειρά. Για παράδειγμα το παρακάτω πρόγραμμα:

```
/* printf1.c */
/* Παράδειγμα χρήσης της συνάρτησης printf() */
main()
{
  int   set=3;
  char  game='A';
  float time=27.25;

  printf("Το %do σετ του %c αγώνα ", set,game);
  printf("διήρκεσε %f λεπτά\n",time);
}
```

θα δώσει αποτέλεσμα:

```
Το 3ο σετ του Α αγώνα διήρκεσε 27.250000 λεπτά
```

Στο παραπάνω πρόγραμμα παρατηρούμε ότι στην οθόνη τυπώθηκε ο αριθμός 27.250000 με έξι ψηφία στα δεξιά της υποδιαστολής, αν και μόνο δύο απ'αυτά είναι σημαντικά. Προβλήματα σαν το προηγούμενο λύνονται, μιας και με τους καθοριστές μορφής της `printf()` μπορούμε να καθορίσουμε το *εύρος του πεδίου* που θα καταλάβει ένα εκτυπούμενο αποτέλεσμα, τον *αριθμό των κλασματικών ψηφίων* του αποτελέσματος καθώς και το αν το αποτέλεσμα θα είναι *στοιχισμένο* δεξιά ή αριστερά μέσα στο πεδίο.

Το *ελάχιστο εύρος* του πεδίου εκτύπωσης ενός αποτελέσματος καθορίζεται από το *καθοριστικό ελάχιστου εύρους* που είναι ένας ακέραιος που τοποθετείται μεταξύ του συμβόλου `%` και του χαρακτήρα του καθοριστή μορφής. Αν το εκτυπούμενο αποτέλεσμα είναι μεγαλύτερο από το ελάχιστο εύρος πεδίου, τότε το αποτέλεσμα θα τυπωθεί ολόκληρο αγνοώντας το ελάχιστο εύρος πεδίου, ενώ αν είναι μικρότερο θα χρησιμοποιηθούν κενά διαστήματα για να συμπληρωθεί. Αν θέλουμε αντί για κενά να τυπωθούν μηδενικά, πρέπει να τοποθετήσουμε το 0 (μηδέν) πριν από το καθοριστικό ελάχιστου εύρους.

Για να καθορίσουμε τον αριθμό των κλασματικών ψηφίων που θέλουμε να εμφανίζονται σε έναν αριθμό κινητής υποδιαστολής, τοποθετούμε μιαν *υποδιαστολή* (τελεία) μετά το καθοριστικό ελάχιστου εύρους του πεδίου και στη συνέχεια έναν ακέραιο που εκφράζει τον αριθμό των κλασματικών ψηφίων που θέλουμε να εμφανίζονται. Για παράδειγμα το `%10.4f` θα εμφανίσει έναν αριθμό κινητής υποδιαστολής πλάτους τουλάχιστον 10 χαρακτήρων με τέσσερα κλασματικά ψηφία.

Όταν εφαρμόζουμε μία μορφή σαν την παραπάνω σε αλφαριθμητικά, ο αριθμός που ακολουθεί την τελεία καθορίζει το μέγιστο μήκος του πεδίου. Για παράδειγμα, το `%5.7s` θα εμφανίσει ένα αλφαριθμητικό μήκους τουλάχιστον πέντε χαρακτήρων και όχι μεγαλύτερο των 7 χαρακτήρων. Αν το αλφαριθμητικό είναι μεγαλύτερο από το μέγιστο μήκος πεδίου, θα αποκοπούν στην εκτύπωση οι επιπλέον χαρακτήρες που βρίσκονται στο τέλος του.

Προφανώς για την εκτύπωση ακεραίων αριθμών, δεν έχει νόημα ο αριθμός που καθορίζει τα κλασματικά ψηφία και ως εκ τούτου αγνοείται.

Εξ ορισμού, η έξοδος είναι *ευθυγραμμισμένη δεξιά* μέσα στο πεδίο. Μπορούμε να στοιχίσουμε την έξοδο αριστερά τοποθετώντας το σύμβολο πλην (−) μετά το `%`. Για παράδειγμα, το `%-10.2f` θα στοιχίσει στα αριστερά έναν αριθμό κινητής υποδιαστολής με δύο κλασματικά ψηφία σ'ένα πεδίο δέκα χαρακτήρων.

Για εφαρμογή των παραπάνω ας δούμε το παρακάτω πρόγραμμα:

```
/* printf2.c */
/* Εκτύπωση κλασματικών αριθμών με χρήση της printf() */
main()
{
  float a,b,c,d,e,f;
  a=3.0;    b=12.5;
  c=523.3;  d=300.0;
  e=1200.5; f=5300.3;

  printf("|%.1f|%.1f|%.1f|\n",a,b,c);
  printf("|%.1f|%.1f|%.1f|\n\n",d,e,f);
  printf("|%8.1f|%8.1f|%8.1f|\n",a,b,c);
  printf("|%8.1f|%8.1f|%8.1f|\n\n",d,e,f);
  printf("|%-8.1f|%-8.1f|%-8.1f|\n",a,b,c);
  printf("|%-8.1f|%-8.1f|%-8.1f|\n\n",d,e,f);
}
```

Το αποτέλεσμα που παίρνουμε αν τρέξουμε το παραπάνω πρόγραμμα είναι αυτό που φαίνεται παρακάτω. Προσπαθήστε να ερμηνεύσετε αυτό το αποτέλεσμα.

```
|3.0|12.5|523.3|
|300.0|1200.5|5300.3|

|     3.0|    12.5|   523.3|
|   300.0|  1200.5|  5300.3|

|3.0     |12.5    |523.3   |
|300.0   |1200.5  |5300.3  |
```

5.3.2 Η ΣΥΝΑΡΤΗΣΗ `scanf()`

Η συνάρτηση `scanf()` χρησιμοποιείται για να διαβάζει τους χαρακτήρες που πληκτρολογεί ο χρήστης. Στη συνέχεια τους ερμηνεύει κατάλληλα και τους καταχωρεί ως τιμές στις μεταβλητές. Η γενική μορφή της είναι:

scanf(*αλφαριθμητικό_ελέγχου , λίστα_ορισμάτων****)***

Το *αλφαριθμητικό ελέγχου* στη `scanf()` αποτελείται από στοιχεία τριών τύπων:

(α) Καθοριστές μορφής.

(β) Χαρακτήρες κενών.

(γ) Χαρακτήρες μη-κενών.

Κάθε καθοριστής μορφής αρχίζει με το σύμβολο `%` και στη συνέχεια περιέχει ένα χαρακτήρα ο οποίος πληροφορεί τη `scanf()` για τον τύπο στον οποίο θα μετατρέψει τα δεδομένα που θα διαβάσει. Οι καθοριστές μορφής για την `scanf()` φαίνονται στον Πίνακα 5.1.

Ένας χαρακτήρας κενού μέσα στο αλφαριθμητικό ελέγχου υποχρεώνει την `scanf()` να αγνοήσει έναν ή περισσότερους χαρακτήρες κενού που δίνονται ως είσοδος. Ένας χαρακτήρας κενού είναι είτε ένα κενό διάστημα, είτε ένας στηλογνώμονας (tab), ή μία νέα γραμμή. Ουσιαστικά ένας χαρακτήρας λευκού κενού μέσα στο αλφαριθμητικό ελέγχου υποχρεώνει την `scanf()` να διαβάσει, αλλά όχι και να αποθηκεύσει, οποιονδήποτε αριθμό χαρακτήρων λευκού κενού, μέχρι να συναντηθεί ο πρώτος χαρακτήρας μη-κενού.

Ένας χαρακτήρας μη-κενού μέσα στο αλφαριθμητικό ελέγχου υποχρεώνει την `scanf()` να διαβάσει και να απορρίψει κάποιο συγκεκριμένο χαρακτήρα. Για παράδειγμα το αλφαριθμητικό ελέγχου "`%d,%d`" υποχρεώνει την `scanf()` να διαβάσει πρώτα έναν ακέραιο, στη συνέχεια να διαβάσει και να απορρίψει ένα κόμμα και τέλος να διαβάσει έναν άλλον ακέραιο.

Όλες οι μεταβλητές που χρησιμοποιούνται για να δέχονται τιμές μέσω της `scanf()` πρέπει να περνάνε μέσω των *διευθύνσεών* τους. Για να αναφερθούμε στη διεύθυνση μίας μεταβλητής τύπου χαρακτήρα ή αριθμού, πρέπει μπροστά από το όνομα της μεταβλητής να τοποθετούμε το σύμβολο `&` που λέγεται *τελεστής διεύθυνσης*. Όμως για να αναφερθούμε στη διεύθυνση μίας αλφαριθμητικής μεταβλητής, δεν πρέπει να τοποθετούμε τον τελεστή διεύθυνσης `&` μπροστά από το όνομα της μεταβλητής.

Μορφή	***Καθοριστές Μορφής***	
	printf()	**scanf()**
χαρακτήρας	%c	%c
προσ/μένος δεκαδικός ακέραιος	%d ή %i	%d
κινητής υποδιαστολής (εκθετική μορφή)	%e	%e ή %f
κινητής υποδιαστολής (τυποποιημένη μορφή)	%f	%e ή %f
κινητής υποδιαστολής (%e ή %f όποιο είναι πιο μικρό)	%g	
απρόσημος οκταδικός ακέραιος	%o	%o
αλφαριθμητικό	%s	%s
απρόσημος δεκαδικός ακέραιος	%u	%u
απρόσημος δεκαεξαδικός ακέραιος	%x	%x
δείκτης	%p	%p
long ακέραιοι	%ld,%li,%lo,%lu,%lx	%D,%U,%O,%X ή %ld κ.λ.π.
short ακέραιοι	%hd,%hi,%ho,%hu,%hx	%hd,%hi,%ho,%hu,%hx

Πίνακας 5.1: *Οι καθοριστές μορφής των* `scanf()` *και* `printf()`

Ας υποθέσουμε ότι σε ένα πρόγραμμα έχουμε δηλώσει μία μεταβλητή ακεραίου τύπου με όνομα num και της έχουμε δώσει την τιμή 2. Αν αργότερα το πρόγραμμα αναφερθεί στο όνομα της μεταβλητής (num) ο μεταγλωττιστής θα επιστρέψει την *τιμή* που είναι αποθηκευμένη στη μεταβλητή, δηλαδή 2. Αν όμως αναφερθούμε στο όνομα της μεταβλητής βάζοντας μπροστά το & (&num) ο μεταγλωττιστής θα επιστρέψει τη *διεύθυνση* στην οποία είναι αποθηκευμένη η num. Το παρακάτω πρόγραμμα δείχνει αυτή τη λειτουργία:

```
/* testaddr.c */
/* Εκτύπωση της διεύθυνσης μίας μεταβλητής */
main()
{
  int num=2;

  printf("\nΤιμή=%d   Διεύθυνση=%ld",num,&num);
  getch();
}
```

Το αποτέλεσμα είναι:

```
Τιμή=2   Διεύθυνση=196562
```

Είναι σίγουρο ότι η διεύθυνση θα είναι διαφορετική σε διαφορετικό Η/Υ.

5.4 ΑΣΚΗΣΕΙΣ

1. Πληκτρολογήστε και εκτελέστε τα παρακάτω προγράμματα. Ερμηνεύστε τα αποτελέσματα:

```
/* printf3.c */
/* Χρήση της συνάρτησης printf() */
main()
{
  char c='a';
  int i=1234;
  float x=-123.456789;

  printf("|%c|%s|%d|%f|%e|\n",c,"hello",i,x,x);
  printf("|%2c|%8s|%5d|%12f|%-13e|\n",c,"hello",i,x,x);
  printf("|%-2c|%-8s|%-5d|%-12f|%-13e|\n",c,"hello",i,x,x);
  printf("|%0c|%6s|%3d|%10f|%11e|\n",c,"hello",i,x,x);
}

/* ************************************************* */

/* printf4.c */
/* Χρήση της συνάρτησης printf() */
main()
{
  char c='a';
  int i=1234;
  float x=-123.456789;

  printf("|%13.2f|%13.2e|\n",x,x);
  printf("|%13.0f|%13.0e|\n",x,x);
  printf("|%13.26f|\n",x);
  printf("|%40.32f|\n",x);
  printf("|%6.1c|%6.1d|\n",c,i);
  printf("|%8.35s|%8.1s|%8.0s|\n","hello","hello","hello");
}

/* ************************************************* */

/* printf5.c */
/* Χρήση της συνάρτησης printf() */
main()
{
  char c='a';
  int i=1234;
  float x=-123.456789;

  printf("|%c|%d|%f|\n",c,c,c);
  printf("|%s|%d|%f|\n","hello","hello","hello");
  printf("|%c|%d|%f|\n",i,i,i);
  printf("|%f|%d|\n",x,x);
}

/* ************************************************* */
```

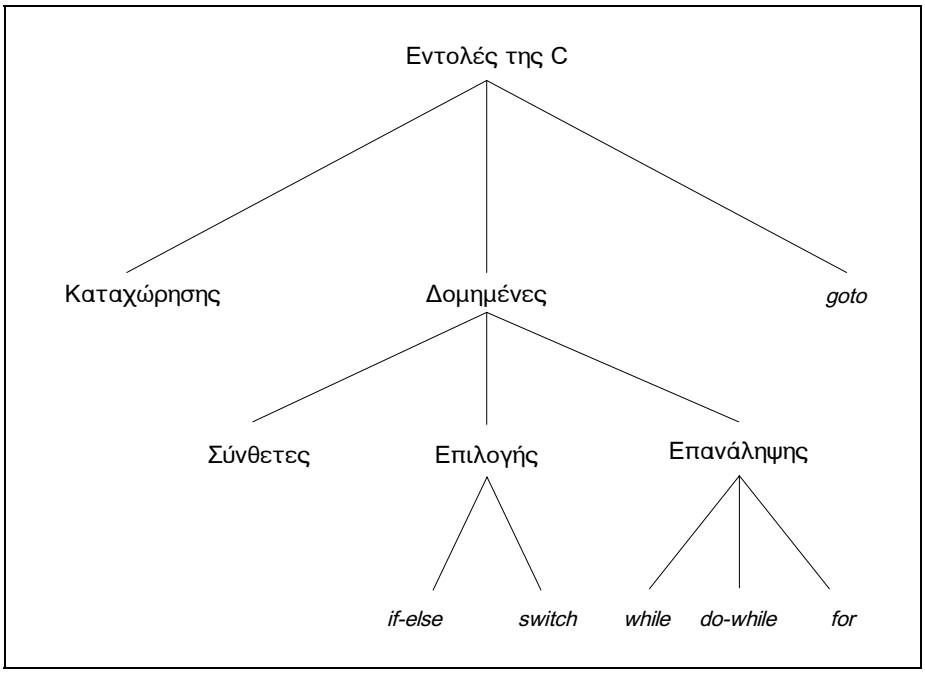
```
/* scanf1.c */
/* Χρήση της συνάρτησης scanf() */
main()
{
  int set;
  char game;
  float time;

  printf("Δώσε αριθμό του σετ, γράμμα αγώνα και ώρα: ");
  scanf("%d %c %f", &set, &game, &time);
  printf("Το %do σετ του %c αγώνα διήρκεσε ",set,game);
  printf("%1f λεπτά\n",time);
}
```

2. Γράψτε ένα πρόγραμμα το οποίο να δέχεται από το πληκτρολόγιο το όνομά σας και την ηλικία σας και στη συνέχεια να τυπώνει στην οθόνη το μήνυμα:

```
Σε λένε "............" και είσαι ...... ετών.
```

ΚΕΦΑΛΑΙΟ 6
ΕΝΤΟΛΕΣ ΕΠΙΛΟΓΗΣ ΚΑΙ ΕΠΑΝΑΛΗΨΗΣ

Στο Σχήμα 6.1 φαίνονται οι διάφοροι τύποι των εντολών της C.



Σχήμα 6.1: *Οι εντολές της C*

Μία εντολή *καταχώρησης* ή *απόδοσης* τιμής, όπως έχουμε δει στις § 4.3 και 4.4 είναι της μορφής:

```
μεταβλητή = παράσταση;
```

Σ'αυτό το κεφάλαιο θα εξετάσουμε τις *δομημένες* εντολές.

Γενικά οι εντολές επιλογής και επανάληψης, λέγονται *εντολές ελέγχου ροής* διότι μας επιτρέπουν να καθορίζουμε τη σειρά εκτέλεσης των εντολών του προγράμματος καθώς και το εάν κάποιες από αυτές θα εκτελεστούν ή όχι. Αν δε χρησιμοποιηθούν οι εντολές ελέγχου ροής σ'ένα πρόγραμμα, όλες οι εντολές του προγράμματος εκτελούνται σειριακά και υποχρεωτικά, χωρίς να υπάρχει δυνατότητα παράλειψης κάποιων εξ αυτών.

Έτσι, όπως θα δούμε παρακάτω, οι εντολές επιλογής μας επιτρέπουν να καθορίζουμε εάν και πότε θα εκτελεστούν κάποια τμήματα του προγράμματός μας και οι εντολές επανάληψης μας επιτρέπουν να επαναλαμβάνουμε κατά βούληση την εκτέλεση κάποιων τμημάτων.

Όμως καταρχάς θα δούμε την έννοια της σύνθετης εντολής.

6.1 Η ΣΥΝΘΕΤΗ ΕΝΤΟΛΗ

Όπως έχουμε αναφέρει στην § 4.4, μία παράσταση όπως η `x=0`, η `i++`, ή η `printf(...)` γίνεται *εντολή* (statement) όταν ακολουθείται από ελληνικό ερωτηματικό (`;`), όπως στις:

```
x=0;
i++;
printf(...);
```

Στη C, το ελληνικό ερωτηματικό είναι σημείο τερματισμού εντολών, κι όχι διαχωριστικό όπως είναι σε άλλες γλώσσες, π.χ. στην Pascal.

Στην περίπτωση που θέλουμε περισσότερες από μία εντολές να είναι συντακτικά ισοδύναμες με μία μόνο εντολή (αυτό θα γίνει περισσότερο κατανοητό στην εξέταση των εντολών `if-else`, `switch`, `while`, `do`, `for` που θα γίνει παρακάτω) τότε πρέπει να *ομαδοποιήσουμε* αυτές τις εντολές σε μία *σύνθετη εντολή* ή μπλοκ, χρησιμοποιώντας τα άγκιστρα `{` και `}`.

Η σύνθετη εντολή λαμβάνεται από το μεταγλωττιστή ολόκληρη (το `{`, οι εντολές που περιέχει και το `}`) ως μία εντολή της C, με το ελληνικό ερωτηματικό αντιληπτό. Έτσι δε χρειάζεται να τοποθετούμε ελληνικό ερωτηματικό μετά το άγκιστρο κλεισίματος (`}`).

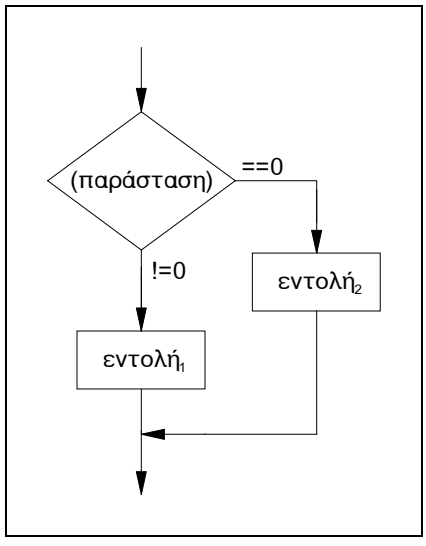
6.2 Η ΕΝΤΟΛΗ ΕΠΙΛΟΓΗΣ `if-else`

Η εντολή `if-else` χρησιμοποιείται για την υπό συνθήκη εκτέλεση εντολών. Η γενική μορφή της είναι:

```
if (παράσταση)
  εντολή1
else
  εντολή2
```

όπου το τμήμα του `else` είναι *προαιρετικό*.

Αν η *παράσταση* είναι αληθής (διαφορετική του μηδενός) τότε εκτελείται η *εντολή*$_1$. Αν είναι ψευδής (ίση με μηδέν) και αν υπάρχει το τμήμα `else`, εκτελείται η *εντολή*$_2$, όπως φαίνεται στο Σχήμα 6.2. Η *εντολή*$_1$ και *εντολή*$_2$ μπορούν να είναι τόσο απλές όσο και σύνθετες εντολές.



Σχήμα 6.2: *Η εντολή if-else*

Έτσι, στο τμήμα κώδικα που ακολουθεί συμβαίνουν τα εξής: όταν η μεταβλητή `a` είναι διάφορη του μηδενός, υπολογίζεται το πηλίκο `-b/a` και τυπώνεται στην οθόνη. Στην περίπτωση που η μεταβλητή `a` είναι ίση με το μηδέν, τότε δεν υπολογίζεται τίποτα και απλώς εμφανίζεται ένα μήνυμα στην οθόνη. Όπως βλέπουμε μετά το `if`

ακολουθεί μία σύνθετη εντολή. Με την ευκαιρία, προσέξτε το ελληνικό ερωτηματικό πριν από το `else`, είναι απαραίτητο !!

```
if (a!=0)
{
  x=-b/a;
  printf("x=%6.2f",x);
}
else printf("Αδύνατη διαίρεση");
```

Ας δούμε τώρα ένα απλό πρόγραμμα το οποίο βρίσκει το μεγαλύτερο από δύο πραγματικούς αριθμούς που πληκτρολογεί ο χρήστης. Αρχικά υποθέτουμε ότι μεγαλύτερος είναι ο πρώτος αριθμός και τον καταχωρούμε στη μεταβλητή `max`, στη συνέχεια συγκρίνουμε την τιμή της `max` με το δεύτερο αριθμό, αν ο δεύτερος αριθμός είναι μεγαλύτερος από την τιμή της `max`, τότε καταχωρούμε το δεύτερο αριθμό στη μεταβλητή `max`. Τέλος τυπώνουμε την τιμή της `max`. Με την ίδια λογική μπορούμε να υπολογίσουμε το μεγαλύτερο περισσοτέρων των δύο αριθμών:

```
/* max1.c */
/* Χρήση της if για εύρεση μεγαλύτερου δύο αριθμών */
main()
{
  float x,y,max;

  puts("\nΔώσε δύο αριθμούς: ");
  scanf("%f %f", &x,&y);
  max=x;
  if (y>x) max=y;
  printf("\nΜεγαλύτερος είναι ο %.2f",max);
}
```

Στη συνέχεια παρουσιάζεται ένα πρόγραμμα που δέχεται έναν ακέραιο αριθμό και τυπώνει την ημέρα της εβδομάδας που αντιστοιχεί σ'αυτόν (το 1 αντιστοιχεί στην Κυριακή και το 7 στο Σάββατο). Στην περίπτωση που ο αριθμός είναι μεγαλύτερος του 7 ή μικρότερος του 1, τυπώνεται κατάλληλο μήνυμα:

```
/* week1.c */
/* Εκτύπωση ημέρας της εβδομάδας που αντιστοιχεί σε έναν ακέραιο */
main()
{
  int i;
  printf("\nΔώσε έναν ακέραιο από 1 έως 7: ");
  scanf("%d",&i);
  if (i==1) puts("Κυριακή");
  if (i==2) puts("Δευτέρα");
  if (i==3) puts("Τρίτη");
  if (i==4) puts("Τετάρτη");
  if (i==5) puts("Πέμπτη");
  if (i==6) puts("Παρασκευή");
  if (i==7) puts("Σάββατο");
  if (i>7 || i<1) puts("Ακυρος αριθμός");
}
```

6.2.1 ΕΝΘΕΤΕΣ ΕΝΤΟΛΕΣ `if`

Φυσικά σε μία δομή `if-else` τόσο η *εντολή*$_1$ όσο και η *εντολή*$_2$ μπορούν να είναι εντολές `if`.

— Η απλούστερη περίπτωση είναι όταν η *εντολή*$_2$ είναι εντολή `if-else` . Τότε λέμε ότι έχουμε μία δομή `if-else-if`, όπως παρακάτω:

```
if(παράσταση1)                          if(παράσταση1)
  εντολή1                                 εντολή1
else                                    else
  if(παράσταση2)                        if(παράσταση2)
    εντολή2          ή ισοδύναμα          εντολή2
  else                                  else
       ...                                ...
        else                            else
          if(παράσταση_n)               if(παράσταση_n)
            εντολή_n                      εντολή_n
          else                          else
            εντολή_n+1                    εντολή_n+1
```

Ο μεταγλωττιστής υπολογίζει τις *παραστάσεις* με τη σειρά και αν κάποια *παράσταση* βρεθεί αληθής (διάφορη του μηδενός) εκτελείται η *εντολή* που σχετίζεται μ'αυτήν την παράσταση και η όλη αλυσίδα τερματίζεται. Αν καμία παράσταση δεν είναι αληθής, τότε εκτελείται η *εντολή* που σχετίζεται με το τελικό `else`. Έτσι λοιπόν το τελευταίο `else` χειρίζεται την περίπτωση "τίποτα από τα παραπάνω" (που λέγεται και "εξ ορισμού" περίπτωση). Αν δεν υπάρχει συγκεκριμένη ενέργεια για την εξ ορισμού περίπτωση, τότε το τελευταίο `else` μπορεί να παραλειφθεί. Μπορούμε να χρησιμοποιήσουμε την παραπάνω δομή για να βελτιώσουμε το πρόγραμμα `week1.c` που είδαμε παραπάνω. Η αρχική έκδοση του προγράμματος, εκτελεί με τη σειρά όλες τις εντολές `if` ακόμα κι αν κάποια από τις προηγούμενες έχει προκύψει αληθής και κατά συνέπεια δεν υπάρχει περίπτωση να αληθεύει καμία επόμενη `if`. Είναι προφανές τώρα τι πρέπει να κάνουμε για να βελτιώσουμε το πρόγραμμα (κάντε το !!).

— Μία κάπως περισσότερο πολύπλοκη περίπτωση είναι η παρακάτω:

```
if (παράσταση1)
  if (παράσταση2)
    εντολή1
  else
    εντολή2
```

Σ'αυτή την περίπτωση σε ποιο `if` αντιστοιχεί το `else` ; Γενικά ισχύει ο επόμενος κανόνας:

Ένα `else` αντιστοιχεί στο τελευταίο `if` που δεν έχει το δικό του `else`.

Έτσι στην παραπάνω περίπτωση το `else` αντιστοιχεί στο δεύτερο `if`.

Αν θέλουμε όμως κάποιο `else` να αντιστοιχεί στο προηγούμενο `if` παρά σ'αυτό που θα αντιστοιχούσε πραγματικά, πρέπει να χρησιμοποιήσουμε άγκιστρα για να περικλείσουμε το ενδιάμεσο `if`. Έτσι αν θέλαμε στην παραπάνω περίπτωση το `else` να αντιστοιχεί στο πρώτο `if` και όχι στο δεύτερο, θα έπρεπε να γράψουμε:

```
if (παράσταση1)
{
  if (παράσταση2)
    εντολή1
}
else
  εντολή2
```

6.2.2 Ο ΤΕΛΕΣΤΗΣ ΣΥΝΘΗΚΗΣ `?:`

Ο τελεστής `?:` είναι ένας *τριαδικός τελεστής* (με τρεις τελεστέους) που χρησιμοποιείται στις παραστάσεις υπό συνθήκη. Μία *παράσταση υπό συνθήκη* έχει την παρακάτω μορφή:

παράσταση$_1$ **?** *παράσταση*$_2$ **:** *παράσταση*$_3$

Υπολογίζεται πρώτα η *παράσταση*$_1$. Αν είναι μη-μηδενική (αληθής), υπολογίζεται η *παράσταση*$_2$ και αυτή είναι η τιμή της παράστασης υπό συνθήκη. Διαφορετικά, υπολογίζεται η *παράσταση*$_3$ και αυτή είναι η τιμή της παράστασης υπό συνθήκη. Μόνο μία από τις *παράσταση*$_2$ και *παράσταση*$_3$ υπολογίζεται.

Οι παραστάσεις υπό συνθήκη χρησιμοποιούνται για να αντικαθιστούν εντολές `if-else` της γενικής μορφής:

```
if (παράσταση1)
  μεταβλητή=παράσταση2;
else
  μεταβλητή=παράσταση3;
```

ως εξής:

```
μεταβλητή=παράσταση1 ? παράσταση2 : παράσταση3;
```

Για παράδειγμα το παρακάτω τμήμα κώδικα που καταχωρεί στην `max` το μεγαλύτερο των `a` και `b`:

```
if (a>b) max=a; else max=b;
```

μπορεί να γραφεί ως:

```
max=a>b?a:b;
```

6.3 Η ΕΝΤΟΛΗ ΕΠΙΛΟΓΗΣ `switch`

Όπως είδαμε προηγουμένως η δομή `if-else-if` επιτρέπει την επιλογή μίας κατάστασης από ένα σύνολο πολλών καταστάσεων με βάση την τιμή κάποιας παράστασης.

Η γλώσσα C όμως διαθέτει μία *πολυκλαδική* εντολή αποφάσεων, την `switch`, η οποία ελέγχει αν η τιμή μίας παράστασης ταυτίζεται με μία τιμή από ένα σύνολο *σταθερών* ακεραίων τιμών, ή χαρακτήρων και διακλαδώνεται ανάλογα. Το συντακτικό της εντολής `switch` είναι το ακόλουθο:

```
switch (παράσταση)
{
   case σταθερά1 : εντολές1
   case σταθερά2 : εντολές2
            ...
   case σταθεράn : εντολέςn
   default       : εντολές
}
```

Η *παράσταση* που ακολουθεί τη λέξη `switch` μπορεί να είναι:

(i) Μία ακέραια σταθερά όπως 1, 2, 3 ...

(ii) Μία συνάρτηση που επιστρέφει ακέραιο αριθμό

(iii) Μία αριθμητική παράσταση, που δίνει ως τιμή ακέραιο αριθμό

(iv) Ένας χαρακτήρας

Κάθε *περίπτωση* (`case`) ακολουθείται από μία σταθερά που μπορεί να είναι μία ακέραια σταθερά (π.χ. 1), ένας χαρακτήρας (π.χ. 'Β'), ή μία ακέραια αριθμητική παράσταση (π.χ. `4*3+8`). *Δεν επιτρέπεται περισσότερες από μία* `case` *να έχουν την ίδια σταθερά*.

Αρχικά υπολογίζεται η τιμή της *παράστασης* και κατόπιν ερευνώνται ένα προς ένα τα `case` για να βρεθεί όμοια τιμή. Εφόσον βρεθεί, τότε η εκτέλεση συνεχίζεται από την εντολή που ακολουθεί την `case`. Διαφορετικά η εκτέλεση συνεχίζεται από την πρώτη εντολή μετά το `default`, αν υπάρχει. Αν καμία σταθερά δε συμφωνεί με την τιμή της παράστασης και δεν υπάρχει `default`, τότε δεν πραγματοποιείται καμία ενέργεια. Το `default` μπορεί να τεθεί οπουδήποτε ανάμεσα στις `case`, όχι μόνο στο τέλος.

Χρησιμοποιώντας την εντολή `switch` το πρόγραμμα `week1.c` της § 6.2 γράφεται όπως φαίνεται παρακάτω. Στο πρόγραμμα αυτό ας προσέξουμε τη χρήση της εντολής `break`. Η εντολή `break` χρησιμοποιείται εδώ για να προκαλέσει άμεση έξοδο από την `switch`. Επειδή οι `case` εξυπηρετούν απλώς σαν σημεία όπου μεταφέρεται ο έλεγχος του προγράμματος (ετικέτες), μετά την εκτέλεση του κώδικα κάποιας `case` το πρόγραμμα *συνεχίζει* με τον κώδικα της επόμενης `case` κ.ο.κ. έως ότου φτάσει σε κάποια `break` ή στο τέλος της εντολής `switch`.

```
/* week2.c */
/* Εκτύπωση ημέρας της εβδομάδας που αντιστοιχεί σε έναν ακέραιο */
main()
{
  int i;

  printf("\nΔώσε έναν ακέραιο από 1 έως 7: ");
  scanf("%d",&i);
  switch (i)
  {
    case 1 : puts("Κυριακή");   break;
    case 2 : puts("Δευτέρα");   break;
    case 3 : puts("Τρίτη");     break;
```

```
    case 4 : puts("Τετάρτη");   break;
    case 5 : puts("Πέμπτη");    break;
    case 6 : puts("Παρασκευή"); break;
    case 7 : puts("Σάββατο");   break;
    default: puts("Ακυρος αριθμός");
  }
}
```

Ωστόσο μπορούμε να εκμεταλλευτούμε, σε μερικές περιπτώσεις, τη συνέχιση της εκτέλεσης του κώδικα της επόμενης `case`, όπως φαίνεται στο παρακάτω πρόγραμμα, όπου υπολογίζεται με έναν "έξυπνο" τρόπο το παραγοντικό ενός αριθμού:

```
/* fact1.c */
/* Υπολογισμός παραγοντικού ενός ακεραίου στο διάστημα 1..8 */
main()
{
  int number;
  unsigned int factorial=1;

  printf("\nΔώσε ακέραιο: ");
  scanf("%d",&number);
  switch (number)
  {
    case 8 : factorial  =8;
    case 7 : factorial *=7;
    case 6 : factorial *=6;
    case 5 : factorial *=5;
    case 4 : factorial *=4;
    case 3 : factorial *=3;
    case 2 : factorial *=2;
    case 1 :
    case 0 : printf("Το παραγοντικό του %d είναι %u\n",
                    number,factorial);
             break;
    default: printf("Αδύνατος ο υπολογισμός του παραγοντικού του %d\n",
                    number);
  }
}
```

Επίσης από το παραπάνω πρόγραμμα φαίνεται πώς μπορούμε να αντιστοιχήσουμε σε περισσότερες από μία `case` την ίδια ενέργεια (το παραγοντικό τόσο του 0 όσο και του 1 είναι 1). Στο παρακάτω τμήμα κώδικα αυτό φαίνεται καλύτερα. Θέλουμε να χαρακτηρίσουμε την επίδοση ενός μαθητή (A, B, C, D) ανάλογα με το βαθμό του (9-10, 7-8, 5-6, 0-4):

```
switch (score)
{
  case 10 :
  case 9  : printf("\nA"); break;
  case 8  :
  case 7  : printf("\nB"); break;
  case 6  :
  case 5  : printf("\nC"); break;
```

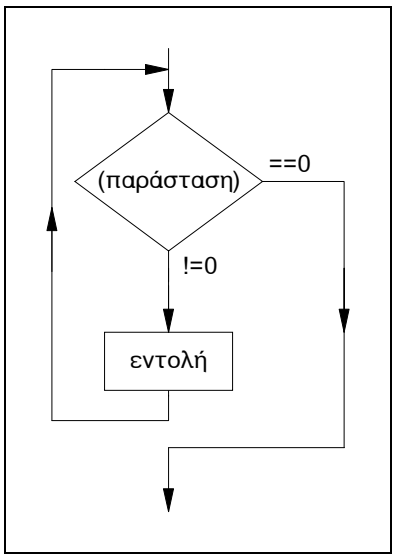
```
  case 4  :
  case 3  :
  case 2  :
  case 1  :
  case 0  : printf("\nD"); break;
  default : printf("\nΑκυρος βαθμός");
}
```

6.4 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `while`

Η εντολή `while` είναι μία *εντολή επανάληψης* και χρησιμοποιείται όταν θέλουμε να εκτελείται συνέχεια κάποια ενέργεια *όσο* μία συνθήκη είναι αληθής. Η εντολή `while` έχει την ακόλουθη μορφή:

while **(***παράσταση***)**
  εντολή

Η *εντολή* που ακολουθεί το `while` μπορεί να είναι οποιαδήποτε απλή ή σύνθετη εντολή. Αρχικά υπολογίζεται η *παράσταση* και αν είναι μη-μηδενική, εκτελείται η *εντολή* και η *παράσταση* υπολογίζεται ξανά. Ο βρόχος συνεχίζεται μέχρι να γίνει η *παράσταση* μηδέν, οπότε η εκτέλεση συνεχίζεται από το σημείο μετά την *εντολή*, όπως φαίνεται στο Σχήμα 6.3.



Σχήμα 6.3: *Η εντολή while*

Είναι φανερό ότι αν η τιμή της *παράστασης* είναι εξαρχής μηδέν, η *εντολή* δε θα εκτελεστεί ούτε μία φορά. Επίσης πρέπει να προβλέπεται η μεταβολή της τιμής της *παράστασης*, από την *εντολή*, διαφορετικά ο βρόχος θα εκτελείται συνέχεια (black operational hole).

Το ακόλουθο πρόγραμμα χρησιμοποιεί ένα βρόχο `while` για να τυπώσει τους αριθμούς 0 έως 9 όπως επίσης και το τρέχον άθροισμα:

```
/* wloop.c */
/* Εκτύπωση των αριθμών από 0 ως 9 */
/* και του τρέχοντος αθροίσματος   */
main()
{
  int count=0;
  int total=0;

  while(count<10)
    printf("Μετρητής=%d, Αθροισμα=%d\n", count++, total+=count);
}
```

Το αποτέλεσμα του προγράμματος είναι:

```
Μετρητής=0, Αθροισμα=0
Μετρητής=1, Αθροισμα=1
Μετρητής=2, Αθροισμα=3
Μετρητής=3, Αθροισμα=6
Μετρητής=4, Αθροισμα=10
Μετρητής=5, Αθροισμα=15
Μετρητής=6, Αθροισμα=21
Μετρητής=7, Αθροισμα=28
Μετρητής=8, Αθροισμα=36
Μετρητής=9, Αθροισμα=45
```

Ο βρόχος `while` είναι ο πλέον κατάλληλος σε περιπτώσεις που δεν είναι γνωστός εκ των προτέρων ο αριθμός των επαναλήψεων και ο βρόχος μπορεί να τερματιστεί απρόσμενα από συνθήκες που ανακύπτουν μέσα στο βρόχο. Ως παράδειγμα ας δούμε το παρακάτω πρόγραμμα:

```
/* charcnt.c */
/* Απαρίθμηση χαρακτήρων πρότασης */
main()
{
  int count=0;

  printf("Γράψε μια πρόταση (για τέλος δώσε ENTER):\n");
  while(getche() != '\r') count++ ;
  printf("\nO αριθμός των χαρακτήρων είναι %d\n",count);
}
```

Αυτό το πρόγραμμα μας προσκαλεί να πληκτρολογήσουμε μία πρόταση. Καθώς εισάγουμε κάθε χαρακτήρα το πρόγραμμα απαριθμεί τους χαρακτήρες που πληκτρολογούμε και όταν πατάμε ENTER τυπώνει το σύνολο. Προσέξτε το χαρακτήρα '`\r`' που σημαίνει επαναφορά κεφαλής (βλέπε § 3.3.2).

Η *εντολή* στο βρόχο `while` μπορεί να είναι και η *κενή εντολή*. Η *κενή εντολή* είναι απλώς ένα ελληνικό ερωτηματικό (`;`), η εκτέλεση αυτής της εντολής δεν έχει κανένα άλλο αποτέλεσμα εκτός της *καθυστέρησης*. Για παράδειγμα ο παρακάτω βρόχος θα εκτελείται συνέχεια μέχρι να πληκτρολογήσουμε το χαρακτήρα 'A':

```
while((ch=getche())!='A') ;
```

Πριν αφήσουμε το βρόχο `while` θα δούμε δύο παραδείγματα. Το πρώτο υπολογίζει το παραγοντικό ενός αριθμού (συγκρίνετε με το `fact1.c` της § 6.3):

```
/* fact2.c */
/* Υπολογισμός παραγοντικού ενός μη-αρνητικού ακεραίου */
main()
{
  unsigned long int factorial;
  int number;

  printf("\nΔώσε ακέραιο: ");
  scanf("%d",&number);
  factorial=1;
  while (number>1) factorial *=number--;
  printf("Το παραγοντικό είναι: %lu\n",factorial);
}
```

Το δεύτερο παράδειγμα υπολογίζει την τιμή του αθροίσματος
`S=1+1/2+1/3+...+1/n`
για δοσμένο κάθε φορά `n`.

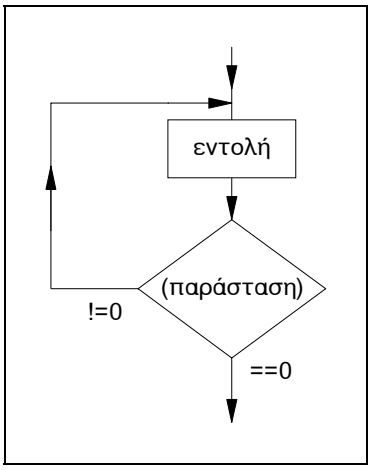
```
/* series1.c */
/* Υπολογισμός του αθροίσματος 1+1/2+1/3+...+1/n */
main()
{
  unsigned long int n, i=0;
  float sum=0.0;

  puts("Δώσε το πλήθος των όρων:");
  scanf("%U",&n);
  while (i<n) sum+=1.0/++i;
  printf("Οι %lu όροι της σειράς έχουν άθροισμα %.6f\n",n,sum);
}
```

6.5 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `do-while`

Όπως είδαμε στην προηγούμενη παράγραφο, ο βρόχος `while` ελέγχει τη συνθήκη τερματισμού στην *αρχή* του βρόχου. Αντίθετα ο βρόχος `do-while`, την ελέγχει στο *τέλος* του, αφού συμπληρωθεί κάθε πέρασμα από το σώμα του βρόχου. Έτσι το σώμα του βρόχου *εκτελείται πάντοτε τουλάχιστον μία φορά*. Η γενική μορφή του βρόχου `do-while` είναι η ακόλουθη:

```
do
  εντολή
while (παράσταση);
```

όπου *εντολή* μπορεί να είναι οποιαδήποτε απλή ή σύνθετη εντολή της C. Εκτελείται η *εντολή* και μετά υπολογίζεται η *παράσταση*. Αν είναι αληθής (μη-μηδενική) εκτελείται ξανά κ.ο.κ. Όταν η *παράσταση* γίνει ψευδής (μηδενική), ο βρόχος τερματίζεται, όπως φαίνεται στο Σχήμα 6.4.



Σχήμα 6.4: *Η εντολή do-while*

Μία εντολή `while` γίνεται ισοδύναμη μίας `do-while` και μία `do-while` γίνεται ισοδύναμη μίας εντολής `while`, ως ακολούθως:

```
while(παράσταση)          if(παράσταση)
  εντολή            ⇔       do
                              εντολή
                            while(παράσταση)

do                          εντολή
  εντολή            ⇔       while(παράσταση)
while(παράσταση)              εντολή
```

Το γνωστό μας πρόγραμμα που τυπώνει τους αριθμούς από 0 έως 9 και το τρέχον άθροισμα, χρησιμοποιώντας την εντολή `do-while` γίνεται:

```
/* doloop.c */
/* Εκτύπωση των αριθμών από 0 ως 9 */
/* και του τρέχοντος αθροίσματος   */
main()
{
  int count=0;
  int total=0;

  do
    printf("Μετρητής=%d, Αθροισμα=%d\n", count++, total+=count);
  while(count<10);
}
```

Το αποτέλεσμα είναι το ίδιο με αυτό του `wloop.c`.

Το παρακάτω πρόγραμμα υπολογίζει το πλήθος (`n`) των όρων του αθροίσματος

`S=1+1/2+1/3+...+1/n`

που χρειάζονται ώστε το άθροισμα να γίνει μεγαλύτερο από κάποιο όριο:

```
/* series2.c */
/* Υπολογισμός του πλήθους των όρων του αθροίσματος 1+1/2+1/3+... */
/* έως ότου αυτό γίνει μεγαλύτερο από κάποιο δεδομένο όριο        */
main()
{
  unsigned long int n=0;  /* το πλήθος των όρων      */
  float limit,sum=0;      /* το όριο και το άθροισμα */

  puts("Δώσε το όριο:");
  scanf("%f",&limit);
  do
  {
    n++;
    sum+=1.0/n;
  }
  while (sum<=limit);
  printf("Οι αριθμός των όρων είναι %lu\n",n);
}
```

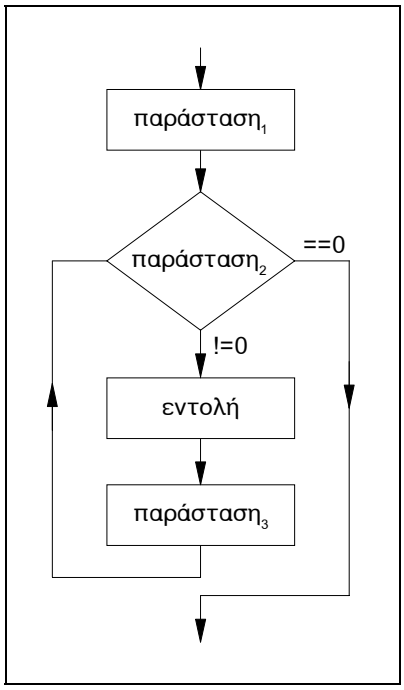
6.6 Η ΕΝΤΟΛΗ ΕΠΑΝΑΛΗΨΗΣ `for`

Η γενική μορφή της εντολής `for` είναι η ακόλουθη:

for($παράσταση_1$ ***;*** $παράσταση_2$ ***;*** $παράσταση_3$***)***
  εντολή

Στη συνηθέστερη περίπτωση, η *παράσταση*$_1$ και *παράσταση*$_3$ είναι παραστάσεις απόδοσης τιμής και η *παράσταση*$_2$ είναι παράσταση συσχέτισης. Η *παράσταση*$_1$ χρησιμοποιείται για να δίνεται αρχική τιμή στη μεταβλητή ελέγχου του βρόχου. Η *παράσταση*$_2$, ως συσχετιστική παράσταση, ελέγχει τη μεταβλητή ελέγχου του βρόχου σε σχέση με μία τιμή, για να καθορίσει τη στιγμή εξόδου από το βρόχο. Αν η *παράσταση*$_2$ είναι αληθής (μη-μηδενική) η *εντολή* εκτελείται, διαφορετικά ο

βρόχος τερματίζεται. Μετά την εκτέλεση της *εντολής* αποτιμάται η *παράσταση*$_3$ η οποία καθορίζει τον τρόπο μεταβολής της μεταβλητής ελέγχου του βρόχου (βήμα), κάθε φορά που επαναλαμβάνεται ο βρόχος. Στη συνέχεια ελέγχεται πάλι η *παράσταση*$_2$ κι αν είναι αληθής εκτελείται η *εντολή* κ.ο.κ., βλέπε Σχήμα 6.5. Είναι προφανές ότι η *εντολή* μπορεί να είναι οποιαδήποτε απλή ή σύνθετη εντολή της C.



