

Η γλώσσα C σε βάθος

5η έκδοση

Πλήρης οδηγός εκμάθησης της γλώσσας C
με εκτενή αναφορά στις δομές δεδομένων

Νίκος Μ. Χατζηγιαννάκης



Περιεχόμενα

Πρόλογος	21
----------------	----

Κεφάλαιο 1: Εισαγωγή στον προγραμματισμό.....	25
--	-----------

Εισαγωγή	27
Πώς να διαβάσετε αυτό το βιβλίο.....	27
Η δομή των κεφαλαίων	28
Γιατί να μάθω προγραμματισμό;	31
Γιατί να μάθω C;.....	31
Πότε να μάθω προγραμματισμό;	32
Μπορούν όλοι να μάθουν προγραμματισμό;.....	32
Η C ως πρώτη γλώσσα προγραμματισμού	32
Ένα επιτυχημένο πείραμα	33
Μια ιστορική αναδρομή στη γλώσσα C	35
Τα χαρακτηριστικά της C	36
Μια δομημένη γλώσσα.....	36
Μια γλώσσα για προγραμματιστές.....	36
Μια μεταφράσιμη γλώσσα	37
Βασικές έννοιες προγραμματισμού	37
Μεταβλητές	38
Σταθερές	40
Παραστάσεις.....	40
Εντολές.....	40
Αναγνωριστικά	41
Σχεδιασμός και ανάπτυξη προγραμμάτων.....	41
Λογικό διάγραμμα	42

Κεφάλαιο 2: Μια πρώτη ματιά στη C.....	55
---	-----------

Εισαγωγή	57
Η δομή ενός προγράμματος στη C	58
Το πρώτο σας πρόγραμμα στη C	60
Ας ξαναγράψουμε το πρώτο μας πρόγραμμα.....	62
Προγράμματα με περισσότερες συναρτήσεις.....	63
Σχόλια προγράμματος.....	64
Δηλώσεις μεταβλητών.....	64

Ανάθεση τιμής σε μεταβλητή	67
Αρχικές τιμές μεταβλητών	67
Μεταβλητές μόνο για ανάγνωση	67
Παραστάσεις	68
Παραστάσεις με μέλη διαφορετικού τύπου	72
Λογικές παραστάσεις	72
Παραδείγματα κώδικα με παραστάσεις	75
Μετατροπή τύπου κατά την ανάθεση τιμής σε μεταβλητή	76
H C και τα άγκιστρά της	77
Σύνθετη πρόταση	77
Είδη προτάσεων	78
H οδηγία #include	79
H οδηγία #define	79

Κεφάλαιο 3: Προετοιμασία για αργότερα..... 89

H C και η μνήμη	91
Ο τελεστής &	92
Ο τελεστής sizeof	92
Τρεις συναρτήσεις παρακαλώ	93
Έξοδος στην οθόνη – Η συνάρτηση printf()	93
Είσοδος από το πληκτρολόγιο – Η συνάρτηση scanf()	96
Τερματισμός προγράμματος – Η συνάρτηση exit()	99
Ολίγη if παρακαλώ	99

Κεφάλαιο 4: Ο τύπος δεδομένων int..... 109

Εισαγωγή	111
Σταθερές τύπου int	111
Μεταβλητές τύπου int	111
Μεταβλητές τύπου int χωρίς πρόσημο	112
Αρχική τιμή μεταβλητής	112
Αριθμητικοί τελεστές	113
Ο τελεστής υπολοίπου %	115
Τελεστές σύντμησης	116
Δυαδικοί αριθμοί	116
Μετατροπή δυαδικού σε δεκαδικό	116
Μετατροπή δεκαδικού σε δυαδικό	117
Οι δυαδικοί τελεστές	118
Χρήση δυαδικών τελεστών για την απομόνωση συγκεκριμένων bit	122
Προτεραιότητα τελεστών	125

Κεφάλαιο 5: Ο τύπος δεδομένων char	135
Εισαγωγή	137
Σταθερές τύπου char	137
Κωδικοποίηση χαρακτήρων σε αριθμούς	137
Χαρακτήρες διαφυγής	138
Κωδικοποίηση Unicode	139
Μεταβλητές τύπου char	140
Μεταβλητές τύπου unsigned char	140
Τρεις συναρτήσεις χειρισμού χαρακτήρων	142
putchar(ch)	142
getch()	142
getchar()	143
Η scanf() και τα μικρά της προβλήματα!	145
Συμβολοσειρές	147
Οι συμβολοσειρές έχουν τιμή;	148
Κεφάλαιο 6: Float, double, και άλλοι τύποι δεδομένων	157
Οι τύποι δεδομένων float και double	159
Σταθερές τύπου float και double	159
Μεταβλητές τύπου float και double	159
Χρήση τελεστών σε δεδομένα τύπου κινητής υποδιαστολής	160
Η χρήση της printf() με δεδομένα κινητής υποδιαστολής	160
Χρήση της συνάρτησης scanf() με δεδομένα τύπου float	161
Οι συναρτήσεις pow() και sqrt()	161
Βασικοί τύποι και παραλλαγές τους	162
Απόλυτος προσδιορισμός τύπου αριθμητικών σταθερών	163
Ο τύπος δεδομένων bool	164
Κεφάλαιο 7: Εντολές συνθήκης	173
Εισαγωγή	175
Η εντολή if σε μια δεύτερη ματιά	175
Η απλή πρόταση if	175
Η πρόταση if-else	177
Η πρόταση if-else if	178
Ένθετες εντολές if	180
Η εντολή switch-case	183
Ο τελεστής συνθήκης ?:	185

Κεφάλαιο 8: Εντολές επανάληψης	199
Εισαγωγή	201
Η εντολή while	202
Η εντολή do-while	204
Η εντολή for	206
Χορεύοντας με τη for!	209
Απλά παραδείγματα.....	209
Ένθετοι βρόχοι for.....	211
Και άλλοι ένθετοι βρόχοι for.....	213
Κλιμακωτή γραφή	214
Η «απαγορευμένη» εντολή goto.....	216
Ο τελεστής «κόμμα» (,).....	218
Η εντολή break	219
Η εντολή continue	221
Επεξεργασία δεδομένων με επαναληπτικές διαδικασίες.....	222
Καταμέτρηση και άθροιση	222
Υπολογισμός μέγιστου και ελάχιστου.....	223
Ανάγνωση χαρακτήρων από το πληκτρολόγιο.....	225
Η χρήσιμη συνάρτηση getch().....	225
Διαχωρισμός ψηφίων ακέραιου αριθμού	226
Κεφάλαιο 9: Συναρτήσεις	245
Εισαγωγή	247
Ορισμός συνάρτησης.....	249
Συναρτήσεις χωρίς παραμέτρους.....	249
Συναρτήσεις με παραμέτρους.....	250
Συναρτήσεις που επιστρέφουν τιμή.....	253
Συναρτήσεις που δεν επιστρέφουν τιμή – Τύπος void	253
Κλήση συνάρτησης	255
Παράμετροι συνάρτησης.....	257
Ορίσματα και μεταβίβαση παραμέτρων.....	257
Χρήση συναρτήσεων βιβλιοθήκης	258
Πρωτότυπα συναρτήσεων	260
Συναρτήσεις χωρίς παραμέτρους, με ρητή δήλωση	261
Ένα ξεκαθάρισμα... ..	262
Μετατροπή τύπου.....	264
Αυτόματη μετατροπή τύπου.....	265

Κεφάλαιο 10: Εμβέλεια μεταβλητών	277
Εισαγωγή	279
Τοπικές μεταβλητές	279
Δήλωση τοπικών μεταβλητών σε σύνθετη πρόταση	282
Δήλωση της μεταβλητής ενός βρόχου <i>for</i> ως τοπικής	282
Καθολικές μεταβλητές	283
Στατικές μεταβλητές	285
 Κεφάλαιο 11: Δείκτες	 297
Εισαγωγή	299
Μεταβλητές δείκτη (Pointer variables)	300
Δήλωση μιας μεταβλητής δείκτη	300
Ένας διαφορετικός τρόπος δήλωσης μιας μεταβλητής δείκτη	302
Οι τελεστές & και *	302
Σύνοψη	306
Αριθμητική δεικτών	306
Λανθασμένη χρήση δεικτών και δείκτες προς άλλες οντότητες	311
Δείκτες τύπου void	312
Δείκτης NULL	313
Εμφάνιση διευθύνσεων και περιεχόμενα δεικτών	313
Δείκτες και συμβολοσειρές	314
Δείκτες σε δείκτες	316
Δείκτες ως παράμετροι συναρτήσεων	320
Συναρτήσεις που επιστρέφουν δείκτη	322
Παραστάσεις αριστερής τιμής (lvalues)	323
 Κεφάλαιο 12: Πίνακες	 337
Εισαγωγή	339
Πίνακες μίας διάστασης (1Δ)	340
Χειρισμός πινάκων μίας διάστασης	342
Πίνακες μίας διάστασης και δείκτες	344
Αρχικές τιμές πίνακα μίας διάστασης	346
Πίνακες δύο διαστάσεων (2Δ)	348
Χειρισμός πινάκων δύο διαστάσεων	349
Αρχικές τιμές πινάκων 2Δ	353
Πίνακες δύο διαστάσεων και δείκτες - πρώτη προσέγγιση	354
Πίνακες με περισσότερες από δύο διαστάσεις	356

Πίνακες χαρακτήρων για την αποθήκευση συμβολοσειρών	360
Χρήση της printf() και της scanf() με πίνακες χαρακτήρων.....	360
Ο τύπος δεδομένων size_t	361
Χειρισμός συμβολοσειρών	362
Πίνακες χαρακτήρων 2Δ για την αποθήκευση συμβολοσειρών	367
Μεταβίβαση πινάκων σε συναρτήσεις	370
Μεταβίβαση πίνακα 1Δ σε συνάρτηση	370
Μεταβίβαση πινάκων πολλών διαστάσεων σε συναρτήσεις.....	374
Εξειδικευμένοι πίνακες.....	382
Πίνακες μεταβλητού μήκους.....	382
Τετραγωνικοί πίνακες.....	383
Πίνακες δεικτών	386
Δείκτες σε πίνακες	387
Αριθμητική δεικτών σε πίνακες	388
Πίνακες δεικτών σε πίνακες	390
Πίνακες πολλών διαστάσεων και δείκτες	391

Κεφάλαιο 13: Προσαρμοσμένοι τύποι δεδομένων 421

Εισαγωγή	423
Δομές	423
Αναφορά στα πεδία μιας δομής.....	428
Απόδοση αρχικών τιμών στα μέλη μιας δομής.....	429
Πίνακες από δομές	430
Ποια είναι η χρησιμότητα των δομών;.....	431
Μεταβίβαση πεδίων δομής σε συνάρτηση	433
Μεταβίβαση ολόκληρης δομής σε συνάρτηση.....	434
Δείκτες σε δομές.....	437
Σύνθετες δομές	441
Πίνακες ως μέλη δομών	442
Χρήση του τελεστή ανάθεσης = με δομές του ίδιου τύπου.....	443
Πεδία εύρους ενός ή περισσότερων bit	443
Ενώσεις.....	447
Απαριθμήσεις	447
Χρήση της δεσμευμένης λέξης typedef.....	450
Χρήση της typedef για πιο σύντομες και ευανάγνωστες δηλώσεις.....	451

Κεφάλαιο 14: Ρεύματα επικοινωνίας και χειρισμός αρχείων..... 465

Εισαγωγή	467
Προκαθορισμένα ρεύματα.....	468
Επίπεδα ρευμάτων	468

Χειρισμός αρχείων.....	469
Αρχεία κειμένου και δυαδικά αρχεία	469
Σειριακή και τυχαία προσπέλαση.....	470
Μνήμη προσωρινής αποθήκευσης	473
Άνοιγμα και κλείσιμο αρχείου.....	474
fopen()	475
fclose().....	478
ferror()	479
clearerr()	479
fflush()	479
fflushall()	479
Αρχεία κειμένου	480
fputc()	480
fgetc()	481
fprintf()	481
fscanf().....	482
feof().....	483
fgets()	485
fputs()	486
Τυχαία προσπέλαση.....	487
fseek().....	487
rewind()	488
ftell().....	488
Δυαδικά αρχεία.....	492
fwrite().....	492
fread()	494
Χρήση δομών για αναπαράσταση εγγραφών	495

Κεφάλαιο 15: Προχωρημένα θέματα..... 517

Αναδρομή	519
Ο μηχανισμός κλήσης συναρτήσεων – χρήση της στοίβας.....	521
Παράμετροι γραμμής εντολών	523
Δείκτες σε συναρτήσεις.....	525
Δήλωση δείκτη προς συνάρτηση.....	526
Ανάθεση τιμής σε μεταβλητή δείκτη προς συνάρτηση	526
Έλεγχος των περιεχομένων ενός δείκτη προς συνάρτηση	528
Δείκτες προς συναρτήσεις ως παράμετροι	528
Πίνακες δεικτών προς συναρτήσεις	528
Εμβόλιμες συναρτήσεις.....	529
Προγράμματα με πολλά πηγαία αρχεία.....	531

Ο μεταγλωττιστής GCC	531
Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων	532
Δημιουργία δικών μας βιβλιοθηκών	534
Ορισμός και δήλωση μεταβλητών και συναρτήσεων.....	536
Συναρτήσεις με μεταβλητό πλήθος ορισμάτων	541
Σύνοψη	545

Κεφάλαιο 16: Αναζήτηση και ταξινόμηση..... 561

Εισαγωγή	563
Σειριακή αναζήτηση	564
Δυαδική αναζήτηση	565
Ταξινόμηση	569
Ταξινόμηση φυσαλίδας	569
Ταξινόμηση επιλογής	572
Ταξινόμηση παρεμβολής	573
Γρήγορη ταξινόμηση	575
Ταξινόμηση πινάκων δύο διαστάσεων	579
Συναρτήσεις βιβλιοθήκης για αναζήτηση και ταξινόμηση	582
Συναρτήσεις σύγκρισης	582
Η συνάρτηση lfind()	587
Η συνάρτηση qsort()	589
Η συνάρτηση bsearch()	591
Χρήση των συναρτήσεων lfind() και qsort() σε πίνακες συμβολοσειρών και δομών	592

Κεφάλαιο 17: Δυναμική διαχείριση μνήμης..... 607

Εισαγωγή	609
Δυναμική κατανομή μνήμης	610
Η συνάρτηση malloc()	611
Η συνάρτηση calloc()	612
Η συνάρτηση free()	613
Η συνάρτηση realloc()	613
Δημιουργία δυναμικών πινάκων	616
Δυναμικοί πίνακες μίας διάστασης	616
Δυναμικοί πίνακες δύο διαστάσεων	617

Κεφάλαιο 18: Δυναμικές δομές δεδομένων 627

Εισαγωγή	629
Πότε χρησιμοποιούνται οι διαφορετικές δυναμικές δομές	631
Η έννοια του κόμβου	632

Συνδεδεμένες λίστες.....	632
Απλά συνδεδεμένη λίστα	632
Κυκλικά συνδεδεμένη λίστα	634
Διπλά συνδεδεμένη λίστα.....	634
Κυκλικά διπλά συνδεδεμένη λίστα	634
Προσθήκη νέου κόμβου σε μια λίστα	634
Υλοποίηση απλά συνδεδεμένης λίστας στη C	637
Δυαδικά δένδρα	656
Δυαδικά δένδρα αναζήτησης (ΔΔΑ)	657
Χειρισμός ενός δυαδικού δένδρου αναζήτησης	659
Προσθήκη νέου κόμβου	659
Αναζήτηση κόμβου	661
Διάσχιση δυαδικού δέντρου αναζήτησης.....	662
Διαγραφή κόμβου από ΔΔΑ.....	664
Υλοποίηση της δομής δυαδικού δένδρου αναζήτησης.....	667
Γράφοι	674
Βασικές έννοιες γράφων.....	674
Αναπαράσταση γράφων με πίνακες γειτνίασης.....	677
Αναπαράσταση γράφων με λίστες γειτνίασης.....	678
Διάσχιση γράφων	682
Πιο δυναμικοί γράφοι.....	695
Προτάσεις για περαιτέρω μελέτη	705
Ισορροπημένα δυαδικά δένδρα αναζήτησης	705
Σταθμισμένοι γράφοι.....	707
Κεφάλαιο 19: Ο προμεταγλωττιστής της C.....	729
Εισαγωγή.....	731
#include	731
#define	732
Μακροεντολές.....	733
Μακροεντολές με περισσότερες προτάσεις.....	734
#undef	735
#if, #else, #elif και #endif.....	735
#ifdef και #ifndef.....	738
#error	739
Αποσφαλμάτωση	740
Η μακροεντολή assert()	741

Κεφάλαιο 20: Από τη C, στη C++	749
Εισαγωγή	751
Η κληρονομιά από τη C	751
Μια αντικειμενοστρεφής γλώσσα	751
Αντικειμενοστρεφής προγραμματισμός	753
Κλάσεις και αντικείμενα	754
Ενθυλάκωση	757
Κληρονομικότητα	757
Πολυμορφισμός	759
Το πρώτο μας πρόγραμμα σε C++	760
Παίζοντας με αντικείμενα	761
Το αντικείμενο cout	761
Το αντικείμενο cin	763
Διαφορές μεταξύ της C και της C++	766
Μερικά από τα νέα διαδικαστικά χαρακτηριστικά της C++	766
Τα αντικειμενοστρεφή χαρακτηριστικά της C++	769
Σκεφτείτε αντικειμενοστρεφώς!	769
 Παράρτημα A: Συναρτήσεις βιβλιοθήκης της C	773
Η καθιερωμένη βιβλιοθήκη της C	775
<ctype.h>	775
<math.h>	776
<stdio.h>	779
<stdlib.h>	785
<string.h>	789
Το ολοκληρωμένο περιβάλλον του Code::Blocks	793
Το ολοκληρωμένο περιβάλλον ανάπτυξης Code::Blocks	795
Εγκατάσταση του Code::Blocks	795
Οι βασικές λειτουργίες του Code::Blocks	796
Παραμετροποίηση του μεταγλωτιστή για χρήση ελληνικών χαρακτήρων	796
Δημιουργία νέου αρχείου πηγαίου κώδικα ή έργου	796
Σύνταξη του κώδικα	797
Αποθήκευση αρχείων	797
Μεταγλώττιση και εκτέλεση του αρχείου πηγαίου κώδικα	798
Μηνύματα μεταγλωτιστή	798
 Αντιστοίχιση ελληνικών και αγγλικών όρων	801
Αντιστοίχιση ελληνικών και αγγλικών όρων	803

Βιβλιογραφία	807
Βιβλιογραφικές πηγές.....	809
Αναφορές στο Διαδίκτυο.....	809
Ευρετήριο	811

ΕΝΔΕΙΚΤΙΚΕΣ
ΣΕΛΙΔΕΣ

Κεφάλαιο

1

Εισαγωγή στον προγραμματισμό

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό ξεκινά με μια πρώτη γνωριμία με το βιβλίο, αναλύει τη δομή των υπόλοιπων κεφαλαίων, και σας προτείνει τρόπους ανάγνωσής του, ανάλογα με το αν διαθέτετε προηγούμενη εμπειρία στον προγραμματισμό.

Περιέχει κάποια ιστορικά στοιχεία για τη γλώσσα C, καθώς και τις απαντήσεις σε κάποιες φιλοσοφικές ερωτήσεις όπως «γιατί να μάθω προγραμματισμό;» ή «γιατί να μάθω C;».

Αναλύει τις βασικές έννοιες του προγραμματισμού, όπως η έννοια της μεταβλητής, και παρουσιάζει τη διαδικασία σχεδιασμού ενός προγράμματος.

Τέλος, σας βοηθά να μάθετε να οργανώνετε τη σκέψη σας και να περιγράφετε τη λειτουργία συστημάτων με έναν εποπτικό τρόπο, χρησιμοποιώντας τα εργαλεία που σας παρέχει ένα λογικό διάγραμμα.

Εισαγωγή

Οκτώβριος του 1983. Ως μεταπτυχιακός φοιτητής, γνώρισα μια δεκαοχτάχρονη αμερικανιδούλα που την έλεγαν BCPL. Η σχέση μας κράτησε έναν χρόνο περίπου. Ευέλικτη αλλά δύστροπη. Δύσκολο να την κατανοήσεις και δυσκολότερο να τη χειριστείς. Την επόμενη χρονιά μου γνώρισε τη μικρότερη αδελφή της, τη C. Από την πρώτη στιγμή με εντυπωσίασε, είχε πιο δομημένη σκέψη, ήταν τρομερά ευέλικτη και πολύ πιο φιλική. Η γνωριμία αυτή δεν άργησε να εξελιχθεί σε έναν μεγάλο έρωτα, ο οποίος κρατάει ακόμη και σήμερα. Από τότε συνάντησα και άλλες πολλές, πολύ πιο νέες και εμφανίσιμες, με πολλές δυνατότητες και προσόντα, όμως πάντα έμεινα πιστός στη μεσήλικα πια αγάπη μου, τη C. Ευτυχώς που είναι μόνο... μια γλώσσα προγραμματισμού!

Και όμως, η πολύχρονη αυτή σχέση είχε και έναν καρπό, την παραγωγή πλούσιου εκπαιδευτικού υλικού που χρησιμοποιήθηκε όλα αυτά τα χρόνια για τη διδασκαλία της γλώσσας C και το οποίο αποτέλεσε τη βάση για τη συγγραφή αυτού του βιβλίου.

Το βιβλίο αυτό απευθύνεται τόσο στον αρχάριο όσο και στον πεπειραμένο προγραμματιστή που θέλει να γνωρίσει τις αρχές και τη φιλοσοφία του δομημένου προγραμματισμού μέσα από μια ευέλικτη και χωρίς όρια γλώσσα όπως η C.

Με ιδιαίτερο τρόπο προσεγγίζονται όλα τα χαρακτηριστικά της γλώσσας και δίνεται έμφαση στην αναλυτική και σε βάθος επεξήγηση των «στρυφνών» της σημείων. Μεγάλο βάρος έχει δοθεί στη διδακτική σειρά αυτού του βιβλίου, ώστε η ανάγνωση και η κατανόηση ενός κεφαλαίου να προϋποθέτει μόνο τις γνώσεις που αποκτήθηκαν στα προηγούμενα κεφάλαια.

Πώς να διαβάσετε αυτό το βιβλίο

Αν είστε γνώστης μιας οποιασδήποτε άλλης γλώσσας προγραμματισμού μπορείτε να παραλείψετε το Κεφάλαιο 1 και να προχωρήσετε κατευθείαν στο επόμενο κεφάλαιο.

Το Κεφάλαιο 2 είναι μια μικρή «περιοδεία» στη γλώσσα. Γίνεται μια πρώτη γνωριμία ώστε να αποκτήσετε μια σφαιρική εικόνα από τη δομή και τα χαρακτηριστικά της και να μπορέσετε να σχεδιάσετε τα πρώτα σας απλά προγράμματα. Τα επόμενα κεφάλαια αναλύουν όλα τα χαρακτηριστικά της γλώσσας, δίνοντας έμφαση στην παρουσίαση βασικών αλλά και πιο πολύπλοκων εννοιών, με απλό, εποπτικό, και κατανοητό τρόπο. Τα Κεφάλαια 15, 16, 17, 18 και 19, περιγράφουν σε βάθος τόσο εξειδικευμένες τεχνικές προγραμματισμού όσο και ιδιαίτερα χαρακτηριστικά της γλώσσας. Τέλος, το Κεφάλαιο 20 σας προετοιμάζει για το επόμενο φυσικό βήμα: Μια πρώτη επαφή με τον αντικειμενοστρεφή προγραμματισμό και τη γλώσσα C++. Το Παράρτημα Α περιέχει μια αναλυτική αναφορά στις πιο συχνά χρησιμοποιούμενες συναρτήσεις της C. Στο Παράρτημα Β θα βρείτε λεπτομέρειες για το περιβάλλον ανάπτυξης **Code::Blocks**¹. Το **Code::Blocks** είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment –

¹ <http://www.codeblocks.org>

IDE) για τις γλώσσες C και C++, το οποίο διατίθεται υπό τη γενική άδεια χρήσης GNU GPL v3.0, πράγμα που σημαίνει ότι διατίθεται δωρεάν στον οποιονδήποτε. Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του **Code::Blocks**. Το Σχήμα 1.1 σας πάει μια μικρή βόλτα στο βιβλίο και περιγράφει συνοπτικά τα περιεχόμενα των κεφαλαίων και των παραρτημάτων του.

Στο **συνοδευτικό υλικό** του βιβλίου θα βρείτε το πρόγραμμα εγκατάστασης για το ολοκληρωμένο περιβάλλον του **Code::Blocks**, τον κώδικα των προγραμμάτων που βρίσκονται στο βιβλίο, τις απαντήσεις όλων των ασκήσεων και των εργασιών. Το συμπιεσμένο αρχείο του συνοδευτικού υλικού μπορείτε να το κατεβάσετε από την ιστοσελίδα www.klidarithmos.gr/CseVathos5e, καθώς και από τον δικτυακό τόπο του βιβλίου στη διεύθυνση <http://c.bytes.gr>.

Σας συνιστώ να κατεβάσετε άμεσα το συνοδευτικό υλικό του βιβλίου και να εγκαταστήσετε το περιβάλλον του **Code::Blocks**, ώστε να μπορείτε να δοκιμάζετε τα παραδείγματα του βιβλίου αλλά και τα δικά σας προγράμματα.

Το σύμβολο ☉ δίπλα από τον τίτλο μιας παραγράφου σημαίνει ότι η παράγραφος παρέχει εξειδικευμένες πληροφορίες τις οποίες μπορείτε να παραλείψετε αν δεν θέλετε ιδιαίτερη εμβάθυνση.

Στο μαύρο πλαίσιο που υπάρχει δίπλα από τα περισσότερα παραδείγματα αυτού του βιβλίου αναφέρεται το όνομα του αρχείου στο οποίο υπάρχει ο πηγαίος κώδικας του παραδείγματος. Αν έχετε ήδη εγκαταστήσει το **Code::Blocks**, αρκεί να διπλοπατήσετε με το ποντίκι στο αρχείο του παραδείγματος για να ανοίξετε το αρχείο μέσα στο ολοκληρωμένο περιβάλλον του **Code::Blocks**. Η μεταγλώττιση και η εκτέλεση του προγράμματος γίνονται πλέον με ένα απλό πάτημα του πλήκτρου του ποντικιού σε ένα εικονίδιο του περιβάλλοντος (δείτε το Παράρτημα Β).

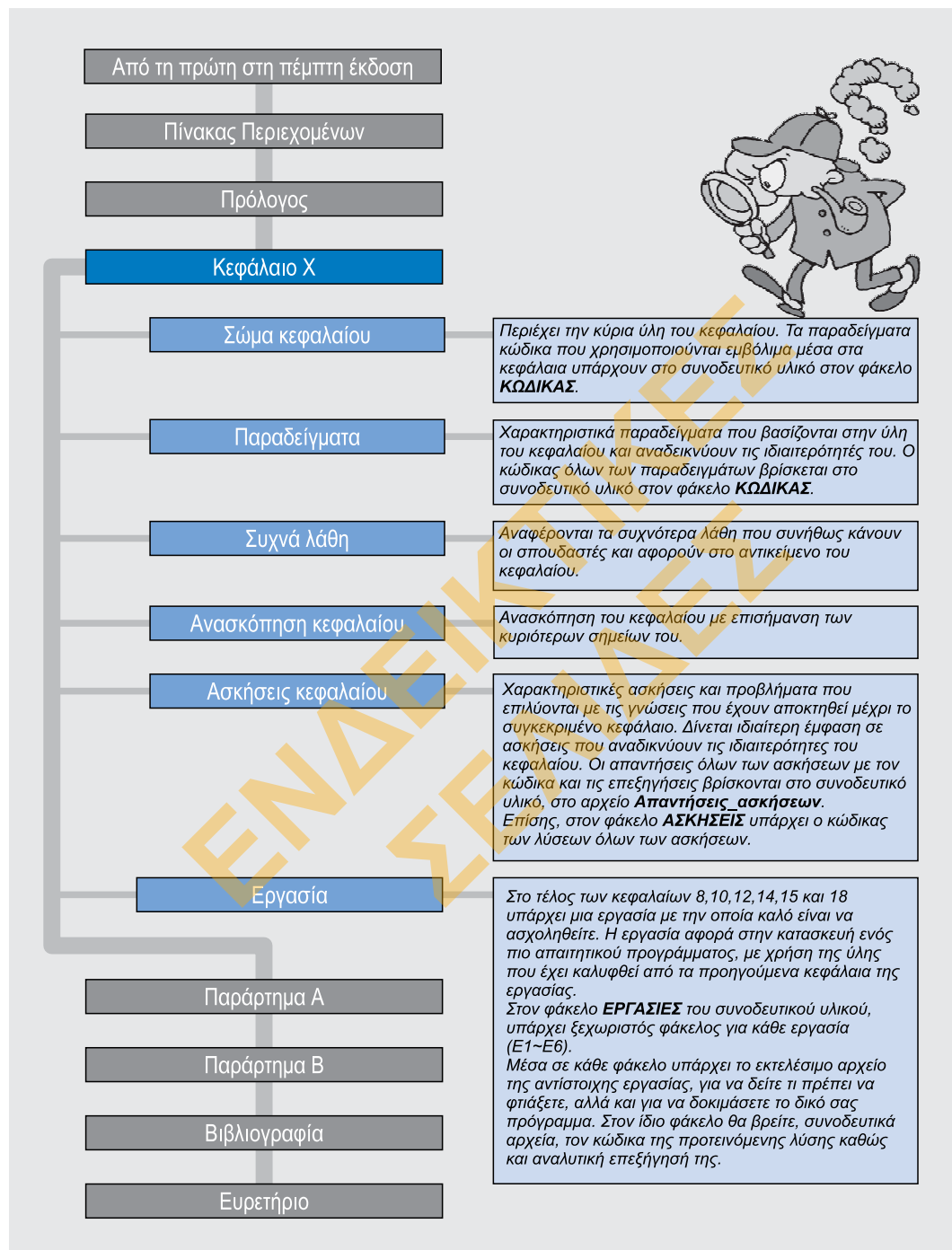
arxeyio.c

Η δομή των κεφαλαίων

Όλα τα κεφάλαια του βιβλίου έχουν την ίδια δομή (Σχήμα 1.2). Στο τέλος κάθε κεφαλαίου υπάρχουν λυμένα παραδείγματα και ασκήσεις προς επίλυση. Ο βαθμός δυσκολίας κάθε άσκησης υποδεικνύεται από έναν αριθμό αστερίσκων (από έναν αστερίσκο για τις εύκολες ασκήσεις μέχρι τρεις για τις πιο απαιτητικές). Σε κάθε κεφάλαιο αναφέρονται τα συχνότερα λάθη που κάνουν οι σπουδαστές, ενώ υπάρχει επίσης μια μικρή ανασκόπηση με τα βασικότερα σημεία του κεφαλαίου. Μετά από ορισμένα κεφάλαια υπάρχουν εργασίες τις οποίες σας προτείνω να προσπαθήσετε να υλοποιήσετε. Για την υλοποίηση κάθε εργασίας ίσως χρειαστεί να αφιερώσετε αρκετές ώρες ή και μέρες. Στο συνοδευτικό υλικό υπάρχουν οι απαντήσεις των ασκήσεων, ο κώδικας των λύσεών τους, καθώς και ο κώδικας όλων των προγραμμάτων που υπάρχουν στο βιβλίο. Επίσης, στο φάκελο ΕΡΓΑΣΙΕΣ θα βρείτε την επεξήγηση και τον κώδικα της λύσης των εργασιών.

1	Εισαγωγή στον προγραμματισμό	Μας εισάγει στα γενικά χαρακτηριστικά της γλώσσας C, στις βασικές έννοιες του προγραμματισμού, καθώς και στην κατανόηση και χρήση λογικών διαγραμμάτων.			
2	Μια πρώτη ματιά στη C	Εδώ θα μάθουμε να δημιουργούμε απλά προγράμματα χρησιμοποιώντας τις βασικές εντολές και συναρτήσεις της γλώσσας.			
3	4	5	6	Τύποι δεδομένων	Στα κεφάλαια αυτά αναλύονται πλήρως οι διαφορετικοί τύποι δεδομένων της C: char, int, float, double και bool.
7	Εντολές συνθήκης	Εδώ γίνεται πλήρης ανάλυση του συντακτικού των εντολών συνθήκης της C. Δηλαδή της εντολής if καθώς και της εντολής switch.			
8	Εντολές επανάληψης	Το κεφάλαιο αυτό ασχολείται με τις εντολές μέσω των οποίων μπορούμε να επιτύχουμε επαναληπτική εκτέλεση προτάσεων.			
9	Συναρτήσεις	Εδώ θα μάθουμε πώς να χωρίζουμε το πρόγραμμά μας σε τμήματα, κάθε ένα από τα οποία υλοποιείται από διαφορετική συνάρτηση.			
10	Εμβέλεια μεταβλητών	Το κεφάλαιο αυτό εξηγεί την εμβέλεια και τον χρόνο ζωής των μεταβλητών και των λοιπών οντοτήτων που ορίζονται σε ένα πρόγραμμα.			
11	Δείκτες	Αναλύεται η χρήση των δεικτών. Απαραίτητη γνώση για την κατανόηση σε βάθος των διάφορων μηχανισμών της γλώσσας.			
12	Πίνακες	Αναλύεται η χρήση πινάκων για την αποθήκευση δεδομένων.			
13	Προσαρμοσμένοι τύποι δεδομένων	Εισάγει και αναλύει τους προσαρμοσμένους τύπους δεδομένων. Τύποι που ορίζονται από τον χρήστη.			
14	Ρεύματα επικοινωνίας	Αναλύει τον τρόπο επικοινωνίας ενός προγράμματος με περιφερειακές συσκευές και ιδιαίτερα με αρχεία.			
15	Προχωρημένα θέματα	Αναφέρεται σε εξειδικευμένα και πιο απαιτητικά θέματα.			
16	Αναζήτηση και ταξινόμηση	Το κεφάλαιο αυτό ασχολείται με την αναζήτηση και ταξινόμηση δεδομένων αποθηκευμένων σε πίνακες. Αναλύονται διάφοροι αλγόριθμοι.			
17	Δυναμική διαχείριση μνήμης	Εδώ θα μάθουμε πώς να δεσμεύουμε, να χρησιμοποιούμε και να αποδεσμεύουμε τμήματα μνήμης με δυναμικό τρόπο.			
18	Δυναμικές δομές δεδομένων	Εξετάζονται αναλυτικά λίστες, δένδρα και γράφοι.			
19	Ο προμεταγλωττιστής της C	Αναλύονται οι περισσότερες οδηγίες του προμεταγλωττιστή της C και παρουσιάζεται η χρήση τους σε προγράμματα.			
20	Από τη C στη C++	Μια πρώτη ματιά στο επόμενο βήμα! Μια αντικειμενοστρεφής γλώσσα προγραμματισμού, ξαδελφάκι της C.			
A	Παράρτημα A	Αναφορά στις πιο χρήσιμες συναρτήσεις βιβλιοθήκης της γλώσσας.			
B	Παράρτημα B	Περιγραφή της εγκατάστασης και της βασικής χρήσης του ολοκληρωμένου περιβάλλοντος ανάπτυξης Code::Blocks			

Σχήμα 1.1 Μια ξενάγηση στο βιβλίο



Σχήμα 1.2 Η δομή των κεφαλαίων

Γιατί να μάθω προγραμματισμό;

Σε μια συνέντευξη του ο Steve Jobs (1955~2011), συνιδρυτής και πρώην πρόεδρος της Apple, ανέφερε: «Πιστεύω ότι ο καθένας θα πρέπει να μάθει να προγραμματίζει έναν υπολογιστή γιατί αυτό σε διδάσκει πώς να σκέφτεσαι». Πέρα όμως από αυτή την αδιαμφισβήτητη αλήθεια, ο προγραμματισμός αποτελεί την πεμπτουσία της επιστήμης των υπολογιστών. Δεν νοείται σπουδαστής, ο οποίος ασχολείται με την πληροφορική σε οποιοδήποτε επίπεδο, που να μην διαθέτει βασικές τουλάχιστον γνώσεις προγραμματισμού. Ακόμα και τα προγράμματα που χρησιμοποιούμε καθημερινά, όπως το Word ή το Excel, καθώς και το κινητό μας τηλέφωνο, μπορούμε να τα παραμετροποιούμε ώστε να αυτοματοποιούμε κάποιες από τις λειτουργίες τους, αρκεί να διαθέτουμε βασικές γνώσεις προγραμματισμού.

Γιατί να μάθω C;

Μια ερώτηση που ακούω συνέχεια από τους φοιτητές είναι η εξής: Ποιος ο λόγος να μάθω C δεδομένου ότι υπάρχουν πιο σύγχρονες γλώσσες οι οποίες είναι πιο φιλικές και έχουν περισσότερες δυνατότητες;

Πάντα απαντώ με ένα παράδειγμα. Αν θέλουμε να γίνουμε πιλότοι, θα αρχίσουμε την εκπαίδευση μας από ένα Airbus ή από ένα Τσέσνα; Το Airbus είναι προφανώς πιο φιλικό, πιο ασφαλές, έχει περισσότερες δυνατότητες, σχεδόν όλες οι λειτουργίες του είναι αυτοματοποιημένες, και σε τελική ανάλυση είναι αυτό που θα πιλοτάρουμε, ως επαγγελματίες πιλότοι, σε μια επιβατική πτήση. Γιατί λοιπόν να αρχίσουμε από ένα Τσέσνα; Νομίζω ότι η απάντηση είναι προφανής. Με το Τσέσνα θα μάθουμε τη βασική φιλοσοφία της πτήσης χωρίς την πολυπλοκότητα των σύνθετων συστημάτων. Θα έχουμε στη διάθεσή μας τα βασικά και απολύτως απαραίτητα όργανα και θα γνωρίσουμε το αποτέλεσμα του κάθε χειρισμού μας. Σε ένα Airbus θα είμαστε περικυκλωμένοι από χιλιάδες διακόπτες και όργανα, από τα οποία όμως ελάχιστα σχετίζονται με την ουσία της πτήσης. Μπορεί να ρυθμίζουν τη θερμοκρασία της καμπίνας, την ένταση των ηχείων ανακοινώσεων, τον φωτισμό, κ.λπ. Επίσης, οι αυτοματοποιημένες λειτουργίες δεν μας βοηθούν να κατανοήσουμε σε βάθος τους κανόνες και τις διαδικασίες της πτήσης.

Ας επανέλθουμε τώρα στον προγραμματισμό. Σίγουρα υπάρχουν γλώσσες πιο σύγχρονες και πιο παραγωγικές, οι οποίες μας βοηθούν να φτιάχνουμε ελκυστικά προγράμματα σε λιγότερο χρόνο. Γιατί λοιπόν να μάθουμε C; Η C μας βοηθάει να κατανοήσουμε σε βάθος τους μηχανισμούς τους οποίους, για λόγους απλότητας και ευχρηστίας, κρύβουν από τον προγραμματιστή οι περισσότερες «παραγωγικές» γλώσσες προγραμματισμού. Ο φοιτητής που σπουδάζει επιστήμες σχετικές με την πληροφορική πρέπει να έχει κατανοήσει σε βάθος όλους αυτούς τους μηχανισμούς. Αν και είναι βέβαιο ότι στην παραγωγική διαδικασία θα χρησιμοποιήσει άλλες γλώσσες, οι γνώσεις που θα αποκτήσει μαθαίνοντας C θα τον βοηθήσουν να επιλύει θέματα και να κατανοεί έννοιες που σε άλλες γλώσσες προγραμματισμού είναι πολύ πιο αφαιρετικές και γενικές.

Ας μην ξεχνάμε, επίσης, ότι οι πλέον σύγχρονες γλώσσες προγραμματισμού, όπως η C++ (C plus-plus), η Java, και η C# (C sharp), βασίζονται στη C! Οι βασικές εντολές των γλωσσών αυτών καθώς και το συντακτικό τους είναι ίδιο με αυτό της C.

Η C θεωρείται η καθιερωμένη γλώσσα στην επιστήμη της Πληροφορικής. Οι περισσότεροι αλγόριθμοι επεξηγούνται με χρήση της γλώσσας C. Επίσης, ο γνώστης της C έχει στη διάθεσή του εκατομμύρια γραμμές έτοιμου κώδικα, είτε για να τον μελετήσει είτε για να τον χρησιμοποιήσει.

Πότε να μάθω προγραμματισμό;

Προγραμματισμό μπορεί και πρέπει να αρχίζει κανείς να μαθαίνει από την παιδική ηλικία. Ο ισχυρισμός ότι είναι απαραίτητη η γνώση μαθηματικών για την εκμάθηση προγραμματισμού είναι ένας μύθος. Αυτό που χρειάζεται μόνο είναι να μπορεί κάποιος να σκέπτεται κάπως δομημένα. Βέβαια, είναι απαραίτητες κάποιες βασικές γνώσεις αριθμητικής ώστε να γίνονται αντιληπτές κάποιες προγραμματιστικές έννοιες και λειτουργίες.

Η αλήθεια είναι ότι κάποια στοιχεία προγραμματισμού μπορούν να διδαχθούν στα παιδιά ακόμα και από την προσχολική ηλικία, μέσα από εκπαιδευτικές γλώσσες όπως η Logo. Όμως, για να μπορέσουν τα παιδιά να χρησιμοποιήσουν μια πραγματική γλώσσα προγραμματισμού θα πρέπει να είναι αρκετά πιο ώριμα. Θεωρώ ότι στην πέμπτη ή στην έκτη Δημοτικού, ανάλογα με την ωριμότητα και τις δυνατότητές του, ένα παιδί μπορεί να αρχίσει να διδάσκεται προγραμματισμό. Αν τώρα κάποιος δεν έμαθε προγραμματισμό από το Δημοτικό ας μην απελπίζεται, αφού μπορεί να αρχίσει σε οποιαδήποτε ηλικία. Το μόνο που χρειάζεται είναι δομημένη σκέψη, καθαρό μυαλό, όρεξη, και χρόνος!

Μπορούν όλοι να μάθουν προγραμματισμό;

Η ερώτηση είναι ανάλογη με την ερώτηση «Μπορούν όλοι να μάθουν να οδηγούν αυτοκίνητο;». Η απάντηση είναι σε γενικές γραμμές ΝΑΙ! Μερικοί όμως δεν θα μάθουν ποτέ ακόμα και αν προσπαθήσουν, η πλειονότητα θα μάθει απλώς να οδηγεί χωρίς ιδιαίτερες δεξιότητες, λίγοι θα γίνουν πραγματικοί δεξιοτέχνες της οδήγησης, και ελάχιστοι με ιδιαίτερο ταλέντο θα γίνουν οδηγοί στη φόρμουλα 1. Η πείρα μου στη διδασκαλία του προγραμματισμού σε φοιτητές με θεωρητικό υπόβαθρο μου έχει δείξει ότι, με την κατάλληλη εκπαιδευτική προσέγγιση, οι περισσότεροι μπορούν να διδαχθούν προγραμματισμό και να αποκτήσουν τις βασικές δεξιότητες.

Η C ως πρώτη γλώσσα προγραμματισμού

Η C είναι αναμφίβολα μια ιδιαίτερη γλώσσα. Δεν μπορώ να τη χαρακτηρίσω εύκολη, ούτε όμως και δύσκολη. Είναι όμως κατάλληλη για να τη διδαχθεί κάποιος ως πρώτη γλώσσα προγραμματισμού; Η άποψη μου είναι ότι ως πρώτη γλώσσα προγραμματισμού θα έπρεπε να διδάσκεται μια απλή γλώσσα μέσω της οποίας ο εκπαιδευόμενος θα έλθει για πρώτη φορά σε επαφή με τις βασικές έννοιες του προγραμματισμού και ιδιαίτερα του

δομημένου προγραμματισμού. Δυστυχώς όμως δεν υπάρχει τέτοια γλώσσα. Οι παραγωγικές γλώσσες δεν είναι απλές και περιλαμβάνουν πολλές δυνατότητες και ιδιοτροπίες, οι οποίες αποσπούν τον αρχάριο σπουδαστή από την ουσία του προγραμματισμού. Η C, παρά τις ιδιοτροπίες της και τη μικρή δυσκολία στα πρώτα στάδια της διδασκαλίας, με την κατάλληλη εκπαιδευτική προσέγγιση μπορεί να διδαχθεί ως πρώτη γλώσσα προγραμματισμού, το οποίο πράγματι συμβαίνει σε αρκετά πανεπιστήμια του κόσμου.

Ένα επιτυχημένο πείραμα

Πριν από τρία περίπου χρόνια μια συνάδελφος και φίλη μου ζήτησε να μιλήσω με τον γιο της, ο οποίος ήθελε απεγνωσμένα να μάθει προγραμματισμό, αλλά ήταν μόλις 12 χρονών και πήγαινε στην έκτη δημοτικού. Από το πρώτο κιόλας λεπτό της συνομιλίας μας διαπίστωσα το έντονο ενδιαφέρον του μικρού για τον προγραμματισμό και εντυπωσιάστηκα από τις γνώσεις του στον τομέα των Η/Υ, τη δομημένη έκφρασή του, και το εμφανές υπόβαθρο που μόνος του είχε αποκτήσει. Βέβαια, όπως όλα τα παιδιά της ηλικίας του, έβλεπε τον προγραμματισμό ως ένα εργαλείο για να φτιάξει τα δικά του παιχνίδια. Αμέσως του εξήγησα ότι, στην περίπτωση που αρχίσουμε μαθήματα προγραμματισμού, για να φτάσει στο επίπεδο να φτιάχνει τα δικά του παιχνίδια θα περάσουν πολλά χρόνια εκπαίδευσης και εμπειρίας!

Το παιδί είχε αναμφισβήτητη πολλή θέληση και από την πρώτη μας επαφή διαισθάνθηκα ότι είχε και ιδιαίτερες ικανότητες. Αμέσως μου δημιουργήθηκε ένα δίλημμα: Με ποια γλώσσα προγραμματισμού έπρεπε να ξεκινήσω. Αρχικά σκέφτηκα τη Visual Basic, που είχε προτείνει και ο ίδιος, με την οποία γρήγορα θα μπορούσε να κάνει αρκετά θεαματικά προγράμματα που κερδίζουν το ενδιαφέρον των παιδιών. Όμως κάποιος που ξεκινάει με τη Visual Basic δύσκολα μετά προσαρμόζεται σε λιγότερο εντυπωσιακές γλώσσες, όπως η C ή η C++, που σε βοηθούν όμως να κατανοήσεις σε βάθος τις τεχνικές προγραμματισμού και τα μυστικά που κρύβονται από πίσω τους. Για να χρησιμοποιήσω το αγαπημένο μου παράδειγμα, είναι σαν κάποιος να μάθει να πιλοτάρει για πρώτη φορά αεροπλάνο σε ένα Airbus, όπου όλα είναι αυτοματοποιημένα, και μετά να τον βάλεις να πετάξει ένα Τσέσνα. Σίγουρα θα απογοητευτεί αμέσως, αλλά το Τσέσνα είναι αυτό που θα πρέπει να πετάξει για να μάθει σε βάθος τις τεχνικές και τα μυστικά της πτήσης.

Έτσι πήρα τη μεγάλη απόφαση: θα δίδασκα στον πιτσιρικά τη γλώσσα C παίρνοντας το ρίσκο να απογοητευτεί και να τα παρατήσει. Ούτε σχήματα, ούτε χρώματα, ούτε εντυπωσιακά παράθυρα, μόνο ένα μαύρο ανιαρό παράθυρο κονσόλας. Του εξήγησα ότι προτιμώ να του μάθω προγραμματισμό σε βάθος παρά κάτι εντυπωσιακό αλλά επιφανειακό! Ότι με αυτό τον τρόπο αυτά που φανταζόταν δεν θα μπορούσαν να πραγματοποιηθούν άμεσα και ότι θα έπρεπε να δείξει θέληση και υπομονή.

Κατά τα διάφορα στάδια της διδασκαλίας πολλές φορές αναγκαζόμουν να του εξηγήσω μαθηματικές έννοιες. Στην ηλικία αυτή δεν γνώριζε ούτε τι είναι η ύψωση σε δύναμη, ούτε τι είναι η τετραγωνική ρίζα!

Αναγνωριστικά

Αναγνωριστικά (identifiers) καλούνται τα ονόματα που χρησιμοποιεί μια γλώσσα για να αναφέρεται στα επιμέρους στοιχεία της. Για παράδειγμα, το όνομα μιας μεταβλητής ή το όνομα μιας συνάρτησης αποτελούν αναγνωριστικά της γλώσσας.

Κάθε γλώσσα έχει τους δικούς της κανόνες για τη σύνταξη των αναγνωριστικών της. Οι κανόνες αυτοί καθορίζουν το πλήθος και το είδος των επιτρεπτών χαρακτήρων. Οι κανόνες για τα αναγνωριστικά της C αναφέρονται στο επόμενο κεφάλαιο.

Σχεδιασμός και ανάπτυξη προγραμμάτων

Είναι προφανές ότι το γράψιμο ενός προγράμματος προϋποθέτει γνώση του αντικειμένου στο οποίο θα αναφέρεται το συγκεκριμένο πρόγραμμα. Για παράδειγμα, αν θέλουμε να φτιάξουμε ένα πρόγραμμα το οποίο θα απευθύνεται σε οδοντίατρους, θα πρέπει καταρχήν να γνωρίζουμε τον τρόπο με τον οποίο δουλεύει ένας οδοντίατρος, τι απαιτήσεις έχει, τι πληροφορίες θέλει να διατηρεί για τους ασθενείς του κ.λπ. Αυτό το στάδιο, κατά το οποίο εμείς ουσιαστικά «μαθαίνουμε» τον τρόπο δουλειάς του οδοντίατρου διακρίνοντας και κατανοώντας τα επιμέρους απλούστερα τμήματα από τα οποία αποτελείται η εργασία του, λέγεται **ανάλυση απαιτήσεων**.

Μετά το στάδιο της ανάλυσης ακολουθεί το στάδιο του **σχεδιασμού**. Στο στάδιο αυτό μπορούμε, αν αυτό κριθεί απαραίτητο, να προτείνουμε νέους καλύτερους τρόπους εργασίας ή ακόμη και νέες δυνατότητες. Η ανάλυση και ο σχεδιασμός δεν έχουν απαραίτητα άμεση σχέση με τον προγραμματισμό, αλλά χρησιμοποιούνται ως αναγκαία εργαλεία για τη σωστή και επαγγελματική κατασκευή προγραμμάτων. Μετά από τον σχεδιασμό ακολουθεί η **ανάπτυξη της εφαρμογής** σε κάποια γλώσσα προγραμματισμού, η **δοκιμή** και, τέλος, η **συντήρηση**.



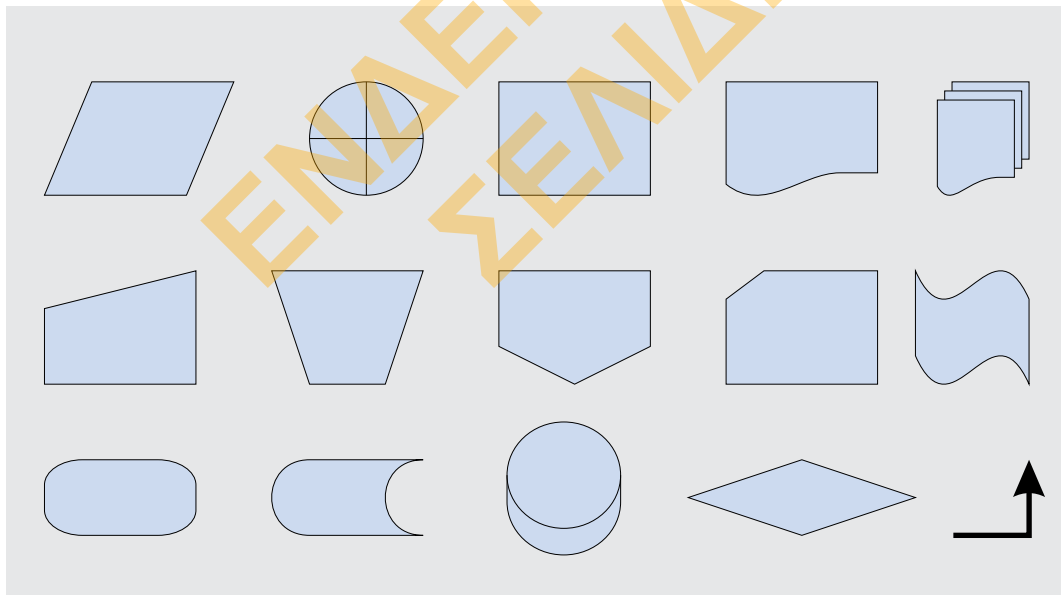
Τα πέντε αυτά στάδια αποτελούν τον κύκλο ζωής ενός προγράμματος, όπως εμφανίζεται στο σχήμα. Παρατηρούμε ότι μετά από το στάδιο της συντήρησης, ο κύκλος μπορεί να ξεκινήσει πάλι από την ανάλυση απαιτήσεων. Αυτό θα συμβεί αν θελήσουμε να κάνουμε αλλαγές ή προσθήκες στην εφαρμογή μας, οπότε ξεκινάει ένας νέος κύκλος ζωής!

Λογικό διάγραμμα

Στα στάδια της ανάλυσης και του σχεδιασμού ο αναλυτής χρησιμοποιεί, εκτός των άλλων εργαλείων του, διαγράμματα για να δώσει μια εικόνα της λειτουργίας του υπό μελέτη συστήματος. Η σχηματική αυτή απεικόνιση ενός συστήματος μπορεί να γίνει με διάφορα εργαλεία και μεθοδολογίες. Ο πιο απλός και κλασικός τρόπος είναι με τη χρήση γεωμετρικών σχημάτων και ονομάζεται λογικό διάγραμμα ή διάγραμμα ροής (flow chart). Το λογικό διάγραμμα αποτέλεσε την πρώτη μέθοδο αναπαράστασης αλγορίθμων. Παρουσιάστηκε για πρώτη φορά από τον Frank Gilbreth το 1921 και αργότερα κυριάρχησε ως μέθοδος αναπαράστασης αλγορίθμων για συστήματα Η/Υ. Ακόμη και οι σύγχρονες τεχνικές αναπαράστασης, όπως τα διαγράμματα ενεργειών της UML³, αποτελούν μια επέκταση των κλασικών λογικών διαγραμμάτων.

Η κατασκευή ενός λογικού διαγράμματος βασίζεται στη χρήση συγκεκριμένων γεωμετρικών σχημάτων. Υπάρχουν δεκάδες διαφορετικά σχήματα που μπορούν να χρησιμοποιηθούν σε ένα λογικό διάγραμμα, ανάλογα με τη πολυπλοκότητα και τη χρήση του (Σχήμα 1.3). Όμως, όπως θα δούμε πολύ σύντομα, σε ένα απλό τυπικό λογικό διάγραμμα τα διαφορετικά σχήματα που χρησιμοποιούνται είναι πολύ περιορισμένα.

Σε αυτό το βιβλίο χρησιμοποιείται συχνά το λογικό διάγραμμα για την επεξήγηση της λειτουργίας μιας εντολής, μιας διαδικασίας, ή ενός προγράμματος.



Σχήμα 1.3 Μερικά από τα σχήματα που χρησιμοποιούνται στα λογικά διαγράμματα

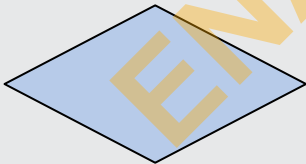
³ Η UML (Unified Modeling Language – Ενοποιημένη Γλώσσα Μοντελοποίησης) είναι μια σύγχρονη μέθοδος γραφικής αναπαράστασης συστημάτων, ιδιαίτερα χρήσιμη στον αντικειμενοστρεφή προγραμματισμό. Η ανάπτυξη της UML ξεκίνησε το 1994 και εξελίσσεται συνεχώς.

Σε ένα απλό λογικό διάγραμμα χρησιμοποιούνται καθορισμένα γεωμετρικά σχήματα που το καθένα υποδεικνύει μια συγκεκριμένη λειτουργία. Κάθε σχήμα αντιστοιχεί σε μία κατηγορία διαδικασιών του υπό μελέτη συστήματος. Για παράδειγμα, ο ρόμβος υποδεικνύει **έλεγχο** ενώ το ορθογώνιο παραλληλόγραμμο **επεξεργασία**. Στο Σχήμα 1.4 αναφέρονται τα σχήματα που συνήθως χρησιμοποιούνται σε ένα απλό λογικό διάγραμμα.

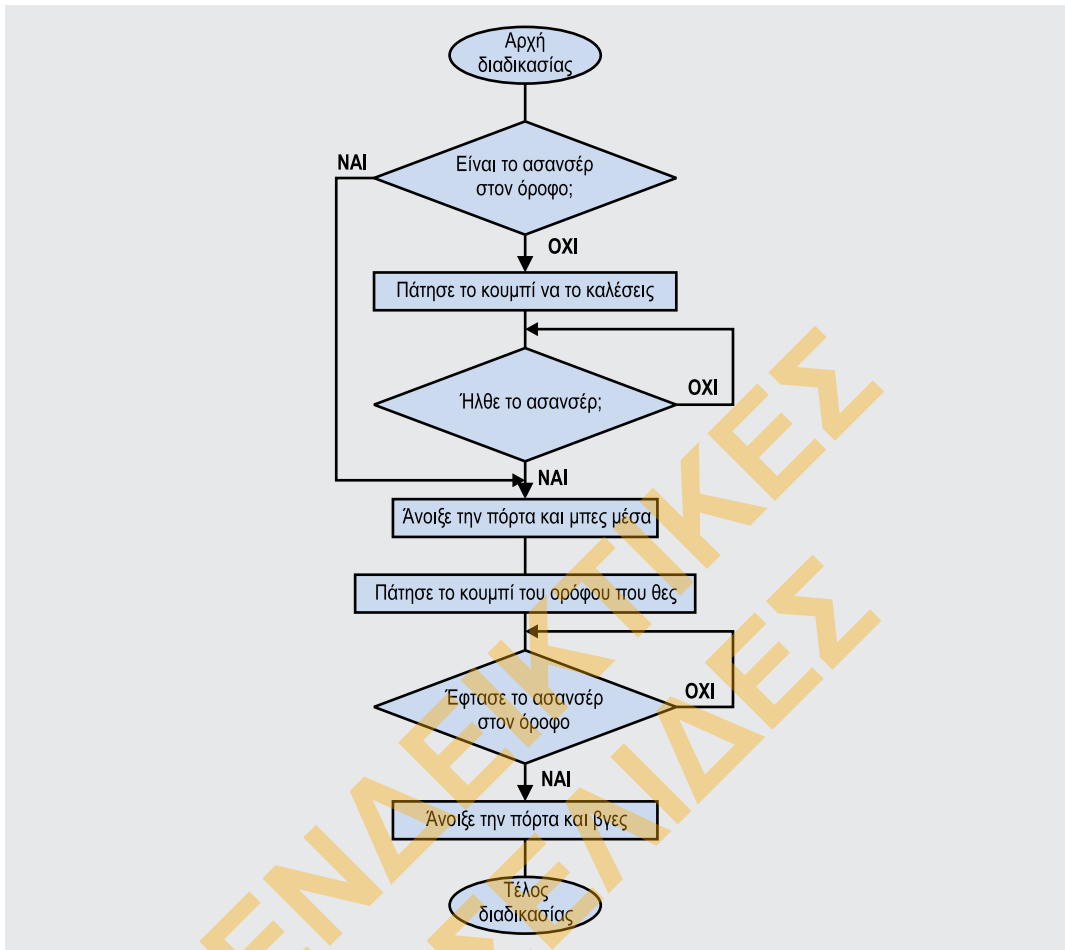
Μέσα σε κάθε γεωμετρικό σχήμα περιγράφεται με απλά λόγια η διαδικασία που εκτελείται.

Στη διαδικασία του ελέγχου είναι δυνατές μόνο δύο αποφάσεις: **Ναι** ή **Όχι**. Ανάλογα με το αποτέλεσμα του ελέγχου, το διάγραμμα ροής μάς οδηγεί σε μια διαφορετική διαδρομή. Μέσα στον ρόμβο περιγράφεται το είδος του ελέγχου που εκτελείται.



Γεωμετρικό σχήμα	Λειτουργία
	Αρχή/Τέλος διαδικασίας
	Επεξεργασία
	Έλεγχος
	Είσοδος/Εξοδος πληροφοριών
	Μεταφορά ελέγχου

Σχήμα 1.4 Τα βασικά σχήματα ενός απλού λογικού διαγράμματος



Σχήμα 1.5 Το λογικό διάγραμμα της διαδικασίας χρήσης ενός ασανσέρ

Το λογικό διάγραμμα του Σχήματος 1.5 δείχνει τη διαδικασία που ακολουθείται όταν κάποιος καλεί το ασανσέρ μέχρι να μεταφερθεί στον όροφο που θέλει. Αναμφίβολα υπάρχουν πολλές εκδοχές για το σύστημα αυτό.

Το λογικό διάγραμμα ενός προβλήματος δεν καθορίζεται μονοσήμαντα. Μπορεί να είναι από εντελώς επιγραμματικό έως πολύ αναλυτικό, ανάλογα με τη χρήση για την οποία προορίζεται.

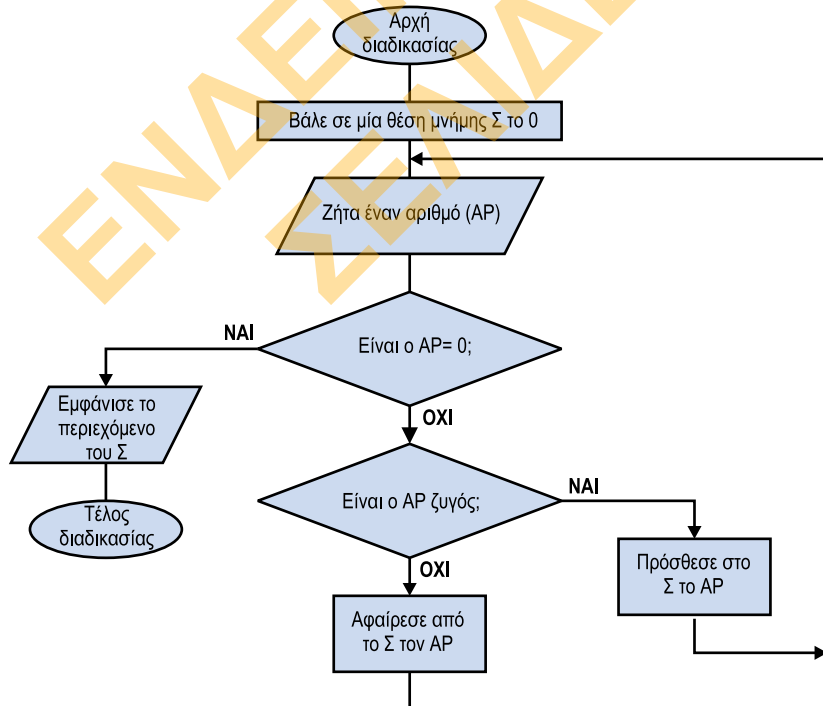
 Το λογικό διάγραμμα είναι ανεξάρτητο από την έννοια του προγραμματισμού και, πολύ περισσότερο, ανεξάρτητο από κάποια συγκεκριμένη γλώσσα προγραμματισμού. Το χρησιμοποιούμε συχνά για να απεικονίζουμε τη λογική ενός προγράμματος πριν ξεκινήσουμε την ανάπτυξή του.