Σχήμα 6.5: *Η εντολή for*

Η εντολή `for` είναι ισοδύναμη με την ακόλουθη εντολή `while`:

```
παράσταση1;
while (παράσταση2)
{
  εντολή
  παράσταση3;
}
```

Η επιλογή της εντολής `while` ή `for` εξαρτάται από τον προγραμματιστή. Για να χρησιμοποιηθεί η εντολή `for` πρέπει να γνωρίζουμε εκ των προτέρων πόσες φορές θα εκτελεστεί ο βρόχος. Το γνωστό μας πρόγραμμα που τυπώνει τους αριθμούς από το 0 έως το 9 και το τρέχον άθροισμα, χρησιμοποιώντας την εντολή `for`, γίνεται:

```
/* forloop.c */
/* Εκτύπωση των αριθμών από 0 ως 9 */
/* και του τρέχοντος αθροίσματος  */
main()
{
  int count, total=0;

  for(count=0,total=0; count<10; count++)
    printf("Μετρητής=%d, Αθροισμα=%d\n",count,total+=count);
}
```

Στο πρόγραμμα που ακολουθεί γίνεται χρήση των εντολών `for` και `while` για την εκτύπωση όλων των πρώτων αριθμών από το 1 έως κάποιο δεδομένο όριο. Ένας αριθμός είναι πρώτος όταν δεν έχει άλλους διαιρέτες εκτός του εαυτού του και της μονάδας. Η λογική του παρακάτω προγράμματος είναι η εξής: Ελέγχουμε έναν-έναν όλους τους αριθμούς από το 1 έως το όριο και αν κάποιος βρεθεί ότι είναι πρώτος, τον τυπώνουμε. Ο έλεγχος για τον καθένα γίνεται υποθέτοντας καταρχάς ότι ο αριθμός είναι πρώτος (`prime=1`) και στη συνέχεια ερευνώντας για πιθανούς διαιρέτες του αριθμού εκτός της μονάδας και του ίδιου, αν βρεθεί κάποιος διαιρέτης τότε αλλάζουμε την αρχική μας υπόθεση (`prime=0`) και ο αριθμός δεν τυπώνεται. Αν ψάχνοντας για τους πιθανούς διαιρέτες του αριθμού, ξεκινώντας από το 2 και αυξάνοντας κατά 1, φτάσουμε μέχρι και την τετραγωνική ρίζα του αριθμού και δεν έχουμε βρει κανέναν διαιρέτη, τότε είναι μάταιο να συνεχίσουμε την έρευνα για πιθανούς διαιρέτες μεγαλύτερους από την τετραγωνική ρίζα (γιατί ;).

```
/* prime.c */
/* Εύρεση των πρώτων αριθμών από το 1 */
/* έως κάποιο δεδομένο όριο           */
#include <math.h>
main()
{
  int nominator,number,prime,limit;

  printf("\nΔώσε το όριο: ");
  scanf("%d",&limit);
  for(number=1;number<=limit;number++)
  {
    nominator=1;  prime=1;
    while((++nominator<=sqrt(number)) && prime)
      if(number%nominator==0) prime=0;
    if(prime) printf("%5d",number);
  }
}
```

Οποιαδήποτε από τις τρεις *παραστάσεις* της εντολής `for` μπορεί να παραλείπεται, όμως τα ελληνικά ερωτηματικά πρέπει να παραμένουν. Την έλλειψη της *παράστασης* ο μεταγλωττιστής τη θεωρεί ως αληθή παράσταση κι έτσι μία δομή `for` της μορφής:

`for(παράσταση`$_1$`;  ;παράσταση`$_3$`) εντολή`

είναι ένας ατέρμων βρόχος που διακόπτεται με άλλα μέσα, όπως μία εντολή `break`.

Η *εντολή* στο βρόχο `for` μπορεί να είναι η *κενή εντολή*. Με τον τρόπο αυτό μπορούμε να δημιουργούμε βρόχους χρονικής καθυστέρησης, π.χ.:

```
for(i=0;i<1000;i++);
```

6.6.1 Ο ΤΕΛΕΣΤΗΣ `comma`

Συχνά σ'ένα βρόχο `for` γίνεται χρήση δύο ή περισσότερων μεταβλητών ελέγχου του βρόχου, όπως π.χ. στο παρακάτω τμήμα κώδικα:

```
for(x=0,y=0; x+y<100; x++,y++) printf("%d",x+y);
```

με το οποίο τυπώνονται οι αριθμοί από 0 έως 98 σε βήματα του 2. Παρατηρούμε ότι χρησιμοποιείται κόμμα (`,`) στις αρχικές τιμές και στα βήματα.

Στην πραγματικότητα το *κόμμα* είναι τελεστής της C (βλέπε Πίνακα 4.5) και λέγεται *τελεστής σειριακής αποτίμησης* μιας και χρησιμεύει στο να χωρίζουμε μία λίστα από ίδια στοιχεία που πρόκειται να αποτιμηθούν με τη σειρά από αριστερά προς τα δεξιά.

Όταν ο τελεστής κόμμα χρησιμοποιείται στη δεξιά πλευρά μίας παράστασης απόδοσης τιμής, τότε η τιμή που αποδίδεται στο αριστερό μέλος (και κατά συνέπεια σε όλη την παράσταση) είναι η τιμή της τελευταίας παράστασης της λίστας των παραστάσεων που χωρίζονται με κόμμα. Για παράδειγμα, όταν εκτελεστούν οι:

```
y=10;
x=(y-=5,100/y);
```

το x θα έχει την τιμή 20, επειδή η αρχική τιμή 10 του y μειώνεται κατά 5 και μετά αυτή η τιμή διαιρεί το 100, δίνοντας αποτέλεσμα 20.

6.7 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί πρόγραμμα που να δέχεται την τιμή ενός αριθμού και να βρίσκει και τυπώνει την *απόλυτη τιμή* αυτού.

```
/* abs.c */
/* Πρόγραμμα εύρεσης απόλυτης τιμής */
main()
{
  float number, abs;

  puts("Αριθμός: ");
  scanf("%f",&number);
  if(number<0)
    abs=-number;
  else
    abs=number;
  printf("Η απόλυτη τιμή του %f είναι %f",number,abs);
 }
```

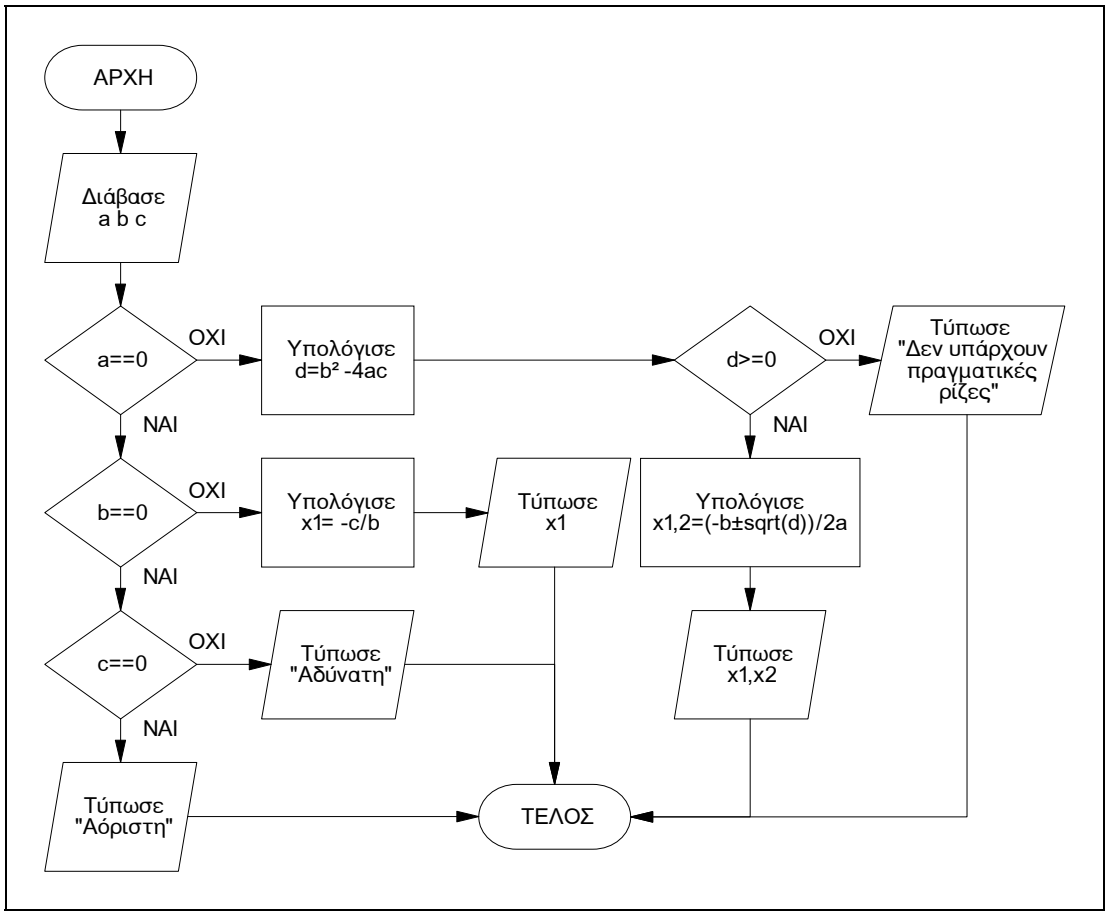
2. Για να υπάρχει τρίγωνο με πλευρές a, b, c πρέπει να ισχύει: a<b+c *και* b<a+c *και* c<a+b. Γράψτε πρόγραμμα που να δέχεται τα a, b, c και να τυπώνει το εμβαδόν του τριγώνου με πλευρές a, b, c *αν υπάρχει*. Το εμβαδόν δίνεται από τον παρακάτω τύπο (τύπος του Ήρωνα):

$E = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$ όπου $2s = a + b + c$

```
/* tr_area.c */
/* Υπολογισμός εμβαδού τριγώνου με τον τύπο του Ηρωνα */
# include <math.h>
main()
{
  float a,b,c,s,area;

  puts("Δώσε τις πλευρές a b c του τριγώνου: ");
  scanf("%f %f %f",&a,&b,&c);
  if(a<b+c && b<a+c && c<a+b)
  {
    s=(a+b+c)/2;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("Εμβαδόν τριγώνου: %f",area);
  }
  else puts("Δεν υπάρχει τέτοιο τρίγωνο");
}
```

3. Γράψτε ένα πρόγραμμα που να υπολογίζει τις ρίζες της δευτεροβάθμιας εξίσωσης $ax^2+bx+c=0$.

Το λογικό διάγραμμα του προγράμματος δίνεται στο Σχήμα 6.6. Σύμφωνα με αυτό, γράφουμε το πρόγραμμα:



Σχήμα 6.6: *Λογικό διάγραμμα λύσης της δευτεροβάθμιας εξίσωσης*

```
/* trionymo.c */
/* Υπολογισμός των ριζών δευτεροβάθμιας εξίσωσης */
# include <math.h>
main()
{
  float a,b,c,d,x1,x2;

  printf("Δώσε τα a b c :");
  scanf("%f %f %f",&a,&b,&c);
  if (a==0)
    if (b==0)
      if (c==0) puts("Η εξίσωση είναι αόριστη");
      else puts("Η εξίσωση είναι αδύνατη");
    else
    {
      x1=-c/b;
```

```
      printf("Μία ρίζα: x1=%0.3f",x1);
    }
  else
  {
    d=b*b-4*a*c;
    if (d>=0)
    {
      x1=(-b+sqrt(d))/(2*a);
      x2=(-b-sqrt(d))/(2*a);
      printf("Ρίζες: x1=%0.3f , x2=%0.3f",x1,x2);
    }
    else puts("Δεν υπάρχουν πραγματικές ρίζες");
  }
}
```

4. Ας υποθέσουμε ότι οι μήνες του έτους, Ιανουάριος έως Δεκέμβριος, αντιστοιχούν στους αριθμούς 1 έως 12. Γράψτε πρόγραμμα που να δέχεται έναν ακέραιο αριθμό και να τυπώνει το όνομα της *εποχής* στην οποία ανήκει ο μήνας, *αν υπάρχει*.

```
/* month.c */
/* Εκτύπωση εποχής στην οποία ανήκει κάποιος μήνας */
main()
{
  int i;

  printf("\nΔώσε μήνα (ακέραιο από 1 έως 12): ");
  scanf("%d",&i);
  switch (i)
  {
    case 1 : case 2 :  case 12: puts("Χειμώνας");  break;
    case 3 : case 4 :  case 5 : puts("Άνοιξη");    break;
    case 6 : case 7 :  case 8 : puts("Καλοκαίρι"); break;
    case 9 : case 10:  case 11: puts("Φθινόπωρο"); break;
    default: puts("Ακυρος αριθμός");
  }
}
```

5. Να γραφεί ένα πρόγραμμα που να μετρά τις *λέξεις*, τους *χαρακτήρες* και τα *ψηφία* μίας φράσης που θα δίνει ο χρήστης από το πληκτρολόγιο.

```
/* count.c */
/* Απαρίθμηση χαρακτήρων, ψηφίων και λέξεων μιας πρότασης */
main()
{
  char ch;
  int character=0,number=0,word=0;
  puts("Γράψε μια πρόταση (για τέλος δώσε ENTER):");
  while ((ch=getche()) != '\r')
  {
    character++ ;
    if(ch>='0' && ch<='9') number++;
    if(ch==' ') word++;
  }
  printf("\nΟ αριθμός των χαρακτήρων είναι %d\n",character);
  printf("\nΟ αριθμός των ψηφίων είναι %d\n",number);
```

```
  printf("\nO αριθμός των λέξεων είναι %d\n",word+1);
}
```

6. Μία μπάλα αφήνεται να πέσει από ένα δεδομένο ύψος πάνω από το οριζόντιο επίπεδο και σε κάθε αναπήδηση φτάνει στο 75% του ύψους που είχε προηγουμένως. Γράψτε ένα πρόγραμμα που να δέχεται το αρχικό ύψος, σε μέτρα, και να υπολογίζει πόσες αναπηδήσεις θα κάνει η μπάλα μέχρι να σταματήσει. Θεωρήστε ότι οι αναπηδήσεις σταματάνε όταν το ύψος τους γίνει μικρότερο των 10mm.

```
/* ball.c */
/* Υπολογισμός αναπηδήσεων μπάλας */
main()
{
  float h;
  int n=0;

  puts("Αρχικό ύψος: ");
  scanf("%f",&h);
  h*=0.75;
  while (h>=0.01)
  {
    n++;
    h*=0.75;
  }
  printf("Το πλήθος των αναπηδήσεων είναι: %d\n",n);
}
```

7. Να γραφεί πρόγραμμα το οποίο να υπολογίζει το πλήθος των όρων του αθροίσματος:

$S = \sqrt{1} + \sqrt{2} + \sqrt{3} + \ldots$

που απαιτούνται, ώστε το άθροισμα να γίνει μεγαλύτερο από κάποιο όριο το οποίο θα δίνεται από το πληκτρολόγιο.

```
/* series3.c */
/* Υπολογισμός πλήθους όρων αθροίσματος */
# include <math.h>
main()
{
  unsigned long int n=0;
  float limit,sum=0;

  puts("Δώσε το όριο:");
  scanf("%f",&limit);
  do
  {
    n++;
    sum+=sqrt(n);
  }
  while (sum<=limit);
  printf("Οι αριθμός των όρων είναι %lu\n",n);
}
```

6.8 ΑΣΚΗΣΕΙΣ

1. Γράψτε ένα πρόγραμμα που θα δέχεται έναν ακέραιο δεκαδικό αριθμό και θα έχει τη δυνατότητα, μέσα από κάποιο menu, να μετατρέπει αυτόν στον αντίστοιχο οκταδικό ή δεκαεξαδικό ακέραιο.
2. Να γραφεί πρόγραμμα που να δέχεται την τιμή ενός ακεραίου αριθμού και να ελέγχει αν αυτός είναι *πρώτος*.
3. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο ένα σύνολο αριθμών και να τυπώνει τη *μέση αριθμητική τιμή* τους, το *μικρότερο* και το *μεγαλύτερο* απ'αυτούς. Το τέλος του συνόλου να δηλώνεται με τον αριθμό 0.
4. Γράψτε πρόγραμμα που να υπολογίζει και εκτυπώνει την τιμή του π (3.14...) για καθέναν από τους τύπους που δίνονται παρακάτω, προσθέτοντας (ή πολ/άζοντας) τους όρους που είναι μεγαλύτεροι του 10^{-6}.

   α) $\pi^4/96 = 1 + 1/3^4 + 1/5^4 + 1/7^4 + \ldots$

   β) $\pi^2/6 = 1 + 1/2^2 + 1/3^2 + 1/4^2 + \ldots$

   γ) $\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \ldots$
5. Γράψτε πρόγραμμα που να υπολογίζει και τυπώνει τις τιμές των παρακάτω παραστάσεων λαμβάνοντας υπόψη όρους που είναι μεγαλύτεροι του 10^{-6}.

   α) $e^x = 1 + x/1! + x^2/2! + x^2/3! + \ldots$

   β) $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$

   γ) $\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \ldots$

   Στις τριγωνομετρικές συναρτήσεις η γωνία να δίνεται σε μοίρες και κατόπιν να μετατρέπεται σε ακτίνια πριν την εφαρμογή των τύπων.
6. Δίνονται διαδοχικά από το πληκτρολόγιο τα βάρη μερικών κιβωτίων (*απροσδιόριστο το πλήθος τους*) με το τελευταίο κιβώτιο να είναι άδειο (βάρος μηδέν). Ζητείται να μετρηθεί: α) Το πλήθος των κιβωτίων που έχουν βάρος μεγαλύτερο των 100 Nt β) Το πλήθος των κιβωτίων που έχουν βάρος μικρότερο ή ίσο των 100 Nt και μεγαλύτερο των 50 Nt και γ) Το πλήθος των κιβωτίων που έχουν βάρος μικρότερο ή ίσο των 50 Nt.
7. Δίνονται από το πληκτρολόγιο δύο χαρακτήρες. Ζητείται να γραφεί πρόγραμμα που να τυπώνει αυτούς τους χαρακτήρες και όλους τους ενδιάμεσους.
8. Να γραφεί πρόγραμμα που να τυπώνει α) Τους χαρακτήρες που αντιστοιχούν στους κωδικούς ASCII από 65 έως 90. β) Τους κωδικούς ASCII που αντιστοιχούν στους πεζούς λατινικούς χαρακτήρες.
9. Γράψτε πρόγραμμα που να υπολογίζει και τυπώνει στο τέλος κάθε έτους τον τόκο ενός γνωστού κεφαλαίου για έναν ορισμένο αριθμό ετών και με γνωστό επιτόκιο.
10. Γράψτε πρόγραμμα που να παράγει έναν πίνακα μετατροπής θερμοκρασιών από βαθμούς Celsius σε βαθμούς Fahrenheit στην περιοχή από -40° C έως +40° C. Ο τύπος μετατροπής δίνεται στην Άσκηση 8 του Κεφαλαίου 4.

11. Να γραφούν προγράμματα που να τυπώνουν καθένα από τα παρακάτω:

```
1             1             1
12            23            13
123           345           135
1234          4567          1357
12345         56789         13579
```

12. Να γραφούν προγράμματα που να τυπώνουν καθένα από τα παρακάτω:

```
*             *****           *          *****
**            ****             **         ****
***           ***               ***        ***
****          **                 ****       **
*****         *                   *****      *


*                *          *******        ***********
**              ***          *****         *****   *****
***            *****           ***         ***       ***
****          *******            *          *           *
***            *****           ***         ***       ***
**              ***          *****         *****   *****
*                *          *******        ***********
```

13. Γράψτε πρόγραμμα που να υπολογίζει την τετραγωνική ρίζα (x) ενός αριθμού (a) χρησιμοποιώντας τον επαναληπτικό αλγόριθμο του Newton:

$$x_{i+1} = \frac{\frac{a}{x_i} + x_i}{2} \qquad \text{με } x_0 \text{ τυχαίο π.χ.: } \quad x_0 = a/2$$

με ακρίβεια 6 κλασματικών ψηφίων.

14. Να γραφεί πρόγραμμα που να υπολογίζει και τυπώνει τα πρώτα 20 μερικά αθροίσματα του επόμενου συνεχούς κλάσματος:

$$S_K = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{\ddots}}}}$$

παρατηρώντας ότι: $S_1=1$ και $S_{K+1}=1+1/S_K$

15. Γράψτε πρόγραμμα που να υπολογίζει τους N (δεδομένο) πρώτους όρους της ακολουθίας Fibonacci. Η ακολουθία ορίζεται ως εξής:

$F_1 = 1$, $F_2 = 1$ και $F_N = F_{N-1} + F_{N-2}$ για $N>2$

16. Γράψτε πρόγραμμα που να υπολογίζει τους συνδυασμούς και τις διατάξεις των M ανά N. Πρέπει να ισχύει: M≥N, M>0, N>0.

Συνδυασμοί : $\binom{M}{N} = \frac{M!}{N!\,(M - N)\,!}$

Διατάξεις : $P_{M,N} = M(M - 1)\ldots(M - N + 1)$

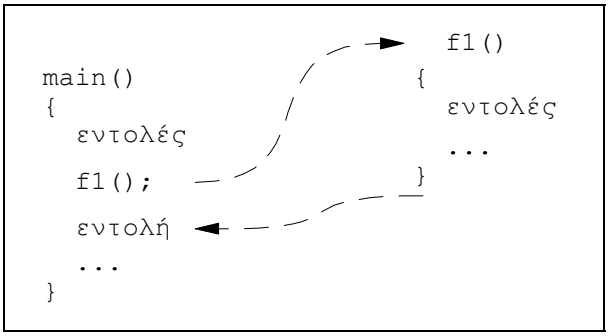
ΚΕΦΑΛΑΙΟ 7
ΣΥΝΑΡΤΗΣΕΙΣ

Η γλώσσα C βασίζεται στην αρχή των *δομικών τμημάτων* (building blocks). Τα δομικά τμήματα καλούνται *συναρτήσεις* (functions). Μία συνάρτηση είναι ένα τμήμα κώδικα, που εκτελεί κάποιο συγκεκριμένο έργο. Χρησιμοποιώντας συναρτήσεις, μπορούμε να χωρίσουμε τις εκτεταμένες διαδικασίες προγραμματισμού σε μικρότερες και αυτόνομες. Αυτό έχει ως αποτέλεσμα, ένα μεγάλο και σύνθετο πρόβλημα, του οποίου η λύση είναι δύσκολη, να διαιρείται σε μικρότερα υποπροβλήματα τα οποία είναι εύκολο να επιλυθούν. Με τον τρόπο αυτό καταφέρνουμε σε κάθε δεδομένη χρονική στιγμή να μην ασχολούμαστε με όλο το πρόβλημα, αλλά με κάποιο τμήμα του το οποίο είναι βέβαια απλούστερο (τεχνική "διαίρει και βασίλευε" - divide and conquer). Για τη λύση του κάθε υποπροβλήματος υλοποιούμε μία συνάρτηση και τέλος συνθέτοντας όλες αυτές τις συναρτήσεις υλοποιούμε το πρόγραμμα που επιλύει το αρχικό πρόβλημα.

Επίσης χρησιμοποιούμε τις συναρτήσεις όταν θέλουμε να γράψουμε μόνο μία φορά, ένα κομμάτι κώδικα προγράμματος που επαναλαμβάνεται συχνά (γι'αυτό λέγονται και *ρουτίνες* - routines).

Ένα πρόγραμμα σε C είναι μία συλλογή από συναρτήσεις. Δεν υπάρχει περιορισμός στον αριθμό των συναρτήσεων που μπορούν να υπάρχουν σ'ένα πρόγραμμα, ούτε και στη σειρά με την οποία αυτές βρίσκονται. Ο μόνος περιορισμός είναι ότι πρέπει να υπάρχει μία και μόνο μία συνάρτηση με όνομα `main()`, η οποία είναι και η πρώτη συνάρτηση του προγράμματος που εκτελείται, οπουδήποτε κι αν βρίσκεται μέσα στο πρόγραμμα. Αυτή με τη σειρά της μπορεί να καλεί άλλες συναρτήσεις, φυσικά και οι άλλες συναρτήσεις έχουν δικαίωμα να καλούν κι αυτές συναρτήσεις με τη σειρά τους.

Όταν μία συνάρτηση (καλούσα) καλεί κάποια άλλη (καλούμενη), ο έλεγχος του προγράμματος μεταφέρεται στην πρώτη εντολή της καλούμενης συνάρτησης, όταν τελειώσει η εκτέλεση της καλούμενης συνάρτησης τότε ο έλεγχος του προγράμματος μεταφέρεται στην αμέσως επόμενη, από την κλήση της συνάρτησης, εντολή της καλούσας συνάρτησης, (Σχήμα 7.1).



Σχήμα 7.1: *Μεταφορά του ελέγχου κατά την κλήση συνάρτησης*

7.1 ΟΡΙΣΜΟΣ ΣΥΝΑΡΤΗΣΗΣ

Η γλώσσα C προσφέρει έτοιμες κάποιες συναρτήσεις (μαθηματικές, συστήματος, αλφαριθμητικών κ.λ.π.) οι οποίες βρίσκονται στις διάφορες *βιβλιοθήκες* της γλώσσας και μπορούμε να τις χρησιμοποιήσουμε. Μέχρι τώρα έχουμε χρησιμοποιήσει μόνο συναρτήσεις σαν τις `printf()`, `scanf()`,`getche()`, τις οποίες βρήκαμε έτοιμες.Τώρα έφτασε η στιγμή να γράψουμε τις δικές μας συναρτήσεις.

Η γενική μορφή ορισμού μίας συνάρτησης είναι:

```
επιστρεφόμενος_τύπος  αναγν_συνάρτ(λίστα_δηλώσ_τυπικών_παραμ)
{
  δηλώσεις_τοπικών_μεταβλητών
  εντολές
}
```

Όπου *επιστρεφόμενος_τύπος* είναι ο τύπος της τιμής που επιστρέφει η συνάρτηση. Μπορεί όμως η συνάρτηση να μην επιστρέφει καμία τιμή, απλώς να επιτελεί μία καθορισμένη εργασία (όπως οι διαδικασίες-procedures στην Pascal), τότε αυτή η συνάρτηση θα έχει τύπο `void`. Αν ο επιστρεφόμενος τύπος μίας συνάρτησης είναι ακέραιος ή `void`, τότε μπορούμε να τον παραλείψουμε από τον ορισμό της συνάρτησης.

Η *λίστα_δηλώσεων_τυπικών_παραμέτρων* είναι μία λίστα που καθορίζει τους τύπους και τα αναγνωριστικά των παραμέτρων που χρησιμοποιούνται για τον ορισμό της συνάρτησης (λέγονται *τυπικές παράμετροι*). Κάθε δήλωση χωρίζεται από την επόμενη με κόμμα. Από αυτή τη λίστα παραλαμβάνονται οι τιμές των *πραγματικών παραμέτρων* (λέγονται και *ορίσματα*) κατά την κλήση της συνάρτησης. Επειδή μία συνάρτηση μπορεί να μη δέχεται παραμέτρους, η λίστα μπορεί να είναι κενή, οι παρενθέσεις όμως είναι *απαραίτητες*.

Στο τμήμα *δηλώσεις_τοπικών_μεταβλητών* δηλώνονται όλες οι μεταβλητές που χρειάζονται για να υλοποιηθεί η συνάρτηση (εκτός των τυπικών παραμέτρων), αυτές οι μεταβλητές έχουν *τοπική εμβέλεια*. Δηλαδή είναι γνωστές μόνο μέσα στη συνάρτηση όπου δηλώνονται.

7.1.1 ΠΡΟΤΥΠΟΠΟΙΗΣΗ ΣΥΝΑΡΤΗΣΗΣ

Μία συνάρτηση είναι γνωστή στο πρόγραμμα που ορίζεται, από το σημείο που ορίζεται (υλοποιείται) και μετά. Για να τη χρησιμοποιήσουμε πριν από τον ορισμό της θα πρέπει να *γνωστοποιήσουμε* τον τύπο της στο υπόλοιπο πρόγραμμα, δηλαδή θα πρέπει να τη δηλώσουμε στην αρχή του προγράμματος, χρησιμοποιώντας ένα *πρότυπο*, ως εξής:

```
επιστρεφόμενος_τύπος  αναγν_συνάρτησης(λίστα_τύπων_τυπικών_παραμ);
```

Η διαδικασία αυτή λέγεται *προτυποποίηση συνάρτησης*. Η προτυποποίηση για συναρτήσεις που επιστρέφουν τιμές ακεραίου τύπου καθώς και για τις συναρτήσεις `void` είναι προαιρετική.

7.1.2 ΑΠΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Ας δούμε τώρα ένα απλό πρόγραμμα που κάνει χρήση μίας συνάρτησης:

```
/* textbox.c */
/* Δημιουργία πλαισίου με '*' */
void line(); /* πρότυπο συνάρτησης, θα μπορούσε να παραλειφθεί */
main()
{
  line();
  printf("* ΜΕΓΑΣ ΑΛΕΞΑΝΔΡΟΣ *\n");
  line();
}

/* line() */
/* Εκτύπωση γραμμής χρησιμοποιώντας 20 '*' */
void line()
{
  int j; /* τοπική μεταβλητή */

  for(j=1;j<=20;j++)printf("*");
  printf("\n");
}
```

Το αποτέλεσμα αυτού του προγράμματος είναι:

```
********************
* ΜΕΓΑΣ ΑΛΕΞΑΝΔΡΟΣ *
********************
```

Στο παρακάτω πρόγραμμα γίνεται πάλι κλήση μίας συνάρτησης η οποία τυπώνει το χαρακτήρα που η δεκαεξαδική του τιμή στον κώδικα ASCII είναι 7, αυτός ο χαρακτήρας καλείται BELL (καμπανάκι) και η εκτύπωσή του προκαλεί έναν ήχο "μπιπ" από το μεγάφωνο του Η/Υ:

```
/* beeptest.c */
/* Ελεγχος συνάρτησης */
void twobeep();
main()
{
  twobeep();
  printf("Πάτα ένα πλήκτρο\n");
  getch();
  twobeep();
}

/* twobeep() */
/* Πρόκληση ηχητικού σήματος από το μεγάφωνο του Η/Υ */
void twobeep()
{
  int k;                 /* τοπική μεταβλητή */
  printf("\x7");         /* πρώτο μπιπ       */
  for(k=1;k<=5000;k++);  /* καθυστέρηση      */
  printf("\x7");         /* δεύτερο μπιπ     */
}
```

Αυτό το πρόγραμμα στην αρχή καλεί μία συνάρτηση, την `twobeep()`, που κάνει ακριβώς αυτό που λέει το όνομά της: ηχεί δύο μπιπ χωρισμένα από ένα σύντομο σιωπηλό διάλειμμα. Έπειτα το πρόγραμμα μας ζητά να πατήσουμε ένα πλήκτρο, όταν το πατήσουμε, ηχεί ξανά δύο μπιπ.

7.2 Η ΕΝΤΟΛΗ `return`

Οι συναρτήσεις που είδαμε στα προηγούμενα παραδείγματα δεν επέστρεφαν καμία τιμή στη συνάρτηση που τις καλούσε. Απλώς, μετά την κλήση τους, εκτελούνταν όλες οι εντολές που περιείχαν και ο έλεγχος του προγράμματος επέστρεφε χωρίς τιμή στην επόμενη εντολή της καλούσας συνάρτησης. Έτσι λοιπόν η εκτέλεση της συνάρτησης τερματιζόταν όταν συναντιόταν το άγκιστρο κλεισίματος (`}`). Μπορούμε όμως να κάνουμε μία συνάρτηση να τερματίσει, και ο έλεγχος να επιστρέψει στην καλούσα συνάρτηση, σε οποιοδήποτε σημείο της χρησιμοποιώντας την εντολή:

```
return;
```

Σ'αυτή την περίπτωση η συνάρτηση τερματίζεται όταν εκτελεστεί η `return` και δεν επιστρέφεται καμία τιμή στην καλούσα συνάρτηση. Είναι προφανές ότι η επιστροφή χωρίς τιμή από μία συνάρτηση, μπορεί να χρησιμοποιηθεί μόνο σε συναρτήσεις που έχουν οριστεί τύπου `void`.

Για παράδειγμα η παρακάτω συνάρτηση εμφανίζει το αποτέλεσμα της ύψωσης ενός αριθμού σ'ένα θετικό ακέραιο:

```
void power()
{
  int exp;
  float base, i=1;

  printf("\nΔώσε τη βάση και τον εκθέτη ");
  scanf("%f %d",&base,&exp);
  if(exp<0)
    return; /* δε μπορεί να χειριστεί αρνητικούς εκθέτες */
  for( ;exp;exp--)
    i*=base;
  printf("Αποτέλεσμα: %.5f",i);
}
```

Αν ο εκθέτης είναι αρνητικός, η εντολή `return` τερματίζει τη συνάρτηση πριν φτάσει στο τελικό άγκιστρο, αλλά δεν επιστρέφει καμία τιμή.

Μία συνάρτηση μπορεί να έχει περισσότερες από μία εντολές `return`. Για να επιστρέψει η καλούμενη συνάρτηση μία τιμή στην καλούσα, πρέπει το `return` να ακολουθείται από μία παράσταση:

```
return(παράσταση);
```

όπου οι παρενθέσεις είναι προαιρετικές, αλλά συνιστώνται. Ο επιστρεφόμενος τύπος είναι αυτός που ορίστηκε κατά τον ορισμό της συνάρτησης. Με τη χρήση της `return`, μόνο μία τιμή μπορεί να επιστραφεί από μία συνάρτηση.

Το παρακάτω πρόγραμμα χρησιμοποιεί τη συνάρτηση `read_int()` η οποία διαβάζει από το πληκτρολόγιο έναν ακέραιο και τον επιστρέφει στη `main()`. Η `main()` προσθέτει μία ακολουθία ακεραίων, της οποίας το τέλος δηλώνεται με τον αριθμό μηδέν:

```
/* add.c */
/* Πρόσθεση ακολουθίας ακεραίων που τελειώνει με 0 */
int read_int();
main()
{
  int i,s=0;

  printf("\nΔώσε ακέραιους, 0 για τέλος:\n");
  while(i=read_int())
    s+=i;
  printf("Το άθροισμα είναι %d\n",s);
}

/* read_int() */
/* Δέχεται ακέραιο και τον επιστρέφει στη main() */
int read_int()
{
  int x;

  scanf("%d",&x);
  return(x);
}
```

Στην παράσταση απόδοσης τιμής:

```
i=read_int()
```

του παραπάνω προγράμματος, η δεξιά πλευρά ονομάζεται *κλήση* της συνάρτησης. Η τιμή της παράστασης είναι η τιμή του `i`. Εφόσον είναι μη-μηδενική, ερμηνεύεται ως αληθής και το `s` ενημερώνεται με την εντολή:

```
s+=i;
```

7.3 ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΗΣ "ΚΑΤ'ΑΞΙΑ" (by value)

Μέχρι τώρα οι συναρτήσεις που χρησιμοποιήσαμε δεν ήταν πολύ ευέλικτες. Τις καλούσαμε και έκαναν αυτά που σχεδιάστηκαν να κάνουν, επιστρέφοντας ή όχι μία τιμή. Η καλούσα συνάρτηση δε μεταβίβαζε πληροφορίες στην καλούμενη (η λίστα δηλώσεων τυπικών παραμέτρων ήταν κενή).

Ο μηχανισμός για τη μεταβίβαση πληροφοριών σε μία συνάρτηση είναι η *παράμετρος*. Μπορούμε να μεταβιβάσουμε σε μία συνάρτηση τις *τιμές* όσων παραμέτρων θέλουμε.

Στο επόμενο πρόγραμμα χρησιμοποιούμε τη συνάρτηση `area()` η οποία δέχεται μία τιμή που αντιπροσωπεύει την ακτίνα μίας σφαίρας και επιστρέφει το εμβαδόν της επιφάνειας της σφαίρας ($E=4\pi r^2$). Η μεταβίβαση της πληροφορίας στην `area()` γίνεται μέσω μίας παραμέτρου. Η `rad` είναι *τυπική* παράμετρος,

ενώ η `radius` είναι *πραγματική*, ο τύπος της πραγματικής παραμέτρου είναι ίδιος με αυτόν της τυπικής:

```
/* sphere.c */
/* Υπολογισμός εμβαδού σφαίρας */
#define PI 3.14159
float area(float);
main()
{
  float radius;

  printf("Δώσε την ακτίνα της σφαίρας: ");
  scanf("%f", &radius);
  printf("Το εμβαδόν της σφαίρας είναι %.2f\n",area(radius));
}

/* area() */
/* Επιστρέφει το εμβαδόν της σφαίρας */
float area(float rad)
{
  return(4*PI*rad*rad);
}
```

Στο παρακάτω πρόγραμμα χρησιμοποιούμε τη συνάρτηση `power()` η οποία βρίσκει τη δύναμη ενός πραγματικού αριθμού, υψωμένου σ'έναν ακέραιο εκθέτη. Η συνάρτηση `power()` έχει δύο παραμέτρους (τη βάση και τον εκθέτη) και επιστρέφει μία τιμή η οποία και τυπώνεται:

```
/* power.c */
/* Υπολογισμός δύναμης ενός πραγματικού σε εκθέτη ακέραιο */
float power(float,int);
main()
{
  float base;
  int exp;

  puts("\nΔώσε τη βάση και τον εκθέτη: ");
  scanf("%f %d",&base, &exp);
  printf("%.5f στην %d = %.5f",base,exp,power(base,exp));
  /* τα base, exp είναι οι πραγματικές παράμετροι */
}

/* power() */
/* Επιστρέφει τη δύναμη */
float power(float b,int e) /* τα b,e είναι οι τυπικές παράμετροι */
{
  int i;
  float p=1.0;

  i=abs(e);     /* επιστρέφει την απόλυτη τιμή του ορίσματός της */
  for( ;i;i--) p*=b;
  if(e>=0) return(p);  /* αν ο εκθέτης είναι μη-αρνητικός */
  else return(1/p);    /* αν ο εκθέτης είναι αρνητικός    */
}
```

Στην κλήση μίας συνάρτησης, οι παράμετροι περιλαμβάνονται μέσα στις παρενθέσεις `()`. Στην περίπτωση του παραπάνω παραδείγματος υπάρχουν δύο παράμετροι, που είναι οι `base` και `exp`. Οι παράμετροι αυτές λέγονται *πραγματικές παράμετροι* ή *ορίσματα* (actual parameters). Αντίθετα οι παράμετροι που χρησιμοποιούνται κατά τον ορισμό της συνάρτησης, λέγονται *τυπικές παράμετροι* ή απλώς *παράμετροι* (formal parameters).

Η κλήση της συνάρτησης όπως έγινε στο προηγούμενο παράδειγμα:

```
power(base,exp)
```

ονομάζεται *κλήση κατ'αξία* (call by value) που είναι ο μόνος τρόπος μεταβίβασης παραμέτρων στη γλώσσα C. Κατ'αυτόν τον τρόπο *η τιμή* μιας πραγματικής παραμέτρου *αντιγράφεται* στην αντίστοιχη τυπική παράμετρο της συνάρτησης.

Έτσι, οι πιθανές αλλαγές που γίνονται στις τιμές των τυπικών παραμέτρων της συνάρτησης, δεν έχουν καμία επίδραση στις τιμές των πραγματικών παραμέτρων. Αυτό φαίνεται και στο ακόλουθο πρόγραμμα:

```
/* value.c */
main()
{
  int i=0;

  printf("\n%3d",i);
  p(i);
  printf("%3d\n",i);
}

/* p() */
p(int j)
{
  do printf("%3d",++j); while(j<5);
}
```

το οποίο τυπώνει:

```
  0  1  2  3  4  5  0
```

Η τιμή της μεταβλητής `i` στη συνάρτηση `main()` δε μπορεί να αλλαχθεί από την `p()`, εφόσον μόνο ένα *αντίγραφο* της τιμής του `i` μεταφέρεται στην `p()`.

7.3.1 ΣΥΝΑΡΤΗΣΗ ΩΣ ΠΑΡΑΜΕΤΡΟΣ ΣΥΝΑΡΤΗΣΗΣ

Μία συνάρτηση, για ένα πρόγραμμα εκπροσωπεί μία τιμή. Θα μπορούσε, λοιπόν, μία συνάρτηση να χρησιμοποιηθεί ως πραγματική παράμετρος σε κάποια άλλη συνάρτηση.

Αυτό γίνεται στο παρακάτω πρόγραμμα, το οποίο δέχεται τρεις ακέραιους αριθμούς από το πληκτρολόγιο και υπολογίζει το γινόμενο (`total`) του τρίτου αριθμού (`c`) επί το μεγαλύτερο των άλλων δύο. Παρατηρούμε ότι η τιμή της συνάρτησης `maxinteger()` χρησιμοποιείται ως πραγματική παράμετρος για τη συνάρτηση `multiply()`. Όταν έρθει η στιγμή να εκτελεστεί η εντολή `total=multiply(maxinteger(a,b),c);` καλείται καταρχάς η συνάρτηση `maxinteger()` η οποία επιστρέφει μία τιμή που χρησιμοποιείται ως παράμετρος της `multiply()`.

```
/* Συνάρτηση ως παράμετρος συνάρτησης */
int maxinteger(int,int);
int multiply(int,int);
main()
{
  int a,b,c,total;

  puts("Δώσε τρεις ακέραιους:");
  scanf("%d %d %d",&a,&b,&c);
  total=multiply(maxinteger(a,b),c);
  printf("Αποτέλεσμα: %d",total);
}

/* maxinteger() */
/* επιστρέφει το μεγαλύτερο από δύο ακεραίους */
int maxinteger(int x,int y)
{
  if(x>y)
    return(x);
  else
    return(y);
}

/* myltiply */
/* πολλαπλασιάζει δύο αριθμούς */
int multiply(int k,int l)
{
  return(k*l);
}
```

7.4 ΤΟΠΙΚΕΣ ΚΑΙ ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Μία μεταβλητή η οποία δηλώνεται μέσα σ'ένα τμήμα κώδικα (μπλοκ), το οποίο οριοθετείται από τα άγκιστρα ανοίγματος και κλεισίματος ({}), είναι *τοπική* (local) για το συγκεκριμένο τμήμα. Η *εμβέλεια* των τοπικών μεταβλητών, περιορίζεται στο τμήμα όπου δηλώνονται. Έτσι, οι μεταβλητές που δηλώνονται μέσα σε μία συνάρτηση είναι γνωστές μόνο μέσα σ'αυτήν και σε καμία άλλη, για το λόγο αυτό λέγονται και *εσωτερικές* (internal). Ας δούμε το παρακάτω πρόγραμμα:

```
/* local.c */
/* Τοπικές μεταβλητές */
main()
{
  int x=10;

  printf("Η x στη main() είναι    : %3d\n",x);
  f();
  printf("Η x στη main() παραμένει: %3d\n",x);
}
```

```
/* f() */
f()
{
  float x=100.123;

  printf("Η x στην f() είναι      : %.3f\n",x);
}
```

Η έξοδος του προγράμματος είναι:

```
Η x στη main() είναι    :  10
Η x στην f() είναι      : 100.123
Η x στη main() παραμένει:  10
```

Στο παραπάνω πρόγραμμα έχουμε δύο μεταβλητές με το ίδιο όνομα `x`, μία ακέραια μεταβλητή στη `main()` και μία μεταβλητή κινητής υποδιαστολής στην `f()`. Η `x` της `main()` δεν έχει καμία σχέση με τη `x` της `f()`, γιατί καθεμιά είναι γνωστή μόνο στον κώδικα του δικού της τμήματος.

Επίσης στο πρόγραμμα που ακολουθεί βλέπουμε ότι πάλι δηλώνονται δύο μεταβλητές με το ίδιο όνομα (`i`) και οι δύο στην ίδια συνάρτηση (`main()`) αλλά σε διαφορετικά τμήματα κώδικα.

```
/* local1.c */
/* Τοπικές μεταβλητές */
main()
{
  int i=10;

  printf("i=%d\n",i);
  {
    float i=3.14;
    printf("i=%f\n",i);
  }
  printf("i=%d",i);
}
```

Η έξοδος του προγράμματος είναι (δικαιολογήστε την):

```
i=10
i=3.1400
i=10
```

Ένα βασικό χαρακτηριστικό των τοπικών μεταβλητών είναι ότι αποκτούν υπόσταση μόνο κατά τη διάρκεια εκτέλεσης του τμήματος του κώδικα όπου δηλώνονται. Δηλαδή μία τοπική μεταβλητή *δημιουργείται* όταν ο έλεγχος του προγράμματος εισέρχεται στο αντίστοιχο τμήμα κώδικα και *καταστρέφεται* με την έξοδο από το τμήμα. Γι'αυτό το λόγο οι τοπικές μεταβλητές μίας συνάρτησης δε μπορούν να διατηρούν την τιμή τους μεταξύ των κλήσεων της συνάρτησης. (Τα παραπάνω δεν ισχύουν αν χρησιμοποιηθεί ο καθοριστής κατηγορίας αποθήκευσης `static`, βλέπε § 7.5). Αν στις τοπικές μεταβλητές δε δώσουμε αρχική τιμή, το περιεχόμενό τους είναι απροσδιόριστο. Οι *τυπικές παράμετροι* μίας συνάρτησης συμπεριφέρονται ως τοπικές μεταβλητές.

Σε αντίθεση με τις τοπικές μεταβλητές, οι *καθολικές* (global) μεταβλητές είναι γνωστές σε ολόκληρο το αρχείο του προγράμματος και μπορούν να χρησιμοποιούνται από οποιοδήποτε τμήμα κώδικα. Η εμβέλειά τους είναι καθολική και ο *χρόνος ζωής* τους είναι ο χρόνος εκτέλεσης του προγράμματος. Ας δούμε το παρακάτω πρόγραμμα, όπου *ορίζεται* η μεταβλητή `count` έξω από όλες τις συναρτήσεις:

```
/* external.c */
/* Καθολικές μεταβλητές */
int count;   /* η count είναι καθολική */
main()
{
  count=100;

  printf("Στην main() count=%d\n",count);  /* εμφανίζει το 100 */
  f1();
}

/* f1() */
f1()
{
  int temp;
  temp=count;

  f2();
  printf("Στην f1()   count=%d\n",count);  /* εμφανίζει το 100 */
}

/* f2() */
f2()
{
  int count=200;
  printf("Στην f2()   count=%d\n",count);  /* εμφανίζει το 200 */
}
```

Η έξοδος του προγράμματος είναι:

```
Στην main() count=100
Στην f2()   count=200
Στην f1()   count=100
```

Παρόλο που ούτε η `main()` ούτε η `f1()` έχουν δηλώσει τη μεταβλητή `count`, ωστόσο μπορούν να τη χρησιμοποιούν. Όμως, η `f2()` έχει δηλώσει μία τοπική μεταβλητή με όνομα `count`. Όταν αναφέρεται στην `count`, η `f2()` αναφέρεται στην τοπική της μεταβλητή, όχι στην καθολική.