Ανασκόπηση Κεφαλαίου 1

- Η C είναι μια μεταγλωττιζόμενη γλώσσα, τα χαρακτηριστικά της οποίας ευνοούν τον δομημένο και τμηματικό προγραμματισμό.
- Η C είναι μια λιτή γλώσσα, της οποίας οι περισσότερες λειτουργίες βασίζονται στις συναρτήσεις βιβλιοθήκης που τη συνοδεύουν.
- Τα διαφορετικά είδη δεδομένων που μπορεί να χειριστεί μια γλώσσα λέγονται τύποι δεδομένων.
- Η C υποστηρίζει τέσσερις βασικούς τύπους δεδομένων: `int`, `char`, `float` και `double`. Τα τελευταία πρότυπα της γλώσσας υποστηρίζουν και τον λογικό τύπο δεδομένων, ο οποίος συμβολίζεται με `bool`.
- Μια μεταβλητή είναι το όνομα μιας θέσης μνήμης, της οποίας η τιμή μπορεί να μεταβάλλεται.
- Το λογικό διάγραμμα είναι ένας εποπτικός τρόπος αναπαράστασης μιας διαδικασίας ή ενός προγράμματος που συμβάλλει στον σχεδιασμό και στην κατανόησή του.
- Στο λογικό διάγραμμα χρησιμοποιούνται συγκεκριμένα, αλλά διαφορετικά, γεωμετρικά σχήματα για την απεικόνιση των διαδικασιών, ανάλογα με το είδος τους.
- Το λογικό διάγραμμα είναι ανεξάρτητο από οποιαδήποτε γλώσσα προγραμματισμού.
- Μέσα σε ένα λογικό διάγραμμα μπορεί να χρησιμοποιηθεί η έννοια της μεταβλητής ως αποθηκευτικού χώρου.
- Ο κύκλος ζωής ενός προγράμματος ξεκινά από την ανάλυση των απαιτήσεων, ακολουθεί ο σχεδιασμός, η κατασκευή του σε κάποια γλώσσα προγραμματισμού (ανάπτυξη εφαρμογής), ο έλεγχος και, τέλος, η συντήρησή του.
- Η UML χρησιμοποιείται για την απεικόνιση μεγάλων σύνθετων συστημάτων, όπου δεν ενδείκνυται η χρήση λογικών διαγραμμάτων.

Ασκήσεις Κεφαλαίου 1

- 1.1** Να σχεδιάστε ένα λογικό διάγραμμα το οποίο να απεικονίζει τη διαδικασία λύσης μιας εξίσωσης δεύτερου βαθμού. ★ ★
- 1.2** Αν γνωρίζουμε ότι α) δίσεκτο είναι ένα έτος όταν διαιρείται ακριβώς με το 4, β) τα έτη που διαιρούνται ακριβώς με το 100 δεν είναι δίσεκτα εκτός αν διαιρούνται με το 400, σχεδιάστε το λογικό διάγραμμα μιας διαδικασίας η οποία να ζητά το έτος και να απαντά αν αυτό είναι δίσεκτο ή όχι. ★ ★
- 1.3** Αν γνωρίζουμε ότι ένας φορολογούμενος δεν πληρώνει φόρο στην περίπτωση που έχει ετήσιο εισόδημα κάτω από 7000€, πληρώνει 10% στην περίπτωση που έχει εισόδημα από 7000€ έως 15000€, και πληρώνει 20% για μεγαλύτερο εισόδημα, σχεδιάστε το λογικό διάγραμμα μιας διαδικασίας η οποία να ζητά το εισόδημα και να υπολογίζει τον φόρο. ★ ★
- 1.4** Μελετήστε το παρακάτω λογικό διάγραμμα. Τι αποτέλεσμα θα έχει η διαδικασία που περιγράφεται, αν δώσουμε με τη σειρά τους αριθμούς 12, 3, 10, 7, 1, 4, και 0. ★ ★ ★



Κεφάλαιο

2

Μια πρώτη ματιά
στη C

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό σας εισάγει στα πολύ βασικά στοιχεία της γλώσσας και στις έννοιες που πρέπει να γνωρίζετε για να δημιουργήσετε τα πρώτα σας απλά προγράμματα.

Παρουσιάζει τη δομή ενός προγράμματος της C και τη βασική χρήση των απαραίτητων συναρτήσεων βιβλιοθήκης, που συνήθως χρησιμοποιούνται σε ένα απλό πρόγραμμα.

Αναφέρονται οι τύποι δεδομένων που υποστηρίζει η C και αναλύεται η χρήση μεταβλητών και παραστάσεων.

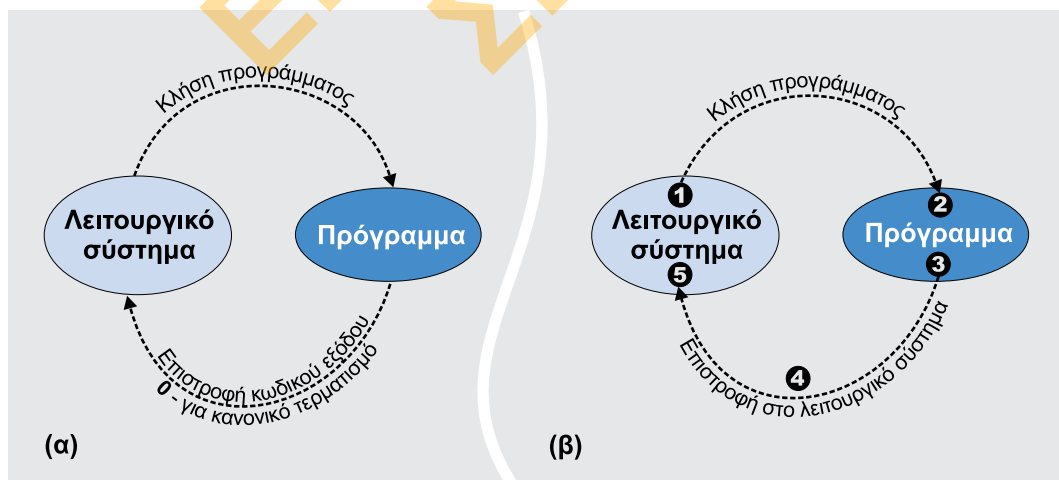
Γίνεται επίσης αναφορά στον τρόπο με τον οποίο χρησιμοποιείται η καθιερωμένη βιβλιοθήκη της γλώσσας, καθώς και στον τρόπο ορισμού σταθερών στα προγράμματά σας.

Εισαγωγή

Στο κεφάλαιο αυτό ξεκινά μια μικρή «περιοδεία» στη γλώσσα C. Γίνεται μια πρώτη γνωριμία, ώστε ο αναγνώστης να αποκτήσει μια σφαιρική εικόνα της δομής και των χαρακτηριστικών της γλώσσας, αλλά και να μπορέσει να κατασκευάσει μερικά απλά προγράμματα. Τα περισσότερα από όσα αναφέρονται στο παρόν κεφάλαιο εξετάζονται πάλι πιο αναλυτικά στα επόμενα κεφάλαια.

Πριν προχωρήσουμε όμως σε αυτή την πρώτη γνωριμία, ας δούμε τη διαδικασία με την οποία εκτελείται ένα οποιοδήποτε πρόγραμμα στον Η/Υ μας. Η διαδικασία αυτή θα μας βοηθήσει να κατανοήσουμε κάποιες έννοιες που θα συναντήσουμε αμέσως μετά.

Γνωρίζουμε ότι οποιοδήποτε σύστημα Η/Υ βρίσκεται υπό τον έλεγχο του λειτουργικού του συστήματος (π.χ. Windows, Linux, MacOS, κ.λπ.). Η εκτέλεση ενός προγράμματος συνήθως ξεκινάει από μια ενέργεια του χρήστη (π.χ. με ένα πάτημα του πλήκτρου του ποντικιού), και μετά το λειτουργικό σύστημα «φορτώνει» το πρόγραμμα στη μνήμη του Η/Υ και μας βάζει στο περιβάλλον του. Με τον τερματισμό του προγράμματος επιστρέφουμε στο περιβάλλον του λειτουργικού συστήματος (Σχήμα 2.1α). Συνήθως, το πρόγραμμα μετά τον τερματισμό του επιστρέφει στο λειτουργικό σύστημα έναν κωδικό (κωδικός εξόδου), ο οποίος είναι απλώς ένας ακέραιος αριθμός. Ο **κωδικός εξόδου** δείχνει αν το πρόγραμμα τερματίστηκε κανονικά ή παρουσίασε κάποιο πρόβλημα. Ο κωδικός **0** σημαίνει ότι το πρόγραμμα τερματίστηκε κανονικά. Οποιοσδήποτε άλλος αριθμός υποδηλώνει πρόβλημα και ανορθόδοξο τερματισμό του προγράμματος! Αυτός ο κωδικός αργότερα ελέγχεται από το λειτουργικό σύστημα, το οποίο έτσι γνωρίζει αν το πρόγραμμα εκτελέστηκε κανονικά. Στην περίπτωση προβλήματος (δηλαδή ο κωδικός εξόδου είναι διαφορετικός από μηδέν), το λειτουργικό σύστημα μπορεί να εκτελέσει κάποια ενέργεια (π.χ. να εμφανίσει κάποιο μήνυμα).



Σχήμα 2.1 Η διαδικασία εκτέλεσης ενός προγράμματος από το λειτουργικό σύστημα

Αναλυτικά, τα στάδια της παραπάνω διαδικασίας είναι τα ακόλουθα πέντε (Σχήμα 2.1β):

1. Κάποια ενέργεια του χρήστη ενεργοποιεί την εκτέλεση ενός προγράμματος (π.χ. διπλοπάτημα με το ποντίκι σε κάποιο εικονίδιο).
2. Το λειτουργικό σύστημα εκτελεί το πρόγραμμα και μας βάζει στο περιβάλλον του.
3. Κάποια ενέργεια του χρήστη έχει ως αποτέλεσμα τον τερματισμό του προγράμματος (π.χ. η επιλογή της εντολής Έξοδος από το μενού Αρχείο).
4. Το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα τον κωδικό εξόδου, με τη λογική που αναφέρθηκε προηγουμένως.
5. Το λειτουργικό σύστημα ελέγχει τον κωδικό που επέστρεψε το πρόγραμμα και αντιδρά ανάλογα (π.χ. εμφανίζει ένα μήνυμα λάθους).

Ο **κωδικός εξόδου** είναι ένας ακέραιος αριθμός τον οποίο επιστρέφει ένα πρόγραμμα στο λειτουργικό σύστημα, μετά από τον τερματισμό του. Ο κωδικός εξόδου καθορίζεται από τον προγραμματιστή. Αυτός αποφασίζει τους κωδικούς εξόδου που θα επιστρέφει το πρόγραμμά του, έχοντας πάντα υπόψη ότι σε περίπτωση ομαλού τερματισμού το πρόγραμμα θα πρέπει να επιστρέφει κωδικό εξόδου 0.

Η δομή ενός προγράμματος στη C

Σκεφτείτε τον τρόπο με τον οποίο λειτουργεί μια εταιρεία. Υπάρχει ο διευθυντής και οι υπάλληλοι που κάνουν συγκεκριμένες δουλειές. Όσο μεγαλώνει μια εταιρεία, τόσο περισσότερους υπαλλήλους χρειάζεται. Αν μια εταιρεία είναι πολύ μικρή, τότε ο μοναδικός που εργάζεται είναι ο διευθυντής ο οποίος κάνει όλες τις δουλειές (one man show). Για να κάνει όμως μια εργασία κάποιος υπάλληλος, θα πρέπει να του την αναθέσει ο διευθυντής ή κάποιος άλλος υπάλληλος (π.χ. ο προϊστάμενός του). Στο παράδειγμα του Σχήματος 2.2α, ο διευθυντής αναθέτει μια εργασία στον υπάλληλο-1 και στον υπάλληλο-2, ενώ ο υπάλληλος-1 αναθέτει την εργασία στον υπάλληλο-3. Τα πάντα όμως ξεκινούν από τις εντολές του διευθυντή! Ένας υπάλληλος μπορεί να κάνει τη δουλειά του χωρίς να επιστρέφει κάτι σε αυτόν που του την ανέθεσε, αλλά μπορεί να του επιστρέψει κάποια αναφορά. Για παράδειγμα, αν ανατεθεί σε έναν υπάλληλο να ταχυδρομήσει κάποιες επιστολές, δεν χρειάζεται να επιστρέψει κάποια αναφορά σε αυτόν που του ανέθεσε την εργασία. Αν όμως του ανατεθεί να ελέγξει τα διαθέσιμα χρήματα που υπάρχουν στο ταμείο, σίγουρα τότε θα πρέπει να επιστρέψει μια αναφορά με το ποσό που εντόπισε.

Όπως έχει ήδη αναφερθεί, ένα πρόγραμμα στη C χωρίζεται συνήθως σε τμήματα (στο παράδειγμά μας, υπαλλήλους), καθένα από τα οποία εκτελεί μια συγκεκριμένη εργασία. Τα τμήματα αυτά λέγονται **συναρτήσεις** και η κάθε συνάρτηση διαθέτει τον δικό της σύνολο εντολών. Η «**συνάρτηση-διευθυντής**» έχει ένα συγκεκριμένο όνομα, **main()**. Για να εκτελεστεί μια συνάρτηση και να διεκπεραιώσει την εργασία για την οποία φτιάχτηκε, πρέπει να κληθεί είτε από τη **main()**, είτε από κάποια άλλη συνάρτηση. Στο παράδειγμα του Σχήματος 2.2β, η **main()** καλεί τη συνάρτηση-1 και τη συνάρτηση-2, ενώ

η συνάρτηση-1 με τη σειρά της καλεί τη συνάρτηση-3. Τα πάντα όμως ξεκινούν από τη συνάρτηση `main()`! Αν η συνάρτηση `main()` δεν καλέσει καμία άλλη συνάρτηση, τότε ο μοναδικός κώδικας που εκτελείται είναι αυτός που βρίσκεται μέσα στη `main()`.

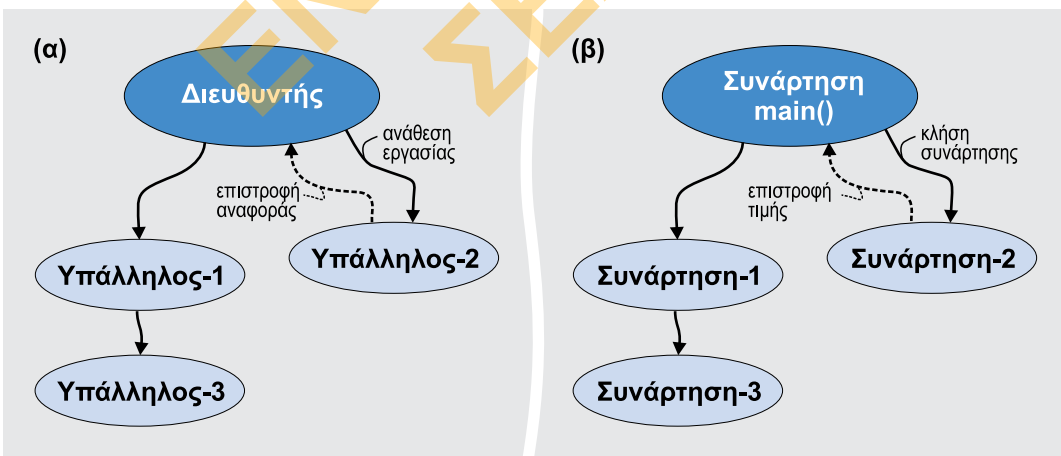
Τα πολύ μικρά προγράμματα, τα οποία δεν χρειάζεται να χωριστούν σε τμήματα, διαθέτουν μόνο μία συνάρτηση, τη `main()`. Κάθε πρόγραμμα στη C διαθέτει **υποχρεωτικά μία** συνάρτηση με όνομα `main()`.

Η συνάρτηση `main()` είναι αυτή που εκτελείται **πρώτη**. Κάθε πρόγραμμα της C πρέπει να έχει **μία και μόνο μία** συνάρτηση `main()`. Η συνάρτηση `main()`, όπως και κάθε συνάρτηση της C, έχει την παρακάτω απλή μορφή:

```
main()
{
    .....
    προτάσεις;
    .....
}
```

Δεν πρέπει να ξεχνάμε το ερωτηματικό στο τέλος κάθε πρότασης.

Μετά από το όνομα της συνάρτησης, ακολουθούν μέσα σε άγκιστρα ({}) οι προτάσεις, δηλαδή ο κώδικας της συνάρτησης. Κάθε πρόταση της `main()` τερματίζεται με **ερωτηματικό** (;). Την ολοκληρωμένη μορφή της `main()` θα τη συναντήσουμε αμέσως μετά. Όπως θα δούμε αργότερα, στην περίπτωση που το πρόγραμμά μας αποτελείται από περισσότερες συναρτήσεις, τότε αυτές συνήθως γράφονται μετά από τη συνάρτηση `main()`.



Σχήμα 2.2 Η δομή ενός προγράμματος στη C

Εκτός από τις συναρτήσεις που γράφουμε εμείς οι ίδιοι (δηλαδή τα ξεχωριστά τμήματα ενός προγράμματος), υπάρχουν και **έτοιμες συναρτήσεις** που διαθέτει η ίδια η γλώσσα, τις οποίες μπορούμε να χρησιμοποιούμε στα προγράμματά μας. Αυτές οι έτοιμες συναρτήσεις αποκαλούνται **συναρτήσεις βιβλιοθήκης**.

Οι συναρτήσεις βιβλιοθήκης εκτελούν διάφορες εργασίες. Υπάρχουν συναρτήσεις για την έξοδο πληροφοριών στην οθόνη, για την είσοδο δεδομένων από το πληκτρολόγιο, για τον υπολογισμό μαθηματικών εννοιών, κ.λπ.

Η **καθιερωμένη βιβλιοθήκη της C** (C standard library) περιλαμβάνει εκατοντάδες συναρτήσεις και αποτελεί αναπόσπαστο μέρος της γλώσσας.

Το πρώτο σας πρόγραμμα στη C

Έφτασε η ώρα να γράψετε το πρώτο σας πρόγραμμα στη C, το οποίο δεν θα μπορούσε να είναι άλλο από το κλασικό "Hello world".

```
#include <stdio.h>
main()
{
    printf("Hello world");
}
```

Το πρόγραμμα αποτελείται μόνο από μία συνάρτηση, τη **main()**, και εμφανίζει στην οθόνη τη φράση **Hello world**.

Στο παραπάνω πρόγραμμα, παρατηρούμε ότι η συνάρτηση **main()** διαθέτει μόνο μία πρόταση που καλεί τη συνάρτηση βιβλιοθήκης **printf()**, η οποία χρησιμοποιείται όταν θέλουμε να εμφανίσουμε πληροφορίες στην οθόνη.

Παρακάτω επεξηγούνται μία-μία οι γραμμές του προγράμματος.

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

Η πρόταση αυτή αναγκάζει τον μεταγλωττιστή της C να συμπεριλάβει το αρχείο **stdio.h** κατά τη διαδικασία της μεταγλώττισης. Το αρχείο αυτό περιέχει τις δηλώσεις των συναρτήσεων βιβλιοθήκης της C για είσοδο/έξοδο. Για να μπορέσουμε να χρησιμοποιήσουμε οποιαδήποτε από τις συναρτήσεις εισόδου/εξόδου στο πρόγραμμά μας (π.χ. τη συνάρτηση **printf()** που ακολουθεί), θα πρέπει οπωσδήποτε⁴ να συμπεριληφθεί (με **#include**) το συγκεκριμένο αρχείο. Τα αρχεία αυτού του τύπου (με προέκταση **.h**) λέγονται **αρχεία κεφαλίδων**. Η χρήση της οδηγίας **#include** επεξηγείται αργότερα στο παρόν κεφάλαιο, και αναλύεται πλήρως στα Κεφάλαια 9, 15, και 19.

⁴ Αρκετά ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE) συμπεριλαμβάνουν κατά τη διαδικασία της μεταγλώττισης συνηθισμένα αρχεία κεφαλίδων ακόμα και αν δεν έχουν οριστεί ρητά στον κώδικα του προγράμματος. Αυτό επιτρέπει τη χρήση μερικών επιπλέον συναρτήσεων, ακόμη και αν ο προγραμματιστής δεν έχει συμπεριλάβει το αρχείο κεφαλίδας στο οποίο δηλώνονται.

main()

Το όνομα της μοναδικής συνάρτησης του προγράμματος. Σας υπενθυμίζω ότι η συνάρτηση **main()** είναι υποχρεωτική σε κάθε πρόγραμμα της C. Στην περίπτωση που το πρόγραμμά μας έχει μόνο μία συνάρτηση, αυτή πρέπει να είναι η **main()**. Αν το πρόγραμμά μας περιέχει περισσότερες από μία συναρτήσεις, η **main()** είναι η πρώτη που εκτελείται.

{

Το αριστερό άγκιστρο σηματοδοτεί την αρχή των προτάσεων της συνάρτησης **main()**.

printf("Hello world");

Η συνάρτηση **printf()** είναι συνάρτηση βιβλιοθήκης της C, και χρησιμοποιείται για την έξοδο πληροφοριών στην οθόνη. Στη συγκεκριμένη περίπτωση θα εμφανίσει το κείμενο "Hello world".

}

Το δεξιό άγκιστρο σηματοδοτεί το τέλος των προτάσεων της συνάρτησης **main()**.

Στη συνέχεια του βιβλίου, σε μερικά παραδείγματα κώδικα χάριν συντομίας μπορεί να παραλείπεται η πρόταση **#include <stdio.h>**, και να εμφανίζεται μόνο η συνάρτηση **main()**. Αν θέλουμε να χρησιμοποιήσουμε τον κώδικα στην πράξη, τότε η γραμμή αυτή πρέπει να προστεθεί πριν από τη **main()**.

Προσοχή: τα κενά διαστήματα, οι χαρακτήρες στηλοθέτη (tab) και οι αλλαγές γραμμής δεν λαμβάνονται υπόψη από τον μεταγλωττιστή της C. Έτσι, το παραπάνω πρόγραμμα θα μπορούσε, χωρίς κανένα πρόβλημα, να γραφεί ως εξής:

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

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

Παρά το γεγονός ότι η C μας επιτρέπει να γράφουμε τον κώδικα μας με έναν ελεύθερο τρόπο, χωρίς να υπολογίζει τα κενά διαστήματα, τους χαρακτήρες στηλοθέτη (tab) και τις αλλαγές γραμμών, εμείς θα πρέπει να υιοθετήσουμε **κανόνες γραφής** που κάνουν τα προγράμματά μας πιο **ευανάγνωστα** και κατανοητά. Στο βιβλίο αυτό ακολουθούνται αυστηρά κάποιοι κανόνες γραφής, στους οποίους θα αναφερθούμε αργότερα. Θα σας συμβούλευα να ακολουθείτε και εσείς τους ίδιους κανόνες!

Ας ξαναγράψουμε το πρώτο μας πρόγραμμα

Ας θυμηθούμε λίγο την εικόνα του Σχήματος 2.1 και τον τρόπο κλήσης ενός προγράμματος από το λειτουργικό σύστημα του Η/Υ μας. Έχουμε ήδη πει ότι το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα έναν κωδικό εξόδου, ο οποίος είναι ένας ακέραιος αριθμός. Στην περίπτωση κανονικής λειτουργίας του προγράμματος, ο κωδικός αυτός πρέπει να είναι ο αριθμός 0. Η συνάρτηση η οποία αποτελεί το σημείο εισόδου ενός προγράμματος της C είναι η συνάρτηση `main()`. Όταν τερματιστεί η εκτέλεση της `main()`, τερματίζεται και το πρόγραμμά μας. Επομένως, την ευθύνη της επιστροφής του κωδικού εξόδου την έχει η συνάρτηση `main()`.

Συνοψίζοντας, έχουμε τα εξής: α) Η συνάρτηση `main()` πρέπει να επιστρέφει έναν ακέραιο αριθμό (τον κωδικό εξόδου) και β) στην περίπτωση κανονικής εκτέλεσης του προγράμματος, ο αριθμός αυτός πρέπει να είναι το 0.

Για να είμαστε λοιπόν και τυπικά σωστοί (σύμφωνα με όσα εισήγαγε το πρότυπο C99), θα πρέπει να ξαναγράψουμε το πρώτο μας πρόγραμμα λαμβάνοντας υπόψη τα παραπάνω. Έτσι, το πρόγραμμα διαμορφώνεται ως εξής:

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

```
int main(void)
```

```
{  
    printf("Hello world");
```

```
    return 0;  
}
```

Το πρόθεμα `int` πριν από το όνομα της συνάρτησης `main()` υποδηλώνει ότι ο **τύπος** της συνάρτησης είναι `int` και επομένως επιστρέφει έναν ακέραιο αριθμό.

Ο τύπος `void`, μέσα στις παρενθέσεις της `main()`, δηλώνει ρητά ότι η `main()` **δεν** έχει παραμέτρους.

Με την εντολή `return 0;` η συνάρτηση `main()` **επιστρέφει**, ως **κωδικό εξόδου** το 0.

Από το πρότυπο C99 και μετά επιβάλλεται η `main()` να είναι τύπου `int`, και επομένως πρέπει να επιστρέφει (με την εντολή `return`) έναν ακέραιο αριθμό. Όμως, αρκετοί μεταγλωττιστές μπορούν να μεταγλωττίσουν χωρίς πρόβλημα το πρόγραμμά μας, ακόμα και αν δεν δηλώσουμε ως `int` τη συνάρτηση `main()` και παραλείψουμε την εντολή `return`. Αυτό γίνεται για λόγους συμβατότητας με τα αρχικά πρότυπα της γλώσσας, τα οποία δεν επέβαλλαν αυτούς τους κανόνες. Ο τύπος `void`, μέσα στις παρενθέσεις της `main()`, δηλώνει ρητά ότι η `main()` **δεν** έχει παραμέτρους. Τον τύπο `void` θα τον συναντήσουμε αργότερα στη συνέχεια του βιβλίου, όπου θα γίνει πλήρως κατανοητή η χρήση του. Μέχρι τότε μη σας απασχολεί!

Από τώρα και στο εξής θα χρησιμοποιούμε την παραπάνω σύνταξη της `main()`.

Προγράμματα με περισσότερες συναρτήσεις

Ένα πιο σύνθετο πρόγραμμα στη C περιλαμβάνει περισσότερες συναρτήσεις, εκτός από τη συνάρτηση `main()`. Ο κώδικας των συναρτήσεων αυτών συνήθως γράφεται μετά από τη συνάρτηση `main()`. Κάθε συνάρτηση πρέπει να έχει το δικό της ξεχωριστό όνομα και τύπο. Παρακάτω φαίνεται η δομή ενός προγράμματος της C με τρεις συναρτήσεις, τη `main()` και άλλες δύο:

```
#include <stdio.h>
int main()
{
    προτάσεις της συνάρτησης main();
    .....
}
τύπος function1()
{
    προτάσεις της συνάρτησης function1();
    .....
}
τύπος function2()
{
    προτάσεις της συνάρτησης function2();
    .....
}
```

Ο **τύπος** μιας συνάρτησης καθορίζει το είδος της τιμής που θα επιστρέψει η συνάρτηση. Επομένως, μια συνάρτηση μπορεί να είναι τύπου `int`, `float`, `char` κ.λπ. Όταν μια συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως `void`.

- Οι συναρτήσεις ορίζονται μετά ή πριν από τη `main()`.
- Κάθε συνάρτηση περιέχει τις δικές της προτάσεις, οι οποίες περικλείονται σε ένα αριστερό και δεξιό άγκιστρο (`{}`).
- Κάθε συνάρτηση πρέπει να έχει το δικό της ξεχωριστό όνομα το οποίο θα πρέπει να πληροί τους κανόνες ονομασίας των αναγνωριστικών της C, στους οποίους θα αναφερθούμε στη συνέχεια του κεφαλαίου.
- Κάθε συνάρτηση πρέπει να έχει τύπο. Ο τύπος της συνάρτησης καθορίζει το είδος της τιμής που θα επιστρέψει η συνάρτηση. Για παράδειγμα, μια συνάρτηση που επιστρέφει έναν ακέραιο αριθμό πρέπει να δηλωθεί ως τύπου `int`, ενώ μια συνάρτηση που επιστρέφει έναν χαρακτήρα πρέπει να δηλωθεί ως `char`. Θα δούμε αργότερα ότι, όταν μια συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως `void`.
- Για να εκτελεστεί μια συνάρτηση, πρέπει να κληθεί από τη `main()` ή από κάποια άλλη συνάρτηση.

Η χρήση περισσότερων συναρτήσεων στα προγράμματα που κατασκευάζουμε θα αναλυθεί στο Κεφάλαιο 9. Μέχρι τότε, όλα τα προγράμματα που θα χρησιμοποιήσουμε θα περιέχουν μόνο μία συνάρτηση, τη συνάρτηση `main()`.

Σχόλια προγράμματος

Ο σωστός σχεδιασμός ενός προγράμματος επιβάλλει τη χρήση σχολίων μέσα στο ίδιο το πρόγραμμα, με σκοπό να το κάνει πιο ευανάγνωστο και κατανοητό. **Τα σχόλια μπορούν να περιλαμβάνουν οποιοδήποτε μήνυμα, αρχίζουν με το ζεύγος χαρακτήρων /*, και τελειώνουν με το ζεύγος χαρακτήρων */.**

Ακολουθεί ένα παράδειγμα χρήσης σχολίων μέσα σε ένα πρόγραμμα:

```
/* Το πρώτο μου πρόγραμμα στη C
   University of Bath U.K 24/10/1983 */
#include <stdio.h>
int main(void)
{
    /* Χρήση της συνάρτησης printf() */
    printf("Hello world\n");
    /* Εμφανίζει ένα σύνολο χαρακτήρων */
    return 0;
}
```

Αναφέραμε ήδη ότι ο τύπος **void**, μέσα στις παρενθέσεις της **main()**, δηλώνει ρητά ότι η **main()** δεν έχει παραμέτρους. Αυτό απαιτείται από τα πρότυπα της γλώσσας, και συγκεκριμένα από το πρότυπο **C99** και μετά. Τον τύπο **void** θα τον συναντήσουμε ξανά και θα τον αναλύσουμε στο Κεφάλαιο 9. Μέχρι τότε, απλώς θα τον βάζουμε μέσα στις παρενθέσεις της **main()**, για να είμαστε συμβατοί με τα τελευταία πρότυπα της γλώσσας.

Παρατηρούμε ότι σχόλια μπορούν να τοποθετηθούν οπουδήποτε μέσα σε ένα πρόγραμμα της C, είτε στην αρχή είτε ενδιάμεσα στον κώδικα.

Το πρότυπο C99 εισήγαγε μια νέα μορφή σχολίων, τα **σχόλια γραμμής**. Τα σχόλια γραμμής ξεκινούν με δύο καθέτους (//) και τελειώνουν στο τέλος της γραμμής.

Για παράδειγμα:

```
// Το πρώτο μου πρόγραμμα στη C
// University of Bath U.K 24/10/1983
#include <stdio.h>
int main(void)
{
    // Χρήση της συνάρτησης printf()
    printf("Hello world\n"); // Θα εμφανίσει το κείμενο Hello world
    return 0;
}
```

2_hello_world.c

Όταν ο μεταγλωττιστής (compiler) της C εντοπίσει τον συνδυασμό **/***, αγνοεί οτιδήποτε βρει μέσα στο πρόγραμμα μέχρι το επόμενο ***/**. Επίσης, ο μεταγλωττιστής αγνοεί οτιδήποτε υπάρχει μετά από τη διπλή κάθετο (**//**) μέχρι το τέλος της γραμμής.

Δηλώσεις μεταβλητών

Πριν χρησιμοποιηθεί οποιαδήποτε μεταβλητή, πρέπει πρώτα να δηλωθεί. Στην πρόταση της δήλωσης ενημερώνουμε τον μεταγλωττιστή της C για το **όνομα** και τον **τύπο** της μεταβλητής. Για παράδειγμα, η πρόταση:

Παραδείγματα

Π2.1 Στο παρακάτω πρόγραμμα υπάρχουν κάποια συνηθισμένα συντακτικά λάθη:

```
#include <stdio.h>;
#include <stdlib.h>
int main(void)
{
    int a,b;
    a=10;
    b=18;
    c=(a+b)/2.0;
    printf('Τέλος');
    return 0
}
```

- Η πρώτη οδηγία `#include` τερματίζεται με ερωτηματικό (;) ενώ η δεύτερη οδηγία `include` δεν έχει το πρόθεμα της δέσης `#`.
- Χρησιμοποιείται η μεταβλητή `c` χωρίς προηγουμένως να έχει δηλωθεί.
- Στην `printf()`, η συμβολοσειρά «Τέλος» πρέπει να βρίσκεται μέσα σε διπλά, και όχι σε μονά, εισαγωγικά.
- Η τελευταία πρόταση `return 0` δεν τερματίζεται με ερωτηματικό (;).

Π2.2 Στο πρόγραμμα που ακολουθεί δηλώνεται η μεταβλητή `a` (η οποία όμως δεν χρησιμοποιείται αργότερα) και εμφανίζεται στην οθόνη η φράση «**Η γλώσσα C σε βάθος**». Η συνάρτηση `printf()`, εκτός από την εμφάνιση των χαρακτήρων, επιστρέφει και μια τιμή (το πλήθος των χαρακτήρων που εμφάνισε), η οποία στη συγκεκριμένη περίπτωση χάνεται.

```
#include <stdio.h>
int main(void)
{
    int a;
    printf("Η γλώσσα C σε βάθος");
    return 0;
}
```

Αντιθέτως, στον κώδικα που ακολουθεί η τιμή που επιστρέφει η `printf()` ανατίθεται στη μεταβλητή `a` η οποία, μετά από την εκτέλεση της πρότασης `a=printf("Η γλώσσα C σε βάθος")` παίρνει την τιμή 19 (το πλήθος των χαρακτήρων που εμφανίστηκαν, μαζί με τα κενά διαστήματα).

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a,b;
    a=printf("Η γλώσσα C σε βάθος");
    b=rand();
    return 0;
}
```

Συμπεριλαμβάνουμε το αρχείο κεφαλίδας `stdlib.h`, για να μπορούμε να χρησιμοποιήσουμε τη συνάρτηση `rand()`.

Στη μεταβλητή `b` καταχωρίζεται ο τυχαίος ακέραιος αριθμός που επιστρέφει η συνάρτηση `rand()`. Επισημαίνεται επίσης ότι, για να μπορέσουμε να χρησιμοποιήσουμε τη συνάρτηση `rand()`, θα πρέπει να συμπεριλάβουμε το αρχείο κεφαλίδας `stdlib.h` στο οποίο δηλώνεται.

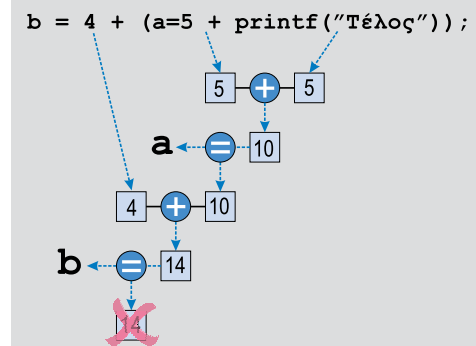
Π2.3 Ποια τιμή θα πάρει η μεταβλητή `b` στο παρακάτω πρόγραμμα;

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a,b;
    a=rand();
    rand();
    b=4+(a=5+printf("Τέλος"));
    return 0;
}
```

Η παράσταση `a=rand()` έχει ως αποτέλεσμα την καταχώριση ενός τυχαίου ακέραιου αριθμού στη μεταβλητή `a`.

Η επόμενη πρόταση `rand();` καλεί τη συνάρτηση `rand()` η οποία επιστρέφει έναν τυχαίο αριθμό⁷, ο οποίος όμως δεν καταχωρίζεται πουθενά και επομένως χάνεται. Ουσιαστικά αυτή η πρόταση είναι άχρηστη. Η προτελευταία πρόταση `b=...` αποτελείται από μια παράσταση, όχι και τόσο παράλογη αν αναλογιστούμε τη διαφορετική φιλοσοφία της C.

Η συνάρτηση `printf("Τέλος")` επιστρέφει την τιμή 5 (το πλήθος των χαρακτήρων που εμφανίσε). Σχεδόν κάθε συ-



⁷ Στην πραγματικότητα, η συνάρτηση `rand()` επιστρέφει έναν ψευδοτυχαίο αριθμό. Δηλαδή, η σειρά των τυχαίων αριθμών που επιστρέφει είναι πάντα η ίδια. Αργότερα θα δούμε ότι με την κατάλληλη χρήση μιας άλλης συνάρτησης, της `srand()`, μπορούμε να τροποποιήσουμε την προεπιλεγμένη σειρά των ψευδοτυχαίων αριθμών και να πάρουμε πραγματικά τυχαίους αριθμούς.

νάρτηση βιβλιοθήκης της C επιστρέφει κάποια τιμή. Η τιμή αυτή μπορεί να μας είναι χρήσιμη μπορεί και όχι. Για παράδειγμα, σπάνια χρησιμοποιούμε την τιμή που επιστρέφει η `printf()`.

- Το 10 (ως αποτέλεσμα της παράστασης $5+5$) καταχωρίζεται στη μεταβλητή `a` και η πλήρης παράσταση επιστρέφει την τιμή 10.
- Το 14 (ως αποτέλεσμα της παράστασης $10+4$) καταχωρίζεται στη μεταβλητή `b`.
- Η πλήρης παράσταση επιστρέφει την τιμή 14, η οποία δεν χρησιμοποιείται.

Η σειρά εκτέλεσης των πράξεων θα γίνει ξεκάθαρη όταν θα αναφερθούμε στην προτεραιότητα των τελεστών στο Κεφάλαιο 4 (σελίδα 125).

Π2.4 Τι θα περιέχουν οι μεταβλητές `a`, `b`, `c` και `d`, μετά το τέλος του παρακάτω κώδικα;

```
#include <stdio.h>
#define ETOS 1980
#define ONOMA "Χρυσάνθη"
int main(void)
{
    int a,b,c=10,d;
    a=2016-ETOS;
    b=a>35;
    c=a==30;
    d=printf(ONOMA);
    return 0;
}
```

- Η μεταβλητή `a` θα περιέχει τη τιμή 36, δηλαδή το αποτέλεσμα της πράξης $2016-1980$. Ας μην ξεχνάμε ότι το αναγνωριστικό `ETOS` αντικαθίσταται από το 1980.
- Η μεταβλητή `b` θα περιέχει τη τιμή 1, δηλαδή το αποτέλεσμα της λογικής παράστασης `a>35`. Επειδή η μεταβλητή `a` περιέχει το 36, η λογική παράσταση είναι αληθής, επομένως επιστρέφει την τιμή 1.
- Η μεταβλητή `c` θα περιέχει τη τιμή 0, δηλαδή το αποτέλεσμα της λογικής παράστασης `a==30`. Επειδή η μεταβλητή `a` περιέχει το 36, η λογική παράσταση είναι ψευδής, επομένως επιστρέφει την τιμή 0.
- Η μεταβλητή `d` θα περιέχει τη τιμή 8 που είναι η τιμή η οποία επιστρέφεται από τη συνάρτηση `printf()`, δηλαδή το πλήθος των χαρακτήρων που εμφάνισε στην οθόνη. Μην ξεχνάτε ότι το αναγνωριστικό `ONOMA` αντικαθίσταται από τη συμβολοσειρά «Χρυσάνθη».

Συχνά λάθη

- ❖ Ξεχνάμε να δηλώσουμε τη `main()` ως τύπου `int` και να προσθέσουμε την εντολή `return 0`. Αυτό είναι τυπικό (σύμφωνα με το πρότυπο C99) και όχι ουσιαστικό λάθος. Αρκετοί μεταγλωττιστές δεν θα έχουν πρόβλημα.
- ❖ Όλες οι δηλωτικές και εκτελέσιμες προτάσεις στη C τερματίζονται με ερωτηματικό (;). Καμιά φορά το ξεχνάμε!
- ❖ Κατά λάθος τερματίζουμε τις οδηγίες `#include` και `#define` με ερωτηματικό.
- ❖ Χρησιμοποιούμε μια μεταβλητή χωρίς προηγουμένως να την έχουμε δηλώσει.
- ❖ Εκτελούμε διαίρεση με ακέραιους αριθμούς και περιμένουμε το αποτέλεσμα να είναι δεκαδικός. Για παράδειγμα, περιμένουμε ότι το αποτέλεσμα της παράστασης $5/2$ θα είναι 2.5, ενώ είναι 2!
- ❖ Χρησιμοποιούμε συναρτήσεις βιβλιοθήκης χωρίς να έχουμε συμπεριλάβει (με `#include`) τα απαραίτητα αρχεία κεφαλίδων.

Ανασκόπηση Κεφαλαίου 2

- Κάθε πρόγραμμα της C πρέπει να περιέχει μία συνάρτηση `main()` η οποία είναι εκείνη που εκτελείται πρώτη. Κάθε πρόγραμμα της C μπορεί να περιέχει και άλλες συναρτήσεις εκτός από τη `main()`, οι οποίες για να εκτελεστούν πρέπει να κληθούν.
- Κάθε μεταβλητή που χρησιμοποιείται πρέπει να δηλώνεται. Οι δηλωτικές προτάσεις των μεταβλητών συνήθως τοποθετούνται στην αρχή μιας συνάρτησης.
- Οι βασικοί τύποι μεταβλητών στη C είναι τέσσερις: ο τύπος `char`, ο τύπος `int`, ο τύπος `float`, και ο τύπος `double`. Τα τελευταία πρότυπα της γλώσσας υποστηρίζουν και τον λογικό τύπο δεδομένων `bool`.
- Οι αριθμητικές παραστάσεις είναι πράξεις μεταξύ αριθμητικών δεδομένων και έχουν αποτέλεσμα έναν αριθμό.
- Οι λογικές παραστάσεις χρησιμοποιούν συγκριτικούς και λογικούς τελεστές και έχουν αποτέλεσμα μια **αληθή** τιμή (τιμή 1 ή true) ή **ψευδή** τιμή (τιμή 0 ή false).
- Ο τελεστής ανάθεσης = καταχωρίζει το αποτέλεσμα της παράστασης στα δεξιά του στη μεταβλητή που βρίσκεται στα αριστερά του, π.χ. `a=b+3`
- Μια σύνθετη πρόταση αποτελείται από απλές προτάσεις μέσα σε ένα αριστερό και δεξιό άγκιστρο. Η C αντιμετωπίζει μια σύνθετη πρόταση ως **μία** πρόταση.

Ασκήσεις Κεφαλαίου 2

- 2.1 Τι θα περιέχουν οι μεταβλητές **a**, **b**, και **c** μετά το τέλος του παρακάτω κώδικα; ★

```
int main(void)
{
    int a,b,c=3;
    a=b=2;
    a=c+b;
    return 0;
}
```

- 2.2 Τι θα περιέχουν οι μεταβλητές **a**, **b**, και **c** μετά το τέλος του παρακάτω κώδικα; ★

```
#define MM 23
int main(void)
{
    const int c=3;
    int a,b;
    a=4+(b=2);
    b=c+b+MM;
    return 0;
}
```

- 2.3 Εντοπίστε τα λάθη στον παρακάτω κώδικα: ★★

```
#include stdio.h
#define MM 23;
int main(void)
{
    const int c=3;
    int a,b;
    a=2;
    float d;
    d=4.3
    a=4+(b=2);
    MM=10;
    3=a;
    c=c+b+MM;
    return 0;
}
```


Κεφάλαιο

Προετοιμασία
για αργότερα

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αρχικά εμβαθύνει στον τρόπο με τον οποίο η C διαχειρίζεται τη μνήμη του υπολογιστή, γνώση απαραίτητη για την κατανόηση των διάφορων μηχανισμών της.

Αναλύει τις βασικές συναρτήσεις εξόδου και εισόδου, μέσω των οποίων η C εμφανίζει πληροφορίες στην οθόνη και διαβάζει δεδομένα από το πληκτρολόγιο.

Παρουσιάζεται η δυνατότητα προσθήκης εντολών ελέγχου στα προγράμματά σας ώστε κάποιες λειτουργίες να εκτελούνται ανάλογα με την τιμή κάποιας συνθήκης.

H C και η μνήμη

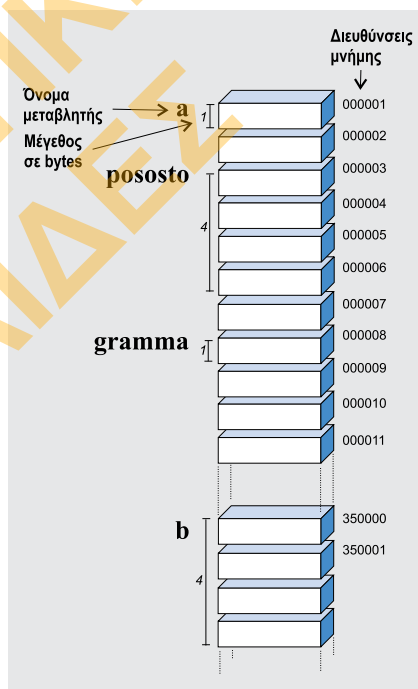
Όπως έχει ήδη αναφερθεί, η C χειρίζεται τέσσερις βασικούς τύπους μεταβλητών: τους **int**, **char**, **float**, και **double**, καθώς και τον τύπο **bool** στις τελευταίες εκδόσεις της γλώσσας. Παρομοιάσαμε την κάθε μεταβλητή σαν ένα κουτί με ένα όνομα και κάποιο περιεχόμενο. Οι μεταβλητές καταλαμβάνουν χώρο στη μνήμη RAM του Η/Υ, αλλά γνωρίζουμε ότι κάθε θέση μνήμης RAM έχει μέγεθος 1 byte. Μήπως αυτό σημαίνει ότι κάθε μεταβλητή ενός προγράμματος καταλαμβάνει ακριβώς τον ίδιο χώρο στη μνήμη του Η/Υ; ΟΧΙ βέβαια.

Κάθε μεταβλητή καταλαμβάνει έναν αριθμό θέσεων μνήμης (byte), ο οποίος εξαρτάται από τον τύπο της μεταβλητής, από τον τύπο του συστήματος Η/Υ στο οποίο δουλεύουμε, αλλά και από τον μεταγλωττιστή που χρησιμοποιούμε.

Στα συστήματα προσωπικών Η/Υ που βασίζονται στους επεξεργαστές 32 bit της INTEL, οι μεταβλητές τύπου **int** καταλαμβάνουν τέσσερα byte, οι **char** και **bool** ένα byte, οι **float** τέσσερα, και οι **double** 8. Αργότερα θα αναφερθούμε και σε άλλους τύπους μεταβλητών, που όμως βασίζονται στους βασικούς τύπους, με διαφορετικά μεγέθη σε byte⁸. Έστω ότι έχουμε τέσσερις μεταβλητές, οι οποίες δηλώνονται με τις παρακάτω προτάσεις:

```
char a, gamma;
float pososto;
int b;
```

Στο διπλανό σχήμα φαίνεται εποπτικά η μνήμη ενός Η/Υ σαν ένα σύνολο από κουτιά (θέσεις μνήμης) το ένα πάνω από το άλλο. Το κάθε κουτί έχει μέγεθος ένα byte και διαθέτει έναν αύξοντα αριθμό (ο αριθμός στα δεξιά), ο οποίος αποτελεί τη **διεύθυνση** του κουτιού μέσα στη μνήμη. Κάθε μεταβλητή που δηλώνεται δεσμεύει κάποιες θέσεις μνήμης, ανάλογα με τον τύπο της. Έτσι, η **a** και η **gamma** δεσμεύουν από μία θέση (την 1 και την 8 αντίστοιχα), η **pososto** τέσσερις (τις 3, 4, 5, και 6), και η **b** επίσης τέσσερις (τις 350000, 350001, 350002, και 350003). Εκτός λοιπόν από το όνομα και το περιεχόμενο μιας μεταβλητής –έννοιες που είναι ήδη γνωστές– εδώ βλέπουμε άλλες δύο νέες έννοιες: τη **διεύθυνση** και το **μέγεθος** μιας μεταβλητής.



⁸ Το μέγεθος που καταλαμβάνει ο κάθε τύπος δεδομένων εξαρτάται από τον τύπο του συστήματός μας, το λειτουργικό σύστημα, αλλά και τον μεταγλωττιστή. Μπορεί να διαφέρει σε συστήματα 16, 32 ή 64bits.

Διεύθυνση μιας μεταβλητής είναι η διεύθυνση του πρώτου byte των θέσεων (ή της θέσης) μνήμης που δεσμεύει η μεταβλητή.

Έτσι, οι διευθύνσεις των παραπάνω μεταβλητών είναι:

a → 000001
b → 350000
pososto → 000003
gramma → 000008

Μέγεθος μιας μεταβλητής είναι ο αριθμός των byte που δεσμεύει αυτή η μεταβλητή, και καθορίζεται από τον τύπο της.

Η C διαθέτει τελεστές οι οποίοι επιστρέφουν τη διεύθυνση και το μέγεθος μιας μεταβλητής.

Ο τελεστής &

Ο τελεστής **&** επιστρέφει τη διεύθυνση μιας μεταβλητής. Έτσι, σύμφωνα με το προηγούμενο σχήμα, η παράσταση **&a** επιστρέφει τιμή 1 και η παράσταση **&gramma** τιμή 8. Η διεύθυνση μιας θέσης μνήμης στα συστήματα προσωπικών υπολογιστών που βασίζονται στους επεξεργαστές 32 bit της INTEL είναι πάντα ένας αριθμός μεγέθους 4 byte, ο οποίος συνήθως αναπαριστάνεται σε δεκαεξαδική μορφή.

Όπως θα δούμε αργότερα, αν και οι διευθύνσεις μνήμης είναι ουσιαστικά ακέραιοι αριθμοί, η C τις χειρίζεται με έναν εντελώς ιδιαίτερο τρόπο.

Ο τελεστής sizeof

Ο τελεστής **sizeof** επιστρέφει το πλήθος των byte που δεσμεύει μια μεταβλητή ή γενικότερα κάποιος τύπος δεδομένων. Για παράδειγμα, με βάση τις δηλώσεις των μεταβλητών της προηγούμενης σελίδας, οι παραστάσεις που ακολουθούν επιστρέφουν τις τιμές που αναφέρονται:

sizeof a → επιστρέφει την τιμή 1
sizeof pososto → επιστρέφει την τιμή 4
12 + sizeof gramma → επιστρέφει την τιμή 13 (12+1).
sizeof(int) → επιστρέφει το 4 (μέγεθος των δεδομένων τύπου int)

Αν χρησιμοποιηθεί κάποιος τύπος δεδομένων, τότε πρέπει οπωσδήποτε να περικλείεται σε παρενθέσεις. Επίσης, ο τελεστής **sizeof** μπορεί να εφαρμοστεί σε παραστάσεις, οπότε αποδίδει το μέγεθος του αποτελέσματος της παράστασης. Για παράδειγμα, η παράσταση **sizeof(pososto/2.0)** επιστρέφει την τιμή 8, αφού το αποτέλεσμα της διαίρεσης **int/double** είναι τύπου **double** και επομένως έχει μέγεθος 8 byte. Στην περίπτωση αυτή, θα πρέπει και η παράσταση να περικλείεται σε παρενθέσεις.

Τρεις συναρτήσεις παρακαλώ

Για την εκμάθηση της C δεν θα ακολουθήσουμε τον κλασικό τρόπο διδασκαλίας (εντολή προς εντολή), όπως πιθανώς ακολουθείται για τη διδασκαλία άλλων γλωσσών προγραμματισμού. Λόγω των ιδιοτήτων της C, πρέπει να γνωρίζουμε κάποιες βασικές έννοιες και να χρησιμοποιούμε ορισμένες εντολές και συναρτήσεις, μόνο και μόνο για να μπορούμε να συνθέσουμε τα πρώτα μας απλά παραδείγματα. Έτσι, χωρίς ακόμα να εμβαθύνουμε και να αναλύσουμε πλήρως το συντακτικό τους, θα γνωρίσουμε τις συναρτήσεις `printf()`, `scanf()`, `exit()`, καθώς και την εντολή `if`.

Έξοδος στην οθόνη – Η συνάρτηση `printf()`

Η συνάρτηση `printf()` μάς δίνει τη δυνατότητα να εμφανίζουμε μορφοποιημένες πληροφορίες στην οθόνη του Η/Υ. Κάνει την ίδια δουλειά με τις εντολές εξόδου (στην οθόνη) άλλων γλωσσών προγραμματισμού (π.χ. η εντολή `print` της BASIC, η Γράψε της ΓΛΩΣΣΑΣ), όμως με διαφορετικό τρόπο.

Μέχρι τώρα χρησιμοποιούσαμε τη συνάρτηση `printf()` μόνο για να εμφανίζουμε μια συμβολοσειρά στην οθόνη. Για παράδειγμα:

```
printf("Η γλώσσα C σε βάθος");
```

Τι γίνεται όμως στην περίπτωση που θέλουμε να εμφανίσουμε στην οθόνη τα περιεχόμενα μεταβλητών ή τα αποτελέσματα παραστάσεων; Τότε θα πρέπει να εμπλουτίσουμε το αλφαριθμητικό της `printf()` με ειδικούς συνδυασμούς χαρακτήρων, ανάλογα με τον τύπο των πληροφοριών που θέλουμε να εμφανίσουμε. Για παράδειγμα, ο παρακάτω κώδικας εμφανίζει τα περιεχόμενα των μεταβλητών `a` και `b`:

```
int a,b;
a=10;
b=20;
printf("%d %d",a,b);
```

Στη θέση του πρώτου `%d` θα εμφανιστεί το περιεχόμενο της `a` και στη θέση του δεύτερου `%d` το περιεχόμενο της `b`.

ενώ η ακόλουθη σύνταξη της `printf()`:

```
printf("a=%d b=%d",a,b);
```

θα εμφανίσει τα περιεχόμενα των μεταβλητών `a` και `b` με τον τρόπο που βλέπετε στο διπλανό πλαίσιο.

10 20

a=10 b=20

Το πλήρες συντακτικό της `printf()` είναι το εξής:

`printf("αλφαριθμητικό μορφοποίησης", παρ1, παρ2, παρ3, ....., παρN);`

όπου **αλφαριθμητικό μορφοποίησης** (format string) είναι ένα σύνολο χαρακτήρων το οποίο περιέχει δύο ειδών πληροφορίες:

- Τους χαρακτήρες που θέλουμε να εμφανίσουμε.

- Ειδικούς συνδυασμούς (με τον χαρακτήρα %) για την εμφάνιση των τιμών των παραστάσεων που ακολουθούν (*παρ1~παρN*). Οι ειδικοί συνδυασμοί είναι οι εξής⁹:

%c για έναν χαρακτήρα (char)
 %d για έναν ακέραιο αριθμό (int)
 %f για έναν πραγματικό αριθμό (float ή double)

Οι *παρ1~παρN* είναι παραστάσεις των οποίων το αποτέλεσμα θα εμφανιστεί στην οθόνη, με τον τρόπο που καθορίζει το αλφαριθμητικό μορφοποίησης.

Το παρακάτω πρόγραμμα δείχνει τη χρήση της `printf()`:

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

3_printf1.c

```
int main(void)
```

```
{
    int a;
    char c;
    float b;
    a=15;
    b=a/2.0;
    c='A';
    printf("a+2=%d b=%f c=%c\n", a+2, b, c);
    printf("Τέλος\n");
    return 0;
}
```

a+2=17 b=7.500000 c=A
 Τέλος

Ας αναλύσουμε την πρόταση `printf()`:

```
printf("a+2=%d b=%f c=%c\n", a+2, b, c);
```

Το αλφαριθμητικό μορφοποίησης περιέχει τα ειδικά σύμβολα `%d`, `%f`, και `%c`, επομένως πρέπει να ακολουθούν τουλάχιστον τρεις παραστάσεις. Η τιμή της πρώτης θα εμφανιστεί ως ακέραιος (στη θέση του `%d`), της δεύτερης ως πραγματικός (στη θέση του `%f`), και της τρίτης ως χαρακτήρας (στη θέση του `%c`). Αν δεν ορίσουμε διαφορετικά, η `printf()` εμφανίζει τους πραγματικούς αριθμούς με έξι δεκαδικά ψηφία.

Οτιδήποτε άλλο υπάρχει μέσα στο αλφαριθμητικό μορφοποίησης θα εμφανιστεί στην οθόνη όπως ακριβώς είναι.

Ο χαρακτήρας `\n` στο τέλος του αλφαριθμητικού μορφοποίησης επιβάλλει μια αλλαγή γραμμής μετά το τέλος της εμφάνισης των χαρακτήρων.

Στην επόμενη πρόταση `printf()` του προγράμματος, `printf("Τέλος\n")`, δεν ακολουθεί καμία παράσταση, και το αλφαριθμητικό μορφοποίησης "Τέλος\n" δεν περιέχει κανένα ειδικό σύμβολο (από τα `%d`, `%f`, και `%c`). Το μόνο που κάνει η συγκεκριμένη συνάρτηση `printf()` είναι να εμφανίζει αυτούσια τα περιεχόμενα του αλφαριθμητικού μορφοποίη-

⁹ Υπάρχουν και άλλοι συνδυασμοί, που αφορούν άλλους τύπους δεδομένων, τους οποίους θα συναντήσουμε αργότερα.

σης. Το `\n` είναι ένας χαρακτήρας (και όχι δύο ξεχωριστοί) από τους λεγόμενους «χαρακτήρες διαφυγής» (escape characters), οι οποίοι δεν εμφανίζονται στην οθόνη αλλά ελέγχουν μια συγκεκριμένη λειτουργία.

Μερικοί από τους χαρακτήρες διαφυγής στη C είναι οι εξής:

- `\n` new line (αλλαγή γραμμής)
- `\b` backspace (μία θέση πίσω)
- `\f` form feed (νέα σελίδα)
- `\r` carriage return (επαναφορά κεφαλής)
- `\t` tab (στηλοθέτης)

Οι χαρακτήρες αυτοί μπορούν να χρησιμοποιηθούν είτε μόνοι τους (`'\n'`), είτε σε συμβολοσειρές χαρακτήρων (character strings) π.χ. `"\fΚορυφή σελίδας\n"`. Αν το αλφαριθμητικό μορφοποίησης σε μια συνάρτηση `printf()` δεν τελειώνει με τον χαρακτήρα αλλαγής γραμμής `'\n'`, τότε η επόμενη εμφάνιση στην οθόνη αρχίζει από τη θέση που σταμάτησε η προηγούμενη `printf()`.

Το σύνολο των χαρακτήρων διαφυγής θα αναφερθεί σε επόμενο κεφάλαιο. Το πρόγραμμα που ακολουθεί δείχνει τη χρήση μερικών από τους παραπάνω χαρακτήρες διαφυγής.

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

```
int main(void)
```

```
{  
    int a=10,b=20;  
    printf("Πρώτη και\nΔεύτερη γραμμή\n");  
    printf("%d\t%d\n",a,b);  
    printf("%d\t%d\n",a+b,a-b);  
    printf("Ens\bd\n");  
    return 0;  
}
```

3_printf2.c

Πρώτη και	
Δεύτερη γραμμή	
10	20
30	-10
End	

Η συνάρτηση `printf()` μπορεί να χρησιμοποιηθεί μόνο με το *αλφαριθμητικό μορφοποίησης*, οπότε απλώς εμφανίζει τους χαρακτήρες που περιέχει. Για παράδειγμα, η πρόταση `printf("Η γλώσσα C σε βάθος")` θα εμφανίσει στην οθόνη τους χαρακτήρες «Η γλώσσα C σε βάθος». Όταν όμως θέλουμε να εμφανίσουμε περιεχόμενα μεταβλητών ή αποτελέσματα παραστάσεων, θα πρέπει να συμπεριλάβουμε μέσα στο αλφαριθμητικό μορφοποίησης ειδικούς συνδυασμούς (`%...`) και μετά από αυτό να ακολουθούν οι μεταβλητές και οι παραστάσεις.

Είσοδος από το πληκτρολόγιο – Η συνάρτηση scanf()

Μέχρι τώρα, για να καταχωρίσουμε μια τιμή σε μια μεταβλητή χρησιμοποιούσαμε τον τελεστή ανάθεσης =. Για παράδειγμα, το παρακάτω τμήμα κώδικα καταχωρίζει στις μεταβλητές **a** και **b** δύο τιμές και κατόπιν υπολογίζει και εμφανίζει το άθροισμά τους.

```
int a,b;
a=10;
b=20;
printf("Άθροισμα=%d\n",a+b);
```

Συνήθως ένα πρόγραμμα δημιουργείται από κάποιον προγραμματιστή και δεν χρησιμοποιείται από τον ίδιο, αλλά από κάποιον χρήστη ο οποίος πιθανότατα δεν γνωρίζει προγραμματισμό.

Στο παραπάνω παράδειγμα, αν ο χρήστης θέλει να υπολογίσει το άθροισμα δύο άλλων αριθμών, θα πρέπει να επέμβει στον κώδικα και να αλλάξει τις τιμές στις δύο προτάσεις **a=10** και **b=20**. Αυτό όμως προϋποθέτει γνώσεις προγραμματισμού και δεν οδηγεί σε δυναμικά προγράμματα. Δεν θα ήταν πολύ καλύτερα το ίδιο το πρόγραμμα να ζητάει από τον χρήστη να πληκτρολογήσει τις τιμές στο πληκτρολόγιο; Φανταστείτε ένα ρολόι στο οποίο, για να αλλάξουμε την ώρα, θα έπρεπε να το ανοίξουμε και να επεμβούμε στον μηχανισμό του. Δεν θα ήταν και τόσο εύχρηστο!

Η C δεν διαθέτει εντολές εισόδου για την ανάγνωση δεδομένων από το πληκτρολόγιο. Η δυνατότητα αυτή υλοποιείται με τη χρήση συναρτήσεων βιβλιοθήκης. Η πιο πλήρης συνάρτηση γι' αυτή τη λειτουργία είναι η συνάρτηση **scanf()**, η οποία διαβάζει δεδομένα από το πληκτρολόγιο και τα καταχωρίζει σε μεταβλητές του προγράμματος. Στη συνάρτηση καθορίζεται ο τύπος των δεδομένων που θα αναγνωστούν, καθώς και ο τρόπος ανάγνωσής τους. Επίσης, η συνάρτηση καθορίζει και τον αποθηκευτικό χώρο στον οποίο θα καταχωριστούν τα δεδομένα που διάβασε. Για παράδειγμα, ο παρακάτω κώδικας διαβάζει από το πληκτρολόγιο δύο ακέραιους αριθμούς και τους καταχωρίζει στις μεταβλητές **a** και **b**:

```
int a,b;
scanf("%d %d",&a,&b);
```

Διαβάζει από το πληκτρολόγιο δύο ακέραιους αριθμούς και τους καταχωρίζει στις μεταβλητές **a** και **b** αντίστοιχα.

Η συνάρτηση **scanf()** έχει μια ιδιομορφία: χρησιμοποιεί τις διευθύνσεις, και όχι τα ονόματα των μεταβλητών!

Η πλήρης σύνταξη της **scanf()** είναι σχεδόν ίδια με της **printf()**:

scanf("αλφαριθμητικό μορφοποίησης", διε1, διε2, διε3,....., διεN); όπου,

το **αλφαριθμητικό μορφοποίησης** (format string) καθορίζει τον τρόπο με τον οποίο πρέπει να **δοθούν** τα δεδομένα, καθώς και τον τύπο τους.

Το αλφαριθμητικό μορφοποίησης πρέπει να ακολουθείται από τις **διευθύνσεις** των μεταβλητών στις οποίες θα καταχωριστούν οι τιμές που πληκτρολογούνται. Το αλφαριθμητικό μορφοποίησης περιέχει τριών ειδών πληροφορίες:

- **Ειδικούς συνδυασμούς** για την **ανάγνωση** των τιμών, ανάλογα με τον τύπο των **μεταβλητών** στις οποίες θα καταχωριστούν.

%c για έναν απλό χαρακτήρα (καταχώριση σε char)

%d για ακέραιο αριθμό (καταχώριση σε int)

%f για πραγματικό αριθμό απλής ακρίβειας (καταχώριση σε float)

%lf για πραγματικό αριθμό διπλής ακρίβειας (καταχώριση σε double)

Υπάρχουν και άλλοι συνδυασμοί στους οποίους θα αναφερθούμε αργότερα.