Οι καθολικές μεταβλητές ονομάζονται επίσης και *εξωτερικές* (external) διότι ορίζονται έξω από όλες τις συναρτήσεις. Σε αντίθεση με τις τοπικές μεταβλητές, οι εξωτερικές διατηρούν την υπόστασή τους μόνιμα, έτσι διατηρούν τις τιμές τους ακόμα και μετά την επιστροφή της συνάρτησης που όρισε την αρχική τιμή τους. Αν στις εξωτερικές μεταβλητές δε δώσουμε αρχική τιμή παίρνουν αυτόματα την τιμή μηδέν.

Είναι σημαντικό να κάνουμε διάκριση ανάμεσα στη *δήλωση* (declaration) και τον *ορισμό* (definition) μιας εξωτερικής μεταβλητής. Μία δήλωση ανακοινώνει το

όνομα και τις ιδιότητες μίας μεταβλητής (κυρίως τον τύπο της), ενώ ένας ορισμός προκαλεί και δέσμευση χώρου στη μνήμη. Γενικά έχουμε:

ορισμός=δήλωση+δέσμευση μνήμης

Μέσα σ'ένα πρόγραμμα μπορούμε να διακρίνουμε τους ορισμούς από τις δηλώσεις. Αν οι γραμμές:

```
int i;
double f;
```

εμφανίζονται έξω από οποιαδήποτε συνάρτηση, *ορίζουν* τις εξωτερικές μεταβλητές `i` και `f` δεσμεύοντας μνήμη γι'αυτές και δηλώνοντάς τες για το υπόλοιπο αρχείο του πηγαίου κώδικα. Από την άλλη οι:

```
extern int i;
extern double f;
```

χρησιμοποιούν τη δεσμευμένη λέξη `extern` για να *δηλώσουν*, για το υπόλοιπο αρχείο πηγαίου κώδικα τις εξωτερικές μεταβλητές `i` και `f`, αλλά δε δημιουργούν τις μεταβλητές, ούτε δεσμεύουν χώρο στη μνήμη.

Στην περίπτωση που ο πηγαίος κώδικας αποτελείται από περισσότερα του ενός αρχεία, θα πρέπει να υπάρχει μόνο ένας *ορισμός* μίας εξωτερικής μεταβλητής σε ένα αρχείο, για όλα τα αρχεία· τα άλλα αρχεία μπορούν να περιέχουν δηλώσεις `extern` για την προσπέλασή της. Δες και § 7.5.

7.5 ΚΑΘΟΡΙΣΤΕΣ ΚΑΤΗΓΟΡΙΑΣ ΑΠΟΘΗΚΕΥΣΗΣ

Οι *καθοριστές κατηγορίας αποθήκευσης* (storage class specifiers) είναι λέξεις οι οποίες χρησιμοποιούνται ως προθέματα στις δηλώσεις των μεταβλητών και αποδίδουν στις μεταβλητές της C ένα χαρακτηριστικό που καλείται *κατηγορία αποθήκευσης*. Η κατηγορία αποθήκευσης καθορίζει δύο χαρακτηριστικά της μεταβλητής: το *χρόνο ζωής* και την *εμβέλειά της* (ή ορατότητα). Οι καθοριστές κατηγορίας αποθήκευσης είναι οι: `auto`, `register`, `extern`, `static`.

auto

Όλες οι μεταβλητές που χρησιμοποιήθηκαν ως τώρα, εκτός από το τελευταίο πρόγραμμα (`external.c`), δηλώνονταν μέσα σε μία συνάρτηση. Οι μεταβλητές που δηλώνονται μέσα σε μία συνάρτηση ή ένα τμήμα κώδικα (μπλοκ) καθώς και οι τυπικές παράμετροι μίας συνάρτησης (τοπικές) είναι εξ ορισμού *αυτόματες*, κι έτσι ο καθοριστής `auto` παραλείπεται γι'αυτές τις μεταβλητές.

Οι αυτόματες μεταβλητές δημιουργούνται όταν καλείται η συνάρτηση που τις περιέχει και καταστρέφονται όταν ο έλεγχος επιστρέφει στη συνάρτηση που έκανε την κλήση. Έτσι δε διατηρούν την τιμή τους μεταξύ διαδοχικών κλήσεων της συνάρτησης. Αν κατά τη δήλωση των αυτόματων μεταβλητών τους αποδίδεται και μία αρχική τιμή, αυτή την αρχική τιμή την παίρνουν κάθε φορά που εκτελείται το τμήμα του κώδικα όπου δηλώνονται. Για παράδειγμα, έστω το παρακάτω πρόγραμμα:

```
/* auto.c */
/* Αυτόματες μεταβλητές */
main()
{
  int i;

  for(i=1;i<=5;i++)
    printf("%d  %d\n",i,f(i));
}

/* f() */
f(int x)
{
  int s=100;  /* αυτόματη μεταβλητή */

  return(s+=x);
}
```

Η έξοδος του προγράμματος είναι:

```
1  101
2  102
3  103
4  104
5  105
```

Αν δε δοθεί αρχική τιμή σε μία μεταβλητή `auto`, τότε το περιεχόμενό της είναι απροσδιόριστο.

register

Ο καθοριστής `register` είναι ισοδύναμος με τον καθοριστή `auto` αλλά υποδεικνύει στο μεταγλωττιστή ότι η μεταβλητή (που πρέπει να είναι ακεραίου τύπου ή χαρακτήρα) θα χρησιμοποιηθεί συχνά (π.χ. ως μετρητής σε μία εντολή for). Ο σκοπός είναι οι μεταβλητές `register` να αποθηκεύονται σε καταχωρητές (registers) της Κεντρικής Μονάδας Επεξεργασίας, αν αυτό είναι δυνατό, διαφορετικά αντιμετωπίζονται σαν αυτόματες μεταβλητές.

Η αποθήκευση των μεταβλητών σε καταχωρητές, αυξάνει την ταχύτητα. Ο καθοριστής `register` μπορεί να εφαρμόζεται μόνο σε τοπικές μεταβλητές. Αν δε δοθεί αρχική τιμή σε μία μεταβλητή `register`, τότε το περιεχόμενό της είναι απροσδιόριστο.

extern

Μεγάλα προγράμματα συνήθως αποτελούνται από πολλά αρχεία πηγαίων προγραμμάτων. Συχνά θέλουμε μία μεταβλητή να είναι γνωστή σε περισσότερα από ένα αρχεία. Στην περίπτωση αυτή η μεταβλητή ορίζεται ως καθολική σε ένα μόνο αρχείο πηγαίου προγράμματος και στη συνέχεια τα υπόλοιπα αρχεία τη δηλώνουν κάνοντας χρήση του καθοριστή `extern`.

Δηλαδή ο καθοριστής `extern` χρησιμοποιείται για να δηλώνει μία μεταβλητή η οποία έχει οριστεί σε άλλο αρχείο του προγράμματος και είναι εξωτερική

μεταβλητή (δηλαδή καθολική). Χρησιμεύει στο να κάνει μία εξωτερική μεταβλητή ορατή σε ένα αρχείο εκτός του αρχείου στο οποίο ορίστηκε. Π.χ. αν στο αρχείο `file1.c` υπάρχει ο κώδικας:

```
/* file1.c */
int var1;  /* αυτός ο ορισμός δεσμεύει μνήμη για την var1 */
main()
{
  ...
}
```

τότε για να είναι η μεταβλητή `var1` ορατή στο αρχείο `file2.c` πρέπει να δηλωθεί ως `extern`:

```
/* file2.c */
function1()
{
  extern int var1;  /* αυτή η δήλωση κάνει ορατή την var1 */
  ...
}
```

Αν ο ορισμός μιας εξωτερικής μεταβλητής συναντηθεί στο αρχείο πηγαίου κώδικα πριν χρησιμοποιηθεί από μία συγκεκριμένη συνάρτηση η οποία υπάρχει στο ίδιο αρχείο, δεν υπάρχει ανάγκη να δηλωθεί ως `extern` μέσα σ'αυτή τη συνάρτηση. Μάλιστα είναι κοινή πρακτική να τοποθετούνται οι ορισμοί όλων των εξωτερικών μεταβλητών στην αρχή του αρχείου πηγαίου κώδικα, και μετά να παραλείπονται όλες οι δηλώσεις `extern`. Π.χ.:

```
/* abc.c */
int var2;
main()
{
  var2=13;
  ...
}

function2()
{  /* δεν είναι απαραίτητο να δηλωθεί extern int var2 */
  printf("%d\n",var2);
}
```

Αν δε δοθεί αρχική τιμή σε μία εξωτερική μεταβλητή, τότε παίρνει αυτόματα αρχική τιμή μηδέν.

static

Ο καθοριστής `static` χρησιμοποιείται για να ορίσει στατικές μεταβλητές στις ακόλουθες δύο περιπτώσεις:

(i) Στον ορισμό μεταβλητών έξω από κάθε συνάρτηση (εξωτερικές στατικές). Στην περίπτωση αυτή η εξωτερική μεταβλητή που έχει καθοριστεί ως `static` έχει εμβέλεια μόνο εντός του αρχείου που έχει οριστεί. Δε μπορεί να χρησιμοποιηθεί από άλλο αρχείο που θέλοντας να έχει πρόσβαση σ'αυτή τη δηλώνει ως `extern`, δηλαδή οι εξωτερικές στατικές μεταβλητές αποτελούν ιδιοκτησία του αρχείου όπου ορίστηκαν. Έτσι ο καθοριστής `static` παρέχει έναν τρόπο *απόκρυψης* μεταβλητών.

(ii) Στη δήλωση μεταβλητών μέσα σε μία συνάρτηση ή ένα τμήμα κώδικα (τοπικές στατικές). Σ'αυτήν την περίπτωση η τοπική μεταβλητή που έχει δηλωθεί ως `static` έχει βέβαια τοπική εμβέλεια, αλλά *διατηρεί* την τιμή της και μετά την έξοδο από το τμήμα όπου έχει δηλωθεί. Αντίθετα από τις αυτόματες μεταβλητές, οι εσωτερικές στατικές μεταβλητές παίρνουν την αρχική τους τιμή (αν στον ορισμό τους υπάρχει και απόδοση αρχικής τιμής) μόνο μία φορά, στην αρχή της εκτέλεσης του προγράμματος. Έτσι αν θεωρήσουμε το παρακάτω πρόγραμμα, που είναι το ίδιο με το `auto.c` που είδαμε προηγουμένως, μόνο που τώρα υπάρχει ο καθοριστής `static` στην δήλωση της μεταβλητής `s`:

```
/* static.c */
/* Αυτόματες μεταβλητές */
main()
{
  int i;

  for(i=1;i<=5;i++)
    printf("%d  %d\n",i,f(i));
}

/* f() */
f(int x)
{
  static int s=100;  /* αυτόματη μεταβλητή */

  return(s+=x);
}
```

Θα πάρουμε στην έξοδο (γιατί ;):

```
1  101
2  103
3  106
4  110
5  115
```

Αν δε δοθεί αρχική τιμή σε μία μεταβλητή `static`, παίρνει αυτόματα αρχική τιμή μηδέν.

Ο Πίνακας 7.1 δείχνει περιληπτικά το χρόνο ζωής και την εμβέλεια των μεταβλητών σε σχέση με την περιοχή στην οποία δηλώνονται και με τους καθοριστές κατηγορίας αποθήκευσης που εφαρμόζονται σ'αυτές.

7.6 ΠΡΟΣΔΙΟΡΙΣΤΕΣ ΤΥΠΩΝ

Ο τύπος μιας μεταβλητής μπορεί να έχει τους πρόσθετους προσδιορισμούς `const`, `volatile`. Αν σε μια μεταβλητή εφαρμοστεί ο προσδιοριστής `const` αυτή μπορεί να πάρει αρχική τιμή αλλά δεν επιτρέπεται καμία επόμενη αλλαγή της τιμής της, π.χ.:

```
const float pi=3.14159;
```

Ο προσδιοριστής `volatile` χρησιμοποιείται για να δηλώνει στο μεταγλωττιστή ότι μία μεταβλητή μπορεί να αλλάζει το περιεχόμενό της με τρόπους που δεν ορίζονται ρητά από το πρόγραμμα. Για παράδειγμα, μεταβλητές που αλλάζουν από το υλικό, όπως ρολόγια πραγματικού χρόνου, διακοπές κ.λ.π. πρέπει να ορίζονται ως `volatile`.

Περιοχή Δήλωσης	***Καθοριστής Κατηγορίας Αποθήκευσης***	***Χρόνος Ζωής***	***Εμβέλεια (Ορατότητα)***
συνάρτηση	`auto` (εξ ορισμού)	συνάρτηση	συνάρτηση
συνάρτηση	`register`	συνάρτηση	συνάρτηση
συνάρτηση	`static`	πρόγραμμα	συνάρτηση
εξωτερικά	-	πρόγραμμα	μόνο στο αρχείο που ορίζεται
εξωτερικά ή σε συνάρτηση	`extern`	πρόγραμμα	στην περιοχή που δηλώνεται
εξωτερικά	`static`	πρόγραμμα	αρχείο

Πίνακας 7.1: *Κατηγορίες Αποθήκευσης*

7.7 ΔΗΛΩΣΕΙΣ ΜΕΤΑΒΛΗΤΩΝ

Με βάση τα όσα αναπτύχθηκαν στις προηγούμενες παραγράφους (§ 7.5, 7.6 το συντακτικό για τη δήλωση μιας μεταβλητής είναι:

(i) Ένας προαιρετικός προσδιοριστής τύπου (βλέπε § 7.6).

(ii) Ένας προαιρετικός καθοριστής κατηγορίας αποθήκευσης (βλέπε § 7.5).

(iii) Το όνομα του τύπου ακολουθούμενο από μία λίστα ενός ή περισσοτέρων αναγνωριστικών μεταβλητών χωρισμένων με κόμμα.

(iv) Μία προαιρετική απόδοση αρχικής τιμής, χρησιμοποιώντας τον τελεστή = και μία παράσταση.

Παραδείγματα δηλώσεων μεταβλητών:

```
int i,j,k;
register char alpha;
static float beta=1.234;
const auto double N=6.023E23;
```

7.8 ΑΝΑΔΡΟΜΗ

Είδαμε στα προηγούμενα ότι μία συνάρτηση μπορεί να κληθεί από οποιαδήποτε άλλη συνάρτηση του προγράμματος. Στην C μία συνάρτηση μπορεί να καλεί τον εαυτό της, άμεσα ή έμμεσα. Αν μία συνάρτηση, μέσα στο τμήμα των εντολών της, περιέχει μία κλήση του εαυτού της, τότε λέγεται *αναδρομική* (recursive).

Η ανάγκη της ύπαρξης αναδρομικών συναρτήσεων προέκυψε από το γεγονός ότι η χρήση της διαδικασίας της αναδρομής είναι ένα ουσιαστικό εργαλείο, το οποίο πολλές φορές χρησιμεύει για την απλοποίηση της λύσης ενός προβλήματος. Πολλά προβλήματα, κυρίως μαθηματικά, είναι από τον ορισμό τους αναδρομικά (αναδρομή είναι ο τρόπος με τον οποίο ορίζουμε κάτι μέσω του ίδιου του οριζόμενου), π.χ. το σύνολο των φυσικών αριθμών μπορεί να οριστεί ως εξής:

1. Ο αριθμός 1 είναι ένας φυσικός αριθμός.
2. Ο επόμενος ενός φυσικού αριθμού είναι ένας φυσικός αριθμός.

όπου ο *επόμενος* ενός φυσικού αριθμού λαμβάνεται προσθέτοντας τη μονάδα στον αριθμό αυτό. Η δύναμη αυτού του αναδρομικού ορισμού είναι η ικανότητά του να ορίζει ένα σύνολο, χρησιμοποιώντας μόνο δύο εντολές. Με τον ίδιο τρόπο, μία αναδρομική συνάρτηση μπορεί να προσδιορίζει ένα άπειρο πλήθος υπολογισμών χρησιμοποιώντας μόνο μερικές εντολές.

Η τεχνική της αναδρομής μπορεί να εφαρμοστεί αποτελεσματικά σε προβλήματα που έχουν την παρακάτω δομή:

1. Η λύση είναι εύκολη για ορισμένες ειδικές συνθήκες, οι οποίες καλούνται καταστάσεις τερματισμού.
2. Για οποιαδήποτε δεδομένη κατάσταση, υπάρχουν καλά ορισμένοι κανόνες για τη μετάβαση σε μία νέα κατάσταση, η οποία είτε είναι μία κατάσταση τερματισμού ή οδηγεί τελικά σε μία τέτοια κατάσταση (αναδρομικό βήμα).

Για να γίνουν κατανοητά τα παραπάνω, θα δούμε ένα συγκεκριμένο παράδειγμα: Έστω ότι μας ζητάνε να υπολογίσουμε το παραγοντικό (`N!`) ενός μη-αρνητικού ακέραιου αριθμού (`N`). Το παραγοντικό ορίζεται ως εξής:

Αν `N=0`, τότε `N!=1`
Για `N>0`, τότε `N!=1*2*3*...*N`

Με βάση αυτόν τον ορισμό, η παρακάτω συνάρτηση χρησιμοποιεί μία επαναληπτική διαδικασία (εντολή `for`) για τον υπολογισμό του παραγοντικού (συγκρίνετε με τη διαδικασία που χρησιμοποιείται στο πρόγραμμα `fact2.c` της § 6.4):

```
unsigned long int factorial(int n)  /* μη-αναδρομική */
{
  int t;
  unsigned long int answer=1;

  for(t=1;t<=n;t++) answer*=t;
  return(answer);
}
```

Το παραγοντικό όμως, μπορεί να οριστεί και αναδρομικά:

Αν `N=0`, τότε `N!=1` (κατάσταση τερματισμού)
Για `N>0`, τότε `N!=N*(N-1)!` (αναδρομικό βήμα)

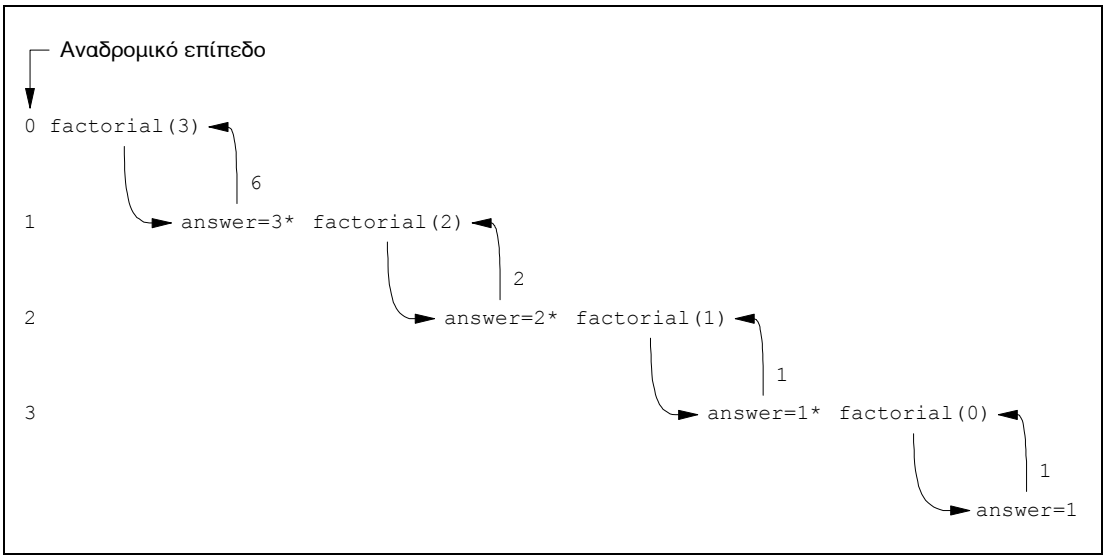
Με βάση τον παραπάνω ορισμό και όπως φαίνεται από το αναδρομικό βήμα, το πρόβλημα εύρεσης του παραγοντικού ενός ακέραιου αριθμού ανάγεται στην εύρεση του παραγοντικού του προηγούμενου μικρότερου ακέραιου. Είναι φανερό ότι το N-1 είναι πλησιέστερο προς το 0 από ότι είναι το N. Η αναδρομική συνάρτηση υπολογισμού του παραγοντικού, είναι:

```
unsigned long int factorial(int n)  /* αναδρομική */
{
  unsigned long int answer;

  if (n==0)
    answer=1;
  else
    answer=n*factorial(n-1);
  return(answer);
}
```

Ας παρακολουθήσουμε τι συμβαίνει σε μία κλήση της προηγούμενης συνάρτησης. Κάθε νέα κλήση της `factorial()` αυξάνει το αναδρομικό επίπεδο. Ο μεταγλωττιστής αποθηκεύει σε *στοίβα* (stack) τις τιμές της παραμέτρου `n` της συνάρτησης όπως επίσης και οποιασδήποτε άλλης τοπικής μεταβλητής, πριν από κάθε νέα κλήση. Όταν ολοκληρωθεί η διαδικασία εκτέλεσης μίας κλήσης, ο έλεγχος του προγράμματος επιστρέφει στην αμέσως επόμενη εντολή μετά την κλήση. Κατά τη διάρκεια της κλήσης υπολογίζεται η έκφραση `n-1` και καταχωρείται σαν νέα τιμή της παραμέτρου `n`. Τελικά η παράμετρος `n` θα γίνει μηδέν και η μεταβλητή `answer` θα πάρει τιμή 1, με την τελευταία κλήση της συνάρτησης `factorial()` (προς τα πίσω διαδικασία). Στη συνέχεια ο μεταγλωττιστής επιστρέφει από κάθε αναδρομικό επίπεδο και υπολογίζει την τρέχουσα τιμή της `answer` πολλαπλασιάζοντας την παράμετρο που έχει αποθηκεύσει σ'αυτό το επίπεδο με την τιμή της `answer` του προηγούμενου επιπέδου. Με αυτό τον τρόπο επιστρέφουμε στο αρχικό επίπεδο (προς τα εμπρός διαδικασία).

Για παράδειγμα, μία κλήση `factorial(3)` περιέχει, κατά τη διάρκεια της εκτέλεσής της, μία κλήση της `factorial(2)`. Για τον υπολογισμό της `factorial(2)` χρειάζεται ο υπολογισμός της `factorial(1)`, η οποία με τη σειρά της χρειάζεται την τιμή της `factorial(0)`, που είναι 1. Έτσι από τη στιγμή αυτή μπορεί να αρχίσει η προς τα εμπρός διαδικασία με τον υπολογισμό της `factorial(1)`, που τώρα μπορεί να υπολογιστεί σαν `1*1`. Στη συνέχεια η `factorial(2)` υπολογίζεται σαν `2*1` και τέλος η `factorial(3)` σαν `3*2`.

Στο Σχήμα 7.2 παρουσιάζεται η εκτέλεση της `factorial(3)`. Υπάρχουν 4 αναδρομικά επίπεδα (0-3) και τρεις αναδρομικές κλήσεις της `factorial()` συν την αρχική κλήση. Τα βέλη που έχουν φορά προς τα κάτω δείχνουν την έκφραση που προκύπτει μετά από την εφαρμογή κάθε αναδρομικού βήματος. Μετά την τελευταία αναδρομική κλήση (`factorial(0)`), η συνάρτηση πρέπει να επιστρέφει από κάθε κλήση με μία τιμή (την τιμή της `answer`). Η τιμή είναι εκείνη που προκύπτει από τον πολλαπλασιασμό του αποτελέσματος του προηγουμένου επιπέδου με την τιμή της παραμέτρου `n` στο τρέχον επίπεδο. Οι τιμές αυτές δηλώνονται δίπλα στα βέλη που έχουν φορά προς τα επάνω.



Σχήμα 7.2: *Παρουσίαση της εκτέλεσης της* `factorial(3)`

Το παρακάτω πρόγραμμα χρησιμοποιεί μία αναδρομική συνάρτηση (την `stackchar()`) η οποία δέχεται από το πληκτρολόγιο μία σειρά χαρακτήρων που τελειώνει με ENTER και την τυπώνει στην οθόνη με την αντίστροφη σειρά. Χρησιμοποιώντας την τεχνική της αναδρομής, διαβάζεται ένας χαρακτήρας σε κάθε αναδρομικό επίπεδο κατά την προς τα πίσω διαδικασία και στη συνέχεια ο χαρακτήρας αυτός τυπώνεται κατά την προς τα εμπρός διαδικασία:

```
/* revline */
/* Αντίστροφη εκτύπωση */
#include <stdio.h>
main()
{
  puts("Δώσε σειρά χαρακτήρων (για τέλος ENTER):");
  stackchar();
}

/* stackchar() */
/* Τυπώνει αντίστροφα μία σειρά χαρακτήρων */
stackchar()
{
  char c;

  c=getche();
  if(c!='\r')
    stackchar();
  putchar(c);
}
```

7.9 ΜΑΘΗΜΑΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΒΙΒΛΙΟΘΗΚΗΣ

Στο αρχείο-επικεφαλίδας `math.h` περιλαμβάνονται δηλώσεις μαθηματικών συναρτήσεων (εκτός της `abs()`, η οποία δηλώνεται στο `stdlib.h`), οι κυριότερες από τις οποίες είναι:

```
double cos(double x)           /* συν(x), το x σε ακτίνια     */
double sin(double x)           /* ημ(x),  το x σε ακτίνια     */
double tan(double x)           /* εφ(x),  το x σε ακτίνια     */
double exp(double x)           /* e^x                         */
double log(double x)           /* ln(x)                       */
double log10(double x)         /* log10(x)                    */
double pow(double x, double y) /* x^y                         */
double sqrt(double x)          /* √x                          */
double floor(double x)         /* floor(4.9)=4.0              */
double ceil(double x)          /* ceil(1.02)=2.0              */
int    abs(int x)              /* |x|, δηλώνεται στο stdlib.h */
double fabs(double x)          /* |x|                         */
double acos(double x)          /* τοξσυν(x),  το x ∈ [-1..1]  */
double asin(double x)          /* τοξημ(x),   το x ∈ [-1..1]  */
double atan(double x)          /* τοξεφ(x)                    */
double cosh(double x)          /* υπερσυν(x), το x σε ακτίνια */
double sinh(double x)          /* υπερημ(x),  το x σε ακτίνια */
double tanh(double x)          /* υπερεφ(x),  το x σε ακτίνια */
```

7.10 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί μία συνάρτηση που να δέχεται ως παραμέτρους δύο πραγματικούς και να επιστρέφει το μεγαλύτερο απ'αυτούς.

```
/* max() */
/* Επιστρέφει το μεγαλύτερο από δύο αριθμούς */
float max(float n1,float n2)
{
  if (n1>n2) return(n1);
  else return(n2);
}
```

2. Γράψτε μία συνάρτηση που να επιστρέφει το μεγαλύτερο από τρεις πραγματικούς αριθμούς που θα δέχεται ως παραμέτρους. Στη συνέχεια χρησιμοποιήστε αυτή τη συνάρτηση σε ένα πρόγραμμα που θα βρίσκει το μεγαλύτερο από πέντε αριθμούς.

```
/* max5.c */
/* Εύρεση μεγαλύτερου από πέντε αριθμούς */
float max(float,float,float);
main()
{
  float a,b,c,d,e,temp,maximum;
```

```
  printf("\nΔώσε πέντε αριθμούς: ");
  scanf("%f %f %f %f %f", &a,&b,&c,&d,&e);
  temp=max(a,b,c);
  maximum=max(temp,d,e);
  printf("\nΜεγαλύτερος είναι ο %f",maximum);
}

/* max() */
/* Επιστρέφει το μεγαλύτερο από τρεις αριθμούς */
float max(float n1,float n2,float n3)
{
  float m;

  if(n1>n2) m=n1;
  else m=n2;
  if(n3>m) m=n3;
  return(m);
}
```

3. Να γραφεί μία συνάρτηση η οποία θα δέχεται από το πληκτρολόγιο την ώρα στη μορφή hh:mm (ώρες:λεπτά) και θα την επιστρέφει σε λεπτά. Στη συνέχεια να γίνει χρήση της συνάρτησης από ένα πρόγραμμα που να υπολογίζει τη διαφορά ανάμεσα σε δύο χρονικές στιγμές, σε λεπτά.

```
/* intimes.c */
/* Υπολογισμός χρονικής διαφοράς */
main()
{
  int mins1,mins2;

  puts("Δώσε την 1η χρονική στιγμή (μορφή hh:mm)");
  mins1=getmins();
  puts("Δώσε την 2η χρονική στιγμή (μορφή hh:mm)");
  mins2=getmins();
  printf("Διαφορά: %d λεπτά", mins2-mins1);
}

/* getmins() */
/* Δέχεται ώρα σε μορφή hh:mm και την επιστρέφει σε λεπτά */
getmins()
{
  int hours,minutes;

  scanf("%d:%d",&hours,&minutes);
  return(hours*60+minutes);
}
```

4. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο τρεις ακέραιους αριθμούς που θα δηλώνουν τις μηνιαίες πωλήσεις ενός υλικού από τους πωλητές Γιάννη, Μαρία, Ελένη αντίστοιχα. Έπειτα, ανάλογα με το ύψος των πωλήσεων, να τυπώνεται για κάθε πωλητή μία οριζόντια γραμμή ανάλογου μήκους, χρησιμοποιώντας το χαρακτήρα '='.

```
/* bargraph.c */
/* Παράσταση πωλήσεων με οριζόντιες μπάρες */
main()
{
  int x,y,z;

  printf("Δώσε τις πωλήσεις των τριών πωλητών (ακέραιοι): ");
  scanf("%d %d %d",&x,&y,&z);
  printf("Γιάννης\t");
  bar(x);
  printf("Μαρία\t");
  bar(y);
  printf("Ελένη\t");
  bar(z);
}

/* bar() */
/* Ζωγραφίζει οριζόντια μπάρα */
bar(int score)
{
  int j;

  for(j=1;j<=score;j++) printf("=");
  printf("\n");
}
```

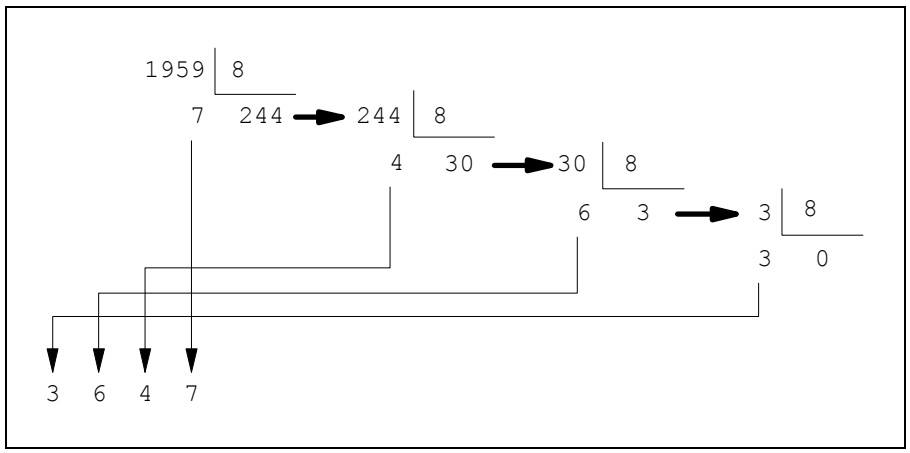
5. Γράψτε ένα πρόγραμμα που να υπολογίζει τα εμβαδά των εξής σχημάτων: ορθογωνίου παραλληλογράμμου, κύκλου, τριγώνου και τραπεζίου. Η επιλογή για το ποιου σχήματος το εμβαδόν θα υπολογιστεί, να γίνεται μέσα από ένα menu επιλογών. Οι επιλογές του menu να καλούν κατάλληλες συναρτήσεις που να υπολογίζουν το αντίστοιχο εμβαδόν.

```
/* areas1.c */
#define PI 3.14159
float square(float,float);
float circle(float);
float triangle(float,float);
float trapezoid(float,float,float);
main()
{
  char select;
  float l,w,r,h,b,b1,b2;

  do
  {
    clrscr();
    puts("1. Εμβαδόν ορθογωνίου");
    puts("2. Εμβαδόν κύκλου");
    puts("3. Εμβαδόν τριγώνου");
    puts("4. Εμβαδόν τραπεζίου");
    puts("5. Εξοδος");
    printf("   Επιλογή: "); scanf("%c",&select);
    switch(select)
```

```
    {
      case '1': printf("Μήκος : "); scanf("%f",&l);
                printf("Πλάτος: "); scanf("%f",&w);
                printf("Εμβαδόν: %f",square(l,w));
                getch(); break;
      case '2': printf("Ακτίνα: "); scanf("%f",&r);
                printf("Εμβαδόν: %f",circle(r));
                getch(); break;
      case '3': printf("Βάση: "); scanf("%f",&b);
                printf("Ύψος: "); scanf("%f",&h);
                printf("Εμβαδόν: %f",triangle(b,h));
                getch(); break;
      case '4': printf("Βάση       : "); scanf("%f",&b1);
                printf("Βάση μεγάλη: "); scanf("%f",&b2);
                printf("Ύψος       : "); scanf("%f",&h);
                printf("Εμβαδόν: %f",trapezoid(b1,b2,h));
                getch(); break;
    }
  }
  while(select!='5');
}

/* square() */
/* Εμβαδόν ορθογωνίου */
float square(float a,float b)
{
  return(a*b);
}

/* circle() */
/* Εμβαδόν κύκλου */
float circle(float a)
{
  return(a*a*PI);
}

/* triangle() */
/* Εμβαδόν τριγώνου */
float triangle(float a,float b)
{
  return(a*b/2);
}

/* trapezoid() */
/* Εμβαδόν τραπεζίου */
float trapezoid(float a,float b,float c)
{
  return((a+b)*c/2);
}
```

6. Να γράψετε μία συνάρτηση που να μετατρέπει έναν ακέραιο αριθμό του δεκαδικού συστήματος (βάση το 10) σε αριθμό σε άλλη *βάση*. Η συνάρτηση να δέχεται ως παραμέτρους τον αρχικό αριθμό και τη βάση (η βάση να είναι ακέραιος από 2 ως 9) και να επιστρέφει το μετασχηματισμένο αριθμό. Στη

συνέχεια, η συνάρτηση αυτή να χρησιμοποιηθεί σε ένα πρόγραμμα που θα δέχεται από το πληκτρολόγιο τον αριθμό στο δεκαδικό σύστημα και τη νέα βάση και τέλος να εμφανίζει στην οθόνη το μετασχηματισμένο αριθμό.
Ο αλγόριθμος είναι ο εξής:
Διαιρούμε τον αριθμό που πρέπει να μετατραπεί δια της βάσης και το μεν ακέραιο υπόλοιπο είναι το λιγότερο σημαντικό ψηφίο του νέου αριθμού, το δε πηλίκο θεωρείται ότι είναι ο δεκαδικός αριθμός που πρέπει να μετασχηματιστεί. Η διαδικασία αυτή σταματάει όταν προκύψει πηλίκο μηδέν. Έτσι π.χ. ο αριθμός 1959 του δεκαδικού συστήματος (βάση 10) μετασχηματίζεται στον 3647 του οκταδικού συστήματος (βάση 8), όπως φαίνεται στο Σχήμα 7.3.



Σχήμα 7.3: *Μετατροπή του 1959 στο οκταδικό*

```
/* convert.c */
/* Αλλαγή βάσης ακεραίου */
#include <math.h>
unsigned long int convert(int,int);
main()
{
  int number,base;

  puts("Δώσε αριθμό και βάση");
  scanf("%d %d",&number,&base);
  printf("Ο %d με βάση %d: %lu",number,base,convert(number,base));
}

/* convert() */
/* Αλλάζει τη βάση ενός ακεραίου */
unsigned long int convert(int n,int b)
{
  int p,y,i=0;
  unsigned long int result=0;

  while(n!=0)
  {
    y=n%b;
    n=n/b;
    result=y*pow(10,i)+result;
```

```
    i++;
  }
  return(result);
}
```

7. Να γραφεί συνάρτηση που να βρίσκει το Μέγιστο Κοινό Διαιρέτη (Μ.Κ.Δ.) δύο ακεραίων. Στη συνέχεια να γίνει χρήση της συνάρτησης αυτής σε κατάλληλο πρόγραμμα.
Για την εύρεση του Μ.Κ.Δ. να χρησιμοποιηθεί ο αλγόριθμος του Ευκλείδη:

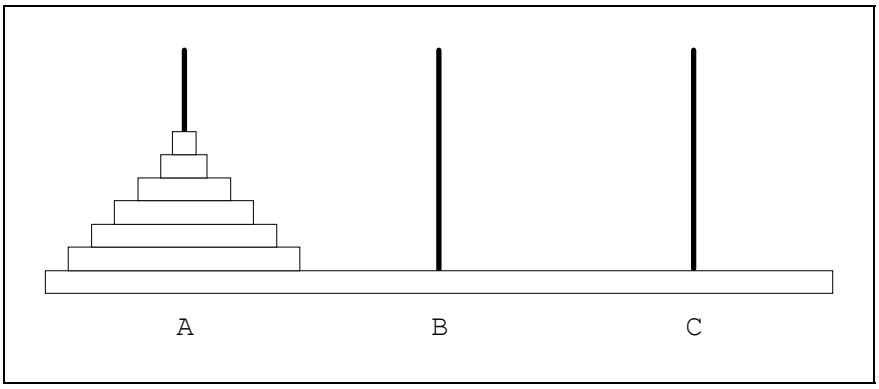
$$\text{ΜΚΔ}(m,n) = \begin{cases} \text{ΜΚΔ}(n, m\%n) & \text{αν } n \neq 0 \\ m & \text{αν } n = 0 \end{cases}$$

```
/* mkd.c */
/* Εύρεση Μ.Κ.Δ. δύο ακεραίων */
main()
{
  int a,b;

  puts("Δώσε δύο ακεραίους:");
  scanf("%d %d",&a,&b);
  printf("Ο Μ.Κ.Δ. είναι %d",mkd(a,b));
}

/* mkd() */
/* Επιστρέφει τον Μ.Κ.Δ. των παραμέτρων της */
mkd(int x,int y)
{
  int answer;

  if(y!=0)
    answer=mkd(y,x%y);
  else
    answer=x;
  return(answer);
}
```

8. Ο μύθος λέει ότι στους καλόγερους του ναού Μπράμα δόθηκε το ακόλουθο πρόβλημα (των πύργων του Ανόι): Έχουμε μία βάση με τρεις στύλους, σε έναν από τους οποίους έχουν τοποθετηθεί 64 δίσκοι, όπως στο Σχήμα 7.4 (στο οποίο φαίνονται μόνο 6 δίσκοι). Το τέλος του κόσμου θα έρθει αν καταφέρουμε να τοποθετήσουμε όλους τους δίσκους σε έναν άλλο στύλο, ακολουθώντας τους παρακάτω κανόνες:

1. Όταν μεταφέρεται ένας δίσκος πρέπει να τοποθετηθεί σε έναν από τους τρεις στύλους.
2. Μόνο ένας δίσκος μπορεί να μεταφερθεί κάθε φορά και πρέπει να είναι από αυτούς που βρίσκονται στην κορυφή.
3. Ένας μεγαλύτερος δίσκος δε μπορεί να τοποθετηθεί πάνω από ένα μικρότερο.

Να γραφεί πρόγραμμα που να επιλύει το πρόβλημα των δίσκων του Ανόι για n δίσκους, όπου το n θα δίνεται από το πληκτρολόγιο.



Σχήμα 7.4: *Οι δίσκοι του Ανόι*

```
/* hanoi.c */
/* Πρόβλημα των δίσκων του Ανόι */
unsigned long int i=0;
main()
{
  int number;

  puts("Δώσε αριθμό δίσκων:");
  scanf("%d",&number);
  move(number,'A','B','C');
  printf("\nΠραγματοποιήθηκαν %lu μετακινήσεις δίσκων",i);
}

/* move() */
/* Αναδρομική συνάρτηση μετακίνησης δίσκων */
move(int n,char arxikos, char boithitikos,char telikos)
{
  if(n==1)
  {
    printf("\nΝα κινηθεί ο δίσκος από τον %c στον %c", arxikos,telikos);
    i++;
  }
  else
  {
    move(n-1,arxikos,telikos,boithitikos);
    move(1,arxikos,' ',telikos);
    move(n-1,boithitikos,arxikos,telikos);
  }
}
```

7.11 ΑΣΚΗΣΕΙΣ

1. Γράψτε μία συνάρτηση `rectang(length,width)` που να εκτυπώνει ένα ορθογώνιο παραλληλόγραμμο, που οι πλευρές του αποτελούνται από αστερίσκους. Το μήκος και το πλάτος του παραλληλογράμμου είναι `length` και `width` αντίστοιχα.

2. Να γραφεί μία συνάρτηση που να υπολογίζει το sin(x) χρησιμοποιώντας τον τύπο:

   $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$

   Ο αριθμός των όρων που θα χρησιμοποιούνται να δίνεται ως παράμετρος στην κλήση της συνάρτησης. Π.χ. `sin(x,3)` σημαίνει να υπολογιστεί το sin(x) με τη χρήση των τριών πρώτων όρων.

3. Ο αναδρομικός ορισμός της δύναμης a^n για $n \in Z^+$ και $a \in R$ είναι:

   $$a^n = \begin{cases} 1 & \text{αν } n=0 \\ a*a^{n-1} & \text{αν } n>0 \end{cases}$$

   Γράψτε αναδρομική συνάρτηση `power(a,n)` που να επιστρέφει τη δύναμη a^n.

4. Να γραφεί αναδρομική συνάρτηση για τον υπολογισμό του n-οστού όρου της ακολουθίας Fibonacci. Οι όροι της ακολουθίας Fibonacci ορίζονται από τις σχέσεις:

   $F_1 = 1$, $F_2 = 1$ και $F_N = F_{N+1} + F_{N+2}$ για $N>2$

5. Γράψτε μία μη-αναδρομική και μία αναδρομική συνάρτηση που να υπολογίζει το άθροισμα, `S`, μιας σειράς ακεραίων θετικών αριθμών από το 1 μέχρι το `N`, όπου `N` γνωστός θετικός ακέραιος αριθμός:

   Μη αναδρομικός τύπος: `S(N)=1+2+3+...N`

   Αναδρομικός τύπος: Αν `N=1`, `S(1)=1`
   Για `N>1`, `S(N)=N+S(N-1)`

6. Να γραφεί μία συνάρτηση που να δέχεται ένα θετικό ακέραιο αριθμό και να τον επιστρέφει με τα ψηφία του σε αντίστροφη σειρά (το τελευταίο να γίνει πρώτο, το προ-τελευταίο να γίνει δεύτερο κ.ο.κ., π.χ. το 123 να το επιστρέφει ως 321).

7. Να γραφεί μία συνάρτηση η οποία να δέχεται έναν ακέραιο αριθμό και να επιστρέφει το πλήθος των ψηφίων του.

ΚΕΦΑΛΑΙΟ 8
ΠΙΝΑΚΕΣ ΚΑΙ ΑΛΦΑΡΙΘΜΗΤΙΚΑ

8.1 ΠΙΝΑΚΕΣ

Πίνακας (array) είναι ένας δομημένος τύπος δεδομένων που χρησιμοποιείται για την καταχώρηση δεδομένων του *ίδιου τύπου* σε διαδοχικές θέσεις της μνήμης. Ο τύπος των δεδομένων του πίνακα λέγεται *βασικός τύπος*. Επειδή τα δεδομένα του πίνακα είναι όλα του ίδιου τύπου, ο πίνακας χαρακτηρίζεται ως *ομογενής δομημένος τύπος*.

8.1.1 ΔΗΛΩΣΗ ΠΙΝΑΚΑ

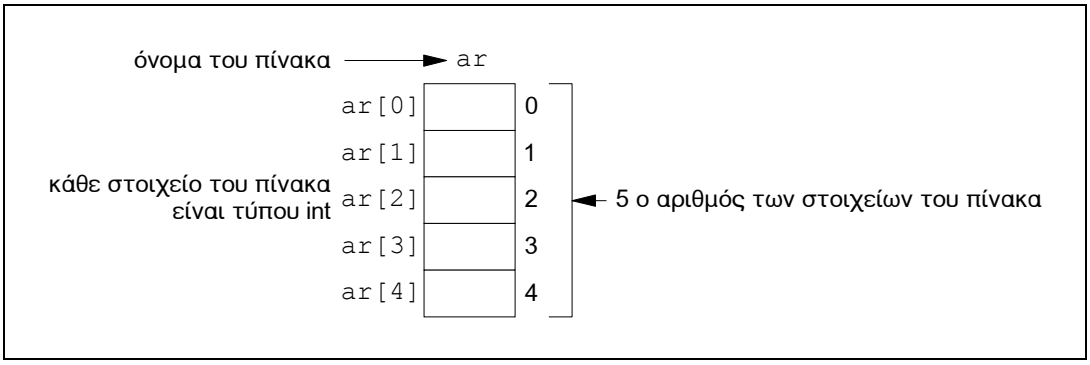
Η γενική μορφή της εντολής δήλωσης ενός πίνακα είναι:

τύπος αναγνωριστικό[μέγεθος];

όπου *τύπος* είναι ο *βασικός τύπος* (base type) του πίνακα, και το *μέγεθος* είναι ένας ακέραιος αριθμός που καθορίζει τον αριθμό των στοιχείων που θα περιέχει ο πίνακας. Π.χ. η εντολή:

```
int ar[5];
```

δηλώνει έναν πίνακα ακεραίων με όνομα `ar`, ικανό να αποθηκεύσει 5 ακέραιες μεταβλητές (καθεμιά από τις μεταβλητές του πίνακα ονομάζεται *στοιχείο*). Το Σχήμα 8.1 είναι μία σχηματική αναπαράσταση του πίνακα `ar`.



Σχήμα 8.1: *Σχηματική αναπαράσταση πίνακα*

8.1.2 ΑΝΑΦΟΡΑ ΣΤΑ ΣΤΟΙΧΕΙΑ ΕΝΟΣ ΠΙΝΑΚΑ

Για να αναφερθούμε στο κάθε στοιχείο του πίνακα ξεχωριστά χρησιμοποιούμε ένα *δείκτη-πίνακα* (subscript). Ο δείκτης-πίνακα είναι ένας ακέραιος αριθμός, ή μία ακέραια παράσταση, που γράφεται μέσα σε αγκύλες (`[]`) που ακολουθούν το αναγνωριστικό του πίνακα. Ας προσέξουμε ότι αυτός ο αριθμός έχει διαφορετική σημασία όταν *αναφερόμαστε* σ'ένα στοιχείο του πίνακα απ'ό,τι όταν *δηλώνουμε*

τον πίνακα, όπου ο αριθμός στις αγκύλες καθορίζει το *μέγεθος* του πίνακα. Όταν αναφερόμαστε σ'ένα στοιχείο του πίνακα, αυτός ο αριθμός καθορίζει τη *θέση* του στοιχείου.

Όλα τα στοιχεία του πίνακα αριθμούνται *αρχίζοντας από το μηδέν*. Έτσι, π.χ. το τέταρτο στοιχείο του πίνακα `ar` που δηλώσαμε παραπάνω θα αναφερθεί ως:

```
ar[3]
```

και το τελευταίο στοιχείο του πίνακα ως (βλέπε Σχήμα 8.1):

```
ar[4]
```

Στο πρόγραμμα που ακολουθεί χρησιμοποιούμε έναν πίνακα `temper` για να αποθηκεύσουμε τις θερμοκρασίες των ημερών μίας εβδομάδας. Χρησιμοποιούμε μία εντολή `for` για την εισαγωγή των θερμοκρασιών στον πίνακα. Στη συνέχεια υπολογίζουμε το άθροισμα και τη μέση τιμή των θερμοκρασιών, την οποίαν εμφανίζουμε στην οθόνη:

```
/* temper.c */
/* Μέση τιμή των θερμοκρασιών μιας εβδομάδας */
#define SIZE 7
main()
{
  float temper[SIZE], sum=0;
  int day;

  for(day=0;day<SIZE;day++)
  {
    printf("Δώσε τη θερμοκρασία της %δης μέρας: ",day+1);
    scanf("%f",&temper[day]);
  }
  for(day=0;day<SIZE;day++)
    sum+=temper[day];
  printf("Η μέση τιμή είναι %.2f",sum/SIZE);
}
```

Χρειάζεται προσοχή όταν χειριζόμαστε πίνακες, γιατί η γλώσσα C δεν εκτελεί έλεγχο ορίων στους πίνακες. Έτσι τίποτα δε μας εμποδίζει να ξεπεράσουμε το τέλος ενός πίνακα. Αν ξεπεράσουμε το τέλος ενός πίνακα κατά τη λειτουργία απόδοσης τιμής, τα δεδομένα που θα αποδώσουμε θα τοποθετηθούν στη μνήμη *έξω από τον πίνακα*, πιθανόν σε θέσεις μνήμης που έχουν κρατηθεί για άλλες μεταβλητές, ή σε θέσεις μνήμης όπου υπάρχει αποθηκευμένο το ίδιο το πρόγραμμα. Αυτό θα οδηγήσει τουλάχιστον σε απρόσμενα αποτελέσματα και δε θα υπάρξει από το μεταγλωττιστή μήνυμα λάθους για να μας ειδοποιήσει για το τι συμβαίνει.

8.1.3 ΠΙΝΑΚΕΣ ΠΟΛΛΩΝ ΔΙΑΣΤΑΣΕΩΝ

Έως τώρα έχουμε δει πίνακες με μία μόνο διάσταση: με ένα μόνο δείκτη-πίνακα. Είναι όμως δυνατόν, οι πίνακες να έχουν περισσότερες της μίας διαστάσεις. Ένας πίνακας δύο διαστάσεων μπορεί να θεωρηθεί ως ένας πίνακας μίας διάστασης, του οποίου τα στοιχεία είναι επίσης πίνακες μίας διάστασης. Γενικεύοντας μπορούμε να ορίζουμε πίνακες με περισσότερες διαστάσεις.

Η γενική μορφή της εντολής δήλωσης ενός πολυδιάστατου πίνακα είναι:

τύπος αναγνωριστικό`[`*μέγεθος*$_1$`][`*μέγεθος*$_2$`]...[`*μέγεθος*$_n$`];`

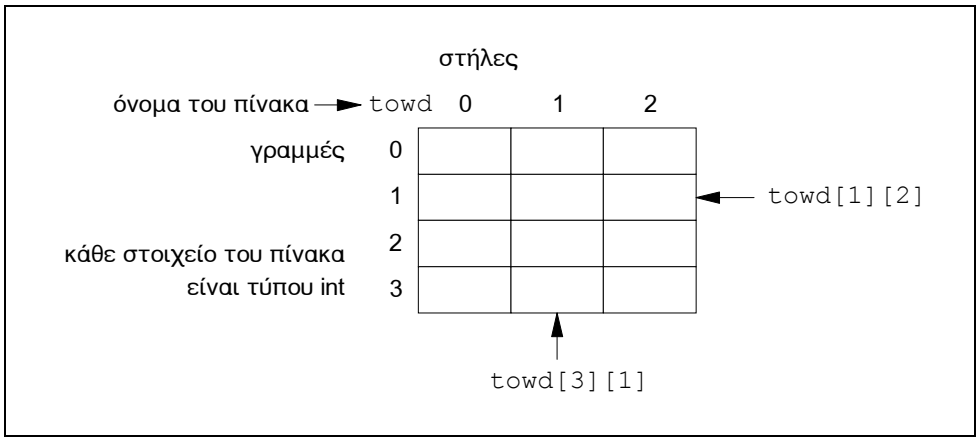
Έτσι, για παράδειγμα, η παρακάτω εντολή δημιουργεί ένα διδιάστατο πίνακα ακεραίων 4x3 (δηλαδή με 4 γραμμές και 3 στήλες):

```
int towd[4][3];
```

Η αναφορά στα στοιχεία ενός πίνακα με περισσότερες της μίας διαστάσεις γίνεται με τη χρήση περισσότερων από ένα δεικτών-πίνακα. Π.χ. για να αναφερθούμε στο στοιχείο της 2ης γραμμής και της 3ης στήλης του πίνακα `towd` γράφουμε:

```
towd[1][2]
```

Στο Σχήμα 8.2 δίνεται μία σχηματική αναπαράσταση του πίνακα `towd`.



Σχήμα 8.2: *Σχηματική αναπαράσταση διδιάστατου πίνακα*

Το πρόγραμμα που ακολουθεί χρησιμοποιεί ένα διδιάστατο πίνακα ακεραίων 4x5, τα στοιχεία του οποίου δίνονται απ'το χρήστη, μόλις ολοκληρωθεί η εισαγωγή ο πίνακας εμφανίζεται στην οθόνη. Στη συνέχεια ο χρήστης καλείται να δώσει έναν ακέραιο, τον οποίον το πρόγραμμα αναζητά μέσα στον πίνακα. Αν τον εντοπίσει τυπώνει τη θέση την οποία βρίσκεται (γραμμή, στήλη):

```
/* found.c */
/* Αναζήτηση στοιχείου σε πίνακα δύο διαστάσεων */
#define LINES   4
#define COLUMNS 5
main()
{
  int matrix[LINES][COLUMNS],i,j,num,found=0;

  for(i=0;i<LINES;i++)            /* εισαγωγή στοιχείων */
    for(j=0;j<COLUMNS;j++)        /* στον πίνακα        */
    {
      printf("Δώσε το στοιχείο %d,%d του πίνακα: ",i,j);
      scanf("%d",&matrix[i][j]);
    }
  for(i=0;i<LINES;i++)            /* εκτύπωση πίνακα */
  {
    for(j=0;j<COLUMNS;j++)
      printf("%5d",matrix[i][j]);
    printf("\n");
  }
```

```
  printf("Δώσε τον ακέραιο που θέλεις να ψάξω: ");
  scanf("%d",&num);
  i=0;
  while(!found && i<LINES)       /* έρευνα του πίνακα */
  {
    j=0;
    while(!found && j<COLUMNS)
      if(matrix[i][j]==num) found=1; else j++;
    if(!found) i++;
  }
  if(found)
    printf("Ο αριθμός %d υπάρχει στη θέση %d,%d",num,i,j);
  else
     printf("Ο αριθμός %d δεν υπάρχει στον πίνακα",num);
}
```

8.1.4 ΑΠΟΔΟΣΗ ΑΡΧΙΚΩΝ ΤΙΜΩΝ ΣΕ ΠΙΝΑΚΕΣ

Για να θέσουμε αρχικές τιμές σ'έναν πίνακα, θα πρέπει αυτός να έχει οριστεί *εξωτερικά* ή να έχει καθοριστεί ως `static` (βλέπε § 7.5). Η γενική μορφή της εντολής απόδοσης αρχικών τιμών σε έναν πίνακα, μοιάζει με των άλλων μεταβλητών:

static *τύπος αναγνωρ.**[μέγ.*$_1$*]...[μέγ.*$_n$*]={**λίστα_τιμών**};*

όπου *λίστα_τιμών* είναι μία λίστα σταθερών που χωρίζονται μεταξύ τους με κόμμα και ο τύπος τους είναι συμβατός με το βασικό τύπο του πίνακα. Αυτή η εντολή τοποθετεί την πρώτη σταθερά στην πρώτη θέση του πίνακα, τη δεύτερη σταθερά στη δεύτερη θέση κ.ο.κ.

Η παρακάτω εντολή αποδίδει αρχικές τιμές σ'ένα μονοδιάστατο πίνακα:

```
static float array[5]={18.3,7.98,6.8,3.1,7.21};
```

Όταν αποδίδουμε αρχικές τιμές σ'ένα μονοδιάστατο πίνακα, μπορούμε αν θέλουμε να παραλείψουμε το *μέγεθος* του πίνακα. Στην περίπτωση αυτή ο μεταγλωττιστής θα μας κάνει τη χάρη να μετρήσει το πλήθος των στοιχείων στη λίστα των αρχικών τιμών και να θέσει αυτό σαν μέγεθος του πίνακα. Έτσι η παραπάνω εντολή θα μπορούσε να ήταν:

```
static float array[]={18.3,7.98,6.8,3.1,7.21};
```

Ως παράδειγμα ας δούμε το παρακάτω πρόγραμμα το οποίο ζητά από το χρήστη να δώσει κάποιο ποσό σε δραχμές και κατόπιν αυτό το ποσό μετατρέπεται στον ελάχιστο αριθμό κερμάτων και χαρτονομισμάτων που χρειάζονται για να συμπληρωθεί αυτό:

```
/* change.c */
/* Μετατροπή ποσού στον ελάχιστο αριθμό νομισμάτων */
# define SIZE 11
main()
{
  static int table[]={10000,5000,1000,500,100,50,20,10,5,2,1};
  /* αρχικές τιμές στον πίνακα table */
  long int amount;
  int i,quantity;
```

```
  printf("Δώσε ποσό σε δραχμές: ");
  scanf("%D", &amount);
  for(i=0;i<SIZE;i++)
  {
    quantity=amount/table[i];
    printf("Νόμισμα %4d δρχ.  τεμάχια=%d\n", table[i],quantity);
    amount %=table[i];
  }
}
```

Οι πολυδιάστατοι πίνακες παίρνουν αρχικές τιμές κατά τον ίδιο τρόπο που παίρνουν και οι μονοδιάστατοι. Για παράδειγμα η παρακάτω εντολή δίνει στον πίνακα `sqrs` ως αρχικές τιμές τους αριθμούς από 1 έως 4 μαζί με τα τετράγωνά τους:

```
static int sqrs[4][2]={{1,1},
                       {2,4},
                       {3,9},
                       {4,16}};
```

ή ισοδύναμα:

```
static int sqrs[4][2]={1,1,
                       2,4,
                       3,9,
                       4,16};
```

Φυσικά οι αρχικές τιμές θα μπορούσαν να γραφούν και σε μία γραμμή, αν και αυτό δε μας βοηθά στη σχηματική αναπαράσταση του πίνακα. Κατά την απόδοση αρχικών τιμών σε πολυδιάστατους πίνακες, μπορούμε αν θέλουμε, να παραλείψουμε την αριστερότερη διάσταση του πίνακα. Αυτό δεν εμποδίζει το μεταγλωττιστή να υπολογίσει αυτή τη διάσταση, αφού όλα τα στοιχεία του πίνακα είναι με τη σειρά αποθηκευμένα κατά γραμμές, από το πρώτο ως το τελευταίο, σε διαδοχικές θέσεις μνήμης. Για παράδειγμα η απόδοση αρχικών τιμών στον πίνακα `sqrs` θα μπορούσε να ήταν:

```
static int sqrs[][2]{1,1,
                     2,4,
                     3,9,
                     4,16};
```

απ'όπου είναι προφανές ότι εφόσον ο πίνακας έχει δύο στήλες και συνολικά οκτώ στοιχεία, το μέγεθος της διάστασης που δεν καθορίζεται είναι 4 (γραμμές x στήλες = 8 ⇔ γραμμές = 8/στήλες).

8.1.5 ΠΙΝΑΚΕΣ ΩΣ ΠΑΡΑΜΕΤΡΟΙ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ

Όπως ξέρουμε τα στοιχεία ενός πίνακα καταλαμβάνουν διαδοχικές θέσεις στη μνήμη του Η/Υ. Κάθε θέση μνήμης έχει χωρητικότητα 1 byte (=8 bits) και της αντιστοιχεί ένας αριθμός που καλείται *διεύθυνση*. Οι διευθύνσεις αρχίζουν από το μηδέν και διαδοχικές θέσεις μνήμης έχουν διευθύνσεις που διαφέρουν κατά ένα.

Ως διεύθυνση του πίνακα εννοείται η διεύθυνση μνήμης όπου βρίσκεται αποθηκευμένο το πρώτο στοιχείο του. Φυσικά, αφού ξέρουμε τον τύπο των στοιχείων του πίνακα (και κατά συνέπεια τον αριθμό των bytes που καταλαμβάνει καθένα απ'αυτά) και τη διεύθυνση του πρώτου στοιχείου του, μπορούμε να βρούμε τη διεύθυνση οποιουδήποτε στοιχείου του.

Το ερώτημα είναι: πώς αναφερόμαστε στη διεύθυνση του πίνακα; Και η απάντηση: χρησιμοποιώντας το *αναγνωριστικό* του πίνακα, χωρίς τις αγκύλες. Από την άλλη, η διεύθυνση του πρώτου στοιχείου του πίνακα (χρησιμοποιώντας τον *τελεστή διεύθυνσης* `&`, που έχουμε δει στην § 5.3.2) είναι:

```
&αναγνωριστικό[0]
```

Έτσι, με βάση τα παραπάνω προκύπτει ότι:

```
διεύθυνση_πίνακα==διεύθυνση_πρώτου_στοιχείου_του
```

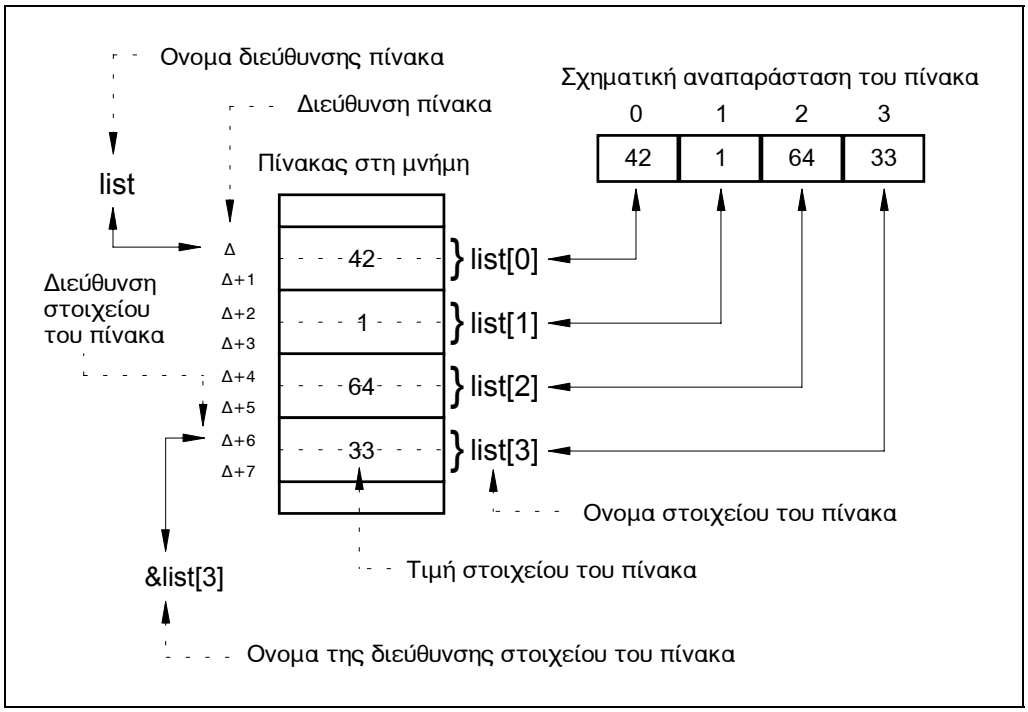
και άρα

```
αναγνωριστικό==&αναγνωριστικό[0]
```

Για να γίνουν κατανοητά τα παραπάνω ας θεωρήσουμε τον ορισμό:

```
static int list[4]={42,1,64,33};
```

Το Σχήμα 8.3 δείχνει τη σχέση μεταξύ της σχηματικής αναπαράστασης του πίνακα και της πραγματικής του υπόστασης στη μνήμη. Επίσης δείχνει τη διαφορά μεταξύ της *διεύθυνσης ενός στοιχείου* του πίνακα και της *τιμής* αυτού.



Σχήμα 8.3: *Η διεύθυνση και τα στοιχεία ενός πίνακα*

Τώρα που κατανοήσαμε τα περί διευθύνσεων των πινάκων, ας δούμε πώς χρησιμοποιούνται οι πίνακες ως παράμετροι σε συναρτήσεις. Μέχρι τώρα έχουμε δει παραδείγματα που χρησιμοποιούσαν μεταβλητές βασικών τύπων δεδομένων ως πραγματικές παραμέτρους σε μία συνάρτηση. Στην περίπτωση αυτή

μεταβιβάζονται στη συνάρτηση οι τιμές των μεταβλητών. Με άλλα λόγια, δημιουργούνται αντίγραφα των μεταβλητών και μόνο σ'αυτά τα αντίγραφα μπορεί να έχει πρόσβαση η συνάρτηση (κλήση κατ'αξία).

Όταν χρησιμοποιούμε έναν πίνακα ως παράμετρο σε μία συνάρτηση, δε μεταβιβάζονται οι τιμές του πίνακα στη συνάρτηση, δηλαδή δε δημιουργείται ένα αντίγραφο του πίνακα, πάνω στο οποίο θα έχει πρόσβαση η συνάρτηση. Αυτό που μεταβιβάζεται είναι η *διεύθυνση* του πίνακα. Έτσι οποιαδήποτε αλλαγή γίνει από τη συνάρτηση στα στοιχεία του πίνακα, είναι πραγματική.

Ως εφαρμογή των παραπάνω ας δούμε ένα πρόγραμμα που δέχεται από το πληκτρολόγιο ακέραιες τιμές για τα στοιχεία ενός διδιάστατου πίνακα και στη συνέχεια βρίσκει και τυπώνει τη μεγαλύτερη απ'αυτές. Η εύρεση της μεγαλύτερης τιμής γίνεται από τη συνάρτηση `max()` η οποία δέχεται ως παράμετρο έναν πίνακα.

Ας παρατηρήσουμε ότι στη δήλωση του πίνακα ως τυπικής παραμέτρου στη συνάρτηση, δε χρειάζεται να πούμε στη συνάρτηση πόσες γραμμές έχει ο πίνακας. Αυτό διότι αυτή η δήλωση δεν κρατάει χώρο στη μνήμη για τον πίνακα. Το μόνο που χρειάζεται να ξέρει η συνάρτηση είναι το πόσες στήλες έχει ο πίνακας, αυτό της επιτρέπει να αναφέρεται με ακρίβεια σε οποιοδήποτε στοιχείο του πίνακα, καθόσον τα στοιχεία του διδιάστατου πίνακα αποθηκεύονται κατά γραμμές σε διαδοχικές θέσεις στη μνήμη. Για παράδειγμα για να βρει τη θέση στη μνήμη (ως προς το πρώτο στοιχείο του πίνακα) που είναι αποθηκευμένο το στοιχείο `matrix[3][1]`, πολλαπλασιάζει τον αριθμό της γραμμής (3) με τον αριθμό των στοιχείων ανά γραμμή (`COLUMNS`, που είναι 5) και μετά προσθέτει τον αριθμό της στήλης (που είναι 1). Το αποτέλεσμα είναι `3*5+1=16`.

```
/* maxnum.c */
/* Εύρεση μεγαλύτερου αριθμού σε πίνακα δύο διαστάσεων */
#define LINES   4
#define COLUMNS 5
main()
{
  int matrix[LINES][COLUMNS],i,j,num;

  for(i=0;i<LINES;i++)
    for(j=0;j<COLUMNS;j++)
    {
      printf("Δώσε το στοιχείο %d,%d του πίνακα: ",i,j);
      scanf("%d",&matrix[i][j]);
    }
  num=max(matrix);
  printf("Το μέγιστο στοιχείο του πίνακα είναι το %d\n",num);
}

/* max() */
/* Επιστρέφει το μεγαλύτερο στοιχείο του πίνακα */
max(int matr[][COLUMNS])
{
  int i,j,m;

  m=matr[0][0];           /* έστω ότι μεγαλύτερο στοιχείο */
```

```
  for(i=0;i<LINES;i++)     /* είναι το 0,0, έπειτα ελέγ-   */
    for(j=0;j<COLUMNS;j++)/* χουμε τα υπόλοιπα στοιχεία,  */
      if(m<matr[i][j])     /* αν κάποιο είναι μεγαλύτερο   */
        m=matr[i][j];      /* θέτουμε αυτό στη μεταβλητή m */
  return(m);
}
```

8.1.6 ΤΑΞΙΝΟΜΗΣΗ ΠΙΝΑΚΑ

Πριν προχωρήσουμε στα αλφαριθμητικά, θα δούμε μία συνάρτηση που ταξινομεί τις τιμές ενός πίνακα. Η ταξινόμηση είναι σημαντική λειτουργία, ιδιαίτερα σε προγράμματα που χειρίζονται βάσεις δεδομένων (databases), στα οποία ο χρήστης θέλει να αναδιατάξει μία αριθμητική ή αλφαβητική λίστα στοιχείων, σε αύξουσα ή φθίνουσα σειρά.

Υπάρχουν αρκετές μέθοδοι ταξινόμησης, μία από τις απλούστερες είναι η μέθοδος της *επιλογής* (sorting by selection). Η μέθοδος αυτή είναι η εξής: Βρίσκουμε το μικρότερο στοιχείο του πίνακα και το ανταλλάσσουμε με το πρώτο στοιχείο αυτού. Έπειτα κάνουμε το ίδιο στον πίνακα που μένει αν αγνοήσουμε το πρώτο στοιχείο του αρχικού πίνακα, κάνουμε το ίδιο στον πίνακα που μένει αν αγνοήσουμε τα δύο πρώτα στοιχεία του αρχικού πίνακα κ.ο.κ. Η μέθοδος σταματάει αν φτάσουμε σε πίνακα με ένα μόνο στοιχείο.

```
/* sortsel.c */
/* Ταξινόμηση με τη μέθοδο της επιλογής */
#define SIZE 5
main()
{
  int list[SIZE],i;

  for(i=0;i<SIZE;i++)
  {
    printf("Δώσε το %do στοιχείο του πίνακα: ",i);
    scanf("%d",&list[i]);
  }
  sort(list);
  printf("Ο πίνακας ταξινομημένος είναι:\n");
  for(i=0;i<SIZE;i++) printf("%d ",list[i]);
}


/* sort() */
/* Ταξινομεί τον πίνακα με τη μέθοδο της επιλογής */
sort(int list[])
{
  int i,j,k,min,temp;

  for(i=0;i<SIZE-1;i++)
  {                              /* υποθέτουμε ότι το ελάχιστο */
    min=i;                       /* στοιχείο βρίσκεται στην 1η */
    for(j=i+1;j<SIZE;j++)        /* θέση του πίνακα, σαρώνουμε */
     if(list[j]<list[min]) min=j;/* τον πίνακα και βρίσκουμε   */
    if(min!=i)                   /* τη θέση του πραγματικά     */
```

```
    {                              /* ελάχιστου στοιχείου,τέλος  */
      temp=list[i];                /* εναλλάσσουμε την τιμή της  */
      list[i]=list[min];           /* πρώτης θέσης με αυτή της   */
      list[min]=temp;              /* θέσης που βρήκαμε το       */
    }                              /* πραγματικά ελάχιστο        */
  }
}
```

8.2 ΑΛΦΑΡΙΘΜΗΤΙΚΑ

Η συνηθέστερη χρήση των μονοδιάστατων πινάκων είναι στη δημιουργία αλφαριθμητικών (strings). Στη C ένα αλφαριθμητικό, δεν είναι τίποτα άλλο από έναν *πίνακα χαρακτήρων*, ο οποίος τερματίζει με το χαρακτήρα '\0' ο οποίος λέγεται *ανύπαρκτος χαρακτήρας* (null character) και έχει αριθμητική τιμή μηδέν. Για το λόγο αυτό θα πρέπει να δηλώνουμε τους πίνακες χαρακτήρων κατά ένα χαρακτήρα μεγαλύτερους από το μεγαλύτερο αλφαριθμητικό που επιθυμούμε να χωράνε.

Όπως μπορούμε να θέσουμε αρχικές τιμές στους πίνακες, έτσι μπορούμε και στα αλφαριθμητικά, μιας κι αυτά είναι πίνακες χαρακτήρων. Έτσι η παρακάτω εντολή:

```
static char str[]={'Κ','α','λ','η','μ','έ','ρ','α','\0'};
```

θέτει την αρχική τιμή "Καλημέρα" στο αλφαριθμητικό `str`. Όμως η C μας παρέχει έναν πιο σύντομο τρόπο για να κάνουμε το ίδιο πράγμα:

```
static char str[]="Καλημέρα";
```

ας προσέξουμε ότι στο δεύτερο τρόπο δε χρειάζεται να κάνουμε χρήση του '\0'.

Το παρακάτω πρόγραμμα εξετάζει πώς είναι αποθηκευμένο ένα αλφαριθμητικό στη μνήμη του Η/Υ. Στο πρόγραμμα αυτό φαίνεται και μία νέα συνάρτηση βιβλιοθήκης, η `strlen()`, την οποία θα δούμε στην επόμενη παράγραφο.

```
/* strexam.c */
main()
{
  char name[81];
  int i;

  puts("Δώσε το όνομά σου: "); gets(name);
  for(i=0;i<strlen(name)+4;i++)
   printf("Δ/νση=%5u  Χαρακτ.='%c'=%3d\n",&name[i],name[i],name[i]);
}
```

Για να δείξουμε τι συμβαίνει μετά το τέλος του αλφαριθμητικού, στο προηγούμενο πρόγραμμα, τυπώνουμε ακόμα τέσσερις χαρακτήρες μετά το τέλος του.

8.2.1 ΣΥΝΑΡΤΗΣΕΙΣ ΒΙΒΛΙΟΘΗΚΗΣ ΓΙΑ ΑΛΦΑΡΙΘΜΗΤΙΚΑ

Έχουμε δει στην § 5.2 τις συναρτήσεις `gets()` και `puts()` για το διάβασμα και την εκτύπωση αλφαριθμητικών. Όταν διαβάζουμε με την `gets()` ένα αλφαριθμητικό από το πληκτρολόγιο, ο χαρακτήρας τερματισμού '\0' προστίθεται αυτόματα στο τέλος του. Εκτός απ'αυτές, η C παρέχει ένα σύνολο από συναρτήσεις χειρισμού αλφαριθμητικών, τις κυριότερες απ'τις οποίες θα δούμε παρακάτω:

strlen()

Η κλήση της `strlen()` έχει την παρακάτω μορφή:

```
strlen(s);
```

όπου `s` είναι ένα αλφαριθμητικό. Η συνάρτηση επιστρέφει το πλήθος των χαρακτήρων (μήκος) του αλφαριθμητικού. Δε μετρά το χαρακτήρα τερματισμού '`\0`'.

strcpy()

Η κλήση της `strcpy()` έχει την παρακάτω μορφή:

```
strcpy(s1,s2);
```

όπου `s1`, `s2` αλφαριθμητικά. Η συνάρτηση αντιγράφει το αλφαριθμητικό `s2` στο `s1`. Το `s1` πρέπει να είναι αρκετά μεγάλο για να χωρέσει το `s2`.

strcat()

Η κλήση της `strcat()` έχει την παρακάτω μορφή:

```
strcat(s1,s2);
```

όπου `s1`, `s2` αλφαριθμητικά. Η συνάρτηση προσαρτά το `s2` στο τέλος του `s1`. Το `s1` πρέπει να είναι αρκετά μεγάλο για να χωρέσει και το `s2`.

strcmp()

Η κλήση της `strcmp()` έχει την παρακάτω μορφή:

```
strcmp(s1,s2);
```

όπου `s1`, `s2` αλφαριθμητικά. Η συνάρτηση συγκρίνει τα `s1`, `s2` και επιστρέφει τιμή αρνητική, μηδενική, ή θετική, αν το `s1` είναι, λεξικογραφικά, μικρότερο από, ίσο με, ή μεγαλύτερο από το `s2` αντίστοιχα.

Παρακάτω δίνονται μερικά προγράμματα που χειρίζονται αλφαριθμητικά, κάνοντας χρήση των συναρτήσεων που προαναφέραμε. Το πρώτο πρόγραμμα χρησιμοποιεί τη συνάρτηση `strcmp()` για να συγκρίνει δύο αλφαριθμητικά:

```
/* compare.c */
/* Σύγκριση δύο αλφαριθμητικών με τη συνάρτηση strcmp() */
main()
{
  char string1[81], string2[81];
  int i;

  printf("\nΔώσε το πρώτο αλφαριθμητικό: "); gets(string1);
  printf("Δώσε το δεύτερο αλφαριθμητικό: "); gets(string2);
  i=strcmp(string1,string2);
  if(i<0) printf("%s < %s",string1,string2);
  else if(i>0) printf("%s > %s",string1,string2);
       else printf("%s == %s",string1,string2);
}
```

Το επόμενο πρόγραμμα ζητά από το χρήστη να πληκτρολογήσει ένα αλφαριθμητικό και έναν ακέραιο αριθμό. Στη συνέχεια *διαγράφει* από το αλφαριθμητικό το χαρακτήρα που βρίσκεται στη θέση που αντιστοιχεί στον αριθμό που έδωσε ο χρήστης:

```
/* delete.c */
/* Διαγραφή χαρακτήρα από ένα αλφαριθμητικό, */
/* με χρήση της συνάρτησης strcpy()          */
main()
{
  char string[81];
  int position;

  printf("\nΔώσε αλφαριθμητικό [ENTER], θέση\n");
  gets(string);
  scanf("%d",&position);
  strdel(string,position);
  puts(string);
}

/* strdel */
/* Διαγράφει χαρακτήρα από αλφαριθμητικό */
strdel(char str[],int n)
{                              /* μετακινούμε το δεύτερο τμήμα του  */
  strcpy(&str[n],&str[n+1]);/* αλφαριθμητικού, μία θέση αριστερά */
}
```

Τέλος το επόμενο πρόγραμμα κάνει το αντίθετο. Διαβάζει ένα αλφαριθμητικό, ένα χαρακτήρα και έναν αριθμό και *παρεμβάλλει* στη θέση του αλφαριθμητικού στην οποία αντιστοιχεί ο αριθμός, το χαρακτήρα:

```
/* insert.c */
/* Εισαγωγή χαρακτήρα σε ένα αλφαριθμητικό, */
/* με χρήση της συνάρτησης strcpy()     */
main()
{
  char character, string[81];
  int position;
  printf("\nΔώσε αλφαριθμητικό [ENTER],χαρακτήρα, θέση\n");
  gets(string);
  scanf("%c %d",&character,&position);
  strins(string,character,position);
  puts(string);
}

/* strins */
/* Παρεμβάλλει χαρακτήρα σε αλφαριθμητικό */
strins(char str[],char ch,int n)
{
  char temp[81];         /* προσωρινή μεταβλητή */
  strcpy(temp,&str[n]);  /* αποθηκεύουμε το δεύτερο μισό στην temp */
  str[n]=ch;             /* εισάγουμε το χαρακτήρα στη θέση του    */
  strcpy(&str[n+1],temp);/* ολισθαίνουμε το δεύτερο μισό δεξιά     */
```

```
}
```

8.3 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο στοιχεία για ένα μονοδιάστατο πίνακα ακεραίων 10 θέσεων. Στη συνέχεια να βρίσκει το μέγιστο κι ελάχιστο στοιχείο αυτού, καθώς και τις θέσεις που βρίσκονται (γραμμή, στήλη).

```
/* minmax1.c */
/* Εύρεση του μικρότερου και μεγαλύτερου στοιχείου    */
/* μονοδιάστατου  πίνακα καθώς και των θέσεών τους   */
#define SIZE 10
main()
{
  int i,max,min,imax,imin,list[SIZE];

  for(i=0;i<SIZE;i++)
  {
    printf("Δώσε ακέραιο: "); /* εισαγωγή στοιχείων */
    scanf("%d",&list[i]);     /* στον πίνακα        */
  }
  for(i=0;i<SIZE;i++)         /* εκτύπωση πίνακα    */
    printf("%d ",list[i]);
  max=min=list[0];            /* υποθέτουμε ότι το μέγιστο   */
  imax=imin=0;                /* και το ελάχιστο στοιχείο    */
  for(i=1;i<SIZE;i++)         /* βρίσκονται στην πρώτη θέση, */
  {                           /* στη συνέχεια σαρώνουμε όλα  */
    if(list[i]>max)           /* τα στοιχεία και ελέγχουμε   */
    {                         /* καθένα απ'αυτά είναι μεγα-  */
      max=list[i];            /* λύτερο από το μέχρι στιγμής */
      imax=i;                 /* μέγιστο ή μικρότερο από το  */
    }                         /* μέχρι στιγμής ελάχιστο, αν  */
    else if(list[i]<min)      /* είναι τότε αλλάζουμε τις    */
    {                         /* μέχρι στιγμής υποθέσεις μας */
      min=list[i];            /* για το μέγιστο και (ή) ελά- */
      imin=i;                 /* χιστο και συνεχίζουμε μέχρι */
    }                         /* το τέλος του πίνακα         */
  }
  printf("\nΜεγαλύτερο στοιχείο %d, θέση %d\n",max,imax);
  printf("\nΜικρότερο  στοιχείο %d, θέση %d\n",min,imin);
}
```

2. Στην § 8.1.6 είδαμε μία μέθοδο ταξινόμησης, αυτή της επιλογής. Μία σχετικά απλή και πιο γρήγορη μέθοδος ταξινόμησης είναι η μέθοδος της φυσαλίδας (bubble sort) κατά την οποία συγκρίνονται τα στοιχεία ανά δύο μεταξύ τους. Συγκεκριμένα, αν υποθέσουμε ότι έχουμε έναν πίνακα με `N` στοιχεία, αρχικά συγκρίνονται το πρώτο με το δεύτερο και αν το πρώτο είναι μεγαλύτερο από το δεύτερο, τότε ανταλλάσσονται. Στη συνέχεια συγκρίνονται το δεύτερο με το τρίτο και αν το δεύτερο είναι μεγαλύτερο από το τρίτο ανταλλάσσονται, κ.ο.κ. Γενικά για `N` στοιχεία γίνονται `N-1` συγκρίσεις. Στο τέλος των συγκρίσεων παρατηρούμε ότι το μεγαλύτερο στοιχείο βρίσκεται στο τέλος του πίνακα και τα μικρότερα στοιχεία

έχουν "ανέβει" προς την αρχή του (σαν φυσαλίδες). Στη συνέχεια επαναλαμβάνεται η διαδικασία συγκρίνοντας από την αρχή τα στοιχεία μεταξύ τους, εκτός από το τελευταίο. Δηλαδή τώρα γίνονται `N-2` συγκρίσεις. Συνολικά η διαδικασία επαναλαμβάνεται `N-1` φορές και κάθε φορά γίνεται μία σύγκριση λιγότερη.
Να γραφεί πρόγραμμα που να ταξινομεί έναν πίνακα με τη μέθοδο της φυσαλίδας.

Στην ουσία το πρόγραμμα είναι ίδιο με το `sortsel.c` της § 8.1.6, μόνο που η συνάρτηση `sort()` θα αλλάξει και θα γίνει όπως παρακάτω:

```
/* sort() */
/* Ταξινομεί τον πίνακα με τη μέθοδο της φυσαλίδας */
sort(int list[])
{
  int i,j,temp;

  for(i=0;i<SIZE-1;i++)
    for(j=0;j<SIZE-1-i;j++)
      if(list[j]>list[j+1])
      {
        temp=list[j];
        list[j]=list[j+1];
        list[j+1]=temp;
      }
}
```

3. Να γραφεί πρόγραμμα για τον προσδιορισμό της θέσης ενός στοιχείου μέσα σε ένα μονοδιάστατο πίνακα (έστω ακεραίων).

Αν ο πίνακας δεν είναι ταξινομημένος, τότε ο μόνος τρόπος αναζήτησης είναι η *σειριακή* (serial) αναζήτηση. Κατά τη σειριακή αναζήτηση συγκρίνουμε το στοιχείο που ζητάμε με το πρώτο στοιχείο του πίνακα. Αν δεν είναι ίδια συγκρίνουμε το ζητούμενο στοιχείο με το δεύτερο στοιχείο του πίνακα κ.ο.κ. Με τη μέθοδο αυτή, για πίνακα `N` στοιχείων, απαιτούνται κατά μέσο όρο `N/2` συγκρίσεις. Το παρακάτω πρόγραμμα πραγματοποιεί τη σειριακή αναζήτηση:

```
/* serfound.c */
/* Σειριακή αναζήτηση σε πίνακα */
#define SIZE 10
main()
{
  int list[SIZE],pos=-1,i,num;

  for(i=0;i<SIZE;i++)

  {
    printf("Δώσε το %do στοιχείο του πίνακα: ",i);
    scanf("%d",&list[i]);
  }
  printf("\nΔώσε τον ακέραιο που θέλεις να ψάξω: ");
  scanf("%d",&num);
  pos=search(list,num);
  if(pos<0) puts("Ο αριθμός δεν υπάρχει στον πίνακα");
  else printf("Υπάρχει στην %d θέση",pos);
```

```
}

/* search() */
/* Σειριακή αναζήτηση */
search(int list[],int n)
{
  int i=0,found=0,pos=-1;

  while(i<SIZE && !found)
  {
    if(list[i]==n)
    {
      found=1;
      pos=i;
    }
    else i++;
  }
  return(pos);
}
```

Στην περίπτωση που τα στοιχεία του πίνακα είναι ταξινομημένα, τότε η διαδικασία αναζήτησης μπορεί να επιταχυνθεί εφαρμόζοντας τη μέθοδο της *δυαδικής* (binary) αναζήτησης. Κατά τη μέθοδο αυτή συγκρίνουμε το ζητούμενο στοιχείο με το μεσαίο στοιχείο του πίνακα. Αν δεν είναι ίσα, τότε εξετάζουμε αν το ζητούμενο στοιχείο βρίσκεται στον αριστερό υποπίνακα (δηλαδή αν ζητούμενο<μεσαίο) ή στον δεξιό υποπίνακα (δηλαδή αν ζητούμενο>μεσαίο). Επαναλαμβάνουμε τη διαδικασία στον νέο υποπίνακα βρίσκοντας πάλι το μεσαίο στοιχείο, κ.ο.κ. Ο μέγιστος αριθμός συγκρίσεων για πίνακα N στοιχείων είναι Log_2N. Η παρακάτω συνάρτηση κάνει δυαδική αναζήτηση στον πίνακα του προηγουμένου προγράμματος, με την προϋπόθεση βέβαια, ότι αυτός είναι ταξινομημένος:

```
/* search() */
/* Δυαδική αναζήτηση σε ταξινομημένο πίνακα */
search(int list[],int n)
{
  int left=0,right=SIZE-1,middle,found=0,i=-1;

  while(left<=right && !found)
  {
    middle=(left+right)/2;
    if(n==list[middle])
    {
      i=middle;
      found=1;
    }
    else
    {
      if(n>list[middle]) left=middle+1;
      else right=middle-1;
    }
  }
  return(i);
}
```

4. Ας υποθέσουμε ότι έχουμε δύο ταξινομημένους μονοδιάστατους πίνακες. Τότε η διαδικασία της *συνταξινόμησης* (merge) είναι να δημιουργήσουμε έναν τρίτο επίσης ταξινομημένο πίνακα που θα περιέχει όλα τα στοιχεία των δύο προηγουμένων πινάκων. Το παρακάτω πρόγραμμα δέχεται από το πληκτρολόγιο τα στοιχεία για δύο μονοδιάστατους πίνακες ακεραίων (τους `list1[]` και `list2[]`), στη συνέχεια ταξινομεί αυτούς με το μέθοδο της φυσαλίδας και κατόπιν τους συνταξινομεί τοποθετώντας τα στοιχεία τους σε έναν τρίτο πίνακα (τον `list[]`):

```
/* merge.c */
/* Συνταξινόμηση */
#define SIZE1 5
#define SIZE2 10
#define SIZE  SIZE1+SIZE2
main()
{
  int list1[SIZE1],list2[SIZE2],list[SIZE],i;

  for(i=0;i<SIZE1;i++)
  {
    printf("Δώσε το %do στοιχείο του 1ου πίνακα: ",i);
    scanf("%d",&list1[i]);
  }
  for(i=0;i<SIZE2;i++)
  {
    printf("Δώσε το %do στοιχείο του 2ου πίνακα: ",i);
    scanf("%d",&list2[i]);
  }
  sort(list1,SIZE1);
  sort(list2,SIZE2);
  merge(list1,list2,list);
  printf("Οι πίνακες ταξινομημένοι είναι:\nlist1=");
  for(i=0;i<SIZE1;i++) printf("%d ",list1[i]);
  printf("\nlist2=");
  for(i=0;i<SIZE2;i++) printf("%d ",list2[i]);
  printf("\nlist =");
  for(i=0;i<SIZE;i++) printf("%d ",list[i]);
}

/* sort() */
/* Ταξινομεί τον πίνακα με τη μέθοδο της φυσαλίδας */
sort(int list[],int s)
{
  int i,j,temp;

  for(i=0;i<s-1;i++)
    for(j=0;j<s-1-i;j++)
      if(list[j]>list[j+1])
      {
        temp=list[j];
        list[j]=list[j+1];
        list[j+1]=temp;
      }
}
```

```
/* merge() */
/* Συνταξινομεί δύο πίνακες */
merge(int list1[],int list2[],int list[])
{
  int i=0,j=0,k=0;

  while(i<SIZE1 && j<SIZE2) /* όσο υπάρχουν στοιχεία     */
  {                         /* και στους δύο πίνακες     */
    if(list1[i]<list2[j])   /* τοποθετούνται στον πίνακα */
    {                       /* list[] με αύξουσα σειρά   */
      list[k]=list1[i];
      i++;
      k++;
    }
    else
    {
      list[k]=list2[j];
      j++;
      k++;
    }
  }
  while(i<SIZE1)        /* στην περίπτωση που τελειώσουν */
  {                     /* πρώτα τα στοιχεία του πίνακα  */
    list[k]=list1[i];   /* list2[], τοποθετούνται όλα τα */
    i++;                /* εναπομείναντα στοιχεία του    */
    k++;                /* list1[] στον list[]           */
  }
  while(j<SIZE2)        /* στην περίπτωση που τελειώσουν */
  {                     /* πρώτα τα στοιχεία του πίνακα  */
    list[k]=list2[j];   /* list1[], τοποθετούνται όλα τα */
    j++;                /* εναπομείναντα στοιχεία του    */
    k++;                /* list2[] στον list[]           */
  }
}
```

5. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο `n` (έστω `n=10`) ακέραιους αριθμούς και να υπολογίζει και τυπώνει τη μέση τιμή τους. Κατόπιν να τυπώνει έναν πίνακα που θα αποτελείται από δύο στήλες. Η πρώτη στήλη πρέπει να περιέχει τους δέκα αριθμούς που δώσαμε και η δεύτερη, για καθέναν από αυτούς, την απόκλισή του από τη μέση τιμή (απόκλιση = μέση τιμή - αριθμός).

```
/* statist1.c */
/* Υπολογισμός μέσης τιμής και αποκλίσεων N αριθμών */
#define N 10
main()
{
  int i;
  float list[N],sum=0,aver;

  for(i=0;i<N;i++)
  {
    printf("Δώσε αριθμό: ");
```

```
    scanf("%f",&list[i]);
    sum+=list[i];
  }
  aver=sum/N;
  printf("\nΜέση Τιμή=%9.5f\n",aver);
  puts("         list[i]   list[i]-Μέση Τιμή");
  for(i=0;i<N;i++)
    printf("%16.4f%16.5f\n",list[i],list[i]-aver);
}
```

6. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο ένα αλφαριθμητικό και ένα χαρακτήρα και να υπολογίζει το πλήθος των εμφανίσεων του χαρακτήρα μέσα στο αλφαριθμητικό.

```
/* ch_in_s.c */
/* Πλήθος εμφανίσεων χαρακτήρα σε αλφαριθμητικό */
main()
{
  char str[81],ch;
  int i=0,length;

  printf("\nΑλφαριθμητικό,Χαρακτήρας:");
  scanf("%s %c",str,&ch);
  length=strlen(str);
  for(i=0;i<length;i++)
    if(ch==str[i])
      i++;
  printf("\nO χαρακτήρας \'%c\' εμφανίζεται %d φορές",ch,i);
}
```

7. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο ένα αλφαριθμητικό και να το τυπώνει αντίστροφα (από το τέλος προς την αρχή του). Για παράδειγμα, το αλφαριθμητικό:

```
Αυτή είναι μία γραμμή
```

να το τυπώνει ως:

```
ήμμαργ αίμ ιανίε ήτυΑ
```

```
/* sreverse.c */
/* Εκτύπωση αλφαριθμητικού με αντίστροφη σειρά */
main()
{
  int length,i;

  puts("Αλφαριθμητικό:");
  gets(str);
  length=strlen(str);
  for(i=length-1;i>=0;i--)
    putch(str[i]);
}
```

8. Να γραφεί πρόγραμμα που να δέχεται από το πληκτρολόγιο ένα αλφαριθμητικό και να το τυπώνει με τις λέξεις να έχουν την ίδια σειρά, αλλά με τα γράμματά τους αντίστροφα. Για παράδειγμα, το αλφαριθμητικό:

```
Αυτή είναι μία γραμμή
```

να το τυπώνει ως:

```
ήτυΑ ιανίε αίμ ήμμαργ
```

```
/* srevers1.c */
/* Εκτύπωση αλφαριθμητικού με αντίστροφη σειρά στα */
/* γράμματα των λέξεων                              */
main()
{
  char str[81];
  int length,i,j,begin=0,end=0;

  puts("Αλφαριθμητικό:");
  gets(str);
  length=strlen(str);
  for(i=0;i<=length;i++)
  {
    if(str[i]!=' ' && str[i]!='\0') end++;
    if(str[i]==' ' || str[i]=='\0')
    {
      for(j=end-1;j>=begin;j--) putch(str[j]);
      putch(' ');
      begin=end+1;
      end++;
    }
  }
}
```

8.4 ΑΣΚΗΣΕΙΣ

1. Γράψτε πρόγραμμα που να δέχεται από το πληκτρολόγιο στοιχεία για ένα διδιάστατο πίνακα ακεραίων 4x4 και να βρίσκει και τυπώνει το μέγιστο κι ελάχιστο στοιχείο αυτού, καθώς και τις θέσεις που βρίσκονται (γραμμή, στήλη).
2. Να γραφεί πρόγραμμα που να δέχεται ακέραιες τιμές για τα στοιχεία ενός πίνακα 3x3 και να υπολογίζει και τυπώνει το άθροισμα των στοιχείων της κύριας και της δευτερεύουσας διαγωνίου του.
3. Γράψτε ένα πρόγραμμα που να δέχεται ως είσοδο τα στοιχεία ενός διδιάστατου πίνακα ακεραίων 4x3 και να εμφανίζει τον πίνακα στην οθόνη. Επίσης να μετράει και να εμφανίζει στην οθόνη το πλήθος των θετικών, αρνητικών και μηδενικών στοιχείων του πίνακα.