- **Χαρακτήρες διαστήματος.** Αναγκάζουν τη **scanf()** να διαβάσει και να αγνοήσει ένα ή περισσότερα κενά διαστήματα, χαρακτήρες στηλοθέτη (tab), ή χαρακτήρες αλλαγής γραμμής, στη σειρά των δεδομένων που πληκτρολογούνται. Για παράδειγμα, το αλφαριθμητικό μορφοποίησης:

`"%d %d"`

διαβάζει δύο ακεραίους, αγνοώντας τα μεταξύ τους κενά, τους χαρακτήρες στηλοθέτη και τους χαρακτήρες αλλαγής γραμμής (αν υπάρχουν).

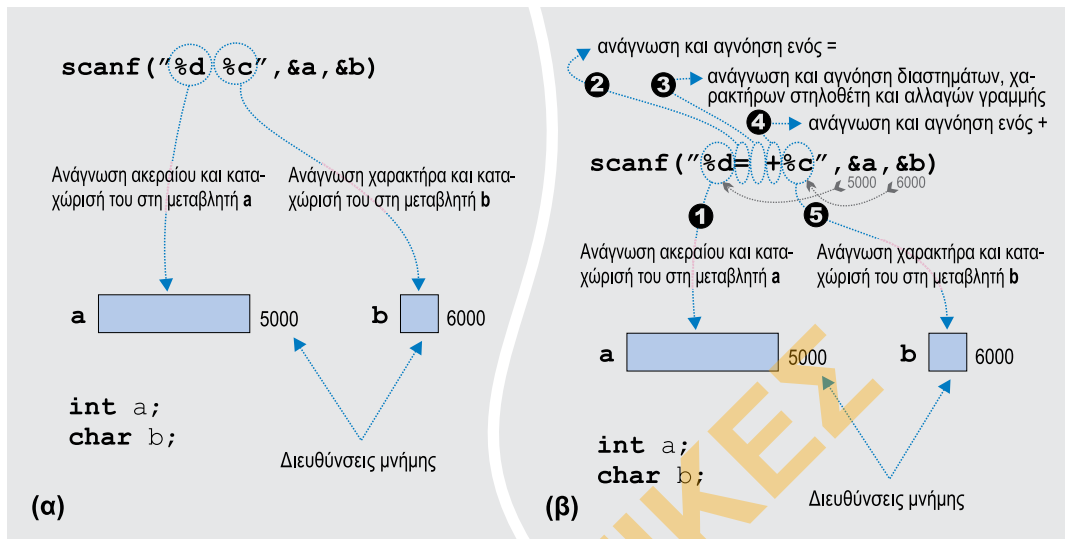
- **Άλλους χαρακτήρες.** Αναγκάζουν τη **scanf()** να διαβάσει και να αγνοήσει έναν συγκεκριμένο χαρακτήρα. Αν ο χαρακτήρας δεν βρεθεί, η **scanf()** τερματίζεται. Για παράδειγμα, το αλφαριθμητικό μορφοποίησης `"%d,%d"` διαβάζει έναν ακέραιο, μετά διαβάζει και αγνοεί ένα κόμμα (,) και τέλος διαβάζει ακόμη έναν ακέραιο. Αν δεν βρεθεί το κόμμα αμέσως μετά από τον πρώτο ακέραιο, τότε η **scanf()** τερματίζεται χωρίς μήνυμα σφάλματος και χωρίς να διαβάσει τον επόμενο αριθμό.

Οι **die1~dieN** είναι παραστάσεις των οποίων το αποτέλεσμα αποδίδει τη διεύθυνση μνήμης μιας μεταβλητής. Τα δεδομένα που πληκτρολογούνται θα καταχωριστούν στις μεταβλητές με αυτές τις διευθύνσεις. Ο τελεστής **&** χρησιμοποιείται συνήθως σε μια συνάρτηση **scanf()** για να αποδώσει τη διεύθυνση μιας μεταβλητής: Για παράδειγμα, η πρόταση:

```
scanf("%d %c", &a, &b);
```

περιμένει να πληκτρολογηθούν ένας ακέραιος αριθμός και ένας χαρακτήρας. Ο αριθμός θα καταχωριστεί στη μεταβλητή **a** (με διεύθυνση **&a**) ενώ ο χαρακτήρας στη μεταβλητή **b** (με διεύθυνση **&b**). Φυσικά, οι μεταβλητές **a** και **b** πρέπει να είναι τύπου **int** και **char** αντίστοιχα. Κατά την πληκτρολόγηση, πρέπει να υπάρχουν μεταξύ του αριθμού και του χαρακτήρα ένα ή περισσότερα διαστήματα, χαρακτήρες στηλοθέτη, ή αλλαγές γραμμής, τα οποία αγνοούνται (Σχήμα 3.1α).

Για να γνωρίζει ο χρήστης τι πρέπει να πληκτρολογήσει, συνήθως πριν από τη **scanf()** χρησιμοποιούμε την **printf()** ή κάποια άλλη συνάρτηση εξόδου (π.χ. την **puts()** που θα συναντήσουμε αργότερα), με την οποία εμφανίζουμε ένα κατάλληλο μήνυμα.



Σχήμα 3.1 Χρήση της συνάρτησης `scanf()`.

Το επόμενο παράδειγμα δείχνει τη χρήση της `scanf()` μέσα σε ένα πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
    int a,b;
    float c;
    printf("δώσε 2 αριθμούς:");
    scanf("%d %d", &a, &b);
    // υπολογισμός μέσου όρου
    c = (a+b)/2.0;
    // εμφάνιση αποτελεσμάτων
    printf("ο μέσος όρος των %d και %d είναι %f\n", a, b, c);
    return 0;
}
```

3 scanf.c

Διαβάζει από το πληκτρολόγιο δύο ακέραιους αριθμούς (που χωρίζονται με κενά διαστήματα, χαρακτήρες στηλοθέτη, ή αλλαγή γραμμής) και τους καταχωρίζει στις μεταβλητές **a** και **b** αντίστοιχα.

Η παρουσία άλλων χαρακτήρων μεταξύ των ειδικών συνδυασμών (%...), έχει ως αποτέλεσμα η `scanf()` να **περιμένει** να διαβάσει και να αγνοήσει τους συγκεκριμένους χαρακτήρες. Στο παράδειγμα του Σχήματος 3.1β η συνάρτηση:

```
scanf("%d= +%c", &a, &b);
```

- Θα διαβάσει έναν ακέραιο αριθμό και θα τον καταχωρίσει στη μεταβλητή **a**.
- Θα διαβάσει και θα αγνοήσει ένα =.
- Θα διαβάσει και θα αγνοήσει ένα ή περισσότερα διαστήματα ή αλλαγές γραμμής.
- Θα διαβάσει και θα αγνοήσει έναν χαρακτήρα +.
- Θα διαβάσει έναν χαρακτήρα και θα τον καταχωρίσει στη μεταβλητή **b**.

Κεφάλαιο

Ο τύπος δεδομένων int

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό παρουσιάζει τον χειρισμό των ακέραιων αριθμών από τη γλώσσα. Αναλύεται η εσωτερική τους αναπαράσταση καθώς και ο τρόπος μετατροπής τους σε δυαδικούς αριθμούς.

Αναφέρονται οι αριθμητικοί τελεστές που χρησιμοποιούνται και η δυνατότητα χειρισμού των ακέραιων αριθμών σε επίπεδο bit.

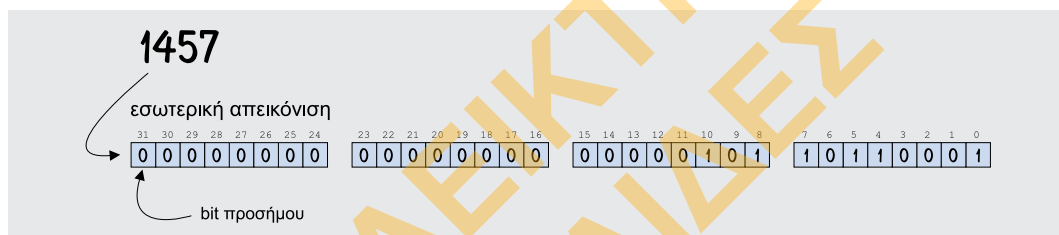
Επίσης δίνεται ιδιαίτερη προσοχή στην προτεραιότητα των τελεστών και σε τελεστές που παρουσιάστηκαν για πρώτη φορά στη γλώσσα C, όπως οι τελεστές μοναδιαίας αύξησης (++) και μοναδιαίας μείωσης (--).

Εισαγωγή

Το κεφάλαιο αυτό παρουσιάζει και αναλύει τον τρόπο χρήσης των ακεραίων (integers) στη γλώσσα C. Στο κεφάλαιο αυτό εξετάζεται η εσωτερική απεικόνιση των ακεραίων σταθερών και μεταβλητών, καθώς και οι τελεστές που εφαρμόζονται σε ακέραια μεγέθη. Πολλοί από αυτούς τους τελεστές εφαρμόζονται και σε άλλους τύπους δεδομένων.

Σταθερές τύπου int

Οι ακέραιες σταθερές δεν είναι τίποτε άλλο από απλοί ακέραιοι αριθμοί. Μια ακέραια σταθερά αποθηκεύεται στη μνήμη αφού μετατραπεί σε δυαδικό αριθμό. Μια σταθερά τύπου `int` μετατρέπεται σε έναν δυαδικό αριθμό μήκους 4 byte. Με 4 byte μπορεί να αναπαρασταθεί οποιοσδήποτε αριθμός από -2.147.483.648 μέχρι +2.147.483.647. Το bit στη θέση 31 (τα 32 bit αριθμούνται από το 0 έως το 31) χρησιμοποιείται για το πρόσημο και τα υπόλοιπα 31 για την αναπαράσταση του αριθμού. Το παρακάτω σχήμα δείχνει την εσωτερική αναπαράσταση του αριθμού 1457 σε ένα 32άμπιτο σύστημα H/Y.



Μεταβλητές τύπου int

Εκτός από τον τύπο `int` που έχουμε ήδη αναφέρει, η C διαθέτει άλλους δύο τύπους ακεραίων μεταβλητών: τους `long int` και `short int`.

Οι μεταβλητές τύπου `int` δηλώνονται ως εξής:

`int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....;`

Οι μεταβλητές τύπου `long int` δηλώνονται ως εξής:

`long int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....;` ή

`long όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....;`

Στα αρχικά πρότυπα της γλώσσας, αν δεν αναφερόταν ο τύπος μιας μεταβλητής, τότε αυτή θεωρούνταν ως τύπου `int`. Από το πρότυπο C99 και μετά δεν υποστηρίζεται η έμμεση δήλωση του τύπου `int`, τόσο στις μεταβλητές όσο και στις συναρτήσεις.

Οι μεταβλητές τύπου `short int` δηλώνονται ως εξής:

`short int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....;` ή

`short όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....;`

☞ Οι μεταβλητές τύπου `int` και `long int` δεσμεύουν 4 byte¹⁰ (32 bit) και μπορούν να αποθηκεύσουν έναν ακέραιο αριθμό από -2.147.483.648 μέχρι +2.147.483.647. Οι μεταβλητές τύπου `short int` δεσμεύουν 2 byte (16 bit) και μπορούν να αποθηκεύσουν έναν ακέραιο αριθμό από -32.768 μέχρι +32.767.

Μεταβλητές τύπου `int` χωρίς πρόσημο

Μπορούμε να ορίσουμε ότι μια μεταβλητή τύπου `int` (`int`, `long int` ή `short int`) δεν θα περιέχει πρόσημο, χρησιμοποιώντας το προσδιοριστικό `unsigned` πριν από το όνομα του τύπου, όπως παρακάτω:

`unsigned int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....; ή`
`unsigned long int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....; ή`
`unsigned short int όνομα_μεταβλητής1, όνομα_μεταβλητής2, ....; ή`

Οι μεταβλητές τύπου `unsigned` δεσμεύουν τα ίδια byte με τις αντίστοιχες προσημασμένες μεταβλητές. Όμως, σε μια μεταβλητή τέτοιου τύπου το πρώτο bit του αριθμού δεν χρησιμοποιείται για το πρόσημο αλλά για την αναπαράσταση του αριθμού. Επομένως, σε μεταβλητές τύπου `unsigned` μπορούν να αποθηκευτούν μόνο θετικοί αριθμοί, αλλά με διπλάσια απόλυτη τιμή (από τους αντίστοιχους προσημασμένους).

Όπως θα δούμε στο επόμενο κεφάλαιο, το προσδιοριστικό `unsigned` μπορεί να χρησιμοποιηθεί και σε μεταβλητές τύπου `char`. Αυτό γίνεται επειδή, όπως θα δούμε πολύ σύντομα, μια μεταβλητή τύπου `char` περιέχει έναν ακέραιο αριθμό!

Αρχική τιμή μεταβλητής

Όταν δηλώνουμε μια μεταβλητή, μπορούμε ταυτόχρονα να της δώσουμε αρχική τιμή. Για παράδειγμα, στην επόμενη πρόταση:

```
int a=4,b=8;
```

δηλώνονται δύο ακέραιες μεταβλητές `a` και `b`, στις οποίες ανατίθενται ταυτόχρονα οι τιμές 4 και 8 αντίστοιχα.

Όπως έχει ήδη αναφερθεί, αν δεν ανατεθεί τιμή σε μια μεταβλητή τότε αυτή έχει απροσδιόριστο περιεχόμενο και όχι 0 όπως ίσως θα περιμέναμε. Η C επιτρέπει επίσης η αρχική τιμή μιας μεταβλητής να προκύπτει από μια παράσταση. Για παράδειγμα, στη πρόταση:

```
int a=2*5,b=a*4;
```

δηλώνονται δύο μεταβλητές `a` και `b`, στις οποίες ανατίθενται ταυτόχρονα οι τιμές 10 και 40 αντίστοιχα.

¹⁰ Το μέγεθος του τύπου δεδομένων `int` εξαρτάται από τον τύπο του συστήματος Η/Υ. Είναι διαφορετικό για συστήματα 16bit, 32bit, ή 64bit. Αυτό ισχύει και για τους άλλους τύπους δεδομένων.

Αριθμητικοί τελεστές

Εκτός από τους «κλασικούς» αριθμητικούς τελεστές, η C εισήγαγε και μερικούς τελεστές οι οποίοι εμφανίστηκαν για πρώτη φορά σε γλώσσα προγραμματισμού, αλλά υιοθετήθηκαν από αρκετές σύγχρονες γλώσσες. Ο διπλάνος πίνακας αναφέρει τους διαθέσιμους αριθμητικούς τελεστές στη C. Δεν θα ασχοληθούμε με τους κλασικούς αριθμητικούς τελεστές, αλλά μόνο με τους νέους τελεστές **μοναδιαίας αύξησης** (`++`) και **μοναδιαίας μείωσης** (`--`) που εισάγει η C. Οι τελεστές `++` και `--` εφαρμόζονται **μόνο** σε μεταβλητές, και μπορούν να προηγούνται (προθεματικοί) ή να ακολουθούν (επιθηματικοί) το όνομα μιας μεταβλητής.

Τελεστής	Λειτουργία
<code>+</code>	Πρόσθεση
<code>-</code>	Αφαίρεση
<code>*</code>	Πολλαπλασιασμός
<code>/</code>	Διαίρεση
<code>%</code>	Υπόλοιπο ακέραιας διαίρεσης
<code>++</code>	Αύξηση κατά 1
<code>--</code>	Μείωση κατά 1

π.χ. `++a; a++; --a; a--;`

Ο τελεστής `++` αυξάνει το περιεχόμενο της μεταβλητής κατά 1, ενώ ο τελεστής `--` μειώνει το περιεχόμενο της μεταβλητής κατά 1. Η λειτουργία αυτή γίνεται ανεξάρτητα από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

```
int main(void)
```

```
{
    int a,b;
    b=a=5;
    ++a;
    --b;
    printf("a=%d b=%d\n",a,b);
    return 0;
}
```

Με την παράσταση `++a` η `a` αυξάνεται κατά 1 (6) και με την παράσταση `--b` η `b` μειώνεται κατά 1 (4). Έτσι η `printf()` θα εμφανίσει στην οθόνη: `a=6 b=4`

Όπως κάθε παράσταση, έτσι και οι παραστάσεις με τους τελεστές `++` και `--` επιστρέφουν μία τιμή. Η τιμή αυτή εξαρτάται από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

- Όταν ο τελεστής βρίσκεται πριν από τη μεταβλητή (προθεματικός), τότε πρώτα γίνεται η πράξη (αύξηση ή μείωση) και μετά επιστρέφεται η νέα τιμή της μεταβλητής (μετά από την αύξηση ή τη μείωση). Επομένως, στην παράσταση χρησιμοποιείται η νέα τιμή της μεταβλητής, μετά από την αύξηση ή τη μείωσή της. Για παράδειγμα, αν η μεταβλητή `a` έχει την τιμή 5, η πρόταση `++a;` αυξάνει το `a` κατά 1 και επιστρέφει την τιμή 6.
- Όταν ο τελεστής βρίσκεται μετά από τη μεταβλητή (επιθηματικός), τότε πρώτα επιστρέφει την τιμή της μεταβλητής (την παλιά τιμή, πριν από την πράξη) και μετά αυξάνει ή μειώνει το περιεχόμενο της μεταβλητής. Επομένως, στην παράσταση χρησιμοποιείται η παλιά τιμή της μεταβλητής πριν από την αύξησή της. Για παράδειγμα, αν η μεταβλητή `a` έχει την τιμή 5, η πρόταση `a++;` επιστρέφει την τιμή 5 και αυξάνει το `a` κατά 1.

Τελεστές σύντμησης

Ο τελεστής ανάθεσης = μπορεί να συνδυαστεί με άλλους αριθμητικούς τελεστές για τη δημιουργία συνδυασμένων τελεστών σε συντετμημένη μορφή. Για παράδειγμα η παράσταση $x = x + 5$ μπορεί να γραφεί και ως $x += 5$, ενώ η $x = x * 5$ μπορεί να γραφεί και ως $x *= 5$. Ο διπλανός πίνακας δείχνει με πέντε παραδείγματα τους συνδυασμούς σύντμησης και τις ισοδύναμες κανονικές παραστάσεις.

Σύντμηση	Αντίστοιχη παράσταση
$x += 5$	$x = x + 5$
$x -= 5$	$x = x - 5$
$x *= 5$	$x = x * 5$
$x /= 5$	$x = x / 5$
$x \% = 5$	$x = x \% 5$

Για λόγους σαφήνειας οι τελεστές σύντμησης δεν χρησιμοποιούνται στον κώδικα των παραδειγμάτων του βιβλίου, είναι όμως πολύ πιθανό να τους συναντήσουμε στη βιβλιογραφία ή σε έτοιμο κώδικα.

Δυαδικοί αριθμοί

Όπως γνωρίζουμε, για να επεξεργαστεί πληροφορίες ο Η/Υ τις μετατρέπει σε δυαδικούς αριθμούς. Η μετατροπή αυτή είναι συνήθως «αόρατη» στον προγραμματιστή, ο οποίος δεν χρειάζεται να γνωρίζει την εσωτερική αναπαράσταση των πληροφοριών. Η C μάς επιτρέπει να χειριζόμαστε πληροφορίες σε επίπεδο δυαδικών αριθμών. Στη συνέχεια αυτού του κεφαλαίου θα γνωρίσουμε τελεστές οι οποίοι επενεργούν σε μεμονωμένα bit ενός ακέραιου αριθμού. Επίσης, η γνώση της εσωτερικής αναπαράστασης των πληροφοριών βοηθάει στην επεξήγηση και την κατανόηση αρκετών λειτουργιών της C.

Μετατροπή δυαδικού σε δεκαδικό

Ένας δυαδικός αριθμός έχει ως βάση αρίθμησης το 2 (όπως ο δεκαδικός το 10). Κάθε δυαδικός αριθμός έχει έναν αντίστοιχο δεκαδικό, και αντιστρόφως. Τα ψηφία ενός αριθμού (δεκαδικού ή δυαδικού) αποτελούν τους συντελεστές των δυνάμεων της βάσης του (10 ή 2 αντίστοιχα).

Συντελεστής	Δύναμη του 10	Σύνολο ψηφίου
4	$x \cdot 1 (10^0)$	4
0	$x \cdot 10 (10^1)$	0
2	$x \cdot 100 (10^2)$	200
1	$x \cdot 1000 (10^3)$	1000
Σύνολο		1204

Ας αναλύσουμε, για παράδειγμα, τον δεκαδικό αριθμό 1204. Από τα δεξιά προς τα αριστερά, το 4 είναι ο συντελεστής του 1 (10^0), το 0 του 10 (10^1), το 2 του 100 (10^2), και το 1 του 1000 (10^3). Επομένως ο αριθμός είναι:

$$1 \cdot 1000 + 2 \cdot 100 + 0 \cdot 10 + 4 \cdot 1$$

δηλαδή 1204.

Αντίστοιχα, τα ψηφία ενός δυαδικού αριθμού αποτελούν τους συντελεστές των δυνάμεων του 2 (και όχι του 10), δηλαδή 1, 2, 4, 8, 16, 32, 64, κ.ο.κ.

Ας εξετάσουμε, για παράδειγμα, τον δυαδικό αριθμό $(1101)_2$. Από τα δεξιά προς τα αριστερά πάλι, το 1 είναι ο συντελεστής του 1 (2^0), το 0 του 2 (2^1), το 1 του 4 (2^2), και το 1 του 8 (2^3). Επομένως, ο αριθμός υπολογίζεται από την παράσταση:

$$1*8 + 1*4 + 0*2 + 1*1$$

δηλαδή είναι ο δεκαδικός αριθμός 13.

Συντελεστής	Δύναμη του 2	Σύνολο ψηφίου
1	$x 1 (2^0)$	1
0	$x 2 (2^1)$	0
1	$x 4 (2^2)$	4
1	$x 8 (2^3)$	8
Σύνολο		13

Μετατροπή δεκαδικού σε δυαδικό

Υποδεικνύει ενότητα που μπορεί να παραλειφθεί σε πρώτη ανάγνωση!

Κάθε δεκαδικός αριθμός έχει έναν αντίστοιχο δυαδικό. Ο τρόπος με τον οποίο βρίσκουμε τον αντίστοιχο δυαδικό αριθμό ενός δεκαδικού αναλύεται με το παρακάτω παράδειγμα, όπου βρίσκουμε τον αντίστοιχο δυαδικό αριθμό του 52. Θα ακολουθούμε τα επόμενα βήματα:

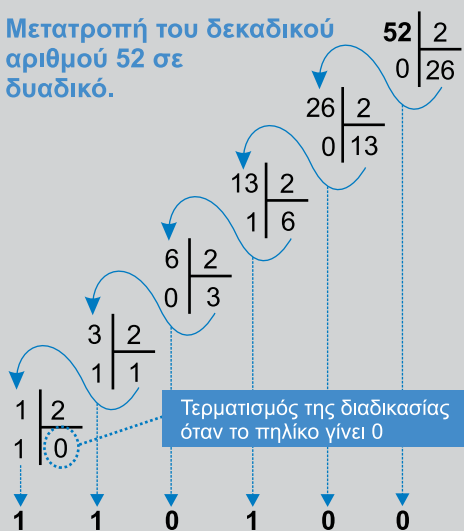
- 52/2 → πηλίκo 26, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^0)
- 26/2 → πηλίκo 13, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^1)
- 13/2 → πηλίκo 6, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^2)
- 6/2 → πηλίκo 3, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^3)
- 3/2 → πηλίκo 1, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^4)
- 1/2 → πηλίκo 0, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^5)

Το γεγονός ότι στην τελευταία διαίρεση το πηλίκo είναι 0 σηματοδοτεί το τέλος της διαδικασίας. Επομένως, ο αντίστοιχος δυαδικός αριθμός του 52 είναι ο $(110100)_2$. Πράγματι, αν εκτελέσουμε τις πράξεις θα προκύψει ο αριθμός 52:

$$1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 \rightarrow 32 + 16 + 0 + 4 + 0 + 0 \rightarrow 52$$

Η διαδικασία μετατροπής του αριθμού 52 στον αντίστοιχο δυαδικό αριθμό φαίνεται στο διπλάνo σχήμα. Διαιρούμε το 52 με το 2 και παίρνουμε υπόλοιπο 0 και πηλίκo 26. Το υπόλοιπο αποτελεί το τελευταίο ψηφίο του δυαδικού αριθμού. Έπειτα διαιρούμε το πηλίκo που βρήκαμε (26) πάλι με το 2. Το νέο υπόλοιπο αποτελεί το προτελευταίο ψηφίο του δυαδικού αριθμού. Αυτό συνεχίζεται μέχρι το πηλίκo να γίνει 0. Το τελευταίο υπόλοιπο αποτελεί το πρώτο ψηφίο του δυαδικού αριθμού. Έτσι, το **52** μετατρέπεται στον δυαδικό αριθμό: **(110100)₂**.

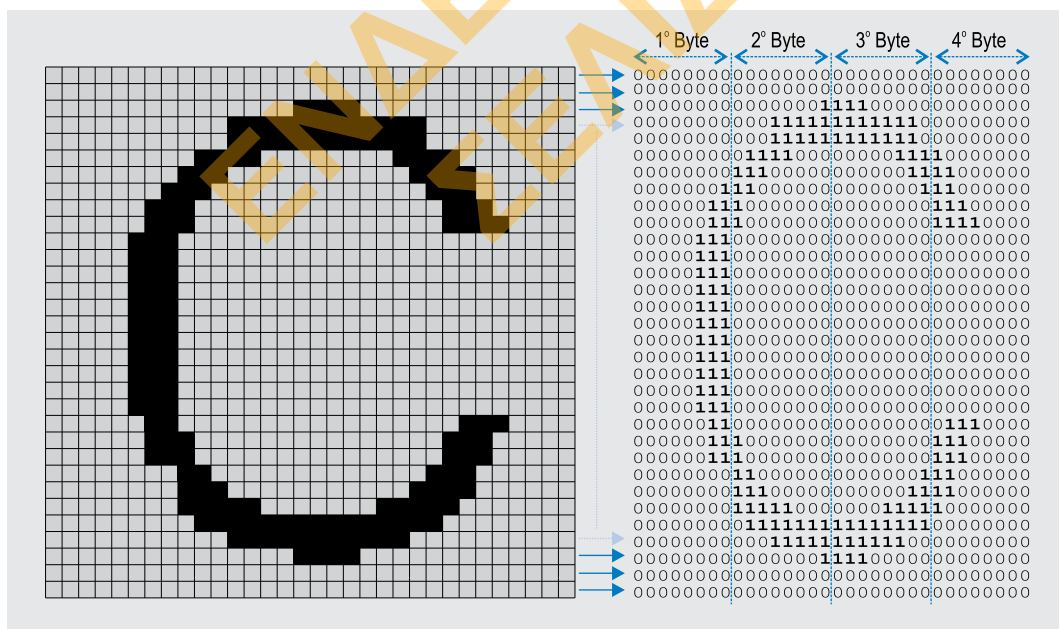
Μετατροπή του δεκαδικού αριθμού 52 σε δυαδικό.



Οι δυαδικοί τελεστές

Όπως γνωρίζουμε, όλες τις πληροφορίες που επεξεργάζεται ένας Η/Υ τις μετατρέπει σε δυαδικούς αριθμούς. Η μετατροπή αυτή είναι συνήθως «αόρατη» στον προγραμματιστή, ο οποίος δεν είναι απαραίτητο να γνωρίζει τον τρόπο εσωτερικής αναπαράστασης των πληροφοριών. Στο παράδειγμα του σχήματος 4.1, μια ασπρόμαυρη εικόνα μετατρέπεται σε σειρές από 1 και 0, τα οποία προσδιορίζουν τα pixel με μαύρο και άσπρο χρώμα αντίστοιχα. Η εικόνα έχει διαστάσεις 32x32 pixel και κάθε σειρά της εικόνας μετατρέπεται σε έναν ακέραιο αριθμό αποτελούμενο από byte. Όμως, δεν μας ενδιαφέρει αυτός ο αριθμός ως σύνολο, αλλά τα επιμέρους bit του. Επομένως, για να μπορέσουμε να τα επεξεργαστούμε, πρέπει να έχουμε τη δυνατότητα να τα απομονώσουμε.

Η C μας επιτρέπει να χειριζόμαστε πληροφορίες σε επίπεδο δυαδικών αριθμών μέσω ειδικών τελεστών οι οποίοι επιδρούν σε μεμονωμένα bit ενός ακέραιου αριθμού. Επιπρόσθετα, η γνώση της εσωτερικής αναπαράστασης των πληροφοριών βοηθάει στην επεξήγηση και την κατανόηση αρκετών λειτουργιών της γλώσσας. Οι δυαδικοί τελεστές (ή τελεστές bit – bitwise) επιτρέπουν στη C να χειρίζεται τους ακέραιους αριθμούς σε επίπεδο bit. Όλοι οι δυαδικοί τελεστές, εκτός από τον τελεστή ~ (συμπλήρωμα), έχουν δύο μέλη, ένα αριστερό και ένα δεξιό. Τα μέλη αυτά (μέλος-A, μέλος-B) πρέπει να είναι ακέραιες παραστάσεις, δηλαδή να αποδίδουν ακέραιες τιμές. Οι δυαδικοί τελεστές είναι έξι, και αναφέρονται στον πίνακα 4.1.



Σχήμα 4.1 Μετατροπή πληροφοριών σε δυαδικούς αριθμούς

& (bitwise AND)	μέλος-A & μέλος-B																																
Εκτελεί την πράξη AND στα αντίστοιχα bit των δύο μελών.	$1 \& 1 \rightarrow 1$ $1 \& 0 \rightarrow 0$ $0 \& 1 \rightarrow 0$ $0 \& 0 \rightarrow 0$																																
(bitwise OR)	μέλος-A μέλος-B																																
Εκτελεί την πράξη OR στα αντίστοιχα bit των δύο μελών.	$1 1 \rightarrow 1$ $1 0 \rightarrow 1$ $0 1 \rightarrow 1$ $0 0 \rightarrow 0$																																
^ (bitwise XOR)	μέλος-A ^ μέλος-B																																
Εκτελεί την πράξη XOR στα αντίστοιχα bit των δύο μελών. Το αποτέλεσμα είναι 1 όταν μόνο ένα από τα δύο μέλη έχει τιμή 1, σε κάθε άλλη περίπτωση το αποτέλεσμα είναι 0.	$1 \wedge 1 \rightarrow 0$ $1 \wedge 0 \rightarrow 1$ $0 \wedge 1 \rightarrow 1$ $0 \wedge 0 \rightarrow 0$																																
~ (συμπλήρωμα)	~μέλος-A																																
Αντιστρέφει τα bit του μέλους-A. <table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	1	0	1	0	0	1	0	1	↓	↓	↓	↓	↓	↓	↓	↓	0	1	0	1	1	0	1	0	$\sim 1 \rightarrow 0$ $\sim 0 \rightarrow 1$
7	6	5	4	3	2	1	0																										
1	0	1	0	0	1	0	1																										
↓	↓	↓	↓	↓	↓	↓	↓																										
0	1	0	1	1	0	1	0																										
<< (ολίσθηση αριστερά)	μέλος-A << μέλος-B																																
Μετακινεί προς τα αριστερά όλα τα bit του μέλους-A τόσες θέσεις όση η τιμή του μέλους-B. Σε κάθε μετατόπιση προς τα αριστερά το αριστερότερο bit χάνεται και το δεξιότερο bit παίρνει την τιμή 0. <table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>✗</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	✗	0	1	0	0	1	0	1	↓	↓	↓	↓	↓	↓	↓	↓	0	1	0	0	1	0	1	0	Π.χ. $b \ll 3$ μετακινεί τα bit της μεταβλητής b τρεις θέσεις αριστερά
7	6	5	4	3	2	1	0																										
✗	0	1	0	0	1	0	1																										
↓	↓	↓	↓	↓	↓	↓	↓																										
0	1	0	0	1	0	1	0																										
>> (ολίσθηση δεξιά)	μέλος-A >> μέλος-B																																
Μετακινεί προς τα δεξιά όλα τα bit του μέλους-A τόσες θέσεις όση η τιμή του μέλους-B. Σε κάθε μετατόπιση προς τα δεξιά το δεξιότερο bit χάνεται και το αριστερότερο bit παίρνει την τιμή 0. <table><tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>✗</td></tr><tr><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td><td>↓</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	7	6	5	4	3	2	1	0	1	0	1	0	0	1	0	✗	↓	↓	↓	↓	↓	↓	↓	↓	0	1	0	1	0	0	1	0	Π.χ. $b \gg 3$ μετακινεί τα bit της μεταβλητής b τρεις θέσεις δεξιά
7	6	5	4	3	2	1	0																										
1	0	1	0	0	1	0	✗																										
↓	↓	↓	↓	↓	↓	↓	↓																										
0	1	0	1	0	0	1	0																										

Πίνακας 4.1 Δυναδικοί τελεστές της C

Δείτε το επόμενο παράδειγμα:

```
#include <stdio.h>
int main(void)
{
    int a,b,c,d;
    a=166;
    b=176;
    c=a && b;
    d=a & b;
    printf("c=%d d=%d\n",c,d);
    return 0;
}
```

4_bitwise1.c**c=1 d=160**

Η λογική παράσταση **a && b** έχει αποτέλεσμα 1 (αληθές) επειδή τα μέλη **a** και **b** θεωρούνται αληθή αφού αποδίδουν τιμές διάφορες του 0 (166 και 176 αντίστοιχα). Αντιθέτως, η παράσταση **a & b** επιβάλλει την πράξη bit AND μεταξύ του 166 και του 176 με αποτέλεσμα 160 (Σχήμα 4.2).

Στο παράδειγμα που ακολουθεί, η λογική παράσταση **a||b** έχει αποτέλεσμα 1, ενώ η παράσταση **a|b** επιβάλλει την πράξη bit OR μεταξύ του 8 και του 4, με αποτέλεσμα 12.

```
#include <stdio.h>
int main(void)
{
    int a,b,c,d;
    a=8;
    b=4;
    c=a || b;
    d=a | b;
    printf("c=%d d=%d\n",c,d);
    return 0;
}
```

4_bitwise2.c**c=1 d=12**

Οι δυαδικοί τελεστές εκτελούν αριθμητικές, και όχι λογικές, πράξεις. Το αποτέλεσμα μιας δυαδικής πράξης μπορεί να είναι ένας οποιοσδήποτε αριθμός, ενώ το αποτέλεσμα μιας λογικής πράξης μπορεί να είναι μόνο το 1 (αληθής) ή το 0 (ψευδής).

Οι δυαδικοί τελεστές είναι ιδιαίτερα χρήσιμοι σε προγράμματα τα οποία ελέγχουν μικροελεγκτές και χρησιμοποιούνται ιδιαίτερα στη ρομποτική. Εκεί η επικοινωνία της συσκευής με το πρόγραμμα γίνεται συνήθως με ακέραιους αριθμούς (των 8, 16 ή 32 bit) των οποίων τα επιμέρους bit υποδηλώνουν διαφορετικές πληροφορίες. Στο πρόγραμμα [4_pontiki.c](#) που ακολουθεί, καθώς και στο παράδειγμα Π4.3, εξετάζονται τέτοιες περιπτώσεις.

& (bitwise AND)		&	7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
166 & 176 → 160			1 0 1 1 0 0 0 0	176		
			1 0 1 0 0 0 0 0	160		
(bitwise OR)			7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
166 176 → 182			1 0 1 1 0 0 0 0	176		
			1 0 1 1 0 1 1 0	182		
^ (bitwise XOR)		^	7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
166 ^ 176 → 22			1 0 1 1 0 0 0 0	176		
			0 0 0 1 0 1 1 0	22		
~ (συμπλήρωμα)		~	7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
~ 166 → 89			0 1 0 1 1 0 0 1	89		
<< (ολίσθηση αριστερά)		<<	7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
166 << 2 → 664			15 14 13 12 11 10 9 8	0 0 0 0 0 1 0 1	664	
>> (ολίσθηση δεξιά)		>>	7 6 5 4 3 2 1 0	1 0 1 0 0 1 1 0	166	
166 >> 2 → 41				0 0 1 0 1 0 0 1	41	

Σχήμα 4.2 Παραδείγματα χρήσης των δυαδικών (bitwise) τελεστών της C

- ❏ Οι δυαδικοί τελεστές εφαρμόζονται μόνο σε ακέραια μεγέθη
- ❏ Δεν πρέπει να γίνεται σύγχυση των λογικών τελεστών (&&, ||) με τους δυαδικούς τελεστές (&, |). Οι δυαδικοί τελεστές εκτελούν αριθμητικές, και όχι λογικές, πράξεις.
- ❏ Προσέξτε την εφαρμογή του τελεστή XOR στο παράδειγμα του Σχήματος 4.2. Το αποτέλεσμα της πράξης $166 \wedge 176$ είναι 22. Αν τώρα εφαρμόσουμε την πράξη XOR στο 22 με τον ίδιο αριθμό, το 176, θα προκύψει πάλι ο αρχικός αριθμός το 166 ($22 \wedge 176 \rightarrow 166$). Αυτή η ιδιότητα είναι χρήσιμη σε περιπτώσεις κωδικοποίησης και θα χρησιμοποιηθεί αργότερα σε αντίστοιχα παραδείγματα.

Συχνά λάθη

- ❏ Οι τελεστές ++ και — χρησιμοποιούνται πολλές φορές λανθασμένα. Μια προσεκτική ματιά στον τρόπο λειτουργίας τους και στα παραδείγματα θα σας βοηθήσει να τους χρησιμοποιείτε σωστά.
- ❏ Ο τελεστής υπολοίπου % εφαρμόζεται μόνο σε ακέραιο τύπο δεδομένων. Αν προσπαθήσουμε να τον εφαρμόσουμε σε δεδομένα τύπου float ή double, ο μεταγλωττιστής θα εμφανίσει ένα σχετικό μήνυμα σφάλματος.
- ❏ Πολλές φορές χρησιμοποιούνται εσφαλμένα οι τελεστές bit αντί των αντίστοιχων λογικών τελεστών. Προσέξτε ότι το λογικό και (AND) είναι ο τελεστής &&, και όχι ο &! Οι λογικοί τελεστές και (AND) και ή (OR) αποτελούνται από δύο χαρακτήρες, ενώ οι αντίστοιχοι τελεστές bit από έναν.

Ανασκόπηση Κεφαλαίου 4

- Ο τύπος δεδομένων int αφορά ακέραιους αριθμούς. Οι μεταβλητές τύπου int δεσμεύουν 4 byte.
- Κατά τη δήλωσή της, μια μεταβλητή μπορεί να πάρει ταυτόχρονα και αρχική τιμή, π.χ. `int a=5;`
- Ο τελεστής ++ αυξάνει μια μεταβλητή τύπου int κατά ένα, ενώ ο τελεστής -- τη μειώνει κατά 1.
- Όταν ο τελεστής ++ ή – βρίσκεται πριν από τη μεταβλητή, τότε πρώτα γίνεται η πράξη (αύξηση ή μείωση), και μετά επιστρέφεται η νέα τιμή της μεταβλητής (μετά από την αύξηση ή μείωση).
- Όταν ο τελεστής ++ ή – βρίσκεται μετά από τη μεταβλητή, τότε πρώτα επιστρέφεται η τιμή της μεταβλητής (η παλιά τιμή, πριν από την πράξη), και μετά αυξάνεται ή μειώνεται το περιεχόμενο της μεταβλητής.
- Χρησιμοποιούμε τον τελεστή υπολοίπου % για να διαπιστώσουμε αν ένας ακέραιος είναι πολλαπλάσιο ενός άλλου ακέραιου αριθμού. Για παράδειγμα, αν `a%2==0`, τότε ο a είναι ζυγός (πολλαπλάσιο του 2), διαφορετικά είναι μονός.
- Η συνάρτηση rand() επιστρέφει έναν (ψευδο)τυχαίο ακέραιο αριθμό.
- Οι δυαδικοί τελεστές μάς δίνουν τη δυνατότητα να απομονώνουμε και να επεξεργαζόμαστε τα επιμέρους bit ενός ακέραιου αριθμού.
- Ο δυαδικός τελεστής XOR χρησιμοποιείται συχνά σε προγράμματα κωδικοποίησης.

Ασκήσεις Κεφαλαίου 4

4.1 Στις επόμενες τέσσερις προτάσεις υποθέτουμε ότι το **x** έχει τιμή 100 πριν από την εκτέλεση κάθε παράστασης. Συμπληρώστε την τιμή του **x** και την τιμή της παράστασης μετά από την εκτέλεση της κάθε πρότασης. ★

Πρόταση	Τιμή του x	Τιμή της παράστασης
x++;		
++x;		
x--;		
--x;		
x-x;		

4.2 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

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

```
int main(void)
```

```
{
```

```
    int a,b,aa,bb,x,y;
```

```
    x=y=100;
```

```
    a=++x;
```

```
    b=y++;
```

```
    aa=++x;
```

```
    bb=y++;
```

```
    printf("Η τιμή του a είναι %d\n",a);
```

```
    printf("Η τιμή του b είναι %d\n",b);
```

```
    printf("Η τιμή του aa είναι %d\n",aa);
```

```
    printf("Η τιμή του bb είναι %d\n",bb);
```

```
    return 0;
```

```
}
```

4.3 Έστω ότι η τιμή του **y** είναι 100 πριν από την εκτέλεση **κάθε** μίας από τις επόμενες παραστάσεις. Ποιες θα είναι οι τιμές των μεταβλητών **x** και **y** μετά από την εκτέλεση κάθε παράστασης; ★★

Παράσταση	Τιμή του x	Τιμή του y
x = y;		
x = --y * 4;		
x = y = y++;		
x = y == 100;		
x = y == y++;		
x = y == ++y;		

4.4 Δίνεται το επόμενο τμήμα κώδικα:

```
int x,y,z;
```

```
z=22;
```

Κεφάλαιο

Ο τύπος δεδομένων char

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό ξεκινά με μια περιγραφή του ιδιαίτερου τρόπου με τον οποίο η γλώσσα C χειρίζεται τους χαρακτήρες.

Παρουσιάζονται οι συναρτήσεις χειρισμού χαρακτήρων και περιγράφονται σε μεγαλύτερο βάθος οι ιδιαιτερότητες των συναρτήσεων εισόδου για την ανάγνωση χαρακτήρων από το πληκτρολόγιο.

Επίσης, ένα από τα βασικά σημεία του κεφαλαίου είναι ο τρόπος με τον οποίο η C χειρίζεται τις συμβολοσειρές (σύνολα χαρακτήρων).

Εισαγωγή

Το κεφάλαιο αυτό εισάγει και αναλύει τον τρόπο χρήσης χαρακτήρων (characters) στη γλώσσα C. Εξετάζουμε τον τρόπο με τον οποίο η C χειρίζεται τις σταθερές και τις μεταβλητές τύπου **char**. Ακόμη, γίνεται αναφορά σε συναρτήσεις που χρησιμοποιούνται για τον χειρισμό χαρακτήρων.

Σταθερές τύπου char

Μια σταθερά τύπου **char** δεν είναι τίποτε άλλο από **έναν χαρακτήρα** που περικλείεται σε **μονά** εισαγωγικά. Για παράδειγμα, οι παρακάτω εκφράσεις:

'A'

'1'

'*'

είναι παραδείγματα σταθερών τύπου **char**.

Κωδικοποίηση χαρακτήρων σε αριθμούς

Όπως κάθε πληροφορία στον Η/Υ, η εσωτερική αναπαράσταση μιας σταθεράς (αλλά και μιας μεταβλητής) τύπου **char** είναι ένας αριθμός. Στην περίπτωση χαρακτήρων, η μετατροπή αυτή δεν είναι μαθηματική, αλλά μια προκαθορισμένη κωδικοποίηση κάθε χαρακτήρα σε έναν ακέραιο αριθμό. Ο αριθμός αυτός λέγεται κωδικός ASCII¹³ του χαρακτήρα και μπορεί να υπολογιστεί με βάση τον Πίνακα 5.1.

Ο κωδικός ενός χαρακτήρα προκύπτει από το άθροισμα του αριθμού της γραμμής και της στήλης που βρίσκεται στον πίνακα. Έτσι, ο χαρακτήρας 'Γ' έχει κωδικό 130 (128+2), ενώ ο χαρακτήρας 'W' έχει κωδικό 87 (80+7). Η τυπική κωδικοποίηση ASCII περιλαμβάνει τους 128 πρώτους χαρακτήρες, με κωδικούς από το 0 έως το 127. Οι χαρακτήρες από το 128 μέχρι το 255 αποτελούν επέκταση του πρότυπου πίνακα χαρακτήρων, και διαφέρουν από σύστημα σε σύστημα και από γλώσσα σε γλώσσα. Οι χαρακτήρες με κωδικούς από 0 μέχρι 31 δεν είναι εκτυπώσιμοι χαρακτήρες, και κάποιοι από αυτούς χρησιμοποιούνται ως χαρακτήρες ελέγχου. Χαρακτηριστικό παράδειγμα χαρακτήρων ελέγχου αποτελούν οι χαρακτήρες διαφυγής της C.

Για παράδειγμα, η πρόταση

```
printf("%c %d\n", 'A', 'A');
```

θα εμφανίσει το A και το 65 (ο κωδικός ASCII του λατινικού A). Δηλαδή, θα εμφανίσει τον χαρακτήρα 'A' ως χαρακτήρα (%c) αλλά και ως ακέραιο αριθμό (%d).

¹³ Το ακρωνύμιο **ASCII** προέρχεται από τα αρχικά **American Standard Code for Information Interchange** (Αμερικανικός Πρότυπος Κώδικας για την Ανταλλαγή Πληροφοριών).

ASCII	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
128	A	B	Γ	Δ	E	Z	H	Θ	I	K	Λ	M	N	Ξ	O	Π
144	P	Σ	T	Υ	Φ	X	Ψ	Ω	α	β	γ	δ	ε	ζ	η	θ
160	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	ς	τ	υ	φ	χ	ψ
176	░	▒	▓		└	┌	┐	┌	┐	┐	┐	┐	┐	┐	┐	┐
192	└	└	└	└	└	└	└	└	└	└	└	└	└	└	└	└
208	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬	▬
224	ω	ά	έ	ή	ϊ	ί	ό	ύ	ϋ	ώ	Α	Έ	Η	Ί	Ό	Υ
240	Ω	±	≥	≤	İ	ÿ	÷	≈	°	·	·	√	n	²	■	

Πίνακας 5.1 Πίνακας κωδικών ASCII

Επομένως, οι ακόλουθες παραστάσεις δεν είναι καθόλου παράξενες:

`5 + '*'` → επιστρέφει την τιμή 47 (5 + 42 που είναι ο κωδικός ASCII του '*').

`printf("%d %c\n", 'A'+1, 'A'+1);` → εμφανίζει στην οθόνη το 66 και το B.

Το 66 εμφανίζεται διότι είναι το αποτέλεσμα της παράστασης `'A'+1` και ο συνδυασμός `%d` υποδεικνύει στην `printf()` να το εμφανίσει ως ακέραιο αριθμό. Το B εμφανίζεται διότι ο συνδυασμός `%c` υποδεικνύει στην `printf()` να εμφανίσει το αποτέλεσμα της παράστασης `'A'+1` (το 66) ως χαρακτήρα. Έτσι, εμφανίζεται ο λατινικός χαρακτήρας B, ο οποίος έχει κωδικό ASCII 66.

Η C χειρίζεται τους χαρακτήρες ως ακέραιους αριθμούς. Αυτό σημαίνει ότι μπορούν να μετέχουν σε αριθμητικές παραστάσεις.

Χαρακτήρες διαφυγής

Οι χαρακτήρες διαφυγής (escape characters) αναφέρονται και ως ακολουθίες διαφυγής (escape sequences). Έχουμε ήδη συναντήσει μερικούς από χαρακτήρες διαφυγής σε προηγούμενα κεφάλαια, καθώς και τον τρόπο χρήσης τους μέσω της συνάρτησης `printf()`. Οι χαρακτήρες αυτοί δεν εμφανίζουν τίποτα στην οθόνη, αλλά ελέγχουν ορισμένες λειτουργίες της. Οι χαρακτήρες διαφυγής της C είναι οι ακόλουθοι:

- `\n` new line (νέα γραμμή)
- `\b` backspace (μία θέση πίσω)
- `\f` form feed (νέα σελίδα)

`\r` carriage return (επαναφορά κεφαλής)
`\t` tab (στηλοθέτης)
`\\` backslash (ανάποδη κάθετος, \)
`'` απλό εισαγωγικό (')
`"` διπλό εισαγωγικό (")
`\0` χαρακτήρας τερματισμού (null character - byte 00000000)

Σύμφωνα με τα παραπάνω, η επόμενη κλήση της συνάρτησης `printf()` έχει το αποτέλεσμα που εμφανίζεται στο διπλανό πλαίσιο:

Οι
"χαρακτήρες διαφυγής"
είναι 9

```
printf("Οι\\n\"χαρακτήρες διαφυγής\"\\nείναι 9\\n");
```

Παρατηρούμε ότι, επειδή τα διπλά εισαγωγικά (") έχουν ειδικό νόημα για την `printf()`, δεν μπορούν να συμπεριληφθούν στο αλφαριθμητικό μορφοποίησης ως κανονικός χαρακτήρας. Για τον λόγο αυτόν πρέπει να χρησιμοποιηθεί ο χαρακτήρας διαφυγής `\"`.

Ο χαρακτήρας του **διαστήματος**, ο χαρακτήρας στηλοθέτη **tab**, και ο χαρακτήρας **αλλαγής γραμμής**, λέγονται **χαρακτήρες λευκού διαστήματος** (whitespace characters). Έτσι, όταν θα αναφερόμαστε σε χαρακτήρες λευκού διαστήματος στο υπόλοιπο του βιβλίου, θα εννοούμε οποιονδήποτε από τους τρεις αυτούς χαρακτήρες.

Κωδικοποίηση Unicode

Με βάση τον επεκτεταμένο πίνακα κωδικοποίησης ASCII μπορούν να κωδικοποιηθούν το πολύ 256 χαρακτήρες, συμπεριλαμβανομένων και των χαρακτήρων ελέγχου. Είναι λοιπόν προφανές ότι είναι αδύνατον να συμπεριληφθούν κωδικοί για αλφάβητα άλλων γλωσσών εκτός της Αγγλικής, αλλά και γλωσσών με περισσότερους χαρακτήρες στο αλφάβητό τους (π.χ. Κινέζικα). Για την αντιμετώπιση αυτού του προβλήματος χρησιμοποιείται το πρότυπο Unicode, το οποίο παρέχει ένα μόνο σύνολο χαρακτήρων με τη δυνατότητα αποθήκευσης πολλών διαφορετικών αλφάβητων. Το πρότυπο Unicode υποστηρίζει διάφορους τρόπους κωδικοποίησης χαρακτήρων οι οποίοι απαιτούν περισσότερα από ένα byte για κάθε χαρακτήρα. Ο προεπιλεγμένος τύπος χαρακτήρων `char` της C υποστηρίζει χαρακτήρες που κωδικοποιούνται με ένα μόνο byte. Τι γίνεται λοιπόν με τους χαρακτήρες που απαιτούν περισσότερα byte; Σύμφωνα με τα τελευταία πρότυπα της γλώσσας, η C μπορεί να υποστηρίξει χαρακτήρες μεγαλύτερου μήκους, 2 και 4 byte, μέσω εξειδικευμένων βιβλιοθηκών.

Επειδή το παρόν βιβλίο προσανατολίζεται κυρίως στην εκπαιδευτική διαδικασία εκμάθησης της C, δεν θα ασχοληθούμε καθόλου με χαρακτήρες Unicode, οι οποίοι χρησιμοποιούνται συνήθως σε εφαρμογές με παραθυρικό περιβάλλον.

Η scanf() και τα μικρά της προβλήματα! ☹

Η συνάρτηση `scanf()`, με τον συνδυασμό `%c` στο αλφαριθμητικό μορφοποίησης, χρησιμοποιείται για την ανάγνωση ενός χαρακτήρα. Για παράδειγμα, η πρόταση:

```
scanf ("%c", &ch) ;
```

θα περιμένει να πληκτρολογηθεί ένας χαρακτήρας, τον οποίο θα καταχωρίσει στη μεταβλητή χαρακτήρα `ch`.

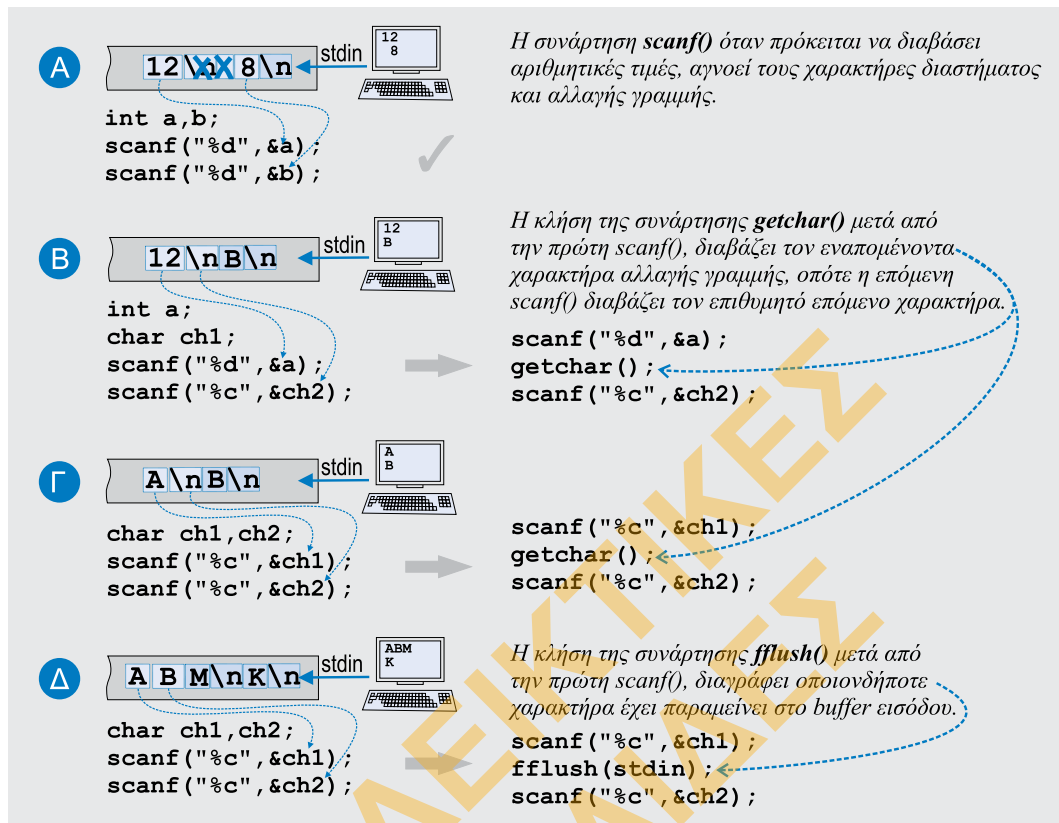
Η `scanf()` αποτελεί αιτία πολλών πονοκεφάλων για τους νέους προγραμματιστές, επειδή ορισμένες φορές συμπεριφέρεται περίεργα και απρόβλεπτα. Όμως, αν αναλύσουμε πιο διεξοδικά τη λειτουργία της, αυτή η απρόβλεπτη συμπεριφορά της γίνεται κατανοητή και διαχειρίσιμη!

Αν διαβάσετε για πρώτη φορά αυτό το βιβλίο και δεν έχετε πείρα στον προγραμματισμό, θα σας συμβούλευα να παραλείψετε τώρα την παρούσα ενότητα και να επανέλθετε αργότερα όταν θα αρχίζετε να γράφετε τα δικά σας προγράμματα. Αν όμως θέλετε να τη διαβάσετε, δώστε προσοχή στα ακόλουθα:

- Οι χαρακτήρες που πληκτρολογούνται από τον χρήστη εισέρχονται σε μια περιοχή προσωρινής αποθήκευσης (buffer εισόδου) από την οποία διαβάζονται. Το πληκτρολόγιο θεωρείται το **καθιερωμένο ρεύμα εισόδου** (standard input stream) και συμβολίζεται με το προσδιοριστικό `stdin`.
- Η συνάρτηση `scanf()`, όταν διαβάζει αριθμητικές τιμές (`int`, `float` ή `double`) αγνοεί όλους τους χαρακτήρες λευκού διαστήματος (κενά διαστήματα, χαρακτήρες αλλαγής γραμμής, και στηλοθέτες) που προηγούνται της τιμής.
- Αντιθέτως, όταν η `scanf()` διαβάζει χαρακτήρες (`char`), δεν αγνοεί τους χαρακτήρες λευκού διαστήματος που προηγούνται του χαρακτήρα.
- Σε κάθε περίπτωση, η συνάρτηση `scanf()` αφήνει στο buffer εισόδου τον τελευταίο χαρακτήρα αλλαγής γραμμής (το `enter`) που πληκτρολογεί ο χρήστης μετά από τις τιμές που δίνει. Ο χαρακτήρας αυτός θα διαβαστεί από την επόμενη `scanf()`.

Τα παραδείγματα που ακολουθούν θα μας βοηθήσουν να κατανοήσουμε περισσότερο τα προβλήματα που είναι πιθανό να συναντήσουμε. Στο Σχήμα 5.2 εμφανίζονται τέσσερις διαφορετικές περιπτώσεις:

A. Στην περίπτωση αυτή δεν υπάρχει πρόβλημα. Και στις δύο κλήσεις της, η συνάρτηση `scanf()` περιμένει να διαβάσει έναν ακέραιο αριθμό. Έστω ότι πληκτρολογείται ο αριθμός 12, ένα `<enter>`, ένα κενό, ο αριθμός 8 και τέλος ένα ακόμα `<enter>`. Η πρώτη `scanf()` θα διαβάσει το 12 και θα το καταχωρίσει στη μεταβλητή `a`. Η επόμενη `scanf()` θα αγνοήσει τόσο το `<enter>` όσο και το διάστημα που ακολουθεί, θα διαβάσει το 8, και θα το καταχωρίσει στη μεταβλητή `b`. Στο buffer εισόδου θα παραμείνει ο χαρακτήρας αλλαγής γραμμής (το τελευταίο `<enter>`).



Σχήμα 5.2 Τα προβλήματα και οι λύσεις στη χρήση της `scanf()`

B. Εδώ αρχίζουν τα προβλήματα! Στην πρώτη κλήση της, η συνάρτηση `scanf()` περιμένει να διαβάσει έναν ακέραιο αριθμό και στη δεύτερη έναν χαρακτήρα. Ας υποθέσουμε ότι πληκτρολογούνται ο αριθμός 12, ένα <enter>, ο χαρακτήρας 'B', και άλλο ένα <enter>. Η πρώτη `scanf()` θα διαβάσει το 12, θα το καταχωρίσει στη μεταβλητή `a`, αλλά θα αφήσει το χαρακτήρα αλλαγής γραμμής ('\n') στον buffer εισόδου. Η επόμενη `scanf()` θα διαβάσει το χαρακτήρα αλλαγής γραμμής και θα καταχωρίσει αυτόν στη μεταβλητή `ch`, αντί του χαρακτήρα 'B', όπως θα ήταν αναμενόμενο. Στον buffer εισόδου θα παραμείνει ο χαρακτήρας 'B' καθώς και ο χαρακτήρας αλλαγής γραμμής (το τελευταίο <enter>).

Η λύση βρίσκεται στην κλήση της `getchar()` μετά από την πρώτη `scanf()`. Η `getchar()` θα διαβάσει και θα απομακρύνει το χαρακτήρα αλλαγής γραμμής από το ρεύμα εισόδου, και η επόμενη `scanf()` θα διαβάσει κανονικά το χαρακτήρα 'B'.

Γ. Και εδώ γίνεται κάτι παρόμοιο! Στην πρώτη κλήση της η συνάρτηση `scanf()` περιμένει να διαβάσει ένα χαρακτήρα, όπως και στη δεύτερη. Ας υποθέσουμε ότι πληκτρολογούνται οι χαρακτήρες: 'A', <enter>, 'B', και <enter>. Η πρώτη `scanf()` θα διαβάσει το

Κεφάλαιο

Float, double, και άλλοι τύποι δεδομένων



Στο κεφάλαιο αυτό παρουσιάζονται οι υπόλοιποι τύποι δεδομένων της C. Αρχικά γίνεται αναφορά στον τρόπο με τον οποίο η γλώσσα χειρίζεται πραγματικούς αριθμούς, δηλαδή αριθμούς με δεκαδικά ψηφία.

Περιγράφεται ο τρόπος με τον οποίο χρησιμοποιούνται οι συναρτήσεις εξόδου και εισόδου με πραγματικά δεδομένα.

Επίσης, παρουσιάζεται ο λογικός τύπος δεδομένων, ο οποίος απουσίαζε από τα αρχικά πρότυπα της γλώσσας, καθώς και παραλλαγές των βασικών τύπων που υποστηρίζονται από διάφορους μεταγλωττιστές της.

Οι τύποι δεδομένων float και double

Ο χειρισμός των δεδομένων κινητής υποδιαστολής (δηλαδή των πραγματικών αριθμών) στη γλώσσα C γίνεται μέσω δύο βασικών τύπων της: του τύπου **float** και του τύπου **double**. Ο τύπος **double** χρησιμοποιείται όταν θέλουμε μεγαλύτερη ακρίβεια στα δεκαδικά ψηφία (double precision). Επίσης, όπως θα δούμε στη συνέχεια, η εσωτερική αναπαράσταση των δύο τύπων είναι διαφορετική, με τον τύπο **double** να καταλαμβάνει διπλάσιο χώρο στη μνήμη του Η/Υ.

Σταθερές τύπου float και double

Εξ ορισμού, στη γλώσσα C μια σταθερά κινητής υποδιαστολής (δηλαδή ένας πραγματικός αριθμός) μετατρέπεται σε **double**. Για να θεωρηθεί τύπου **double**, ένας αριθμός πρέπει απαραίτητα να περιέχει τον χαρακτήρα της υποδιαστολής (.). Έτσι, οι αριθμοί: 123.45, 123.0, -123., 0.1

είναι παραδείγματα σταθερών τύπου **double**. Όμως, όπως θα δούμε στη συνέχεια, μπορεί να προσδιοριστεί ρητά ο τύπος μιας σταθεράς κινητής υποδιαστολής όταν προστεθεί το κατάλληλο επίθημα (suffix). Για παράδειγμα, η σταθερά 123.4F αναφέρεται στον πραγματικό αριθμό 123.4 τύπου **float**, και όχι **double**.

Μεταβλητές τύπου float και double

Υπάρχουν δύο βασικοί τύποι μεταβλητών κινητής υποδιαστολής (floating point), ο τύπος **float** και ο τύπος **double**. Οι μεταβλητές τύπου **float** καταλαμβάνουν 4 byte, ενώ οι τύπου **double** καταλαμβάνουν 8 byte. Οι μεταβλητές τύπου **double** έχουν μεγαλύτερη ακρίβεια δεκαδικών ψηφίων.

Οι μεταβλητές τύπου **float** δηλώνονται με τον ακόλουθο τρόπο:

float *όνομα_μεταβλητής1, όνομα_μεταβλητής2,;*

double *όνομα_μεταβλητής1, όνομα_μεταβλητής2,;*

long float *όνομα_μεταβλητής1, όνομα_μεταβλητής2,;*

Ο τύπος **long float** είναι ισοδύναμος με τον **double**.

Στο πρόγραμμα που ακολουθεί δηλώνονται δύο μεταβλητές **f** και **d** τύπου **float** και **double**, αντίστοιχα, στις οποίες ανατίθενται αρχικές τιμές.