4. Γράψτε ένα πρόγραμμα που να δέχεται ως είσοδο τα στοιχεία ενός διδιάστατου πίνακα πραγματικών 4x3 και να αναστρέφει αυτόν. Ανάστροφος ενός πίνακα, είναι ένας άλλος πίνακας του οποίου οι γραμμές είναι οι στήλες και οι στήλες είναι οι γραμμές του πρώτου. Το πρόγραμμα να εμφανίζει στην οθόνη τόσο τον αρχικό πίνακα όσο και τον ανεστραμμένο.

5. Γράψτε ένα πρόγραμμα που να δέχεται από το πληκτρολόγιο τα στοιχεία για δύο μονοδιάστατους πίνακες ακεραίων 10 θέσεων ο καθένας και να υπολογίζει το άθροισμα αυτών των δύο πινάκων. Άθροισμα δύο μονοδιάστατων πινάκων είναι ένας τρίτος πίνακας, το κάθε στοιχείο του οποίου είναι το άθροισμα των αντίστοιχων στοιχείων των δύο πινάκων που προστίθενται.

6. Να δοθούν οι τιμές που θα πάρουν τα στοιχεία του πίνακα `p`, μετά την εκτέλεση του παρακάτω βρόχου:

```
main()
{
  int p[31],i=0;

  p[0]=0;
  while(i<15)
  {
    p[30-i]=p[0]-1;
    i++;
    p[i]=p[31-i]+1
  }
}
```

7. Γράψτε ένα πρόγραμμα που να καθορίζει την τιμή των στοιχείων του διδιάστατου πίνακα `a`, που δηλώνεται ως εξής:

   `int a[20][20];`

   σύμφωνα με τον παρακάτω κανόνα:

   `a[i][j]=1`, όταν το άθροισμα `i+j` είναι άρτιο και

   `a[i][j]=-1`, όταν το άθροισμα `i+j` είναι περιττό.

   Στο τέλος το πρόγραμμα πρέπει να τυπώνει τα στοιχεία του πίνακα `a`.

8. Έστω το παρακάτω πρόγραμμα. Δώστε τις τιμές που θα πάρουν τα στοιχεία του πίνακα `p`, μετά την εκτέλεσή του:

```
#define N 10
main();
{
  int p[N],x,i;

  puts("ΔΩΣΕ x= ");
  scanf("%d",&x);
  p[0]=x;
  for(i=1;i<N;i++) p[i]=p[i-1]*x;
}
```

9. Ο διδιάστατος πίνακας `a` δηλώνεται ως εξής:

   ```
   int a[20][20];
   ```

   Γράψτε ένα τμήμα προγράμματος που να μηδενίζει τα στοιχεία του πίνακα, εκτός από αυτά που βρίσκονται στη διαγώνιό του και στις "γειτονικές" διαγωνίους του.

10. Γράψτε ένα πρόγραμμα, που να δέχεται ως είσοδο `n` (`n=10`) ακέραιους αριθμούς και να υπολογίζει και τυπώνει τη μέση τιμή τους:

    $$\bar{x} = \frac{a_1 + a_2 + \ldots + a_n}{n}$$

    και την τυπική τους απόκλιση:

    $$s = \sqrt{\frac{(a_1 - \bar{x})^2 + (a_2 - \bar{x})^2 + \ldots + (a_n - \bar{x})^2}{n}}$$

11. Εξηγήστε τι κάνει το παρακάτω πρόγραμμα:

    ```
    #define SIZE 10
    main()
    {
      int i,s=0,a[SIZE];

      for(i=0;i<SIZE;i++) a[i]=i+1;
      for(i=0;i<SIZE;i++) s+=a[i]*a[i];
      printf("\n%d",s);
      getch();
    }
    ```

12. Γράψτε ένα πρόγραμμα που να δέχεται ένα αλφαριθμητικό με ελληνικούς χαρακτήρες και να το μετατρέπει με κεφαλαίους χαρακτήρες. Συμβουλευτείτε τον πίνακα ASCII.

13. Γράψτε ένα πρόγραμμα που να δέχεται ένα αλφαριθμητικό και να εξετάζει αν είναι παλινδρομικό (καρκινικό). Ένα αλφαριθμητικό είναι παλινδρομικό αν διαβάζεται το ίδιο και από τα δεξιά προς τα αριστερά, όπως π.χ. το "radar" και το "ΑΝΝΑ".

ΚΕΦΑΛΑΙΟ 9
ΑΠΑΡΙΘΜΗΣΕΙΣ, ΔΟΜΕΣ ΚΑΙ ΕΝΩΣΕΙΣ

Στο Κεφάλαιο αυτό θα δούμε πώς μπορεί ο προγραμματιστής της C να ορίζει τους δικούς του τύπους δεδομένων. Δηλαδή τύπους που θα έχουν ό,τι ονόματα επιθυμεί ο προγραμματιστής και το σύνολο των δυνατών τιμών τους θα καθορίζεται από τον ίδιο.

Επίσης θα δούμε δύο ακόμη δομημένους τύπους, εκτός από τον Πίνακα που εξετάσαμε στο Κεφάλαιο 8. Αυτοί είναι οι Δομές και οι Ενώσεις.

9.1 ΑΠΑΡΙΘΜΗΣΕΙΣ

Υπάρχουν πολλές περιπτώσεις που σε ένα πρόγραμμα, μία μη-αριθμητική μεταβλητή παίρνει τιμές από ένα μικρό σύνολο διακριτών τιμών. Η χρησιμοποίηση αριθμών για την αναπαράσταση μη-αριθμητικών δεδομένων, μπορεί σε ορισμένες περιπτώσεις να προκαλέσει σύγχυση.

Η C επιτρέπει στον προγραμματιστή να ορίζει τους δικούς του, νέους, τύπους δεδομένων, με αναγραφή των στοιχείων του συνόλου τιμών τους. Οι τύποι αυτοί ονομάζονται *απαριθμήσεις* (enumerations) ή απαριθμητοί τύποι. Η γενική μορφή της εντολής δήλωσης μίας απαρίθμησης, είναι:

enum *αναγνωριστικό_τύπου_απαρίθμησης* ***{****λίστα_τιμών****};***

όπου στη *λίστα_τιμών* αναγράφονται όλες οι δυνατές τιμές του τύπου με όνομα *αναγνωριστικό_τύπου_απαρίθμησης*.

Ένα παράδειγμα δήλωσης ενός τύπου απαρίθμησης, είναι:

```
enum colour{red, green, yellow};
```

Με την παραπάνω δήλωση, ορίσαμε ένα δικό μας τύπο με όνομα `colour` και σύνολο τιμών `{red,green,yellow}`. Στη συνέχεια μπορούμε να ορίσουμε μεταβλητές αυτού του τύπου, είτε αμέσως μετά το δεξιό άγκιστρο που τερματίζει τη λίστα των τιμών:

```
enum colour{red, green, yellow} a,b,c;
```

είτε μετά τη δήλωση της απαρίθμησης και χρησιμοποιώντας το όνομά της:

```
enum colour a,b,c;
```

Μία μεταβλητή απαρίθμησης μπορεί να πάρει οποιαδήποτε τιμή από τη *λίστα_τιμών*. Έτσι χρησιμοποιώντας τη μεταβλητή `c` που ορίστηκε παραπάνω, οι παρακάτω εντολές είναι σωστές:

```
c=green;
if(c==red) printf("Είναι κόκκινο\n");
```

Η παρακάτω εντολή, όμως:

```
c=blue;
```

δεν είναι σωστή, γιατί η τιμή `blue` δεν ανήκει στη λίστα των τιμών του τύπου `enum colour`.

Ο μεταγλωττιστής της C χειρίζεται τις τιμές της *λίστας_τιμών* ως ακέραιους αριθμούς. Κάθε τιμή της λίστας αντιστοιχεί σε έναν ακέραιο, αρχίζοντας από το μηδέν. Έτσι στο παραπάνω παράδειγμα, στην τιμή `red` αντιστοιχεί το `0`, στην `green` το `1` και στην `yellow` το `2`. Αυτός ο τρόπος ανάθεσης τιμών μπορεί να παρακαμφθεί από τον προγραμματιστή, θέτοντας στις τιμές της λίστας διαφορετικές ακέραιες τιμές:

```
enum colour{red=10, green=20, yellow=30};
```

Αν δε θέσουμε σε όλες τις τιμές της λίστας ακέραιες τιμές, τότε ο μεταγλωττιστής αποδίδει στις τιμές της λίστας που ακολουθούν την τελευταία τιμή στην οποία έχουμε θέσει ακέραια τιμή, ακέραιες τιμές μεγαλύτερες από αυτήν. Για παράδειγμα η παρακάτω εντολή θέτει την τιμή `50` στο `green`:

```
enum colour{red, green=50, yellow};
```

Μετά από αυτή την εντολή, στις τιμές της λίστας αντιστοιχούν οι εξής ακέραιοι: στο `red` το `1`, στο `green` το `50` και στο `yellow` το `51`.

Το ότι κάθε τιμή της *λίστας_τιμών* μίας απαρίθμησης χειρίζεται από το μεταγλωττιστή της C ως ακέραιος αριθμός, σημαίνει ότι μπορούμε να χρησιμοποιούμε τις τιμές της *λίστας_τιμών* σε πράξεις όπως θα χρησιμοποιούσαμε τους ακεραίους αριθμούς. Για παράδειγμα ας θεωρήσουμε το παρακάτω πρόγραμμα:

```
/* enums1.c */
/* Επίδειξη απαριθμήσεων */
main()
{
  enum colour {red,green,yellow}; /* ορισμός τύπου     */
  enum colour a,b,c;              /* δήλωση μεταβλητών */
  int i;

  a=red; b=green; c=yellow;
  i=a+b;
  printf("%d\n",i);
  if(a<c)
    printf("Η τιμή red είναι μικρότερη από την τιμή yellow\n");
}
```

Η έξοδος αυτού του προγράμματος, είναι η παρακάτω (γιατί;):

```
1
Η τιμή red είναι μικρότερη από την τιμή yellow
```

Το σημαντικότερο ίσως μειονέκτημα των απαριθμήσεων είναι ότι δε μπορεί κανείς να διαβάσει από το πληκτρολόγιο ή να εμφανίσει στην οθόνη απ'ευθείας τις τιμές της *λίστας_τιμών*. Για να γίνει κατανοητό αυτό, ας υποθέσουμε ότι έχουμε το παρακάτω τμήμα κώδικα:

```
enum day{kyr,dey,tri,tet,pem,par,sab};
enum day d;
d=kyr;
```

Τότε η παρακάτω εντολή είναι, προφανώς, λάθος:

```
printf("%s",d);
```

διότι η τιμή της μεταβλητής `d`, που είναι `kyr`, δεν είναι αλφαριθμητικό αλλά ένας ακέραιος (στη συγκεκριμένη περίπτωση το `0`). Έτσι η παραπάνω εντολή θα έπρεπε να γραφτεί:

```
printf("%d",d);
```

η οποία θα τύπωνε στην οθόνη τον αριθμό `0`.

Αν δε θέλουμε να τυπώνεται στην οθόνη ο ακέραιος που αντιστοιχεί στην τιμή μίας μεταβλητής απαρίθμησης, αλλά το αναγνωριστικό της τιμής αυτής, θα πρέπει να χρησιμοποιήσουμε κάποια από τις εντολές επιλογής `if` ή `switch`. Έτσι αν θέλουμε στο παραπάνω παράδειγμα να εμφανίζονται στην οθόνη τα ονόματα των ημερών και όχι οι ακέραιοι που αντιστοιχούν στις τιμές που μπορεί να πάρει η μεταβλητή `d`, θα έπρεπε να γράψουμε:

```
switch(d)
{
  case kyr: printf("Κυριακή");   break;
  case dey: printf("Δευτέρα");   break;
  case tri: printf("Τρίτη");     break;
  case tet: printf("Τετάρτη");   break;
  case pem: printf("Πέμπτη");    break;
  case par: printf("Παρασκευή"); break;
  case sab: printf("Σάββατο");   break;
}
```

9.2 ΔΟΜΕΣ

Όπως είδαμε στο Κεφάλαιο 8, τα στοιχεία ενός πίνακα πρέπει να είναι όλα του ίδιου τύπου. Όταν όμως θέλουμε να ομαδοποιήσουμε έναν αριθμό μεταβλητών οι οποίες είναι διαφορετικού τύπου και να αναφερόμαστε σ'αυτές, για ευκολία, με ένα όνομα, τότε ο πίνακας είναι ακατάλληλος. Γι'αυτές τις περιπτώσεις η γλώσσα C μας παρέχει ένα δομημένο τύπο δεδομένων που ονομάζεται *δομή* (structure). Σε μερικές γλώσσες προγραμματισμού, ιδιαίτερα στην Pascal, οι δομές ονομάζονται *εγγραφές* (records).

Ένα κλασικό παράδειγμα δομής είναι η ομαδοποίηση στοιχείων υπαλλήλων για μία εφαρμογή μισθοδοσίας. Σε κάθε υπάλληλο αντιστοιχεί ένα σύνολο στοιχείων, όπως:

Ονοματεπώνυμο

Διεύθυνση

Βαθμός

Χρόνια Υπηρεσίας

Οικογενειακή Κατάσταση κ.λ.π.

Καθένα απ'τα στοιχεία αυτά ονομάζεται *μέλος* (member) της δομής. Μερικά από τα μέλη θα μπορούσαν να είναι κι αυτά με τη σειρά τους δομές, όπως το ονοματεπώνυμο, η διεύθυνση κ.λ.π.

9.2.1 ΔΗΛΩΣΗ ΔΟΜΗΣ

Η γενική μορφή της εντολής δήλωσης ενός τύπου δομής είναι:

```
struct  αναγνωριστικό_τύπου_δομής
{
  τύπος_μέλους1  αναγνωριστικό_μέλους1;
  τύπος_μέλους2  αναγνωριστικό_μέλους2;
                   ...
  τύπος_μέλουςn  αναγνωριστικό_μέλουςn;
};
```

Η δεσμευμένη λέξη `struct` ξεκινά τη δήλωση της δομής, που είναι μία λίστα δηλώσεων κλεισμένων σε άγκιστρα. Το *αναγνωριστικό_τύπου_δομής* καλείται και *ετικέτα δομής* (structure tag). Η ετικέτα *δίνει όνομα* στη δομή και στη συνέχεια χρησιμοποιείται ως συντομογραφία για τη λίστα των δηλώσεων που υπάρχουν μέσα στα άγκιστρα.

Τα αναγνωριστικά των μελών της ίδιας δομής πρέπει να είναι διαφορετικά μεταξύ τους. Ωστόσο ένα αναγνωριστικό μέλους ή η ετικέτα της δομής, μπορεί να είναι το ίδιο με το αναγνωριστικό μίας άλλης μεταβλητής που δεν είναι μέλος αυτής της δομής.

Ένα παράδειγμα δήλωσης δομής είναι:

```
struct date
{
  int day;           /* ημέρα */
  char mon_name[13]; /* μήνας */
  int year;          /* έτος  */
};
```

Αυτή η δομή ομαδοποιεί τρεις μεταβλητές που χρησιμοποιούνται για την περιγραφή μίας ημερομηνίας της μορφής: `10 Νοεμβρίου 1996`.

Μία δήλωση `struct` ορίζει έναν τύπο. Έτσι το δεξιό άγκιστρο που τερματίζει τη λίστα των μελών, μπορεί να ακολουθείται από μία λίστα μεταβλητών, όπως άλλωστε συμβαίνει και με τους βασικούς τύπους. Δηλαδή, η:

```
struct date
{
  int day;           /* ημέρα */
  char mon_name[13]; /* μήνας */
  int year;          /* έτος  */
} a,b,c;
```

είναι συντακτικά ισοδύναμη με την:

```
int a,b,c;
```

με την έννοια ότι καθεμιά από τις παραπάνω εντολές ορίζει τα `a`, `b`, `c` ως μεταβλητές του κατονομαζόμενου τύπου, προκαλώντας δέσμευση μνήμης γι'αυτές.

Μία δήλωση δομής που δεν ακολουθείται από λίστα μεταβλητών, δε δεσμεύει χώρο στη μνήμη, απλά ορίζει έναν τύπο. Για να ορίσουμε μετά μεταβλητές αυτού

του τύπου, χρησιμοποιούμε την ετικέτα της δομής. Έτσι για παράδειγμα, με δεδομένη την παραπάνω δήλωση της `date`, η εντολή:

```
struct date d1;
```

ορίζει μία μεταβλητή `d1` τύπου `struct date` χωρίς να χρειάζεται να επαναληφθεί η λίστα των μελών.

9.2.2 ΑΝΑΦΟΡΑ ΣΤΑ ΜΕΛΗ ΜΙΑΣ ΔΟΜΗΣ

Μπορούμε να αναφερόμαστε σε συγκεκριμένα μέλη μίας δομής χρησιμοποιώντας τον τελεστή '`.`', ο οποίος καλείται *τελεστής τελεία*, ως εξής:

αναγνωριστικό_δομής.αναγνωριστικό_μέλους

Έτσι η εντολή:

```
d1.year=1996;
```

καταχωρεί στο μέλος `year` της `d1` την τιμή `1996`.

Στο παρακάτω πρόγραμμα ορίζουμε τον τύπο δομής `person` να διαθέτει δύο μέλη: `name` και `salary`. Ακολούθως δηλώνουμε δύο μεταβλητές, `person1` και `person2`, στις οποίες καταχωρούμε από το πληκτρολόγιο τα ονόματα και τους μισθούς δύο υπαλλήλων. Στη συνέχεια εμφανίζουμε στην οθόνη τα στοιχεία των δύο υπαλλήλων και τέλος βρίσκουμε και τυπώνουμε το όνομα του υπαλλήλου με το μεγαλύτερο μισθό.

```
/* struct1.c */
/* Χρήση δομών */
#include <stdlib.h>   /* για χρήση της συνάρτησης atof() */
main()
{
  struct person       /* ορισμός τύπου */
  {
    char name[40];
    float salary;
  };
  struct person person1,person2; /* δήλωση μεταβλητών */
  char numstr[10];

  printf("Δώσε το όνομα του 1ου υπαλλήλου: "); gets(person1.name);
  printf("Δώσε το μισθό του 1ου υπαλλήλου: "); gets(numstr);
  person1.salary=atof(numstr);
  printf("Δώσε το όνομα του 2ου υπαλλήλου: "); gets(person2.name);
  printf("Δώσε το μισθό του 2ου υπαλλήλου: "); gets(numstr);
  person2.salary=atof(numstr);
  clrscr();
  printf("1ος: %s : %f δρχ\n",person1.name,person1.salary);
  printf("2ος: %s : %f δρχ\n",person2.name,person2.salary);
  if(person1.salary>person2.salary)
    printf("Τα περισσότερα χρήματα παίρνει ο 1ος\n");
  else if(person2.salary>person1.salary)
    printf("Τα περισσότερα χρήματα παίρνει ο 2ος\n");
  else printf("Παίρνουν τον ίδιο μισθό\n");
}
```

Στο παραπάνω πρόγραμμα χρησιμοποιήθηκαν οι συναρτήσεις `gets()` και `atof()` για να διαβαστεί ο μισθός των υπαλλήλων (πραγματικός αριθμός) από το πληκτρολόγιο. Η συνάρτηση `atof()` (ASCII to float) δέχεται ως όρισμα ένα αλφαριθμητικό και το μετατρέπει σε αριθμό κινητής υποδιαστολής. Το αλφαριθμητικό πρέπει να περιέχει έγκυρο αριθμό κινητής υποδιαστολής. Αν δε συμβαίνει αυτό, η συνάρτηση επιστρέφει μηδέν. Έτσι η εντολή:

```
x=atof("1.23");
```

καταχωρεί στη `x` (που πρέπει να είναι τύπου `float`) την *τιμή* `1.23`.

Σχετική συνάρτηση είναι η `atoi()` (ASCII to integer) η οποία μετατρέπει το αλφαριθμητικό που δέχεται ως όρισμα, σε ακέραιο αριθμό. Πάλι το αλφαριθμητικό πρέπει να περιέχει έγκυρο ακέραιο αριθμό. Τα *πρότυπα* αυτών των συναρτήσεων υπάρχουν στο αρχείο-επικεφαλίδας `stdlib.h`.

Ο λόγος που χρησιμοποιήθηκε ο συνδυασμός των `gets()` και `atof()` αντί της `scanf()` για να διαβαστεί ένας αριθμός `float`, είναι μία ιδιομορφία που παρουσιάζει η `scanf()`. Συγκεκριμένα όταν η `scanf()` διαβάζει έναν αριθμό που πληκτρολογεί ο χρήστης πατώντας στο τέλος το πλήκτρο ENTER, αφαιρεί από το buffer του πληκτρολογίου τον αριθμό, αλλά όχι και το χαρακτήρα νέας γραμμής. Έτσι αν μετά την `scanf()` ακολουθεί μία `gets()`, βρίσκει στο buffer το χαρακτήρα νέας γραμμής και, φυσικά, το διαβάζει νομίζοντας ότι ο χρήστης έχει εισάγει ένα κενό αλφαριθμητικό. Γενικά η `scanf()` είναι μία μάλλον ογκώδης συνάρτηση, όσον αφορά το πλήθος των bytes που προσθέτει στο μεταγλωττισμένο πρόγραμμά μας κι έτσι καλό είναι να αποφεύγεται. Η συνάρτηση `gets()` προτιμάται για τα αλφαριθμητικά και ο συνδυασμός `gets()` και `atoi()` ή `atof()` για τους αριθμούς.

Εδώ να σημειώσουμε και μία δυνατότητα της Turbo C σε σχέση με το πρότυπο της C όπως καθορίστηκε από τους K&R. Στο πρότυπο, είναι αδύνατο να καταχωρηθούν οι τιμές των μελών μίας δομής σε μία άλλη μεταβλητή του ίδιου τύπου δομής, με τη χρήση μίας απλής εντολής καταχώρησης. Στην Turbo C αυτό είναι δυνατό. Δηλαδή αν οι `d1` και `d2` είναι δομές του ίδιου τύπου, μπορεί να χρησιμοποιηθεί η εντολή:

```
d1=d2;
```

η οποία έχει ως αποτέλεσμα τα μέλη της δομής `d1` να πάρουν τις ίδιες τιμές με αυτά της `d2`.

9.2.3 ΕΝΘΕΤΕΣ ΔΟΜΕΣ

Οι δομές μπορούν να περιέχουν ως μέλη άλλες δομές. Για παράδειγμα, ας θεωρήσουμε τις παρακάτω δηλώσεις δομών, που μπορούν να χρησιμοποιηθούν για αποθήκευση του ονόματος, της ημερομηνίας γέννησης και της διεύθυνσης ενός ατόμου:

```
struct name
{
  char fname[10];     /* όνομα    */
  char lname[20];     /* επώνυμο */
};
```

```
struct date
{
  int day;           /* ημέρα */
  char mon_name[13]; /* μήνας */
  int year;          /* έτος  */
};

struct address
{
  char street[15];   /* οδός    */
  int number;        /* αριθμός */
};

struct person
{
  struct name persname; /* ονοματεπώνυμο       */
  struct date birth;    /* ημερομηνία γέννησης */
  struct address house; /* διεύθυνση           */
};

struct person fred;
```

Τότε η εντολή:

```
fred.birth.year=1966;
```

Θέτει στο πεδίο χρονολογίας γέννησης της μεταβλητής `fred`, την τιμή 1966.

9.2.4 ΑΠΟΔΟΣΗ ΑΡΧΙΚΩΝ ΤΙΜΩΝ ΣΕ ΔΟΜΕΣ

Όπως και οι μεταβλητές των βασικών τύπων και οι πίνακες, έτσι και οι μεταβλητές δομής μπορούν να πάρουν αρχικές τιμές. Η γενική μορφή της εντολής απόδοσης αρχικών τιμών σε μία δομή, είναι παρόμοια μ'αυτή που χρησιμοποιείται για τους πίνακες. Κάθε μεταβλητή ακολουθείται από το σημείο = και μετά μία λίστα αρχικών τιμών, κλεισμένη σε αγκύλες. Κάθε αρχική τιμή πρέπει να είναι του ίδιου τύπου με τον τύπο του μέλους στο οποίο αντιστοιχεί. Έτσι π.χ. η παρακάτω εντολή:

```
struct date d1={28,"Απριλίου",1989};
```

θέτει αρχικές τιμές στα μέλη της `d1` που είναι τύπου `struct date` που ορίσαμε στην § 9.2.1.

9.2.5 ΔΟΜΕΣ ΩΣ ΠΑΡΑΜΕΤΡΟΙ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ

Όπως μεταβιβάζουμε σε μία συνάρτηση την *τιμή* μίας απλής μεταβλητής, με τον ίδιο τρόπο μπορούμε να μεταβιβάσουμε την *τιμή* ενός μέλους μίας δομής ή ακόμα και τις *τιμές* όλων των μελών της (ολόκληρη τη δομή).

Ως παράδειγμα θα ξαναγράψουμε το πρόγραμμα `struct1.c` της § 9.2.2 ώστε να χρησιμοποιήσουμε συναρτήσεις για την εισαγωγή των δεδομένων από το χρήστη, την εμφάνισή τους στην οθόνη και την εύρεση αυτού με το μεγαλύτερο μισθό.

```
/* struct2.c */
/* Δομές ως παράμετροι συναρτήσεων */
#include <stdlib.h>   /* για χρήση της συνάρτησης atof() */
struct person
{
  char name[40];
  float salary;
};
struct person newname(int);
void list(struct person,struct person);
void maxim(float,float);
main()
{
  struct person person1,person2;

  clrscr();
  person1=newname(1);
  person2=newname(2);
  list(person1,person2);
  maxim(person1.salary,person2.salary);
}

/* newname */
/* Εισάγει στοιχεία για έναν υπάλληλο */
struct person newname(int x)
{
  char numstr[10];
  struct person p;

  printf("Δώσε το όνομα του %dου υπαλλήλου: ",x); gets(p.name);
  printf("Δώσε το μισθό του %dου υπαλλήλου: ",x); gets(numstr);
  p.salary=atof(numstr);
  return(p);
}

/* list() */
/* Τυπώνει τα στοιχεία των δύο υπαλλήλων */
void list(struct person p1,struct person p2)
{
  clrscr();
  printf("1ος: %s : %f δρχ\n",p1.name,p1.salary);
  printf("2ος: %s : %f δρχ\n",p2.name,p2.salary);
}

/* maxim() */
/* Βρίσκει και τυπώνει ποιος παίρνει το μεγαλύτερο μισθό */
void maxim(float p1s,float p2s)
{
  if(p1s>p2s)
    printf("Τα περισσότερα χρήματα παίρνει ο 1ος\n");
  else if(p2s>p1s)
    printf("Τα περισσότερα χρήματα παίρνει ο 2ος\n");
  else printf("Παίρνουν τον ίδιο μισθό\n");
```

```
}
```

9.2.6 ΠΙΝΑΚΕΣ ΔΟΜΩΝ

Μία μέθοδος εύκολου χειρισμού ενός μεγάλου αριθμού δομών του ίδιου τύπου, είναι χρησιμοποιώντας έναν πίνακα με στοιχεία δομές. Κάθε δομή θα μπορεί να προσπελαστεί με χρήση του δείκτη-πίνακα.

Το παρακάτω πρόγραμμα χρησιμοποιεί έναν πίνακα ικανό να αποθηκεύσει 25 δομές. Κάθε δομή είναι τύπου `person`. Η εισαγωγή στοιχείων καθώς και η εμφάνιση αυτών στην οθόνη, πραγματοποιούνται με βοήθεια ενός menu επιλογών:

```
/* arstr.c */
/* Πίνακας δομών */
#include <stdlib.h>
struct person
{
  char name[40];
  float salary;
};
struct person array[25];
int n=0;
main()
{
  char select;

  do
  {
    clrscr();
    puts("1. Εισαγωγή στοιχείων υπαλλήλου");
    puts("2. Εμφάνιση λίστας υπαλλήλων");
    puts("3. Εξοδος");
    printf("Δώσε επιλογή: "); select=getche();
    switch(select)
    {
      case '1': newname(); break;
      case '2': list();    break;
    }
  }
  while(select!='3');
}

/* newname */
/* Εισάγει στοιχεία για έναν υπάλληλο */
newname()
{
  char numstr[10];
  float temp;
  clrscr();
  printf("Δώσε το όνομα του %dου υπαλλήλου: ",n+1);
  gets(array[n].name);
  printf("Δώσε το μισθό του %dου υπαλλήλου: ",n+1);
  gets(numstr);
  array[n++].salary=atof(numstr);
```

```
}

/* list() */
/* Τυπώνει τα στοιχεία των υπαλλήλων */
list()
{
  int i;

  clrscr();
  if(n<1) printf("Η λίστα είναι άδεια");
  for(i=0;i<n;i++)
    printf("%d %s %f\n",i+1,array[i].name,array[i].salary);
  getch();
}
```

9.3 ΕΝΩΣΕΙΣ

Τα μέλη μίας δομής βρίσκονται αποθηκευμένα στη μνήμη του Η/Υ *ταυτόχρονα*. Όμως μερικές φορές αυτό δεν είναι απαραίτητο διότι η φύση του προβλήματος είναι τέτοια, ώστε κάθε χρονική στιγμή *μόνο ένα απ'αυτά* να περιέχει χρήσιμη πληροφορία. Δηλαδή θα θέλαμε ο ίδιος χώρος μνήμης να χρησιμοποιείται σε διαφορετικές στιγμές από πολλές διαφορετικές μεταβλητές, οι οποίες μπορεί να μην είναι καν του ίδιου τύπου. Αυτό επιτυγχάνεται με τη χρησιμοποίηση των *ενώσεων* (unions). Οι ενώσεις είναι ανάλογες με τις *μεταβαλλόμενες εγγραφές* (variant records) της Pascal.

Η γενική μορφή της εντολής δήλωσης ενός τύπου ένωσης είναι ανάλογη με αυτή της δομής:

```
union  αναγνωριστικό_τύπου_ένωσης
{
  τύπος_μέλους1  αναγνωριστικό_μέλους1;
  τύπος_μέλους2  αναγνωριστικό_μέλους2;
                    ...
  τύπος_μέλουςn  αναγνωριστικό_μέλουςn;
};
```

Η δήλωση μεταβλητών τύπου ένωσης γίνεται ακριβώς με τον ίδιο τρόπο όπως για τις δομές. Η αναφορά στα μέλη μίας ένωσης γίνεται με τον ίδιο τρόπο όπως και στις δομές.

Ο χώρος μνήμης που καταλαμβάνει μία ένωση είναι τόσο μεγάλος, ώστε να χωράει το μεγαλύτερο από τους τύπους των μελών της. Είναι ευθύνη του προγραμματιστή να παρακολουθεί ποιος τύπος είναι αποθηκευμένος κάθε χρονική στιγμή σε μία μεταβλητή τύπου ένωσης. Αν κάτι αποθηκευτεί με κάποιο τύπο και ανακτηθεί με άλλο, τα αποτελέσματα είναι απροσδιόριστα.

Στο παρακάτω παράδειγμα δηλώνουμε έναν τύπο ένωσης (`intfloat`) και μία μεταβλητή ένωσης (`x`). Χρησιμοποιείται στο πρόγραμμα ο μοναδικός (unary) τελεστής `sizeof` ο οποίος εφαρμοζόμενος σε μία μεταβλητή ή σε έναν τύπο, δίνει τον αριθμό των bytes που καταλαμβάνονται από τη μεταβλητή ή από τον τύπο.

```
/* union.c */
/* Χρήση ενώσεων */
main()
{
  union intflo        /* ορισμός τύπου */
  {
    int intnum;
    float floatnum;
  };
  union intflo x;     /* δήλωση μεταβλητής */

  printf("Bytes που καταλαμβάνει η ένωση: %d\n",sizeof(union intflo));
  x.intnum=321;
  printf("x.intnum=%d\n",x.intnum);
  x.floatnum=123.456;
  printf("x.floatnum=%.3f\n",x.floatnum);
}
```

Η έξοδος αυτού του προγράμματος είναι:

```
Bytes που καταλαμβάνει η ένωση: 4
x.intnum=321
x.floatnum=123.456
```

Παρόλο που η ένωση έχει ως μέλη έναν ακέραιο και έναν πραγματικό αριθμό, το μέγεθός της είναι μόνο 4 bytes. Έτσι είναι αρκετά μεγάλη για να χωρέσει το ένα ή το άλλο μέλος, αλλά *όχι και τα δύο ταυτόχρονα*.

9.4 Η ΔΗΛΩΣΗ `typedef`

Η C μας παρέχει τη δυνατότητα να επανακαθορίσουμε το όνομα ενός ήδη υπάρχοντος τύπου δεδομένων, χρησιμοποιώντας τη δήλωση `typedef`. Μία δήλωση `typedef` είναι της μορφής:

typedef *αναγνωριστικό_τύπου νέο_αναγνωριστικό_τύπου;*

Χρησιμοποιώντας τη δήλωση `typedef`, στην πραγματικότητα δε δημιουργούμε έναν καινούργιο τύπο, αλλά ορίζουμε ένα καινούργιο όνομα (συνώνυμο) για έναν υπάρχοντα τύπο. Η χρήση της `typedef` μπορεί να συντελέσει στο να γίνουν τα προγράμματα πιο ξεκάθαρα γιατί το όνομα ενός τύπου μπορεί να γίνει πιο μικρό και πιο νοηματικό.

Ως παράδειγμα, ας θεωρήσουμε την επόμενη εντολή με την οποία επανακαθορίζεται ο τύπος `unsigned char` (βλέπε Πίνακα 4.1) σε τύπο με όνομα `BYTE`:

```
typedef unsigned char BYTE;
```

Στη συνέχεια μπορούμε να δηλώσουμε μεταβλητές τύπου `unsigned char` γράφοντας:

```
BYTE b1,b2;
```

αντί για:

```
unsigned char b1,b2;
```

Το παρακάτω πρόγραμμα είναι το πρόγραμμα `enums1.c` της § 9.1. Η αλλαγή που κάνουμε είναι να μετονομάσουμε τον τύπο `enum colour` σε `COLOURS`, για ευκολία στη δήλωση μεταβλητών:

```
/* enums1.c */
/* Επίδειξη απαριθμήσεων και typedef */
main()
{
  enum colour {red,green,yellow}; /* ορισμός τύπου enum colour       */
  typedef enum colour COLOURS;    /* επανακαθορισμός του τύπου
                                     enum colour σε COLOURS          */
  COLOURS a,b,c;                  /* δήλωση μεταβλητών τύπου COLOURS */
  int i;

  a=red; b=green; c=yellow;
  i=a+b;
  printf("%d\n",i);
  if(a<c)
    printf("Η τιμή red είναι μικρότερη από την τιμή yellow\n");
}
```

9.5 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί πρόγραμμα το οποίο θα δέχεται από το πληκτρολόγιο την ωριαία αντιμισθία και τις ώρες εργασίας για κάθε ημέρα της εβδομάδας και να υπολογίζει τον εβδομαδιαίο μισθό ενός ωρομισθίου υπαλλήλου. Υποθέστε ότι ο υπάλληλος λαμβάνει ένα επιπλέον επίδομα 500 δρχ για κάθε ώρα εργασίας και ότι οι ώρες εργασίας του Σαββάτου υπολογίζονται αυξημένες κατά το ήμισυ ενώ τις Κυριακής υπολογίζονται διπλές.

```
/* enums2.c */
/* Υπολογισμός εβδομαδιαίου μισθού ωρομισθίου υπαλλήλου */
#define BONUS 500
main()
{
  typedef enum day {dey,tri,tet,pem,par,sab,kyr} DAYS;
  DAYS workday;
  float hour_salary,hours,salary=0.0,bonus=0.0;

  printf("Δώσε το ωρομίσθιο: ");
  scanf("%f",&hour_salary);
  printf("Δώσε ώρες εργασίας, από Δευτέρα ως Κυριακή\n");
  for(workday=dey;workday<=kyr;workday++)
  {
    switch(workday)
    {
      case dey: printf("Δευτέρα  : "); scanf("%f",&hours); break;
      case tri: printf("Τρίτη    : "); scanf("%f",&hours); break;
      case tet: printf("Τετάρτη  : "); scanf("%f",&hours); break;
      case pem: printf("Πέμπτη   : "); scanf("%f",&hours); break;
      case par: printf("Παρασκευή: "); scanf("%f",&hours); break;
```

```
      case sab: printf("Σάββατο  : "); scanf("%f",&hours);
                                       hours*=1.5;         break;
      case kyr: printf("Κυριακή  : "); scanf("%f",&hours);
                                       hours*=2;           break;
    }
    bonus =bonus+hours*BONUS;
    salary=salary+hours*hour_salary;
  }
  salary=salary+bonus;
  printf("Ο εβδομαδιαίος μισθός είναι %6.2f", salary);
}
```

2. Να γραφεί πρόγραμμα το οποίο με τη βοήθεια ενός menu να επιτελεί τις παρακάτω εργασίες:

α) Να δέχεται από το πληκτρολόγιο και να καταχωρεί στη μνήμη τα στοιχεία (ονοματεπώνυμο, φύλο, βαθμό) το πολύ 50 μαθητών.

β) Να παρέχει τη δυνατότητα αναζήτησης οποιουδήποτε μαθητή με βάση το ονοματεπώνυμο αυτού.

γ) Να δίνει τη δυνατότητα ταξινόμησης της λίστας όλων των μαθητών, με βάση το βαθμό τους (από το μικρότερο προς το μεγαλύτερο).

δ) Να παρουσιάζει στην οθόνη όλους τους μαθητές της λίστας με όλα τους τα στοιχεία.

```
/* struct.c */
/* Πίνακας δομών */
#define LIM 50
#include <stdlib.h>
#include <stdio.h>
struct person        /* ορισμός δομής */
{
  char name[40];
  char sex;
  float score;
};
typedef struct person STUDENT;
STUDENT array[LIM];  /* πίνακας δομών */
int n=0;             /* το n μετράει το πλήθος των μαθητών
                        που έχουν εισαχθεί στον πίνακα */

main()
{
  char select, student[40];

  do
  {
    clrscr();
    puts("1. Εισαγωγή μαθητή");
    puts("2. Αναζήτηση μαθητή");
    puts("3. Ταξινόμηση μαθητών με βάση το βαθμό");
    puts("4. Εμφάνιση λίστας μαθητών");
    puts("5. Εξοδος");
    printf("Δώσε επιλογή: ");
```

```
    select=getche();
    switch(select)
    {
      case '1': clrscr();
                puts("ΕΙΣΑΓΩΓΗ ΜΑΘΗΤΗ");
                newname();
                break;
      case '2': clrscr();
                puts("ΑΝΑΖΗΤΗΣΗ ΜΑΘΗΤΗ");
                if(n<1) puts("Η λίστα είναι άδεια");
                else
                {
                  printf("Δώσε ονοματεπώνυμο: ");
                  gets(student);
                  search(student);
                }
                getch();
                break;
      case '3': clrscr();
                puts("ΤΑΞΙΝΟΜΗΣΗ ΛΙΣΤΑΣ");
                sort();
                puts("Η ταξινόμηση έγινε");
                getch();
                break;
      case '4': clrscr();
                puts("ΕΜΦΑΝΙΣΗ ΛΙΣΤΑΣ");
                list();
                getch();
                break;
    }
  }
  while(select!='5');
}


/* newname */
/* Εισαγωγή στοιχείων για ένα μαθητή */
newname()
{
  char numstr[81];


  if(n<LIM)
  {
    printf("Δώσε το όνομα του %dου μαθητή: ",n+1);
    gets(array[n].name);
    printf("Δώσε το φύλλο (Α/Γ) του %dου μαθητή: ",n+1);
    array[n].sex=getche();
    getch();
    printf("\nΔώσε το βαθμό του %dου μαθητή: ",n+1);
    gets(numstr);
    array[n].score=atof(numstr);
    n++;
  }
```

```
  else
  {
    puts("Η λίστα είναι πλήρης");
    getch();
  }
}


/* search */
/* Αναζήτηση μαθητή */
search(char str[])
{
  int found=0,i=0;
  if(n<1) puts("Η λίστα είναι άδεια");
  else
  {
    while(i<n && !found)
    {
      if(strcmp(array[i].name,str)==0) found=1;
      else i++;
    }
    if(found)
    {
      puts("Ο μαθητής βρέθηκε");
      printf("%s %c %0.2f",array[i].name,array[i].sex,array[i].score);
    }
    else puts("Ο μαθητής δε βρέθηκε");
  }
}


/* sort */
/* Ταξινόμηση λίστας μαθητών με τη μέθοδο της επιλογής */
sort()
{
  int i,j,k,min;
  STUDENT temp;

  for(i=0;i<n-1;i++)
  {
    min=i;
    for(j=i+1;j<n;j++)
      if(array[j].score<array[min].score) min=j;
    if(i!=min)
    {
      temp=array[i];
      array[i]=array[min];
      array[min]=temp;
    }
  }
}
```

```
/* list() */
/* Εκτύπωση στοιχείων όλων των μαθητών */
list()
{
  int i;

  if(n<1) printf("Η λίστα είναι άδεια");
  for(i=0;i<n;i++)
    printf("%d %s %c %0.2f\n",i+1,array[i].name,array[i].sex,
           array[i].score);
}
```

3. Δημιουργείστε έναν τύπο ένωσης για τέσσερα γεωμετρικά σχήματα: κύκλο, τετράγωνο, ορθογώνιο και τρίγωνο. Για τον κύκλο η ένωση θα πρέπει να αποθηκεύει την ακτίνα του, για το τετράγωνο το μήκος της πλευράς του, για το ορθογώνιο τα μήκη των δύο πλευρών του και για το τρίγωνο τα μήκη των τριών πλευρών του.
Στη συνέχεια γράψτε πρόγραμμα το οποίο:

α) Να διαβάζει ένα από τα γράμματα Κ (κύκλος), Τ (τετράγωνο), Ο (ορθογώνιο), R (τρίγωνο) και τις κατάλληλες αριθμητικές τιμές για το εκάστοτε σχήμα και

β) Να υπολογίζει και τυπώνει το μήκος της περιμέτρου του σχήματος.

```
/* union1.c */
/* Χρήση ένωσης για υπολογισμό περιμέτρων γεωμετρικών σχημάτων */
#include <math.h>
struct two
{
  float a,b;
};
struct three
{
  float a,b,c;
};
union values
{
  float r;
  struct two   twod;
  struct three thrd;
};
main()
{
  union values s;
  char c;
  printf("Κύκλος K\nΤετράγωνο T\nΟρθογώνιο O\nΤρίγωνο R\n
         Επιλογή: ");
  scanf("%c",&c);
  switch(c)
  {
    case 'K' : printf("Δώσε ακτίνα: ");
               scanf("%f",&s.r);
               printf("Περίμετρος=%f",2*M_PI*s.r);
               break;
```

```
    case 'T' : printf("Δώσε το μήκος της πλευράς: ");
               scanf("%f",&s.r);
               printf("Περίμετρος=%f",4*s.r);
               break;
    case 'O' : printf("Δώσε τα μήκη των πλευρών: ");
               scanf("%f %f",&s.twod.a,&s.twod.b);
               printf("Περίμετρος=%f",2*(s.twod.a+s.twod.b));
               break;
    case 'R' : printf("Δώσε τα μήκη των πλευρών: ");
               scanf("%f %f %f",&s.thrd.a,&s.thrd.b,&s.thrd.c);
               printf("Περίμετρος=%f",s.thrd.a+s.thrd.b+s.thrd.c);
               break;
  }
}
```

9.6 ΑΣΚΗΣΕΙΣ

1. Μία δομή με μέλη: Έτος, Μήνας, Ημέρα, μπορεί να χρησιμοποιηθεί για την καταχώριση ημερομηνιών. Γράψτε ένα πρόγραμμα που να χρησιμοποιεί τέτοιου είδους δομές και να κάνει τα εξής: να δέχεται τη σημερινή ημερομηνία και την ημερομηνία γέννησης κάποιου ατόμου και να υπολογίζει και εμφανίζει στην οθόνη την ηλικία του.

2. Να γραφεί πρόγραμμα το οποίο να δέχεται ένα σύνολο ονομάτων (π.χ. 10) για το καθένα απ'τα οποία υπάρχει επιπλέον η πληροφορία για το φύλο, ηλικία και χρώμα μαλλιών. Δοθέντων του φύλλου, του χρώματος μαλλιών και μιας περιοχής ηλικίας, το πρόγραμμα να τυπώνει τα ονόματα αυτών που πληρούν τις προϋποθέσεις.

3. Δημιουργήστε έναν κατάλληλο τύπο δεδομένων ικανό να αποθηκεύει ένα μιγαδικό αριθμό (a+ib). Στη συνέχεια γράψτε κατάλληλες συναρτήσεις οι οποίες να δέχονται ως παραμέτρους δύο μιγαδικούς αριθμούς και να επιστρέφουν τα αποτελέσματα της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης αυτών, όπως επίσης και το αποτέλεσμα της ύψωσης σε εκθέτη ενός μιγαδικού:

   $(a + ib) + (c + id) = (a + c) + i(b + d)$

   $(a + ib) - (c + id) = (a - c) + i(b - d)$

   $(a + ib) * (c + id) = (ac - bd) + i(ad + bc)$

   $$\frac{a + ib}{c + id} = \frac{ac + bd}{c^2 + d^2} + i\,\frac{bc - ad}{c^2 + d^2} \quad \text{αν} \quad c^2 + d^2 \neq 0$$

   $(a + ib)^n = p^n(\cos n\theta + i \sin n\theta)$ όπου $p = \sqrt{a^2 + b^2}$ $\theta = \tan^{-1}(b / a)$

4. Η εξίσωση ευθείας που διέρχεται από δύο σημεία $P_1(x_1, y_1)$ και $P_2(x_2, y_2)$ είναι:

   $y - y_1 = m(x - x_1)$ όπου $m = (y_2 - y_1) / (x_2 - x_1)$

   (αν $x_1=x_2$ τότε η ευθεία είναι κάθετη στον άξονα Ox και έχει εξίσωση: $x=x_1$).

Το μήκος του ευθύγραμμου τμήματος P_1 P_2 δίνεται από τον τύπο:

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Γράψτε ένα πρόγραμμα που τυπώνει την εξίσωση της ευθείας καθώς και το μήκος του ευθύγραμμου τμήματος που διέρχεται από τα σημεία P_1 και P_2 , όπου οι συντεταγμένες αυτών θα δίνονται από το πληκτρολόγιο.

5. Να γραφούν συναρτήσεις που να μετατρέπουν τις καρτεσιανές συντεταγμένες σε πολικές και αντίστροφα. Οι τύποι μετατροπής από πολικές σε καρτεσιανές συντεταγμένες είναι:

$$x = r \cos\theta, \quad y = r \sin\theta$$

και οι τύποι μετατροπής από καρτεσιανές σε πολικές, είναι:

$$r = \sqrt{x^2 + y^2} \quad \text{και} \quad \theta = \tan^{-1}(y / x) \quad \text{αν} \quad x \neq 0$$

Στη συνέχεια χρησιμοποιείστε αυτές τις συναρτήσεις σ'ένα πρόγραμμα που να δέχεται τις συντεταγμένες ενός σημείου καθώς και ένα χαρακτήρα που δηλώνει το είδος τους (Κ: καρτεσιανές, Ρ: πολικές) και να τις μετατρέπει από καρτεσιανές σε πολικές ή αντίστροφα.

ΚΕΦΑΛΑΙΟ 10
ΔΕΙΚΤΕΣ

Ένας *δείκτης* (pointer) είναι μία μεταβλητή που περιέχει μία διεύθυνση μνήμης. Στις περισσότερες περιπτώσεις, αυτή η διεύθυνση είναι η θέση στη μνήμη μίας άλλης μεταβλητής. Όταν μία μεταβλητή περιέχει τη διεύθυνση μίας άλλης μεταβλητής, τότε λέμε ότι η πρώτη μεταβλητή *δείχνει* τη δεύτερη. Οι δείκτες χρησιμοποιούνται στον προγραμματισμό για πολλούς λόγους. Οι σημαντικότεροι απ'αυτούς είναι:

- Για την επιστροφή περισσότερων της μίας τιμών από μία συνάρτηση.
- Για τη μεταβίβαση πινάκων και αλφαριθμητικών σε συναρτήσεις.
- Για εύκολο και γρήγορο χειρισμό πινάκων.
- Για τη δημιουργία πολύπλοκων δομών δεδομένων, όπως *συνδεδεμένες λίστες* και *δένδρα*.
- Για άμεση πρόσβαση σε συγκεκριμένες θέσεις μνήμης.

Όπως καταλαβαίνουμε από τα παραπάνω, ήδη έχουμε χρησιμοποιήσει δείκτες στα προγράμματα που έχουμε γράψει έως τώρα. Στο Κεφάλαιο 8, όταν χρησιμοποιούσαμε πίνακες ως παραμέτρους σε συναρτήσεις, στην ουσία μεταβιβάζαμε στη συνάρτηση τη *διεύθυνση* του πίνακα. Αυτή η διεύθυνση είναι ένα παράδειγμα μίας *σταθεράς δείκτη*. Η σταθερά δείκτη είναι μία διεύθυνση, η μεταβλητή δείκτη είναι μία θέση μνήμης που χρησιμοποιείται για αποθήκευση διευθύνσεων.

10.1 ΔΗΛΩΣΗ ΜΕΤΑΒΛΗΤΗΣ ΔΕΙΚΤΗ

Όπως κάθε μεταβλητή, μία μεταβλητή δείκτη πρέπει να δηλώνεται πριν χρησιμοποιηθεί. Η γενική μορφή της εντολής δήλωσης μίας μεταβλητής δείκτη είναι:

τύπος ***αναγνωριστικό;***

όπου *τύπος* είναι ο τύπος των δεδομένων που θα δείχνει ο δείκτης. Μία μεταβλητή δείκτη δεσμεύει 2 bytes μνήμης.

Ένα παράδειγμα δήλωσης δεικτών είναι:

```
int *px, *py;
```

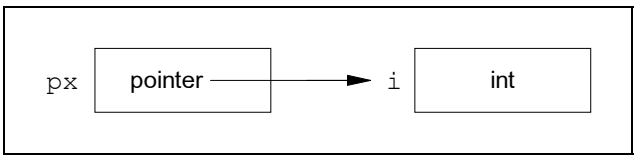
Η δήλωση αυτή δεσμεύει 2 bytes στα οποία μπορεί να αποθηκευτεί η διεύθυνση μίας μεταβλητής ακεραίου τύπου και δίνει σ'αυτά τα 2 bytes το όνομα `px`. Δεσμεύει επίσης άλλα 2 bytes για αποθήκευση διεύθυνσης μεταβλητής ακεραίου τύπου και δίνει σ'αυτά το όνομα `py`.

10.2 ΤΕΛΕΣΤΕΣ ΔΕΙΚΤΩΝ

Η C μας παρέχει δύο τελεστές σχετικούς με τους δείκτες, το τελεστή `&` και τον `*`. Ο `&` λέγεται *τελεστής διεύθυνσης* και είναι ο μοναδικός (unary) τελεστής που επιστρέφει τη διεύθυνση του τελεστέου του, όπως έχουμε δει και σε προηγούμενα κεφάλαια. Για παράδειγμα η εντολή:

```
px=&i;
```

(όπου υποθέτουμε ότι η μεταβλητή `i` έχει δηλωθεί ως ακεραίου τύπου) καταχωρεί στην `px` (που έχει δηλωθεί παραπάνω ως δείκτης σε ακέραιο) τη διεύθυνση της μεταβλητής `i`. Έτσι η `px` "δείχνει" την `i`, Σχήμα 10.1.



Σχήμα 10.1: *Δείκτης σε ακέραιο*

Ο δεύτερος τελεστής, `*`, είναι και αυτός μοναδικός και είναι συμπληρωματικός του `&`. Λέγεται τελεστής *έμμεσης αναφοράς* ή *ανακατεύθυνσης* (indirection ή dereference) και όταν εφαρμόζεται σε μία μεταβλητή δείκτη, επιστρέφει την *τιμή* της μεταβλητής που δείχνει ο δείκτης. Για παράδειγμα η εντολή:

```
j=*px;
```

καταχωρεί στη μεταβλητή `j` (που υποθέτουμε ότι είναι ακεραίου τύπου) το περιεχόμενο της μεταβλητής που δείχνει η `px`, δηλαδή την τιμή της `i`.

Δυστυχώς, στη C, το σύμβολο του πολλαπλασιασμού και ο δεύτερος τελεστής είναι ίδια. Όμως τόσο ο `*` όσο και ο `&` έχουν μεγαλύτερη προτεραιότητα από τους αριθμητικούς τελεστές εκτός από το μοναδικό μείον, με το οποίο έχουν την ίδια προτεραιότητα, (Πίνακας 4.5). Επίσης το `*` χρησιμοποιείται σε εντολές δήλωσης μεταβλητών δεικτών:

```
int *ptr;
```

όσο και σε άλλες εντολές:

```
*ptr=3;
```

με διαφορετική σημασία σε καθεμιά απ'αυτές τις δύο ομάδες εντολών. Όταν το `*` χρησιμοποιείται σε μία δήλωση μεταβλητής σημαίνει *"μεταβλητή τύπου δείκτη σε..."*, ενώ σε άλλες εντολές σημαίνει *"τιμή της μεταβλητής που δείχνεται από το δείκτη..."*.

Για να γίνουν κατανοητά τα παραπάνω, ας δούμε το παρακάτω πρόγραμμα σε συνδυασμό με το Σχήμα 10.2:

```
main()
{
  int x,y,*pint;       /*δηλώνουμε τους ακέραιους x,y και το
                         δείκτη σε ακέραιο pint*/
  char str[7],*pchar; /*δηλώνουμε τον πίνακα str και το
                         δείκτη σε χαρακτήρα pchar*/
                     /* Σχήμα 10.2 α */
```

```
  x=3;        /*δίνουμε στο x την τιμή 3*/
  pint=&x;    /*δίνουμε στο pint τη διεύθυνση του x*/
                      /* Σχήμα 10.2 β */


  y=*pint;    /*ο y παίρνει το περιεχόμενο της θέσης μνήμης
                που δείχνει ο pint*/
                      /* Σχήμα 10.2 γ */


  *pint=10;   /*η μεταβλητή που δείχνει ο pint, δηλαδή ο x,
                παίρνει την τιμή 10*/
                      /* Σχήμα 10.2 δ */


  *pint+=20; /*προστίθεται 20 στη μεταβλητή που δείχνει ο pint*/
  pchar=str; /*ο pchar δείχνει το str*/
                      /* Σχήμα 10.2 ε */


  strcpy(str,"ΑΛΦΑ"); /*το str παίρνει τιμή το "ΑΛΦΑ"*/
  puts(str);          /*το οποίο και τυπώνεται*/
                     /* Σχήμα 10.2 στ */


  printf("%p",pint);  /*τυπώνεται η τιμή του δείκτη pint, δηλαδή η
                        διεύθυνση του x και όχι η τιμή του x*/
  printf("\n%d",x);   /*τυπώνεται η τιμή του x, δηλαδή 30*/
}
```

10.3 ΠΑΡΑΣΤΑΣΕΙΣ - ΔΕΙΚΤΕΣ

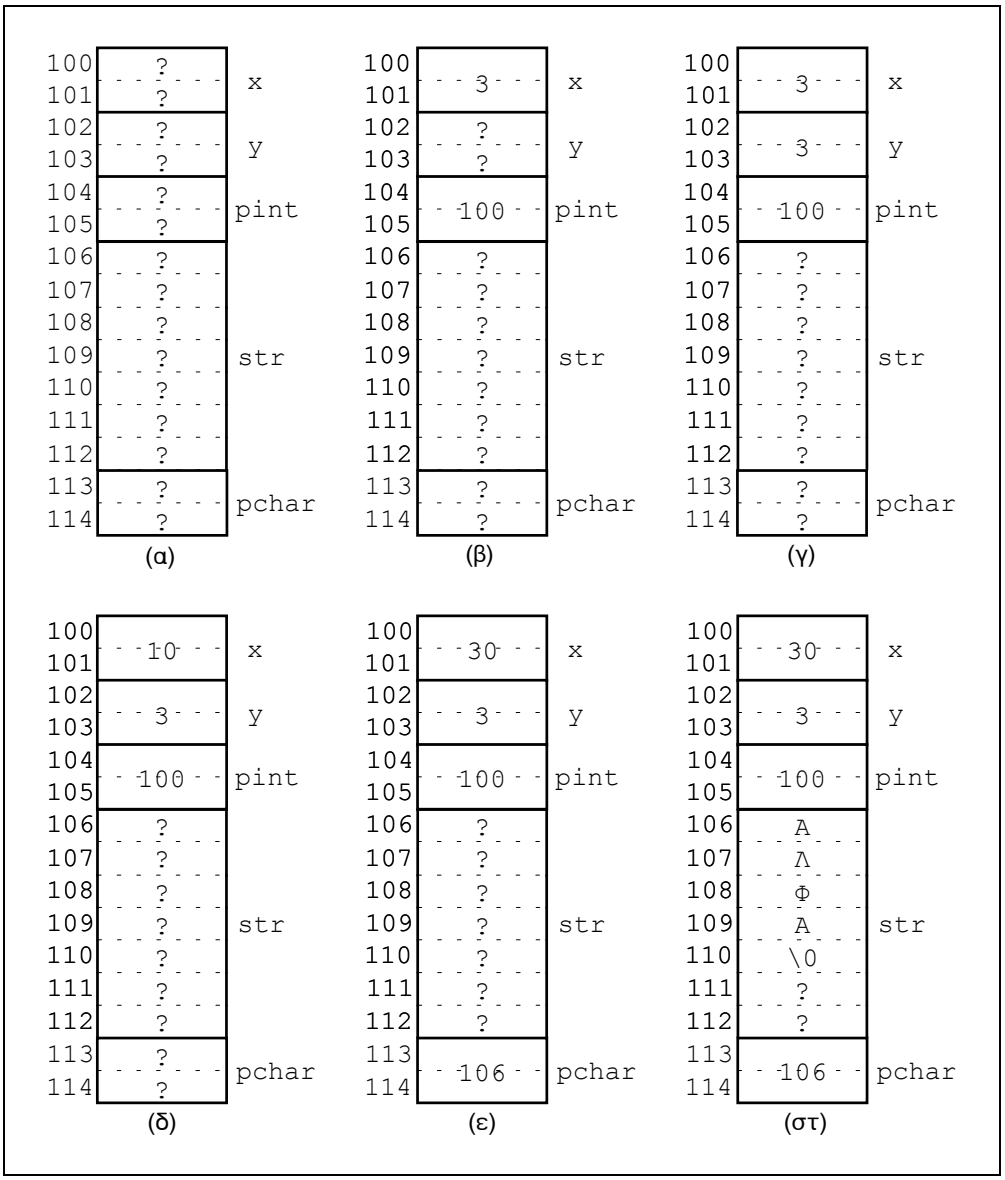
Οι παραστάσεις που περιλαμβάνουν δείκτες, υπακούουν στους ίδιους κανόνες με αυτούς που υπακούουν όλες οι παραστάσεις της C.

Στη συνέχεια εξετάζουμε μερικές ειδικές πλευρές των παραστάσεων - δεικτών.

10.3.1 ΑΠΟΔΟΣΗ ΤΙΜΩΝ ΣΕ ΔΕΙΚΤΕΣ

Όπως και με οποιαδήποτε άλλη μεταβλητή, μπορούμε να αποδώσουμε την τιμή ενός δείκτη σε κάποιον άλλο δείκτη, με μία παράσταση καταχώρησης, π.χ.:

```
int x,*p1,*p2;
p1=&x;    /* ο p1 δείχνει την x */
p2=p1;    /* ο p2 δείχνει εκεί που δείχνει και ο p1 */
```



Σχήμα 10.2: *Η κατάσταση στη μνήμη κατά τη διάρκεια εκτέλεσης του προγράμματος*

10.3.2 ΑΡΙΘΜΗΤΙΚΗ ΔΕΙΚΤΩΝ

Εφόσον οι διευθύνσεις που περιέχονται στους δείκτες είναι ακέραιες τιμές, μπορούμε να κάνουμε μ'αυτές πράξεις. Οι πράξεις αυτές είναι η αύξηση και η μείωση κατά ένα (++, −−), η πρόσθεση και η αφαίρεση (+, −). Για να καταλάβουμε τι συμβαίνει στην αριθμητική των δεικτών, ας θεωρήσουμε ότι ο `ptr` είναι ένας δείκτης σε έναν ακέραιο με τρέχουσα τιμή 1000. Μετά την εντολή:

```
ptr++;
```

η τιμή του `ptr` δε θα είναι 1001 αλλά 1002 !. Κάθε φορά που ο `ptr` υφίσταται μία μοναδιαία αύξηση ή μείωση, δείχνει τον επόμενο ή τον προηγούμενο *ακέραιο* και εφόσον οι ακέραιοι καταλαμβάνουν 2 bytes μνήμης, η τιμή του `ptr` θα αυξάνεται ή θα μειώνεται κατά 2. Δεν περιοριζόμαστε μόνο σε μοναδιαίες

αυξήσεις ή μειώσεις. Μπορούμε να προσθέσουμε ή να αφαιρέσουμε ακέραιους σε δείκτες. Έτσι μετά την εντολή:

```
ptr+=9;
```

η τιμή του `ptr` γίνεται 1018.

Γενικά, λοιπόν, αν `ptr` είναι δείκτης και `i` ακέραιος, τότε η πράξη:

```
ptr ± i
```

είναι ισοδύναμη με την:

```
(char*)ptr ± sizeof(*ptr)*i
```

δηλαδή: μετατροπή του δείκτη σε δείκτη-σε-χαρακτήρα (1 byte) και πρόσθεση ή αφαίρεση `i` φορές το μέγεθος (σε bytes) της μεταβλητής που δείχνει ο `ptr`.

Μπορούμε να προσθέτουμε δείκτες μεταξύ τους, καθώς και να αφαιρούμε. Ειδικά η αφαίρεση δύο δεικτών, που δείχνουν μεταβλητές του ίδιου τύπου, χρησιμοποιείται για να βρούμε την απόσταση των δύο μεταβλητών στη μνήμη.

10.3.3 ΣΥΓΚΡΙΣΗ ΔΕΙΚΤΩΝ

Μπορούμε να συγκρίνουμε δύο δείκτες χρησιμοποιώντας τους συσχετιστικούς τελεστές του Πίνακα 4.3. Για παράδειγμα, αν έχουμε δύο δείκτες, `ptr1` και `ptr2`, η παρακάτω εντολή είναι αποδεκτή:

```
if(ptr1<ptr2)
  printf("Ο ptr1 δείχνει σε χαμηλότερη διεύθυνση
          από τον ptr2");
```

Μία ενδιαφέρουσα χρήση των δεικτών φαίνεται στο παρακάτω παράδειγμα, το οποίο εξετάζει τα περιεχόμενα της μνήμης RAM του Η/Υ μας. Το πρόγραμμα δέχεται από εμάς την αρχική διεύθυνση μνήμης απ'όπου θέλουμε να αρχίσει την εξέταση και στη συνέχεια εμφανίζει τα περιεχόμενα του κάθε byte της μνήμης σε δεκαεξαδική μορφή. Στο πρόγραμμα έχει ληφθεί πρόνοια ώστε όταν γεμίσει μία οθόνη με 23 γραμμές (σε κάθε γραμμή εμφανίζονται τα περιεχόμενα 8 bytes), να αναμένεται το πάτημα κάποιου πλήκτρου από το χρήστη, αν το πλήκτρο είναι το 'x' το πρόγραμμα σταματά, διαφορετικά συνεχίζει.

Στο πρόγραμμα βλέπουμε επίσης τη χρήση της δεσμευμένης λέξης `far`, οποία επιτρέπει στους δείκτες να αναφέρονται σε θέσεις που δε βρίσκονται στο ίδιο τμήμα μνήμης (segment) που βρίσκονται και οι δείκτες.

```
/* dispmem.c */
/* Παρουσίαση των περιεχομένων της μνήμης, αρχίζοντας από
   κάποια διεύθυνση που δίνει ο χρήστης */
main()
{
  unsigned long int start;

  printf("Δώσε την αρχική διεύθυνση: ");
  scanf("%U",&start);
  dump(start);
}
```

```
dump(unsigned long int st)
{
  char far *p;
  int c,l;
  char ch;

  p=(char far *) st;
  do
  {
    for(l=1;l<24;l++)
    {
      for(c=1;c<=8;c++,p++)
        printf("%5x",*p);
      printf("\n");
    }
    ch=getch();
  }
  while(ch!='x'); /* δίνοντας x σταματάμε την εκτέλεση */
}
```

10.4 ΔΕΙΚΤΕΣ ΚΑΙ ΣΥΝΑΡΤΗΣΕΙΣ

Είδαμε στο Κεφάλαιο 7 ότι όταν καλούμε μία συνάρτηση με πραγματικές παραμέτρους, οι *τιμές* των πραγματικών αντιγράφονται στις αντίστοιχες *τυπικές παραμέτρους* της συνάρτησης (κλήση κατ'αξία: call by value). Έτσι η συνάρτηση δε μπορεί να αλλάξει τις τιμές των πραγματικών παραμέτρων και μπορεί να επιστρέψει *μόνο μία τιμή* στην καλούσα συνάρτηση.

Όμως αν θέλουμε μία συνάρτηση να αλλάζει τις τιμές κάποιων μεταβλητών της καλούσας συνάρτησης, τότε αντί να μεταβιβάσουμε τις μεταβλητές αυτές ως πραγματικές (στην περίπτωση αυτή η συνάρτηση δε μπορεί να αλλάξει τις τιμές τους) μεταβιβάζουμε τις *διευθύνσεις* αυτών. Έτσι η συνάρτηση δε μπορεί να αλλάξει τις διευθύνσεις, πράγμα που δεν έχει άλλωστε νόημα, αλλά μπορεί να τις χρησιμοποιήσει για *έμμεση* προσπέλαση στις μεταβλητές κι έτσι να αλλάξει τις τιμές τους. Με τον τρόπο αυτό η καλούμενη συνάρτηση επικοινωνεί με την καλούσα με περισσότερες από μία *τιμές*.

Στο παρακάτω πρόγραμμα χρησιμοποιείται μία συνάρτηση, `swap()`, η οποία δέχεται ως πραγματικές παραμέτρους τις *διευθύνσεις* δύο μεταβλητών και αλλάζει αμοιβαία τις τιμές τους:

```
/* swap.c */
/* Μεταβίβαση διευθύνσεων σε συνάρτηση */
main()
{
  int x,y;

  scanf("%d %d",&x,&y);
  swap(&x,&y);          /* μεταβιβάζουμε στη συνάρτηση
                           τις διευθύνσεις των x,y     */
  printf("x=%d, y=%d\n",x,y);
}
```

```
/* swap() */
/* Εναλλάσσει τις τιμές δύο μεταβλητών */
swap(int *px, int *py)
{
  int temp;
  temp=*px;
  *px=*py;
  *py=temp;
}
```

10.5 ΔΕΙΚΤΕΣ ΚΑΙ ΠΙΝΑΚΕΣ

Στη γλώσσα C υπάρχει μία ισχυρή σχέση μεταξύ των πινάκων και των δεικτών. Όπως ήδη είδαμε στο Κεφάλαιο 8, το αναγνωριστικό ενός πίνακα είναι το *όνομα* μίας διεύθυνση (της διεύθυνσης του πρώτου στοιχείου του πίνακα) με άλλα λόγια ένας *δείκτης* (pointer). Έτσι η δήλωση:

```
int a[10];
```

σημαίνει ότι το `a` είναι *το όνομα της διεύθυνσης* της πρώτης, από 10, μεταβλητής τύπου `int`. Κατά συνέπεια, με λίγη σκέψη προκύπτει ότι:

- Το `a` ισοδυναμεί με το `&a[0]`
- Το `*a` ισοδυναμεί με το `a[0]`
- Το `a+1` ισοδυναμεί με το `&a[1]`
- Το `a+n` ισοδυναμεί με το `&a[n]` (όπου `n`: ακέραιος)
- Το `a[n]` ισοδυναμεί με το `*(a+n)` (όπου `n`: ακέραιος)

Με βάση τα παραπάνω, τα δύο προγράμματα που ακολουθούν κάνουν την ίδια δουλειά: τυπώνουν τις τιμές ενός πίνακα. Το πρώτο χρησιμοποιεί τη γνωστή σε μας μέθοδο προσπέλασης στα στοιχεία ενός πίνακα με τη χρήση του *δείκτη-πίνακα* (subscript) ενώ το δεύτερο χρησιμοποιεί το γεγονός ότι το αναγνωριστικό ενός πίνακα είναι *δείκτης* (pointer):

```
/* array.c */
/* Εκτύπωση των τιμών ενός πίνακα */
main()
{
  static int nums[]={10,13,8,7,15};
  int i;
  for(i=0;i<5;i++) printf("%d ",nums[i]);
}

/* ************************************************ */

/* parray.c */
/* Εκτύπωση των τιμών ενός πίνακα, με χρήση δεικτών (pointers) */
main()
{
  static int nums[]={10,13,8,7,15};
  int i;
  for(i=0;i<5;i++) printf("%d ",*(nums+i));
}
```

10.6 ΔΕΙΚΤΕΣ ΚΑΙ ΑΛΦΑΡΙΘΜΗΤΙΚΑ

Αφού τα αλφαριθμητικά είναι πίνακες και οι πίνακες, όπως είδαμε, συνδέονται στενά με τους δείκτες, είναι φυσικό και τα αλφαριθμητικά να έχουν μία στενή σχέση με τους δείκτες. Στην ουσία το όνομα ενός αλφαριθμητικού είναι δείκτης σε χαρακτήρα.

Έχουμε δει στην § 8.2 πώς παίρνουν αρχικές τιμές τα αλφαριθμητικά ως πίνακες χαρακτήρων, π.χ. η εντολή:

```
static char message[]="string";
```

ορίζει ένα αλφαριθμητικό με όνομα `message` ως πίνακα και του δίνει αρχική τιμή.

Όμως εφόσον το όνομα ενός αλφαριθμητικού είναι δείκτης σε χαρακτήρα, εξίσου καλά μπορούμε να ορίσουμε ένα αλφαριθμητικό και να του δώσουμε αρχική τιμή ως εξής:

```
char *message="string";
```

Οποιαδήποτε από τις παραπάνω δύο μορφές κι αν χρησιμοποιήσουμε σ'ένα πρόγραμμα, μία εντολή:

```
puts(message);
```

θα έχει ως αποτέλεσμα να τυπωθεί στην οθόνη η λέξη `string`. Όμως οι δύο μορφές έχουν μία *λεπτή διαφορά*. Η μορφή του πίνακα φυλάει στη μνήμη έναν πίνακα αρκετά μεγάλο (στην περίπτωση αυτή 7 bytes) για να τοποθετηθεί η λέξη και ο χαρακτήρας τερματισμού '`\0`'. Στη διεύθυνση του πρώτου χαρακτήρα, έχει δοθεί ως όνομα το όνομα του πίνακα, `message`.

Στη μορφή του δείκτη, φυλάσσεται με τον ίδιο τρόπο ένας πίνακας, μόνο που φυλάσσεται επίσης και μία *μεταβλητή δείκτη*, η μεταβλητή αυτή έχει το όνομα `message` και δείχνει στον πίνακα. Το Σχήμα 10.3 δείχνει τις δύο περιπτώσεις.

Όταν θέτουμε αρχικές τιμές με τη μορφή πίνακα η `message` είναι *σταθερά δείκτη*, δηλαδή μία διεύθυνση που δε μπορεί να αλλαχθεί. Στη μορφή του δείκτη όμως, η `message` είναι μία *μεταβλητή δείκτη*. Για παράδειγμα η εντολή:

```
puts(++message);
```

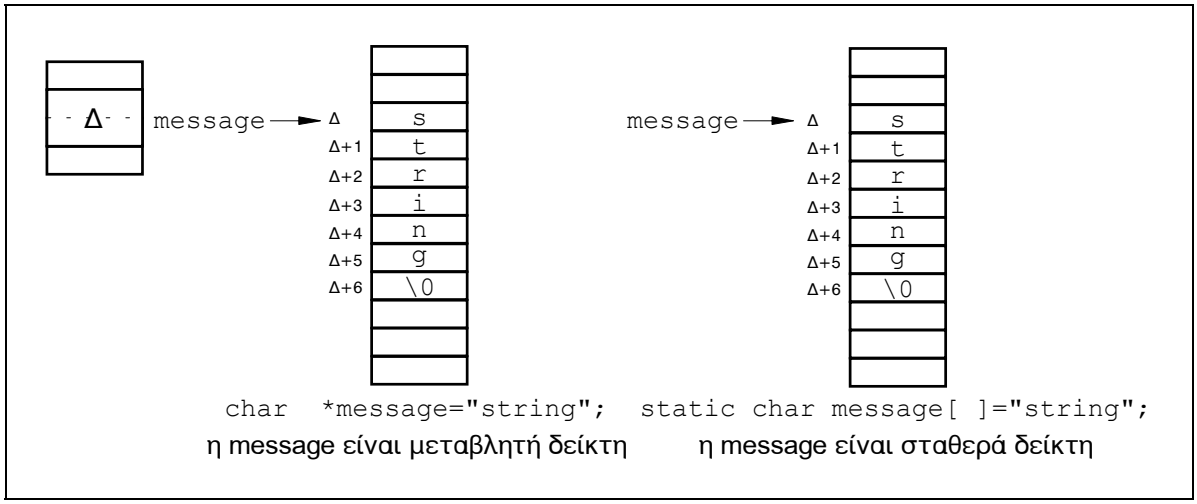
θα τυπώσει το αλφαριθμητικό που αρχίζει από το δεύτερο χαρακτήρα του αλφαριθμητικού: `tring`.

Αυτή η επιπλέον ευελιξία που έχει η προσέγγιση με δείκτη, μπορεί συχνά να αποτελέσει πλεονέκτημα, όπως θα δούμε.

Στην § 8.2.1 είδαμε μερικές συναρτήσεις βιβλιοθήκης για αλφαριθμητικά. Τώρα που είδαμε τη σχέση των δεικτών και των αλφαριθμητικών, θα μπορούσαμε να γράψουμε αυτές τις συναρτήσεις μόνοι μας.

Έτσι, για παράδειγμα, η συνάρτηση `strlen()` που υπολογίζει το μήκος ενός αλφαριθμητικού θα μπορούσε να γραφεί ως εξής:

```
int strlen(char *s)
{
  int i=0;
  while(*s++) i++; /* όσο δε βρίσκουμε το χαρακτήρα
                      τερματισμού... */
  return(i);
}
```



Σχήμα 10.3: *Αλφαριθμητικό ως δείκτης και ως πίνακας*

Η `strcpy()` που αντιγράφει ένα αλφαριθμητικό σε ένα άλλο, θα μπορούσε να γραφεί:

```
void strcpy(char *s1,char *s2)
{
  while(*s1++=*s2++);
}
```

Ενώ η `strcat()` θα μπορούσε να ήταν:

```
void strcat(char *s1,char *s2)
{
  while(*s1) s1++;   /* βρίσκουμε το τέλος του s1 */
  while(*s1++=*s2++);
}
```

Και η `strcmp()`:

```
int strcmp(char *s1,char *s2)
{
  for( ;*s1==*s2;s1++,s2++)
    if(*s1=='\0') return(0);
  return(*s1-*s2);
}
```

Μία ακόμη συνάρτηση βιβλιοθήκης της C που δουλεύει με αλφαριθμητικά είναι η `strchr()` η οποία ορίζεται στο αρχείο-επικεφαλίδας `string.h`. Η κλήση της συνάρτησης αυτής έχει τη μορφή `strchr(s,c)` και επιστρέφει ένα δείκτη στην πρώτη εμφάνιση του χαρακτήρα `c` μέσα στο αλφαριθμητικό `s`. Αν ο χαρακτήρας δε βρεθεί, τότε ο δείκτης παίρνει ως τιμή τη σταθερά `NULL`, η οποία ορίζεται στο αρχείο-επικεφαλίδας `stdio.h`.

Το παρακάτω πρόγραμμα χρησιμοποιεί τη συνάρτηση `strchr()`, ζητά από το χρήστη να πληκτρολογήσει μία πρόταση και το χαρακτήρα για τον οποίο θα ψάξει. Στη συνέχεια τυπώνει τη διεύθυνση της αρχής της πρότασης, τη διεύθυνση του χαρακτήρα και τη θέση του χαρακτήρα σε σχέση με την αρχή της πρότασης. Αυτή σχετική θέση είναι απλά η διαφορά ανάμεσα στις δύο διευθύνσεις:

```
/* search.c */
/* Αναζήτηση χαρακτήρα σε αλφαριθμητικό */
#include <string.h> /* για χρήση της strchr() */
#include <stdio.h>  /* για χρήση της NULL     */
main()
{
  char ch, line[81], *ptr;

  puts("\nΔώσε μια πρόταση"); gets(line);
  puts("Δώσε το χαρακτήρα που θέλεις να ψάξω");
  ch=getche();
  ptr=strchr(line,ch);
  if(ptr!=NULL)
  {
    printf("\nΗ πρόταση αρχίζει στη διεύθυνση %p",line);
    printf("\nΠρώτη εμφάνιση του χαρακτήρα στη διεύθυνση %p",ptr);
    printf("\nΑπόσταση του χαρακτήρα από την αρχή %d",ptr-line);
  }
  else puts("\nΟ χαρακτήρας δεν υπάρχει στην πρόταση");
}
```

10.7 ΠΙΝΑΚΕΣ ΔΕΙΚΤΩΝ ΚΑΙ ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Αφού οι δείκτες είναι μεταβλητές, μπορούν κι αυτοί να αποθηκεύονται σε πίνακες όπως οι άλλες μεταβλητές. Η δήλωση π.χ. ενός πίνακα 10 θέσεων που περιέχει ως στοιχεία δείκτες σε ακεραίους, γίνεται ως εξής:

```
int *ar[10];
```

Για να καταχωρήσουμε τη *διεύθυνση* μίας ακέραιας μεταβλητής με όνομα `var` στο τρίτο στοιχείο του πίνακα, θα γράψουμε:

```
ar[2]=&var;
```

και για να αναφερθούμε στην *τιμή* της `var` θα γράψουμε:

```
*ar[2]
```

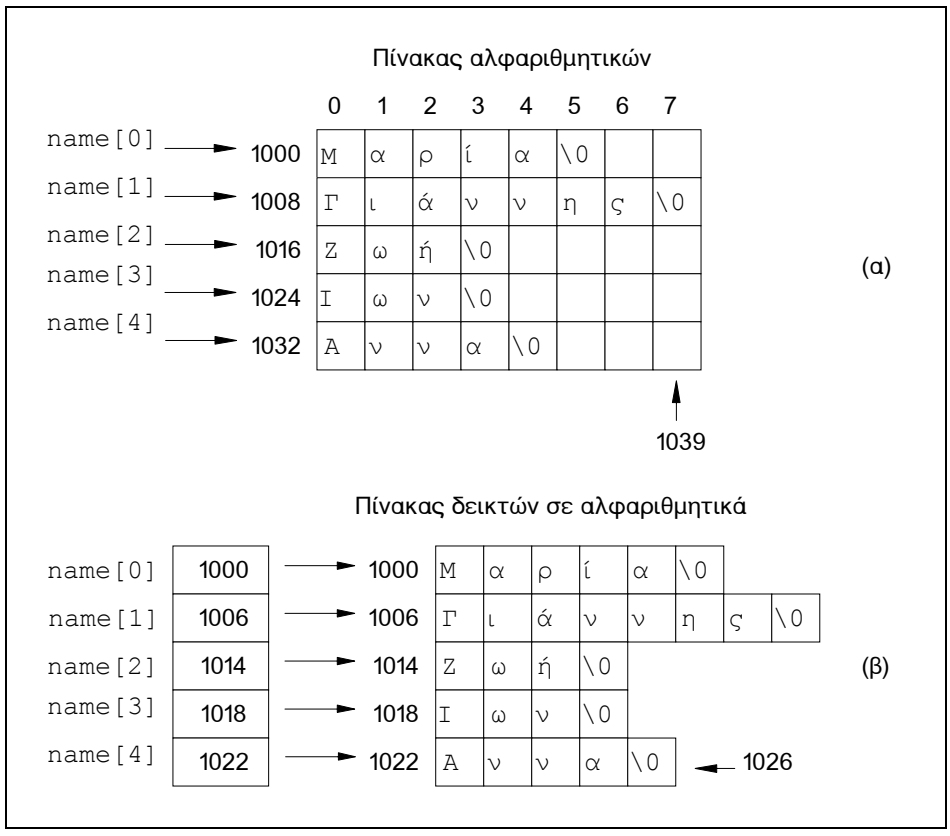
Οι πίνακες δεικτών χρησιμοποιούνται συνήθως για να κρατούν διευθύνσεις σε αλφαριθμητικά. Για να γίνει κατανοητό αυτό, ας θεωρήσουμε την παρακάτω εντολή απόδοσης αρχικών τιμών σ'έναν πίνακα αλφαριθμητικών:

```
static char name[5][8]={"Μαρία",
                        "Γιάννης",
                        "Ζωή",
                        "Ιων",
                        "Αννα"};
```

Σ'αυτή την περίπτωση τ'αλφαριθμητικά αποθηκεύονται στη μνήμη σ'έναν πίνακα με 5 γραμμές και 8 στήλες, όπως φαίνεται στο Σχήμα 10.4 α. Θα μπορούσαμε εξίσου καλά να θέσουμε αρχικές τιμές σ'έναν πίνακα δεικτών σε αλφαριθμητικά, ως εξής:

```
static char *name[]={"Μαρία",
                     "Γιάννης",
                     "Ζωή",
                     "Ιων",
                     "Αννα"};
```

Σ'αυτή την περίπτωση τ'αλφαριθμητικά αποθηκεύονται συνεχόμενα στη μνήμη, δε σχηματίζουν πίνακα και έτσι δεν υπάρχει χαμένος χώρος ανάμεσά τους. Έχει όμως δημιουργηθεί ένας πίνακας από *δείκτες* σ'αυτά τα αλφαριθμητικά, Σχήμα 10.4 β.



Σχήμα 10.4: *Πίνακας αλφαριθμητικών και πίνακας δεικτών σε αλφαριθμητικά*

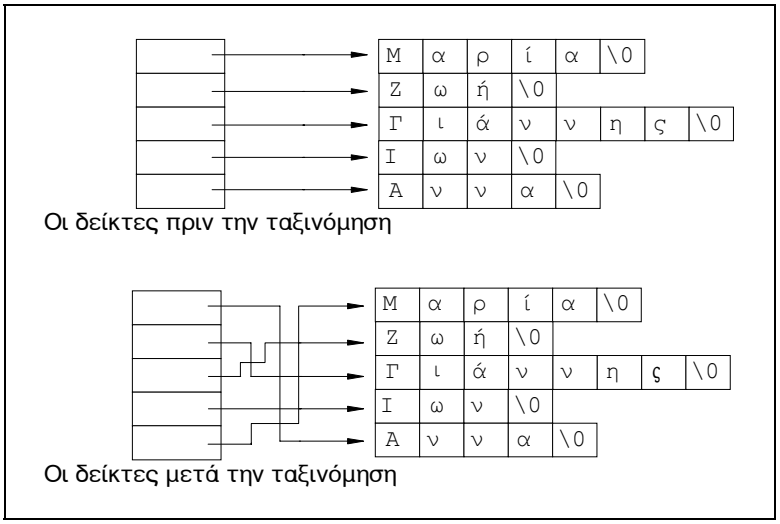
Όπως βλέπουμε από το Σχήμα 10.4, η περίπτωση απόδοσης αρχικών τιμών σε πίνακα δεικτών σε αλφαριθμητικά καταλαμβάνει λιγότερο χώρο στη μνήμη. Έτσι

ένας από τους λόγους που θέτουμε αρχικές τιμές στα αλφαριθμητικά σαν δείκτες είναι η αποτελεσματικότερη χρήση της μνήμης. Ένας άλλος λόγος είναι για να αποκτήσουμε μεγαλύτερη ευελιξία στο χειρισμό των αλφαριθμητικών στον πίνακα, όπως θα φανεί από το επόμενο παράδειγμα.

Στην § 8.1.6 παρουσιάσαμε ένα πρόγραμμα ταξινόμησης (`sortsel.c`) που μπορούσε να ταξινομεί έναν πίνακα ακεραίων. Θα χρησιμοποιήσουμε τώρα τον ίδιο αλγόριθμο για να ταξινομήσουμε έναν πίνακα αλφαριθμητικών. Όμως δεν πρόκειται να μετακινήσουμε τα ίδια τα αλφαριθμητικά μέσα στον πίνακα, αλλά θα ταξινομήσουμε έναν πίνακα από δείκτες σε αλφαριθμητικά, βλέπε Σχήμα 10.5.

```
/* sortstr.c */
/* Ταξινόμηση αλφαριθμητικών */
#define MAXNUM 30
#define MAXLEN 81
main()
{
  char name[MAXNUM][MAXLEN];  /* πίνακας αλφαριθμητικών */
  char *ptr[MAXNUM];          /* πίνακας δεικτών σε αλφαριθμητικά */
  int count=0;

  while(count<MAXNUM)
  {
    printf("%do αλφαριθμητικό: ",count+1);
    gets(name[count]);
    if(strlen(name[count])==0) break; /* η εισαγωγή σταματά δίνοντας*/
    ptr[count++]=name[count];         /* κενό αλφαριθμητικό         */
  }
  sort(ptr,count);
  display(ptr,count);
}


/* sort() */
/* Ταξινομεί τον πίνακα δεικτών με τη μέθοδο της επιλογής */
sort(char *p[],int c)
{
  int i,j,min;
  char *temp;

  for(i=0;i<c-1;i++)
  {
    min=i;
    for(j=i+1;j<c;j++)
      if(strcmp(p[min],p[j])>0) min=j;
    if(i!=min)
    {
      temp=p[i];
      p[i]=p[min];
      p[min]=temp;
     }
  }
}
```

```
/* display() */
/* Εμφανίζει τα αλφαριθμητικά ταξινομημένα */
display(char *p[],int c)
{
  int i;
  for(i=0;i<c;i++)printf("\n%do αλφαριθμητικό: %s",i+1,p[i]);
}
```



Σχήμα 10.5: *Ταξινόμηση πίνακα δεικτών σε αλφαριθμητικά*

Στο παραπάνω πρόγραμμα χρησιμοποιήσαμε την εντολή:

`ptr[count++]=name[count];`

στην οποία, αναφερόμενοι στον πίνακα `name` χρησιμοποιούμε μόνο ένα δείκτη-πίνακα (subscript) παρόλο που ο πίνακας αυτός είναι διδιάστατος. Τι σημαίνει λοιπόν το `name[count]`, αν ο πίνακας `name` είναι διδιάστατος;

Ας θυμηθούμε ότι μπορούμε να αντιληφθούμε ένα διδιάστατο πίνακα σαν ένα μονοδιάστατο πίνακα από μονοδιάστατους πίνακες. Σ'αυτή την περίπτωση η δήλωση

`char name[MAXNUM][MAXLEN];`

μπορεί να ερμηνευτεί ως δήλωση μονοδιάστατου πίνακα με `MAXNUM` στοιχεία, το καθένα εκ των οποίων είναι ένας μονοδιάστατος πίνακας χαρακτήρων, μήκους `MAXLEN`. Έτσι, αναφερόμενοι στα στοιχεία του πίνακα `name` με ένα μόνο δείκτη-πίνακα (subscript) `n`, στην ουσία αναφερόμαστε στη διεύθυνση του `n+1` αλφαριθμητικού, δηλαδή στη διεύθυνση του πρώτου χαρακτήρα του `n+1` αλφαριθμητικού.

Η ικανότητα της γλώσσας C να χειρίζεται τμήματα ενός πίνακα ως πίνακες, είναι στην πραγματικότητα μία ειδική μορφή ενός άλλου θέματος της C, που λέγεται πολλαπλή εμμεσότητα ή διπλή ανακατεύθυνση ή "δείκτες σε δείκτες". Για να γίνει κατανοητό αυτό ας δούμε πώς μπορούμε να αναφερθούμε με έκφραση δεικτών (pointer) σε κάποιο στοιχείο ενός διδιάστατου πίνακα. Έστω ο πίνακας:

```
int table[L][C]; /* πίνακας με L γραμμές και C στήλες */
```

τότε για να αναφερθούμε στο στοιχείο που βρίσκεται στην `i` γραμμή και `j` στήλη αυτού του πίνακα θα γράψουμε:

```
table[i][j]                    (1)
```

Όμως ο πίνακας `table` μπορεί να ερμηνευτεί και ως εξής: Ένας μονοδιάστατος πίνακας με `L` στοιχεία, το καθένα απ'τα οποία είναι ένας μονοδιάστατος πίνακας με `C` ακέραιους:

```
τύπος table[L];
```

όπου `τύπος: int list[C]`

Για να αναφερθούμε τώρα με δείκτη (pointer) στο στοιχείο `i` του πίνακα `table`, γράφουμε, όπως έχουμε δει στην § 10.5:

```
*(table+i)
```

όμως το στοιχείο `i` του πίνακα `table` είναι ένας ολόκληρος πίνακας με *όνομα* το όνομα της διεύθυνσης του πρώτου στοιχείου του, δηλαδή: `όνομα==*(table+i)`. Για να αναφερθούμε τώρα στο `j` στοιχείο αυτού του πίνακα γράφουμε: `*(όνομα+j)`, δηλαδή:

```
*(*(table+i)+j)                (2)
```

όμως αυτό είναι το στοιχείο `table[i][j]`. Έτσι οι παραστάσεις (1) και (2) είναι ισοδύναμες:

```
table[i][j]==*(*(table+i)+j)
```

πράγμα που μας λέει ότι: ένα στοιχείο ενός διδιάστατου πίνακα μπορεί να αναφερθεί μ'ένα δείκτη σ'έναν άλλο δείκτη. Έτσι λοιπόν, βλέπουμε ότι υπάρχουν δύο τρόποι για να αναφερθούμε στα στοιχεία ενός διδιάστατου πίνακα:

(i) Με τη χρήση παράστασης πίνακα: `table[i][j]`

(ii) Με τη χρήση παράστασης δεικτών (pointer): `*(*(table+i)+j)`

10.8 ΟΡΙΣΜΑΤΑ ΓΡΑΜΜΗΣ ΔΙΑΤΑΓΩΝ

Η `main()` είναι μία συνάρτηση και ως τέτοια μπορεί να δέχεται παραμέτρους. Οι *πραγματικές παράμετροι* στη συνάρτηση `main()` δίνονται όταν εκτελούμε το πρόγραμμά μας, από το περιβάλλον του Λειτουργικού Συστήματος. Για να γίνει αυτό, δημιουργούμε πρώτα το εκτελέσιμο αρχείο του προγράμματός μας και στη συνέχεια, σε περιβάλλον Λειτουργικού Συστήματος πληκτρολογούμε το όνομα του προγράμματος και, χωρισμένες με κενά, τις παραμέτρους που θέλουμε να μεταβιβάσουμε στη `main()`. Αυτές οι παράμετροι λέγονται *ορίσματα γραμμής διαταγών*.

Η `main()` δέχεται δύο παραμέτρους, που συνήθως τους δίνουμε τα παρακάτω ονόματα:

`argc` (από το arguments counter - μετρητής ορισμάτων) που είναι τύπου ακεραίου και σ'αυτήν αποθηκεύεται ο αριθμός των ορισμάτων της γραμμής διαταγών. Η τιμή της θα είναι πάντα τουλάχιστον 1, γιατί το όνομα του προγράμματος είναι το πρώτο όρισμα.