```
#include <stdio.h>
int main(void)
{
    float f=5.3;
    double d=12.123456789;
    ....
    return 0;
}
```

Χρήση τελεστών σε δεδομένα τύπου κινητής υποδιαστολής

Οι διαθέσιμοι τελεστές για πράξεις με δεδομένα τύπου κινητής υποδιαστολής (floating point) είναι πολύ περιορισμένοι σε σχέση με τους τελεστές που εφαρμόζονται σε δεδομένα τύπου `int` και `char`. Οι τελεστές που εφαρμόζονται σε δεδομένα τύπου `float` (ή `double`) αναφέρονται στον διπλανό πίνακα.

Τελεστής	Πράξη
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
++	Μοναδιαία αύξηση
--	Μοναδιαία μείωση

- ❏ Ο αριθμητικός τελεστής `%` δεν εφαρμόζεται σε δεδομένα κινητής υποδιαστολής παρά μόνο σε ακέραια.
- ❏ Επίσης, στα αρχικά πρότυπα της γλώσσας, η εφαρμογή των τελεστών `++` και `--` επιτρεπόταν μόνο σε μεταβλητές ακέραιου τύπου (`int` και `char`). Μερικοί μεταγλωττιστές της C έχουν ακόμα αυτόν τον περιορισμό.

Η χρήση της `printf()` με δεδομένα κινητής υποδιαστολής

Ας δούμε το επόμενο πρόγραμμα, του οποίου το αποτέλεσμα φαίνεται στο διπλανό πλαίσιο:

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a=21.234;
    b=5467.1;
    printf("%f\n",a);
    printf("%f\n",b);
    return 0;
}
```

21.234
5467.1

Παρατηρούμε ότι, αν και τα αποτελέσματα εμφανίζονται σωστά, μάλλον θα προτιμούσαμε οι χιλιάδες να εμφανίζονται κάτω από τις χιλιάδες, οι εκατοντάδες κάτω από τις εκατοντάδες, κ.ο.κ. Για τη μορφοποίηση της εξόδου αριθμών, η ακολουθία `%f` στο αλφαριθμητικό ελέγχου της `printf()` μπορεί να διαμορφωθεί ως εξής:

`%n.mf`

όπου `n` είναι ο συνολικός αριθμός των θέσεων που θα καταλάβει ο αριθμός (μαζί με την υποδιαστολή) και `m` είναι ο αριθμός των θέσεων μετά από την υποδιαστολή. Έτσι, το `%8.3f` σημαίνει ότι ο αριθμός θα εμφανιστεί σε 8 θέσεις, από τις οποίες οι 3 θα είναι για το δεκαδικό μέρος. Για μορφοποιημένη έξοδο, το προηγούμενο πρόγραμμα θα μπορούσε να τροποποιηθεί ως εξής:

```
#include <stdio.h>
int main(void)
{
    float a,b;
    a=21.234;
    b=5467.1;
    printf("%8.3f\n",a);
    printf("%8.3f\n",b);
    return 0;
}
```

21.234
5467.100

Χρήση της συνάρτησης scanf() με δεδομένα τύπου float

Η `scanf()` χρησιμοποιείται για την είσοδο πραγματικών αριθμών από το πληκτρολόγιο. Για παράδειγμα, το επόμενο πρόγραμμα ζητάει και εμφανίζει δύο πραγματικούς αριθμούς.

```
#include <stdio.h>
int main(void)
{
    float a;
    double b;
    printf("Δώσε αριθμό:");
    scanf("%f",&a); // Διαβάζει έναν αριθμό τύπου float και τον καταχωρίζει στη μεταβλητή a
    printf("Δώσε αριθμό:");
    scanf("%lf",&b); // Διαβάζει έναν αριθμό τύπου double και τον καταχωρίζει στη μεταβλητή b
    printf("Έδωσες %f και %f\n",a,b);
    return 0;
}
```

6_float_scanf.c

🔍 Παρατηρούμε ότι, όταν πρόκειται να διαβαστεί ένας αριθμός διπλής ακρίβειας (τύπου `double`), στο αλφαριθμητικό ελέγχου της συνάρτησης `scanf()`, η ακολουθία `%f` αντικαθίσταται από την `%lf`.

Οι συναρτήσεις pow() και sqrt()

Η C δεν διαθέτει τελεστή ύψωσης σε δύναμη, όπως άλλες γλώσσες προγραμματισμού, αλλά ούτε φυσικά και σύμβολο τετραγωνικής ρίζας. Η ύψωση σε δύναμη γίνεται μέσω της συνάρτησης βιβλιοθήκης `pow()` και ο υπολογισμός της τετραγωνικής ρίζας με τη συνάρτηση `sqrt()`. Η συνάρτηση `pow()` έχει δύο παραμέτρους (τη βάση και τον εκθέτη) και επιστρέφει μια τιμή τύπου `double`. Η συνάρτηση `sqrt()` έχει μία παράμετρο και επιστρέφει επίσης μια τιμή τύπου `double`.

Το πρόγραμμα που ακολουθεί υπολογίζει και εμφανίζει στην οθόνη την τιμή της παραστάσης 4^5 καθώς και την τετραγωνική ρίζα του 200.

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

6_pow_sqrt.c

```

int main(void)
{
    double c,t;
    c=pow(4,5); // κλήση της συνάρτησης pow(), υπολογισμός του 45
    printf("4 εις την 5 = %f\n",c);
    t=sqrt(200); // κλήση της συνάρτησης sqrt(), υπολογισμός της τετραγωνικής ρίζας του 200
    printf("Η τετραγωνική ρίζα του 200 είναι %f\n",t);
    return 0;
}

```

Παρά το γεγονός ότι στο προηγούμενο πρόγραμμα οι παράμετροι που μεταβιβάζονται στις συναρτήσεις `pow()` και `sqrt()` είναι ακέραιοι αριθμοί, οι συναρτήσεις επιστρέφουν εξ ορισμού αριθμούς τύπου `double`, και θα πρέπει να καταχωριστούν σε κατάλληλες μεταβλητές. Τόσο η συνάρτηση `pow()` όσο και η συνάρτηση `sqrt()` ορίζονται στο αρχείο κεφαλίδας `math.h`, το οποίο θα πρέπει να συμπεριληφθεί στο πρόγραμμα με την οδηγία `#include`. Η λογική των *αρχείων κεφαλίδων* (header files), καθώς και η φιλοσοφία χρήσης των συναρτήσεων βιβλιοθήκης, αναλύονται διεξοδικά στο Κεφάλαιο 9.

Βασικοί τύποι και παραλλαγές τους

Ο πίνακας που ακολουθεί περιλαμβάνει όλους τους βασικούς τύπους της C, καθώς και παραλλαγές τους. Στον πίνακα αναφέρεται το μέγεθος σε byte που καταλαμβάνει ο κάθε τύπος, καθώς και το εύρος των αριθμών που μπορεί να απεικονίσει.

Τύπος	Περιγραφή	Byte	Εύρος
char	Ένας χαρακτήρας ASCII	1	-128 ~ 127 unsigned: 0 ~ 255
int	Ακέραιος	4	-2.147.483.648 ~ 2.147.483.647 unsigned: 0 ~ 4.294.967.295
short int	Ακέραιος μικρού μήκους	2	-32.768 ~ +32.767 unsigned: 0 ~ 65.535
long int	Ακέραιος μεγάλου μήκους	4	-2.147.483.648 ~ 2.147.483.647 unsigned: 0 ~ 4.294.967.295
long long int	Ακέραιος πολύ μεγάλου μήκους	8	-9.223.372.036.854.775.808 ~ 9.223.372.036.854.775.807 unsigned: 0 ~ 18.446.744.073.709.551.615
float	Πραγματικός απλής ακρίβειας	4	-3.4x10 ³⁸ ~ +3.4x10 ³⁸ (6~7 δεκαδικά ψηφία)
double	Πραγματικός διπλής ακρίβειας	8	-1.7 x10 ³⁰⁸ ~ 1.7 x10 ³⁰⁸ (14~15 δεκαδικά ψηφία)
long double	Πραγματικός πολύ μεγάλης ακρίβειας	12	Πολύ μεγαλύτερο εύρος και ακρίβεια

Το πρόθεμα **unsigned** μπροστά από τους ακέραιους τύπους έχει ως αποτέλεσμα να μην επιτρέπεται η καταχώριση αρνητικών αριθμών στις αντίστοιχες μεταβλητές, αλλά μόνο θετικών με διπλάσια όμως απόλυτη τιμή.

Εκτός από τους βασικούς τύπους και τις παραλλαγές τους, όσο εξελίσσονται τα πρότυπα της γλώσσας υποστηρίζονται όλο και περισσότεροι και πιο εξειδικευμένοι τύποι δεδομένων. Επίσης, αρκετοί μεταγλωττιστές υποστηρίζουν επιπλέον τύπους δεδομένων που δεν αναφέρονται σε κανένα πρότυπο.

Απόλυτος προσδιορισμός τύπου αριθμητικών σταθερών

Η C μας δίνει τη δυνατότητα να προσδιορίσουμε τον τύπο στον οποίο θέλουμε να μετατραπεί μια σταθερά, παρακάμπτοντας με αυτό τον τρόπο την αυτοματοποιημένη διαδικασία μετατροπής τύπου που υποστηρίζει η γλώσσα. Αυτό γίνεται όταν αμέσως μετά από τη σταθερά ακολουθεί ένα επίθημα (suffix) του τύπου. Στον πίνακα που ακολουθεί αναφέρονται τα επιθήματα τύπων που χρησιμοποιούνται στη C.

Τύπος δεδομένων	Επίθημα τύπου	Παραδείγματα
<i>int</i>		12, 2456, -23
<i>long int</i>	<i>L</i>	12L, 2456L, -23L
<i>unsigned int</i>	<i>U</i>	12U, 2456U
<i>unsigned long int</i>	<i>UL</i>	12UL, 2456UL
<i>float</i>	<i>F</i>	12.34F, 0.23F, 100.0F
<i>double</i>		12.34, 0.23, 100.0
<i>long double</i>	<i>L</i>	12.34L, 0.23L, 100.0L

🔍 Παρατηρούμε ότι το επίθημα τύπου έχει διαφορετική σημασία ανάλογα με τη σταθερά που προηγείται. Π.χ. η σταθερά 10L μετατρέπεται σε long int ενώ η 10.0L σε long double.

Δεκαεξαδικές σταθερές

Μέχρι τώρα, οι ακέραιες αριθμητικές σταθερές που χρησιμοποιήσαμε ανήκουν στο δεκαδικό σύστημα αρίθμησης, το οποίο ως βάση του έχει τον αριθμό 10.

Υπάρχουν όμως περιπτώσεις (κυρίως όταν προγραμματίζουμε συστήματα ή συσκευές σε πολύ χαμηλό επίπεδο) στις οποίες θα μας διευκόλυνε να χρησιμοποιούσαμε ακέραιες σταθερές στο δεκαεξαδικό σύστημα αρίθμησης.

Το δεκαεξαδικό σύστημα αρίθμησης έχει ως βάση τον αριθμό 16 (αντί του 10), και περιλαμβάνει 16 ψηφία: Τα ψηφία από το 0 μέχρι το 9 και τα γράμματα από το A μέχρι το F (τα οποία χρησιμοποιούνται για τους δεκαδικούς αριθμούς από το 10 μέχρι το 15).

Μια ακέραια δεκαεξαδική σταθερά πρέπει να αρχίζει με 0x (μηδέν και λατινικό x).

0xFF → αντιστοιχεί στον δεκαδικό αριθμό 255 ($15 \cdot 16^1 + 15 \cdot 16^0$)

0x1A → αντιστοιχεί στον δεκαδικό αριθμό 26 ($1 \cdot 16^1 + 10 \cdot 16^0$)

Οι δεκαεξαδικοί αριθμοί χρησιμοποιούνται πολύ συχνά για τον έλεγχο συσκευών σε επίπεδο δυαδικών αριθμών, αφού ένας δεκαεξαδικός αριθμός μπορεί να μετατραπεί άμεσα και εύκολα στον αντίστοιχο δυαδικό του. Ο διπλάνος πίνακας δείχνει τα 16 ψηφία του δεκαεξαδικού συστήματος αρίθμησης, με τον αντίστοιχο δεκαδικό και δυαδικό αριθμό στον οποίον μετατρέπονται. Παρατηρούμε ότι ένα δεκαεξαδικό ψηφίο μετατρέπεται σε 4 bit, δηλαδή σε μισό byte. Ένα byte επομένως αναπαρίσταται με δύο δεκαεξαδικά ψηφία.

11111111 → FF

00011010 → 1A

Ας μελετήσουμε τον δυαδικό αριθμό 00011010 και ας χωρίσουμε το byte σε δύο τμήματα των τεσσάρων bit το καθένα: 0001 και 1010. Σύμφωνα με τον παραπάνω πίνακα, το πρώτο τμήμα μετατρέπεται στο δεκαεξαδικό ψηφίο 1 και το δεύτερο στο ψηφίο A, οπότε ο δεκαεξαδικός που προκύπτει είναι ο 1A. Το αντίστροφο είναι επίσης πολύ εύκολο, για παράδειγμα ο δεκαεξαδικός B6DE μετατρέπεται άμεσα στον δυαδικό 1011011011011110! Αντίθετα η μετατροπή ενός δεκαδικού αριθμού σε δυαδικό και αντίστροφα είναι μια πιο πολύπλοκη διαδικασία.

Μπορούμε να χρησιμοποιήσουμε την συνάρτηση `printf()` με τον συνδυασμό `%X`, στο αλφαριθμητικό μορφοποίησης για να εμφανίσουμε έναν ακέραιο αριθμό σε δεκαεξαδική μορφή. Για παράδειγμα, το παρακάτω τμήμα κώδικα εμφανίζει τον αριθμό 26 τόσο σε δεκαδική όσο και σε δεκαεξαδική μορφή.

Μπορούμε να χρησιμοποιήσουμε την συνάρτηση `printf()` με τον συνδυασμό `%X`, στο αλφαριθμητικό μορφοποίησης για να εμφανίσουμε έναν ακέραιο αριθμό σε δεκαεξαδική μορφή. Για παράδειγμα, το παρακάτω τμήμα κώδικα εμφανίζει τον αριθμό 26 τόσο σε δεκαδική όσο και σε δεκαεξαδική μορφή.

```
int a=26;
printf("a=%d a=%X\n",a,a);
```

a=26 a=1A

Ο τύπος δεδομένων bool

Το πρότυπο C99 εισήγαγε ακόμα έναν τύπο, τον λογικό τύπο `bool`, ο οποίος απουσιάζει από το αρχικό πρότυπο της γλώσσας. Η προσθήκη αυτή έγινε ώστε να υπάρχει συμβατότητα με τη C++, η οποία περιλαμβάνει αυτόν τον τύπο δεδομένων στους βασικούς τύπους της.

Ο τύπος `bool` είναι ένας λογικός τύπος δεδομένων με επιτρεπτές τιμές `true` ή `false`.

Δεκαεξαδικό ψηφίο	Δεκαδικό ψηφίο/αριθμός	Δυαδικός αριθμός
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Κεφάλαιο

Εντολές συνθήκης

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Στο κεφάλαιο αυτό παρουσιάζονται οι εντολές με τις οποίες μπορούμε να εκτελούμε λειτουργίες του προγράμματος «υπό συνθήκη».

Περιγράφεται η λογική με την οποία η C θεωρεί μια συνθήκη ως αληθή ή ψευδή, καθώς και ένας ιδιαίτερος τελεστής ο οποίος μπορεί να χρησιμοποιηθεί για τον έλεγχο λογικών παραστάσεων.

Περιέχει επίσης παραδείγματα τα οποία υλοποιούν προγράμματα με μια πιο σύνθετη λογική.

Εισαγωγή

Το επόμενο βήμα που θα εκτελεστεί σε ένα πρόγραμμα πιθανόν να εξαρτάται από μια συνθήκη. Για παράδειγμα, ένα πρόγραμμα θα μπορούσε να ζητάει τον βαθμό ενός φοιτητή σε κάποιο μάθημα, και αν ο βαθμός είναι μεγαλύτερος ή ίσος με το 5 να εμφανίζει τη λέξη «πέρασες», διαφορετικά να εμφανίζει «κόπηκες».

Η C διαθέτει δύο εντολές με τις οποίες έχει τη δυνατότητα εκτέλεσης προτάσεων υπό συνθήκη («υπό όρους»). Αυτές είναι η εντολή **if** και η εντολή **switch-case**.

Η εντολή **if** σε μια δεύτερη ματιά

Έχουμε ήδη συναντήσει την εντολή **if**, η οποία ελέγχει μια συνθήκη και ανάλογα με το αποτέλεσμα του ελέγχου (αληθές ή ψευδές) εκτελεί (ή δεν εκτελεί) κάποιες προτάσεις. Η εντολή **if** συντάσσεται με τρεις τρόπους:

- ως απλή πρόταση **if**
- ως συνδυασμός **if-else**
- ως συνδυασμός **if-else if**

Η απλή πρόταση **if**

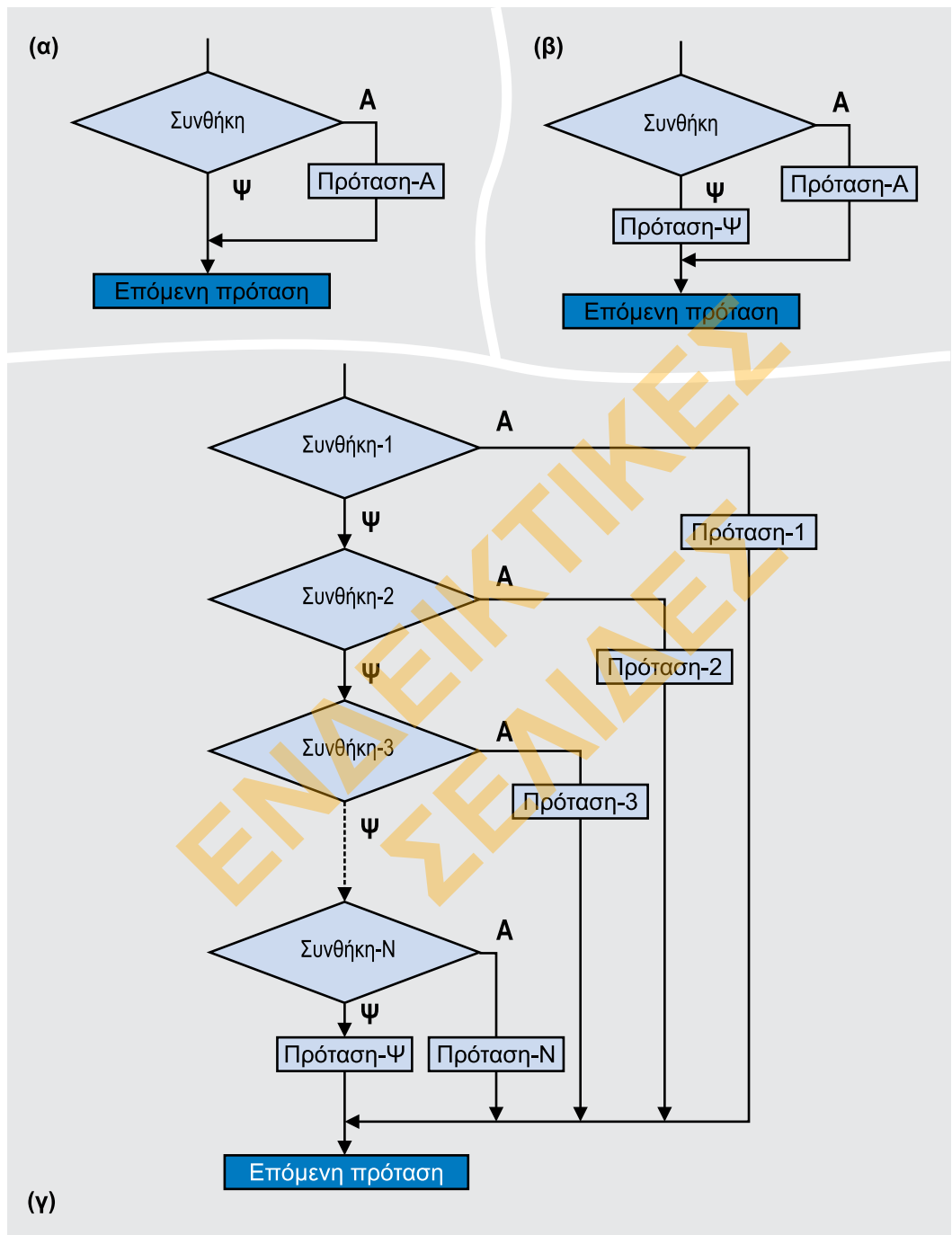
Η απλούστερη σύνταξη της **if** είναι η εξής:

if (συνθήκη) πρόταση-A;

Σε αυτή τη μορφή, η **if** ελέγχει τη συνθήκη και, αν είναι αληθής (**A**), εκτελεί την **πρόταση-A** που ακολουθεί. Αν η συνθήκη είναι ψευδής (**Ψ**), εκτελείται η πρόταση αμέσως μετά από την εντολή **if**.

Στη C, η **συνθήκη** μπορεί να είναι μια οποιαδήποτε παράσταση και όχι απαραίτητα μια λογική παράσταση. Αν η παράσταση επιστρέφει τιμή διάφορη του μηδενός θεωρείται αληθής, διαφορετικά (αν επιστρέφει μηδέν) θεωρείται ψευδής. Στο λογικό διάγραμμα του Σχήματος 7.1α φαίνεται η λογική της απλής πρότασης **if**. Η **πρόταση-A** μπορεί να είναι μια απλή ή σύνθετη πρόταση (compound statement).

Η C θεωρεί αληθή οποιαδήποτε παράσταση που επιστρέφει τιμή διάφορη του μηδενός. Έτσι, στη θέση της **συνθήκης** μπορεί να τοποθετηθεί **οποιαδήποτε παράσταση** που αποδίδει αριθμητική τιμή (δείτε το Παράδειγμα Π7.3). Συνήθως, βέβαια, η συνθήκη αποτελείται από μια λογική παράσταση με αποτέλεσμα 1 (αληθές) ή 0 (ψευδές).



Σχήμα 7.1 Η εντολή *if*

Παράδειγμα χρήσης της απλής πρότασης **if**:

if (a==5) printf("a=5\n"); η οποία μπορεί επίσης να γραφεί

```
if (a==5)
    printf("a=5\n");           ή
```

```
if (a==5)
{
    printf("a=5\n");
}
```

Όπως αναφέραμε, η **πρόταση-Α** μπορεί να είναι και μια σύνθετη πρόταση:

```
if (a==5)
{
    printf("a=5\n");
    printf("μια σύνθετη πρόταση\n");
}
```

Η πρόταση **if-else**

if (συνθήκη) Πρόταση-Α; else Πρόταση-Ψ;

Σε αυτή τη μορφή, η **if** ελέγχει τη συνθήκη και, αν είναι αληθής (**Α**), εκτελεί την **Πρόταση-Α** που ακολουθεί. Αν η συνθήκη είναι ψευδής (**Ψ**), εκτελείται η **Πρόταση-Ψ**. Μετά εκτελείται κανονικά η επόμενη από την **if** πρόταση του προγράμματος. Η λογική της πρότασης **if-else** φαίνεται στο λογικό διάγραμμα του Σχήματος 7.1β.

Οι προτάσεις Α και Ψ μπορεί να είναι απλές ή σύνθετες προτάσεις.

Παράδειγμα χρήσης της **if-else**:

```
if (a==5) printf("Ναι"); else printf("Όχι");
```

ή και

```
if (a==5)
    printf("Ναι");
else
    printf("Όχι");
if (a==5)
{
    printf("Ναι\n");
    printf("Σύνθετη πρόταση - Α\n");
}
else
{
    printf("Όχι\n");
    printf("Σύνθετη πρόταση - Β\n");
}
```

← Απλές προτάσεις

← Σύνθετες προτάσεις

Η πρόταση if-else if

Αφού μετά από το **else** μπορεί να υπάρχει οποιαδήποτε πρόταση, αυτή μπορεί να είναι και άλλη μια πρόταση **if**. Η λογική της πρότασης **if-else if** φαίνεται στο λογικό διάγραμμα του Σχήματος 7.1γ. Η τελευταία **Πρόταση-Ψ** εκτελείται μόνο όταν όλες οι προηγούμενες συνθήκες είναι ψευδείς. Π.χ.:



Οι **προτάσεις** μπορεί να είναι απλές ή σύνθετες.

Το επόμενο πρόγραμμα περιμένει από τον χρήστη να πληκτρολογήσει έναν ακέραιο αριθμό, τον οποίο καταχωρίζει στη μεταβλητή **a**. Αν ο αριθμός που δώσαμε είναι 5 θα εμφανιστεί **a=5**, αν είναι 8 θα εμφανιστεί **a=8**, και αν είναι μεγαλύτερος από το 20 θα εμφανιστεί η λέξη «**μεγάλο**».

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

7_lf.c

```
int main(void)
{
    int a;
    scanf("%d",&a);
    if (a==5)
        printf("a=5\n");
    else if (a==8)
        printf("a=8\n");
    else if (a>20)
        printf("μεγάλο\n");
    else
        printf("ΤΕΛΟΣ\n");
    return 0;
}
```

Η λέξη «**ΤΕΛΟΣ**» θα εμφανιστεί μόνο αν ο αριθμός που δώσαμε δεν είναι ούτε 5, ούτε 8, ούτε μεγαλύτερος του 20 (π.χ. αν είναι 15).

Η πρόταση **if-else if** χρησιμοποιείται σε περιπτώσεις διαδοχικών ελέγχων. Στο Παράδειγμα Π7.4 που ακολουθεί διασαφηνίζεται η χρήση της σε ένα πρόβλημα κλιμακωτής χρέωσης.

Το πρόβλημα του κλιμακωτού υπολογισμού

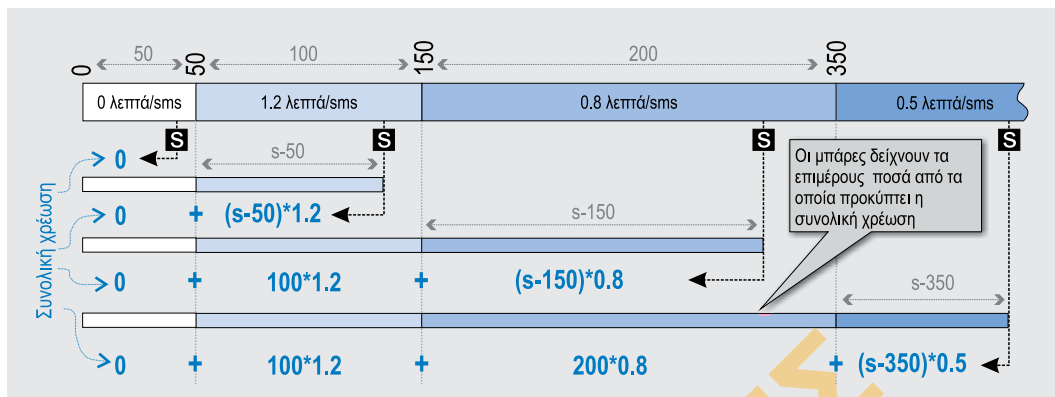
Αρκετές φορές πρέπει να υπολογίζουμε τιμές βάσει κάποιας λογικής «κλιμάκωσης». Για παράδειγμα, ας υποθέσουμε ότι μια εταιρεία κινητής τηλεφωνίας έχει την ακόλουθη πολιτική για τη χρέωση των SMS που στέλνονται: για τα πρώτα 50 SMS δεν χρεώνει τίποτα, για τα επόμενα 100 χρεώνει 1.2 λεπτά/SMS, για τα επόμενα 200 0.8 λεπτά/sms και για όλα τα υπόλοιπα 0.5 λεπτά/SMS. Υπάρχει επομένως κλιμακωτή και όχι ενιαία χρέωση. Το παρακάτω πρόγραμμα χρησιμοποιεί τη παραπάνω σύνταξη της **if**, με διαδοχικούς ελέγχους του πλήθους των SMS, για να υπολογίσει το ποσό της χρέωσης στην κάθε περίπτωση.

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

7_sms.c

```
int main(void)
{
    int s;
    float poso;
    scanf("%d",&s);
    if (s<=50)
        poso=0;
    else if (s<150)
        poso=(s-50)*1.2;
    else if (s<350)
        poso=100*1.2+(s-150)*0.8;
    else
        poso=100*1.2+200*0.8+(s-350)*0.5;
    printf("Η χρέωση των %d SMS είναι %f ευρώ\n",s,poso/100);
    return 0;
}
```

Στην περίπτωση που το πλήθος των SMS (s) είναι μικρότερο ή ίσο του 50, το ποσό της χρέωσης είναι 0. Στην περίπτωση που το πλήθος είναι από 50 μέχρι 150, τότε πληρώνουμε 1.2 λεπτά ανά SMS αλλά όχι στα 50 πρώτα που είναι δωρεάν. Οπότε η χρέωση θα είναι $(s-50)*1.2$. Τώρα αν στείλουμε περισσότερα από 150 αλλά λιγότερα από 350, για τα πρώτα 50 έχουμε χρέωση 0, για τα επόμενα 100 (τα οποία σίγουρα έχουμε στείλει) $100*1.2$ και για τα υπόλοιπα (πάνω από τα 150) $0.8/\text{SMS}$, δηλαδή $(s-150)*0.8$! Με την ίδια λογική, αν έχουμε στείλει περισσότερα από 350 η χρέωση είναι: $0 + 100*1.2 + 200*0.8 + (s-350)*0.5$. Στο Σχήμα 7.2 φαίνεται με εποπτικό τρόπο ο τρόπος υπολογισμού της χρέωσης σε κάθε περίπτωση.



Σχήμα 7.2 Παράδειγμα κλιμακωτής χρέωσης

Ένθετες εντολές if

Μια εντολή **if** μπορεί να είναι ένθετη (ή εμφωλευμένη) μέσα σε μια άλλη εντολή **if**. Για παράδειγμα, στο πρόγραμμα που ακολουθεί η δεύτερη εντολή **if** ($a > 10$) βρίσκεται μέσα στο τμήμα της πρώτης **if** το οποίο θα εκτελεστεί μόνο στην περίπτωση που η λογική παράσταση $a > b$ είναι αληθής. Η τρίτη εντολή **if** ($b == 10$) βρίσκεται μέσα στο τμήμα **else** της πρώτης **if** και θα εκτελεστεί μόνο στην περίπτωση που η λογική παράσταση $a > b$ είναι ψευδής.

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

7_nested_if.c

```
int main(void)
```

```
{
```

```
    int a,b;
```

```
    scanf("%d %d", &a, &b);
```

```
    if (a>b)
```

```
        if (a>10)
```

```
            printf("To a είναι μεγαλύτερο του b και μεγαλύτερο του 10");
```

```
        else
```

```
            printf("To a είναι μεγαλύτερο του b και μικρότερο ή ίσο του 10");
```

```
    else
```

```
        if (b==10)
```

```
            printf("To a δεν είναι μεγαλύτερο του b και το b είναι 10");
```

```
        else
```

```
            printf("To a δεν είναι μεγαλύτερο του b και το b δεν είναι 10");
```

```
    return 0;
```

```
}
```

Ένθετη εντολή if

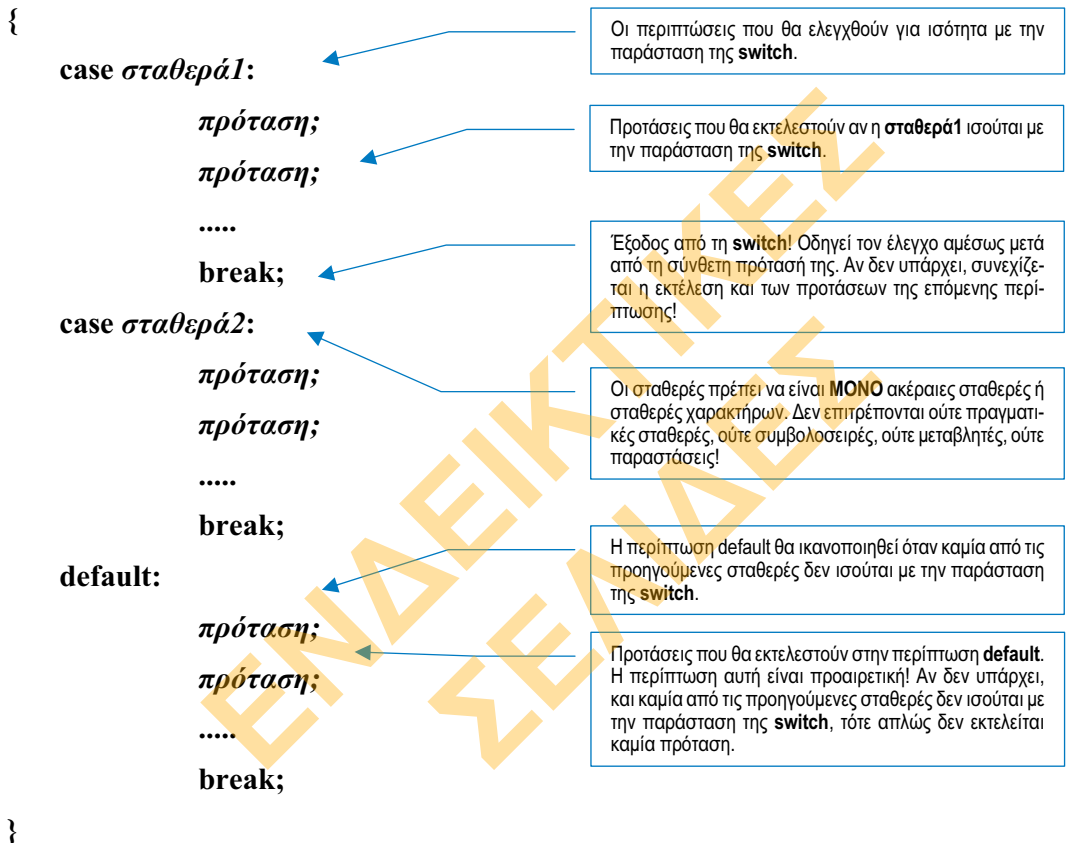
Ένθετη εντολή if

Η εντολή switch-case

Η πρόταση **switch-case** ελέγχει αν η τιμή μιας **ακέραιας** παράστασης ισούται με συγκεκριμένες σταθερές (και μόνο σταθερές).

Το συντακτικό της είναι το εξής:

switch (παράσταση)



Ανάλογα με τη σταθερά με την οποία ισούται η **παράσταση**, εκτελούνται οι προτάσεις από την **case** της συγκεκριμένης σταθεράς μέχρι την επόμενη **break**.

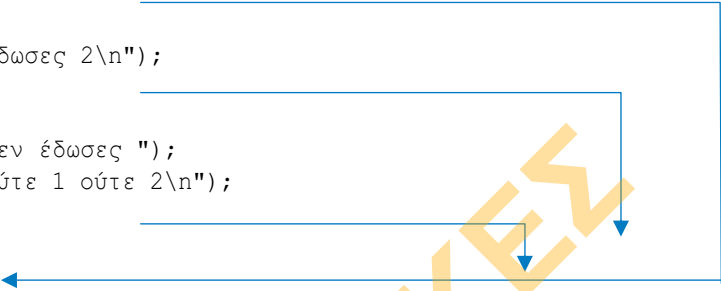
Οι προτάσεις κάτω από την **default** εκτελούνται μόνο όταν η παράσταση δεν ισούται με καμία από τις σταθερές των προτάσεων **case**.

🔗 Η κάθε **case** ακολουθείται από μια **σταθερά** και μια άνω και κάτω τελεία (:), και όχι από ερωτηματικό (;)

🔗 Οι σταθερές είναι είτε **ακέραιες σταθερές** είτε **σταθερές χαρακτήρα**. Δεν επιτρέπεται να είναι συμβολοσειρές ή αριθμοί κινητής υποδιαστολής.

Τα επόμενα τμήματα κώδικα δείχνουν τη λειτουργία της **switch**.

```
switch (a)
{
    case 1:
        printf("έδωσες 1\n");
        break;
    case 2:
        printf("έδωσες 2\n");
        break;
    default:
        printf("Δεν έδωσες ");
        printf("ούτε 1 ούτε 2\n");
        break;
}
.....
.....
```



Η εντολή **break** μεταφέρει τον έλεγχο του προγράμματος στην πρόταση αμέσως μετά από τη δομή της **switch**.

Ανάλογα με την τιμή της μεταβλητής **a**, στην οθόνη θα εμφανιστούν τα εξής:

- *έδωσες 1* αν η **a** ισούται με 1
- *έδωσες 2* αν η **a** ισούται με 2
- *Δεν έδωσες ούτε 1 ούτε 2* αν η **a** δεν είναι ίση ούτε με το 1 ούτε με το 2.

Στο παράδειγμα που ακολουθεί, αν η μεταβλητή **a** έχει τιμή 1 θα εκτελεστούν οι δύο προτάσεις **printf()**, τόσο της **case 1** όσο και της **case 2**, μέχρι την πρώτη εντολή **break**.

```
switch (a)
{
    case 1:
        printf("έδωσες 1\n");
    case 2:
        printf("έδωσες 2\n");
        break;
    default:
        printf("Δεν έδωσες ");
        printf("ούτε 1 ούτε 2\n");
        break;
}
```

Την εντολή **break** θα τη συναντήσουμε ξανά στις εντολές επανάληψης **while**, **do-while**, και **for**.

Η λειτουργικότητα της εντολής **switch** μπορεί να υλοποιηθεί και από την εντολή **if**. Μελετήστε τα παραδείγματα Π7.2 και Π7.5.

Ο τελεστής συνθήκης ?:

Ο τελεστής συνθήκης **?:** που παρέχει η C είναι ένας εύκολος τρόπος για τον έλεγχο μιας συνθήκης. Σε μερικές περιπτώσεις, μπορεί να αντικαταστήσει την πρόταση **if-else**, όταν η **if** συντάσσεται με απλές, και όχι με σύνθετες, προτάσεις.

Ο τρόπος χρήσης του τελεστή **?:** σε μια παράσταση είναι ο εξής:

Συνθήκη ? παράσταση1 : παράσταση2;

Πρώτα ελέγχεται η **Συνθήκη** και, αν είναι αληθής (διάφορη του 0), τότε υπολογίζεται η **παράσταση1** και το αποτέλεσμά της είναι το αποτέλεσμα όλης της παράστασης. Αν είναι ψευδής (0), τότε υπολογίζεται η **παράσταση2** και το αποτέλεσμά της είναι το αποτέλεσμα όλης της παράστασης. Η **Συνθήκη** είναι συνήθως μια λογική παράσταση, αλλά επιτρέπεται να είναι οποιαδήποτε παράσταση. Θεωρείται αληθής αν έχει τιμή διάφορη του μηδενός.

Για παράδειγμα, στον παρακάτω κώδικα

```
x=10;
y=x>9?100:200;
```

το **y** θα πάρει τιμή 100. Η παράσταση **y=x>9?100:200;** είναι ισοδύναμη με την παρακάτω πρόταση **if**:

```
if (x>9)
    y=100;
else
    y=200;
```

Παρόμοια, αν υποθέσουμε ότι **a=5** και **b=8**, τότε οι επομένως παραστάσεις και προτάσεις θα έχουν τα αποτελέσματα που αναφέρονται δίπλα τους:

<code>3>4?56:34</code>	→	34
<code>a==b?a:b</code>	→	8
<code>a>b?a:b</code>	→	8
<code>a+5>b?a+2:b-3</code>	→	7
<code>c=a>b?a:b;</code>	→	η μεταβλητή <i>c</i> θα πάρει τη μεγαλύτερη τιμή από τις <i>a</i> και <i>b</i>
<code>a>b?printf("%d",a):printf("%d",b);</code>	→	Θα εμφανίσει το 8
<code>c=8?a:b;</code>	→	η μεταβλητή <i>c</i> θα πάρει την τιμή της <i>a</i> , δηλαδή το 5. Η συνθήκη (το 8) θεωρείται αληθής ως διάφορη του μηδενός.
<code>c=a-b+3?10:20;</code>	→	η μεταβλητή <i>c</i> θα πάρει την τιμή 20 διότι η συνθήκη <i>a-b+3</i> έχει τιμή 0 και θεωρείται ψευδής.

Παραδείγματα

Π7.1 Το επόμενο πρόγραμμα βρίσκει τις πραγματικές λύσεις μιας εξίσωσης δευτέρου βαθμού $ax^2+bx+c=0$. Στην περίπτωση που η εξίσωση δεν έχει πραγματικές ρίζες, εμφανίζεται ανάλογο μήνυμα.

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

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

```
int main(void)
```

```
{
```

```
    float a,b,c,r1,r2,d;
```

```
    printf("Δωσε τους συντελεστές a, b και c:");
```

```
    scanf("%f %f %f",&a,&b,&c);
```

```
    d=pow(b,2)-4*a*c; // Υπολογισμός της διακρίνουσας
```

```
    if (d>0)
```

```
    {
```

```
        r1=(-b + sqrt(d))/(2*a);
```

```
        r2=(-b - sqrt(d))/(2*a);
```

```
        printf("Οι ρίζες είναι %f και %f\n",r1,r2);
```

```
    }
```

```
    else if (d==0)
```

```
    {
```

```
        r1=-b/(2*a);
```

```
        printf("Διπλή ρίζα %f\n",r1);
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Δεν υπάρχουν πραγματικές ρίζες");
```

```
    }
```

```
    return 0;
```

```
}
```

7_p7_1.c

Διακρίνουσα θετική. Δύο πραγματικές ρίζες.

Διακρίνουσα 0. Μια διπλή ρίζα.

Διακρίνουσα αρνητική. Δεν υπάρχουν πραγματικές ρίζες.

🔗 Η διακρίνουσα υπολογίζεται βάση του μαθηματικού τύπου $\Delta = b^2 - 4 \cdot a \cdot \gamma$ όπου a, b και γ είναι οι συντελεστές της εξίσωσης $a \cdot x^2 + b \cdot x + \gamma = 0$.

🔗 Οι πραγματικές ρίζες της εξίσωσης υπολογίζονται βάσει του διπλανού μαθηματικού τύπου.

$$\rho_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2 \cdot a}$$

🔗 Για να μπορέσουμε να χρησιμοποιήσουμε τις συναρτήσεις `pow()` και `sqrt()`, θα πρέπει να συμπεριλάβουμε στο πρόγραμμά μας το αρχείο κεφαλίδας `<math.h>` μέσα στο οποίο δηλώνονται όλες οι μαθηματικές συναρτήσεις.

Κεφάλαιο

Εντολές επανάληψης



Στο κεφάλαιο αυτό παρουσιάζονται οι εντολές μέσω των οποίων μπορούμε να επιτύχουμε την επαναληπτική εκτέλεση λειτουργιών, αλλά και εντολές με τις οποίες μπορούμε να τη διακόψουμε.

Περιγράφεται αναλυτικά το συντακτικό των εντολών, καθώς και οι ιδιαιτερότητές τους στις οποίες οφείλεται η ευελιξία τους αλλά και πολλά προβλήματα.

Για ιστορικούς μόνο λόγους παρουσιάζεται και η «απαγορευμένη» εντολή **goto**, η οποία μας οδηγεί σε χαοτικά προγράμματα και είναι ο απόλυτος εχθρός του δομημένου προγραμματισμού. Εξακολουθεί όμως να υποστηρίζεται από τη γλώσσα, κυρίως για λόγους συμβατότητας.

Οι επαναληπτικές δομές επιτρέπουν την κατασκευή πιο πολύπλοκων προγραμμάτων. Υπάρχουν παραδείγματα επεξεργασίας μεγάλου πλήθους δεδομένων και παρουσιάζονται τεχνικές που θα σας φανούν πολύ χρήσιμες στη συνέχεια.

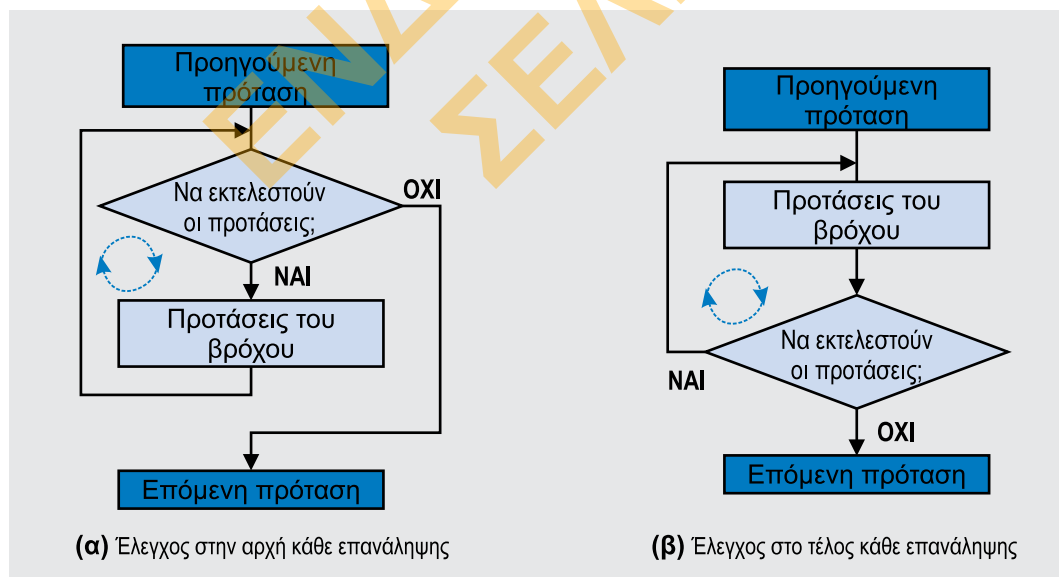
Εισαγωγή

Ένα από τα βασικά πλεονεκτήματα της επεξεργασίας δεδομένων μέσω υπολογιστικών συστημάτων είναι η δυνατότητα της επανάληψης των διαδικασιών. Δηλαδή, η δυνατότητα των γλωσσών προγραμματισμού να επιβάλλουν την επαναλαμβανόμενη εκτέλεση ενός τμήματος κώδικα μέσα σε ένα πρόγραμμα. Κάθε γλώσσα προγραμματισμού διαθέτει δομές για την υλοποίηση αυτής της διαδικασίας. Μια δομή επανάληψης καλείται επίσης *βρόχος* (loop) και οι προτάσεις που εκτελούνται αποτελούν το *σώμα* του βρόχου.

Σε κάθε περίπτωση, η επαναληπτική εκτέλεση των προτάσεων εξαρτάται από τον έλεγχο κάποιας συνθήκης (π.χ. «όσο το a είναι μεγαλύτερο από το 5 επανέλαβε τις παρακάτω προτάσεις»). Ο έλεγχος αυτός σε κάποιες δομές γίνεται πριν από την εκτέλεση της κάθε επανάληψης των προτάσεων του βρόχου (Σχήμα 8.1α) και σε κάποιες δομές μετά από την εκτέλεσή τους (Σχήμα 8.1β).

Κάποιες δομές επανάληψης χρησιμοποιούνται όταν δεν γνωρίζουμε εκ των προτέρων το πλήθος των επαναλήψεων (π.χ. «μέτρα τα αυτοκίνητα που περνάνε μέχρι να περάσει κάποιο με κόκκινο χρώμα») και κάποιες άλλες χρησιμοποιούνται όταν το γνωρίζουμε (π.χ. «ζήτη 100 αριθμούς και υπολόγισε το άθροισμά τους»).

Η C υποστηρίζει τρεις δομές επανάληψης που υλοποιούνται με τις αντίστοιχες εντολές **while**, **do-while** και **for**. Οι δομές επανάληψης είναι παρόμοιες μεταξύ τους και στην πραγματικότητα, με ελάχιστες τροποποιήσεις, η λειτουργικότητα μιας δομής μπορεί να υλοποιηθεί από μια άλλη.



Σχήμα 8.1 Δομές επανάληψης

Η εντολή while

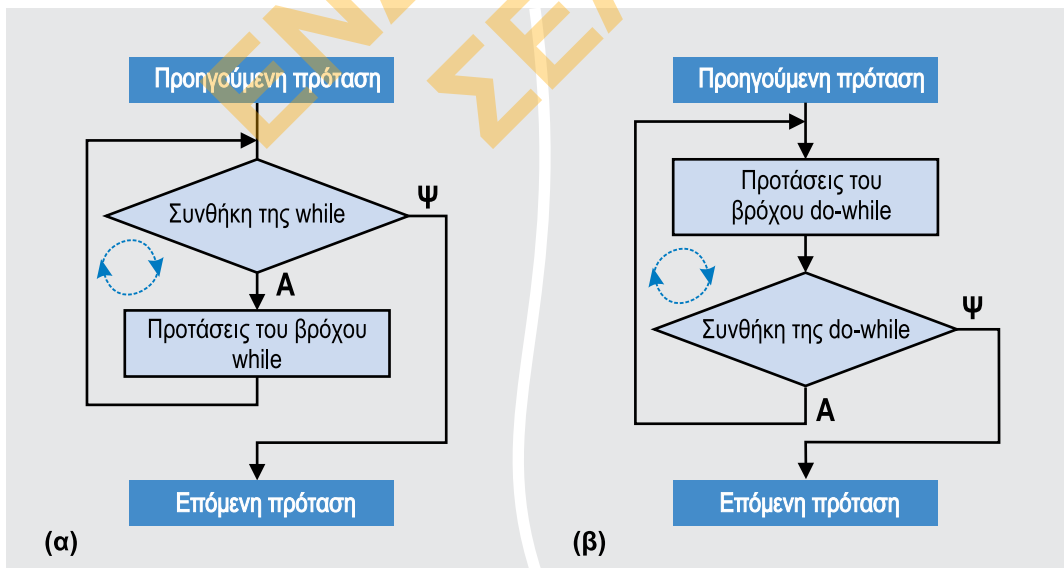
Η εντολή (ή βρόχος – loop) **while** υλοποιεί τη μία από τις τρεις δομές επανάληψης της C. Με τη χρήση της εντολής **while** είναι δυνατή η επαναληπτική εκτέλεση μίας ή περισσότερων προτάσεων. Η επανάληψη ή μη επανάληψη των προτάσεων εξαρτάται από την τιμή μιας παράστασης συνθήκης (συνήθως λογικής). Για παράδειγμα, ο παρακάτω βρόχος **while** εκτελείται όσο η τιμή της μεταβλητής **a** είναι μεγαλύτερη από το 0.

```
a=5;
while (a>0)
{
    printf("a=%d\n",a);
    --a;
}
```

Ελέγχεται αν η τιμή της μεταβλητής **a** είναι μεγαλύτερη από 0. Αν είναι, συνεχίζεται η εκτέλεση των προτάσεων του βρόχου. Η επαναληπτική εκτέλεση των προτάσεων θα σταματήσει όταν η τιμή της **a** γίνει 0.

Η εντολή **while** εκτελεί επαναληπτικά μια πρόταση (απλή ή σύνθετη) **ενόσω** μια συνθήκη (λογική ή μη) είναι αληθής. Το λογικό διάγραμμα του Σχήματος 8.2α δείχνει τη φιλοσοφία της εντολής **while**, ενώ το συντακτικό της είναι το ακόλουθο:

```
while (συνθήκη) πρόταση;   ή
while (συνθήκη)
    πρόταση;               ή
while (συνθήκη)
{
    Προτάσεις του βρόχου;
}
```



Σχήμα 8.2 Οι εντολές **while** (α) και **do-while** (β)

Η συνθήκη μπορεί να είναι **οποιαδήποτε παράσταση** που επιστρέφει μια αριθμητική τιμή. Αν η τιμή της είναι 0 θεωρείται ψευδής, διαφορετικά θεωρείται αληθής.

Με την παρακάτω εντολή **while** εμφανίζεται συνεχώς η φράση «Η γλώσσα C σε βάθος», αφού η παράσταση έχει πάντα τιμή 1, δηλαδή είναι αληθής. Σε αυτό το σημείο η εκτέλεση του προγράμματος θα «κολλήσει». Ένας τέτοιος βρόχος καλείται **ατέρμων βρόχος** (infinite loop).

```
while (1) printf("Η γλώσσα C σε βάθος\n");
```

Το πρόγραμμα που ακολουθεί εμφανίζει την τιμή της μεταβλητής **a** ενόσω αυτή είναι μεγαλύτερη του μηδενός. Μετά το τέλος του βρόχου, εμφανίζει τη λέξη «ΤΕΛΟΣ».

```
#include <stdio.h>
int main(void)
```

8_while1.c

```
{
    int a=5;
    while (a>0)
    {
        printf("a=%d\n",a);
        --a;
    }
    printf("ΤΕΛΟΣ\n");
    return 0;
}
```

a=5
a=4
a=3
a=2
a=1
ΤΕΛΟΣ

Το επόμενο πρόγραμμα περιμένει να πατηθεί το πλήκτρο του διαστήματος (κωδικός ASCII 32). Ο βρόχος **while** εκτελείται συνέχεια, ενόσω το πλήκτρο που πατιέται δεν είναι το πλήκτρο του διαστήματος.

```
#include <stdio.h>
int main(void)
```

8_while2.c

```
{
    char ch='\0';
    while (ch!=32) ch=getch();
    printf("επιτέλους το πάτησες\n");
    return 0;
}
```

Ο βρόχος **while** περιέχει μία πρόταση.

Εκμεταλλευόμενοι την ευελιξία της C, θα μπορούσαμε να γράψουμε το προηγούμενο πρόγραμμα με τον ακόλουθο τρόπο.

```
#include <stdio.h>
int main(void)
```

8_while3.c

```
{
    char ch='\0';
    while ((ch=getch())!=32);
    printf("επιτέλους το πάτησες\n");
    return 0;
}
```

Προσοχή: ο βρόχος **while** δεν περιέχει καμία πρόταση. Ό,τι υπάρχει μέσα στις παρενθέσεις είναι η **συνθήκη** της **while**!

Η παράσταση `ch=getch()` περιμένει να πατηθεί ένα πλήκτρο χαρακτήρα, καταχωρίζει τον χαρακτήρα στη μεταβλητή `ch`, και επιστρέφει ως τιμή την τιμή του `ch`. Η τιμή αυτή, με χρήση του τελεστή `!=`, ελέγχεται αν είναι διάφορη του 32 (κωδικός του διαστήματος).

☞ Η συγκεκριμένη `while` δεν έχει **καμία** πρόταση για εκτέλεση. Απλώς ελέγχει συνεχώς τη τιμή της συνθήκης. Για να τερματιστεί ο βρόχος, θα πρέπει ο κωδικός του χαρακτήρα να γίνει 32 (κωδικός ASCII του διαστήματος).

Η εντολή `do-while`

Ο εντολή **`do-while`** (ή βρόχος `do-while`) κάνει παρόμοια δουλειά με τη `while`, με τη διαφορά ότι ελέγχει τη συνθήκη στο τέλος, και όχι στην αρχή, του βρόχου.

Η σύνταξη της **`do-while`** για μια απλή πρόταση είναι η εξής:

`do`

πρόταση;

`while (συνθήκη);`

ενώ για μια σύνθετη πρόταση είναι η παρακάτω:

`do`

`{`

Προτάσεις του βρόχου;

`} while (συνθήκη);`

Το λογικό διάγραμμα του Σχήματος 8.2β δείχνει τη φιλοσοφία της `do-while`.

☞ Αντίθετα με τη `while`, η `do-while` πρέπει να τερματιστεί με ερωτηματικό (;) μετά από τη συνθήκη.

☞ Παρατηρούμε επίσης ότι οι προτάσεις μέσα στον βρόχο `do-while` θα εκτελεστούν **τουλάχιστον μία φορά**, ανεξάρτητα από την τιμή της συνθήκης.

☞ Όπως και στη `while`, η συνθήκη μπορεί να είναι οποιαδήποτε αριθμητική παράσταση.

Στο τμήμα κώδικα που ακολουθεί, ο χρήστης πληκτρολογεί έναν αριθμό ο οποίος καταχωρίζεται στη μεταβλητή `a`. Οι προτάσεις του βρόχου `do-while` εμφανίζουν τη τιμή της μεταβλητής `a` και ακολούθως τη μειώνουν κατά 1. Η επαναληπτική λειτουργία εκτελείται όσο η τιμή της μεταβλητής `a` είναι μεγαλύτερη από το 0. Για παράδειγμα, αν ο χρήστης πληκτρολογήσει τον αριθμό 4, ο βρόχος θα εκτελεστεί 4 φορές. Ακόμα και αν δώσει αρνητική τιμή ο χρήστης, ο βρόχος θα εκτελεστεί μία φορά!


```
int a;  
scanf("%d",&a);  
do  
{  
    printf("%d\n",a);  
    --a;  
} while (a>0);
```

Οι προτάσεις του βρόχου θα εκτελεστούν τουλάχιστον μία φορά, ακόμα και αν ο χρήστης πληκτρολογήσει αρνητική τιμή!

Ας υποθέσουμε ότι γίνεται ένας έρανος στον οποίο προσπαθούμε να μαζέψουμε το ποσό των 1000€. Ζητάμε από διάφορους πολίτες να συνεισφέρουν με κάποιο ποσό και σταματάμε όταν το συνολικό ποσό γίνει ίσο ή υπερβεί τα 1000€.

Το πρόγραμμα που ακολουθεί υλοποιεί αυτή την απαίτηση.

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

8_do_while.c

```
int main(void)  
{  
    float poso,sum=0;  
    int plithos=0,polla=0,liga=0;  
    do  
    {  
        printf("Δώσε ποσό: ");  
        scanf("%f",&poso);  
        sum=sum+poso;  
        plithos++;  
        if (poso>=50)  
            polla++;  
        if (poso<=2)  
            liga++;  
    } while (sum<1000);  
    printf("Συνολικό ποσό: %f\n",sum);  
    printf("Ατομα που συμμετείχαν στον έρανο: %d\n",plithos);  
    printf("Ατομα που έδωσαν από 50 ευρώ και πάνω %d\n",polla);  
    printf("Ατομα που έδωσαν μέχρι 2 ευρώ %d\n",liga);  
    return 0;  
}
```

Το ποσό που δίνει ο χρήστης προστίθεται κάθε φορά στη μεταβλητή **sum**.

Η μεταβλητή **plithos** μετράει το σύνολο των ατόμων που συμμετείχαν στον έρανο, η **polla** μετρά το σύνολο των ατόμων που έδωσαν από 50€ και πάνω και η **liga** μετρά τα άτομα που έδωσαν μέχρι 2€.

Η επαναληπτική διαδικασία συνεχίζεται όσο η μεταβλητή **sum** έχει τιμή μικρότερη του 1000.

Στο τέλος, το πρόγραμμα εμφανίζει το συνολικό ποσό που συγκεντρώθηκε, το πλήθος των πολιτών που συμμετείχαν στον έρανο, το πλήθος των ατόμων που έδωσαν από 50€ και πάνω, καθώς και το πλήθος των ατόμων που έδωσαν μέχρι 2€.

Συνήθως ο βρόχος **do-while** χρησιμοποιείται όταν η επανάληψη του βρόχου εξαρτάται από τιμές που εισάγονται από τον χρήστη σε προτάσεις οι οποίες βρίσκονται μέσα στον βρόχο. Στις περιπτώσεις αυτές θέλουμε οι προτάσεις του βρόχου να εκτελεστούν τουλάχιστον μία φορά ώστε ο χρήστης να έχει την ευκαιρία να εισαγάγει τις πρώτες τιμές.

Η εντολή for

Στους βρόχους **while** και **do-while** δεν είναι καθορισμένο το πλήθος των επαναλήψεων. Αν θέλουμε να μετράμε το πλήθος των επαναλήψεων, πρέπει να χρησιμοποιήσουμε κάποια μεταβλητή-μετρητή. Για παράδειγμα, το τμήμα κώδικα που ακολουθεί εμφανίζει τη φράση «Η γλώσσα C σε βάθος» 8 φορές. Η μεταβλητή **i** χρησιμοποιείται για την καταμέτρηση των επαναλήψεων.

```
int i=1;
while (i<=8)
{
    printf("Η γλώσσα C σε βάθος\n");
    i++;
}
```

Για να υλοποιηθεί η εκτέλεση του βρόχου συγκεκριμένες φορές, χρησιμοποιήθηκαν τρεις παραστάσεις: Η παράσταση εκχώρησης αρχικής τιμής, η παράσταση συνθήκης και η παράσταση αύξησης τιμής.

Η εντολή (ή βρόχος) **for** ενσωματώνει όλα τα στοιχεία του προηγούμενου παραδείγματος, και είναι ο καταλληλότερος τρόπος για την επαναληπτική εκτέλεση προτάσεων για συγκεκριμένο αριθμό φορών. Η κλασική χρήση του βρόχου **for** προϋποθέτει τη χρήση μιας μεταβλητής από την τιμή της οποίας εξαρτάται η επαναληπτική εκτέλεση του βρόχου. Η μεταβλητή αυτή παίρνει μια αρχική τιμή και μετά από κάθε εκτέλεση του βρόχου αυξάνεται ή μειώνεται (συνήθως κατά 1). Η επαναληπτική διαδικασία σταματάει όταν η τιμή της μεταβλητής ξεπεράσει τη μέγιστη (ή ελάχιστη) τιμή που έχει οριστεί. Στο επόμενο παράδειγμα χρησιμοποιείται η μεταβλητή **i** ως μεταβλητή του βρόχου **for**. Η πρόταση του βρόχου θα εκτελεστεί 8 φορές, έτσι θα εμφανιστούν στην οθόνη 8 γραμμές με το κείμενο «Η γλώσσα C σε βάθος».

```
int i;
for (i=1; i<=8; i=i+1) // Η παράσταση i=i+1 μπορεί επίσης να γραφεί απλούστερα ως i++
{
    printf("Η γλώσσα C σε βάθος\n");
}
```

Αρχική τιμή της μεταβλητής i (1).

Μέγιστη τιμή της μεταβλητής (8).

Η πρόταση θα εκτελεστεί 8 φορές, μία για κάθε τιμή της μεταβλητής i.

Η εντολή **for** είναι ίσως η εντολή που χρησιμοποιείται πιο συχνά για επαναληπτικές διαδικασίες. Ενώ σε άλλες γλώσσες προγραμματισμού η **for** έχει έναν «αυστηρό» τρόπο σύνταξης, η σύνταξή της στη C, αν και σαφώς καθορισμένη, της προσφέρει μια μοναδική ευελιξία. Το συντακτικό της **for** είναι το εξής:

for (*Πρόταση_Πριν; Συνθήκη; Πρόταση_Μετά*)
 πρόταση_βρόχου;

ή, για σύνθετη πρόταση βρόχου:

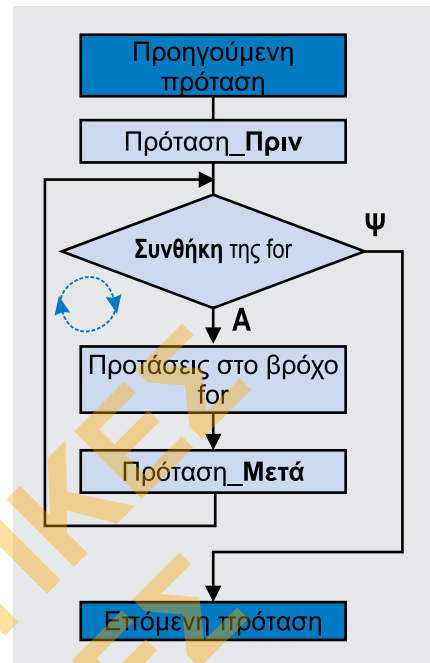
```
for (Πρόταση_Πριν; Συνθήκη; Πρόταση_Μετά)
{
    προτάσεις;
}
```

Η φιλοσοφία λειτουργίας της **for** γίνεται κατανοητή αν μελετήσουμε το διπλανό διάγραμμα ροής. Παρατηρούμε ότι η **Πρόταση_Πριν** εκτελείται μία μόνο φορά στην αρχή του βρόχου.