`argv` (από το arguments vector - διάνυσμα ορισμάτων) που είναι ένας πίνακας δεικτών στα ορίσματα της γραμμής διαταγών, δηλαδή ένας πίνακας δεικτών σε αλφαριθμητικά, όπου κάθε αλφαριθμητικό είναι ένα όρισμα.

Για να γίνουν κατανοητά τα παραπάνω, ας θεωρήσουμε το παρακάτω πρόγραμμα:

```
/* name.c */
/* Ορίσματα γραμμής διαταγών */
main(int argc,char *argv[])
{
  if(argc!=2) puts("Ξέχασες να δώσεις το όνομά σου");
  else printf("Γεια σου %s",argv[1]);
}
```

Για να τρέξουμε αυτό το πρόγραμμα, το μεταγλωττίζουμε, το διασυνδέουμε και από το περιβάλλον του Λειτουργικού Συστήματος το τρέχουμε, δίνοντας σαν όρισμα το όνομά μας, π.χ.

```
name Νίκο
```

Το αποτέλεσμα που θα δούμε είναι:

```
Γεια σου Νίκο
```

Παρακάτω δίνουμε ένα ενδιαφέρον και χρήσιμο πρόγραμμα, που χρησιμοποιεί ορίσματα γραμμής διαταγών για να εκτελεί διαδοχικά διάφορες διαταγές του DOS. Το πρόγραμμα κάνει χρήση της συνάρτησης `system()` (που ορίζεται στο αρχείο-επικεφαλίδας `stdlib.h`) η οποία δέχεται ως παράμετρο ένα αλφαριθμητικό και το μεταβιβάζει, σαν διαταγή, στον επεξεργαστή διαταγών του DOS (δηλαδή στο COMMAND.COM).

```
/* comline.c */
/* Διαδοχική εκτέλεση διαταγών του DOS που δίνονται στη
   γραμμή διαταγών */
#include <stdlib.h>
main(int argc, char *argv[])
{
  int i;

  if(argc<2) puts("Ξέχασες να δώσεις ορίσματα");
  else for(i=1;i<argc;i++) system(argv[i]);
}
```

Αφού δημιουργήσουμε το εκτελέσιμο αρχείο, δίνουμε από το περιβάλλον του Λειτουργικού Συστήματος DOS, την παρακάτω γραμμή διαταγών, που εκτελεί τις διαταγές VER, CHKDSK και DIR του DOS με τη σειρά:

```
comline ver chkdsk dir
```

10.9 ΔΕΙΚΤΕΣ ΚΑΙ ΔΟΜΕΣ

Η C επιτρέπει δείκτες σε τύπους δομών, όπως επιτρέπει δείκτες σε οποιουδήποτε τύπου δεδομένα. Η γενική μορφή της εντολής δήλωσης ενός δείκτη σε δομή είναι:

struct *αναγνωριστικό_τύπου_δομής* ******* *αναγνωριστικό;*

όπου *αναγνωριστικό* είναι το αναγνωριστικό του δείκτη. Ο τύπος της δομής πρέπει να έχει οριστεί προηγουμένως. Για παράδειγμα, αν έχουμε τις παρακάτω δηλώσεις:

```
struct date
{
  int day;
  char mon_name[13];
  int year;
};
struct date d1;
```

τότε η εντολή:

```
struct date *ptr;
```

δηλώνει ένα δείκτη με όνομα `ptr` σε δομή τύπου `struct date`. Η εντολή:

```
ptr=&d1;
```

καταχωρεί στο δείκτη `ptr` τη διεύθυνση της μεταβλητής δομής `d1`. Για να αναφερθούμε στο μέλος `year` της μεταβλητής `d1` μπορούμε να γράψουμε:

```
d1.year
```

όπως ξέρουμε, ή

```
(*ptr).year
```

σε έκφραση δείκτη. Όπου οι παρενθέσεις γύρω από το δείκτη είναι απαραίτητες γιατί ο τελεστής τελεία έχει μεγαλύτερη προτεραιότητα από τον τελεστή `*` (βλέπε Πίνακα 4.5). Επίσης μπορούμε αντί της τελευταίας παράστασης, να αναφερθούμε στο μέλος `year` ως εξής:

```
ptr->year
```

όπου χρησιμοποιούμε τον τελεστή `->`, που λέγεται και *τελεστής-βέλος*.

10.10 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί συνάρτηση που να δέχεται ως παραμέτρους τις διευθύνσεις τριών μεταβλητών ακεραίου τύπου και να τοποθετεί στην πρώτη μεταβλητή τη μέγιστη τιμή των τριών, στην δεύτερη την αμέσως μικρότερη τιμή και στην τρίτη να τοποθετεί την ελάχιστη.

```
/* sort() */
/* Αμοιβαία αλλαγή των τιμών τριών μεταβλητών */
sort(int *a,int *b,int *c)
{
  int max,mid,min;

  if(*a>*b)
    if(*b>*c)
      { max=*a; mid=*b; min=*c; }
    else
      if(*a>*c)
        { max=*a; mid=*c; min=*b; }
      else
        { max=*c; mid=*a; min=*b; }
  else
```

```
    if(*b>*c)
      if(*a>*c)
        { max=*b; mid=*a; min=*c; }
      else
        { max=*b; mid=*c; min=*a; }
    else
      { max=*c; mid=*b; min=*a; }
  *a=max; *b=mid; *c=min;
}
```

Το πρόγραμμα που θα καλούσε αυτή τη συνάρτηση, θα ήταν κάπως έτσι:

```
main()
{
  int x,y,z;

  scanf("%d %d %d",&x,&y,&z);
  sort(&x,&y,&z);
  printf("%d %d %d",x,y,z);
}
```

2. Στην § 9.2.2 είχαμε αναφέρει τη συνάρτηση είναι η `atoi()` (ASCII to integer) η οποία μετατρέπει το αλφαριθμητικό που δέχεται ως παράμετρο, σε ακέραιο αριθμό. Γράψτε μία δική σας εκδοχή αυτής της συνάρτησης, με όνομα `my_atoi()`.

```
/* my_atoi */
/* Μετατροπή string σε ακέραιο */
long int my_atoi(char *s)
{
  long int result=0;
  int sign;
  char *p;

  p=s;
  while(*p==' ') p++;  /* αγνοούμε τα πιθανά κενά στην αρχή */
  sign=(*p=='-')?-1:1; /* βρίσκουμε το πρόσημο              */
  if(*p=='+' || *p=='-') p++;
  while(*p>='0' && *p<='9')    /* όσο υπάρχουν ψηφία...     */
  {
    result=10*result+(*p-'0'); /* σχηματίζεται ο ακέραιος   */
    p++;
  }
  return(result*sign);
}
```

3. Να γραφεί μία συνάρτηση η οποία να δέχεται ως παράμετρο έναν ακέραιο αριθμό και να επιστρέφει το όνομα του μήνα που αντιστοιχεί σ'αυτόν (1: Ιανουάριος, 2: Φεβρουάριος κ.λ.π.).

```
/* month_name() */
/* Επιστρέφει το όνομα του n-οστού μήνα */
char *month_name(int n)
{
  static char *name[]={"Απαράδεκτος μήνας",
                       "Ιανουάριος","Φεβρουάριος","Μάρτιος",
                       "Απρίλιος","Μάιος","Ιούνιος",
                       "Ιούλιος","Αύγουστος","Σεπτέμβριος",
                       "Οκτώβριος","Νοέμβριος","Δεκέμβριος"
                      };
  return(n<1 || n>12)?name[0]:name[n];
}
```

4. Στο αρχείο-επικεφαλίδας `dos.h` δηλώνονται οι παρακάτω δομές:

```
struct date
{
  int da_year;  /* έτος  */
  int da_mon;   /* μήνας */
  int da_day;   /* ημέρα */
}

struct time
{
  int ti_hour;  /* ώρες                        */
  int ti_min;   /* λεπτά                       */
  int ti_sec;   /* δευτερόλεπτα                */
  int ti_hund;  /* εκατοστά του δευτερολέπτου */
}
```

Επίσης ορίζονται οι συναρτήσεις `getdate()` και `gettime()` ως εξής:

```
void getdate(struct date *d)
void gettime(struct time *t)
```

οι οποίες δέχονται από το σύστημα την ημερομηνία και την ώρα και τις τοποθετούν στις δομές που δείχνονται από το `d` και το `t` αντίστοιχα.

Ακόμα, ορίζονται οι συναρτήσεις `setdate()` και `settime()` ως εξής:

```
void setdate(struct date *d)
void settime(struct time *t)
```

οι οποίες θέτουν την ημερομηνία και την ώρα του συστήματος, όπως καθορίζονται από τις δομές που δείχνονται από το `d` και το `t` αντίστοιχα.

Γράψτε ένα πρόγραμμα που να εμφανίζει στην οθόνη την ημερομηνία και την ώρα του συστήματος. Στη συνέχεια να επιτρέπει στο χρήστη (αν επιθυμεί) να αλλάξει την ημερομηνία και την ώρα.

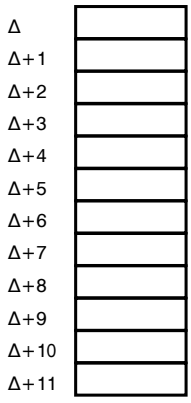
```
/* timedate.c */
/* Εμφάνιση και αλλαγή ημερομηνίας και ώρας του συστήματος */
#include <dos.h>
main()
{
  int h,m,s,hu;
  struct date *dp,d;
  struct time *tp,t;
  char ch;
```

```
  dp=&d;
  tp=&t;
  getdate(dp);
  printf("Ημερομηνία: %d-%d-%d\n", d.da_day,d.da_mon,d.da_year);
  printf("Αλλαγή ημερομηνίας (Y/N) ");
  ch=getche();
  if(ch=='Y' || ch=='y')
  {
    printf("\nΔώσε ημερομηνία (dd-mm-yy) ");
    scanf("%d-%d-%d",&d.da_day, &d.da_mon, &d.da_year);
    setdate(dp);
  }
  gettime(tp);
  printf("\nΩρα : %d:%d:%d:%d\n",
          t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
  printf("Αλλαγή ώρας (Y/N) ");
  ch=getche();
  if(ch=='Y' || ch=='y')
  {
    printf("\nΔώσε ώρα (hh:mm:ss:hu) ");
    scanf("%d:%d:%d:%d",&h,&m,&s,&hu);
    t.ti_hour=h; t.ti_min=m; t.ti_sec=s; t.ti_hund=hu;
    settime(tp);
  }
}
```

10.11 ΑΣΚΗΣΕΙΣ

1. Δείξτε τα περιεχόμενα της μνήμης μετά από κάθε εντολή. Οι εντολές εκτελούνται με τη σειρά που φαίνονται παρακάτω και η καθεμία λειτουργεί με τα αποτελέσματα της προηγούμενης.

```
int i,j,*pi;
char s[6];

i=0;
pi=&i;
j=*pi;
strcpy(s,"ΑΚ");
pi=&pi;
pi++;
j=3;
*pi=0;
pi=&j;
i=*pi;
puts(s);
```



2. Δώστε το αποτέλεσμα της εκτέλεσης του παρακάτω προγράμματος:

```
main()
{
  static int arr[]={4,5,6};
  int j;

  for(j=0;j<3;j++)
    printf("%d",*arr+j);
}
```

3. Δώστε το αποτέλεσμα του παρακάτω προγράμματος:

```
main()
{
  static int arr[]={4,5,6};
  int j;

  for(j=0;j<3;j++)
    printf("%d",arr+j);
}
```

4. Ποιο είναι το αποτέλεσμα του παρακάτω προγράμματος:

```
main()
{
  static int arr[]={4,5,6};
  int j,*ptr;

  ptr=arr;
  for(j=0;j<3;j++)
    printf("%d",*ptr++);
}
```

5. Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
main()
{
  char str[10];

  strcpy(str,"ABC");
  printf("%c\n",str[2]);
  printf("%d\n",str[3]);
  printf("%s\n",str);
  printf("%c\n",*(str+1));
  printf("%s\n",str+1);
  printf("%c\n",(str+1)[1]);
}
```

ΚΕΦΑΛΑΙΟ 11
ΑΡΧΕΙΑ

Όταν ένας αριθμός δεδομένων απαιτείται να διατηρηθεί για μελλοντική χρήση από διάφορα προγράμματα ή είναι μεγάλου μεγέθους και απαιτεί μεγαλύτερη ποσότητα από τη διαθέσιμη κύρια μνήμη του Η/Υ, τότε για την αποθήκευσή τους χρησιμοποιείται η περιφερειακή μνήμη του Η/Υ, π.χ. δίσκος. Κάθε οργανωμένη συλλογή δεδομένων, που αποθηκεύεται σε κάποιο περιφερειακό μέσο αποθήκευσης, ονομάζεται *αρχείο* (file).

Η C θεωρεί όλα τα αρχεία σαν σειρές από bytes χωρίς καμία δομή. Ένα πρόγραμμα όμως, μπορεί να θεωρεί ότι ένα αρχείο έχει κάποια *λογική δομή*, π.χ. ότι αποτελείται από *αλφαριθμητικά* ή αποτελείται από *δομές*. Η παραπάνω θεώρηση όμως, γίνεται μόνο σε επίπεδο λογικής του συγκεκριμένου προγράμματος, οι συναρτήσεις εισόδου και εξόδου της C χειρίζονται μόνο σειρές από bytes.

Αυτό είναι πολύ βολικό γιατί έτσι οι πληροφορίες του ίδιου αρχείου μπορούν να χρησιμοποιηθούν από διαφορετικά προγράμματα -τα οποία δε χρειάζεται να γνωρίζουν τη δομή του αρχείου- και για διαφορετικούς σκοπούς κάθε φορά.

Γενικά διακρίνουμε δύο είδη αρχείων, τα *αρχεία κειμένου* (text files) και τα *δυαδικά αρχεία* (binary files). Τα αρχεία κειμένου αποτελούνται μόνον από χαρακτήρες ASCII και έτσι είναι άμεσα αναγνώσιμα από τον άνθρωπο. Τα δεδομένα ενός αρχείου κειμένου χωρίζονται σε γραμμές με τη χρήση του χαρακτήρα νέας γραμμής `'\n'`.

Τα δυαδικά αρχεία περιέχουν αριθμητικές τιμές σε δυαδική μορφή και κατά συνέπεια δεν είναι άμεσα αναγνώσιμα από τον άνθρωπο. Οι χαρακτήρες που περιέχονται σ'ένα δυαδικό αρχείο αναγνωρίζονται, όπως και στο αρχείο κειμένου, οι αριθμοί όμως αποθηκεύονται όπως είναι στην κύρια μνήμη: σε δυαδική μορφή. Έτσι τα δυαδικά αρχεία είναι συνήθως μικρότερα σε μέγεθος από τα αρχεία κειμένου, αφού για έναν ακέραιο π.χ. τον 12345 χρειάζονται μόνο 2 bytes, αντί 5 bytes που θα χρειάζονταν σ'ένα αρχείο κειμένου (1 χαρακτήρας = 1 byte).

11.1 ΑΝΟΙΓΜΑ ΚΑΙ ΚΛΕΙΣΙΜΟ ΑΡΧΕΙΟΥ

Η πρώτη εργασία που πρέπει να γίνει κατά το χειρισμό των αρχείων, είναι το *άνοιγμα* (opening). Το άνοιγμα έχει την έννοια της προετοιμασίας του αρχείου για είσοδο και έξοδο στοιχείων. Κατά το άνοιγμα, το αρχείο που είναι γνωστό με ένα όνομα στο Λειτουργικό Σύστημα, συνδέεται με μία μεταβλητή του προγράμματος (που στη συνέχεια θα ονομάζουμε *δείκτη-αρχείου*). Στη συνέχεια αυτή η μεταβλητή χρησιμοποιείται ως όρισμα στις συναρτήσεις χειρισμού του αρχείου. Η μεταβλητή αυτή πρέπει να είναι τύπου *δείκτη σε* `FILE`. Ο τύπος `FILE` ορίζεται στο αρχείο-επικεφαλίδας `stdio.h`.

Το άνοιγμα ενός αρχείου γίνεται με τη συνάρτηση `fopen()` της οποίας το πρότυπο υπάρχει στο αρχείο-επικεφαλίδας `stdio.h`:

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

όπου το `path` προσδιορίζει *το πλήρες όνομα του αρχείου* που θέλουμε να ανοίξουμε, όπως αυτό είναι γνωστό στο Λειτουργικό Σύστημα και το `mode` καθορίζει τον *τρόπο* με τον οποίο θα ανοιχτεί το αρχείο. Η συνάρτηση `fopen()` επιστρέφει μία τιμή τύπου δείκτη σε `FILE`, η οποία πρέπει να αποθηκευτεί σε ένα δείκτη-αρχείου. Αν για οποιοδήποτε λόγο το αρχείο δε μπορεί να ανοιχτεί, η `fopen()` επιστρέφει τιμή μηδέν (καθορισμένη στο `stdio.h` ως `NULL`).

Οι παρακάτω εντολές δηλώνουν ένα δείκτη-αρχείου, με όνομα `fptr`, και ανοίγουν ένα αρχείο με όνομα `TEXTFILE.TXT` για εγγραφή (δηλώνεται με το `"w"`):

```
FILE *fptr;
fptr=fopen("textfile.txt","w");
```

Ο τρόπος με τον οποίο θα ανοιχτεί ένα αρχείο μπορεί να είναι για ανάγνωση (read), για εγγραφή (write), για προσθήκη (append) κ.λ.π. Αυτό καθορίζεται από την τιμή του δεύτερου ορίσματος της `fopen()`, οι δυνατές τιμές αυτού του ορίσματος φαίνονται στον Πίνακα 11.1.

Τιμή	***Σημασία***
`"r"`	Άνοιγμα αρχείου κειμένου για ανάγνωση. Το αρχείο πρέπει να προϋπάρχει.
`"w"`	Άνοιγμα αρχείου κειμένου για εγγραφή. Αν το αρχείο υπάρχει τα περιεχόμενά του καταστρέφονται, αν δεν υπάρχει δημιουργείται.
`"a"`	Άνοιγμα αρχείου κειμένου για προσθήκη στο τέλος του. Αν το αρχείο δεν υπάρχει, δημιουργείται.
`"rb"`	Άνοιγμα δυαδικού αρχείου για ανάγνωση. Το αρχείο πρέπει να προϋπάρχει.
`"wb"`	Άνοιγμα δυαδικού αρχείου για εγγραφή. Αν το αρχείο υπάρχει τα περιεχόμενά του καταστρέφονται, αν δεν υπάρχει δημιουργείται.
`"ab"`	Άνοιγμα δυαδικού αρχείου για προσθήκη στο τέλος του. Αν το αρχείο δεν υπάρχει, δημιουργείται.
`"r+"`	Άνοιγμα αρχείου κειμένου για ανάγνωση και εγγραφή. Το αρχείο πρέπει να προϋπάρχει.
`"w+"`	Άνοιγμα αρχείου κειμένου για εγγραφή και ανάγνωση. Αν το αρχείο δεν υπάρχει, δημιουργείται.
`"a+"`	Άνοιγμα αρχείου κειμένου για προσθήκη στο τέλος του και ανάγνωση. Αν το αρχείο δεν υπάρχει, δημιουργείται.
`"r+b"`	Άνοιγμα δυαδικού αρχείου για ανάγνωση και εγγραφή. Το αρχείο πρέπει να προϋπάρχει.
`"w+b"`	Άνοιγμα δυαδικού αρχείου για εγγραφή και ανάγνωση. Αν το αρχείο δεν υπάρχει, δημιουργείται.
`"a+b"`	Άνοιγμα δυαδικού αρχείου για προσθήκη στο τέλος του και ανάγνωση. Αν το αρχείο δεν υπάρχει, δημιουργείται.

Πίνακας 11.1: *Αποδεκτές τιμές του τρόπου ανοίγματος ενός αρχείου*

Όταν τελειώσει η μεταφορά των δεδομένων από και προς το αρχείο, τότε το αρχείο πρέπει να *κλείσει* (close). Το κλείσιμο έχει την έννοια του να οριστικοποιήσει (σώσει) τις αλλαγές που έγιναν στο αρχείο και να αποδεσμεύσει το αρχείο από το πρόγραμμα. Η συνάρτηση η οποία κλείνει ένα αρχείο είναι η `fclose()`, το πρότυπο της οποίας υπάρχει στο αρχείο-επικεφαλίδας `stdio.h` και είναι:

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

όπου `fptr` είναι ο δείκτης-αρχείου που επιστρέφεται από την κλήση της `fopen()`. Το μηδέν ως επιστρεφόμενη τιμή σημαίνει επιτυχές κλείσιμο, οποιαδήποτε άλλη τιμή δηλώνει την ύπαρξη κάποιου λάθους (π.χ. δεν υπάρχει αρκετός χώρος στο δίσκο).

11.2 ΠΡΟΣΠΕΛΑΣΗ ΑΡΧΕΙΟΥ

11.2.1 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΧΑΡΑΚΤΗΡΩΝ

Η συνάρτηση `putc()` χρησιμοποιείται για το γράψιμο ενός χαρακτήρα σ'ένα αρχείο το οποίο έχει προηγουμένως ανοιχτεί για εγγραφή μέσω της συνάρτησης `fopen()`. Το πρότυπο της `putc()`, το οποίο υπάρχει στο αρχείο-επικεφαλίδας `stdio.h`, είναι:

```
int putc(int ch, FILE *fptr);
```

όπου `fptr` είναι ο δείκτης-αρχείου που επιστρέφεται από την `fopen()`, και `ch` είναι ο προς έξοδο χαρακτήρας. Ο δείκτης-αρχείου καθορίζει σε ποιο αρχείο δίσκου θα γίνει η εγγραφή. Αν η `putc()` είναι επιτυχής θα επιστρέψει το χαρακτήρα που γράφτηκε. Αν αποτύχει θα επιστρέψει την τιμή `EOF` (καθορίζεται στο `stdio.h` με τιμή -1 και αντιπροσωπεύει το *τέλος αρχείου* - **E**nd **O**f **F**ile).

Το παρακάτω πρόγραμμα δημιουργεί ένα αρχείο με όνομα `CHARTEXT.TXT` και γράφει σ'αυτό χαρακτήρες που δίνονται από το πληκτρολόγιο, έως ότου δώσουμε `ENTER`:

```
/* writec.c */
/* Εγγραφή χαρακτήρων σε αρχείο */
#include <stdio.h>
main()
{
  FILE *fptr;
  int ch;

  if((fptr=fopen("chartext.txt","w"))==NULL)
  {
    puts("Δε μπορώ να ανοίξω το αρχείο CHARTEXT.TXT");
    exit(0);
  }
  while((ch=getche())!='\r') putc(ch,fptr);
  fclose(fptr);
}
```

Κάθε byte μέσα σε ένα αρχείο αντιστοιχεί σε μία θέση η οποία δείχνεται από ένα δείκτη που λέγεται *δείκτης-θέσης του αρχείου*. Ο δείκτης-θέσης δείχνει στο byte του αρχείου όπου θα γίνει η επόμενη προσπέλαση. Κάθε φορά που γράφουμε (ή διαβάζουμε) ένα αντικείμενο σε (ή από) ένα αρχείο, ο δείκτης-θέσης δείχνει μετά

το τέλος αυτού του αντικειμένου, έτσι ώστε η επόμενη εγγραφή ή ανάγνωση να συνεχιστεί από εκείνο το σημείο και μετά. Όταν ανοίγουμε ένα αρχείο για εγγραφή ή ανάγνωση, ο δείκτης-θέσης τίθεται στην αρχή του αρχείου. Όταν ανοίγουμε ένα αρχείο για προσθήκη (append) τότε ο δείκτης-θέσης τίθεται στο τέλος του αρχείου.

Για να διαβάσουμε χαρακτήρες από ένα αρχείο που προηγουμένως το έχουμε ανοίξει για διάβασμα μέσω της `fopen()`, χρησιμοποιούμε τη συνάρτηση `getc()`, της οποίας το πρότυπο, το οποίο υπάρχει στο αρχείο-επικεφαλίδας `stdio.h` , είναι:

```
int getc(FILE *fptr);
```

όπου `fptr` είναι ο δείκτης-αρχείου που επιστρέφεται από την `fopen()`. Η `getc()` επιστρέφει τον επόμενο χαρακτήρα από το αρχείο ή `EOF` όταν φτάσει στο τέλος του, ή συμβεί κάποιο λάθος κατά την ανάγνωση.

Το παρακάτω πρόγραμμα διαβάζει από ένα αρχείο, όλους τους χαρακτήρες και τους εμφανίζει στην οθόνη:

```
/* readc.c */
/* Ανάγνωση χαρακτήρων από αρχείο */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  int ch;

  if(argc!=2)
  {
    puts("Δώσε: readc όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"r"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  while((ch=getc(fptr))!=EOF) printf("%c",ch);
  fclose(fptr);
}
```

Στο παραπάνω πρόγραμμα το όνομα του αρχείου του οποίου τα περιεχόμενα θέλουμε να δούμε, δίνεται ως όρισμα στη γραμμή διαταγών (§ 10.8). Έτσι αν π.χ. θέλουμε να δούμε τα περιεχόμενα του αρχείου `CHARTEXT.TXT` που δημιουργήσαμε με το πρόγραμμα `writec.c`, πρέπει αφού μεταγλωττίσουμε και διασυνδέσουμε το `readc.c`, από το περιβάλλον του Λειτουργικού Συστήματος να δώσουμε:

```
readc chartext.txt
```

Έτσι είναι προφανές ότι το πρόγραμμα `readc.c` προσομοιάζει την εντολή TYPE του DOS.

Όταν ανοίγουμε ένα αρχείο για δυαδική είσοδο, είναι πιθανόν η `getc()` να διαβάσει μιαν ακέραια τιμή ίση με το `EOF` (δηλαδή -1). Αν συμβεί κάτι τέτοιο η

`getc()` θα σταματήσει το διάβασμα νομίζοντας ότι έχει φτάσει στο τέλος του αρχείου. Για να λύσει αυτό το πρόβλημα η C παρέχει τη συνάρτηση `feof()`, η οποία καθορίζει πού βρίσκεται το σημάδι τέλους αρχείου όταν διαβάζονται δυαδικά δεδομένα. Η συνάρτηση `feof()` δέχεται ως όρισμα ένα δείκτη-αρχείου και επιστρέφει 1 αν έχουμε φτάσει στο τέλος του αρχείου και 0 αν δεν έχουμε φτάσει. Έτσι η παρακάτω εντολή διαβάζει ένα δυαδικό αρχείο μέχρι το τέλος του:

```
while(!feof(fptr)) ch=getc(fptr);
```

Η `feof()` μπορεί επίσης να χρησιμοποιηθεί και σε αρχεία κειμένου.

11.2.2 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ

Η εγγραφή και ανάγνωση αλφαριθμητικών σε αρχεία επιτυγχάνεται με χρήση των συναρτήσεων `fputs()` και `fgets()` αντίστοιχα. Τα πρότυπά τους, που υπάρχουν στο αρχείο-επικεφαλίδας `stdio.h`, είναι:

```
int   fputs(char *str, FILE *fptr);

char *fgets(char *str, int n, FILE *fptr);
```

Η συνάρτηση `fputs()` δουλεύει ακριβώς όπως και η γνωστή συνάρτηση `puts()` (βλέπε § 5.2), με τη διαφορά ότι η `fputs()` γράφει το αλφαριθμητικό στο αρχείο που καθορίζεται από το `fptr`. Η συνάρτηση `fputs()`, όταν είναι επιτυχής επιστρέφει τον τελευταίο χαρακτήρα του αλφαριθμητικού που γράφτηκε, ενώ όταν αποτυγχάνει επιστρέφει `EOF`.

Η συνάρτηση `fgets()` διαβάζει το επόμενο αλφαριθμητικό από το καθοριζόμενο αρχείο μέχρι να διαβάσει είτε ένα χαρακτήρα νέας γραμμής είτε `n-1` χαρακτήρες και το τοποθετεί στον πίνακα χαρακτήρων που δείχνεται από το `str`. Αφού διαβαστούν οι χαρακτήρες, τοποθετείται στον πίνακα ένας χαρακτήρας τερματισμού `'\0'`. Όταν η `fgets()` είναι επιτυχής επιστρέφει την τιμή του αλφαριθμητικού που διαβάστηκε, ενώ όταν αποτυγχάνει επιστρέφει τιμή `NULL`.

Τα παρακάτω δύο προγράμματα `writes.c` και `reads.c` γράφουν και διαβάζουν αλφαριθμητικά σε αρχείο, αντίστοιχα:

```
/* writes.c */
/* Εγγραφή αλφαριθμητικών σε αρχείο */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  char string[81];

  if(argc!=2)
  {
    puts("Δώσε: writes όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"w"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
```

```
  }
  while(strlen(gets(string))>0)
  {
    fputs(string,fptr);
    fputs("\n",fptr);
  }
  fclose(fptr);
}


/* reads.c */
/* Ανάγνωση αλφαριθμητικών από αρχείο */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  char string[81];

  if(argc!=2)
  {
    puts("Δώσε: reads όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"r"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  while(fgets(string,81,fptr)!=NULL) printf("%s",string);
  fclose(fptr);
}
```

11.2.3 ΜΟΡΦΟΠΟΙΗΜΕΝΗ ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ

Οι αντίστοιχες συναρτήσεις των `printf()` και `scanf()` που χρησιμοποιούνται για μορφοποιημένη είσοδο και έξοδο από το πληκτρολόγιο και στην οθόνη αντίστοιχα (βλέπε § 5.3), είναι οι `fprintf()` και `fscanf()` για είσοδο και έξοδο από αρχείο. Τα πρότυπα αυτών των συναρτήσεων υπάρχουν στο αρχείο-επικεφαλίδας `stdio.h`. Οι `fprintf()` και `fscanf()` λειτουργούν ακριβώς όμοια όπως οι `printf()` και `scanf()`, μόνο που δέχονται ένα επιπλέον όρισμα που είναι ο δείκτης-αρχείου στο οποίο θα γράψουμε ή από το οποίο θα διαβάσουμε:

```
fprintf(fptr, αλφαριθμητικό_ελέγχου, λίστα_ορισμάτων);

fscanf(fptr, αλφαριθμητικό_ελέγχου, λίστα_ορισμάτων);
```

Η `fprintf()` επιστρέφει τον αριθμό των χαρακτήρων που γράφτηκαν, ή έναν αρνητικό αριθμό σε περίπτωση σφάλματος. Η `fscanf()` επιστρέφει τον αριθμό των αντικειμένων που διαβάστηκαν, ή `EOF` όταν συναντήσει το τέλος αρχείου ή συμβεί κάποιο σφάλμα.

Το παρακάτω πρόγραμμα δέχεται από το πληκτρολόγιο ονοματεπώνυμο και μισθό και τα καταχωρεί στο αρχείο το όνομα του οποίου δίνουμε στη γραμμή

διαταγών. Η εισαγωγή στοιχείων σταματάει αν δώσουμε ως όνομα ένα κενό αλφαριθμητικό (απλώς ENTER):

```
/* writef.c */
/* Εγγραφή μορφοποιημένων δεδομένων σε αρχείο */
#include <stdio.h>
#include <stdlib.h> /* για χρήση της συνάρτησης atof() */
main(int argc, char *argv[])
{
  FILE *fptr;
  char name[40],numstr[10];
  float salary;

  if(argc!=2)
  {
    puts("Δώσε: writef όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"w"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  do
  {
    printf("\nΟνομα : "); gets(name);
    printf("Μισθός: ");
    gets(numstr); salary=atof(numstr);
    if(strlen(name)>0) fprintf(fptr,"%s %f",name,salary);
  }
  while(strlen(name)>0);
  fclose(fptr);
}
```

Το επόμενο πρόγραμμα διαβάζει μορφοποιημένα δεδομένα από αρχείο, που δημιουργήθηκε από το παραπάνω πρόγραμμα και τα εμφανίζει στην οθόνη:

```
/* readf.c */
/* Ανάγνωση μορφοποιημένων δεδομένων από αρχείο */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  char name[40];
  float salary;

  if(argc!=2)
  {
    puts("Δώσε: readf όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"r"))==NULL)
  {
```

```
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  while(fscanf(fptr,"%s %f",&name,&salary)!=EOF)
    printf("%s %f\n",name,salary);
  fclose(fptr);
}
```

11.2.4 ΕΙΣΟΔΟΣ ΚΑΙ ΕΞΟΔΟΣ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ

Υπάρχουν δύο συναρτήσεις οι οποίες μας επιτρέπουν να γράφουμε και να διαβάζουμε (σε και από ένα αρχείο) κατευθείαν *ολόκληρες δομές δεδομένων* (π.χ. εγγραφές (δομές), πίνακες κ.λ.π.). Οι συναρτήσεις αυτές είναι οι `fwrite()` και `fread()`, των οποίων τα πρότυπα βρίσκονται στο `stdio.h`:

```
int fwrite(void *buffer, int size, int n, FILE *fptr);

int fread(void *buffer, int size, int n, FILE *fptr);
```

Στην περίπτωση της `fread()`, ο `buffer` είναι ένας *δείκτης* (pointer) σε μία περιοχή της μνήμης η οποία θα δεχτεί τα δεδομένα που διαβάζονται από το αρχείο. Στην `fwrite()`, ο `buffer` είναι ένας *δείκτης* (pointer) στις πληροφορίες που θα γραφτούν στο αρχείο. Και στις δύο συναρτήσεις το `size` καθορίζει *τον αριθμό των bytes* που θα διαβαστούν ή θα γραφτούν. Το όρισμα `n` καθορίζει *το πλήθος των στοιχείων* (το καθένα μεγέθους `size`) που θα διαβαστούν ή θα γραφτούν. Τέλος το `fptr` είναι ένας *δείκτης-αρχείου* που ήδη έχει ανοιχτεί. Οι συναρτήσεις επιστρέφουν τον αριθμό των στοιχείων που πραγματικά γράφτηκαν ή διαβάστηκαν. Έτσι π.χ. το παρακάτω πρόγραμμα δημιουργεί στο δίσκο ένα αρχείο στο οποίο καταχωρούνται από το πληκτρολόγιο ονοματεπώνυμο και μισθός κάποιων υπαλλήλων:

```
/* writer.c */
/* Γράψιμο εγγραφών σε αρχείο */
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
  struct record
  {
    char name[40];
    float salary;
  } empl;
  char numstr[10];
  FILE *fptr;
  char select;

  if(argc!=2)
  {
    puts("Δώσε: writer όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"wb"))==NULL)
```

```
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  do
  {
    printf("\nΔώσε όνομα: "); gets(empl.name);
    printf("Δώσε μισθό: ");   gets(numstr);
    empl.salary=atof(numstr);
    fwrite(&empl,sizeof(empl),1,fptr);
    printf("Θα δώσεις άλλον υπάλληλο (N/O)");
    select=getche();
  }
  while(select=='N'||select=='ν'||select=='N'||select=='n');
  fclose(fptr);
}
```

Το επόμενο πρόγραμμα διαβάζει τα δεδομένα από κάποιο αρχείο εγγραφών που δημιουργήθηκε με το προηγούμενο πρόγραμμα και τα εμφανίζει στην οθόνη:

```
/* readr.c */
/* Ανάγνωση εγγραφών από αρχείο */
#include <stdio.h>
main(int argc, char *argv[])
{
  struct record
  {
    char name[40];
    float salary;
  };
  struct record empl;
  char numstr[10];
  FILE *fptr;

  if(argc!=2)
  {
    puts("Δώσε: readr όνομα_αρχείου"); exit(0);
  }
  if((fptr=fopen(argv[1],"rb"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  while(fread(&empl,sizeof(empl),1,fptr)==1)
    printf("\nΟνομα: %s Μισθός: %f",empl.name,empl.salary);
  fclose(fptr);
}
```

Τέλος ας δούμε ένα παράδειγμα στο οποίο ένας ολόκληρος πίνακας γράφεται σ'ένα αρχείο:

```
/* warray.c */
/* Εγγραφή πίνακα σε αρχείο */
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
  static table[]={1,2,3,4,5,6,7,8,9,10};
  FILE *fptr;

  if(argc!=2)
  {
    puts("Δώσε: warray όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"wb"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s",argv[1]);
    exit(0);
  }
  fwrite(table,sizeof(table),1,fptr);
  fclose(fptr);
}
```

11.3 ΑΛΛΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΧΕΙΡΙΣΜΟΥ ΑΡΧΕΙΩΝ

Μερικές ακόμη συναρτήσεις που χρησιμοποιούνται με τα αρχεία, περιγράφονται παρακάτω. Τα πρότυπά τους βρίσκονται στο αρχείο-επικεφαλίδας `stdio.h`.

fseek()

Μέχρι τώρα όλες μας οι αναγνώσεις και εγγραφές σε αρχείο, γίνονταν σειριακά. Δηλαδή όταν γράφαμε σε ένα αρχείο, παίρναμε μία ομάδα από στοιχεία - χαρακτήρες, αλφαριθμητικά, δομές κ.λ.π.- και τα τοποθετούσαμε ανά ένα κάθε φορά σε διαδοχικές θέσεις εντός του αρχείου. Αντίστοιχα όταν διαβάζαμε, αρχίζαμε από την αρχή του αρχείου και συνεχίζαμε μέχρι να φτάσουμε στο τέλος του.

Είναι δυνατόν επίσης να προσπελάσουμε άμεσα οποιοδήποτε στοιχείο δεδομένων του αρχείου, οπουδήποτε μέσα στο αρχείο κι αν βρίσκεται αυτό. Αυτό γίνεται τοποθετώντας κατάλληλα το δείκτη-θέσης του αρχείου χρησιμοποιώντας τη συνάρτηση `fseek()`. Το πρότυπο αυτής είναι:

```
int fseek(FILE *fptr, long int offset, int whence);
```

όπου `fptr` είναι ένας δείκτης-αρχείου που επιστρέφεται με μία κλήση στην `fopen()`, `offset` είναι ο αριθμός των bytes που δηλώνει την απόσταση της νέας θέσης του δείκτη-θέσης από την `whence`. Η `whence` μπορεί να είναι 0 που υποδηλώνει την αρχή του αρχείου, 1 που υποδηλώνει την τρέχουσα θέση ή 2 που υποδηλώνει το τέλος του αρχείου, ή εναλλακτικά ένα από τα ονόματα `SEEK_SET`, `SEEK_CUR`, `SEEK_END` αντίστοιχα, καθόσον στο `stdio.h` υπάρχουν οι ακόλουθες οδηγίες:

```
#define SEEK_SET  0
#define SEEK_CUR  1
#define SEEK_END  2
```

Τιμή επιστροφής μηδέν σημαίνει ότι η `fseek()` πέτυχε. Μη-μηδενική τιμή σημαίνει αποτυχία.

Προσοχή: Η `fseek()` δεν πρέπει να χρησιμοποιείται σε αρχεία κειμένου.

ftell()

Επιστρέφει την τρέχουσα τιμή του δείκτη-θέσης αρχείου (σε bytes μετρημένα απ'την αρχή του αρχείου) για το καθορισμένο αρχείο. Όταν συμβεί κάποιο σφάλμα επιστρέφει -1L. Το πρότυπό της είναι:

```
long ftell(FILE *fptr);
```

rewind()

Τοποθετεί το δείκτη-θέσης του αρχείου που δέχεται ως όρισμα, στην αρχή του αρχείου. Το πρότυπό της είναι:

```
void rewind(FILE *fptr);
```

Είναι προφανές ότι η συνάρτηση `rewind()` είναι ισοδύναμη της:

```
fseek(FILE *fptr, 0L, SEEK_SET);
```

remove()

Διαγράφει από το δίσκο το αρχείο που δέχεται ως όρισμα (`name`). Το πρότυπό της είναι:

```
int remove(char *name);
```

Επιστρέφει 0 αν το αρχείο διαγράφτηκε με επιτυχία και -1 αν έχει συμβεί σφάλμα.

rename()

Μετονομάζει ένα αρχείο (παλαιό όνομα `old_name`, νέο όνομα `new_name`). Το πρότυπό της είναι:

```
int rename(char *old_name, char *new_name);
```

Επιστρέφει 0 όταν είναι επιτυχής και -1 αν έχει συμβεί σφάλμα.

11.4 ΠΑΡΑΔΕΙΓΜΑΤΑ

1. Να γραφεί πρόγραμμα που να μετρά τα γράμματα, τις λέξεις, τις προτάσεις και τη μέση τιμή των γραμμάτων ανά λέξη, για ένα αρχείο κειμένου το όνομα του οποίου θα δίνεται ως όρισμα στη γραμμή διαταγών.

```
/* wordcntf.c */
/* Καταμέτρηση λέξεων και γραμμάτων σε αρχείο κειμένου */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  char ch;
  int alfa=0,white=1,word=0;

  if(argc!=2)
  {
    puts("Δώσε: wordcntf όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"r"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s", argv[1]);
    exit(0);
  }
  while((ch=getc(fptr))!=EOF)
    switch(ch)
    {
      case ' ' :          /* αν ο χαρακτήρας είναι κενό, tab */
      case '\t':          /* νέα γραμμή, μετράται ως λευκός  */
      case '\n': white++; break;
      default  : if(ch>='A' && ch<='Z' || ch>='a' && ch<='z' ||
                    ch>='Α' && ch<='ψ' || ch>='ω' && ch<='ώ')
                    alfa++;            /* βλέπε πίνακα ASCII */
                 if(white) { white=0; word++;}
    }
  fclose(fptr);
  if(alfa==0) puts("Το αρχείο δεν περιέχει λέξεις");
  else
    printf("Λέξεις: %d\nΓράμματα: %d\nΜέση τιμή γραμμάτων
            ανά λέξη: %f",word,alfa,1.0*alfa/word);
}
```

2. Να γραφεί πρόγραμμα που να μετατρέπει όλους τους πεζούς ελληνικούς χαρακτήρες ενός αρχείου κειμένου (αρχείο-πηγή) σε κεφαλαίους και να τους γράφει σε άλλο αρχείο (αρχείο-προορισμού). Για τη μετατροπή των χαρακτήρων συμβουλευτείτε τον πίνακα ASCII.
Τα ονόματα των αρχείων πηγής και προορισμού, να δίνονται ως ορίσματα στη γραμμή διαταγών.

```
/* fconvert.c */
/* Μετατροπή αρχείου με ελληνικούς χαρακτήρες σε κεφαλαίους */
#include <stdio.h>
```

```
main(int argc, char *argv[])
{
  FILE *fptr1,*fptr2;
  char ch;

  if(argc!=3)
  {
    puts("Δώσε: fconvert όνομα_αρχείου1 όνομα_αρχείου2");
    exit(0);
  }
  if((fptr1=fopen(argv[1],"r"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s", argv[1]);
    exit(0);
  }
  if((fptr2=fopen(argv[2],"w"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s", argv[2]);
    exit(0);
  }
  while((ch=getc(fptr1))!=EOF)
  {
    if (ch>='α' && ch<='σ')      ch-=24; /* α..σ      */
    else if(ch>='τ' && ch<='ψ') ch-=25;  /* τ..ψ      */
    else if(ch=='ς') ch='Σ';             /* ς         */
    else if(ch=='ω') ch='Ω';             /* ω         */
    else if(ch=='ά') ch='A';             /* τονούμενα */
    else if(ch=='έ') ch='E';
    else if(ch=='ή') ch='H';
    else if(ch=='ί') ch='I';
    else if(ch=='ό') ch='O';
    else if(ch=='ύ') ch='Y';
    else if(ch=='ώ') ch='Ω';
    putc(ch,fptr2);
  }
  fclose(fptr1); fclose(fptr2);
}
```

3. Η C χειρίζεται τις διάφορες περιφερειακές συσκευές ως αρχεία. Τα αρχεία αυτά ανοίγουν αυτόματα στην αρχή κάθε προγράμματος και κλείνουν στο τέλος του. Τα ονόματα των δεικτών σ'αυτά τα αρχεία είναι προκαθορισμένα στο αρχείο-επικεφαλίδας `stdio.h` και είναι τα ακόλουθα:

Όνομα	*Συσκευή*
`stdin`	standard συσκευή εισόδου (πληκτρολόγιο).
`stdout`	standard συσκευή εξόδου (οθόνη).
`stderr`	standard συσκευή λάθους (οθόνη).
`stdaux`	standard βοηθητική συσκευή (σειριακή θύρα).
`stdprn`	standard συσκευή εκτύπωσης (εκτυπωτής).

Γράψτε ένα πρόγραμμα το οποίο να τυπώνει ένα αρχείο κειμένου (το όνομα του οποίου θα δίνεται ως όρισμα στην γραμμή διαταγών) στον εκτυπωτή.

```
/* fprint.c */
/* Εκτύπωση αρχείου στον εκτυπωτή */
#include <stdio.h>
main(int argc, char *argv[])
{
  FILE *fptr;
  char string[81];

  if(argc!=2)
  {
    puts("Δώσε: fprint όνομα_αρχείου");
    exit(0);
  }
  if((fptr=fopen(argv[1],"r"))==NULL)
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s", argv[1]);
    exit(0);
  }
  while(fgets(string,80,fptr)!=NULL)
  {
    strcat(string,"\r");
    fputs(string,stdprn);
  }
  fclose(fptr);
}
```

4. Να γραφεί πρόγραμμα το οποίο να διαβάζει τα περιεχόμενα ενός αρχείου (το οποίο έχει ανοιχτεί για δυαδική ανάγνωση) byte προς byte και να εμφανίζει τα περιεχόμενά του στην οθόνη. Η απεικόνιση να αποτελείται από δύο μέρη: το δεκαεξαδικό κωδικό κάθε χαρακτήρα και τον ίδιο το χαρακτήρα - αν είναι εκτυπώσιμος. Είναι προφανές ότι αυτό το πρόγραμμα θα προσομοιάζει τη λειτουργία dump (-d) του προγράμματος DEBUG του DOS.