Μετά ελέγχεται η **Συνθήκη**. Αν είναι αληθής, εκτελούνται μία φορά οι προτάσεις του βρόχου **for** και αμέσως μετά η **Πρόταση_Μετά**. Έπειτα ελέγχεται πάλι η συνθήκη κ.ο.κ.

Η επαναληπτική εκτέλεση των προτάσεων του βρόχου θα σταματήσει όταν η συνθήκη γίνει ψευδής και ο έλεγχος του προγράμματος μεταφέρεται στην πρόταση αμέσως μετά από τη **for**.

Προσοχή, η **Πρόταση_Πριν**, η **Συνθήκη** και η **Πρόταση_Μετά** πρέπει να βρίσκονται μέσα σε παρενθέσεις και να χωρίζονται μεταξύ τους με ερωτηματικά (;)



Το επόμενο πρόγραμμα εμφανίζει τους αριθμούς από το 1 μέχρι το 8:

```
#include <stdio.h>
int main(void)
{
    int i;
    for (i=1; i<=8; i++)
        printf("i=%d\n", i);
    return 0;
}
```

8_for.c

i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8

- Στην αρχή, η μεταβλητή **i** παίρνει τιμή 1 (**i=1**).
- Το πρόγραμμα ελέγχει αν η παράσταση **i<=8** είναι αληθής.
- **Αν ναι** (είναι **i<=8**), εκτελεί τις εντολές του βρόχου μία φορά και μετά εκτελεί την πρόταση **i++** αυξάνοντας το **i** κατά 1.
- Μετά η διαδικασία συνεχίζεται και πάλι με τον έλεγχο της λογικής παράστασης, κ.ο.κ.
- **Αν όχι** (δεν είναι **i<=8**), ο βρόχος τερματίζεται.

Για να μπορέσουμε να εκμεταλλευτούμε την ευελιξία που προσφέρει η εντολή **for** στη C, θα πρέπει να γίνει πλήρως κατανοητή η φιλοσοφία της. Ας συνοψίσουμε με απλά λόγια το συντακτικό της πριν προχωρήσουμε στα επόμενα παραδείγματα.

Στην εντολή **for** υπάρχει μια πρόταση που εκτελείται μόνο **μία φορά στην αρχή**, μία **συνθήκη** (συνήθως λογική) που ελέγχεται **κάθε φορά πριν από την εκτέλεση του βρόχου**, και μία πρόταση που εκτελείται **κάθε φορά μετά από την εκτέλεση του βρόχου**. Η συνθήκη μπορεί να είναι οποιαδήποτε αριθμητική ή λογική παράσταση. Στη C δεν είναι απαραίτητο αυτές οι τρεις προτάσεις να έχουν κάποια σχέση μεταξύ τους.

Στο επόμενο παράδειγμα οι τρεις προτάσεις της εντολής **for** δεν έχουν κάποια σχέση μεταξύ τους:

```
for (a=2;b>4;c++)
{
    .....
}
```

- Καταχωρίζει στη μεταβλητή **a** το 2.
- Ελέγχει αν το **b>4** και, αν είναι, εκτελεί τις εντολές του βρόχου.
- Κάθε φορά, μετά από την εκτέλεση του βρόχου αυξάνει το **c** κατά 1 (**c++**).
- Η εκτέλεση του βρόχου σταματάει όταν το **b** δεν είναι πλέον μεγαλύτερο από το 4.

Ο επόμενος βρόχος **for δημιουργεί απλώς μια καθυστέρηση:**

```
for (i=1;i<=1000;i++);
```

🔗 Παρατηρούμε ότι δεν υπάρχουν προτάσεις στο σώμα του βρόχου.

Ο βρόχος που ακολουθεί απλώς περιμένει να πατηθεί το πλήκτρο του χαρακτήρα A:

```
for (ch='*';ch!='A';ch=getch());
```

🔗 Ούτε εδώ υπάρχουν προτάσεις στο σώμα του βρόχου.

Ο επόμενος βρόχος θα εμφανίσει στην οθόνη τους αριθμούς από το 1 μέχρι το 1000:

```
for (a=1;a<=1000;printf("%d\n",a++));
```

🔗 Και πάλι δεν υπάρχουν προτάσεις στο σώμα του βρόχου.

Μετά το τέλος ενός βρόχου **for**, η μεταβλητή του βρόχου περιέχει την τελευταία τιμή για την οποία **δεν εκτελέστηκε** ο βρόχος. Για παράδειγμα:

```
for (i=1;i<=20;i=i+6)
{
    προτάσεις βρόχου;
}
printf("i=%d\n",i);
```

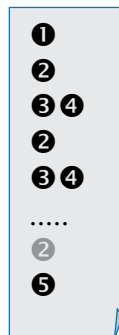
Η μεταβλητή **i** αυξάνεται κατά 6!

Μόλις ολοκληρωθεί ο βρόχος, η μεταβλητή **i** θα έχει την τιμή 25 που είναι η επόμενη τιμή του **i** μετά το 19 για την οποία **δεν** έγινε η επανάληψη του βρόχου. Ο βρόχος επαναλαμβάνεται για τις τιμές του **i**: 1, 7, 13, και 19.

Χορεύοντας με τη for!

Για να κατανοήσουμε πλήρως τον τρόπο λειτουργίας της **for**, αλλά και για να τον θυμόμαστε πιο εύκολα, θα προσομοιάσουμε κάθε πρότασή της με τα βήματα ενός χορού. Στον παρακάτω βρόχο **for** έχουμε αριθμήσει τις προτάσεις του βρόχου από το 1 μέχρι το 4, με τη σειρά με την οποία εκτελούνται. Με 5 έχουμε αριθμήσει την αμέσως επόμενη πρόταση μετά από τον βρόχο.

```
for (1; 2; 4)
{
    3;
}
5
```



Αν λοιπόν «χορεύαμε» στον ρυθμό της **for**, τα βήματα θα ήταν: μία φορά το **1** και μετά συνεχώς θα επαναλαμβάναμε τα βήματα **2-3-4** μέχρι στο βήμα **2** η παράσταση να μην είναι αληθής. Τότε τελειώνει ο χορός και η επαναληπτική διαδικασία! Μετά το τέλος της επαναληπτικής διαδικασίας εκτελείται η αμέσως επόμενη πρόταση μετά από τον βρόχο, δηλαδή η πρόταση **5**.

Απλά παραδείγματα

Το επόμενο τμήμα κώδικα εμφανίζει τους αριθμούς από το 5 μέχρι το 0:

```
for (i=5; i>=0; i--)
    printf("%d\n", i);
```

Το επόμενο τμήμα κώδικα εμφανίζει τα πολλαπλάσια του 5, από το 5 μέχρι το 100:

```
for (i=5; i<=100; i=i+5)
    printf("%d\n", i);
```

Το επόμενο τμήμα κώδικα εμφανίζει όλους τους λατινικούς χαρακτήρες από τον χαρακτήρα 'a' έως τον 'z' και δίπλα από κάθε χαρακτήρα τον κωδικό ASCII του:

```
char ch;
for (ch='a'; ch<='z'; ch++)
{
    printf("%c %d\n", ch, ch);
    printf("-----\n");
}
```

```
a 97
-----
b 98
-----
c 99
-----
...
```

Δεν πρέπει να ξεχνάμε ότι μια μεταβλητή χαρακτήρα περιέχει έναν ακέραιο αριθμό: τον κωδικό ASCII του χαρακτήρα που έχουμε καταχωρίσει σε αυτήν.

Επεξεργασία δεδομένων με επαναληπτικές διαδικασίες

Καταμέτρηση και άθροιση

Αρκετά συχνά, στα προγράμματά μας πρέπει να μετράμε ένα πλήθος (για παράδειγμα, το πλήθος των αριθμών που είναι μεγαλύτεροι από 10), αλλά και να αθροίζουμε τις διαφορετικές τιμές που λαμβάνει μια μεταβλητή. Για την καταμέτρηση ενός πλήθους χρησιμοποιούμε πάντα μια μεταβλητή στην οποία δίνουμε αρχική τιμή 0, και κάθε φορά που ικανοποιείται μια συνθήκη αυξάνουμε τη μεταβλητή κατά 1. Για το συνολικό άθροισμα διαφορετικών τιμών χρησιμοποιούμε πάλι μια μεταβλητή, στην οποία δίνουμε επίσης αρχική τιμή 0, και κάθε φορά που δημιουργείται μια νέα τιμή αυξάνουμε τη μεταβλητή κατά την τιμή αυτή.

Το επόμενο πρόγραμμα διαβάζει έναν προς έναν οκτώ ακέραιους αριθμούς που δίνονται από το πληκτρολόγιο και εμφανίζει το άθροισμά τους. Για την καταμέτρηση των αριθμών χρησιμοποιείται η μεταβλητή `cnt`, και για το άθροισμά τους χρησιμοποιείται η μεταβλητή `sum`. Στον κώδικα του προγράμματος χρησιμοποιείται η συνάρτηση `scanf()`, η οποία διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή `a`.

```
#include <stdio.h>
int main(void)
{
    int a,cnt=0,sum=0;
    while (cnt<8)
    {
        printf("Δώσε αριθμό:");
        scanf("%d",&a);
        cnt++;
        sum=sum+a;
    }
    printf("Άθροισμα=%d\n",sum);
    return 0;
}
```

8_sum.c

Θέτουμε ως αρχική τιμή το 0 στις μεταβλητές `cnt` και `sum`.

Όταν δοθούν 8 αριθμοί, η επαναληπτική διαδικασία σταματάει.

Διαβάζει έναν αριθμό από το πληκτρολόγιο και τον καταχωρίζει στη μεταβλητή `a`.

Κάθε φορά που δίνεται ένας αριθμός (`a`), η `cnt` αυξάνεται κατά 1. Επίσης, ο αριθμός (`a`) προστίθεται στη `sum`.

Τέλος, εμφανίζει το συνολικό άθροισμα όλων των αριθμών που δόθηκαν.

Ας υποθέσουμε ότι δίνουμε με τη σειρά τους αριθμούς 3, 4, 78, 9 ... όπως φαίνεται στην πρώτη στήλη του διπλανού πίνακα. Κάθε φορά που εκτελείται η πρόταση `cnt++`, αυξάνεται η τιμή της μεταβλητής `cnt` κατά 1. Έτσι, την πρώτη φορά που θα διαβάσει τον πρώτο αριθμό η `cnt` θα γίνει 1, όταν διαβάσει το 4 θα γίνει 2, με το 78 θα γίνει 3, κ.ο.κ. Όταν διαβάσει τον τελευταίο αριθμό, η `cnt` θα γίνει 8.

Με λίγα λόγια, χρησιμοποιήσαμε τη μεταβλητή `cnt` για την καταμέτρηση των αριθμών που διαβάσαμε.

a	cnt	sum
3	1	3
4	2	7
78	3	85
9	4	94
12	5	106
34	6	140
45	7	185
6	8	191

Με την πρόταση `sum=sum+a`, κάθε φορά που το πρόγραμμα διαβάζει έναν αριθμό, τον προσθέτει στο τρέχον περιεχόμενο της `sum`. Την πρώτη φορά (όταν η `a` περιέχει το 3) η

`sum` θα γίνει 3, όταν το πρόγραμμα διαβάσει το 4 η `sum` θα γίνει 7 (3 που είχε από πριν + 4), όταν διαβάσει το 78 η `sum` θα γίνει 85 (7+78), κ.ο.κ. Όταν θα έχει διαβάσει τον τελευταίο αριθμό, η `sum` θα είναι 191, το οποίο είναι το άθροισμα όλων των αριθμών που δώσαμε μέχρι εκείνη τη στιγμή. Επομένως, η μεταβλητή `sum` χρησιμοποιήθηκε για να υπολογίσουμε το άθροισμα των αριθμών που δώσαμε.

Όταν θέλουμε να **μετρήσουμε** ένα πλήθος, χρησιμοποιούμε μια μεταβλητή στην οποία δίνουμε αρχική τιμή 0 (π.χ. `cnt=0`) και κάθε φορά που ικανοποιείται μια συνθήκη αυξάνουμε την τιμή της μεταβλητής κατά 1 (`cnt++`). Τη μεταβλητή αυτή την ονομάζουμε **μετρητή**. Όταν θέλουμε να **αθροίσουμε** τις διαφορετικές τιμές που παίρνει μια μεταβλητή (π.χ. η `a`) χρησιμοποιούμε μια θέση μνήμης (την οποία καλούμε **αθροιστή**) στην οποία δίνουμε αρχική τιμή 0 (π.χ. `sum=0`), και κάθε φορά που η μεταβλητή παίρνει μια νέα τιμή αυξάνουμε τον αθροιστή κατά το περιεχόμενο της μεταβλητής αυτής (π.χ. `sum=sum+a`).

Υπολογισμός μέγιστου και ελάχιστου

Το παρακάτω πρόγραμμα διαβάζει δέκα ακέραιους αριθμούς που δίνονται από το πληκτρολόγιο και εμφανίζει τον μέγιστο και τον ελάχιστο από αυτούς. Ο μέγιστος αριθμός αποθηκεύεται στη θέση `max` και ο ελάχιστος στη θέση `min`.

Η τεχνική είναι η εξής: Διαβάζουμε τον πρώτο αριθμό και τον θεωρούμε ταυτόχρονα τον μέγιστο αλλά και τον ελάχιστο που έχουμε διαβάσει μέχρι στιγμής, καταχωρίζοντάς τον στις μεταβλητές `max` και `min`. Κατόπιν, με μια επαναληπτική διαδικασία, διαβάζουμε έναν προς έναν τους υπόλοιπους αριθμούς και τους συγκρίνουμε με τα περιεχόμενα των μεταβλητών `max` και `min`. Αν ο αριθμός είναι μεγαλύτερος από τη θέση `max` τον καταχωρίζουμε στη `max`, ενώ αν είναι μικρότερος από τη `min` τον καταχωρίζουμε στη `min`. Τελικά οι μεταβλητές `max` και `min` θα περιέχουν τον μεγαλύτερο και τον μικρότερο αριθμό αντίστοιχα.

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

8_minmax1.c

```
int main(void)
```

```
{
```

```
    int i, ar, min, max;
```

```
    for (i=1; i<=10; i++)
```

```
    {
```

```
        printf("Δώσε αριθμό:");
```

```
        scanf("%d", &ar);
```

```
        if (i==1)
```

```
            max=min=ar;
```

```
        if (ar>max)
```

```
            max=ar;
```

```
        else if (ar<min)
```

```
            min=ar;
```

```
    }
```

Διαβάζει έναν προς έναν τους 10 αριθμούς

Αν ο αριθμός είναι ο πρώτος που δόθηκε ($i=1$), καταχωρίζεται ως αρχική τιμή τόσο στη μεταβλητή `max` όσο και στη μεταβλητή `min`.

Στην περίπτωση που ο αριθμός `ar` έχει τιμή μεγαλύτερη από την τρέχουσα μέγιστη τιμή (`max`), καταχωρίζεται στη θέση `max`, ενώ αν έχει τιμή μικρότερη από την τρέχουσα ελάχιστη τιμή (`min`), καταχωρίζεται στη θέση `min`.

```
printf("max=%d\nmin=%d\n",max, min);
return 0;
}
```

Εμφανίζει τον μεγαλύτερο και μικρότερο αριθμό από αυτούς που δόθηκαν.

Το πρόγραμμα που ακολουθεί δεν γνωρίζει εκ των προτέρων το πλήθος των αριθμών, αλλά χρησιμοποιεί την τιμή 9999 ως τιμή-σήμα η οποία σηματοδοτεί το τέλος των ακέραιων αριθμών. Το πρόγραμμα υπολογίζει τον μέγιστο και τον ελάχιστο από αυτούς. Η τιμή-σήμα δεν περιλαμβάνεται στον υπολογισμό. Ο μέγιστος αριθμός φυλάσσεται στη θέση **max** και ο ελάχιστος στη θέση **min**. Η τεχνική είναι ίδια με εκείνη του προηγούμενου παραδείγματος. Η διαφορά είναι ότι η επαναληπτική διαδικασία υλοποιείται με έναν βρόχο **while**. Επίσης, κάθε φορά που εισάγεται ένας αριθμός, πρέπει να γίνεται έλεγχος μήπως είναι ο αριθμός-σήμα, οπότε δεν θα πρέπει να συμπεριληφθεί στους υπολογισμούς.

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

8_minmax2.c

```
int main(void)
```

```
{
```

```
    int i,ar,min,max,pl;
```

```
    pl=0;
```

```
    do
```

```
    {
```

```
        printf("Δώσε αριθμό:");
```

```
        scanf("%d",&ar);
```

```
        if (ar!=9999)
```

```
        {
```

```
            pl++;
```

```
            if (pl==1)
```

```
                min=max=ar;
```

```
            if (ar>max)
```

```
                max=ar;
```

```
            else if (ar<min)
```

```
                min=ar;
```

```
        }
```

```
    } while (ar!=9999);
```

```
    if (pl!=0)
```

```
        printf("max=%d\nmin=%d\n",max, min);
```

```
    else
```

```
        printf("Δεν έδωσες κανέναν αριθμό");
```

```
    return 0;
```

```
}
```

Διαβάζει έναν προς έναν τους αριθμούς

Αν ο αριθμός που δόθηκε δεν είναι αριθμός-σήμα, τότε προχωράμε στην επεξεργασία του.

Αν ο αριθμός είναι αποδεκτός, η θέση **pl** αυξάνεται κατά 1.

Αν ο αριθμός είναι ο πρώτος που δόθηκε (**pl==1**), καταχωρίζεται ως αρχική τιμή τόσο στη μεταβλητή **max**, όσο και στη μεταβλητή **min**.

Συγκρίνει κάθε αριθμό με τις θέσεις **max** και **min**.

Η επαναληπτική διαδικασία σταματάει όταν δοθεί ο αριθμός-σήμα 9999!

Εμφανίζει τον μεγαλύτερο και μικρότερο αριθμό από αυτούς που δόθηκαν. Αν δεν έχει δοθεί κανένας αριθμός εμφανίζει κατάλληλο μήνυμα.

Παρατηρούμε ότι στο παραπάνω πρόγραμμα χρησιμοποιούμε τη μεταβλητή **pl** για να μετράμε το πλήθος των αποδεκτών αριθμών που δίνονται χωρίς αυτό να ζητείται. Αυτό γίνεται μόνο και μόνο για να γνωρίζουμε αν δόθηκε έστω και ένας αριθμός (**pl!=0**), ή αν ο αριθμός που δόθηκε είναι ο πρώτος (**pl==1**).

Συχνά λάθη

- ❖ Ένα συχνό λάθος είναι ότι ξεχνάμε να τερματίσουμε τη δομή **do...while** με ερωτηματικό.
- ❖ Ξεχνάμε επίσης να τοποθετήσουμε **ολόκληρη** τη λογική παράσταση της **while** μέσα σε παρενθέσεις. Δηλαδή, γράφουμε **while a>5** αντί του σωστού **while (a>5)**.
- ❖ Κατά λάθος μπορεί να χωρίσουμε τις προτάσεις της **for** (μέσα στην παρένθεση) με κόμματα και όχι με ερωτηματικά.
- ❖ Νομίζουμε ότι οι προτάσεις μέσα στη παρένθεση της **for** πρέπει να σχετίζονται μεταξύ τους. Στην πραγματικότητα όμως μπορεί να μην έχουν καμία σχέση.
- ❖ Αν χρησιμοποιήσουμε την **goto** κάνουμε το πιο μοιραίο λάθος του δομημένου προγραμματισμού!

Ανασκόπηση Κεφαλαίου 8

- Η εντολή **while** χρησιμοποιείται για την επαναλαμβανόμενη εκτέλεση προτάσεων ενόσω μια λογική συνθήκη είναι αληθής. Ο έλεγχος της συνθήκης γίνεται κάθε φορά πριν από την εκτέλεση των προτάσεων.
- Η εντολή **do-while** χρησιμοποιείται για την επαναλαμβανόμενη εκτέλεση προτάσεων ενόσω μια συνθήκη είναι αληθής. Ο έλεγχος της συνθήκης γίνεται κάθε φορά μετά από την εκτέλεση των προτάσεων, επομένως οι προτάσεις είναι βέβαιο ότι θα εκτελεστούν τουλάχιστον μία φορά.
- Η εντολή **for** χρησιμοποιείται επίσης για την επαναλαμβανόμενη εκτέλεση προτάσεων. Το πλήθος των επαναλήψεων στη **for** συνήθως εξαρτάται από την τιμή μιας μεταβλητής.
- Η εκτέλεση ενός βρόχου **for** καθορίζεται από τρεις προτάσεις οι οποίες μπορεί να είναι εντελώς ανεξάρτητες μεταξύ τους.
- Η εντολή **goto**, αν και υποστηρίζεται από τη C για τη μεταφορά της ροής του προγράμματος σε ένα διαφορετικό σημείο, θα πρέπει να μην χρησιμοποιείται ποτέ επειδή οδηγεί σε δυσνόητο και μη δομημένο κώδικα.
- Η εντολή **break** τερματίζει έναν βρόχο **while**, **do-while**, ή **for**.
- Η εντολή **continue** επιβάλλει την εκτέλεση της επόμενης επανάληψης σε έναν βρόχο **while**, **do-while** ή **for**, παρακάμπτοντας τις ενδιάμεσες προτάσεις.
- Ο τελεστής κόμμα (,) συνδέει πολλές παραστάσεις μαζί. Αν και εκτελούνται όλες οι παραστάσεις, το αποτέλεσμα ολόκληρης της πρότασης είναι το αποτέλεσμα μόνο της τελευταίας παράστασης.

Ασκήσεις Κεφαλαίου 8

- 8.1** Να γραφεί ένα πρόγραμμα το οποίο να υπολογίζει το άθροισμα των αριθμών από το 1 μέχρι το 1000. ★★
- 8.2** Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει χαρακτήρες από το πληκτρολόγιο. Όταν διαβάσει τον χαρακτήρα αστερίσκου (*), το πρόγραμμα πρέπει να σταματάει και να εμφανίζει το πλήθος των λατινικών κεφαλαίων χαρακτήρων που πληκτρολογήθηκαν. ★★

- 8.3** Τι κάνει το επόμενο πρόγραμμα (το 27 είναι ο κωδικός του πλήκτρου <Esc>); ★★

```
#include <stdio.h>
#include <stdbool.h>
int main(void)
{
    char ch;
    int a;
    bool fl=false;
    ch=1;
    a=0;
    while (ch!=27)
    {
        ch=getch();
        if (ch=='*') fl=true;
        if (fl) ++a;
    }
    printf("%d χαρακτήρες\n",a);
    return 0;
}
```

- 8.4** Τι κάνει το επόμενο πρόγραμμα; Ποιο το νόημα του περιεχομένου των μεταβλητών `num1` και `num2`; ★★

```
#include <stdio.h>
int main(void)
{
    int a,num,num1,num2;
    num1=num2=0;
    for (a=1;a<=100;++a)
    {
        scanf("%d",&num);
        switch (num%2)
        {
            case 0:
```

8.31 Ξαναγράψτε το παρακάτω πρόγραμμα χρησιμοποιώντας δομές επιλογής και επανάληψης που κάνουν περιττή τη χρήση της **goto**. ★★

```
.....
int main(void)
{
    float b1,b2,b3,mo;
    arxi:
    scanf("%f %f %f",&b1,&b2,&b3);
    if (b1==0 && b2==0 && b3==0) goto telos;
    mo=(b1+b2+b3)/3;
    if (mo<5) goto kopikes;
    printf("Πέρασες!!!\n");
    printf("Μέσος όρος %f\n",mo);
    goto arxi;
    kopikes:
    printf("Κόπηκες!!!\n");
    printf("Μέσος όρος %f\n",mo);
    goto arxi;
    telos:
    printf("Τέλος\n");
    return 0;
}
```

8.32 Ξαναγράψτε το παρακάτω πρόγραμμα χωρίς τη χρήση της **goto**. ★★★

```
.....
int main(void)
{
    int i,j;
    i=1;
    arxi1:
    if (i>5) goto telos1;
    j=1;
    arxi2:
    if (j>5) goto telos2;
    if ((i+j)%2==0) goto edo;
    if (i*j>=16) goto telos2;
    printf("%d %d\n",i,j);
    edo:
    j++;
    goto arxi2;
    telos2:printf("=====\n");
    i++;
    goto arxi1;
    telos1:printf("Τέλος\n");
    return 0;
}
```

1η Εργασία - Καρναβάλι

Για να είμαστε έγκαιρα προετοιμασμένοι για τις αποκριάτικες στολές, θέλουμε να γνωρίζουμε πότε είναι η Κυριακή των Απόκρεω κάθε έτους. Ο αλγόριθμος που ακολουθεί βρίσκει την ημερομηνία της αποκριάς για κάθε έτος E που δίνουμε.

Υπολογίζουμε τις παρακάτω τιμές:

- A = Υπόλοιπο της ακέрайς διαίρεσης του E με το 19
- B = Υπόλοιπο της ακέрайς διαίρεσης του E με το 4
- Γ = Υπόλοιπο της ακέрайς διαίρεσης του E με το 7
- Δ = Υπόλοιπο της ακέрайς διαίρεσης του $(19 * A + 16)$ με το 30
- Z = Υπόλοιπο της ακέрайς διαίρεσης του $(2 * B + 4 * \Gamma + 6 * \Delta)$ με το 7
- Φ = 1 αν το έτος E είναι δίσεκτο και 0 αν δεν είναι

Για τον υπολογισμό του Φ υπενθυμίζουμε ότι δίσεκτα είναι τα έτη που διαιρούνται ακριβώς με το 4 και δεν διαιρούνται ακριβώς με το 100. Επίσης, δίσεκτα είναι τα έτη που διαιρούνται ακριβώς με το 400. Δηλαδή, αν το έτος είναι δίσεκτο η τιμή του Φ πρέπει να είναι 1, διαφορετικά 0.

Η ημερομηνία των Απόκρεω υπολογίζεται από την παράσταση $\Delta + Z + \Phi + 13$ η οποία επιστρέφει το πλήθος των ημερών από την 1η Φεβρουαρίου (συμπεριλαμβανομένης) μέχρι τις Απόκριες! Για παράδειγμα, αν το αποτέλεσμα είναι 15, η Κυριακή των Απόκρεω θα είναι 15 Φεβρουαρίου. Αν το αποτέλεσμα είναι 35 τότε θα είναι 7 Μαρτίου (35 - 28 ημέρες που έχει ο Φεβρουάριος) στην περίπτωση κανονικού έτους, και 6 Μαρτίου (35 - 29 ημέρες που έχει ο Φεβρουάριος) στην περίπτωση δίσεκτου έτους.

Σημείωση: Η Κυριακή των Απόκρεω είναι πάντοτε τον Φεβρουάριο ή τον Μάρτιο και σύμφωνα με τον τύπο δεν μπορεί να είναι νωρίτερα από τις 13 Φεβρουαρίου!

Να φτιάξετε ένα πρόγραμμα το οποίο θα υπολογίζει και θα εμφανίζει την ημερομηνία που πέφτουν οι Απόκριες. Το πρόγραμμα θα πρέπει να έχει επαναληπτική λειτουργία. Θα πρέπει να ζητάει από τον χρήστη ένα έτος έναρξης και ένα έτος λήξης, και να εμφανίζει την ημερομηνία της Κυριακής των Απόκρεω για όλα τα έτη από το έτος έναρξης μέχρι το έτος λήξης.

Οδηγίες - Βοήθεια:

- Η ακέрайς διαίρεση γίνεται με τον τελεστή διαίρεσης / όταν εφαρμόζεται σε ακέрайς δεδομένα.
- Το υπόλοιπο της ακέрайς διαίρεσης υπολογίζεται με χρήση του τελεστή %.
- Αρχικά σχεδιάστε το πρόγραμμα χωρίς επαναληπτική λειτουργία, επιβεβαιώστε ότι λειτουργεί σωστά, και μετά προσθέστε αυτή τη δυνατότητα.
- Στον φάκελο ΕΡΓΑΣΙΕΣ/Ε1 του συνοδευτικού υλικού θα βρείτε το εκτελέσιμο αρχείο του προγράμματος, για να δείτε πώς λειτουργεί αλλά και για να δοκιμάσετε το πρόγραμμα που θα φτιάξετε εσείς!



Κεφάλαιο

Συναρτήσεις

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ

Στο κεφάλαιο αυτό παρουσιάζεται η δυνατότητα τμηματοποίησης του προγράμματος με τη χρήση συναρτήσεων. Έτσι, ο κώδικας του προγράμματος παύει να βρίσκεται πλέον μέσα σε μία μόνο συνάρτηση, αλλά «καταμερίζεται» σε διαφορετικές συναρτήσεις, κάθε μία από τις οποίες υλοποιεί μια ξεχωριστή λειτουργία του.

Αναλύεται ο τρόπος ορισμού και κλήσης συναρτήσεων, με ιδιαίτερη έμφαση στον τρόπο με τον οποίο μπορούμε να μεταβιβάσουμε πληροφορίες σε μια συνάρτηση, αλλά και το πώς μια συνάρτηση επιστρέφει πληροφορίες στον κώδικα που την καλεί.

Τα παραδείγματα ακολουθούν τη φιλοσοφία της «τμηματοποίησης» του προγράμματος σε συναρτήσεις, δείχνοντας ότι η προσέγγιση αυτή είναι μονόδρομος για τον σωστό σχεδιασμό και υλοποίηση προγραμμάτων.

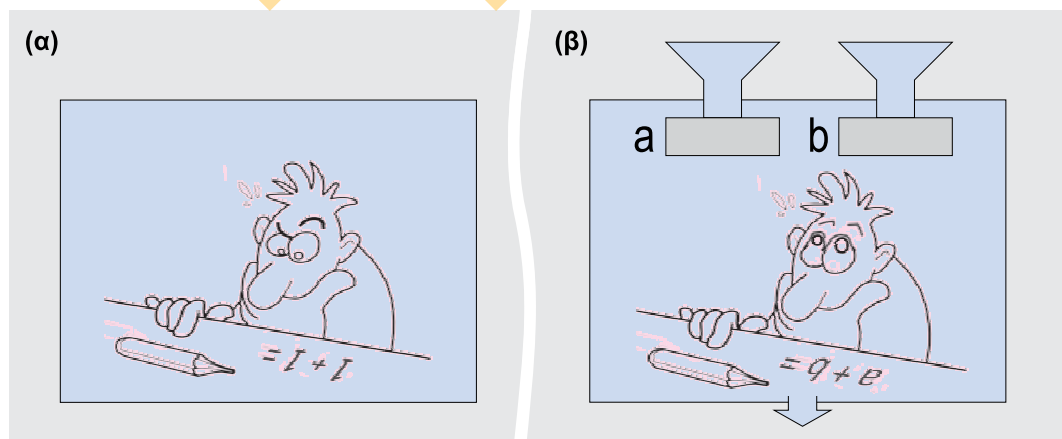
Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφερθούμε στη χρήση των συναρτήσεων στη C. Μέχρι στιγμής είδαμε τη χρήση της `main()`, μιας συνάρτησης απαραίτητης σε κάθε πρόγραμμα της C. Επίσης συναντήσαμε και ορισμένες συναρτήσεις βιβλιοθήκης, όπως οι: `printf()`, `scanf()`, `getchar()`, κ.ά.

Οι συναρτήσεις βοηθούν τον προγραμματιστή να «τεμαχίζει» μεγάλα προγράμματα σε μικρότερα τμήματα, κάθε ένα από τα οποία επιτελεί μια συγκεκριμένη λειτουργία.

Φανταστείτε μια συνάρτηση σαν έναν υπάλληλο που κάνει μια συγκεκριμένη δουλειά και τον έχουμε κλείσει μέσα σε ένα κουτί (εντάξει, δεν είναι πολύ δημοκρατικό αλλά το κάνουμε για εκπαιδευτικούς λόγους)! Αν το κουτί είναι ερμητικά κλειστό, τότε δεν έχουμε τη δυνατότητα να μεταβιβάσουμε στον υπάλληλο καμία πληροφορία. Κάθε φορά θα κάνει ακριβώς την ίδια δουλειά. Για παράδειγμα, στην περίπτωση του Σχήματος 9.1α, ο υπάλληλός μας θα υπολογίζει κάθε φορά την πράξη $1+1$!

Αν θέλουμε τώρα ο υπάλληλος να μην προσθέτει πάντα τους ίδιους αριθμούς, αλλά αριθμούς που θα παίρνει από το εξωτερικό περιβάλλον του, θα πρέπει να βρούμε έναν τρόπο για να του μεταβιβάσουμε αυτές τις πληροφορίες. Το κουτί μας δεν πρέπει πλέον να είναι ερμητικά κλειστό αλλά να διαθέτει οπές μέσω των οποίων θα μπορούμε να μεταβιβάσουμε τις πληροφορίες που θέλουμε στον υπάλληλο. Στο παράδειγμα του Σχήματος 9.1β, το κουτί διαθέτει δύο οπές οι οποίες καταλήγουν στα κουτιά-μεταβλητές `a` και `b`. Εκτός των άλλων, ο υπάλληλός μας πάσχει και αυτός από Αλτσχάιμερ, οπότε όποιες πληροφορίες και να του μεταβιβάσουμε θα τις ξεχάσει αμέσως αν δεν αποθηκευτούν κάπου. Αν ρίξουμε έναν αριθμό στη πρώτη οπή θα καταλήξει στο κουτί-μεταβλητή `a`, ενώ αν ρίξουμε έναν αριθμό στη δεύτερη οπή θα καταλήξει στο κουτί-μεταβλητή `b`. Ο υπάλληλος τώρα υπολογίζει το άθροισμα των περιεχομένων των κουτιών `a` και `b`, δηλαδή το άθροισμα των αριθμών που του μεταβιβάσαμε.



Σχήμα 9.1 Οι συναρτήσεις και η επικοινωνία με το περιβάλλον τους.

Αυτές οι «οπές» λέγονται **παράμετροι** και δίνουν τη δυνατότητα στον κώδικα του προγράμματος να μεταβιβάζει πληροφορίες στις συναρτήσεις.

Στο παράδειγμα του Σχήματος 9.1β παρατηρούμε ότι υπάρχει και μια άλλη οπή, αυτή με το βέλος στο κάτω μέρος του κουτιού, μέσω της οποίας ο υπάλληλος (συνάρτηση) μπορεί να επιστρέψει στο περιβάλλον του κάποια πληροφορία (τιμή).

Δηλαδή, οι συναρτήσεις μπορούν να επιστρέψουν κάποια τιμή στο περιβάλλον που τις καλεί. Υπάρχουν βεβαίως και συναρτήσεις που δεν επιστρέφουν καμία τιμή.

Όπως αναφέρθηκε στην εισαγωγή, ένα πρόγραμμα στη C έχει την ακόλουθη γενική μορφή:

```
main()
{
    προτάσεις της συνάρτησης main();
}
function-1()
{
    προτάσεις της συνάρτησης function-1();
}
function-2()
{
    προτάσεις της συνάρτησης function-2();
}
function-N()
{
    προτάσεις της συνάρτησης function-N();
}
```

Ορισμός της απαραίτητης
συνάρτησης `main()`.

Ορισμοί των υπόλοιπων
(προαιρετικών) συναρτήσεων
του προγράμματος,

Οι συναρτήσεις εκτελούνται μόνο αν κληθούν από μια άλλη συνάρτηση. Η μόνη συνάρτηση που εκτελείται αυτόματα είναι η συνάρτηση `main()`.

Στη C, μια συνάρτηση μπορεί να επιστρέψει μια τιμή. Μια συνάρτηση έχει έναν τύπο (`int`, `float` κ.λπ.) ανάλογα με τον τύπο δεδομένων που επιστρέφει ως τιμή. Όπως θα γίνει κατανοητό στη συνέχεια του παρόντος κεφαλαίου, ακόμα και αν δεν επιστρέφει καμία τιμή θα πρέπει να δηλωθεί με τον κατάλληλο τύπο (`void`).

Επίσης, μια συνάρτηση μπορεί να έχει παραμέτρους μέσω των οποίων ο κώδικας που καλεί τη συνάρτηση μπορεί να της «μεταβιβάσει» πληροφορίες.

Στη C, κάθε συνάρτηση ανήκει σε κάποιον τύπο δεδομένων. Μια συνάρτηση μπορεί να έχει καμία, μία, ή περισσότερες παραμέτρους.

Ορισμός συνάρτησης

Μια συνάρτηση ορίζεται με τον ακόλουθο τρόπο:

```

τύπος_συνάρτησης όνομα_συνάρτησης (τύπος_παράμετρος1, τύπος_παράμετρος2, ...)
{
    προτάσεις_συνάρτησης;
}

```

- 🔗 Ο **τύπος της συνάρτησης** καθορίζει τον τύπο της τιμής που επιστρέφει η συνάρτηση.
- 🔗 Στην περίπτωση που η συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως τύπου **void**.
- 🔗 Οι **παράμετροι** της συνάρτησης (αν υπάρχουν) ορίζονται, μαζί με τον **τύπο** τους, μέσα στις παρενθέσεις που ακολουθούν το όνομα της συνάρτησης.
- 🔗 Οι μεταβλητές που δηλώνονται μέσα στο σώμα μιας συνάρτησης έχουν εμβέλεια μόνο μέσα στη συγκεκριμένη συνάρτηση και λέγονται **τοπικές** μεταβλητές.
- 🔗 Οι παράμετροι μιας συνάρτησης αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης.

Συναρτήσεις χωρίς παραμέτρους

Στην απλούστερη μορφή τους, οι συναρτήσεις δεν έχουν παραμέτρους. Σε αυτή την περίπτωση το πρόγραμμα που τις καλεί δεν μπορεί να τους μεταβιβάσει πληροφορίες. Μπορούμε να παρομοιάσουμε μια τέτοια συνάρτηση σαν ένα ερμητικά κλειστό κουτί το οποίο επιτελεί κάποια λειτουργία, αλλά δεν έχουμε καμία δυνατότητα να τροποποιήσουμε τη λειτουργία του (Σχήμα 9.2α).

Για παράδειγμα, η παρακάτω συνάρτηση εμφανίζει τους αριθμούς από το 5 μέχρι το 10. Δεν επιστρέφει καμία τιμή και για τον λόγο αυτόν δηλώνεται ως **void**. Δεν διαθέτει καμία παράμετρο και, κάθε φορά που καλείται, εκτελεί ακριβώς την ίδια λειτουργία.

```

void display_numbers()
{
    int i;
    for (i=5; i<=10; i++) printf("%d\n", i);
}

int main(void)
{
    printf("Πρώτη κλήση της συνάρτησης\n");
    display_numbers();
    printf("Δεύτερη κλήση της συνάρτησης\n");
    display_numbers();
    return 0;
}

```

Η συνάρτηση εμφανίζει τους αριθμούς 5, 6, 7, 8, 9, και 10.

Κλήση της συνάρτησης

Κλήση της συνάρτησης

Η συνάρτηση `main()` καλεί τη συνάρτηση `display_numbers()` δύο φορές. Κάθε φορά η συνάρτηση εμφανίζει τους ίδιους αριθμούς. Από τη στιγμή που η συνάρτηση δεν έχει παραμέτρους, δεν υπάρχει τρόπος να επηρεάσουμε τη λειτουργία της.

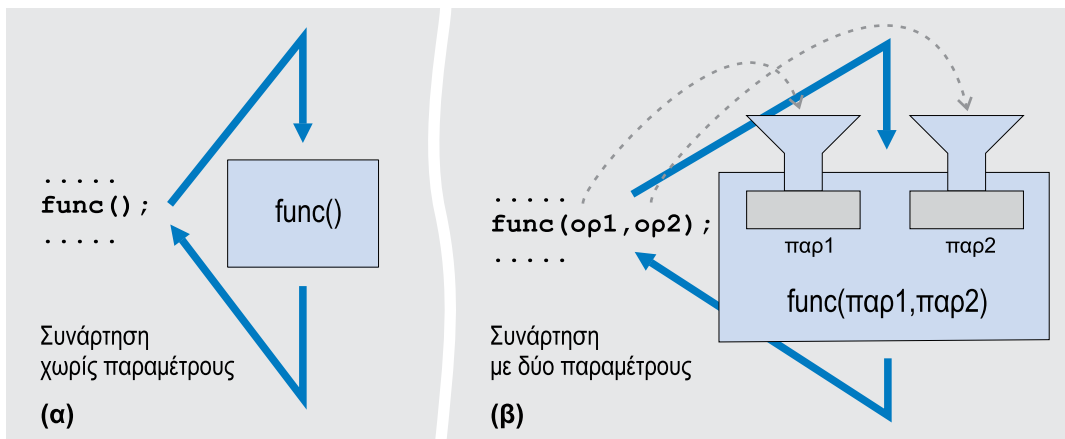
Συναρτήσεις με παραμέτρους

Οι παράμετροι είναι ένας τρόπος με τον οποίο ένα τμήμα προγράμματος μεταβιβάζει πληροφορίες σε μια συνάρτηση.

Μπορούμε να παρομοιάσουμε μια τέτοια συνάρτηση σαν ένα κουτί το οποίο διαθέτει οπές, μέσω των οποίων μπορούμε να μεταβιβάζουμε πληροφορίες. Ο κώδικας της συνάρτησης χρησιμοποιεί αυτές τις πληροφορίες και μπορεί να τροποποιεί ανάλογα τη λειτουργία του. Όταν καλούμε μια συνάρτηση με παραμέτρους, την καλούμε με ισάριθμα ορίσματα.

Το Σχήμα 9.2β δείχνει μια συνάρτηση με δύο παραμέτρους, τη `func(παρ1,παρ2)`, σαν ένα κουτί με δύο οπές οι οποίες καταλήγουν στις μεταβλητές-παραμέτρους `παρ1` και `παρ2` αντίστοιχα. Όταν καλούμε τη συνάρτηση `func()`, πρέπει να χρησιμοποιήσουμε ισάριθμα ορίσματα `func(ορ1,ορ2)`. Η τιμή του πρώτου ορίσματος μεταβιβάζεται στην πρώτη παράμετρο (οπή), και η τιμή του δεύτερου ορίσματος μεταβιβάζεται στη δεύτερη.

Στο πρόγραμμα που ακολουθεί, η συνάρτηση `display_numbers()` είναι μια παραλλαγή της συνάρτησης που συναντήσαμε στη προηγούμενη ενότητα, η οποία τώρα διαθέτει δύο παραμέτρους ώστε ο κώδικας που την καλεί να μπορεί να καθορίζει το εύρος των αριθμών που εμφανίζει. Η συνάρτηση χρησιμοποιεί τρεις μεταβλητές, τις `i`, `apo` και `eos`. Οι μεταβλητές `apo` και `eos` αποτελούν τις παραμέτρους της συνάρτησης. Και οι τρεις αυτές μεταβλητές καλούνται **τοπικές μεταβλητές** και έχουν εμβέλεια μόνο μέσα στη συγκεκριμένη συνάρτηση. Ο ρόλος και η εμβέλεια των τοπικών μεταβλητών θα αναλυθεί στο επόμενο κεφάλαιο.



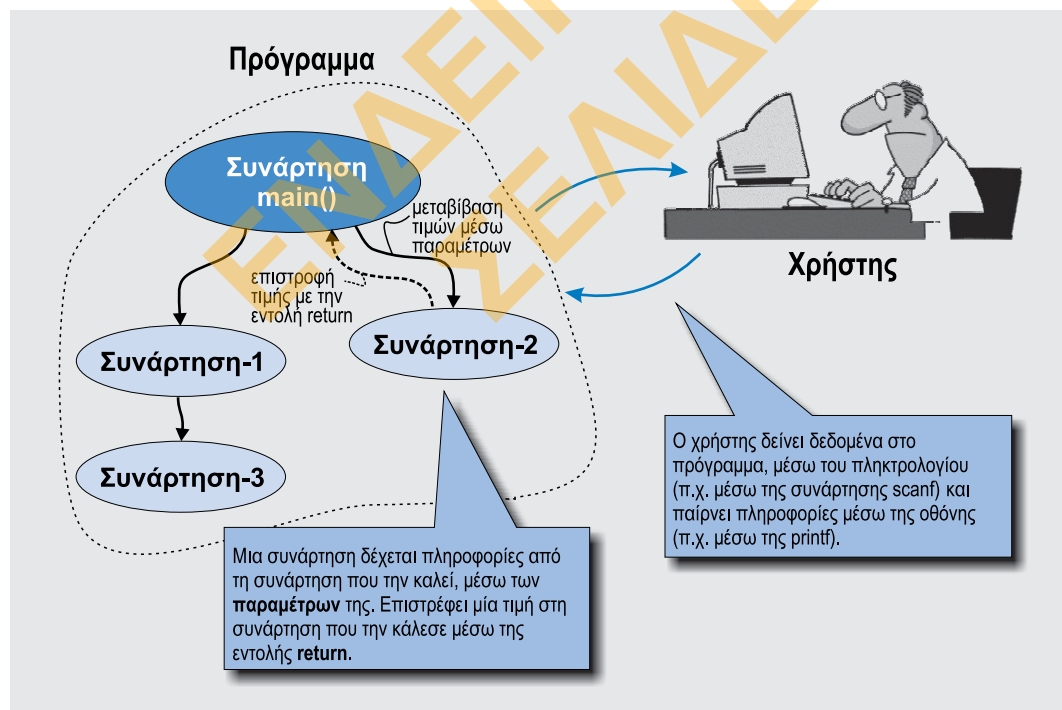
Σχήμα 9.2 Συνάρτηση χωρίς παραμέτρους (α) και συνάρτηση με παραμέτρους (β).

Ένα ξεκαθάρισμα...

Αμφιταλαντευόμουν αν θα έπρεπε να προσθέσω αυτή την ενότητα στην αρχή ή στο τέλος του κεφαλαίου. Παρά το γεγονός ότι φαίνεται να είναι «εισαγωγική», προτίμησα να την προσθέσω στο τέλος ώστε να ξεκαθαριστούν κάποια πράγματα, τα οποία έχω διαπιστώσει ότι προκαλούν σύγχυση στους σπουδαστές, και αφορούν στη μεταβίβαση παραμέτρων.

Ας μην ξεχνάμε ότι σε αυτό το παιχνίδι συμμετέχουν δύο παίκτες: Ο χρήστης και το πρόγραμμα! Ο χρήστης βλέπει το πρόγραμμα σαν κάτι μαγικό που απλώς του κάνει τη δουλειά του. Έτσι, δίνει δεδομένα στο πρόγραμμα και παίρνει αποτελέσματα από αυτό, χωρίς να τον απασχολεί αν τη δουλειά την έκανε η συνάρτηση `main()` ή οποιαδήποτε άλλη συνάρτηση του προγράμματος. Ο χρήστης αγνοεί την ύπαρξη των συναρτήσεων και βλέπει το πρόγραμμα ως μια ενιαία οντότητα.

Ας παρομοιάσουμε λοιπόν ένα πρόγραμμα με μια εταιρεία η οποία μπορεί να αποτελείται από έναν ή περισσότερους υπαλλήλους (συναρτήσεις). Ένας από αυτούς είναι ο διευθυντής (η συνάρτηση `main()`). Για να υλοποιήσει η εταιρεία (πρόγραμμα) αυτά που θέλει ο πελάτης (χρήστης), επιστρατεύει τους υπαλλήλους της (συναρτήσεις). Για την ολοκλήρωση των εργασιών, οι υπάλληλοι-συναρτήσεις ανταλλάσσουν μεταξύ τους πληροφορίες.

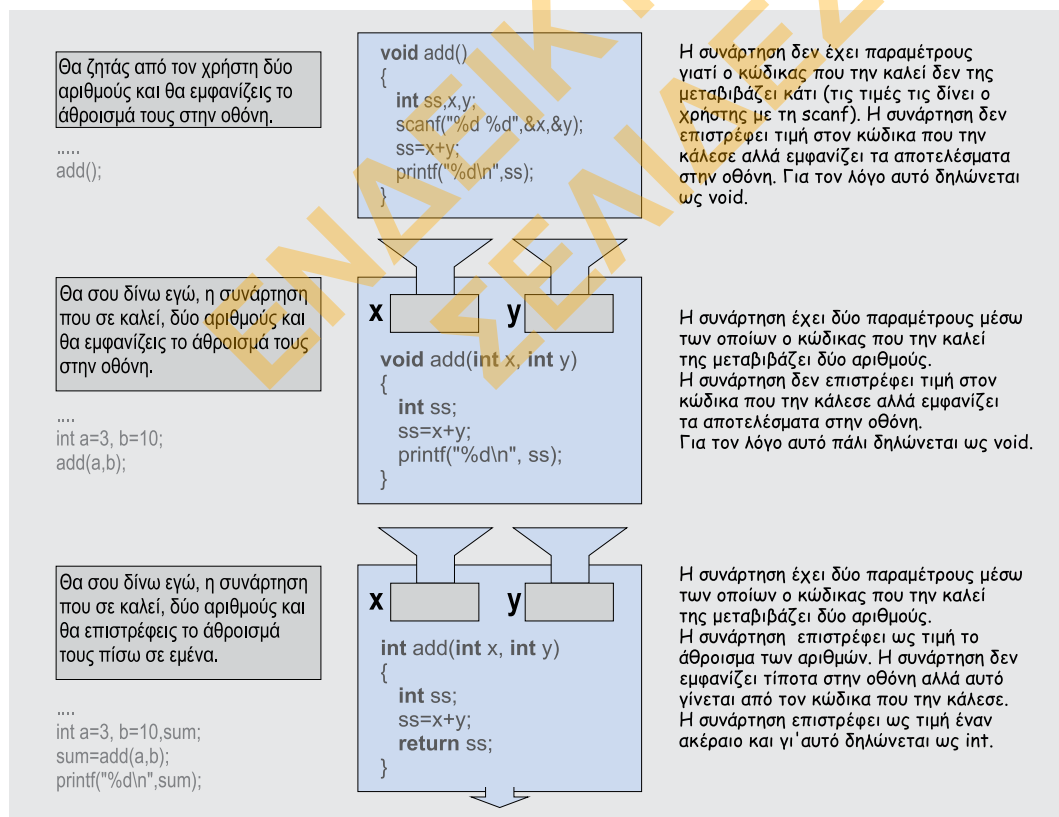


Σχήμα 9.5 Ανταλλαγή πληροφοριών μεταξύ προγράμματος-χρήστη και συναρτήσεων.

Μια συνάρτηση δέχεται πληροφορίες μέσω των παραμέτρων της. Σε αρκετούς σπουδα-
στές υπάρχει η εσφαλμένη εντύπωση ότι οι τιμές των παραμέτρων μιας συνάρτησης πα-
ρέχονται από τον χρήστη και ότι η τιμή που επιστρέφει η συνάρτηση εμφανίζεται στην
οθόνη. **Στην πραγματικότητα, οι τιμές μεταβιβάζονται στις παραμέτρους μιας συ-
νάρτησης από τη συνάρτηση που την καλεί, και η τιμή που επιστρέφει μια συνάρ-
τηση επιστρέφεται στη συνάρτηση που την κάλεσε.** Φυσικά, μια συνάρτηση μπορεί
να δέχεται δεδομένα από τον χρήστη και να εμφανίζει πληροφορίες στην οθόνη, αλλά
αυτό δεν έχει καμία σχέση με τις παραμέτρους της και με την τιμή που επιστρέφει. Το
Σχήμα 9.5 δείχνει εποπτικά αυτή τη διαδικασία.

Ο χρήστης επικοινωνεί με το πρόγραμμα μέσω συναρτήσεων εισόδου και εξόδου (π.χ. scanf, printf κ.λ.π.). Οι συναρτήσεις δέχονται πληροφορίες μέσω των παραμέτρων τους και επιστρέφουν μια τιμή μέσω της εντολής **return**.

Μελετήστε προσεκτικά τα παραδείγματα συναρτήσεων του Σχήματος 9.6, οι οποίες υ-
λοποιούν τις απαιτήσεις που εμφανίζονται στα πλαίσια αριστερά.



Σχήμα 9.6 Παραδείγματα συναρτήσεων

Ασκήσεις Κεφαλαίου 9

9.1 Βρείτε τα λάθη του επόμενου προγράμματος: ★

```
#include <stdio.h>
int main(void)
{
    double x,y;
    printf("*****\n");
    scanf("%f %f",x,y);
    printf("mo=%f\n",mo(x,y));
    return 0;
}
float mo(int a, int b);
{
    if (a==0 && b==0)
        return 0;
    else
        mesos=(a+b)/2.0;
    return mesos;
}
```

9.2 Να γραφεί μια συνάρτηση που να κάνει τα εξής:

- Να δέχεται τρεις παραμέτρους.
- Αν η πρώτη παράμετρος είναι 1, να επιστρέφει ως τιμή το άθροισμα των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος είναι 2, να επιστρέφει ως τιμή το γινόμενο των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος είναι 3, να επιστρέφει ως τιμή τον μέσο όρο των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος δεν είναι ούτε 1, ούτε 2, ούτε 3, να εμφανίζει στην οθόνη το μήνυμα λάθους «Αντικανονική κλήση συνάρτησης» και να σταματάει το πρόγραμμα με κωδικό εξόδου 1.

Να επιλέξετε εσείς τον τύπο της συνάρτησης και τον τύπο των παραμέτρων της. ★★

9.3 Να γραφεί μια συνάρτηση με όνομα `total()` που να δέχεται ως όρισμα έναν αριθμό και να επιστρέφει ως τιμή το άθροισμα των αριθμών από το 1 μέχρι την τιμή της παραμέτρου. Για παράδειγμα, η `total(1250)` πρέπει να επιστρέφει ως τιμή το άθροισμα των αριθμών από το 1 μέχρι το 1250. ★★

- 9.16** Δίνεται μια συνάρτηση με το ακόλουθο πρωτότυπο:

```
float display_random(int ar);
```

Η συνάρτηση εμφανίζει στην οθόνη τόσους ακέραιους τυχαίους αριθμούς όση είναι η τιμή της παραμέτρου της `ar`. Η συνάρτηση επιστρέφει ως τιμή τον μέσο όρο των αριθμών που εμφάνισε. Να γραφεί ένα πρόγραμμα το οποίο να καλεί την παραπάνω συνάρτηση για να εμφανίσει 100 τυχαίους αριθμούς. Επίσης, το πρόγραμμα πρέπει να εμφανίζει τον μέσο όρο αυτών των αριθμών. Προσοχή, δεν σας ζητείται να γράψετε τη συνάρτηση. ★★

- 9.17** Στα μαθηματικά, αριθμοί *bouncy* λέγονται οι ακέραιοι θετικοί αριθμοί των οποίων τα ψηφία δεν είναι ούτε σε αύξουσα ούτε σε φθίνουσα σειρά. Για παράδειγμα οι αριθμοί 12445667 και 77654330 δεν είναι *bouncy*, ενώ οι 122574 και 447821 είναι. Δεν υπάρχει κανένας αριθμός *bouncy* μικρότερος από το 100. Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει το πλήθος των αριθμών *bouncy* που βρίσκονται μεταξύ του 100 και του 9999. Το αν ένας αριθμός είναι *bouncy* ή όχι θα το ελέγχει μια συνάρτηση στην οποία θα μεταβιβάζεται ο αριθμός. Θα σας βοηθήσει το παράδειγμα [8_ayxoysa.c](#). ★★★

- 9.18** Τι κάνει η παρακάτω συνάρτηση; Τι θα επιστρέψει αν κληθεί με όρισμα τον αριθμό 12345 και τι θα επιστρέψει αν κληθεί με τον αριθμό 111; Να γραφεί επίσης ένα ολοκληρωμένο πρόγραμμα που να δείχνει τη χρήση της συνάρτησης. ★★

```
int rev(int num)
{
    int digit, sum=0;
    do
    {
        digit=num%10;
        sum=(sum *10) + digit;
        num=num/10;
    } while (num!=0);
    return sum;
}
```

- 9.19** Παλίνδρομος είναι ένας ακέραιος θετικός αριθμός ο οποίος αν διαβαστεί αντίστροφα προκύπτει πάλι ο ίδιος αριθμός. Για παράδειγμα, παλίνδρομοι είναι οι αριθμοί 121, 12321, 333, κ.λπ. Επίσης, παλίνδρομοι είναι όλοι οι μονοψήφιοι αριθμοί. Να γραφεί μια συνάρτηση η οποία να ελέγχει αν ένας αριθμός είναι παλίνδρομος. Ο αριθμός θα πρέπει να μεταβιβάζεται στη συνάρτηση. Να γραφεί επίσης ένα ολοκληρωμένο πρόγραμμα που να δείχνει τη χρήση της συνάρτησης. ★★★

Κεφάλαιο

10

Εμβέλεια μεταβλητών

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Η «τμηματοποίηση» ενός προγράμματος σε συναρτήσεις εγείρει ένα ερώτημα: Τι είναι γνωστό και πού; Οι μεταβλητές που δηλώνονται μέσα ή έξω από μια συνάρτηση είναι προσπελάσιμες από τον υπόλοιπο κώδικα του προγράμματος;

Το κεφάλαιο αυτό αναλύει την εμβέλεια των μεταβλητών, δηλαδή τον χώρο μέσα στον οποίο είναι γνωστή μια μεταβλητή ανάλογα με τον τρόπο και το σημείο δήλωσής της.

Εξετάζει τι συμβαίνει όταν έχουμε επικάλυψη εμβελειών, και προσδιορίζει τον χρόνο ζωής των μεταβλητών, δηλαδή πότε χάνουν τα περιεχόμενά τους

Εισαγωγή

Σε ένα πρόγραμμα της C μπορούν να δηλωθούν μεταβλητές σε διαφορετικά σημεία του προγράμματος. Η θέση και το είδος της δήλωσης μιας μεταβλητής καθορίζει την εμβέλειά της και τον χρόνο ζωής της.

Εμβέλεια μιας μεταβλητής (variable scope) είναι ο χώρος του προγράμματος μέσα στον οποίο είναι γνωστή η μεταβλητή. Ο χρόνος ζωής μιας μεταβλητής καθορίζει πότε μια μεταβλητή καταστρέφεται και χάνει τα περιεχόμενά της.

Ανάλογα με τη θέση και τον τρόπο δήλωσής τους, οι μεταβλητές χωρίζονται σε **τοπικές** (local), **καθολικές** (global), και **στατικές** (static).

Τοπικές μεταβλητές

Όπως έχουμε δει μέχρι τώρα, οι μεταβλητές μιας συνάρτησης δηλώνονται σε οποιοδήποτε σημείο της, αρκεί να έχουν δηλωθεί πριν να χρησιμοποιηθούν. Στην περίπτωση που η συνάρτηση έχει τυπικές παραμέτρους, αυτές δηλώνονται μέσα στις παρενθέσεις. Οι τυπικές παράμετροι είναι και αυτές τοπικές μεταβλητές της συνάρτησης.

Οι μεταβλητές μιας συνάρτησης, καθώς και οι τυπικές της παράμετροι (αν υπάρχουν), καλούνται **τοπικές μεταβλητές** (local variables) και έχουν ισχύ μόνο μέσα στη συνάρτηση που δηλώθηκαν, παραμένοντας άγνωστες στο υπόλοιπο πρόγραμμα. Οι τοπικές μεταβλητές καταστρέφονται μόλις λήξει η εμβλείά τους (χρόνος ζωής).

Στο παράδειγμα που ακολουθεί γίνεται κατανοητή η χρήση των τυπικών παραμέτρων και των τοπικών μεταβλητών.

```
#include <stdio.h>
int add(int x, int y);

int main(void)
{
    int a,b,sum;
    printf("Δώσε δύο αριθμούς:");
    scanf("%d %d",&a,&b);
    sum=add(a,b);
    printf("Αθροισμα=%d\n",sum);
    return 0;
}
```

Προκαταβολική δήλωση της συνάρτησης `add()`.

Τοπικές μεταβλητές της συνάρτησης `main()`.

Στην επόμενη συνάρτηση `add()`, η `ss` είναι τοπική μεταβλητή της συνάρτησης και έχει ισχύ μόνο μέσα στη συνάρτηση `add()`.

Οι τυπικές παράμετροι `x` και `y` αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης, και έχουν ισχύ μόνο μέσα στη συνάρτηση.

```
int add(int x, int y)
{
    int ss;
    ss=x+y;
    return ss;
}
```

Οι τυπικές παράμετροι παίζουν επίσης τον ρόλο τοπικών μεταβλητών.

Δήλωση της τοπικής μεταβλητής **ss**.

Η συνάρτηση **add()** επιστρέφει ως τιμή το περιεχόμενο της μεταβλητής **ss**.

Οι τυπικές παράμετροι παίζουν και αυτές τον ρόλο των τοπικών μεταβλητών, με τη διαφορά ότι σε αυτές αντιγράφονται οι τιμές των ορισμάτων με τα οποία καλείται η συνάρτηση.

Αν δεν γίνει κατανοητή η χρήση και η εμβέλεια των τυπικών παραμέτρων και των τοπικών μεταβλητών, είναι πολύ εύκολο να παρεισφρήσουν λάθη στα προγράμματά μας. Στη συνέχεια βλέπετε το προηγούμενο πρόγραμμα, γεμάτο ενδεχόμενα λάθη.

```
#include <stdio.h>
int add(int x, int y);
int main(void)
{
    int sum;
    float a,b;
    printf("Δωσε δύο αριθμούς:");
    scanf("%f %f",&a,&b);
    sum=add(a,b);
    printf("ss=%d\n",ss);
    printf("Αθροισμα=%d\n",sum);
    return 0;
}
```

Τα ορίσματα με τα οποία πρέπει να κληθεί η **add()** πρέπει να είναι τύπου **int**, και όχι **float**. Αυτό μπορεί να έχει συνέπεια την απώλεια πληροφοριών.

Η συνάρτηση **main()** αγνοεί την ύπαρξη της **ss**, η οποία είναι τοπική μεταβλητή της **add()**.

```
int add(int x, int y)
{
    int ss;
    ss=a+b;
    return ss;
}
```

Η συνάρτηση **add()** αγνοεί την ύπαρξη των **a** και **b**, οι οποίες είναι τοπικές μεταβλητές της **main()**.

Στη συνέχεια βλέπετε ένα πιο ολοκληρωμένο πρόγραμμα με τέσσερις συναρτήσεις, εκτός από τη **main()**. Το πρόγραμμα ζητάει δύο ακέραιους αριθμούς και υπολογίζει το άθροισμα, το γινόμενο, και τον μέσο όρο τους.

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

10_emveleia.c

```
int add(int x, int y);
int gin(int x, int y);
float mo(int x, int y);
void display(int a, int b);
```

Προκαταβολικές δηλώσεις των συναρτήσεων

```
int main(void)
{
    int a,b;
    printf("Υπολογισμός Αθροίσματος, Γινομένου και Μέσου όρου\n");
    printf("-----\n");
    printf("Δώσε δύο αριθμούς:");
    scanf("%d %d",&a,&b);    // διαβάζει δύο ακέραιους αριθμούς
    display(a,b);            // καλεί τη συνάρτηση display() με ορίσματα τους δύο αριθμούς
    return 0;
}
// υπολογισμός αθροίσματος
int add(int x, int y)
{
    int ss;
    ss=x+y;    // στη μεταβλητή ss καταχωρίζεται το άθροισμα των παραμέτρων της συνάρτησης
    return ss; // η συνάρτηση επιστρέφει την τιμή της ss
}
// υπολογισμός γινομένου
int gin(int x, int y)
{
    int gg;
    gg=x*y;    // στη μεταβλητή gg καταχωρίζεται το γινόμενο των παραμέτρων της συνάρτησης
    return gg; // η συνάρτηση επιστρέφει την τιμή της gg
}
// υπολογισμός μέσου όρου
float mo(int x, int y)
{
    float mm;
    mm=(x+y)/2.0;    // στη μεταβλητή mm καταχωρίζεται ο μέσος όρος των παραμέτρων της συνάρτησης
    return mm;    // η συνάρτηση επιστρέφει την τιμή της mm
}
// εμφάνιση αποτελεσμάτων
void display(int a, int b)
{
    printf("Αθροισμα=%d\n",add(a,b));
    printf("Γινόμενο=%d\n",gin(a,b));
    printf("Μέσος όρος=%f\n",mo(a,b));
}
```

Οι τυπικές παράμετροι **a** και **b** αποτελούν επί-
σης τοπικές μεταβλητές της συνάρτησης
display(). Δεν έχουν καμία σχέση με τις μετα-
βλητές **a** και **b** που δηλώνονται στη συνάρτηση
main().

Οι συναρτήσεις **add()** και **gin()** επιστρέφουν τιμές τύπου **int**. Η συνάρτηση **mo()** επιστρέφει τιμή **float** και η συνάρτηση **display()** δεν επιστρέφει καμία τιμή. Παρατηρούμε, επίσης, ότι και οι τέσσερις συναρτήσεις δηλώνονται δύο φορές, μία πριν από τη **main()** και μία όταν ορίζονται. Αυτό είναι απαραίτητο επειδή, σε αντίθετη περίπτωση, όταν θα τις συναντούσε ο μεταγλωττιστής στη **main()**, δεν θα γνώριζε ούτε τον τύπο τους, ούτε τον τύπο των παραμέτρων τους. Αυτή η δήλωση, όπως έχουμε ήδη αναφέρει, ονομάζεται **προκαταβολική δήλωση** (forward declaration).

Παρά το γεγονός ότι μπορούμε να δηλώσουμε μια τοπική μεταβλητή σε οποιοδήποτε σημείο μιας συνάρτησης, μια καλή πρακτική είναι να τις δηλώνουμε όλες μαζί σε προτάσεις αμέσως μετά από το αριστερό άγκιστρο (`{`) της συνάρτησης. Με αυτή την τακτική μπορούμε με μια γρήγορη ματιά να διαπιστώσουμε ποιες τοπικές μεταβλητές δηλώνονται στη συνάρτηση, αντί να ψάχνουμε να τις εντοπίσουμε σε διαφορετικά σημεία του κώδικά της.

Δήλωση τοπικών μεταβλητών σε σύνθετη πρόταση

Όπως ακριβώς δηλώνεται μια τοπική μεταβλητή μέσα σε μια συνάρτηση, μπορεί να δηλωθεί και μέσα σε μια σύνθετη πρόταση (compound statement), συνήθως αμέσως μετά το αριστερό της άγκιστρο.

```
int main(void)
```

```
{
    int a=5,b=5;
    if (a==b)
    {
        float mo;
        mo=(a+b)/2.0;
        printf("mo=%f",mo);
    }
    return 0;
}
```

Χώρος εμβέλειας των μεταβλητών `a` και `b`

Χώρος εμβέλειας της μεταβλητής `mo`

Μια τοπική μεταβλητή η οποία δηλώνεται μέσα σε μια σύνθετη πρόταση έχει εμβέλεια μόνο μέσα στη σύνθετη πρόταση στην οποία δηλώθηκε.

Δήλωση της μεταβλητής ενός βρόχου `for` ως τοπικής

Από το πρότυπο C99 και μετά, η C επιτρέπει τον ορισμό και την ανάθεση αρχικής τιμής μεταβλητής μέσα σε έναν βρόχο `for` ως ακολούθως:

```
int main(void)
```

```
{
    for (int i=1;i<=10;i++)
    {
        printf("Τοπικές μεταβλητές\n");
        printf("i=%d\n",i);
    }
    // Η μεταβλητή i είναι άγνωστη σε αυτό το σημείο
    return 0;
}
```

Ορισμός της μεταβλητής `i`.

Χώρος εμβέλειας της μεταβλητής `i`.

Με αυτόν τον τρόπο η μεταβλητή `i` γίνεται τοπική μεταβλητή του βρόχου `for` και μπορεί να χρησιμοποιηθεί μόνο μέσα σε αυτόν.

Καθολικές μεταβλητές

Στη C μπορούμε να δηλώσουμε μια μεταβλητή που να μην ανήκει σε καμία συνάρτηση. Μια τέτοια μεταβλητή δηλώνεται έξω από οποιαδήποτε συνάρτηση και καλείται **καθολική μεταβλητή** (global variable).

Οι καθολικές μεταβλητές έχουν εμβέλεια σε όλο το πρόγραμμα, και μπορούν να χρησιμοποιηθούν από οποιαδήποτε συνάρτηση του προγράμματος. Οι καθολικές μεταβλητές καταστρέφονται όταν ολοκληρωθεί η εκτέλεση του προγράμματος (χρόνος ζωής).

Στο επόμενο πρόγραμμα, οι μεταβλητές **a** και **b** δηλώνονται ως καθολικές μεταβλητές τις οποίες μπορούν να προσπελάζουν όλες οι συναρτήσεις.

```

int a,b;
int main(void)
{
    a=8;
    b=10;
    printf("Άθροισμα=%d\n", add(a,b));
    return 0;
}
int add(int x, int y)
{
    int ss;
    ss=x+y;
    printf("a=%d b=%d\n", a,b);
    return ss;
}
  
```

Δήλωση των καθολικών μεταβλητών **a** και **b**.

Χρήση των **a** και **b** μέσα στη **main()**.

Μεταβίβαση τιμών στις παραμέτρους **x** και **y**.

Χρήση των **a** και **b** μέσα στην **add()**.

Η χρήση καθολικών μεταβλητών παρέχει έναν εναλλακτικό τρόπο μεταβίβασης πληροφοριών μέσα σε μια συνάρτηση.

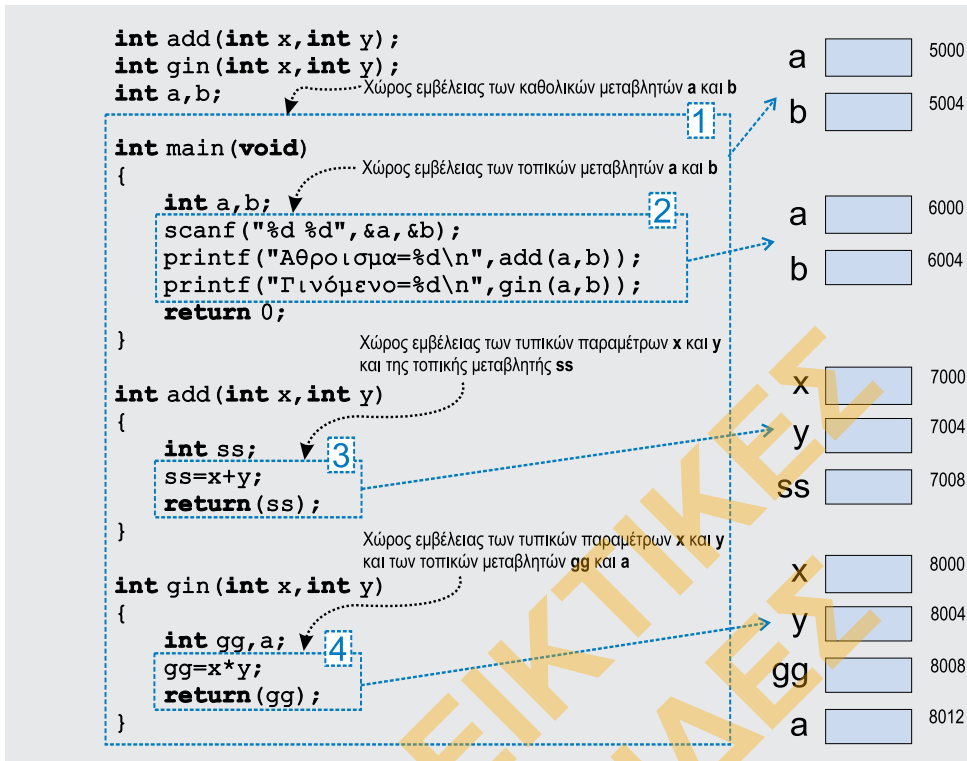
```

int add()
{
    int ss;
    ss=a+b;
    return ss;
}
  
```

Παρατηρούμε ότι με αυτόν τον τρόπο δεν υπάρχει ανάγκη μεταβίβασης των τιμών των **a** και **b** στη συνάρτηση **add()** μέσω παραμέτρων, αφού η συνάρτηση μπορεί να προσπελάζει απευθείας τις μεταβλητές **a** και **b**.

☞ Σε μια καθολική μεταβλητή έχουν πρόσβαση όλες οι συναρτήσεις που βρίσκονται μετά από το σημείο του προγράμματος όπου δηλώθηκε. Όταν λοιπόν θέλουμε όλες ανεξαιρέτως οι συναρτήσεις να έχουν πρόσβαση σε μια καθολική μεταβλητή, πρέπει να δηλώσουμε τη μεταβλητή αυτή στην αρχή του προγράμματος, πριν από τη **main()**.

☞ Στην περίπτωση που έχει δηλωθεί μια τοπική μεταβλητή με όνομα ίδιο με μια καθολική μεταβλητή, τότε στον χώρο εμβέλειας της τοπικής μεταβλητής αναφερόμαστε στην τοπική μεταβλητή, ενώ οπουδήποτε αλλού αναφερόμαστε στην καθολική μεταβλητή.



Σχήμα 10.1 Εμβέλεια τοπικών και καθολικών μεταβλητών

Η μέθοδος μεταβίβασης τιμών σε μια συνάρτηση μέσω καθολικών μεταβλητών είναι μια πολύ κακή προγραμματιστική τεχνική και πρέπει οπωσδήποτε να αποφεύγεται.

Το παράδειγμα του Σχήματος 10.1 δείχνει την εμβέλεια τοπικών και καθολικών μεταβλητών. Παρατηρούμε ότι, κάθε φορά που δηλώνεται μια τοπική μεταβλητή (ή μια τυπική παράμετρος), δεσμεύεται διαφορετική θέση μνήμης, παρόλο που μπορεί να υπάρχει άλλη τοπική ή καθολική μεταβλητή με το ίδιο όνομα. Οι μεταβλητές `a` και `b` της συνάρτησης `main()` του παραδείγματος χρησιμοποιούν εντελώς διαφορετικές θέσεις μνήμης από τις καθολικές μεταβλητές `a` και `b`.

Όταν γίνεται αναφορά σε μια μεταβλητή, χρησιμοποιείται η μεταβλητή που έχει τον πλησιέστερο χώρο εμβέλειας.

Για παράδειγμα, όταν μέσα στον χώρο εμβέλειας 4 γίνει αναφορά στη μεταβλητή `a`, ουσιαστικά γίνεται αναφορά στη θέση με διεύθυνση 8012, και όχι στη θέση της καθολικής μεταβλητής `a` (με διεύθυνση 5000). Η αναφορά στη μεταβλητή `b` (μέσα στον χώρο 4)

Κεφάλαιο

11

Δείκτες

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Οι δείκτες είναι ένας ιδιαίτερος και έμμεσος τρόπος διαχείρισης της μνήμης. Μπορούμε να προσπελάζουμε μια θέση μνήμης χρησιμοποιώντας τη διεύθυνσή της και όχι το όνομά της.

Το κεφάλαιο αυτό περιγράφει πλήρως τη χρήση δεικτών στη γλώσσα C και την ιδιαίτερη αριθμητική τους. Επίσης γίνεται αναφορά στη σχέση δεικτών και συμβολοσειρών και παρουσιάζεται ο τρόπος χειρισμού μιας συμβολοσειράς μέσω ενός δείκτη.

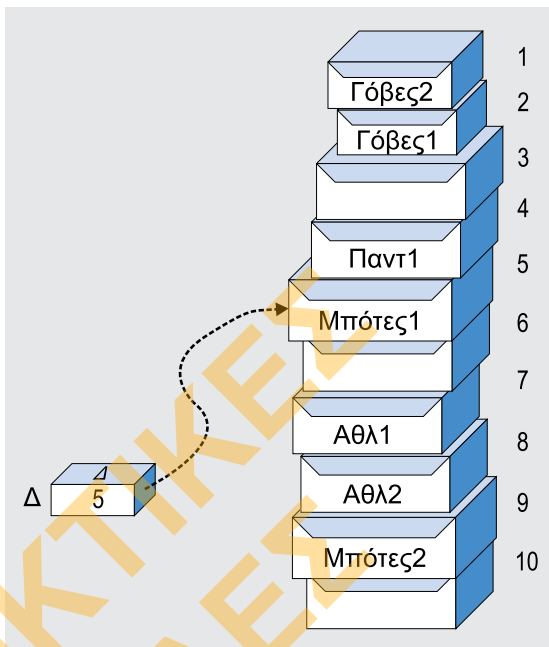
Επειδή η κατανόηση και η χρήση διάφορων λειτουργιών της C βασίζεται στους δείκτες, σας συμβουλεύω να μελετήσετε με πολύ προσοχή το παρόν κεφάλαιο.

Εισαγωγή

Οι δείκτες (pointers) είναι ένα από τα πιο παρεξηγημένα ζητήματα των γλωσσών προγραμματισμού. Οι αρχάριοι στον προγραμματισμό δυσκολεύονται αρκετά στην κατανόηση και τη χρήση τους, ενώ οι έμπειροι σπαζοκεφαλιάζουν με τις ιδιοτροπίες τους. Η βασική φιλοσοφία των δεικτών, όμως, είναι πολύ απλή.

Ας φανταστούμε ότι έχουμε κουτιά διαφόρων μεγεθών, το ένα πάνω από το άλλο, τα οποία περιέχουν παπούτσια. Κάποια κουτιά είναι άδεια και κάποια γεμάτα. Σε κάθε γεμάτο κουτί αναγράφεται ένα ενδεικτικό όνομα ώστε να καταλαβαίνουμε τι περιέχει. Κάθε κουτί έχει έναν αύξοντα αριθμό (1, 2, 3, ...) ο οποίος προσδιορίζει τη σειρά του στη στοίβα.

Σε ένα ξεχωριστό κουτί (Δ) βάζουμε τον αύξοντα αριθμό του κουτιού με τα παπούτσια που θα φορέσουμε το βράδυ, π.χ. το 5 αν θέλουμε να βάλουμε το ζευγάρι που βρίσκεται στο κουτί **Μπότες1** (με α/α 5).



Το κουτί Δ δείχνει στο πραγματικό κουτί που θα χρησιμοποιήσουμε (το κουτί με α/α 5). Μπορούμε να πούμε ότι το κουτί Δ είναι ένας **δείκτης** ο οποίος δείχνει στο κουτί **Μπότες1**.

Θα μπορούσαμε φυσικά να θυμόμαστε το κουτί που θα χρησιμοποιούσαμε από το όνομά του (**Μπότες1**). Η χρήση του δείκτη Δ, όμως, είναι ένας έμμεσος τρόπος πρόσβασης στο κουτί **Μπότες1**. Έτσι, λοιπόν, για να φορέσουμε τα παπούτσια μας το βράδυ έχουμε δύο επιλογές:

- Να ανοίξουμε το κουτί **Μπότες1** (αν το θυμόμαστε) – ο άμεσος τρόπος.
- Να ανοίξουμε το κουτί Δ, να οδηγηθούμε στο κουτί που μας ενδιαφέρει, δηλαδή το κουτί με α/α 5, και να το ανοίξουμε για να πάρουμε τα περιεχόμενά του. Με αυτόν τον τρόπο δεν χρειάζεται πλέον να γνωρίζουμε το όνομα του κουτιού.

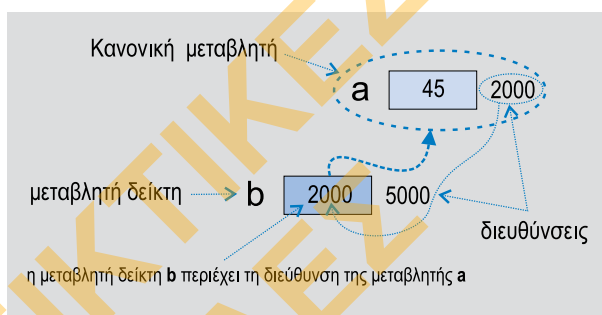
Στο συγκεκριμένο παράδειγμα, το κουτί-δείκτης είναι ένα κουτί το οποίο περιέχει τον α/α ενός άλλου κουτιού.

Το μέγεθος του κουτιού-δείκτη **είναι ανεξάρτητο** από το μέγεθος του κουτιού στο οποίο δείχνει, αφού σε κάθε περίπτωση θα περιέχει μόνο τον α/α του.

Το προηγούμενο παράδειγμα δείχνει με απλό και κατανοητό τρόπο τη φιλοσοφία των δεικτών. Στη συνέχεια του κεφαλαίου θα εξετάσουμε σε βάθος τη χρήση των δεικτών στη C. Η παρουσίαση των δεικτών θα ολοκληρωθεί στα επόμενα κεφάλαια, αφού γίνει πρώτα κατανοητή η χρήση των πινάκων, αλλά και των προσαρμοσμένων τύπων δεδομένων που υποστηρίζει η C.

Μεταβλητές δείκτη (Pointer variables)

Οι μεταβλητές δείκτη έχουν ένα ιδιαίτερο χαρακτηριστικό. Μέσα σε μια μεταβλητή δείκτη καταχωρίζεται η διεύθυνση μιας άλλης μεταβλητής. Η φιλοσοφία μιας μεταβλητής δείκτη φαίνεται στο διπλανό σχήμα. Στη μεταβλητή δείκτη **b** έχει καταχωριστεί η διεύθυνση της μεταβλητής **a** ($b = \&a$).



Σε αυτή την περίπτωση λέμε ότι η μεταβλητή δείκτη **b** «δείχνει» στη μεταβλητή **a**!

Δήλωση μιας μεταβλητής δείκτη

Μια μεταβλητή δείκτη δηλώνεται όπως οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο τελεστής αναφοράς *****. Για παράδειγμα, η πρόταση:

```
int a, *b;
```

δηλώνει μια μεταβλητή **a** τύπου **int** και μια **μεταβλητή δείκτη b** τύπου **int**.

Το γεγονός ότι η συγκεκριμένη μεταβλητή δείκτη δηλώθηκε ως τύπου **int** σημαίνει ότι η μεταβλητή **b** θα περιέχει τη διεύθυνση μιας μεταβλητής τύπου **int**. Όπως θα δούμε αργότερα, ο τύπος της θέσης μνήμης στην οποία δείχνει μια μεταβλητή δείκτη παίζει καθοριστικό ρόλο στην αριθμητική δεικτών.

Ο τελεστής αναφοράς ***** είναι ο ίδιος με τον τελεστή πολλαπλασιασμού. Η C, ανάλογα με τον τρόπο και την πρόταση εφαρμογής του, τον ερμηνεύει κατάλληλα.

Προς το παρόν, αρκεί να έχουμε υπόψη ότι μια **μεταβλητή δείκτη** περιέχει τη **διεύθυνση** μιας άλλης μεταβλητής.

```
printf(b);
return 0;
}
```

Δείκτες

Η `printf()` δέχεται ως όρισμα μια συμβολοσειρά για το αλφαριθμητικό μορφοποίησης, δηλαδή τη διεύθυνση της πρώτης θέσης μνήμης που κατέχει. Επομένως, η πρόταση `printf(b)` έχει ως αποτέλεσμα να εμφανιστεί στην οθόνη το σύνολο χαρακτήρων στο οποίο δείχνει ο δείκτης `b`.

Ένας διαφορετικός τρόπος δήλωσης μιας μεταβλητής δείκτη

Μπορεί να συναντήσουμε, ή ακόμα και να χρησιμοποιήσουμε, έναν κάπως διαφορετικό τρόπο δήλωσης μιας μεταβλητής δείκτη. Γνωρίζουμε πλέον ότι στην παρακάτω πρόταση

```
int *b;
```

δηλώνεται μια μεταβλητή δείκτη `b`.

Θα μπορούσαμε να τοποθετήσουμε τον τελεστή αναφοράς `*` στο τέλος του ονόματος του τύπου, και όχι στην αρχή του ονόματος της μεταβλητής.

Για παράδειγμα, η πρόταση `int* p1`; δηλώνει μια μεταβλητή δείκτη με όνομα `p1`. Προσοχή, όμως, διότι η παρακάτω πρόταση

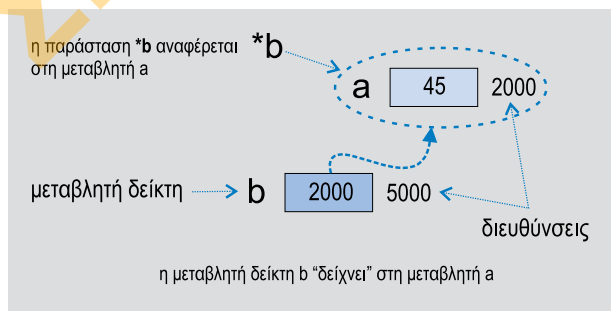
```
int* p1, p2;
```

δηλώνει μια μεταβλητή δείκτη σε `int` με όνομα `p1` και μια κανονική μεταβλητή τύπου `int` με όνομα `p2`, δηλαδή είναι ισοδύναμη με την πρόταση `int *p1, p2`!

Οι τελεστές & και *

Μέχρι στιγμής έχουμε γνωρίσει τον τελεστή `&`, ο οποίος αποδίδει τη διεύθυνση μιας μεταβλητής (θέσης μνήμης). Γνωρίζουμε ότι μπορούμε να καταχωρίσουμε τη διεύθυνση μιας μεταβλητής σε μια μεταβλητή δείκτη. Ο τελεστής **αναφοράς** `*`, πέρα από τη χρήση του για τη δήλωση μιας μεταβλητής δείκτη, μας δίνει τη δυνατότητα να προσπελάζουμε τη θέση μνήμης στην οποία δείχνει ένας δείκτης. Στο παράδειγμα του σχήματος, η μεταβλητή δείκτη `b` περιέχει τη διεύθυνση της μεταβλητής `a` (π.χ. με μια πρόταση `b=&a`). Χρησιμοποιώντας τον τελεστή αναφοράς `*`, μπορούμε να προσπελάσουμε τη μεταβλητή `a` μέσω του δείκτη `b`. Για παράδειγμα, η πρόταση:

```
*b=100;
```



Σύνοψη

- Οι δείκτες είναι μεταβλητές που περιέχουν διευθύνσεις μνήμης, συνήθως διευθύνσεις άλλων μεταβλητών.
- Το μέγεθος μιας μεταβλητής δείκτη είναι συγκεκριμένο και ανεξάρτητο από τον τύπο με τον οποίο δηλώθηκε.
- Ο τελεστής & αποδίδει τη διεύθυνση μιας θέσης μνήμης.
- Ο τελεστής αναφοράς * χρησιμοποιείται σε συνδυασμό με μια μεταβλητή δείκτη για την αναφορά στη μεταβλητή **στην οποία δείχνει** ο δείκτης.

Αριθμητική δεικτών

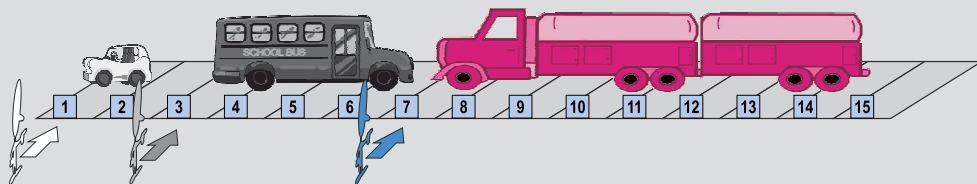
Οι δείκτες ακολουθούν τη δική τους λογική στις αριθμητικές πράξεις, κάτι που δεν είναι όμως καθόλου παράδοξο αν αναλογιστούμε τη βαθύτερη φιλοσοφία τους.

Θα ξεκινήσω με ένα παράδειγμα της καθημερινότητας. Ρίξτε μια ματιά στο Σχήμα 11.3α της επόμενης σελίδας. Στο σχήμα απεικονίζονται 15 θέσεις πάρκινγκ. Στις θέσεις αυτές μπορούν να παρκάρουν τριών ειδών αυτοκίνητα. **Μικρά** αυτοκίνητα IX (τα λευκά) τα οποία καταλαμβάνουν **μία** μόνο θέση, **λεωφορεία** (τα γκρι) τα οποία καταλαμβάνουν **τέσσερις** θέσεις και **νταλίκες** (τα μπλε) οι οποίες καταλαμβάνουν **οκτώ** θέσεις η κάθε μία. Ήδη βλέπουμε παρκαρισμένα **τρία οχήματα**: ένα IX στη θέση 1, ένα λεωφορείο στις θέσεις 3~6 και μια νταλίκα στις θέσεις 7~14.

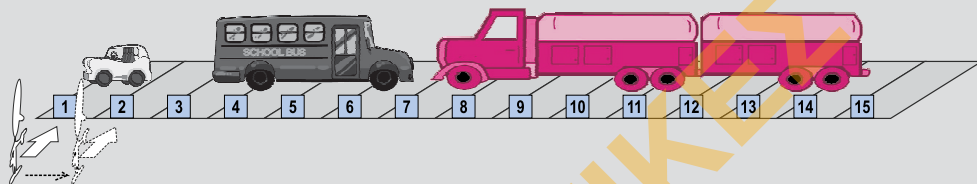
Υποθέτουμε επίσης ότι υπάρχουν **τρεις «δείκτες»**, ένας λευκός, ένας γκρι, και ένας μπλε. Η δουλειά του λευκού «δείκτη» είναι να δείχνει σε μικρά οχήματα (λευκά) τα οποία καταλαμβάνουν μία θέση, η δουλειά του γκρι είναι να δείχνει σε λεωφορεία (γκρι) τα οποία καταλαμβάνουν τέσσερις θέσεις, και η δουλειά του μπλε δείκτη είναι να δείχνει σε νταλίκες (μπλε) οι οποίες καταλαμβάνουν οκτώ θέσεις. Κάθε ένας από αυτούς τους δείκτες δείχνει στην πρώτη από τις θέσεις που καταλαμβάνει καθένα από τα τρία παρκαρισμένα οχήματα.

Αν μετακινήσουμε τον λευκό δείκτη κατά μία θέση, τότε θα πρέπει να δείξει στη θέση 2 η οποία είναι η αμέσως επόμενη θέση παρκαρίσματος, διαθέσιμη ή μη (Σχήμα 11.3β). Όμως, όταν μετακινηθεί ο γκρι δείκτης κατά μία θέση, θα πρέπει να δείξει στη θέση 7 η οποία είναι η αμέσως επόμενη θέση παρκαρίσματος, ώστε να προσπεράσει τις θέσεις που καταλαμβάνει το λεωφορείο στο οποίο δείχνει (Σχήμα 11.3γ). Αντίστοιχα, η μετακίνηση του μπλε δείκτη θα έχει ως αποτέλεσμα την αύξησή του κατά 8, ώστε να προσπεράσει τις θέσεις που καταλαμβάνει η νταλίκα και επομένως να δείχνει στην επόμενη θέση, τη θέση 15 (Σχήμα 11.3δ)!

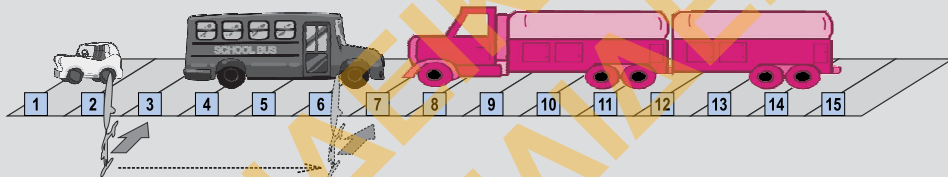
Ο χειρισμός των δεικτών στη C γίνεται όπως στο παραπάνω παράδειγμα. Ένας δείκτης στη C «δείχνει» σε μεταβλητές συγκεκριμένου τύπου (char, int, float, double, κλπ) οι οποίες, ανάλογα με τον τύπο τους, έχουν διαφορετικό μέγεθος και επομένως καταλαμβάνουν διαφορετικό πλήθος θέσεων (θέσεις παρκαρίσματος) στη μνήμη του Η/Υ.



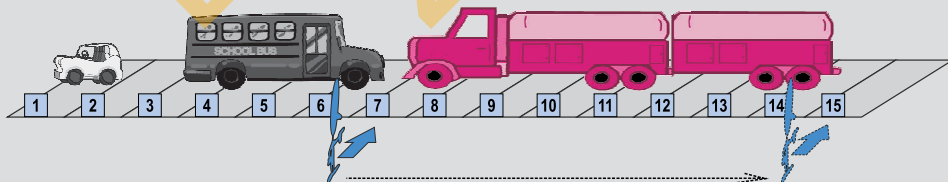
(α) Κάθε κατηγορία δείκτη (λευκός - γκρι - μπλε) δείχνει στον αντίστοιχο τύπο οχήματος



(β) Μετακίνηση στην επόμενη θέση (αύξηση κατά μια θέση) ενός λευκού δείκτη έχει ως αποτέλεσμα την αύξησή του κατά 1 ώστε να δείχνει στην επόμενη θέση παρκαρίσματος.

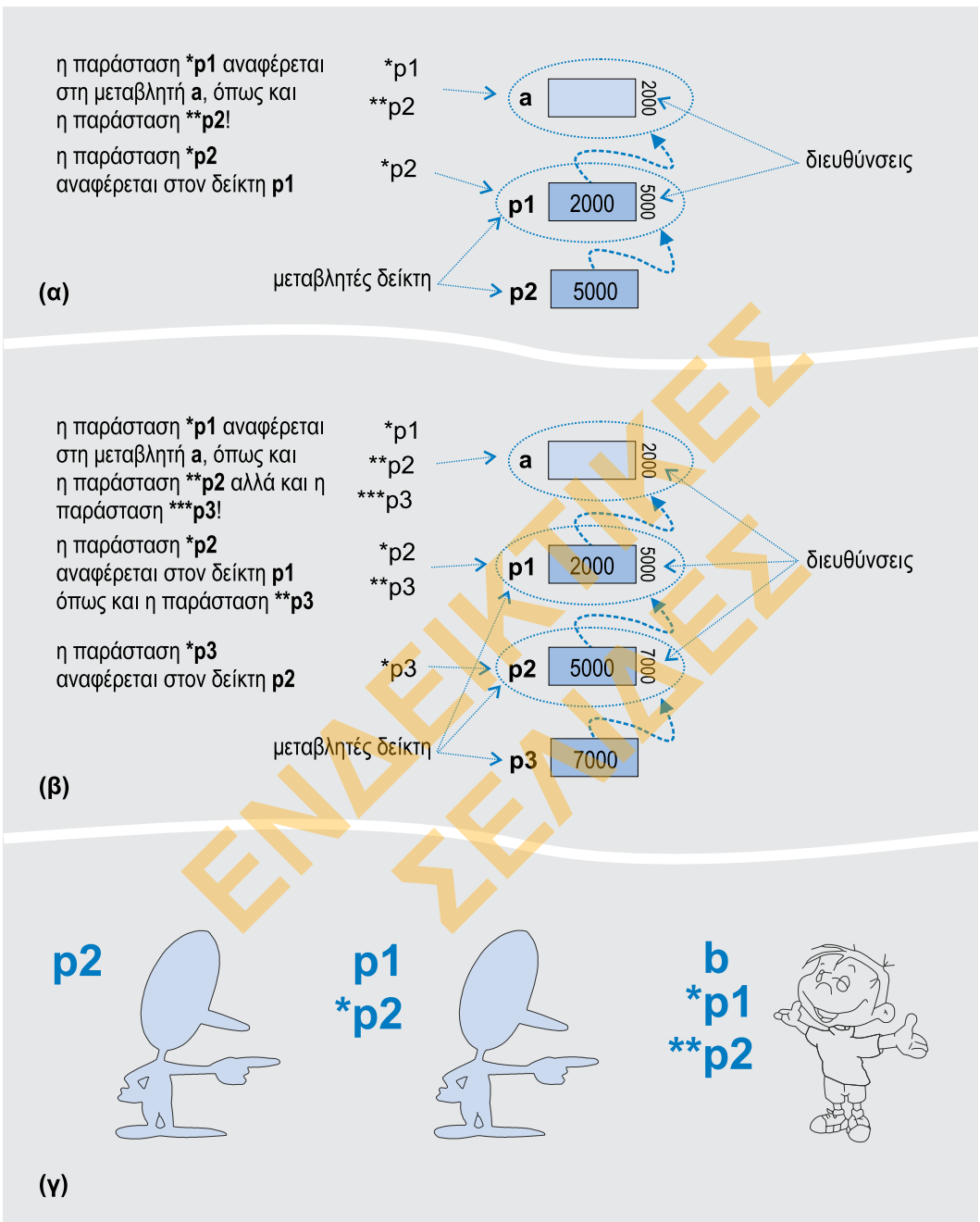


(γ) Μετακίνηση στην επόμενη θέση (αύξηση κατά μια θέση) ενός γκρι δείκτη έχει ως αποτέλεσμα την αύξησή του κατά 4 ώστε να δείχνει στην επόμενη θέση παρκαρίσματος.



(δ) Μετακίνηση στην επόμενη θέση (αύξηση κατά μια θέση) ενός μπλέ δείκτη έχει ως αποτέλεσμα την αύξησή του κατά 8 ώστε να δείχνει στην επόμενη θέση παρκαρίσματος.

Σχήμα 11.3 Η αριθμητική των δεικτών.



Σχήμα 11.6 Δείκτες σε δείκτες.

Το παράδειγμα του Σχήματος 11.6α υλοποιείται από τις παρακάτω προτάσεις:

```
int a,*p1,**p2;
p1=&a;      // Ο δείκτης p1 τώρα δείχνει στη μεταβλητή a (διότι περιέχει τη διεύθυνση της a)
p2=&p1;     // Ο δείκτης p2 τώρα δείχνει στον δείκτη p1 (διότι περιέχει τη διεύθυνση του p1)
```

Για να προσπελάσουμε τη μεταβλητή **a** μπορούμε να χρησιμοποιήσουμε είτε τον δείκτη **p1**:

```
*p1=100;    // Καταχωρίζει το 100 στη μεταβλητή που δείχνει ο p1 (στην a).
```

είτε τον δείκτη **p2**:

```
**p2=200;   // Καταχωρίζει το 200 στη μεταβλητή που δείχνει ο δείκτης, στον οποίο δείχνει ο δείκτης p2!
```

Η χρήση του τελεστή αναφοράς * δύο φορές (**) έχει ως αποτέλεσμα να αναφερόμαστε στη θέση στην οποία δείχνει ένας δείκτης, στον οποίο όμως αναφερόμαστε μέσω ενός άλλου δείκτη.

Το ακόλουθο πρόγραμμα δείχνει τη χρήση των δεικτών σε δείκτες.

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

```
int main(void)
```

```
{
```

```
    int a=5,b=10,*p1,**p2;
```

```
    p1=&a;
```

```
    *p1=100;
```

```
    printf("a=%d b=%d\n",a,b);
```

```
    p2=&p1;
```

```
    **p2=200;
```

```
    printf("a=%d b=%d\n",a,b);
```

```
    *p2=&b;
```

```
    **p2=300;
```

```
    printf("a=%d b=%d\n",a,b);
```

```
    p1=&a;
```

```
    **p2=400;
```

```
    printf("a=%d b=%d\n",a,b);
```

```
    return 0;
```

```
}
```

11_pointer_to_pointer1.c

Δηλώνεται ένας δείκτης σε **int** (p1) και ένας δείκτης σε δείκτη **int** (p2)!

Στη μεταβλητή **a** καταχωρίζεται το 100

Στη μεταβλητή **a** καταχωρίζεται το 200

Στη μεταβλητή **b** καταχωρίζεται το 300

Στη μεταβλητή **a** καταχωρίζεται το 400

a=100 b=10
a=200 b=10
a=200 b=300
a=400 b=300

Το μοτίβο αυτό μπορεί να συνεχιστεί σε όσα επίπεδα θέλουμε: Δηλαδή, μπορούμε να έχουμε δείκτες σε δείκτες οι οποίοι δείχνουν σε δείκτες κ.ο.κ! Στο πρόγραμμα που ακολουθεί έχουμε τρία επίπεδα δεικτών. Ο δείκτης **p1** δείχνει στην μεταβλητή **a**, ο **p2** δείχνει στον δείκτη **p2** και ο **p3** στον δείκτη **p2** (Σχήμα 11.6β)!

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

```
int main(void)
```

```
{
```

```
    int a=5,*p1,**p2,***p3;
```

```
    printf("a=%d\n",a);
```

```
    p1=&a;
```

11_pointer_to_pointer2.c

Δηλώνεται ένας δείκτης σε **int** (p1), ένας δείκτης σε δείκτη **int** (p2) και ένας δείκτης σε δείκτη που δείχνει σε δείκτη **int** (p3)!

Παραδείγματα

Π11.1 Το επόμενο πρόγραμμα μετράει τους χαρακτήρες 'A' που βρίσκονται μέσα στη συμβολοσειρά «ANANΑΣ».

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
```

11_p11_1a.c

```
{
    char *ptr, ch;
    int c=0;
    ptr="ANANΑΣ";
    while (*ptr!='\0')
    {
        if (*ptr=='A') c++;
        ptr++;
    }
    // Εμφανίζει το πλήθος των χαρακτήρων 'A'
    printf("Η λέξη έχει %d A\n", c);
    return 0;
}
```

Αρχικά ο δείκτης **ptr** δείχνει στην αρχή της συμβολοσειράς.

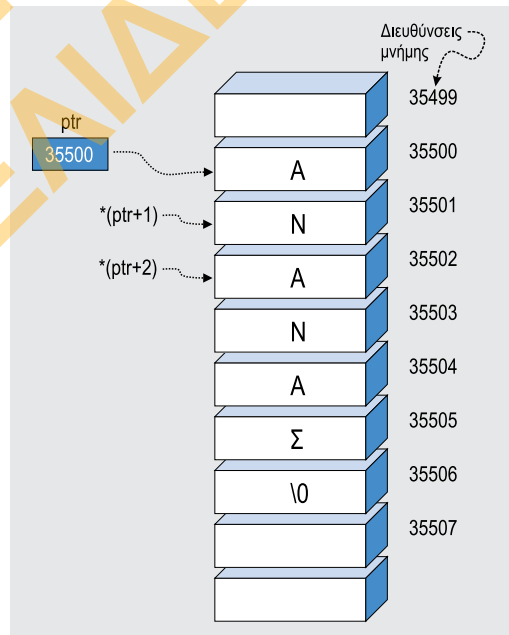
Όταν ο δείκτης **ptr** δείξει στον τερματικό χαρακτήρα '\0' η επαναληπτική διαδικασία διακόπτεται.

Ο δείκτης **ptr** δείχνει στον επόμενο χαρακτήρα της συμβολοσειράς.

Αρχικά καταχωρίζεται στη μεταβλητή **ptr** η διεύθυνση της συμβολοσειράς «ANANΑΣ» (στο παράδειγμα, η 35500). Η παράσταση ***ptr** αναφέρεται στη θέση μνήμης όπου δείχνει κάθε φορά ο δείκτης **ptr**. Η παράσταση **ptr++** αυξάνει τον δείκτη ώστε να δείχνει στην επόμενη θέση. Αρχικά ο δείκτης **ptr** δείχνει στον πρώτο χαρακτήρα, το 'A', μετά στο 'N', μετά στο δεύτερο 'A' κ.ο.κ.

Η διαδικασία θα σταματήσει όταν ο δείκτης **ptr** δείξει στο '\0', το οποίο σηματοδοτεί το τέλος του συνόλου χαρακτήρων. Η μεταβλητή **c** χρησιμοποιείται για να μετράει τις θέσεις που περιέχουν τον χαρακτήρα 'A'.

Ακολουθεί μια διαφορετική έκδοση του παραπάνω προγράμματος, η οποία χρησιμοποιεί μια συνάρτηση για την καταμέτρηση των εμφανίσεων ενός χαρακτήρα μέσα σε μια συμβολοσειρά.




```
#include <stdio.h>
int count(const char *p, char ch);
int main(void)
{
    char *ptr;
    int a,b;
    ptr="ANANAS";
    a=count(ptr, 'A');
    printf("Η λέξη έχει %d A\n",a);
    b=count("ΠΑΠΑΤΡΕΧΑΣ", 'Π');
    printf("Η λέξη έχει %d Π\n",b);
    return 0;
}

int count(const char *p, char ch)
{
    int c=0;
    while (*p!='\0')
    {
        if (*p==ch) c++;
        p++;
    }
    return c;
}
```

11_p11_1b.c

Στη συνάρτηση **count()** μεταβιβάζεται το περιεχόμενο του δείκτη **ptr**.

Η λέξη έχει 3 Α
Η λέξη έχει 2 Π

Στην **count()** μεταβιβάζεται απευθείας η διεύθυνση της συμβολοσειράς.

Η συνάρτηση **count()** επιστρέφει ως τιμή το πλήθος των χαρακτήρων της συμβολοσειράς που μεταβιβάστηκε μέσω της πρώτης παραμέτρου (**p**), οι οποίοι είναι ίσοι με τον χαρακτήρα της δεύτερης παραμέτρου (**ch**).

- Η συνάρτηση **count()** διαθέτει δύο παραμέτρους: στην πρώτη μεταβιβάζεται η διεύθυνση της συμβολοσειράς, και στη δεύτερη μεταβιβάζεται ο χαρακτήρας την εμφάνιση του οποίου θέλουμε να καταμετρήσουμε.
- Την πρώτη φορά που καλείται η συνάρτηση, της μεταβιβάζεται το περιεχόμενο του δείκτη **ptr** ο οποίος περιέχει την αρχική διεύθυνση της συμβολοσειράς «ANANAS», καθώς και ο χαρακτήρας 'A'. Η συνάρτηση επιστρέφει το πλήθος των εμφανίσεων του χαρακτήρα στη συμβολοσειρά, δηλαδή τον αριθμό 3.
- Την επόμενη φορά που καλείται η συνάρτηση της μεταβιβάζονται απευθείας η διεύθυνση της συμβολοσειράς «ΠΑΠΑΤΡΕΧΑΣ» και ο χαρακτήρας 'Π'. Η συνάρτηση επιστρέφει το πλήθος των εμφανίσεων του χαρακτήρα 'Π', δηλαδή τον αριθμό 2.
- Σε κάθε περίπτωση, η συνάρτηση **count()** «διασχίζει» τη συμβολοσειρά, ξεκινώντας από τη διεύθυνση που της μεταβιβάστηκε μέσα στην παράμετρο **p**, μέχρι να εντοπίσει τον χαρακτήρα τερματισμού '\0'. Με άλλα λόγια, όσο η παράσταση ***p!='\0'** είναι αληθής, ο δείκτης **p** αυξάνεται κάθε φορά κατά 1 ώστε να δείχνει στον επόμενο χαρακτήρα.
- Το προσδιοριστικό **const** σημαίνει ότι η συνάρτηση **count()** δεν μπορεί να τροποποιήσει, μέσω του δείκτη **p**, τα περιεχόμενα της συμβολοσειράς που της μεταβιβάζεται.

Συχνά λάθη

- ❖ Το συχνότερο λάθος είναι ότι μπερδεύουμε την αναφορά σε έναν δείκτη με την αναφορά στη θέση μνήμης στην οποία δείχνει ένας δείκτης. Για παράδειγμα, αν έχουμε δηλώσει έναν δείκτη `int *p`, με αναφορά στο `p` αναφερόμαστε στον ίδιο τον δείκτη, ενώ με το `*p` αναφερόμαστε στη θέση όπου δείχνει ο δείκτης!
- ❖ Αναθέτουμε σταθερές τιμές σε έναν δείκτη. Για παράδειγμα, αν έχουμε έναν δείκτη `p`, δεν μπορούμε έτσι απλά, με μια εντολή ανάθεσης, να του αναθέσουμε μια σταθερή τιμή, π.χ. `p=100`! Η τιμή που αναθέτουμε σε έναν δείκτη πρέπει οπωσδήποτε να είναι μια διεύθυνση.
- ❖ Μπερδεύουμε την αριθμητική των δεικτών. Για παράδειγμα, αν έχουμε έναν δείκτη `p`, η παράσταση `p=p+2` μπορεί να αυξήσει τη τιμή του `p` κατά 2, κατά 8, ή κατά 16, ανάλογα με τον τύπο με τον οποίο έχει δηλωθεί ο δείκτης!
- ❖ Μπερδεύουμε τη χρήση δεικτών σε συμβολοσειρές. Για παράδειγμα, αν έχουμε έναν δείκτη `p` ο οποίος δείχνει σε μια συμβολοσειρά, για να αναφερθούμε σε ολόκληρη τη συμβολοσειρά χρησιμοποιούμε απλώς το `p`, ενώ με το `*p` αναφερόμαστε στον πρώτο χαρακτήρα της συμβολοσειράς.
- ❖ Χρησιμοποιούμε έναν δείκτη χωρίς να του έχουμε προηγουμένως καταχωρίσει κάποια διεύθυνση (ή την τιμή `NULL`). Η χρήση ενός «αδέσποτου» δείκτη έχει καταστροφικά αποτελέσματα!

Ανασκόπηση Κεφαλαίου 11

- Η μεταβλητή δείκτη είναι μια θέση μνήμης η οποία περιέχει τη διεύθυνση μιας άλλης θέσης μνήμης.
- Η μεταβλητή δείκτη δηλώνεται όπως και οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο τελεστής αναφοράς `*`. Για παράδειγμα, η πρόταση `int a,*b` δηλώνει μία ακέραια μεταβλητή `a` και μία μεταβλητή δείκτη με όνομα `b`. Παρόμοια, η πρόταση `int *p1,*p2`; δηλώνει δύο δείκτες σε `int`, τους `p1` και `p2`.
- Σε μια μεταβλητή δείκτη μπορούμε να καταχωρίσουμε μόνο τη διεύθυνση μιας θέσης μνήμης. Για παράδειγμα, η `b=&a` καταχωρίζει στη `b` τη διεύθυνση της μεταβλητής `a`.
- Όταν η μεταβλητή δείκτη `b` περιέχει τη διεύθυνση μιας άλλης μεταβλητής `a`, τότε λέμε ότι ο δείκτης `b` «δείχνει» στη μεταβλητή `a`.

- Χρησιμοποιώντας τον τελεστή αναφοράς * μπορούμε να προσπελάζουμε, μέσω μιας μεταβλητής δείκτη, τη θέση στην οποία δείχνει η μεταβλητή. Δηλαδή, μπορούμε να αναφερόμαστε στην **a** μέσω της **b**. Για παράδειγμα, η πρόταση ***b=5** καταχωρίζει το 5 στη μεταβλητή **a** (το ***b=5** είναι ισοδύναμο με το **a=5**).
- Όταν αυξάνουμε κατά 1 μια μεταβλητή δείκτη, τότε αυτή αυξάνεται κατά τόσες μονάδες όσα είναι τα byte του τύπου δεδομένων με τον οποίο δηλώθηκε. Για παράδειγμα, το **b++** αυξάνει το **b** κατά 4 επειδή ο τύπος **int** (με τον οποίο δηλώθηκε η **b**) έχει μέγεθος 4 byte. Το αντίστοιχο γίνεται και στη μείωση.
- Κάθε μονάδα που προστίθεται σε έναν δείκτη τύπου **char** αυξάνει τον δείκτη κατά 1, όπως και σε έναν δείκτη τύπου **bool**, ενώ σε έναν δείκτη τύπου **int** τον αυξάνει κατά 4, σε έναν δείκτη τύπου **float** επίσης κατά 4, και σε έναν δείκτη τύπου **double** κατά 8.
- Πέρα από τους βασικούς τύπους δεδομένων, ένας δείκτης μπορεί να δείχνει και σε άλλες οντότητες, όπως πίνακες ή προσαρμοσμένους τύπους που θα συναντήσουμε στη συνέχεια. Γενικά, η αύξηση ενός δείκτη που δείχνει σε δεδομένα τύπου **type** κατά **n**, αυξάνει τον δείκτη κατά **n*sizeof(type)**!
- Ο δείκτης **null** είναι ένας δείκτης με τιμή **NULL** η οποία ορίζεται μέσα στο **stdio.h**. Ένας δείκτης με τιμή **NULL** δεν δείχνει πουθενά.
- Η χρήση ενός δείκτη ο οποίος δεν έχει αρχικοποιηθεί οδηγεί σε καταστροφικά αποτελέσματα και συνήθως σε «κρέμασμα» του προγράμματος.
- Ένας δείκτης μπορεί να δείχνει σε έναν άλλο δείκτη. Χρησιμοποιώντας τον τελεστή αναφοράς * δύο φορές, μπορούμε να αναφερθούμε στη μεταβλητή στην οποία δείχνει ένας δείκτης μέσω ενός άλλου δείκτη που δείχνει σε αυτόν. Αυτό μπορεί να συνεχιστεί σε όσα επίπεδα θέλουμε.
- Η μεταβίβαση μιας συμβολοσειράς σε μια συνάρτηση γίνεται μέσω ενός δείκτη σε **char**. Το προσδιοριστικό **const** μπροστά από την παράμετρο-δείκτη υποδηλώνει ότι η συνάρτηση δεν μπορεί να επηρεάσει το περιεχόμενο της συμβολοσειράς.
- Μια συνάρτηση στη C μπορεί να επιστρέψει ως τιμή έναν δείκτη, δηλαδή τη διεύθυνση μιας θέσης μνήμης. Μια τέτοια συνάρτηση δηλώνεται με ένα * πριν από το όνομά της, ή μετά από τον τύπο της.
- Οι παραστάσεις που αναφέρονται σε θέσεις μνήμης μπορούν να τοποθετούνται αριστερά από τον τελεστή ανάθεσης =. Τέτοιου είδους παραστάσεις (όπως και οι απλές μεταβλητές) λέγονται παραστάσεις αριστερής τιμής (lvalues). Μια τέτοια παράσταση είναι, για παράδειγμα, η ***p**, όπου **p** είναι ένας δείκτης.

Ασκήσεις Κεφαλαίου 11

- 11.1 Τι θα περιέχουν οι μεταβλητές **a**, **b**, και **c** μετά το τέλος του επόμενου προγράμματος; ★

```
#include <stdio.h>
int main(void)
{
    int a,b,c,*m,*p;
    a=100;
    b=50;
    m=&a;
    p=&b;
    c=*p + *m;
    (*p)++;
    p=m;
    (*p)--;
    return 0;
}
```

- 11.2 Τι κάνει το επόμενο πρόγραμμα; ★

```
#include <stdio.h>
int main(void)
{
    char *p;
    int a=5,b=10;
    p="a,b=%d,%d\n";
    printf(p,a,b);
    return 0;
}
```

Τι θα έκανε αν, αντί για την πρόταση

`printf(p,a,b);` υπήρχε η πρόταση `printf(p+4,a,b);`

- 11.3 Δίνεται το επόμενο πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
    char *p;
    p="αρνάκι άσπρο και παχύ";
    .....
}
```

Να συμπληρωθεί κατάλληλα ώστε να ζητάει ένα γράμμα και μετά να μετράει πόσες φορές υπάρχει το γράμμα αυτό στο σύνολο χαρακτήρων «αρνάκι άσπρο και παχύ». ★★

11.17 Να γραφεί μια συνάρτηση τύπου **void** στην οποία να μεταβιβάζεται ένας ακέραιος αριθμός. Η συνάρτηση πρέπει να επιστρέφει το μικρότερο και το μεγαλύτερο ψηφίο του, το άθροισμα, αλλά και το πλήθος όλων των ψηφίων του αριθμού (για την τεχνική της απομόνωσης των ψηφίων μελετήστε τον κώδικα του προγράμματος 8_digits2.c και το Παράδειγμα Π8.5). Να γραφεί επίσης ένα πρόγραμμα που να καλεί τη συγκεκριμένη συνάρτηση και να δείχνει τη λειτουργία της. ★★ ★

11.18 Υπάρχει κάποιο λάθος στην ακόλουθη συνάρτηση; Αν υπάρχει, σχολιάστε το. ★★

```
int test(const int *a)
{
    *a=100;
}
```

11.19 Τι καταλαβαίνετε από τα πρωτότυπα των ακόλουθων συναρτήσεων; ★★

```
int findchar(const char *str, char ch);
void replchar(char *str, char ch);
void noswap(const int *a1, const int *a2);
void mayswap(int *a1, int *a2);
```

11.20 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★★

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

void do_it(const char *p);

int main(void)
{
    char *p;
    p="Η γλώσσα C σε βάθος";
    do_it(p);
    return 0;
}

void do_it(const char *p)
{
    int c=0,r,i;
    char *start;
    start=p;
    while(*(p++)!='\0') c++;
    for (i=1;i<=100;i++)
    {
        r=rand()%c;
        if (*(start+r)!=' ') putchar(*(start+r));
    }
    putchar('\n');
}
```

Κεφάλαιο

12

Πίνακες

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Οι πίνακες παρέχουν τη δυνατότητα αποθήκευσης μεγάλου πλήθους δεδομένων. Το κεφάλαιο αυτό περιγράφει τη χρήση πινάκων στη γλώσσα C και παρουσιάζει τον τρόπο χειρισμού πινάκων μίας και περισσότερων διαστάσεων.

Γίνεται ιδιαίτερη αναφορά στη σχέση πινάκων και δεικτών και αναλύεται πλήρως η χρήση πινάκων χαρακτήρων για την αποθήκευση συμβολοσειρών.

Αναπτύσσονται επίσης εξειδικευμένα θέματα, όπως πίνακες μεταβλητού μήκους, δείκτες σε πίνακες και πίνακες δεικτών.

Εισαγωγή

Ας υποθέσουμε ότι θέλουμε να αποθηκεύσουμε 10 αριθμούς σε ισάριθμες θέσεις μνήμης και κατόπιν να υπολογίσουμε και να εμφανίσουμε το άθροισμά τους. Σύμφωνα με όσα έχουμε μάθει μέχρι τώρα, θα έπρεπε να γράψουμε το ακόλουθο πρόγραμμα:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int a,b,c,d,e,f,k,l,m,n,sum;
    printf("Δώσε 10 αριθμούς>");
    scanf("%d %d %d %d %d %d %d %d %d %d", &a, &b, &c, &d, &e, &f, &k, &l, &m, &n);
    sum=a+b+c+d+e+f+k+l+m+n;
    printf("Άθροισμα: %d\n", sum);
    return 0;
}
```

Στις 10 θέσεις μνήμης καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Υπολογίζει και εμφανίζει το άθροισμα των 10 θέσεων μνήμης.

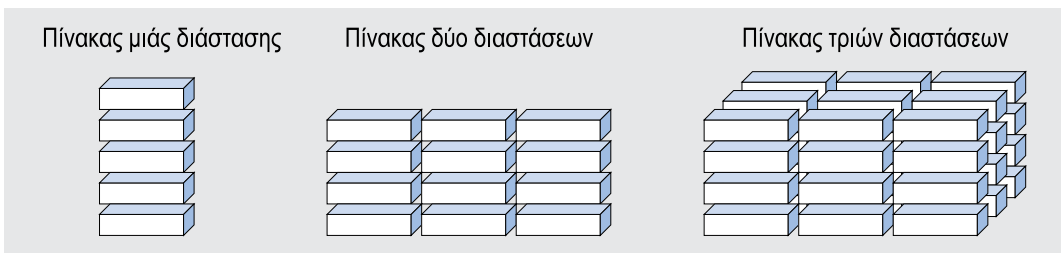
Σκεφτείτε τι θα γινόταν αν θέλαμε να κάνουμε το ίδιο όχι για 10, αλλά για 100 αριθμούς! Το πρόγραμμα θα άρχιζε να γίνεται ιδιαίτερα πολύπλοκο με 100 διαφορετικές μεταβλητές, ενώ με πολύ περισσότερους αριθμούς θα ήταν αδύνατον να υλοποιήσουμε αυτή την εργασία κατ' αυτόν τον τρόπο. Μη ξεχνάμε ότι μία από τις βασικές δυνατότητες ενός υπολογιστικού συστήματος είναι η επεξεργασία μεγάλου όγκου δεδομένων.

Μάλλον, λοιπόν, θα πρέπει να υπάρχει καλύτερος και πιο έξυπνος τρόπος!

Για την αποθήκευση μεγάλου πλήθους δεδομένων, σε οποιαδήποτε γλώσσα προγραμματισμού, χρησιμοποιούνται οι **πίνακες** (ή *συστοιχίες* – arrays).

Οι **πίνακες** αποτελούν έναν διαφορετικό τρόπο διαχείρισης της μνήμης. Ο πίνακας είναι μια ομάδα θέσεων μνήμης οι οποίες αναφέρονται με ένα κοινό όνομα. Μια συγκεκριμένη θέση μνήμης αυτής της ομάδας καθορίζεται από έναν ή περισσότερους αριθμοδείκτες.

Οι πίνακες αποτελούνται από θέσεις του ίδιου τύπου και, ανάλογα με τη διάταξή τους, μπορεί να έχουν μία ή περισσότερες διαστάσεις, όπως φαίνεται στο Σχήμα 12.1.



Σχήμα 12.1 Πίνακες μίας και περισσότερων διαστάσεων

Πίνακες μίας διάστασης (1Δ)

Μπορούμε να παρομοιάσουμε έναν πίνακα μίας διάστασης σαν μια ομάδα θέσεων μνήμης, τη μία πάνω από την άλλη. Κάθε θέση μνήμης έχει έναν αριθμό αναφοράς ο οποίος ξεκινάει από το 0 (η πρώτη 0, η δεύτερη 1, κ.ο.κ).

Στη C, η πρόταση δήλωσης ενός πίνακα είναι παρόμοια με την πρόταση δήλωσης οποιασδήποτε άλλης μεταβλητής:

τύπος_δεδομένων όνομα_πίνακα[μέγεθος];

Για παράδειγμα, η πρόταση

```
int a[10];
```

δηλώνει έναν πίνακα μίας διάστασης με όνομα **a** και 10 θέσεις μνήμης.

Ένα μεμονωμένο στοιχείο του πίνακα αναφέρεται με το όνομα του πίνακα και έναν αριθμοδείκτη μέσα σε αγκύλες ο οποίος προσδιορίζει τη θέση του στοιχείου μέσα στον πίνακα.

Στη C, η πρώτη θέση μνήμης ενός πίνακα έχει αριθμοδείκτη 0.

Επομένως, ένας πίνακας `int a[10]` θα έχει 10 θέσεις μνήμης, από την `a[0]` μέχρι την `a[9]`.

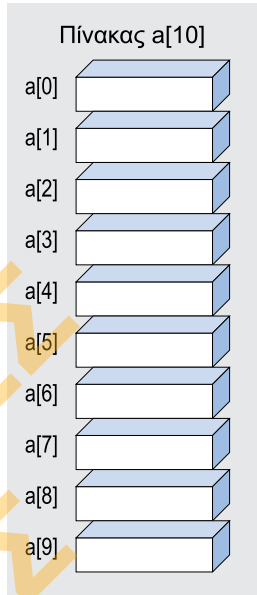
Όλα τα στοιχεία ενός πίνακα έχουν τον ίδιο τύπο με τον οποίο δηλώθηκε ολόκληρος ο πίνακας.

Οι θέσεις μνήμης ενός πίνακα χρησιμοποιούνται όπως οι υπόλοιπες μεταβλητές. Για παράδειγμα, οι επόμενες προτάσεις

```
a[5]=45;
```

```
a[8]=67;
```

θα καταχωρίσουν το 45 στην έκτη θέση μνήμης του πίνακα **a** και το 67 στην ένατη (`a[0]` είναι η πρώτη θέση μνήμης, `a[5]` είναι η έκτη, και `a[8]` η ένατη!).



Συνήθως, για να προσδιορίσουμε τη θέση μνήμης ενός πίνακα χρησιμοποιούμε ως αριθμοδείκτη μια μεταβλητή. Αλλάζοντας την τιμή της μεταβλητής, μπορούμε να προσπελάσουμε διαφορετικές θέσεις του πίνακα.

Για παράδειγμα, το επόμενο τμήμα προγράμματος:

```
i=5;
```

```
a[i]=45;
```

```
i=8;
```

```
a[i]=67;
```

θα καταχωρίσει στη θέση `a[5]` του πίνακα `a` τον αριθμό 45 και στη θέση `a[8]` τον αριθμό 67, ενώ το επόμενο τμήμα προγράμματος:

```
for (i=0;i<10;i++) a[i]=45;
```

θα συμπληρώσει όλες τις θέσεις μνήμης ενός πίνακα `a` 10 θέσεων με τον αριθμό 45.

Το επόμενο πρόγραμμα συμπληρώνει τον πίνακα `a` με 10 τυχαίους αριθμούς, και εμφανίζει στην οθόνη μόνον αυτούς που είναι μεγαλύτεροι από 1000:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i,a[10];
    // Συμπλήρωση πίνακα με τυχαίους αριθμούς
    for (i=0;i<10;i++)
        a[i]=rand();
    // Εμφάνιση στοιχείων του πίνακα
    for (i=0;i<10;i++)
        if (a[i]>1000) printf("%d\n",a[i]);
    return 0;
}
```

12_random1.c

Στις 10 θέσεις του πίνακα καταχωρίζονται τυχαίοι αριθμοί.

Εμφανίζονται στην οθόνη οι αριθμοί του πίνακα που είναι μεγαλύτεροι από το 1000.

Η διάσχιση ενός πίνακα μίας διάστασης συνήθως γίνεται με μια επαναληπτική δομή `for`. Η μεταβλητή που χρησιμοποιείται για τον έλεγχο της `for` χρησιμοποιείται επίσης ως αριθμοδείκτης για τον προσδιορισμό της θέσης μνήμης του πίνακα, π.χ. `a[i]`!

Το επόμενο πρόγραμμα συμπληρώνει τον πίνακα `ar` με 10 αριθμούς που δίνει ο χρήστης από το πληκτρολόγιο. Στη συνέχεια, υπολογίζει και εμφανίζει το άθροισμα των αριθμών.

```
#include <stdio.h>
int main(void)
{
    int i;
    float ar[10],sum;
    for (i=0;i<10;i++)
        scanf("%f",&ar[i]);
    sum=0;
    for (i=0;i<10;i++)
        sum=sum+ar[i];
    printf("Το άθροισμα των αριθμών είναι %f\n",sum);
    return 0;
}
```

12_arithmoi1.c

Στις 10 θέσεις του πίνακα καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Ο τελεστής `&` αποδίδει τη διεύθυνση μιας θέσης μνήμης του πίνακα.

Στη μεταβλητή `sum` προστίθενται όλοι οι αριθμοί που περιέχει ο πίνακας `ar`.

Το πρόγραμμα που ακολουθεί συμπληρώνει έναν πίνακα `a` με 10 τυχαίους αριθμούς και τους εμφανίζει αντίστροφα. Στη συνέχεια υπολογίζει και εμφανίζει το πλήθος των ζυγών αριθμών του πίνακα.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i,a[10],pl=0;
    for (i=0;i<10;i++)
        a[i]=rand();
    for (i=9;i>=0;i--)
        printf("%d\n",a[i]);
    for (i=0;i<10;i++)
        if (a[i]%2==0) pl++;
    printf("Πλήθος ζυγών = %d\n",pl);
    return 0;
}
```

12_arithmoi2.c

Στις 10 θέσεις του πίνακα καταχωρίζονται τυχαίοι αριθμοί.

Εμφανίζονται στην οθόνη οι αριθμοί του πίνακα με αντίστροφη σειρά.

Υπολογίζει και εμφανίζει το πλήθος των ζυγών αριθμών του πίνακα (το υπόλοιπο της διαίρεσης με το 2 πρέπει να είναι 0).