```
/* dump.c */
/* Πρόγραμμα δυαδικής απεικόνισης αρχείου */
#include <stdio.h>
#define LENGTH 10
#define TRUE   1
#define FALSE  0
main(int argc, char *argv[])
{
  FILE *fptr;
  int ch,i,eof;
  unsigned char string[LENGTH+1];     /* buffer για τους χαρακτήρες  */

  if(argc!=2)
  {
    puts("Δώσε: dump όνομα_αρχείου"); exit(0);
  }
  if((fptr=fopen(argv[1],"rb"))==NULL)/* άνοιγμα για δυαδική ανάγνωση*/
  {
    printf("Δε μπορώ να ανοίξω το αρχείο %s", argv[1]);
    exit(0);
  }
```

```
  eof=FALSE;
  do
  {
    for(i=0;i<LENGTH;i++)
    {
      if((ch=getc(fptr))==EOF) eof=TRUE;
      printf("%5X",ch); /* εκτύπωση χαρακτήρα σε δεκαεξαδική μορφή */
      if(ch>31)         /* αν ο χαρακτήρας είναι εκτυπώσιμος       */
        string[i]=ch;   /* τοποθετείται στον buffer όπως είναι,    */
      else              /* διαφορετικά τοποθετείται η '.'          */
        string[i]='.';
    }
    string[LENGTH]='\0';
    printf("    %s\n",string); /* εκτύπωση του buffer */
  }
  while(eof==FALSE);
  fclose(fptr);
}
```

5. Το παρακάτω πρόγραμμα, διαχειρίζεται πλήρως ένα αρχείο εγγραφών (κάθε εγγραφή έχει δύο πεδία: ονοματεπώνυμο και τηλέφωνο). Οι εργασίες που επιτελεί είναι: Εισαγωγή, Ενημέρωση, Διαγραφή, Αναζήτηση, Ταξινόμηση και Παρουσίαση. Το πρόγραμμα αποτελείται από συναρτήσεις, μία για κάθε εργασία, έτσι οι συναρτήσεις αυτές μπορούν να χρησιμοποιηθούν και σε οποιοδήποτε άλλο πρόγραμμα που χρειάζεται μία από τις παραπάνω εργασίες.

```
/* agenda.c */
/* Διαχείριση τηλεφωνικού καταλόγου */
#include <stdio.h>
struct record
{
  char name[50];
  char phone[10];
} person;
int size=sizeof(struct record);
FILE     *fptr;
void     add();
void     update();
void     delete();
long int search();
void     sort();
void     list();

main()
{
  char select;
  do
  {
    clrscr();
    puts("ΔΙΑΧΕΙΡΙΣΗ ΤΗΛΕΦΩΝΙΚΟΥ ΚΑΤΑΛΟΓΟΥ\n");
    puts("1. Εισαγωγή   εγγραφής");
    puts("2. Ενημέρωση  εγγραφής");
    puts("3. Διαγραφή   εγγραφής");
```

```
    puts("4. Αναζήτηση  τηλεφώνου");
    puts("5. Ταξινόμηση καταλόγου");
    puts("6. Παρουσίαση καταλόγου");
    puts("7. Ε Ξ Ο Δ Ο Σ\n");
    printf("Επιλογή: "); select=getche();
    switch(select)
    {
      case '1': add();    break;
      case '2': update(); break;
      case '3': delete(); break;
      case '4': search(); break;
      case '5': sort();   break;
      case '6': list();   break;
    }
  }
  while(select!='7');
}


/* add */
/* προσθήκη εγγραφής στον κατάλογο */
void add()
{
  clrscr();
  fptr=fopen("AGENDA.DAT","ab");
  printf("Ονομα   : ");
  gets(person.name);
  printf("Τηλέφωνο: ");
  gets(person.phone);
  fwrite(&person,size,1,fptr);
  fclose(fptr);
}


/* update */
/* ενημέρωση εγγραφής */
void update()
{
  long int pos;
  pos=search();
  if(pos>(long int) 0)
  {
    fptr=fopen("AGENDA.DAT","r+b");
    printf("Νέο τηλέφωνο: ");
    gets(person.phone);
    fseek(fptr,pos-size,SEEK_SET);
    fwrite(&person,size,1,fptr);
    fclose(fptr);
  }
}

/* delete */
/* διαγραφή εγγραφής */
void delete()
```

```
{
  long int pos;
  FILE *temp;
  char sname[50];
  pos=search();
  if(pos>(long int) 0)
  {
    strcpy(sname,person.name);
    fptr=fopen("AGENDA.DAT","rb");
    temp=fopen("TEMP.DAT","wb");
    fread(&person,size,1,fptr);
    while(!feof(fptr))
    {
      if(strcmp(sname,person.name)!=0)
        fwrite(&person,size,1,temp);
      fread(&person,size,1,fptr);
    }
    fclose(fptr); fclose(temp);
    remove("AGENDA.DAT");
    rename("TEMP.DAT","AGENDA.DAT");
    remove("TEMP.DAT");
  }
}

/* search */
/* αναζήτηση εγγραφής με κλειδί το ονοματεπώνυμο */
long int search()
{
  long int pos=0L;
  int found=0;
  char sname[50];
  clrscr();
  if((fptr=fopen("AGENDA.DAT","rb"))==NULL)
    puts("Δε μπορώ να ανοίξω το αρχείο AGENDA");
  else
  {
    printf("Ονομα   : ");
    gets(sname);
    fread(&person,size,1,fptr);
    while(!feof(fptr) && !found)
    {
      if(strcmp(person.name,sname)==0)
      {
        found=1;
        printf("%s   %s\n",person.name,person.phone);
        pos=ftell(fptr);
      }
      if(!found) fread(&person,size,1,fptr);
    }
    fclose(fptr);
    if(!found)
      puts("Ανύπαρκτο όνομα");
  }
  getch();
```

```
  return(pos);
}


/* sort() */
/* ταξινόμηση του αρχείου ως προς το όνομα     */
/* χρησιμοποιείται η απλή μέθοδος της επιλογής */
void sort()
{
  int fsize=0;
  long int pos,posmin,pos1;
  struct record min,temp;
  clrscr();
  if((fptr=fopen("AGENDA.DAT","r+b"))==NULL)
    puts("Δε μπορώ να ανοίξω το αρχείο AGENDA");
  else
  { /* βρίσκουμε το πλήθος των εγγραφών του αρχείου */
    while(fread(&person,size,1,fptr)==1)
      fsize++;
    for(pos=0;pos<fsize-1;pos++)
    {
      fseek(fptr,pos*size,SEEK_SET);
      fread(&person,size,1,fptr);
      posmin=pos;
      min=person;
      for(pos1=pos+1;pos1<fsize;pos1++)
      {
        fseek(fptr,pos1*size,SEEK_SET);
        fread(&temp,size,1,fptr);
        if(strcmp(temp.name,min.name)<0)
        {
          min=temp;
          posmin=pos1;
        }
      }
      if (strcmp(min.name,person.name)!=0)
      {
        fseek(fptr,pos*size,SEEK_SET);
        fwrite(&min,size,1,fptr);
        fseek(fptr,posmin*size,SEEK_SET);
        fwrite(&person,size,1,fptr);
      }
    }
    fclose(fptr);
    puts("Το αρχείο ταξινομήθηκε");
  }
  getch();
}


/* list */
/* παρουσίαση όλων των εγγραφών του καταλόγου */
void list()
{
```

```
  int i=0;
  clrscr();
  if((fptr=fopen("AGENDA.DAT","rb"))==NULL)
    puts("Δε μπορώ να ανοίξω το αρχείο AGENDA");
  else
  {
    fread(&person,size,1,fptr);
    while(!feof(fptr))
    {
      printf("%3d. %s   %s\n",++i,person.name,person.phone);
      fread(&person,size,1,fptr);
    }
  }
  getch();
}
```

11.5 ΑΣΚΗΣΕΙΣ

1. Γράψτε πρόγραμμα που να μετρά τη συχνότητα των γραμμάτων σ'ένα αρχείο κειμένου, το όνομα του οποίου θα δίνεται ως όρισμα στη γραμμή διαταγών. Ποιο είναι το περισσότερο χρησιμοποιούμενο γράμμα στα ελληνικά κείμενα και ποιο στα αγγλικά;
2. Ας υποθέσουμε ότι ορίζουμε τέσσερις κατηγορίες λέξεων ανάλογα με το μήκος τους:

   α) Στην πρώτη κατηγορία ανήκουν οι λέξεις που έχουν 1-3 γράμματα

   β) Στη δεύτερη, αυτές που έχουν 4-6 γράμματα

   γ) Στην τρίτη, αυτές που έχουν 7-9 γράμματα

   δ) Στην τέταρτη, οι λέξεις που έχουν 10 και πλέον γράμματα.

   Να γραφεί πρόγραμμα που να διαβάζει ένα αρχείο κειμένου και να μετρά το πλήθος των λέξεων που ανήκουν σε κάθε κατηγορία.
3. Γράψτε ένα πρόγραμμα που να διαβάζει από το δίσκο ένα αρχείο πηγαίου προγράμματος της C και να ελέγχει αν το πλήθος των αριστερών αγκίστρων (`{`) είναι ίσο με το πλήθος των δεξιών αγκίστρων (`}`). Το όνομα του αρχείου πηγαίου προγράμματος να δίνεται ως όρισμα στη γραμμή διαταγών.
4. Να γραφεί πρόγραμμα που να αντιγράφει ένα αρχείο σε ένα άλλο, δηλαδή να προσομοιάζει τη διαταγή COPY του DOS.
   Στη συνέχεια το πρόγραμμα να ελέγχει αν η αντιγραφή έγινε σωστά, δηλαδή να προσομοιάζει τη διαταγή COMP του DOS.
5. Γράψτε πρόγραμμα που να δημιουργεί ένα αρχείο πραγματικών αριθμών τους οποίους θα δέχεται από το πληκτρολόγιο. Οι αριθμοί που θα καταχωρούνται να είναι αυτοί που θα βρίσκονται στο διάστημα `[0..10]`.
6. Να γραφεί πρόγραμμα που να βρίσκει τη μέση τιμή των αριθμών που υπάρχουν στο αρχείο που δημιουργήθηκε από το πρόγραμμα της Άσκησης 5. Στη συνέχεια να βρίσκει το πλήθος των αριθμών οι οποίοι είναι μεγαλύτεροι της μέσης τιμής, καθώς και αυτών που είναι μικρότεροι αυτής.

7. Να γραφεί πρόγραμμα για τη συνένωση δύο αρχείων του ίδιου τύπου. Τα δύο αρχεία να δίνονται ως ορίσματα στη γραμμή διαταγών και να δημιουργείται ένα τρίτο αρχείο με τα στοιχεία του πρώτου και στη συνέχεια τα στοιχεία του δεύτερου.

8. Γράψτε πρόγραμμα, το οποίο με τη βοήθεια ενός menu να μας δίνει τη δυνατότητα για τις εξής επιλογές:

   α) Δημιουργία αρχείου ακεραίων.

   β) Ταξινόμηση κατά αύξουσα σειρά, του αρχείου ακεραίων.

   γ) Παρουσίαση των δεδομένων, του αρχείου ακεραίων στην οθόνη.

   Για την ταξινόμηση του αρχείου μπορείτε να χρησιμοποιήσετε τη μέθοδο της επιλογής, που έχει περιγραφεί στην § 8.1.6, ή τη μέθοδο της φυσαλίδας που περιγράφεται στο Παράδειγμα 2 της § 8.3.

9. Υποθέστε ότι με την Άσκηση 8 δημιουργήσαμε και ταξινομήσαμε δύο αρχεία ακεραίων. Γράψτε ένα πρόγραμμα που να δημιουργεί ένα τρίτο αρχείο από τη συνταξινόμηση των δύο προηγουμένων. Η διαδικασία της συνταξινόμησης έχει περιγραφεί στο Παράδειγμα 4 του Κεφαλαίου 8.

10. Ένα γραφείο συνοικεσίων έχει δημιουργήσει ένα αρχείο, το οποίο περιέχει στοιχεία των υποψήφιων γαμπρών και των υποψήφιων νυφών. Για κάθε πελάτη έχει καταγραφεί το επίθετο, το όνομα, η διεύθυνση, ο αριθμός τηλεφώνου, το φύλλο (Α ή Θ), η ηλικία, το ύψος, το βάρος, το χρώμα των ματιών, το χρώμα των μαλλιών, το επάγγελμα, η παλιά οικογενειακή κατάσταση, το κύριο χόμπι, οι αδυναμίες και οι παραξενιές.
    Να γραφεί πρόγραμμα που να δημιουργεί και να ενημερώνει το αρχείο. Επίσης να δέχεται τις προτιμήσεις ενός νέου πελάτη και να δίνει μία λίστα πιθανών συζύγων.

11. Να γραφεί πρόγραμμα που να δέχεται ως όρισμα στην γραμμή διαταγών ένα αρχείο κειμένου και να δημιουργεί ένα άλλο αρχείο, έτσι ώστε η κάθε γραμμή του να μην είναι μεγαλύτερη από 40 χαρακτήρες. Για το λόγο αυτό θα πρέπει να προστίθενται λέξεις σε κάθε γραμμή έως ότου να συμπληρώνεται ο αριθμός των 40 χαρακτήρων, ενώ σε περίπτωση που μία λέξη δε χωράει σε μία γραμμή να τοποθετείται στην επόμενη.

12. Γράψτε πρόγραμμα το οποίο να δημιουργεί ένα αρχείο εγγραφών μαθητών (κύριο). Η κάθε εγγραφή να περιλαμβάνει τις πληροφορίες: 1. Αριθμός μητρώου, 2. Ονοματεπώνυμο, 3. Φύλλο και 4. Βαθμός.
    Στη συνέχεια να γραφεί ένα άλλο πρόγραμμα, που να δημιουργεί δύο νέα αρχεία παίρνοντας τις εγγραφές του κύριου αρχείου, έτσι ώστε στο ένα να υπάρχουν όλοι οι μαθητές και στο άλλο όλες τις μαθήτριες.

ΚΕΦΑΛΑΙΟ 12
Ο ΠΡΟΕΠΕΞΕΡΓΑΣΤΗΣ

Κάθε πηγαίο πρόγραμμα της C πριν μεταγλωττιστεί περνά από τη φάση της *προεπεξεργασίας*, βλέπε Σχήμα 2.1. Τη διαδικασία αυτή την επιτελεί ένα πρόγραμμα που *λέγεται προεπεξεργαστής της C* (C preprocessor). Ο προεπεξεργαστής ψάχνει στο πηγαίο πρόγραμμα και εκτελεί οδηγίες που απευθύνονται σ'αυτόν. Κάθε *οδηγία προς τον προεπεξεργαστή* (preprocessor directive) αρχίζει με το σύμβολο `#`. Σ'αυτό το Κεφάλαιο θα δούμε τις σημαντικότερες οδηγίες προς τον προεπεξεργαστή. Αν και δύο απ'αυτές, την `#define` και την `#include` τις έχουμε ήδη χρησιμοποιήσει σε διάφορα προγράμματα, εδώ θα τις δούμε πιο αναλυτικά.

12.1 `#define` και `#undef`

Η γενική μορφή αυτής της οδηγίας `#define` είναι:

#define *αναγνωριστικό ακολουθία_συμβόλων*

Αυτή η οδηγία κάνει τον προεπεξεργαστή να αντικαταστήσει όλες τις εμφανίσεις του *αναγνωριστικού* στο αρχείο πηγαίου κώδικα, με την καθοριζόμενη *ακολουθία_συμβόλων* (όπως η επιλογή "βρες και αντικατέστησε" (find and replace) στους επεξεργαστές κειμένου).

Η πιο απλή χρήση της οδηγίας `#define` είναι η ανάθεση ονομάτων σε σταθερές, όπως έχουμε δει και στην § 3.3.2. Για παράδειγμα βλέπε το πρόγραμμα `sphere.c` στην § 7.3 .

Η *ακολουθία_συμβόλων* μπορεί ακόμα να είναι και μία εντολή της C. Ας υποθέσουμε ότι στο πρόγραμμά μας θέλουμε να τυπώσουμε το μήνυμα "Λάθος" σε διάφορα σημεία. Χρησιμοποιώντας της οδηγία:

```
#define ERROR  printf("\nΛάθος\n");
```

αν έχουμε στο πρόγραμμα μία εντολή σαν κι αυτή:

```
if (input>640) ERROR
```

πριν τη μεταγλώττιση, θα επεκταθεί από τον προεπεξεργαστή σε:

```
if (input>640) printf("\nΛάθος\n");
```

Μία οδηγία `#define` μπορεί να έχει και παραμέτρους:

#define *αναγνωριστικό****(****λίστα_αναγνωριστικών****)*** *ακολουθία_συμβόλων*

όπου δεν υπάρχει κενό ανάμεσα στο *αναγνωριστικό* και στην παρένθεση '(`'`. Στην περίπτωση αυτή μιλάμε για *μακροεντολές* (macros).

Το παρακάτω πρόγραμμα χρησιμοποιεί αυτή τη δυνατότητα για να τυπώνει χωρίς μπελά δύο αριθμούς κινητής υποδιαστολής:

```
/* macroprn.c */
/* Χρήση μακροεντολών */
#define PR(n) printf("%.2f\n",n);
main()
{
  float num1, num2=455.89;

  num1=1.0/4.0;
  PR(num1)
  PR(num2)
}
```

Σ'αυτό το πρόγραμμα όταν ο προεπεξεργαστής βλέπει τη φράση "`PR(n)`" την επεκτείνει στην εντολή: `printf("%.2f\n",n);`

Οι μακροεντολές έχουν κάποια από τα χαρακτηριστικά των *συναρτήσεων*, όπως θα γίνει πιο σαφές στο επόμενο παράδειγμα:

```
/* sphereM.c */
/* Υπολογισμός εμβαδού σφαίρας, χρησιμοποιώντας μακροεντολή */
#define PI 3.14159          /* καθορισμός του "π"            */
#define AREA(X) (4*PI*X*X)  /* μακροεντολή για το εμβαδόν    */
main()
{
  float radius;

  puts("Δώσε την ακτίνα της σφαίρας:");
  scanf("%f",&radius);
  printf("Το Εμβαδόν της σφαίρας είναι %.2f\n",AREA(radius));
}
```

Αυτό το πρόγραμμα κάνει το ίδιο με το πρόγραμμα `sphere.c` της § 7.3, μόνο που τώρα αντί για συνάρτηση χρησιμοποιεί μακροεντολή.

Στο Παράδειγμα 1 της § 7.10 είδαμε μία συνάρτηση, την `max()`, η οποία επέστρεφε το μεγαλύτερο από δύο πραγματικούς αριθμούς. Θα μπορούσαμε αντί αυτής να χρησιμοποιήσουμε μακροεντολή:

```
#define max(n1,n2)  n1>n2?n1:n2
```

Αυτή η μακροεντολή μπορεί να χρησιμοποιηθεί για οποιουδήποτε τύπου μεταβλητές `n2` και `n2`, ενώ όταν ορίζουμε μία συνάρτηση πρέπει να δηλώσουμε και τους τύπους των παραμέτρων τους.

Η χρήση μακροεντολών αντί των συναρτήσεων έχει ένα *πλεονέκτημα*: το πρόγραμμα εκτελείται ταχύτερα, γιατί η κλήση της συνάρτησης απαιτεί κάποιο χρόνο. Ωστόσο το *μειονέκτημα* των μακροεντολών είναι ότι το μέγεθος του προγράμματος αυξάνεται εξ'αιτίας του επαναλαμβανόμενου κώδικα.

Όταν ορίζουμε μακροεντολές, χρειάζεται προσοχή στις παρενθέσεις που χρησιμοποιούμε. Ας υποθέσουμε ότι το πρόγραμμά μας έχει τις εξής γραμμές:

```
#define SUM(x,y) x+y
...
ans=10*SUM(3,4);
```

ποια τιμή θα δοθεί στην `ans`; Σκεφτόμενος κάποιος επιπόλαια θα νομίσει ότι η `ans` θα πάρει την τιμή 70 (`10*(3+4)`), όμως αυτό είναι λάθος. Ο προεπεξεργαστής θα επεκτείνει την εντολή απόδοσης τιμής ως εξής:

```
ans=10*3+4;
```

απ'όπου προκύπτει ότι η `ans` παίρνει την τιμή 34. Αν όμως είχαμε την οδηγία:

```
#define SUM(x,y) (x+y)
```

τώρα η εντολή απόδοσης τιμής επεκτείνεται σε

```
ans=10*(3+4);
```

και η `ans` παίρνει την τιμή 70.

Ας υποθέσουμε τώρα ότι έχουμε τις ακόλουθες γραμμές κώδικα:

```
#define PRODUCT(x,y) (x*y)
...
ans=PRODUCT(2+3,4);
```

ποια τιμή θα πάρει η `ans`; Αν νομίζουμε ότι θα πάρει την τιμή 20 κάνουμε λάθος, διότι έχουμε:

```
ans=(2+3*4)
```

απ'όπου προκύπτει ότι η `ans` παίρνει την τιμή 14. Αν όμως είχαμε γράψει την οδηγία:

```
#define PRODUCT(x,y) ((x)*(y))
```

τότε θα είχαμε:

```
ans=((2+3)*4)
```

δηλαδή η `ans` παίρνει την τιμή 20.

Η *εμβέλεια* ενός αναγνωριστικού ή μίας μακροεντολής που έχει οριστεί με μία `#define` εκτείνεται από το σημείο ορισμού του μέχρι το τέλος του αρχείου που μεταγλωττίζεται, ή μέχρι να συναντηθεί η οδηγία:

#undef *αναγνωριστικό*

Δηλαδή η οδηγία `#undef` χρησιμοποιείται για την *ακύρωση* του ορισμού ενός αναγνωριστικού ή μίας μακροεντολής που έχει οριστεί με `#define`.

12.2 #include

Η οδηγία `#include` δίνει εντολή στον προεπεξεργαστή να *συμπεριλάβει* ένα άλλο αρχείο πηγαίου κώδικα, σ'αυτό που περιέχει την οδηγία. Η γενική μορφή της είναι:

#include *"όνομα_αρχείου"*
ή
#include ***<***_όνομα_αρχείου_***>***

Αν το *όνομα_αρχείου* βρίσκεται μέσα σε εισαγωγικά (" ") η αναζήτηση του αρχείου αυτού γίνεται στον κατάλογο (directory) που περιέχει το αρχείο πηγαίου κώδικα. Αν το *όνομα_αρχείου* βρίσκεται σε γωνιώδεις αγκύλες (< >) η αναζήτηση γίνεται στον κατάλογο των αρχείων-επικεφαλίδας (header files) που συνήθως ονομάζεται INCLUDE.

Ένα παράδειγμα που δείχνει γιατί πιθανώς θα θέλαμε να συμπεριλάβουμε ένα αρχείο σ'ένα άλλο, είναι το εξής: Ας υποθέσουμε ότι γράφουμε πολλά

προγράμματα μαθηματικών που συνεχώς αναφέρονται σε αλγορίθμους για τον υπολογισμό των εμβαδών διαφόρων σχημάτων. Θα μπορούσαμε να τοποθετήσουμε όλους αυτούς τους αλγορίθμους ως μακροεντολές σ'ένα ξεχωριστό αρχείο. Ύστερα αντί να πρέπει να ξαναγράφουμε όλες τις μακροεντολές κάθε φορά που γράφουμε ένα πρόγραμμα που τις χρησιμοποιεί, θα μπορούμε να τις εισάγουμε στο πηγαίο αρχείο χρησιμοποιώντας την οδηγία `#include`. Ένα τέτοιο ξεχωριστό αρχείο θα μπορούσε να ήταν κάπως έτσι:

```
#define PI                                3.14159
#define AREA_CIRCLE(radius)               (PI*radius*redius)
#define AREA_SQUARE(length,width)         (length*width)
#define AREA_TRIANGLE(base,height)        (base*height/2)
#define AREA_ELLIPSE(radius1,radius2)     (PI*radius1*radius2)
#define AREA_TRAPEZOID(height,b1,b2)      (height*(b1+b2)/2)
```

Θα μπορούσαμε να ονομάσουμε αυτό το αρχείο `areas.h`. Η προέκταση '`h`' χρησιμοποιείται για τα αρχεία-επικεφαλίδας, τα οποία είναι μία ομάδα εντολών που τοποθετείται στην αρχή του προγράμματός μας.

Η γλώσσα C παρέχεται μ'ένα πλήθος από αρχεία-επικεφαλίδας για όλες τις συναρτήσεις βιβλιοθήκης. Κάθε αρχείο περιέχει τους ορισμούς και τα πρότυπα για μία κατηγορία συναρτήσεων καθώς και μακροεντολές. Τα αρχεία αυτά βρίσκονται ομαδοποιημένα στον κατάλογο INCLUDE. Στην πραγματικότητα κάποιες πολύ συνηθισμένες "συναρτήσεις" βιβλιοθήκης της C είναι μακροεντολές που ορίζονται στα αρχεία-επικεφαλίδας. Π.χ. οι "συναρτήσεις" `getchar()` και `putchar()` που είδαμε στην § 5.1, στην ουσία είναι μακροεντολές που ορίζονται με τη βοήθεια των συναρτήσεων `getc()` και `putc()`, που είδαμε στην § 11.2.1. Το αρχείο `stdio.h` περιέχει αυτούς τους ορισμούς:

```
#define getchar()  getc(stdin)
#define putchar(c) putc((c),stdout)
```

Έτσι αν θελήσουμε να χρησιμοποιήσουμε κάποια απ'αυτές τις συναρτήσεις στο πρόγραμμά μας, πρέπει να χρησιμοποιήσουμε την οδηγία: `#include <stdio.h>`.

Επίσης στα αρχεία-επικεφαλίδας υπάρχουν και τα *πρότυπα* (βλέπε § 7.1.1) των συναρτήσεων βιβλιοθήκης. Έτσι είναι καλό να συμπεριλαμβάνουμε τα κατάλληλα αρχεία-επικεφαλίδας, πριν χρησιμοποιήσουμε κάποια συνάρτηση βιβλιοθήκης.

12.3 `#if`, `#else`, `#endif`, `#elif`, `#ifdef` και `#ifndef`

Σε ορισμένες περιπτώσεις μπορεί να είναι επιθυμητή ή όχι η μεταγλώττιση ορισμένων τμημάτων του κώδικα ενός αρχείου. Οι οδηγίες `#if`, `#else`, `#end` και `#elif`, χρησιμοποιούνται ως *οδηγίες μεταγλώττισης υπό συνθήκη*.

Η γενική μορφή της οδηγίας `#if` είναι:

```
#if σταθερή_παράσταση
  εντολή1
#else
  εντολή2
#endif
```

όπου το τμήμα του `#else` είναι *προαιρετικό*, ενώ το `#endif` είναι *υποχρεωτικό*.

Αν η *σταθερή_παράσταση* που ακολουθεί το `#if` είναι αληθής (μη-μηδενική) τότε θα μεταγλωττιστεί η *εντολή*$_1$, ενώ αν είναι ψευδής (ίση με μηδέν) και υπάρχει το τμήμα `#else`, θα μεταγλωττιστεί η *εντολή*$_2$.

Για παράδειγμα το επόμενο απόσπασμα προγράμματος, χρησιμοποιεί την τιμή της `ACTIVE_COUNTRY` για να ορίσει το λεκτικό του νομίσματος:

```
/* if.c */
/* Μεταγλώττιση υπό συνθήκη με #if #else #endif */
#define US      0
#define ENGLAND 1
#define GREECE  2
#define FRANCE  3
#define ACTIVE_COUNTRY GREECE
#if ACTIVE_COUNTRY==US
  char currency[]="δολάριο";
#else
  #if ACTIVE_COUNTRY==ENGLAND
    char currency[]="λίρα";
  #else
    #if ACTIVE_COUNTRY==GREECE
      char currency[]="δραχμή";
    #else
      #if ACTIVE_COUNTRY==FRANCE
        char currency[]="φράγκο";
      #endif
    #endif
  #endif
#endif
main()
{
  puts(currency); /* προφανώς τυπώνεται η λέξη "δραχμή" */
}
```

Στο παραπάνω πρόγραμμα βλέπουμε ότι οι `#else` ακολουθούνται από `#if`. Στην περίπτωση αυτή μπορούμε να αντικαταστήσουμε τις ακολουθίες:

```
#else
  #if
```

με την οδηγία:

```
#elif
```

Έτσι το παραπάνω πρόγραμμα γίνεται:

```
/* elif.c */
```

```
/* Μεταγλώττιση υπό συνθήκη με #if #elif #endif */
#define US      0
#define ENGLAND 1
#define GREECE  2
#define FRANCE  3
#define ACTIVE_COUNTRY GREECE
#if ACTIVE_COUNTRY==US
  char currency[]="δολάριο";
#elif ACTIVE_COUNTRY==ENGLAND
  char currency[]="λίρα";
#elif ACTIVE_COUNTRY==GREECE
  char currency[]="δραχμή";
#elif ACTIVE_COUNTRY==FRANCE
  char currency[]="φράγκο";
#endif
main()
{
  puts(currency);
}
```

Μία άλλη μέθοδος μεταγλώττισης υπό συνθήκη, χρησιμοποιεί τις οδηγίες `#ifdef` και `#ifndef` που σημαίνουν "if defined" (αν έχει οριστεί) και "if not defined" (αν δεν έχει οριστεί).

Η γενική μορφή της `#ifdef` είναι:

```
#ifdef  αναγνωριστικό_μακροεντολής
  εντολή1
#else
  εντολή2
#endif
```

όπου το τμήμα του `#else` είναι *προαιρετικό*. Όπως είναι αναμενόμενο η *εντολή*$_1$ θα μεταγλωττιστεί αν και μόνο αν έχει προηγουμένως οριστεί το *αναγνωριστικό_μακροεντολής*, με μία `#define`, διαφορετικά, αν υπάρχει τμήμα `#else`, θα εκτελεστεί η *εντολή*$_2$.

Η γενική μορφή της `#ifndef` είναι:

```
#ifndef  αναγνωριστικό_μακροεντολής
  εντολή1
#else
  εντολή2
#endif
```

όπου το τμήμα του `#else` είναι *προαιρετικό*. Στην περίπτωση αυτή η *εντολή*$_1$ θα μεταγλωττιστεί αν και μόνο αν *δεν* έχει οριστεί προηγουμένως το *αναγνωριστικό_μακροεντολής*, αν όμως έχει οριστεί και υπάρχει τμήμα `#else`, θα εκτελεστεί η *εντολή*$_2$.

Οι `#ifdef` και `#ifndef` μπορούν να χρησιμοποιήσουν την `#else`, όμως δε μπορούν να χρησιμοποιήσουν την `#elif`.

12.4 ΑΣΚΗΣΕΙΣ

1. Εξηγήστε τα αποτελέσματα του παρακάτω προγράμματος:

```
#define SQUARE(x)      x*x
#define SUM(x,y)       x+y
#define PRODUCT(x,y)  (x*y)
#define TYPE(x)        printf("%d\n",x)
main()
{
  TYPE(SQUARE(3+1));
  TYPE(125/SQUARE(5));
  TYPE(10*SUM(3,4));
  TYPE(PRODUCT(2+3,4));
}
```

2. Δημιουργείστε ένα αρχείο-επικεφαλίδας με όνομα `embada.h` το οποίο να περιέχει μακροεντολές για τον υπολογισμό εμβαδών των εξής σχημάτων: ορθογωνίου παραλληλογράμμου, κύκλου, τριγώνου και τραπεζίου. Στη συνέχεια γράψτε ένα πρόγραμμα που να υπολογίζει τα εμβαδά των παραπάνω σχημάτων, κάνοντας χρήση του αρχείου `embada.h`.

3. Ορίστε μία μακροεντολή `swap(type,x,y)` που να εναλλάσσει τις τιμές των μεταβλητών `x` και `y`, καθεμιά τύπου `type`. Η δόμηση σε μπλοκ (σύνθετη εντολή) θα βοηθήσει.

4. Ποιο είναι το αποτέλεσμα του παρακάτω προγράμματος:

```
/* define.c */
int N=100; /* εξωτερική μεταβλητή */
main()
{
  printf("%d\n",N);
  #define N 123
  printf("%d\n",N);
  f();
}

f()
{
  printf("%d\n",N);
  #undef N
  printf("%d\n",N);
}
```

5. Ας υποθέσουμε ότι το αρχείο `a.h` έχει τα εξής περιεχόμενα:

```
/* a.h */
#define N 1000
```

Ενώ το αρχείο `b.c` τα παρακάτω:

```
/* b.c */
#include <a.h>
main()
{
  #ifndef N
    printf("Δεν έχει οριστεί το N στο A.H\n");
  #else
    #if N<=100
      float matrix[N][N];
      .......
      .......
    #else
      printf("Το N είναι πολύ μεγάλο\n");
    #endif
  #endif
}
```

Ποια θα είναι τα αποτελέσματα όταν εκτελέσουμε το αρχείο `b.c`;

ΠΑΡΑΡΤΗΜΑ Α'
ΤΟ ΣΥΝΟΛΟ ΧΑΡΑΚΤΗΡΩΝ ASCII

Δεκαδικός	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*		*Χαρακτήρας ελέγχου*
0	00		Ctrl-2	NUL	Null
1	01	☺	Ctrl-A	SOH	Start Of Heading
2	02	☻	Ctrl-B	STX	Start Of Text
3	03	♥	Ctrl-C	ETX	End Of Text
4	04	♦	Ctrl-D	EOT	End Of Transmission
5	05	♣	Ctrl-E	ENQ	Enquiry
6	06	♠	Ctrl-F	ACK	Acknowledge
7	07	•	Ctrl-G	BEL	Bell
8	08	◘	Ctrl-H	BS	Backspace
9	09	○	Ctrl-I	HT	Horizontal Tabulation
10	0A	◙	Ctrl-J	LF	Line Feed
11	0B	♂	Ctrl-K	VT	Vertical Tabulation
12	0C	♀	Ctrl-L	FF	Form Feed
13	0D	♪	Ctrl-M	CR	Carriage Return
14	0E	♫	Ctrl-N	SO	Shift Out
15	0F	☼	Ctrl-O	SI	Shift In
16	10	►	Ctrl-P	DLE	Data Link Escape
17	11	◄	Ctrl-Q	DC1	Device Control 1
18	12	↕	Ctrl-R	DC2	Device Control 2
19	13	‼	Ctrl-S	DC3	Device Control 3
20	14	¶	Ctrl-T	DC4	Device Control 4
21	15	§	Ctrl-U	NAK	Negative Acknowledge
22	16	▬	Ctrl-V	SYN	Synchronous Idle
23	17	↨	Ctrl-W	ETB	End of Transmission Block
24	18	↑	Ctrl-X	CAN	Cancel
25	19	↓	Ctrl-Y	EM	End Of Medium
26	1A	→	Ctrl-Z	SUB	Substitute
27	1B	←	Ctrl-[	ESC	Escape
28	1C	∟	Ctrl-\	FS	File Separator
29	1D	↔	Ctrl-]	GS	Group Separator
30	1E	▲	Ctrl-6	RS	Record Separator
31	1F	▼	Ctrl-_	US	Unit Separator

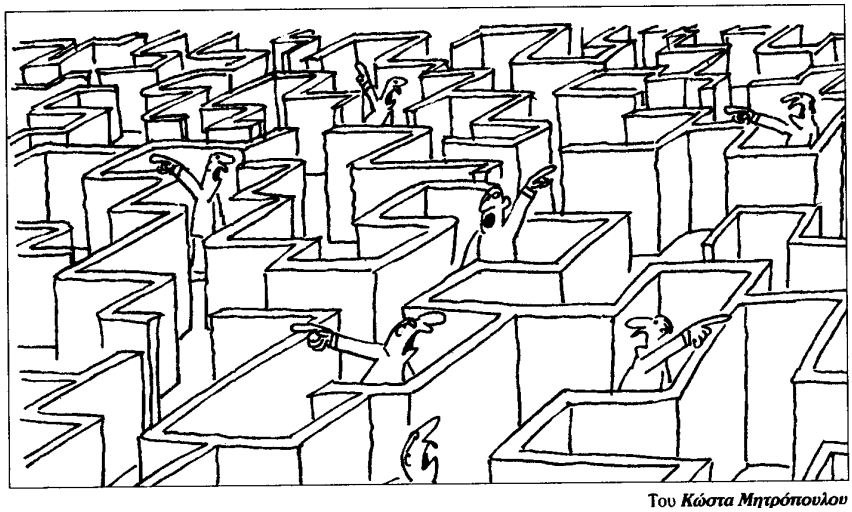
Δεκαδικός	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*	*Δεκαδικός*	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*
32	20		Space	74	4A	J	J
33	21	!	!	75	4B	K	K
34	22	"	"	76	4C	L	L
35	23	#	#	77	4D	M	M
36	24	$	$	78	4E	N	N
37	25	%	%	79	4F	O	O
38	26	&	&	80	50	P	P
39	27	'	'	81	51	Q	Q
40	28	(	(	82	52	R	R
41	29	)	)	83	53	S	S
42	2A	*	*	84	54	T	T
43	2B	+	+	85	55	U	U
44	2C	,	,	86	56	V	V
45	2D	-	-	87	57	W	W
46	2E	.	.	88	58	X	X
47	2F	/	/	89	59	Y	Y
48	30	0	0	90	5A	Z	Z
49	31	1	1	91	5B	[	[
50	32	2	2	92	5C	\	\
51	33	3	3	93	5D	]	]
52	34	4	4	94	5E	^	^
53	35	5	5	95	5F	_	_
54	36	6	6	96	60	`	`
55	37	7	7	97	61	a	a
56	38	8	8	98	62	b	b
57	39	9	9	99	63	c	c
58	3A	:	:	100	64	d	d
59	3B	;	;	101	65	e	e
60	3C	<	<	102	66	f	f
61	3D	=	=	103	67	g	g
62	3E	>	>	104	68	h	h
63	3F	?	?	105	69	i	i
64	40	@	@	106	6A	j	j
65	41	A	A	107	6B	k	k
66	42	B	B	108	6C	l	l
67	43	C	C	109	6D	m	m
68	44	D	D	110	6E	n	n
69	45	E	E	111	6F	o	o
70	46	F	F	112	70	p	p
71	47	G	G	113	71	q	q
72	48	H	H	114	72	r	r
73	49	I	I	115	73	s	s

Δεκαδικός	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*	*Δεκαδικός*	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*
116	74	t	t	157	9D	ζ	ζ
117	75	u	u	158	9E	η	η
118	76	v	v	159	9F	θ	θ
119	77	w	w	160	A0	ι	ι
120	78	x	x	161	A1	κ	κ
121	79	y	y	162	A2	λ	λ
122	7A	z	z	163	A3	μ	μ
123	7B	{	{	164	A4	ν	ν
124	7C	|	|	165	A5	ξ	ξ
125	7D	}	}	166	A6	ο	ο
126	7E	~	~	167	A7	π	π
127	7F	Δ	Del	168	A8	ρ	ρ
128	80	Α	Α	169	A9	σ	σ
129	81	Β	Β	170	AA	ς	ς
130	82	Γ	Γ	171	AB	τ	τ
131	83	Δ	Δ	172	AC	υ	υ
132	84	Ε	Ε	173	AD	φ	φ
133	85	Ζ	Ζ	174	AE	χ	χ
134	86	Η	Η	175	AF	ψ	ψ
135	87	Θ	Θ	176	B0	▒	Alt-176
136	88	Ι	Ι	177	B1	▓	Alt-177
137	89	Κ	Κ	178	B2	▒	Alt-178
138	8A	Λ	Λ	179	B3	┃	Alt-179
139	9B	Μ	Μ	180	B4	┤	Alt-180
140	9C	Ν	Ν	181	B5	╡	Alt-181
141	9D	Ξ	Ξ	182	B6	╢	Alt-182
142	8E	Ο	Ο	183	B7	╖	Alt-183
143	8F	Π	Π	184	B8	╕	Alt-184
144	90	Ρ	Ρ	185	B9	╣	Alt-185
145	91	Σ	Σ	186	BA	║	Alt-186
146	92	Τ	Τ	187	BB	╗	Alt-187
147	93	Υ	Υ	188	BC	╝	Alt-188
148	94	Φ	Φ	189	BD	╜	Alt-189
149	95	Χ	Χ	190	BE	╛	Alt-190
150	96	Ψ	Ψ	191	BF	┐	Alt-191
151	97	Ω	Ω	192	C0	└	Alt-192
152	98	α	α	193	C1	┴	Alt-193
153	99	β	β	194	C2	┬	Alt-194
154	9A	γ	γ	195	C3	├	Alt-195
155	9B	δ	δ	196	C4	─	Alt-196
156	9C	ε	ε	197	C5	┼	Alt-197

Δεκαδικός	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*	*Δεκαδικός*	*Δεκαεξ/κός*	*Χαρακτήρας*	*Πλήκτρο*
198	C6	╞	Alt-198	227	E3	ή	Alt-227
199	C7	╟	Alt-199	228	E4	ϊ	Alt-228
200	C8	╚	Alt-200	229	E5	ί	Alt-229
201	C9	╔	Alt-201	230	E6	ό	Alt-230
202	CA	╩	Alt-202	231	E7	ύ	Alt-231
203	CB	╦	Alt-203	232	E8	ϋ	Alt-232
204	CC	╠	Alt-204	233	E9	ώ	Alt-233
205	CD	═	Alt-205	234	EA	ΰ	Alt-234
206	CE	╬	Alt-206	235	EB	δ	Alt-235
207	CF	╧	Alt-207	236	EC	∞	Alt-236
208	D0	╨	Alt-208	237	ED	ϕ	Alt-237
209	D1	╤	Alt-209	238	EE	∈	Alt-238
210	D2	╥	Alt-210	239	EF	∩	Alt-239
211	D3	╙	Alt-211	240	F0	≡	Alt-240
212	D4	╘	Alt-212	241	F1	±	Alt-241
213	D5	╒	Alt-213	242	F2	≥	Alt-242
214	D6	╓	Alt-214	243	F3	≤	Alt-243
215	D7	╫	Alt-215	244	F4	⌠	Alt-244
216	D8	╪	Alt-216	245	F5	⌡	Alt-245
217	D9	┘	Alt-217	246	F6	÷	Alt-246
218	DA	┌	Alt-218	247	F7	≈	Alt-247
219	DB	█	Alt-219	248	F8	°	Alt-248
220	DC	▄	Alt-220	249	F9	∙	Alt-249
221	DD	▌	Alt-221	250	FA	·	Alt-250
222	DE	▐	Alt-222	251	FB	√	Alt-251
223	DF	▀	Alt-223	252	FC	^{n}	Alt-252
224	E0	ω	ω	253	FD	^{2}	Alt-253
225	E1	ά	Alt-225	254	FE	■	Alt-254
226	E2	έ	Alt-226	255	FF		Alt-255

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. AMMERAAL LEENDERT, *"C for Programmers"*, John Wiley & Sons Ltd, 1986. Και στα ελληνικά: *"Προγραμματίζοντας στη Γλώσσα C"*, Εκδόσεις Γκιούρδα, 1986.
2. KERNIGHAN BRIAN - RITCHIE DENNIS, *"The C Programming Language" (Second Edition)*, Prentice Hall Inc., 1988. Και στα ελληνικά: *"Η Γλώσσα Προγραμματισμού C"*, Κλειδάριθμος, 1990.
3. LAFORE ROBERT, *"Turbo C Programming for PC"*, The Waite Group Inc., 1987. Και στα ελληνικά: *"Turbo C Προγραμματισμός για PC"*, Εκδόσεις Γκιούρδα, 1992.
4. SCHILD HERBERT, *"Turbo C: The Pocket Reference"*, McGraw-Hill, Inc., 1988. Και στα ελληνικά: *"Turbo C: Οδηγός Άμεσης Αναφοράς"*, Κλειδάριθμος, 1990.
5. SCHILD HERBERT, *"Using Turbo C"*, McGraw-Hill, Inc., 1989. Και στα ελληνικά: *"Εγχειρίδιο εκμάθησης Turbo C"*, Κλειδάριθμος, 1989.
6. SCHILD HERBERT, *"Μάθετε εύκολα τη Γλώσσα C"*, Παρατηρητής, 1988.
7. WORTMAN L.A. - SIDEBOTTOM T.O., *"The C Programming Tutor"*, Prentice Hall Inc, London, 1984.
8. ΓΛΑΜΠΕΔΑΚΗΣ Μ. - ΑΤΜΑΤΖΙΔΗΣ Α., *"Turbo C"*, ΙΩΝ, 1994.
9. ΛΙΒΑΔΑΣ ΚΩΝ/ΝΟΣ, *"Εισαγωγή στη Γλώσσα C"*, IF-THEN-ELSE, 1992.



Του *Κώστα Μητρόπουλου*

```
extract_num-
ber_and_incr (destination, source) int
*destination; unsigned char **source; {  extract_num-
ber (destination, *source); *source += 2; } #ifndef EXTRACT_MAC-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \ extract_number_and_incr (&dest, &src) #endif /*
not EXTRACT_MACROS */ #endif /* DEBUG */ /* If DEBUG is defined, Regex prints
many voluminous messages about what it is doing (if the variable `debug' is nonzero).  If
linked with the main program in `iregex.c', you can enter patterns and strings interactively.
And if linked with the main program in `main.c' and the other test files, you can run the al-
ready-written tests.  */ #ifdef DEBUG /* We use standard I/O for debugging.  */ #include <stdio.h>
/* It is useful to test things that ``must'' be true when debugging.  */ #include <assert.h> static int
debug = 0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
DEBUG_PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
(x1, x2, x3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
BUG_PRINT_COMPILED_PATTERN(p, s, e)\ if (debug) print_partial_compiled_pattern (s, e) #define DE-
BUG_PRINT_DOUBLE_STRING(w, s1, sz1, s2, sz2)     \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
extern void printchar(); /* Print the fastmap in human-readable form.  */ void print_fastmap (fastmap)
char *fastmap; { unsigned was_a_range = 0; unsigned i = 0;   while (i < (1 << BYTEWIDTH)) { if (fastmap[i++])
{ was_a_range = 0; printchar (i - 1); while (i < (1 << BYTEWIDTH)  &&  fastmap[i]) { was_a_range = 1; i++; } if
(was_a_range) { printf ("-"); printchar (i - 1); } } } putchar ('\n');  } /* Print a compiled pattern string in hu-
man-readable form, starting at the START pointer into it and ending just before the pointer END.  */ void
print_partial_compiled_pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; un-
signed char *p = start; unsigned char *pend = end; if (start == NULL) { printf ("(null)\n"); return; } /* Loop over
pattern commands.  */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
break; case exactn: mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); printchar (*p++); }
while (--mcnt); break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
*p++); break; case stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
break; case duplicate: printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
break; case charset: case charset_not: { register int c; printf ("/charset%s", (re_opcode_t) *(p -
1) == charset_not ? "_not" : ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
unsigned char map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(map_byte & (1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
line: printf ("/begline"); break; case endline: printf ("/endline"); break; case on_failure_-
jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
break; case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
("/on_failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
tract_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-
be_pop_jump: extract_number_and_incr (&mcnt, &p); printf
("/maybe_pop_jump/0/%d", mcnt); break; case pop_failure_-
jump: extract_number_and_incr (&mcnt, &p); printf ("/pop_-
failure_jump/0/%d", mcnt); break;         case jump_past_alt:
extract_number_and_incr (&mcnt, &p); printf ("/-
```

ISBN: 978-618-00-3016-7