Χειρισμός πινάκων μίας διάστασης

Ο χειρισμός πινάκων μίας διάστασης συνήθως γίνεται με έναν βρόχο **for** που διασχίζει όλες τις θέσεις του πίνακα. Το επόμενο πρόγραμμα δημιουργεί έναν πίνακα ακεραίων 100 θέσεων και τον συμπληρώνει με τυχαίους αριθμούς. Στη συνέχεια επεξεργάζεται τον πίνακα, υπολογίζει το άθροισμα των στοιχείων του και, τέλος, αναζητά μέσα στον πίνακα έναν αριθμό που δίνει ο χρήστης.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
int main(void)
{
    int i,a[100],sum=0,ar;
    bool found=false;
    for (i=0;i<100;i++) a[i]=rand();
    for (i=0;i<100;i++) printf("%d\n",a[i]);
    for (i=0;i<100;i++) sum=sum+a[i];
    printf("Το άθροισμα των αριθμών είναι:%d\n",sum);
    printf("Δώσε αριθμό για εύρεση:");
    scanf("%d",&ar);
    for (i=0;i<100;i++)
    {
        if (a[i]==ar) found=true;
        if (found) break;
    }
    if (found)
        printf("Υπάρχει στον πίνακα\n");
    else
        printf("Δεν υπάρχει στον πίνακα\n");
    return 0;
}
```

12_sum_1d.c

Στις 100 θέσεις του πίνακα καταχωρίζονται τυχαίοι αριθμοί.

Εμφανίζονται στην οθόνη οι αριθμοί του πίνακα.

Υπολογίζει και εμφανίζει το άθροισμα όλων των στοιχείων του πίνακα.

Εντοπίζει αν ο αριθμός **ar** που έδωσε ο χρήστης υπάρχει μέσα στον πίνακα. Στην περίπτωση που εντοπιστεί ο αριθμός (**found==true**), η διάσχιση του πίνακα διακόπτεται.

Το πρόγραμμα χρησιμοποιεί τη λογική μεταβλητή **found**, με αρχική τιμή **false**. Στη μεταβλητή καταχωρίζει την τιμή **true** (1) στην περίπτωση που εντοπίσει τον αριθμό **ar** μέσα στον πίνακα. Κατόπιν ελέγχει την τιμή αυτής της μεταβλητής. Αν εξακολουθεί να περιέχει την αρχική τιμή **false** σημαίνει ότι ο αριθμός δεν εντοπίστηκε.

Το πρόγραμμα που ακολουθεί συμπληρώνει έναν πίνακα πραγματικών αριθμών, 10 θέσεων, με αριθμούς που πληκτρολογεί ο χρήστης. Στη συνέχεια βρίσκει και εμφανίζει τον μικρότερο και τον μεγαλύτερο αριθμό του πίνακα, καθώς και τις θέσεις του πίνακα στις οποίες τους εντόπισε.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i, th_min, th_max;
    float a[10], min, max;
    for (i=0; i<10; i++)
        scanf("%f", &a[i]);
    min=max=a[0];
    th_min=th_max=0;
    for (i=0; i<10; i++)
    {
        if (a[i]<min)
        {
            min=a[i];
            th_min=i;
        }
        if (a[i]>max)
        {
            max=a[i];
            th_max=i;
        }
    }
    printf("Ο μικρότερος αριθμός είναι: %f στη θέση %d\n", min, th_min);
    printf("Ο μεγαλύτερος αριθμός είναι: %f στη θέση %d\n", max, th_max);
    return 0;
}
```

12_minmax_1d.c

Στις θέσεις του πίνακα καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Στις μεταβλητές **min** και **max** καταχωρίζεται η τιμή της πρώτης θέσης μνήμης του πίνακα.

Αν κάποιο στοιχείο του πίνακα έχει τιμή μικρότερη από τη **min**, τότε στη **min** καταχωρίζεται η τιμή του στοιχείου και στη **th_min** καταχωρίζεται η θέση όπου εντοπίστηκε, δηλαδή η τιμή του **i**.

Στην περίπτωση που κάποιο στοιχείο του πίνακα έχει τιμή μικρότερη από τη **max**, στη **max** καταχωρίζεται η τιμή αυτή και στη **th_max** καταχωρίζεται η θέση που εντοπίστηκε, δηλαδή η τιμή του **i**.

Στο τέλος οι **min** και **max** θα περιέχουν αντίστοιχα τη μικρότερη και τη μεγαλύτερη τιμή του πίνακα, και οι μεταβλητές **th_min** και **th_max** τις αντίστοιχες θέσεις τους.

Στις θέσεις **th_min** και **th_max** καταχωρίζεται ως αρχική τιμή το 0, για την περίπτωση που ο μικρότερος ή ο μεγαλύτερος αριθμός βρίσκονται στην πρώτη θέση μνήμης του πίνακα (θέση 0). Η τεχνική της εύρεσης του μεγαλύτερου και του μικρότερου από ένα σύνολο αριθμών έχει αναλυθεί στο Κεφάλαιο 8 (σελ. 223). Αν χρειάζεται, ανατρέξτε πάλι σε αυτό το κεφάλαιο για μια ανασκόπηση!

Πίνακες μίας διάστασης και δείκτες

Υπάρχει μια πολύ στενή σχέση μεταξύ των πινάκων και των δεικτών.

Με απλά λόγια, το όνομα ενός πίνακα είναι ένας δείκτης ο οποίος περιέχει τη διεύθυνση της πρώτης θέσης μνήμης του.

Ο δείκτης αυτός δείχνει σε θέσεις μνήμης του τύπου με τον οποίο δηλώθηκε ο πίνακας. Στο παράδειγμα του Σχήματος 12.2α, με τη δημιουργία του πίνακα `a` δημιουργείται στην ουσία και μια μεταβλητή δείκτη, προς θέσεις μνήμης τύπου `int`, με όνομα `a`. Στον δείκτη αυτόν καταχωρίζεται αυτόματα η διεύθυνση της πρώτης θέσης του πίνακα. Τις θέσεις μνήμης ενός πίνακα μπορούμε να τις προσπελάσουμε με δύο διαφορετικούς τρόπους: είτε χρησιμοποιώντας τον αριθμό αναφοράς τους μέσα στον πίνακα (`a[5]`), είτε χρησιμοποιώντας τη λογική των δεικτών:

```
a[0]=12    ⇔    *a=12
a[1]=15    ⇔    *(a+1)=15
a[2]=20    ⇔    *(a+2)=20
a[9]=34    ⇔    *(a+9)=34
```

Οι προηγούμενες παραστάσεις στα αριστερά είναι απολύτως ισοδύναμες με τις αντίστοιχες τους στα δεξιά. Έτσι, η παράσταση `*(a+9)` αναφέρεται στη θέση μνήμης με διεύθυνση 5036 αφού, με την αριθμητική δεικτών, αν προστεθεί στο `a` (που έχει τιμή 5000) το 9 το αποτέλεσμα θα είναι 5036! Ο `a` είναι ένας δείκτης προς δεδομένα τύπου `int`, οπότε η πρόσθεση κάθε μονάδας στον δείκτη `a` έχει ως αποτέλεσμα την αύξησή του κατά 4 ($a+9 \Rightarrow 5000 + 4*9 \Rightarrow 5036$).

Ας υποθέσουμε ότι έχουμε τη πρόταση δήλωσης `type a[n]`; η οποία δηλώνει έναν πίνακα `a` τύπου `type` με `n` θέσεις. Από τα παραπάνω συμπεραίνουμε ότι δημιουργείται ταυτόχρονα και ένας δείκτης `a` προς θέσεις μνήμης τύπου `type`, στον οποίο καταχωρίζεται η διεύθυνση της πρώτης θέσης του πίνακα. Έστω ότι η διεύθυνση αυτή είναι η `base`. Για να μπορέσει ο μεταγλωττιστής να προσπελάσει μια θέση μνήμης `a[i]` του πίνακα, θα πρέπει να μπορεί να υπολογίσει τη διεύθυνσή της. Αυτό το κάνει χρησιμοποιώντας τον παρακάτω τύπο:

διεύθυνση της `a[i] \rightarrow base + i*sizeof(type)`

Όπως βλέπουμε, υπάρχει στενή σχέση μεταξύ πινάκων και δεικτών. Ποια όμως είναι η διαφορά τους;

Η κάθε μία από τις ακόλουθες προτάσεις δημιουργεί έναν δείκτη `ptr` προς θέσεις μνήμης τύπου `int`:

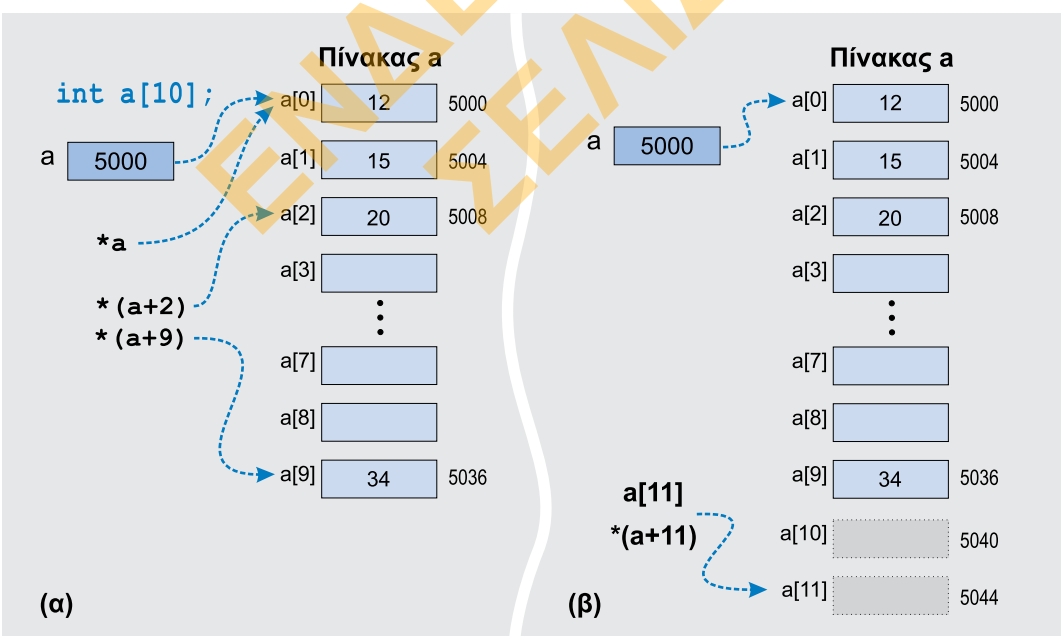
```
int ptr[100];
int *ptr;
```

Έτσι, προκύπτει το ερώτημα : **είναι ισοδύναμες αυτές οι προτάσεις**; Η απάντηση είναι **ΟΧΙ**, επειδή η πρώτη πρόταση, εκτός από το ότι δημιουργεί έναν δείκτη `ptr`, δεσμεύει επίσης 100 θέσεις μνήμης τύπου `int` και **δίνει στον δείκτη ως αρχική τιμή** τη διεύθυνση της πρώτης θέσης μνήμης. Η δεύτερη πρόταση απλώς δημιουργεί έναν δείκτη προς θέσεις μνήμης τύπου `int`, χωρίς να του δίνει αρχική τιμή και χωρίς να δεσμεύει καμία άλλη θέση μνήμης.

Οι αγκύλες (`[]`) είναι στην πραγματικότητα ένας **τελεστής**. Θα τον ονομάσουμε τελεστή προσπέλασης πίνακα. Η λειτουργία του τελεστή `[]` είναι η ακόλουθη: Αν `a` είναι ένας πίνακας μίας διάστασης και `i` είναι ένας ακέραιος, τότε η παράσταση `a[i]` είναι ισοδύναμη με την `*(a+i)`.

Προσοχή: Η C δεν ελέγχει αν έχει ξεπεραστεί το όριο ενός πίνακα, το οποίο πολλές φορές (σχεδόν πάντα) έχει καταστροφικές συνέπειες. Ας εξετάσουμε τον πίνακα `a` του Σχήματος 12.2α, ο οποίος διαθέτει δέκα θέσεις μνήμης, από την `a[0]` μέχρι την `a[9]`. Παρόλα αυτά, μπορούμε κάλλιστα να προσπελάσουμε και να χρησιμοποιήσουμε τη θέση `a[11]`, χωρίς η C να εμφανίσει κανένα μήνυμα σφάλματος (Σχήμα 12.2β). Το τι θα επακολουθήσει εξαρτάται μόνο από την τύχη μας.

Στη C, ο έλεγχος των ορίων ενός πίνακα αποτελεί ευθύνη του προγραμματιστή.



Σχήμα 12.2 Σχέση πινάκων 1Δ και δεικτών

Η αναφορά `a[0]` επιστρέφει τη **διεύθυνση** της πρώτης θέσης της πρώτης γραμμής του πίνακα, η `a[1]` επιστρέφει τη **διεύθυνση** της πρώτης θέσης της δεύτερης γραμμής του πίνακα, η `a[2]` επιστρέφει τη **διεύθυνση** της πρώτης θέσης της τρίτης γραμμής του πίνακα, κ.ο.κ. Στο Σχήμα 12.5 φαίνεται ο τρόπος υπολογισμού της διεύθυνσης που επιστρέφει η αναφορά `a[2]`. Η παράσταση `a[2]` είναι ισοδύναμη με την `&a[2][0]`. Στην πραγματικότητα, ο δείκτης `a` είναι ένας **δείκτης σε πίνακα** `int[3]`. Έτσι, η προσπέλαση των θέσεων μνήμης ενός πίνακα 2Δ μέσω δεικτών είναι αρκετά πολύπλοκη διαδικασία. Στη συνέχεια του κεφαλαίου θα περιγραφεί αναλυτικά η σχέση δεικτών και πινάκων πολλών διαστάσεων, όπου (ελπίζω) θα διασαφηνιστεί περισσότερο ο εσωτερικός χειρισμός τους από τη C.

Πίνακες με περισσότερες από δύο διαστάσεις

Η C υποστηρίζει πίνακες με περισσότερες από δύο διαστάσεις (αναφέρονται επίσης ως *πολυδιάστατοι πίνακες*). Η δήλωση ενός πίνακα με πολλές διαστάσεις γίνεται όπως στο επόμενο παράδειγμα:

```
int pin[4][6][10];
```

Με την πρόταση αυτή δηλώνεται ένας πίνακας τριών διαστάσεων, με τα εξής όρια:

0~3 ➡ για την πρώτη διάσταση

0~5 ➡ για τη δεύτερη διάσταση

0~9 ➡ για την τρίτη διάσταση

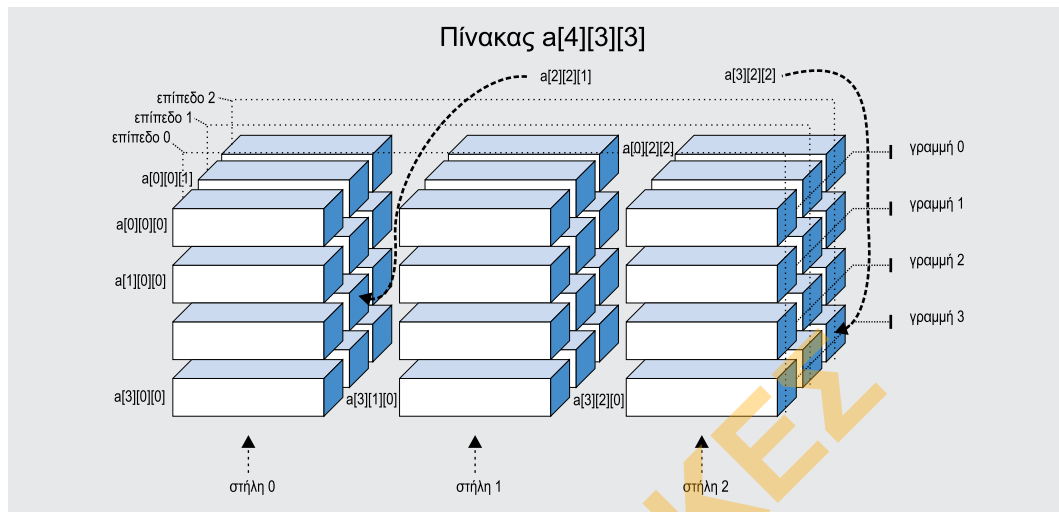
Ο πίνακας έχει συνολικά $4 \times 6 \times 10 = 240$ θέσεις μνήμης

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα πολλών διαστάσεων, ο ενδεδειγμένος τρόπος είναι η χρήση ένθετων βρόχων `for`, με τη βοήθεια των οποίων αλλάζουν οι αριθμοί αναφοράς για κάθε θέση μνήμης. Για παράδειγμα, ο κώδικας που ακολουθεί συμπληρώνει έναν πίνακα τριών διαστάσεων με τυχαίους αριθμούς:

```
int main(void)
{
    int pin[4][6][10], i, j, k;
    for (i=0; i<4; i++)
        for (j=0; j<6; j++)
            for (k=0; k<10; k++)
                pin[i][j][k]=rand();
    return 0;
}
```

Με τα τρία ένθετα `for` γίνονται όλοι οι συνδυασμοί των `i`, `j`, και `k` (μέσα στα όρια των διαστάσεων του πίνακα), με αποτέλεσμα να προσπελάζονται όλες οι θέσεις του.

Αν και στην πραγματικότητα οι θέσεις ενός πίνακα πολλών διαστάσεων είναι συνεχόμενες, αρκετές φορές είναι πολύ βολικό να έχουμε στο μυαλό μας μια εποπτική και σίγουρα πιο παραστατική εικόνα του πίνακα. Έτσι, μπορούμε να παρομοιάσουμε έναν πίνακα τριών διαστάσεων σαν ένα σύνολο από θέσεις μνήμης, διατεταγμένες σε γραμμές, στήλες, αλλά και επίπεδα.



Σχήμα 12.6 Απεικόνιση πίνακα τριών διαστάσεων

Στο Σχήμα 12.6, ο πίνακας **a** τριών διαστάσεων (**a[4][3][3]**) απεικονίζεται ως ένα σύνολο θέσεων μνήμης σε διάταξη τεσσάρων γραμμών, τριών στηλών, και τριών επιπέδων. Για να αναφερθούμε σε μια θέση μνήμης πρέπει να καθορίσουμε τη γραμμή, τη στήλη, και το επίπεδο στο οποίο βρίσκεται. Χρησιμοποιώντας τις τρεις διαστάσεις του χώρου, μπορούμε να απεικονίσουμε πίνακες μέχρι τρεις διαστάσεις. Πώς μπορούμε όμως να αναπαραστήσουμε σχηματικά έναν πίνακα περισσότερων διαστάσεων; Είναι δυνατό να σχεδιάσουμε σε χαρτί έναν πίνακα τεσσάρων, πέντε, ή και περισσότερων διαστάσεων;

Και βέβαια μπορούμε, αλλά θα πρέπει να ξεχάσουμε τον τρισδιάστατο χώρο που χρησιμοποιούσαμε μέχρι στιγμής, και να φανταστούμε τις θέσεις μνήμης σαν χωριά στα οποία για να φτάσουμε θα πρέπει να επιλέξουμε τον σωστό δρόμο σε κάθε σταυροδρόμι.

Έτσι, μπορούμε να φανταστούμε έναν πίνακα μίας διάστασης σαν μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία (Σχήμα 12.7α), με τον κάθε δρόμο να καταλήγει σε ένα χωριό (μια θέση μνήμης). Επομένως, για να φτάσουμε σε ένα χωριό (θέση μνήμης), θα πρέπει να γνωρίζουμε μόνο έναν αριθμό: τον δρόμο που πρέπει να ακολουθήσουμε από την πλατεία. Αντίστοιχα, έναν πίνακα δύο διαστάσεων μπορούμε να τον παρομοιάσουμε σαν μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία, με τον κάθε δρόμο να καταλήγει σε μια άλλη πλατεία από την οποία ξεκινούν οι δρόμοι για τα χωριά-θέσεις μνήμης (Σχήμα 12.7β). Τώρα, για να φτάσουμε σε ένα χωριό (θέση μνήμης), πρέπει να επιλέξουμε ποιον δρόμο θα ακολουθήσουμε στην πρώτη και ποιον δρόμο στη δεύτερη πλατεία. Έναν πίνακα τριών διαστάσεων, όπως τον πίνακα του Σχήματος 12.6, μπορούμε να τον παρομοιάσουμε και πάλι σαν μια πιο πολύπλοκη συστοιχία δρόμων όπου για να φτάσουμε σε κάποια θέση-χωριό πρέπει να περάσουμε από τρεις πλατείες (Σχήμα 12.7γ). Στη πρώτη πλατεία θα επιλέξουμε ανάμεσα σε τέσσερις δρόμους και στη δεύτερη και τρίτη πλατεία ανάμεσα σε τρεις. Παρόμοια, ένας πίνακας τεσσάρων διαστάσεων

μπορεί να παρομοιαστεί με μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία. Κάθε δρόμος καταλήγει σε μια άλλη πλατεία, κ.ο.κ (Σχήμα 12.7δ). Τελικά, για να φτάσουμε σε ένα χωριό (θέση μνήμης), θα πρέπει να επιλέξουμε ποιον δρόμο θα ακολουθήσουμε σε κάθε μία από τις τέσσερις πλατείες που περνάμε με τη σειρά.

Έτσι, ο πίνακας `a[4][2][3][2]` του Σχήματος 12.7δ απεικονίζεται με τέσσερα επίπεδα, όπου σε κάθε επίπεδο (διάσταση) θα πρέπει να επιλέξουμε τον δρόμο που θα πρέπει να πάρουμε σε κάθε πλατεία. Ο δρόμος που επιλέγουμε σε κάθε πλατεία είναι η τιμή της συγκεκριμένης διάστασης του πίνακα. Για παράδειγμα, η θέση μνήμης `a[3][0][2][1]` είναι η θέση στην οποία θα καταλήξουμε αν επιλέξουμε τον δρόμο 3 στην πρώτη πλατεία, τον δρόμο 0 στη δεύτερη πλατεία, τον δρόμο 2 στην τρίτη πλατεία, και τον δρόμο 1 στην τέταρτη (μην ξεχνάμε ότι η αρίθμηση των διαστάσεων ξεκινά από το 0). Με τη λογική αυτή μπορούμε να απεικονίσουμε πίνακες με όσες διαστάσεις θέλουμε. Σε αυτή την περίπτωση θα αυξηθούν τα επίπεδα, και για να καταλήξουμε σε κάποια θέση μνήμης θα πρέπει να περάσουμε από περισσότερες «πλατείες» – τόσες όσες οι διαστάσεις του πίνακα.

Σε κάθε περίπτωση, όσες διαστάσεις και αν έχει ένας πίνακας, οι θέσεις μνήμης του είναι συνεχόμενες. Το πλήθος όμως των διαστάσεων καθορίζει το μαθηματικό μοντέλο με το οποίο η γλώσσα υπολογίζει τη διεύθυνση μιας θέσης ενός πίνακα.

Μέγεθος πίνακα – εφαρμογή του τελεστή `sizeof`

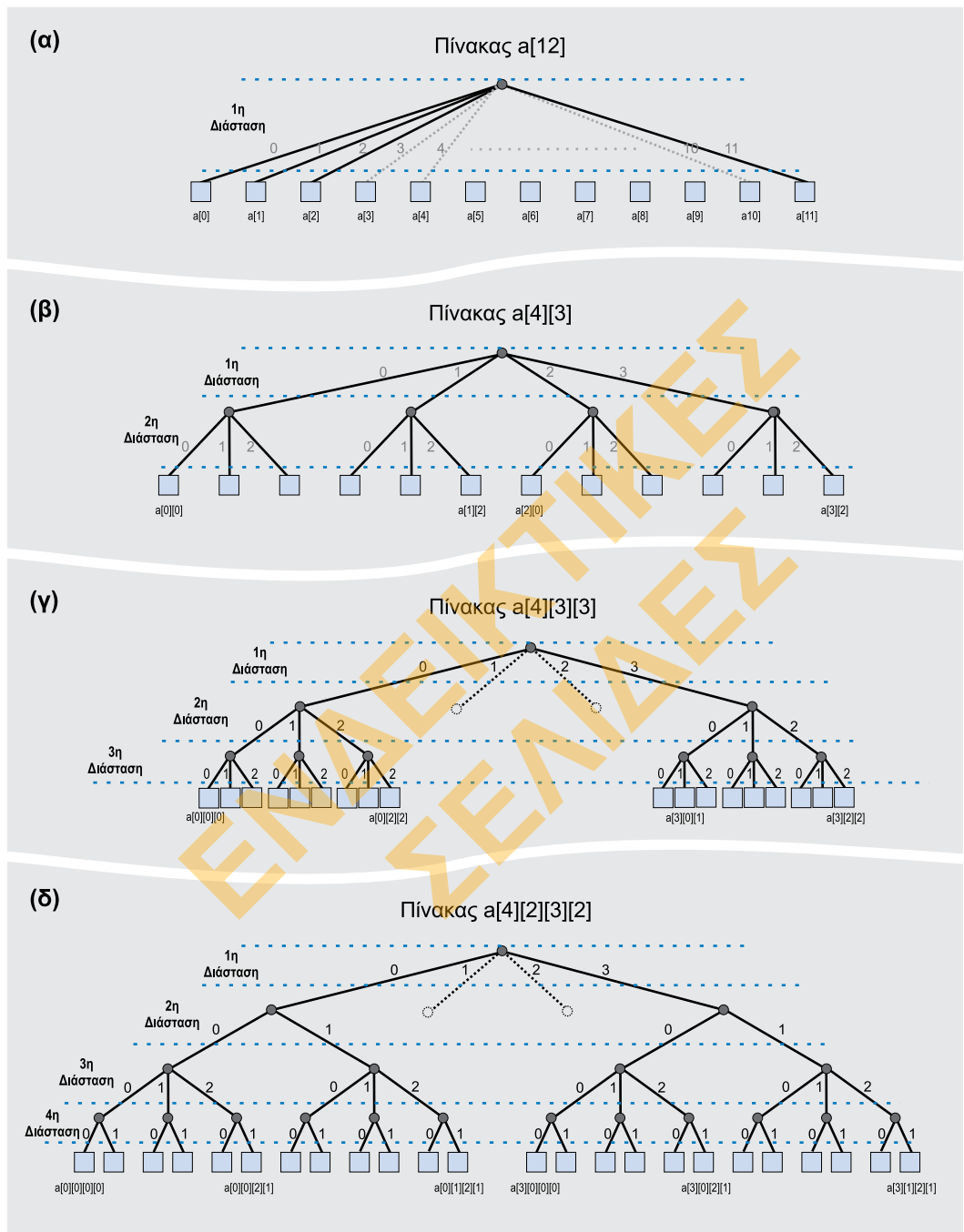
Ο τελεστής `sizeof` μπορεί να μας δώσει το συνολικό μέγεθος σε byte των θέσεων μνήμης ενός πίνακα. Έστω ότι έχουμε τις ακόλουθες δηλώσεις πινάκων:

```
int a[100], b[4][5][20];
double c[5][10];
```

Οι παραστάσεις που ακολουθούν επιστρέφουν τις τιμές που αναφέρονται:

- `sizeof a` ➔ επιστρέφει την τιμή 400 (100 θέσεις * 4 byte η κάθε μία)
- `sizeof b` ➔ επιστρέφει την τιμή 1600 (4*5*20*4 byte η κάθε μία)
- `sizeof c` ➔ επιστρέφει την τιμή 400 (5*10*8 byte η κάθε μία)
- `sizeof(c)` ➔ ίδιο με το προηγούμενο, με χρήση παρενθέσεων (επιστρέφει 400)

Ο τελεστής `sizeof` μπορεί να χρησιμοποιηθεί και χωρίς το όνομα συγκεκριμένου πίνακα, με αναφορά μόνο στον τύπο και τις διαστάσεις του. Για παράδειγμα, η παράσταση `sizeof(int[5][10])` θα επιστρέψει την τιμή 200, που είναι το μέγεθος ενός πίνακα `int` δύο διαστάσεων 5x10. Σε αυτή την περίπτωση η χρήση των παρενθέσεων είναι υποχρεωτική.



Σχήμα 12.7 Απεικόνιση πινάκων με διαφορετικό (ιεραρχικό) τρόπο

Πίνακες χαρακτήρων για την αποθήκευση συμβολοσειρών

Η C δεν υποστηρίζει κάποιο συγκεκριμένο τύπο μεταβλητής για την αποθήκευση συμβολοσειρών. **Για την αποθήκευση μιας συμβολοσειράς** (ή αλφαριθμητικού – *character string*) **χρησιμοποιείται ένας μονοδιάστατος πίνακας χαρακτήρων**. Σε κάθε θέση του πίνακα αποθηκεύεται ένας μόνο χαρακτήρας της συμβολοσειράς. Το σύνολο χαρακτήρων τερματίζεται πάντα με τον χαρακτήρα τερματισμού '\0'. Επομένως, ο πίνακας στον οποίο θα αποθηκευτεί μια συμβολοσειρά πρέπει να έχει τουλάχιστον μία θέση επιπλέον από το πλήθος των χαρακτήρων που θέλουμε να αποθηκεύσουμε. Μαζί με τη δήλωση ενός πίνακα χαρακτήρων είναι δυνατό να του δοθεί ταυτόχρονα ένα σύνολο χαρακτήρων ως αρχική τιμή. Για παράδειγμα, η πρόταση:

```
char lex[10]="ΝΙΚΟΣ";
```

δηλώνει έναν πίνακα χαρακτήρων δέκα θέσεων και καταχωρίζει 6 χαρακτήρες (τους 'N', 'I', 'K', 'O', 'Σ' και το '\0') στις 6 πρώτες θέσεις του. Οι υπόλοιπες θέσεις του αρχικοποιούνται με τον χαρακτήρα του κενού (με κωδικό ASCII 0).

- ☞ Ο χειρισμός των συμβολοσειρών γίνεται μέσω συναρτήσεων βιβλιοθήκης της C. Στις συναρτήσεις αυτές μεταβιβάζουμε κατά κανόνα τη **διεύθυνση** της συμβολοσειράς (δηλαδή τη διεύθυνση του πρώτου χαρακτήρα της).
- ☞ Η C χειρίζεται τις συμβολοσειρές γνωρίζοντας **μόνο** τη διεύθυνσή τους. Αναγνωρίζει το τέλος μιας συμβολοσειράς από τον χαρακτήρα τερματισμού '\0'.

Οι συναρτήσεις που χρησιμοποιούνται για τον χειρισμό των συμβολοσειρών δεν είναι παρά απλά προγράμματα χειρισμού πινάκων στους οποίους είναι καταχωρισμένοι οι χαρακτήρες. Μπορούμε κάλλιστα να γράψουμε δικό μας κώδικα και να μην χρησιμοποιήσουμε τις έτοιμες συναρτήσεις. Στη συνέχεια του κεφαλαίου αναφέρονται μερικές από τις συναρτήσεις βιβλιοθήκης για τον χειρισμό συμβολοσειρών. Στον ορισμό κάθε συνάρτησης αναφέρεται επίσης το όνομα του αρχείου κεφαλίδας στο οποίο δηλώνεται. Αν θέλουμε να χρησιμοποιήσουμε κάποια από αυτές τις συναρτήσεις στο πρόγραμμά μας, θα πρέπει να συμπεριλάβουμε με **#include** το αντίστοιχο αρχείο κεφαλίδας.

Χρήση της printf() και της scanf() με πίνακες χαρακτήρων

Οι συναρτήσεις **printf()** και **scanf()** μπορούν να χρησιμοποιηθούν για την εμφάνιση και την ανάγνωση συμβολοσειρών αντίστοιχα, με κατάλληλους ειδικούς χαρακτήρες στο αλφαριθμητικό μορφοποίησης.

Και οι δύο συναρτήσεις χρησιμοποιούν την ειδική ακολουθία χαρακτήρων **%s** τόσο για την εμφάνιση, όσο και για την ανάγνωση συμβολοσειρών.

char lex[10];	
lex[0]	N
lex[1]	I
lex[2]	K
lex[3]	O
lex[4]	Σ
lex[5]	\0
	⋮
lex[8]	
lex[9]	

Στον παρακάτω κώδικα, η συνάρτηση `printf()` εμφανίζει τα περιεχόμενα του πίνακα χαρακτήρων `lex`, καθώς και τη συμβολοσειρά «ΤΕΛΟΣ».

```
char lex[10]="ΝΙΚΟΣ";
printf("Ο πίνακας περιέχει: %s\n",lex);
printf("----- %s -----\n","ΤΕΛΟΣ");
```

Ο πίνακας περιέχει: ΝΙΚΟΣ
----- ΤΕΛΟΣ -----

Με τη συνάρτηση `scanf()` μπορούμε να διαβάσουμε από το πληκτρολόγιο μια συμβολοσειρά και να την καταχωρίσουμε σε έναν πίνακα χαρακτήρων.

```
char lex[10];
printf("Δώσε λέξη: ");
scanf("%s",lex);
printf("Εδωσες: %s\n",lex);
```

Δώσε λέξη: Μένη
Εδωσες: Μένη

Όπως βλέπετε, στη συνάρτηση `scanf()` μεταβιβάζουμε το όνομα του πίνακα στον οποίο θέλουμε να καταχωριστεί το σύνολο χαρακτήρων. Ας μη ξεχνάμε ότι το όνομα του πίνακα είναι ένας δείκτης ο οποίος περιέχει τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.

Θα πρέπει να σημειώσουμε επίσης ότι η `scanf()` θεωρεί τον χαρακτήρα του διαστήματος ως χαρακτήρα διαχωρισμού μεταξύ των δεδομένων που πληκτρολογούνται. Επομένως, αν στον κώδικα του προηγούμενου παραδείγματος πληκτρολογήσουμε μια φράση η οποία περιέχει κενά διαστήματα, η `scanf()` θα καταχωρίσει στον πίνακα `lex` μόνο την πρώτη λέξη. Αν θέλουμε να καταχωριστεί ολόκληρη η φράση, θα πρέπει να χρησιμοποιήσουμε τις συναρτήσεις `gets()` και `fgets()` στις οποίες θα αναφερθούμε αργότερα.

Δώσε λέξη: Μένη Γιώργος
Εδωσες: Μένη

Ο τύπος δεδομένων `size_t`

Ο τύπος `size_t` καθιερώθηκε από το πρότυπο C99 και είναι ένας μη προσημασμένος ακέραιος τύπος δεδομένων μεγέθους τουλάχιστον 16 bit. Ο τύπος αυτός ορίζεται στα αρχεία κεφαλίδων `stdlib.h` και `stddef.h`. Χρησιμοποιείται όπου χρειάζεται να προσδιοριστεί μέγεθος μνήμης, δηλαδή ένας αριθμός από byte. Για παράδειγμα, ο τελεστής `sizeof` αποδίδει μια τιμή τύπου `size_t`. Επίσης, τύπου `size_t` μπορεί να είναι και οποιαδήποτε συνάρτηση που επιστρέφει ως τιμή ένα πλήθος από byte, όπως και κάθε παράμετρος η οποία προσδιορίζει έναν αριθμό byte.

Στη συνέχεια θα συναντήσουμε τον τύπο `size_t` σε πρωτότυπα συναρτήσεων. Μην τρελαινόμαστε όμως, δεν είναι τίποτα περισσότερο από ένας ακόμα ακέραιος τύπος δεδομένων.

Παράδειγμα χρήσης συναρτήσεων χειρισμού συμβολοσειρών

Στο πρόγραμμα που ακολουθεί χρησιμοποιούνται οι συναρτήσεις που αναφέρθηκαν, για την επεξεργασία συγκεκριμένων συμβολοσειρών.

```
#include <stdio.h>
#include <string.h>
int main(void)
```

12_string_func.c

```
{
    char lex[30],ans[30];
    strcpy(lex,"ΠΟΛΙΤΙΣΜΙΚΗ");
    printf("Μήκος=%d\n",strlen(lex));
    strcat(lex," ");
    strcat(lex,"ΤΕΧΝΟΛΟΓΙΑ");
    puts(lex);
    printf("Μήκος=%d\n",strlen(lex));
    printf("Δώσε ένα τμήμα του Παν. Αιγαίου:");
    gets(ans);
    if (strcmp(lex,ans)==0)
        printf("Ναι είσαι δικός μας!!!\n");
    else if (strstr(lex,ans)!=NULL)
        printf("Είσαι μισο-δικός μας\n");
    else
        printf("Ατύχησες\n");
    puts(strcpy(lex,"ΤΕΛΟΣ"));
    return 0;
}
```

Καταχωρίζει στον πίνακα χαρακτήρων **lex** τη συμβολοσειρά «ΠΟΛΙΤΙΣΜΙΚΗ».

Εμφανίζει το πλήθος χαρακτήρων της συμβολοσειράς του πίνακα **lex**, δηλαδή το 11.

Προσθέτει στο τέλος της συμβολοσειράς του πίνακα **lex** έναν χαρακτήρα διαστήματος.

Προσθέτει στο τέλος του **lex** τη συμβολοσειρά «ΤΕΧΝΟΛΟΓΙΑ», και μετά εμφανίζει τα περιεχόμενα του **lex** καθώς και το μήκος του!

Διαβάζει μια φράση από το πληκτρολόγιο και την καταχωρίζει στον πίνακα χαρακτήρων **ans**.

Εμφανίζει το μήνυμα αν η φράση **ans** είναι ακριβώς ίδια με τα περιεχόμενα του **lex**.

Εμφανίζει το μήνυμα αν η φράση **ans** περιέχεται μέσα στη φράση του **lex**, διαφορετικά εμφανίζει «Ατύχησες»!

Καταχωρίζει στον πίνακα χαρακτήρων **lex** τη συμβολοσειρά «ΤΕΛΟΣ» και την εμφανίζει!

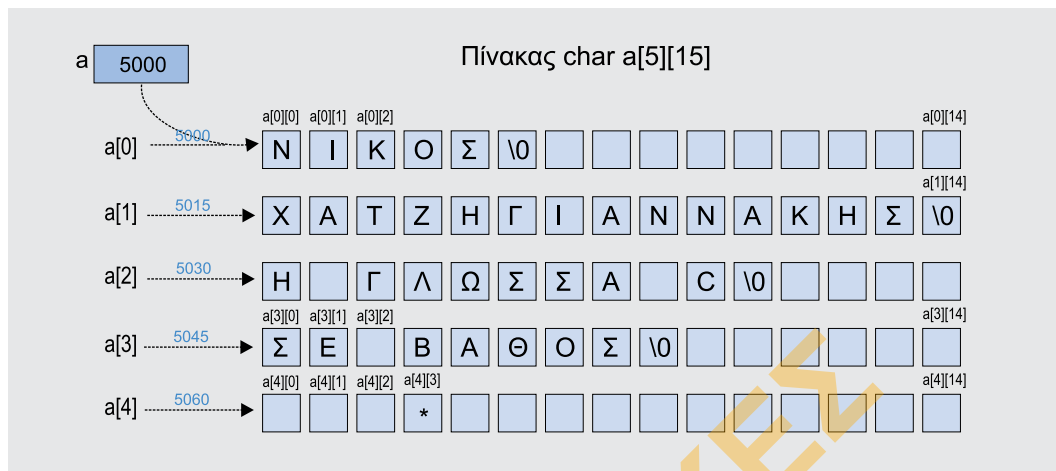
Παρατηρήστε ότι η τελευταία κλήση της **puts()** χρησιμοποιεί τον δείκτη που επιστρέφει η **strcpy()** για να εμφανίσει τα περιεχόμενα του πίνακα **lex**.

Πίνακες χαρακτήρων 2Δ για την αποθήκευση συμβολοσειρών

Οι πίνακες χαρακτήρων δύο διαστάσεων χρησιμοποιούνται για την αποθήκευση πολλών συμβολοσειρών. Σε κάθε γραμμή του πίνακα καταχωρίζεται **μία** συμβολοσειρά. Η γραμμή πρέπει να έχει επαρκές πλήθος θέσεων ώστε να χωρούν όλοι οι χαρακτήρες, συμπεριλαμβανομένου και του χαρακτήρα τερματισμού '\0' που ακολουθεί κάθε συμβολοσειρά. Για παράδειγμα, ο επόμενος κώδικας προγράμματος:

```
char a[5][15];
strcpy(a[0],"ΝΙΚΟΣ");
strcpy(a[1],"ΧΑΤΖΗΓΙΑΝΝΑΚΗΣ");
strcpy(a[2],"Η ΓΛΩΣΣΑ C");
strcpy(a[3],"ΣΕ ΒΑΘΟΣ");
a[4][3]='*';
```

καταχωρίζει στις γραμμές του πίνακα **a** τα αντίστοιχα σύνολα χαρακτήρων. Στην τελευταία γραμμή του πίνακα στη θέση **a[4][3]** καταχωρίζεται ένας αστερίσκος (*). Το Σχήμα 12.8 δείχνει εποπτικά πώς θα διαμορφωθούν τα περιεχόμενα του πίνακα.



Σχήμα 12.8 Καταχώριση πολλών συμβολοσειρών σε πίνακα χαρακτήρων 2Δ

Σε μια γραμμή του πίνακα **a** θα μπορούσε να αποθηκευτεί επίσης ένα σύνολο χαρακτήρων που εισάγεται από το πληκτρολόγιο. Για παράδειγμα, η πρόταση:

```
gets(a[4]);
```

περιμένει να πληκτρολογηθούν χαρακτήρες, και μόλις πατηθεί το <ENTER> τους αποθηκεύει στην πέμπτη γραμμή του πίνακα **a**. Αν πληκτρολογηθούν περισσότεροι από δεκατέσσερις χαρακτήρες (δεκαπέντε μαζί με το \0), τότε τα αποτελέσματα θα είναι απρόβλεπτα.

Παρατηρούμε ότι, τόσο στη συνάρτηση **strcpy()** όσο και στην **gets()**, μεταβιβάζεται η διεύθυνση της πρώτης θέσης της γραμμής του πίνακα στην οποία θέλουμε να γίνει η καταχώριση.

Για παράδειγμα, το πρόγραμμα που ακολουθεί καταχωρίζει στον πίνακα **a** πέντε συμβολοσειρές που πληκτρολογεί ο χρήστης και κατόπιν τις εμφανίζει στην οθόνη.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
```

12_fill_2d_char.c

```
{
    char a[5][15];
    int i;
    for (i=0;i<5;i++) gets(a[i]);
    for (i=0;i<5;i++) puts(a[i]);
    return 0;
}
```

Οι συμβολοσειρές που πληκτρολογεί ο χρήστης καταχωρίζονται στον πίνακα **a**.

Εμφανίζει τις αποθηκευμένες συμβολοσειρές.

Τώρα να εύχεστε ο χρήστης να μην πληκτρολογήσει συμβολοσειρά με περισσότερους από 14 χαρακτήρες!

Παράδειγμα χειρισμού πίνακα συμβολοσειρών 2Δ

Το πρόγραμμα που ακολουθεί καταχωρίζει σε έναν πίνακα χαρακτήρων δύο διαστάσεων 10 συμβολοσειρές που πληκτρολογεί ο χρήστης. Στη συνέχεια χρησιμοποιεί τις συναρτήσεις που έχουμε αναφέρει για να επεξεργαστεί τις αποθηκευμένες συμβολοσειρές.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char fraseis[10][20];
    int i,j;
    for (i=0;i<10;i++)
    {
        printf("Δώσε φράση %d ->",i+1);
        gets(fraseis[i]);
    }
    for (i=0;i<10;i++)
        puts(fraseis[i]);
    putchar(fraseis[3][2]);
    for (i=0;i<10;i++)
        putchar(fraseis[i][0]);
    putchar('\n');
    for (i=strlen(fraseis[5])-1;i>=0;i--)
        putchar(fraseis[5][i]);
    putchar('\n');
    for (i=0;i<10;i++)
        putchar(fraseis[i][strlen(fraseis[i])-1]);
    putchar('\n');
    for (i=0;i<10;i++)
        for (j=0;j<strlen(fraseis[i]);j++)
        {
            if (fraseis[i][j]>='0' && fraseis[i][j]<='9')
                fraseis[i][j]='-';
        }
    for (i=0;i<10;i++)
        puts(fraseis[i]);
    return 0;
}
```

12_strings_2d.c

Οι συμβολοσειρές που πληκτρολογεί ο χρήστης καταχωρίζονται στον πίνακα **fraseis**.

Εμφανίζει στην οθόνη τις αποθηκευμένες συμβολοσειρές.

Εμφανίζει τον 3ο χαρακτήρα της 4ης φράσης.

Εμφανίζει τον πρώτο χαρακτήρα της κάθε φράσης.

Εμφανίζει αντίστροφα τους χαρακτήρες της 6ης φράσης.

Εμφανίζει τον τελευταίο χαρακτήρα της κάθε φράσης.

Αντικαθιστά κάθε αριθμητικό χαρακτήρα (0-9) όλων των φράσεων με μια παύλα ('-').

Εμφανίζει στην οθόνη τις αποθηκευμένες συμβολοσειρές μετά από την αλλαγή των αριθμητικών τους ψηφίων.

- 🔗 Με την παράσταση **fraseis[i]** αναφερόμαστε στην συμβολοσειρά της *i*-οστής γραμμής του πίνακα.
- 🔗 Με την παράσταση **fraseis[i][j]** αναφερόμαστε στον *j*-οστό χαρακτήρα της συμβολοσειράς της *i*-οστής γραμμής του πίνακα.
- 🔗 Με την παράσταση **fraseis[i][strlen(fraseis[i])-1]** αναφερόμαστε στον τελευταίο χαρακτήρα της συμβολοσειράς της *i*-οστής γραμμής του πίνακα.
- 🔗 Μη ξεχνάμε ότι η αρίθμηση τόσο των γραμμών όσο και των στηλών ξεκινάει από το 0.

Μεταβίβαση πινάκων σε συναρτήσεις

Στη C, η μεταβίβαση των παραμέτρων μιας συνάρτησης γίνεται πάντα κατ' αξία (pass by value). Αν η παράμετρος είναι ένας πίνακας, τότε στη συνάρτηση μεταβιβάζεται η **διεύθυνση** της πρώτης θέσης μνήμης του πίνακα.

Μεταβίβαση πίνακα 1Δ σε συνάρτηση

Για τη μεταβίβαση πινάκων μίας διάστασης, η τυπική παράμετρος της συνάρτησης αρκεί να είναι ένας δείκτης του ίδιου τύπου όπως ο τύπος του πίνακα. Το παρακάτω πρόγραμμα καλεί τη συνάρτηση `printit()` για να εμφανίσει τα περιεχόμενα του πίνακα `array`.

```
#include <stdio.h>
void printit(int *p);
int main(void)
```

12_random2.c

```
{
    int array[100];
    ....
    printit(array);
}
void printit(int *p)
{
    int i;
    for (i=0;i<100;i++)
        printf("%d\n",p[i]);
}
```

Η διεύθυνση του πίνακα `array` μεταβιβάζεται στον δείκτη `p`.

Η συνάρτηση `printit()` έχει πρόσβαση στον πίνακα `array` μέσω του δείκτη `p`. Μέσα στη συνάρτηση, ο `p` χρησιμοποιείται σαν να ήταν κανονικός πίνακας. Ας μην ξεχνάμε ότι: $p[i] \Leftrightarrow *(p+i)$

Ο συνήθης τρόπος δήλωσης της τυπικής παραμέτρου στην οποία μεταβιβάζεται η διεύθυνση ενός πίνακα, είναι με έναν πίνακα του ίδιου τύπου, χωρίς όμως να ορίζεται το μέγεθός του:

```
void printit(int p[])
{
    int i;
    for (i=0;i<100;i++)
        printf("%d\n",p[i]);
}
```

Η τυπική παράμετρος δηλώνεται ως πίνακας χωρίς μέγεθος. Το αποτέλεσμα είναι η έμμεση δημιουργία ενός δείκτη `p`. Ακόμα και αν οριστεί το μέγεθος του πίνακα `p` (π.χ. `p[100]`), αυτό δεν είναι λάθος, αλλά δεν λαμβάνεται καθόλου υπόψη!

 Η τυπική παράμετρος της συνάρτησης στην οποία θα μεταβιβαστεί η διεύθυνση ενός πίνακα μίας διάστασης πρέπει να είναι ένας πίνακας ίδιου τύπου, χωρίς όμως να χρειάζεται να οριστεί το μέγεθός του. Η δήλωση `p[]` **δεν δημιουργεί** ένα νέο πίνακα στη συνάρτηση, ούτε δεσμεύει θέσεις μνήμης, απλώς δημιουργεί έμμεσα έναν δείκτη `p`. Εναλλακτικά, μπορεί να χρησιμοποιηθεί ένας δείκτης ίδιου τύπου.

Μέσα στη συνάρτηση `fillit()`, ο χειρισμός του πίνακα γίνεται ως συνήθως μέσω της τυπικής παραμέτρου `p`. Για παράδειγμα, η παράσταση `p[3]=12` θα εκχωρήσει το 12 στην τέταρτη θέση μνήμης του πίνακα `array`.

Στο Σχήμα 12.9, ο πίνακας `a` δηλώνεται στη συνάρτηση `main()` και μεταβιβάζεται στη συνάρτηση `fillit()` μέσω της τυπικής της παραμέτρου `p`.

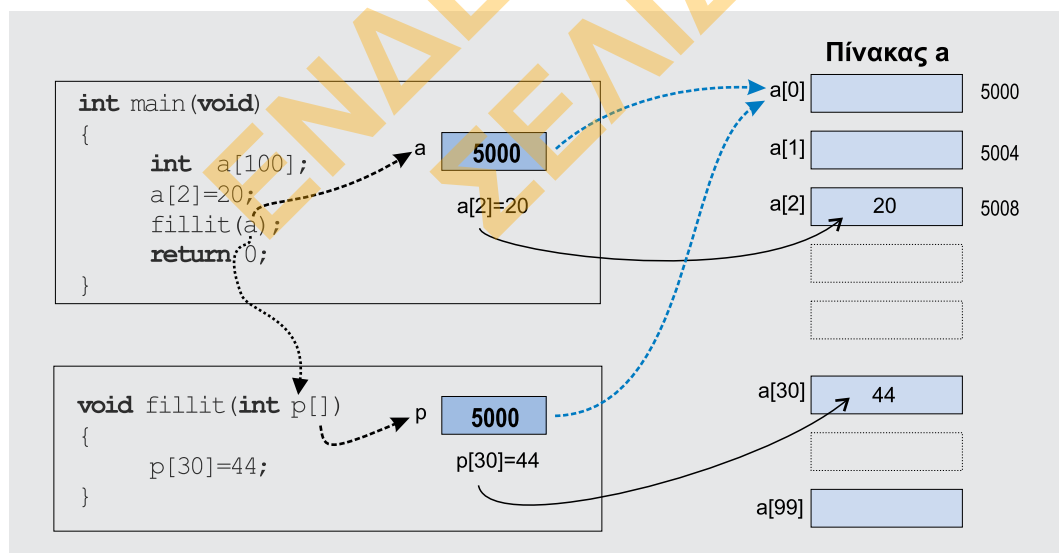
Μέσα από τη `main()`, ο πίνακας μπορεί να προσπελαστεί μέσω του ονόματος `a`, ενώ μέσα από τη συνάρτηση `fillit()` ο ίδιος πίνακας `a` προσπελάζεται μέσω του ονόματος `p`.

Ο πίνακας `a` μέσα στη `fillit()` είναι άγνωστος. Για την προσπέλαση του πίνακα `a` από την `fillit()` χρησιμοποιείται το αναγνωριστικό `p`.

Όταν θέλουμε να μεταβιβάσουμε έναν πίνακα μίας διάστασης σε μια συνάρτηση, η αντίστοιχη παράμετρος πρέπει να είναι είτε ένας πίνακας είτε ένας δείκτης του ίδιου τύπου. Αν είναι πίνακας, τότε μπορούμε να παραλείψουμε το μέγεθός του.

Παραδείγματα συναρτήσεων επεξεργασίας πινάκων

Για την επεξεργασία ενός πίνακα, συνήθως χρησιμοποιούμε μια συνάρτηση στην οποία μεταβιβάζουμε τον πίνακα. Στη συνέχεια παρουσιάζονται οι περιπτώσεις επεξεργασίας που συναντήσαμε στην αρχή του κεφαλαίου ως συναρτήσεις, στη παράμετρο των οποίων, μεταβιβάζουμε έναν πίνακα τύπου `int` 100 θέσεων μνήμης.



Σχήμα 12.9 Μεταβίβαση πίνακα μίας διάστασης σε συνάρτηση

Συμπλήρωση πίνακα με τυχαίες τιμές:

```
void fill_it(int p[])
{
    int i;
    for (i=0;i<100;i++)
        p[i]=rand();
}
```

Η πρόταση αυτή εκτελείται 100 φορές, με τιμές του *i* από 0 μέχρι 99. Κάθε φορά που εκτελείται, ένας τυχαίος αριθμός καταχωρίζεται σε μια θέση μνήμης του πίνακα: την πρώτη φορά στην **p[0]**, τη δεύτερη στην **p[1]**, κ.ο.κ

Εμφάνιση περιεχομένων πίνακα:

```
void print_it(int p[])
{
    int i;
    for (i=0;i<100;i++)
        printf("%d\n",p[i]);
}
```

Η πρόταση αυτή εκτελείται 100 φορές με τιμές του *i* από 0 μέχρι 99. Κάθε φορά που εκτελείται, εμφανίζονται στην οθόνη τα περιεχόμενα μιας θέσης μνήμης του πίνακα: την πρώτη φορά της **p[0]**, τη δεύτερη της **p[1]**, κ.ο.κ

Υπολογισμός αθροίσματος περιεχομένων πίνακα:

```
int athroisma(int p[])
{
    int i,sum;
    sum=0;
    for (i=0;i<100;i++)
        sum=sum+p[i];
    return sum;
}
```

Κάθε φορά που εκτελείται η πρόταση **sum=sum+p[i]**, προστίθενται στη **sum** τα περιεχόμενα μιας θέσης μνήμης του πίνακα. Την πρώτη φορά προστίθενται τα περιεχόμενα της **p[0]**, τη δεύτερη τα περιεχόμενα της **p[1]**, κ.ο.κ. Στο τέλος, η **sum** θα περιέχει το άθροισμα των περιεχομένων όλων των θέσεων μνήμης του πίνακα. Η συνάρτηση επιστρέφει ως τιμή το περιεχόμενο της **sum**.

🔗 Ο κώδικας που καλεί τις παραπάνω συναρτήσεις πρέπει να μεταβιβάσει στην παράμετρο **p** τη διεύθυνση ενός πίνακα **int** 100 θέσεων.

Εμφάνιση ελάχιστης και μέγιστης τιμής περιεχομένων πίνακα:

```
void minmax(int p[])
{
    int min,max,i;
    min=max=p[0];
    for (i=0;i<100;i++)
    {
        if (p[i]<min) min=p[i];
        if (p[i]>max) max=p[i];
    }
    printf("Ο μικρότερος αριθμός είναι: %d\n",min);
    printf("Ο μεγαλύτερος αριθμός είναι: %d\n",max);
}
```

Κάθε φορά που εκτελούνται αυτές οι προτάσεις, ελέγχεται αν η τιμή μιας θέσης μνήμης του πίνακα είναι μεγαλύτερη από την τρέχουσα μεγαλύτερη ή μικρότερη τιμή που έχει βρεθεί. Αν όντως βρεθεί μικρότερη ή μεγαλύτερη τιμή, τότε αυτή αντικαθιστά την υπάρχουσα τιμή στις μεταβλητές **min** και **max** αντίστοιχα. Αυτό γίνεται για κάθε θέση μνήμης, και τελικά η **min** και η **max** θα περιέχουν αντίστοιχα τη μικρότερη και τη μεγαλύτερη τιμή από όλα τα στοιχεία του πίνακα.

Για την εύρεση του μεγαλύτερου και του μικρότερου αριθμού ακολουθείται η τεχνική η οποία παρουσιάστηκε στο Κεφάλαιο 8, αλλά και στην αρχή του παρόντος κεφαλαίου. Οι μεταβλητές **min** και **max** περιέχουν τη μέχρι στιγμής μικρότερη και μεγαλύτερη τιμή

Εξειδικευμένοι πίνακες

Ορισμένοι πίνακες χαρακτηρίζονται ως εξειδικευμένοι ως προς τον τρόπο δημιουργίας τους ή τα περιεχόμενά τους.

Πίνακες μεταβλητού μήκους

Η C υποστηρίζει τη δημιουργία πινάκων των οποίων οι διαστάσεις δεν είναι γνωστές κατά τη μεταγλώττιση, αλλά κατά την εκτέλεση ενός προγράμματος. Οι πίνακες αυτοί ονομάζονται πίνακες μεταβλητού μήκους (Variable Length Arrays ή VLAs) και προστέθηκαν στη γλώσσα C με το πρότυπο C99.

Το μέγεθος ενός πίνακα μεταβλητού μήκους προκύπτει από την τιμή μιας μεταβλητής και καθορίζεται κατά τον χρόνο εκτέλεσης του προγράμματος.

Το ακόλουθο πρόγραμμα ζητάει από τον χρήστη να πληκτρολογήσει έναν ακέραιο αριθμό, και δημιουργεί έναν πίνακα του οποίου το πλήθος των θέσεων καθορίζεται από τον αριθμό που παρέχει ο χρήστης.

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

12_vla.c

```
int main(void)
```

```
{  
    int i, ar;  
    printf("Πόσους αριθμούς θα δώσεις ? ");  
    scanf("%d", &ar);  
    float pin[ar];  
    for (i=0; i<ar; i++)  
        scanf("%f", &pin[i]);  
    return 0;  
}
```

Ο αριθμός που πληκτρολογεί ο χρήστης καταχωρίζεται στη μεταβλητή `ar`.

Δημιουργείται ένας πίνακας τόσων θέσεων όση η τιμή της μεταβλητής `ar`.

Η παρακάτω συνάρτηση δημιουργεί έναν τοπικό πίνακα του οποίου το πλήθος των θέσεων καθορίζεται από την τιμή της παραμέτρου της:

```
void fill_random(int ar)
```

```
{  
    int i, p[ar];  
    for (i=0; i<ar; i++) p[i]=rand();  
}
```

Δημιουργείται ο πίνακας `p` με το πλήθος θέσεων που καθορίζεται από την τιμή της παραμέτρου `ar`.

Συμπληρώνει τον πίνακα `p` με τυχαίους αριθμούς.

Είναι προφανές ότι και στις δύο περιπτώσεις θα πρέπει να υπάρχει επαρκής μνήμη για τη δημιουργία του πίνακα κατά τον χρόνο εκτέλεσης (run time). Σε αντίθετη περίπτωση, το πρόγραμμά μας θα καταρρεύσει!

Τετραγωνικοί πίνακες

Τετραγωνικός ονομάζεται ένας πίνακας δύο διαστάσεων με ίσο αριθμό γραμμών και στηλών. Για παράδειγμα, ο πίνακας $a[6][6]$ είναι τετραγωνικός. Οι τετραγωνικοί πίνακες διαθέτουν δύο διαγωνίους. Την κύρια διαγώνιο από επάνω αριστερά (ΒΔ) προς κάτω δεξιά (ΝΑ) και τη δευτερεύουσα από επάνω δεξιά (ΒΑ) προς κάτω αριστερά (ΝΔ) – Σχήμα 12.10α.

Οι τετραγωνικοί πίνακες έχουν ιδιότητες που χρησιμοποιούνται ευρύτατα κατά την επεξεργασία τους. Αρκετές φορές χρησιμοποιούμε τα σημεία του ορίζοντα για να προσδιορίσουμε ένα τμήμα ενός τετραγωνικού πίνακα. Η κύρια διαγώνιος χωρίζει τον πίνακα σε δύο τμήματα: Το νοτιοδυτικό (ΝΔ) και το βορειοανατολικό (ΒΑ), ενώ η δευτερεύουσα στο βορειοδυτικό (ΒΔ) και το νοτιοανατολικό (ΝΑ).

Αν υποθέσουμε ότι η γραμμή συμβολίζεται με τον αριθμοδείκτη i και η στήλη με τον j , τότε για τις θέσεις της κυρίας διαγωνίου (κ.δ) ισχύει $i=j$, για τις θέσεις κάτω αριστερά της κ.δ (ΝΔ τμήμα) ισχύει $i>j$, και για τις θέσεις επάνω δεξιά της κ.δ (ΒΑ τμήμα) ισχύει $j>i$ (Σχήμα 12.10β).

Όσον αφορά τις θέσεις της δευτερεύουσας διαγωνίου (δ.δ) ισχύει $i+j=n$, όπου n είναι το μέγεθος της διάστασης του πίνακα (στο συγκεκριμένο παράδειγμα 6). Όσο για τις θέσεις επάνω αριστερά της δ.δ (ΒΔ τμήμα), ισχύει $i+j<n$, ενώ για τις θέσεις κάτω δεξιά της δ.δ (ΝΑ τμήμα) ισχύει $i+j>n$ (Σχήμα 12.10γ).

Γενικά, μια θέση μνήμης επάνω στη κύρια διαγώνιο ενός τετραγωνικού πίνακα $n \times n$ είναι η $a[i][i]$ ενώ στη δευτερεύουσα είναι η $a[i][n-i]$.

Επεξεργασία των διαγωνίων ενός τετραγωνικού πίνακα

Το τμήμα κώδικα που ακολουθεί εμφανίζει το άθροισμα των περιεχομένων της κυρίας διαγωνίου ενός τετραγωνικού πίνακα ακεραίων $a[6][6]$.

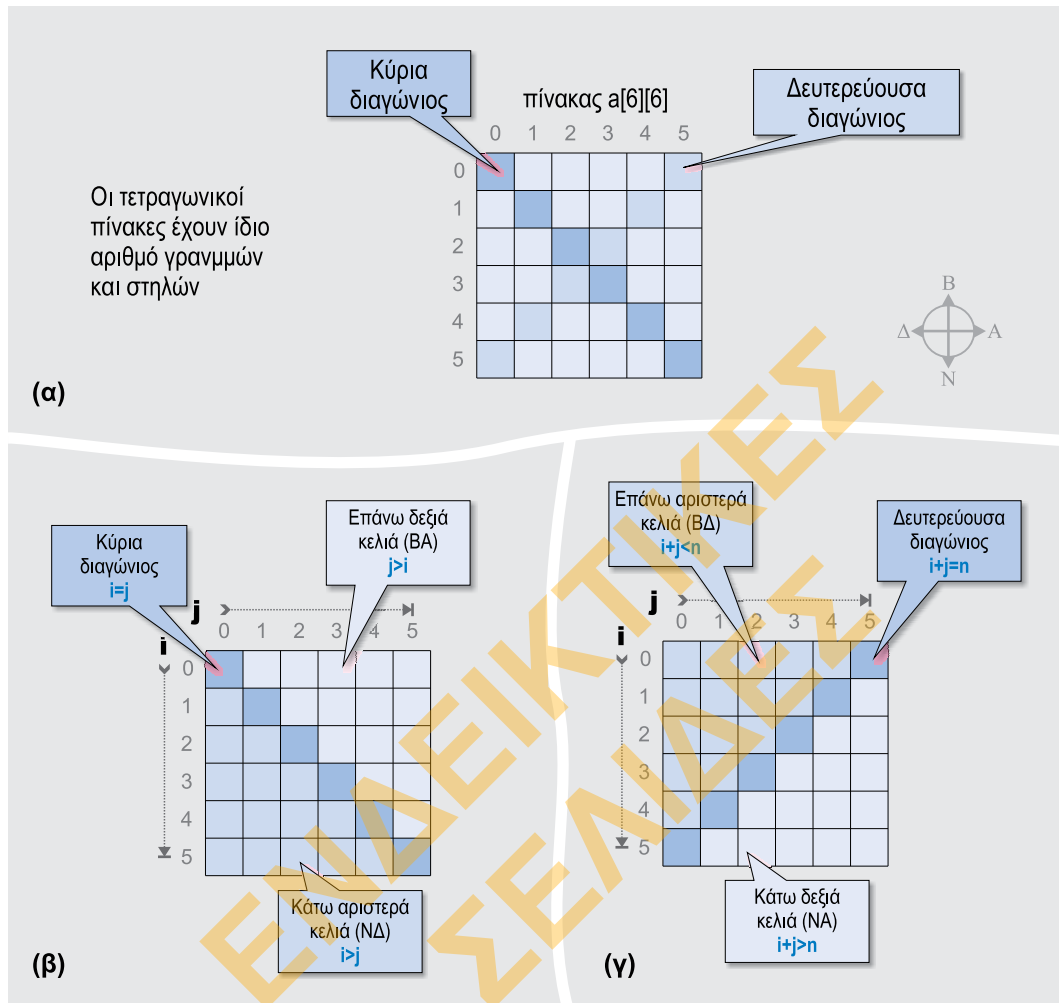
```
s=0;
for (i=0;i<6;i++) s=s+a[i][i];
printf("ΑΘΡΟΙΣΜΑ Κ.Δ = %d\n",s);
```

Το ακόλουθο τμήμα κώδικα εμφανίζει το άθροισμα των περιεχομένων της δευτερεύουσας διαγωνίου του τετραγωνικού πίνακα ακεραίων $a[6][6]$.

```
s=0;
for (i=0;i<6;i++) s=s+a[i][6-i];
printf("ΑΘΡΟΙΣΜΑ Δ.Δ = %d\n",s);
```

Το παρακάτω τμήμα κώδικα εμφανίζει το άθροισμα των περιεχομένων και των δύο διαγωνίων του τετραγωνικού πίνακα ακεραίων $a[6][6]$.

```
s=0;
for (i=0;i<6;i++) s=s+a[i][i]+a[i][6-i];
printf("ΑΘΡΟΙΣΜΑ ΔΙΑΓΩΝΙΩΝ = %d\n",s);
```



Σχήμα 12.10 Τετραγωνικοί πίνακες

Επεξεργασία των τμημάτων ενός τετραγωνικού πίνακα

Το πρόγραμμα που ακολουθεί εκμεταλλεύεται τις ιδιότητες των κελιών της κύριας διαγώνιου ($i=j$), των κελιών BA της κ.δ ($j>i$) και των κελιών NA της κ.δ ($i>j$). Καταχωρίζει τον αριθμό 1 στα κελιά της κύριας διαγώνιου, τον αριθμό 5 στο NA τμήμα του, και τον αριθμό 8 στο BA τμήμα του.

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

```
int main(void)
{
    int a[6][6], i, j;
    for (i=0; i<6; i++)
```

Πίνακες πολλών διαστάσεων και δείκτες

Μια πρώτη προσέγγιση στη σχέση των πινάκων δύο διαστάσεων και των δεικτών έγινε σε προηγούμενη ενότητα. Σε εκείνη την ενότητα, όμως, σκοπίμως παραλήφθηκαν κάποιες λεπτομέρειες ώστε να γίνουν κατανοητές μόνο οι βασικές έννοιες, χωρίς να χρειαστεί μια εις βάθος ανάλυση του τρόπου με τον οποίο η C υλοποιεί εσωτερικά τους πίνακες δύο και περισσότερων διαστάσεων. Πριν ξεκινήσουμε την ανάλυσή μας για τη σχέση των δεικτών και των πινάκων, ας θυμηθούμε τα παρακάτω:

- Η διεύθυνση ενός πίνακα είναι η διεύθυνση της πρώτης θέσης μνήμης του.
- Το μέγεθος ενός πίνακα, το οποίο αποδίδεται από τον τελεστή `sizeof`, είναι το συνολικό πλήθος σε byte όλων των θέσεων μνήμης του πίνακα.
- Η δήλωση ενός πίνακα δημιουργεί ταυτόχρονα έναν δείκτη με περιεχόμενο τη διεύθυνση του πίνακα.
- Οι αγκύλες (`[]`), που χρησιμοποιούνται για την προσπέλαση των θέσεων ενός πίνακα, στην πραγματικότητα είναι ένας τελεστής που τον ονομάζουμε τελεστή προσπέλασης πίνακα.
- Οι θέσεις που καταλαμβάνει ένας πίνακας στη μνήμη, ανεξάρτητα από το πλήθος των διαστάσεών του, είναι πάντα συνεχόμενες.

Ας θυμηθούμε τη σχέση των μονοδιάστατων πινάκων και των δεικτών. Στο Σχήμα 12.13α παρουσιάζεται ένας πίνακας τύπου `int` τεσσάρων θέσεων με όνομα `a`. Το όνομα του πίνακα είναι στην πραγματικότητα ένας δείκτης σε `int`, με αρχική τιμή τη διεύθυνση του πίνακα. Η αναφορά σε μια θέση του πίνακα μπορεί να γίνει είτε με τον τελεστή προσπέλασης πίνακα (`[]`), για παράδειγμα `a[3]`, είτε με χρήση του τελεστή αναφοράς (`*`) όπως `*(a+3)`. Ο τελεστής προσπέλασης `[]` εφαρμόζεται μόνο σε δείκτες (μην ξεχνάμε ότι το όνομα ενός πίνακα είναι ένας δείκτης), έτσι από τα αριστερά του πρέπει να υπάρχει πάντα ένας δείκτης.

Η παράσταση `p[n]` αναφέρεται στη θέση μνήμης στην οποία δείχνει ο δείκτης `p` όταν αυξηθεί κατά `n` (πάντα σύμφωνα με την αριθμητική δεικτών).

Τι γίνεται όμως με τους πίνακες δύο διαστάσεων; Στην πραγματικότητα, ένας πίνακας δύο διαστάσεων δημιουργεί έναν δείκτη σε πίνακες μίας διάστασης. Στο παράδειγμα του Σχήματος 12.13β, με τη δήλωση `int b[3][4]` δημιουργείται ένας πίνακας τύπου `int` δύο διαστάσεων (3x4). Ταυτόχρονα δημιουργείται ένας δείκτης `b`, ο οποίος δείχνει σε πίνακες τύπου `int` μίας διάστασης τεσσάρων θέσεων. Όπως αναφέραμε, όμως, ένας μονοδιάστατος πίνακας τύπου `int` υλοποιείται από έναν δείκτη σε `int`. Επομένως, ο δείκτης `b` δείχνει σε δείκτες τύπου `int`! Αυτοί οι μονοδιάστατοι πίνακες αντιστοιχούν στις τρεις γραμμές του πίνακα `b`. Ενώ έχουμε συνηθίσει να φανταζόμαστε εποπτικά έναν πίνακα δύο διαστάσεων ως θέσεις μνήμης διατεταγμένες σε γραμμές και στήλες, στην πραγματικότητα αυτές οι θέσεις είναι συνεχόμενες, όπως ακριβώς δείχνει το σχήμα.

Ας μελετήσουμε τώρα προσεκτικά τον τρόπο με τον οποίο προσπελάζονται οι θέσεις μνήμης ενός πίνακα 2Δ στη C, μέσα από το παράδειγμα του Σχήματος 12.13β. Ας υποθέσουμε ότι θέλουμε να προσπελάσουμε τη θέση `b[2][3]`. Για τον σκοπό αυτό, θα πρέπει να υπολογίσουμε τη διεύθυνση της συγκεκριμένης θέσης μνήμης. Ας δούμε βήμα προς βήμα πώς γίνεται αυτό.

Αρχικά εφαρμόζεται ο τελεστής προσπέλασης πίνακα `[2]` στον δείκτη `b`:

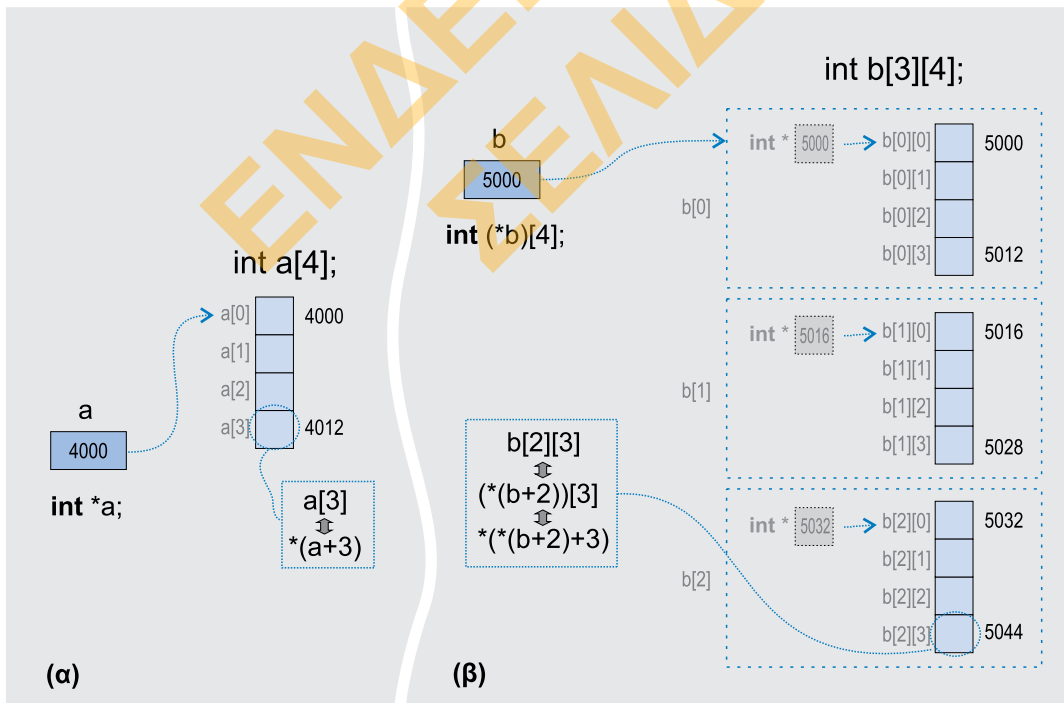
`b[2] ↔ *(b+2) →` αναφέρεται στον δείκτη `int` με περιεχόμενο 5032!

Για τον υπολογισμό αυτό, προστέθηκε στη διεύθυνση του `b` (5000) το $2 * 16$, δηλαδή το 32. Μην ξεχνάμε ότι ο δείκτης `b` είναι ένας δείκτης προς μονοδιάστατους πίνακες τύπου `int` τεσσάρων θέσεων, με μέγεθος 16 byte ο καθένας. Επομένως, για κάθε αύξηση του δείκτη `b` κατά ένα, αυτός αυξάνεται κατά 16!

Το επόμενο βήμα είναι η εφαρμογή του τελεστή προσπέλασης πίνακα `[3]` στο αποτέλεσμα του προηγούμενου βήματος, δηλαδή στον δείκτη `int` με περιεχόμενο 5032:

`((int *)5032)[3] →` αναφέρεται στη ζητούμενη θέση με διεύθυνση 5044!

Για τον υπολογισμό αυτό, προστέθηκε στη διεύθυνση 5032 το $3 * 4$, δηλαδή το 12. Θυμηθείτε ότι το 5032 είναι δείκτης σε `int`, επομένως για κάθε αύξησή του κατά ένα αυτός αυξάνεται κατά 4!



Σχήμα 12.13 Σχέση δεικτών και πινάκων σε πίνακες μίας και δύο διαστάσεων

Ένας πίνακας δύο διαστάσεων είναι ένας δείκτης σε πίνακες μίας διάστασης του ίδιου τύπου (με τον τύπο του πίνακα), οι οποίοι με τη σειρά τους είναι απλοί δείκτες σε θέσεις μνήμης επίσης του ίδιου τύπου. Για τον υπολογισμό της διεύθυνσης μιας θέσης μνήμης του πίνακα προστίθεται ο αριθμοδείκτης της πρώτης διάστασης (που προσδιορίζει τη γραμμή όπου βρίσκεται η θέση μέσα στον πίνακα) στον πρώτο δείκτη και τον αριθμοδείκτη της δεύτερης διάστασης (που προσδιορίζει τη στήλη όπου βρίσκεται η θέση μέσα στον πίνακα) στη διεύθυνση που προκύπτει. Οι υπολογισμοί γίνονται αυτόματα με τη λογική της αριθμητικής των δεικτών.

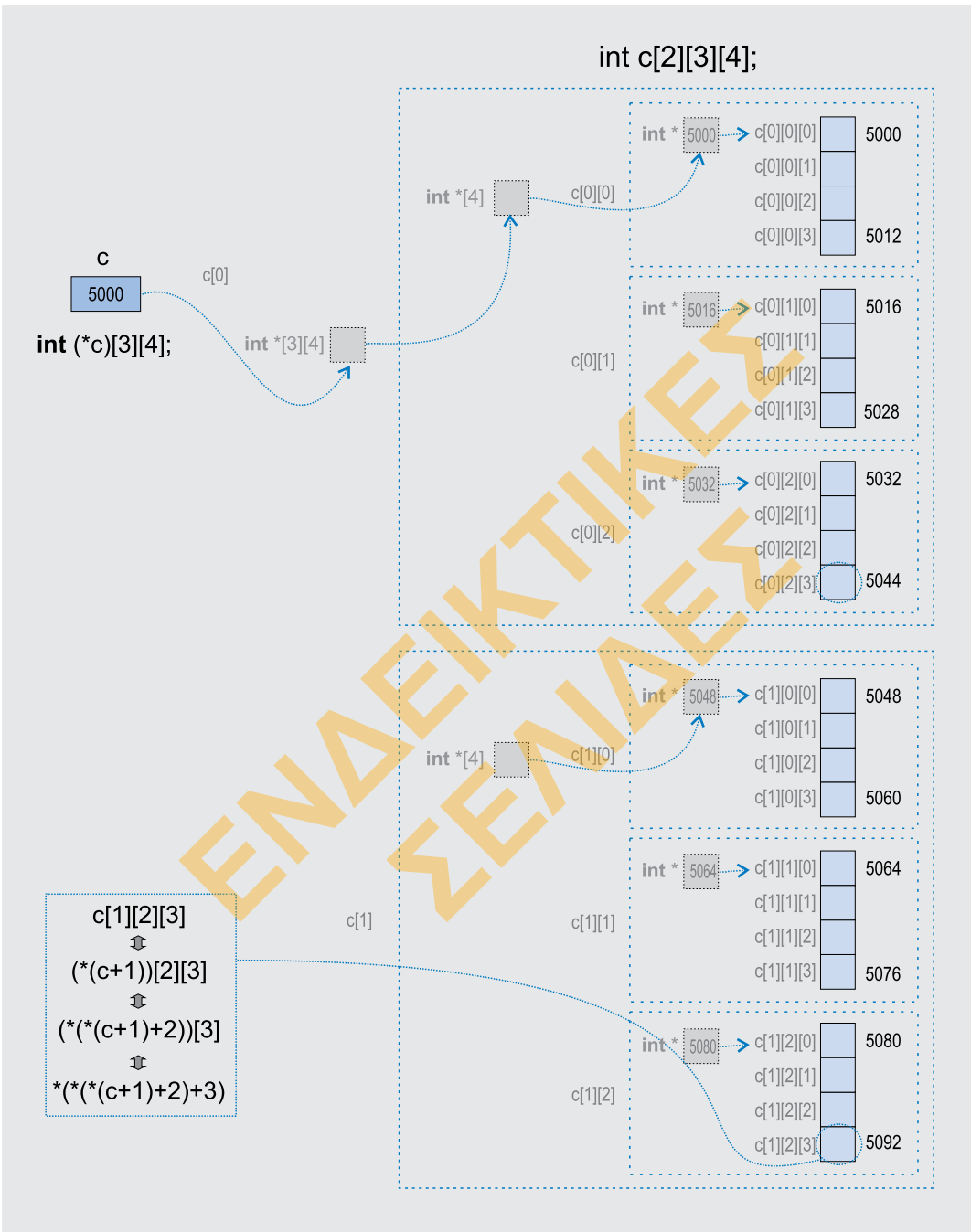
Γενικά, ένας πίνακας διαστάσεων $\delta_1 \times \delta_2 \times \delta_3 \dots \times \delta_n$, είναι ένας δείκτης σε πίνακες διαστάσεων $\delta_2 \times \delta_3 \dots \times \delta_n$, οι οποίοι με τη σειρά τους είναι δείκτες σε πίνακες διαστάσεων $\delta_3 \dots \times \delta_n$ κ.ο.κ., μέχρι το χαμηλότερο επίπεδο πινάκων μίας διάστασης, οι οποίοι είναι πλέον δείκτες σε θέσεις μνήμης και όχι σε πίνακες.

Ας μελετήσουμε τον πίνακα `c` του Σχήματος 12.14. Ο πίνακας είναι τύπου `int` τριών διαστάσεων $2 \times 3 \times 4$. Επομένως, σύμφωνα με όσα έχουμε ήδη αναλύσει, το όνομα του πίνακα είναι στην ουσία ένας δείκτης σε πίνακες `int` δύο διαστάσεων 3×4 , οι οποίοι με τη σειρά τους είναι δείκτες σε πίνακες `int` μίας διάστασης τεσσάρων θέσεων, που με τη σειρά τους είναι δείκτες σε θέσεις μνήμης επίσης τύπου `int`.

- Όταν προστεθεί μία μονάδα στον πρώτο δείκτη, ο οποίος δείχνει σε πίνακες `int` 3×4 , τότε αυτός αυξάνεται κατά 48, όσο δηλαδή είναι το μέγεθος σε byte ενός τέτοιου πίνακα ($3 \times 4 \times 4$). Έτσι προκύπτει το περιεχόμενο του δεύτερου δείκτη.
- Όταν προστεθεί μία μονάδα στον δεύτερο δείκτη, ο οποίος δείχνει σε πίνακες `int` μίας διάστασης τεσσάρων θέσεων, τότε αυτός αυξάνεται κατά 16, όσο δηλαδή είναι το μέγεθος σε byte ενός τέτοιου πίνακα (4×4). Έτσι προκύπτει το περιεχόμενο του τρίτου δείκτη.
- Όταν προστεθεί μία μονάδα στον τρίτο δείκτη, ο οποίος δείχνει σε θέσεις μνήμης τύπου `int`, τότε αυτός αυξάνεται κατά 4, όσο δηλαδή είναι το μέγεθος σε byte μίας θέσης μνήμης τέτοιου τύπου. Με αυτόν τον τρόπο προκύπτει η διεύθυνση της θέσης μνήμης.

Επομένως, αν υποθέσουμε ότι η διεύθυνση του πίνακα `c` είναι το 5000, ο υπολογισμός της διεύθυνσης της θέσης μνήμης `c[1][2][3]` γίνεται ως ακολούθως:

- Περιεχόμενο του πρώτου δείκτη `c`: 5000
- Με την εφαρμογή του τελεστή `[1]` στον πρώτο δείκτη προστίθεται το 1 στο 5000 δίνοντας αποτέλεσμα 5048. Το 5048 αποτελεί τώρα το περιεχόμενο του 2ου δείκτη.
- Με την εφαρμογή του τελεστή `[2]` στον δεύτερο δείκτη προστίθεται το 2 στο 5048 δίνοντας αποτέλεσμα $5048 + 32 \rightarrow 5080$. Το 5080 αποτελεί τώρα το περιεχόμενο του 3ου δείκτη.
- Με την εφαρμογή του τελεστή `[3]` στον τρίτο δείκτη προστίθεται το 3 στο 5080 δίνοντας αποτέλεσμα $5080 + 12 \rightarrow 5092$, που αποτελεί τη διεύθυνση της θέσης `c[1][2][3]`!



Σχήμα 12.14 Σχέση δεικτών και πινάκων σε πίνακες τριών διαστάσεων

Συχνά λάθη

- ❖ Πιστεύουμε ότι σε έναν πίνακα 10 θέσεων, π.χ. `a[10]`, η πρώτη θέση είναι η `a[1]` και η τελευταία η `a[10]`. Στη C, η πρώτη θέση του πίνακα είναι η θέση `a[0]`, και η τελευταία η `a[9]`.
- ❖ Δεν αντιλαμβανόμαστε πλήρως ότι το όνομα ενός πίνακα είναι και ένας δείκτης ο οποίος απλώς περιέχει μια διεύθυνση.
- ❖ Νομίζουμε ότι όταν μεταβιβάζουμε έναν πίνακα σε μια συνάρτηση, η συνάρτηση αποκτά ένα αντίγραφο του πίνακα. Αυτό δεν ισχύει! Η συνάρτηση έχει πρόσβαση στον **ίδιο** πίνακα που της μεταβιβάζεται, αλλά μέσω του **ονόματος** της παραμέτρου της.
- ❖ Προσπαθούμε να καταχωρίσουμε μια συμβολοσειρά σε έναν πίνακα χαρακτήρων χρησιμοποιώντας τον τελεστή ανάθεσης `=`. Αυτό γίνεται μόνο κατά την πρόταση δήλωσης του πίνακα, όπου του δίνουμε ταυτόχρονα και αρχικές τιμές. Αν αργότερα θελήσουμε να καταχωρίσουμε μια συμβολοσειρά στον πίνακα, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση βιβλιοθήκης `strcpy()`.
- ❖ Προσπαθούμε να συγκρίνουμε συμβολοσειρές με τους γνωστούς τελεστές σύγκρισης. Αυτό δεν είναι δυνατόν! Για τη σύγκριση συμβολοσειρών πρέπει να χρησιμοποιούμε τη συνάρτηση βιβλιοθήκης `strcmp()`.

Ανασκόπηση Κεφαλαίου 12

- Ένας πίνακας είναι μια διάταξη θέσεων μνήμης με ένα κοινό όνομα.
- Οι πίνακες που χρησιμοποιούνται πιο συχνά είναι οι πίνακες μίας και δύο διαστάσεων. Μπορούμε όμως να ορίσουμε πίνακες περισσότερων διαστάσεων, ανάλογα με τις ανάγκες μας.
- Ένας πίνακας ορίζει έμμεσα έναν δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- Ένας πίνακας δηλώνεται όπως και οποιασδήποτε άλλη μεταβλητή, και ακολουθείται από το πλήθος και την τιμή των διαστάσεων του. Για παράδειγμα, η πρόταση `int a[4][5]` δηλώνει έναν πίνακα `a`, δύο διαστάσεων, με είκοσι θέσεις (4×5), με την πρώτη διάσταση από το 0 μέχρι το 3 και τη δεύτερη από το 0 μέχρι το 4.
- Για την αναφορά σε μια θέση ενός πίνακα χρησιμοποιείται το όνομα του πίνακα και αριθμοί αναφοράς (αριθμοδείκτες) για την κάθε διάσταση, ώστε να προσδιοριστεί η θέση μνήμης μέσα στον πίνακα. Για παράδειγμα, η `a[2][3]` αναφέρεται στη θέση μνήμης με αριθμοδείκτες 2 και 3 για την πρώτη και δεύτερη διάσταση αντίστοιχα.

- Το όνομα ενός πίνακα αποτελεί ταυτόχρονα και έναν δείκτη με τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα. Για τον χειρισμό ενός πίνακα μπορούν να χρησιμοποιηθούν και δείκτες.
- Για την αποθήκευση ενός συνόλου χαρακτήρων απαιτείται ένας πίνακας χαρακτήρων. Για παράδειγμα, ο πίνακας `char lex[30]` επιτρέπει την καταχώριση ενός συνόλου το πολύ 30 χαρακτήρων (μαζί με τον χαρακτήρα τερματισμού `'\0'`).
- Οι πίνακες χαρακτήρων δύο διαστάσεων χρησιμοποιούνται για την καταχώριση πολλών συμβολοσειρών.
- Ο χειρισμός των συμβολοσειρών γίνεται με τις συναρτήσεις βιβλιοθήκης `strcpy()`, `strcmp()`, `strlen()`, `strcat()`, `gets()`, `puts()`, κ.λπ. Στις συναρτήσεις αυτές μεταβιβάζονται οι διευθύνσεις των συμβολοσειρών. Κάποιες από αυτές τις συναρτήσεις δηλώνονται στο αρχείο κεφαλίδας `string.h`, το οποίο θα πρέπει να συμπεριλάβουμε στο πρόγραμμά μας αν θέλουμε να τις χρησιμοποιήσουμε.
- Για τον χειρισμό πινάκων συνήθως χρησιμοποιούνται βρόχοι `for`. Για την προσπέλαση όλων των θέσεων μνήμης ενός πίνακα, ανάλογα με το πλήθος των διαστάσεών του, χρησιμοποιούνται και αντίστοιχοι ένθετοι βρόχοι `for`.
- Όταν μεταβιβάζουμε έναν πίνακα σε μια συνάρτηση, στην ουσία μεταβιβάζουμε τη διεύθυνση της πρώτης θέσης μνήμης του. Η τυπική παράμετρος πρέπει να είναι ένας πίνακας του ίδιου τύπου, όπου υποχρεωτικά πρέπει να δηλώσουμε τη μέγιστη τιμή των διαστάσεών του πλην της πρώτης. Για παράδειγμα, για τη μεταβίβαση ενός πίνακα `a` μίας διάστασης 10 θέσεων, αρκεί να γράψουμε `a[]` και για τη μεταβίβαση ενός πίνακα `a` τριών διαστάσεων $10 \times 4 \times 5$ πρέπει να γράψουμε `a[][4][5]`.
- Από το πρότυπο C99, η C υποστηρίζει και πίνακες μεταβλητού μήκους (VLA). Το μέγεθος της κάθε διάστασης ενός τέτοιου πίνακα μπορεί να προσδιοριστεί κατά τον χρόνο εκτέλεσης του προγράμματος.
- Στη C, οι δείκτες και οι πίνακες είναι άρρηκτα συνδεδεμένοι. Στην πραγματικότητα, η λειτουργικότητα των πινάκων υλοποιείται με εκτεταμένη χρήση δεικτών. Ειδικά στους πίνακες πολλών διαστάσεων χρησιμοποιούνται πίνακες δεικτών σε πίνακες, σε πολλά επίπεδα, ανάλογα με τις διαστάσεις του πίνακα.

Ασκήσεις Κεφαλαίου 12

12.1 Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; ★

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

int main(void)
{
    int a[10]={5,8,9,4,2,1,7,5,6,2},i,m=0,c=0;
    for (i=0;i<10;i++)
    {
        if (a[i]%2!=0) m++;
        if (a[i]>=5) c++;
    }
    printf("%d %d\n",m,c);
    printf("=====\n");
    for (i=9;i>=0;i--)
    {
        if (a[i]%2==0) printf("%d %d\n",i,a[i]);
    }
    return 0;
}
```

- 12.2** Να γραφεί ένα πρόγραμμα το οποίο να συμπληρώνει έναν πίνακα 100 θέσεων με τυχαίους ακέραιους αριθμούς. Κατόπιν, το πρόγραμμα πρέπει να βρίσκει τον μεγαλύτερο αριθμό του πίνακα, καθώς και τη θέση στην οποία βρίσκεται μέσα στον πίνακα (π.χ. ο μεγαλύτερος είναι ο 85 και βρίσκεται στη θέση 43 του πίνακα). ★★
- 12.3** Να γραφεί ένα πρόγραμμα το οποίο να ζητάει από τον χρήστη να πληκτρολογήσει 10 βαθμούς φοιτητών και να τους καταχωρίζει σε έναν κατάλληλο πίνακα. Κατόπιν, το πρόγραμμα πρέπει να εμφανίζει στην οθόνη τους βαθμούς που είναι μεγαλύτεροι από τον μέσο όρο της βαθμολογίας. ★★
- 12.4** Να γραφεί ένα πρόγραμμα το οποίο να ζητάει 100 αριθμούς, να τους καταχωρίζει σε έναν πίνακα μίας διάστασης, και να εμφανίζει τον μέσο όρο τους. ★
- 12.5** Να γραφεί ένα πρόγραμμα το οποίο να ζητάει 100 αριθμούς και να τους καταχωρίζει σε έναν πίνακα μίας διάστασης. Το πρόγραμμα πρέπει να κάνει τους απαραίτητους ελέγχους ώστε οι αριθμοί που δίνονται να βρίσκονται σε αύξουσα σειρά (δηλαδή, κάθε αριθμός που δίνεται πρέπει να είναι ίσος ή μεγαλύτερος από τον προηγούμενό του), διαφορετικά να μην δέχεται τον συγκεκριμένο αριθμό. ★★

12.35 Ποια από τα επόμενα αληθεύουν; ★

- ❑ Ένας πίνακας ορίζει έμμεσα και έναν δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- ❑ Σε έναν πίνακα χαρακτήρων μίας διάστασης μπορούν να καταχωριστούν πολλές συμβολοσειρές.
- ❑ Για μπορεί μια συνάρτηση να προσπελάσει έναν πίνακα μίας διάστασης, πρέπει να της μεταβιβάσουμε μόνο τη διεύθυνση του πίνακα.
- ❑ Για να μπορεί μια συνάρτηση να προσπελάσει έναν πίνακα με περισσότερες από μία διαστάσεις, εκτός από τη διεύθυνση του πίνακα πρέπει να γνωρίζει και τις τιμές των διαστάσεών του, εκτός από την πρώτη.
- ❑ Κατά τη δήλωση ενός πίνακα, οι τιμές των διαστάσεών του επιτρέπεται να είναι και μεταβλητές. Για παράδειγμα, η `int a[b]` δημιουργεί έναν πίνακα `a` με πλήθος θέσεων μνήμης ίσο με την τιμή της μεταβλητής `b`.

12.36 Με δεδομένο έναν πίνακα `a` 100 θέσεων μνήμης ο οποίος είναι συμπληρωμένος με τυχαίους αριθμούς, να γραφεί μια συνάρτηση η οποία να αντιγράφει όλους τους αριθμούς του πίνακα `a` που είναι μεγαλύτεροι από έναν αριθμό `ar` σε έναν άλλον πίνακα `b` (πάλι 100 θέσεων) χωρίς να αφήνει κενές θέσεις. Η συνάρτηση θα επιστρέφει ως τιμή το πλήθος των αριθμών που αντέγραψε. Οι πίνακες `a`, `b`, καθώς και ο αριθμός `ar`, πρέπει να μεταβιβάζονται ως ορίσματα στη συνάρτηση. ★ ★**12.37** Η αστυνομική διεύθυνση καταγράφει τις παραβάσεις που κάνουν τα αυτοκίνητα κάθε μήνα σε διάστημα ενός έτους. Να γραφεί ένα πρόγραμμα το οποίο θα διαβάσει τους αριθμούς κυκλοφορίας 100 αυτοκινήτων και θα τους καταχωρίζει σε έναν κατάλληλο πίνακα χαρακτήρων. Επίσης, για κάθε αυτοκίνητο θα διαβάσει τις παραβάσεις που έκανε κάθε μήνα για ένα ολόκληρο έτος και θα τις καταχωρίζει σε έναν πίνακα τύπου `int` δύο διαστάσεων (100×12). Το πρόγραμμα θα πρέπει να εμφανίζει τα ακόλουθα: ★ ★ ★

- Για κάθε αυτοκίνητο, τον αριθμό κυκλοφορίας του και το συνολικό (για όλο το έτος) πλήθος παραβάσεων.
- Τα αυτοκίνητα με σύνολο παραβάσεων πάνω από 10.
- Τα αυτοκίνητα με το μικρότερο συνολικό πλήθος παραβάσεων.

Κάθε μία από τις παραπάνω απαιτήσεις να υλοποιείται από μια κατάλληλη συνάρτηση.

12.38 Το σύστημα κρατήσεων μιας αεροπορικής εταιρείας δεν πρέπει να επιτρέπει την καταχώριση του ίδιου ονόματος δύο φορές σε μία πτήση. Να γραφεί ένα πρόγραμμα το οποίο θα διαβάζει τα ονόματα των επιβατών ενός αεροσκάφους 100 θέσεων και θα τα καταχωρίζει σε έναν κατάλληλο πίνακα. Η καταχώριση θα δια-

- 12.52** Συμπληρώστε κατάλληλα το πρόγραμμα που ακολουθεί ώστε να καταχωριστούν σε όλες τις θέσεις του πίνακα **a** τυχαίοι αριθμοί. Μετά, το πρόγραμμα πρέπει να εμφανίζει τα περιεχόμενα των θέσεων μνήμης του πίνακα. Όμως, πρέπει να χρησιμοποιήσετε τον πίνακα με τη λογική των **δεικτών** και **όχι** των πινάκων. Στον κώδικα που θα γράψετε **δεν** πρέπει να χρησιμοποιήσετε σε κανένα σημείο τον **τελεστή** προσπέλασης πίνακα **[]**. ★ ★ ★

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

int main(void)
{
    int a[5][10][20], i, j, k;
    ....
}
```

- 12.53** Τι θα εμφανίσει στην οθόνη το ακόλουθο πρόγραμμα; Συμπληρώστε τις κατάλληλες προτάσεις πριν από την εντολή **return** ώστε σε όλες τις συμβολοσειρές του πίνακα ο χαρακτήρας της παύλας να αντικαθίσταται από τον χαρακτήρα του διαστήματος. Εμφανίστε ξανά τις συμβολοσειρές που περιέχονται στον πίνακα. Θα πρέπει να χρησιμοποιήσετε λογική **δεικτών** και **όχι** πινάκων. **Δεν** πρέπει να χρησιμοποιήσετε σε κανένα σημείο τον **τελεστή** προσπέλασης πίνακα **[]**. ★ ★ ★

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

int main(void)
{
    char lex[5][25], *p="-ΑΙΓΑΙΟΥ";
    int i;
    strcpy(*(lex+2), "ΜΥΤΙΛΗΝΗ");
    strcpy(lex, "ΛΕΣΒΟΣ");
    strcpy(*(lex+1), "ΑΙΓΑΙΟ");
    strcpy(*(lex+1)+6, "-ΠΕΛΑΓΟΣ");
    strcpy(*(lex+4), "-----");
    *(*(lex+4))='*';
    *(*(lex+4)+4)='*';
    strcpy(*(lex+3), "ΠΑΝΕΠΙΣΤΗΜΙΟ");
    strcat(*(lex+3), p);
    for (i=0; i<5; i++)
        printf("%s\n", *(lex+i));
    // Συμπληρώστε εδώ
    ....
    return 0;
}
```

12.54 Δίνονται οι ακόλουθες δηλώσεις πινάκων: ★★

```

int a[5][10];
char x[100];
double d[5][10][20];

```

Σημειώστε τι είναι η κάθε μία από τις ακόλουθες παραστάσεις:

a	_____
a[2]	_____
d	_____
d[3]	_____
d[3][4]	_____
x[3]	_____
x	_____
a[5][5]	_____
d[0][0][0]	_____

Αν είναι δείκτης, να αναφέρετε τον τύπο δεδομένων στον οποίο δείχνει. Αν είναι απλή θέση μνήμης, να αναφέρετε απλώς τον τύπο της.

Σημειώστε επίσης τη θέση μνήμης στην οποία αναφέρονται οι παραστάσεις που ακολουθούν. Αν δεν αναφέρονται σε θέση μνήμης, να περιγράψετε τι είναι:

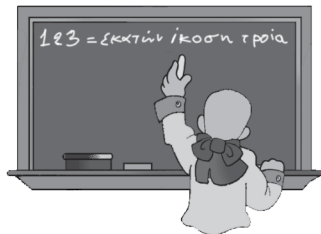
***d	_____
*(x+2)	_____
x+3	_____
((a+3)+2)	_____
**a	_____
*(a+1)	_____
((*(d+4)+5)+8)	_____
*(d+4)	_____

12.55 Σημειώστε την οντότητα που δηλώνει κάθε μία από τις ακόλουθες προτάσεις: ★★

int a[5][10];	_____
int *a[5][10];	_____
int (*a)[5][10];	_____
int (*a[5])[10];	_____
int *a[5])[5][10];	_____

3η Εργασία – Αριθμός ολογράφως

Αρκετές φορές, σε εφαρμογές όπου πρέπει να εμφανίζονται αριθμητικά αποτελέσματα αυτά πρέπει να εμφανίζονται ολογράφως. Δηλαδή, πρέπει να εμφανίζεται η λεκτική και όχι η αριθμητική αναπαράσταση ενός αριθμού. Για παράδειγμα, αντί να εμφανίζεται ο αριθμός 123 πρέπει να εμφανίζεται η φράση «εκατόν είκοσι τρία».



Κατασκευάστε λοιπόν ένα πρόγραμμα που να ζητάει από τον χρήστη έναν αριθμό και να τον εμφανίζει ολογράφως. Το πρόγραμμα θα πρέπει να δουλεύει με αριθμούς από -999.999.999 μέχρι +999.999.999. Στο πλαίσιο που ακολουθεί εμφανίζεται η μορφή των αποτελεσμάτων που θα πρέπει να εμφανίζει το πρόγραμμα σε διάφορες περιπτώσεις.

Δώσε έναν αριθμό : 4567

Τέσσερις Χιλιάδες Πεντακόσια Εξήντα Επτά

Δώσε έναν αριθμό : -999999

Μείον Εννιακόσιες Ενενήντα Εννέα Χιλιάδες Εννιακόσια Ενενήντα Εννέα

Δώσε έναν αριθμό : 146.12

Εκατό Σαράντα Εξι Κόμμα Δώδεκα

Δώσε έναν αριθμό : -1000.534

Μείον Χίλια Κόμμα Πενήντα Τρία

Οδηγίες και βοήθεια:

- Το πρόγραμμα θα πρέπει να λειτουργεί και με αρνητικούς αριθμούς. Τότε θα πρέπει να εμφανίζει τη λέξη «Μείον» μπροστά από τη λεκτική αναπαράστασή του.
- Το πρόγραμμα θα πρέπει να λειτουργεί και με πραγματικούς αριθμούς. Τότε όμως θα εμφανίζει ολογράφως μόνο τα δύο πρώτα δεκαδικά. Για παράδειγμα, αν ο αριθμός είναι ο 123,345, το πρόγραμμα πρέπει να εμφανίσει τη φράση «εκατόν είκοσι τρία κόμμα τριάντα τέσσερα».
- Αρχικά κατασκευάστε το πρόγραμμα έτσι ώστε να λειτουργεί με ακέραιους θετικούς αριθμούς. Όταν το ολοκληρώσετε και το ελέγξετε, υλοποιήστε τη δυνατότητα χρήσης αρνητικών και πραγματικών αριθμών.
- Διασπάστε τον αριθμό σε μονάδες, εκατοντάδες, χιλιάδες, και εκατομμύρια. Για τους πραγματικούς αριθμούς, διαχωρίστε το ακέραιο μέρος από το δεκαδικό.
- Χρησιμοποιήστε έναν πίνακα χαρακτήρων μέσα στον οποίο θα καταχωρίζεται η περιγραφή του αριθμού.
- Χρησιμοποιήστε συναρτήσεις για κάθε τμήμα του. Για παράδειγμα, χρησιμοποιήστε μια συνάρτηση για τις δεκάδες, μια άλλη για τις εκατοντάδες, κ.λπ.
- Στον φάκελο ΕΡΓΑΣΙΕΣ/Ε3 του συνοδευτικού υλικού θα βρείτε το εκτελέσιμο αρχείο του προγράμματος.
- Ετοιμάστε μπόλικο καφέ, θα τον χρειαστείτε!

Κεφάλαιο

Προσαρμοσμένοι
τύποι
δεδομένων



Η C μας δίνει τη δυνατότητα να δημιουργήσουμε δικούς μας, προσαρμοσμένους τύπους δεδομένων. Το κεφάλαιο αυτό παρουσιάζει αυτή τη δυνατότητα της γλώσσας.

Δίνεται ιδιαίτερο βάρος στον πιο συχνά χρησιμοποιούμενο προσαρμοσμένο τύπο δεδομένων, τις δομές.

Αναλύονται επίσης οι ενώσεις, οι απαριθμήσεις, καθώς και οι δομές με πεδία εύρους συγκεκριμένων bit.

Τέλος, γίνεται αναλυτική αναφορά στον τρόπο με τον οποίο μπορούμε να δημιουργήσουμε συνώνυμα τύπων, το οποίο βοηθάει πολύ σε πολύπλοκες δηλώσεις.

Εισαγωγή

Μέχρι τώρα γνωρίσαμε τους τέσσερις βασικούς τύπους δεδομένων της C, `int`, `char`, `float`, και `double`, παραλλαγές τους (π.χ. `long double`), καθώς και το νέο τύπο δεδομένων `bool` που εισάγει το πρότυπο C99. Ο προγραμματιστής είναι αναγκασμένος να προσαρμόζει τα δεδομένα του, καθώς και το πρόγραμμα του, στους τύπους δεδομένων που διαθέτει μια γλώσσα. Φανταστείτε να μετακομίζαμε και να είχαμε στη διάθεσή μας κουτιά τριών ή τεσσάρων συγκεκριμένων μεγεθών για τη φύλαξη και μεταφορά των πραγμάτων μας. Σε κάθε κουτί θα έπρεπε να αποθηκεύσουμε ένα μόνο αντικείμενο. Είναι προφανές ότι σε κάποια κουτιά θα περίσσευε χώρος, ενώ κάποια αντικείμενα θα έπρεπε να τα αποσυναρμολογήσουμε για να χωρέσουν στα διαθέσιμα κουτιά. Επομένως, δεν θα ήταν πολύ βολικό αν είχαμε τη δυνατότητα να κατασκευάσουμε δικά μας κουτιά, στις κατάλληλες διαστάσεις, ανάλογα με τα αντικείμενα που θέλαμε να μεταφέρουμε; Οι βασικοί τύποι δεδομένων που διαθέτει μια γλώσσα μπορούν να παρομοιαστούν με τα υπάρχοντα κουτιά συγκεκριμένου μεγέθους, ενώ όταν φτιάχνουμε δικά μας κουτιά είναι σαν να δημιουργούμε δικούς μας τύπους δεδομένων.

Η C παρέχει πέντε διαφορετικούς τρόπους για να δημιουργούμε δικούς μας τύπους δεδομένων (εκτός από τους βασικούς τύπους που έχουμε ήδη γνωρίσει). Οι προσαρμοσμένοι τύποι δεδομένων, ή οριζόμενοι από τον χρήστη τύποι (*user-defined data types*), ανήκουν στις επόμενες κατηγορίες:

- **Δομές (Structures)**
Οι δομές είναι συλλογές από μεταβλητές ή πίνακες, οι οποίες αναφέρονται με ένα κοινό όνομα.
- **Πεδία εύρους ενός ή περισσότερων bit (Bit fields)**
Είναι μια παραλλαγή δομής η οποία επιτρέπει την εύκολη προσπέλαση των επιμέρους bit ενός πεδίου.
- **Ενώσεις (Unions)**
Οι ενώσεις επιτρέπουν σε δύο ή περισσότερες μεταβλητές ή πίνακες, ακόμη και διαφορετικού τύπου, να μοιράζονται το ίδιο τμήμα μνήμης.
- **Απαριθμήσεις (Enumerations)**
Οι απαριθμήσεις είναι λίστες συμβόλων.
- **Typedef**
Με τη δεσμευμένη λέξη **typedef** μπορούμε να δημιουργούμε συνώνυμα για **υπάρχοντες** τύπους δεδομένων. **Με την typedef δεν δημιουργείται ένας νέος τύπος δεδομένων.**

Δομές

Πολλές φορές, στα προγράμματά μας έχουμε δεδομένα που σχετίζονται μεταξύ τους. Για παράδειγμα, το ονοματεπώνυμο, το έτος σπουδών, και οι βαθμοί ενός φοιτητή αποτε-

λούν δεδομένα που αναφέρονται στο ίδιο άτομο. Μέχρι τώρα, αυτό που κάναμε ήταν να χρησιμοποιούμε ξεχωριστές μεταβλητές ή πίνακες για την καταχώριση των δεδομένων μας. Για το συγκεκριμένο παράδειγμα θα χρειαζόμασταν έναν πίνακα χαρακτήρων (για το ονοματεπώνυμο), μία ακέραια μεταβλητή (για το έτος σπουδών), και έναν πίνακα πραγματικών αριθμών για τους βαθμούς του φοιτητή. Η σχέση που φαίνεται να έχουν αυτά τα δεδομένα μεταξύ τους, δηλαδή ότι αποτελούν στοιχεία του ίδιου φοιτητή, στην πραγματικότητα δεν υφίσταται και βρίσκεται μόνο μέσα στο μυαλό του προγραμματιστή. Η λύση στο παραπάνω πρόβλημα είναι η δημιουργία **σύνθετων** τύπων δεδομένων, οι οποίοι αποτελούνται από απλούστερους βασικούς τύπους. Αυτοί οι σύνθετοι τύποι λέγονται **δομές** (structures). Μια δομή στη C είναι μια συλλογή από μεταβλητές (ή πίνακες) κάθε τύπου, οι οποίες αναφέρονται με ένα κοινό όνομα. Με τη δημιουργία μιας δομής έχουμε τη δυνατότητα να ορίσουμε μεταβλητές αυτού του τύπου.

Μια δομή αποτελείται από επιμέρους πληροφορίες, οι οποίες καλούνται πεδία.

Η δομή είναι συνήθως μια λογική ενότητα από πληροφορίες που σχετίζονται μεταξύ τους. Για παράδειγμα, το όνομα, η διεύθυνση, το τηλέφωνο, και η ηλικία ενός ατόμου είναι ένα χαρακτηριστικό παράδειγμα μιας δομής με τέσσερα πεδία:

- το πεδίο του ονόματος,
- το πεδίο της διεύθυνσης,
- το πεδίο του τηλεφώνου και
- το πεδίο της ηλικίας

Μια δομή ορίζεται με μια πρόταση **struct**. Το επόμενο παράδειγμα ορίζει μια δομή με όνομα **stoxeia** με τα τέσσερα παραπάνω πεδία:

```
struct stoxeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
};
```

Ορίζεται η δομή **stoxeia** αλλά δεν δηλώνεται καμία μεταβλητή αυτής της δομής.

☞ Με την προηγούμενη πρόταση **δεν** δημιουργήθηκε καμία μεταβλητή, ούτε δεσμεύτηκε κάποιος χώρος μνήμης. Απλώς ορίστηκε ένας νέος τύπος δεδομένων με όνομα **stoxeia**. **Προσοχή:** Η δομή **stoxeia** δεν είναι μεταβλητή, αλλά ένας τύπος δεδομένων όπως ο **char**, ο **int** κ.λπ.

☞ Το τέλος της δήλωσης **struct** οριοθετείται με ελληνικό ερωτηματικό (;).

Μεταβλητές μιας δομής μπορούν να δηλωθούν είτε με μια ξεχωριστή πρόταση δήλωσης είτε ταυτόχρονα με τον ορισμό της δομής.

Μεταβλητές της δομής **stoixeia** μπορούν να δηλωθούν με προτάσεις της μορφής:

```
struct stoixeia pelatis,filos;
```

Με αυτήν την πρόταση δηλώνονται δύο μεταβλητές του τύπου δομής **stoixeia**, με ονόματα **pelatis** και **filos** αντίστοιχα. Εναλλακτικά, η δήλωση μεταβλητών μπορεί να γίνει μαζί με τον ορισμό της δομής:

```
struct stoixeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelatis,filos;
```

Ορίζεται η δομή **stoixeia** και ταυτόχρονα δηλώνονται οι μεταβλητές δομής **pelatis** και **filos**.

Εδώ δηλώνονται και πάλι οι δύο μεταβλητές δομής **pelatis** και **filos**.

Μπορούμε ακόμα να δηλώσουμε μεταβλητές δομής χωρίς να ορίσουμε κάποιο όνομα για τον τύπο της δομής. Για παράδειγμα, στην παρακάτω πρόταση:

```
struct
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} pelatis,filos;
```

Δηλώνονται δύο μεταβλητές δομής **pelatis** και **filos** αλλά δεν ορίζεται κανένας τύπος δομής.

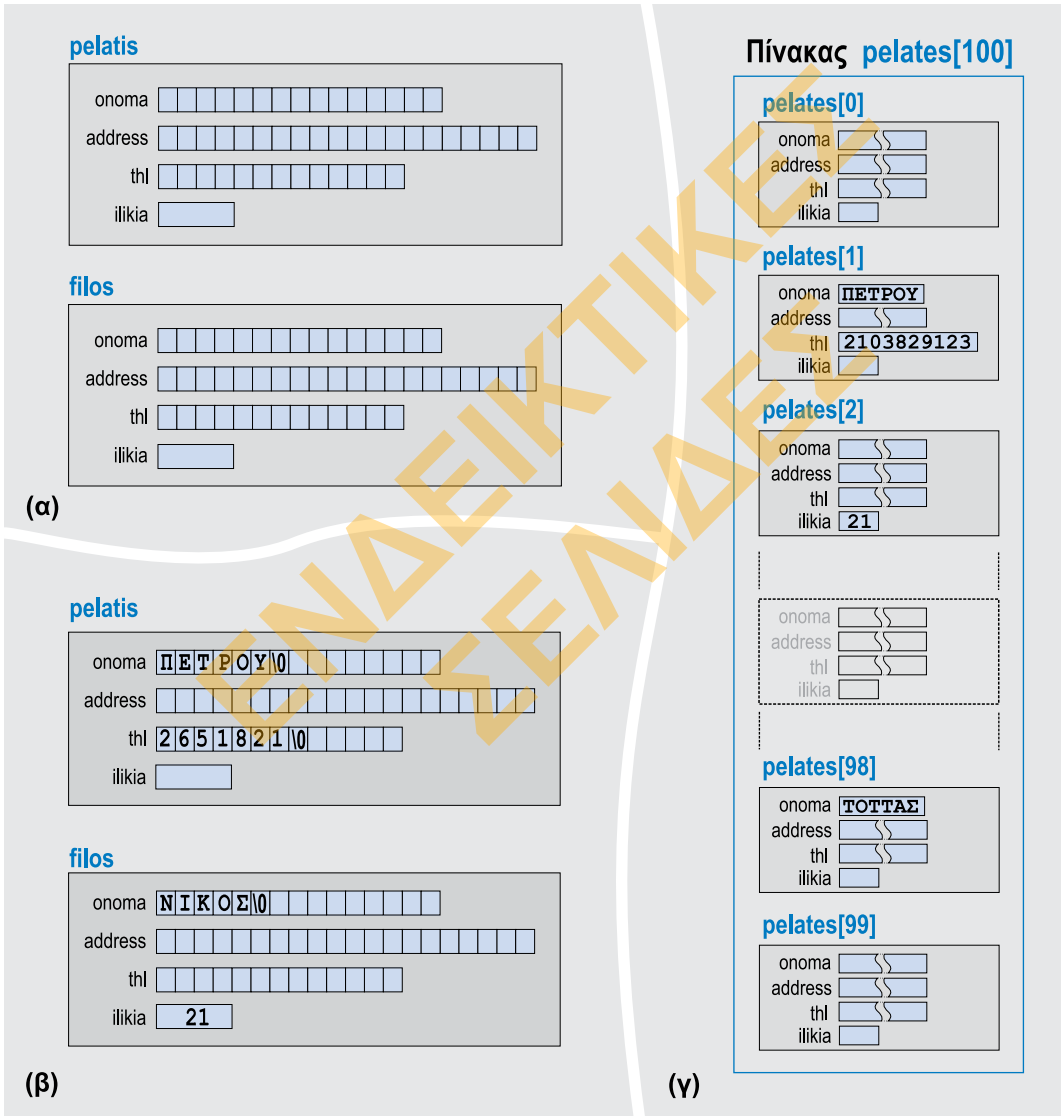
Εδώ δηλώνονται οι δύο μεταβλητές, με την προηγούμενη δομή, αλλά δεν ορίζεται κάποιο όνομα για το νέο τύπο²⁰ δεδομένων (όπως π.χ. το όνομα **stoixeia**). Σε αυτή την περίπτωση, δεν είναι δυνατό να δηλωθούν αργότερα, με μια ξεχωριστή πρόταση δήλωσης, μεταβλητές με την ίδια δομή. Γι' αυτόν τον λόγο, αυτός ο τρόπος δήλωσης δεν χρησιμοποιείται συχνά.

Ο ορισμός μιας δομής συνήθως γίνεται πριν από τη **main()** και έξω από κάθε συνάρτηση, ώστε η δομή αυτή να έχει εμβέλεια σε όλο το πρόγραμμα. Αν μαζί με τον ορισμό της δομής δηλώνονται και μεταβλητές-δομής, τότε και αυτές έχουν καθολική εμβέλεια, κάτι το οποίο δεν είναι όμως καλή προγραμματιστική τεχνική.

Με όποιον τρόπο όμως και αν δηλωθούν, η κάθε μία από τις μεταβλητές **pelatis** και **filos** καταλαμβάνει χώρο 52 byte στη μνήμη (15+20+13+4), και έχει τη σχηματική μορφή που φαίνεται στο Σχήμα 13.1α. Ο τελεστής **sizeof** αποδίδει το συνολικό μέγεθος (σε byte) μιας δομής—όπως ακριβώς και με τους βασικούς τύπους δεδομένων.

²⁰ Με την αυστηρή έννοια του όρου, η C δεν δημιουργεί ένα νέο τύπο δεδομένων. Αν συνέβαινε κάτι τέτοιο, τότε για τη δήλωση μεταβλητών αυτού του τύπου θα αρκούσε μια πρόταση της μορφής **stoixeia s1,s2;** χωρίς να χρειάζεται το πρόθεμα **struct!** Όμως, για χάρη απλότητας και εκπαιδευτικής προσέγγισης, μπορείτε να θεωρείτε μια δομή ως έναν τύπο δεδομένων.

Προσοχή: Υπάρχει περίπτωση το συνολικό πλήθος των byte που καταλαμβάνει μια μεταβλητή δομής να μην ισούται με το άθροισμα των byte των επιμέρους μελών της. Αυτό οφείλεται στην αυτόματη παρεμβολή αποθηκευτικού χώρου ανάμεσα στα μέλη. Ο χώρος αυτός δεν χρησιμοποιείται, αλλά παρεμβάλλεται για την αποδοτικότερη διαχείριση της μνήμης και εξαρτάται από τον μεταγλωττιστή. Σε κάθε περίπτωση, πάντως, ο τελεστής **sizeof** αποδίδει το πραγματικό μέγεθος μιας μεταβλητής δομής.



Σχήμα 13.1 Μεταβλητές δομής και πίνακες δομών

Η γενική μορφή δήλωσης ενός τύπου δομής είναι:

```
struct όνομα_τύπου_δομής
{
    τύπος όνομα_πεδίου;
    τύπος όνομα_πεδίου;
    .....
    τύπος όνομα_πεδίου;
} [μεταβλητές_δομής];
```

Η ορισμός μιας δομής μπορεί να είναι καθολικός (πριν από τη `main()`), οπότε ο ίδιος ο τύπος αλλά και οι μεταβλητές που δηλώνονται μαζί του έχουν εμβέλεια σε όλο το πρόγραμμα.

```
struct stoixeia
{
    char onoma[15];
    char address[20];
    char thl[13];
    int ilikia;
} filis;
```

Ο τύπος **stoixeia** έχει εμβέλεια σε όλο το πρόγραμμα.

Η μεταβλητή δομής **filis** έχει εμβέλεια σε όλο το πρόγραμμα.

```
int main(void)
{
    struct stoixeia pelatis;
    .....
    .....
    return 0;
}
```

Η μεταβλητή δομής **pelatis** έχει εμβέλεια μόνο μέσα στη `main()`.

Όταν μια δομή ορίζεται τοπικά, τότε ο τύπος της δομής είναι γνωστός μόνο μέσα στη συνάρτηση (ή σύνθετη πρόταση) στην οποία δηλώνεται.

```
int main(void)
{
    struct stoixeia
    {
        char onoma[15];
        char address[20];
        char thl[13];
        int ilikia;
    } pelatis;
    .....
    struct stoixeia filis;
    .....
    return 0;
}
```

Ο τύπος **stoixeia** έχει εμβέλεια μόνο μέσα στη `main()`.

Η μεταβλητή δομής **pelatis** έχει εμβέλεια μόνο μέσα στη `main()`.

Και η μεταβλητή δομής **filis** έχει εμβέλεια μόνο μέσα στη `main()`.

Αναφορά στα πεδία μιας δομής

Η αναφορά στα επιμέρους πεδία μιας μεταβλητής δομής γίνεται μέσω του τελεστή **τελεία (.)**. Έτσι, η επόμενη πρόταση,

```
pelatis.ilikia=36;
```

αποθηκεύει τον αριθμό 36 στο πεδίο της ηλικίας της μεταβλητής δομής **pelatis** (όπως έχει οριστεί στο παράδειγμα της προηγούμενης ενότητας).

Γενικά, για να αναφερθούμε σε ένα πεδίο μιας μεταβλητής δομής χρησιμοποιούμε το όνομα της μεταβλητής ακολουθούμενο από μία τελεία και το όνομα του πεδίου: **μεταβλητή_δομής.όνομα_πεδίου**.

Σύμφωνα με το προηγούμενο παράδειγμα με το οποίο δηλώνονται οι μεταβλητές **pelatis** και **filos** του τύπου δομής **stoixeia**, οι επόμενες προτάσεις έχουν ως αποτέλεσμα την καταχώριση χαρακτήρων όπως φαίνεται στο Σχήμα 13.1β:

```
strcpy(pelatis.onoma, "ΠΕΤΡΟΥ");
strcpy(pelatis.thl, "2651821");
strcpy(filos.onoma, "ΝΙΚΟΣ");
filos.ilikia=21;
```

Οι επόμενες προτάσεις περιμένουν από τον χρήστη να πληκτρολογήσει τη διεύθυνση και την ηλικία του πελάτη και τα αποθηκεύουν στα πεδία **address** και **ilikia** αντίστοιχα.

```
gets(pelatis.address);
scanf("%d", &pelatis.ilikia);
```

Προσέξτε τη χρήση του τελεστή **&**, ο οποίος αποδίδει την διεύθυνση του πεδίου **ilikia** της μεταβλητής δομής **pelatis**.

Οι προτάσεις που ακολουθούν αλλάζουν τον πρώτο χαρακτήρα του πεδίου του ονόματος της μεταβλητής δομής **filos** από **N** σε **Z** (η πρώτη), και το δεύτερο ψηφίο του πεδίου του τηλεφώνου της μεταβλητής δομής **pelatis** από 6 σε 3 (η δεύτερη)!

```
filos.onoma[0]='Z';
pelatis.thl[1]='3';
```

Ο επόμενος κώδικας εμφανίζει στην οθόνη έναν προς έναν τους χαρακτήρες της διεύθυνσης (πεδίο **address**) της μεταβλητής δομής **pelatis**:

```
int t;
for (t=0; pelatis.address[t]!='\0'; t++)
    putchar(pelatis.address[t]);
```

Εμφανίζεται ο **t** χαρακτήρας του πεδίου **address** της μεταβλητής δομής **pelatis**.

Ας μην ξεχνάμε ότι τα πεδία μιας δομής που είναι πίνακες πρέπει να τα χειριζόμαστε ως πίνακες, και τα πεδία που είναι μεταβλητές πρέπει να τα χειριζόμαστε ως μεταβλητές του συγκεκριμένου τύπου. Πιο συγκεκριμένα, στα πεδία που είναι πίνακες χαρακτήρων χρησιμοποιούνται οι συναρτήσεις χειρισμού συμβολοσειρών τόσο για την αποθήκευση όσο και για τον χειρισμό τους.

Ενώσεις

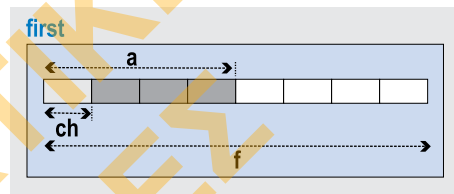
Η **ένωση** (union) στη C είναι ένας χώρος στη μνήμη ο οποίος χρησιμοποιείται (από κοινού) από διαφορετικές μεταβλητές ή πίνακες που μπορεί να είναι ακόμα και διαφορετικού τύπου.

Ο ορισμός μιας **ένωσης** γίνεται όπως και ο ορισμός μιας δομής:

```
union όνομα_τύπου
{
    τύπος όνομα_πεδίου;
    τύπος όνομα_πεδίου;
    .....
} [μεταβλητές, ...];
```

Στο επόμενο παράδειγμα ορίζεται ένας τύπος **union** και μία μεταβλητή αυτού του τύπου με όνομα **first**:

```
union test
{
    int a;
    double f;
    char ch;
} first;
```



Και τα τρία πεδία μοιράζονται **τον ίδιο χώρο μνήμης**, ξεκινώντας από την ίδια διεύθυνση. Η μεταβλητή **first** καταλαμβάνει 8 byte – όσο είναι το μεγαλύτερο σε μέγεθος πεδίο της (το **f** που είναι τύπου **double**). Όμως, μέσω των πεδίων **a** και **ch** μπορούμε να προσπελάσουμε είτε τα 4 πρώτα byte (με το **a**) είτε μόνο το πρώτο byte (με το **ch**).

Με δεδομένη τη μεταβλητή **first**, όπως δηλώθηκε προηγουμένως, ισχύουν τα εξής:

- Με την πρόταση **first.ch='A'** καταχωρίζουμε το χαρακτήρα A στο πρώτο από τα 8 byte της **first**.
- Με την πρόταση **first.a=345** καταχωρίζουμε τον ακέραιο αριθμό 345 στα πρώτα 4 byte της **first**, αντικαθιστώντας τα περιεχόμενα των 4 πρώτων byte.
- Με την πρόταση **first.f=123.3467821** καταχωρίζουμε τον αριθμό διπλής ακρίβειας 123.3467821 στα 8 byte της **first**, αντικαθιστώντας τα περιεχόμενα και των 8 byte.

Απαρίθμησης

Ο τύπος **απαρίθμησης** (enumeration) ορίζει ένα σύνολο από τιμές που μπορεί να πάρει μια μεταβλητή αυτού του τύπου. Ο ορισμός μιας **απαρίθμησης** γίνεται ως εξής:

```
enum όνομα_τύπου {τιμή1, τιμή2, .....,} [μεταβλητές];
```

Για παράδειγμα, η επόμενη πρόταση ορίζει έναν τύπο απαρίθμησης με όνομα **days** και μια λίστα τιμών της απαρίθμησης. Οι τιμές είναι οι **mon**, **tue**, κ.ο.κ.

```
enum days {mon,tue,wed,thu,fri,sat,sun};
```

Παραδείγματα

Π13.1 Έστω ότι έχουμε έναν πίνακα δομών με τη διπλανή μορφή. Στον πίνακα είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Η επόμενη συνάρτηση εντοπίζει τη θέση μνήμης του πίνακα που περιέχει τα στοιχεία του μαθητή με συγκεκριμένο όνομα. Η συνάρτηση επιστρέφει έναν δείκτη στη θέση (δομή) στην οποία εντόπισε το όνομα. Αν το όνομα δεν εντοπιστεί, επιστρέφει τιμή NULL.

```
struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];
```

```
struct stoixeia *find_onom(struct stoixeia math[], char onoma[])
{
    int i;
    for (i=0; i<100; i++)
    {
        if (strcmp(math[i].eponymo, onoma)==0)
            return &math[i];
    }
    return NULL;
}
```

Για κάθε θέση μνήμης του πίνακα, η συνάρτηση **strcmp()** συγκρίνει το πεδίο **eponymo** με το **onoma** που δόθηκε για αναζήτηση.

Αν εντοπιστεί το όνομα τότε επιστρέφει τη διεύθυνση της θέσης στην οποία βρέθηκε το όνομα, διαφορετικά επιστρέφει τιμή NULL.

Η κλήση της συνάρτησης για τον εντοπισμό ενός ονόματος μέσα στον πίνακα **mathites** γίνεται με τον παρακάτω κώδικα:

```
.....
struct stoixeia *ptr;
char lex[30];
.....
printf("Δώσε όνομα για έρευνα:");
gets(lex);
ptr=find_onom(mathites, lex);
if (ptr==NULL)
{
    puts("Το όνομα δεν βρέθηκε");
}
else
{
    puts("Στοιχεία μαθητή");
    puts("=====");
    printf("Επώνυμο: %s\n", ptr->eponymo);
    printf("Τάξη: %s\n", ptr->taxi);
    printf("ΜΟ: %f\n", ptr->mesos_oros);
    printf("Ηλικία: %d\n", ptr->ilikia);
}
```

13_p13_1&2.c

Η συνάρτηση επιστρέφει έναν δείκτη ο οποίος καταχωρίζεται στη μεταβλητή δείκτη **ptr**.

Ασκήσεις Κεφαλαίου 13

- 13.1** Να γραφεί ένα πρόγραμμα το οποίο να καταχωρίζει σε μια μεταβλητή δομής τα στοιχεία ενός αυτοκινήτου (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς τον σωστό τύπο δομής που πρέπει να χρησιμοποιήσετε. ★★
- 13.2** Να γραφεί ένα πρόγραμμα το οποίο να καταχωρίζει σε έναν πίνακα από δομές τα στοιχεία για μέχρι 100 αυτοκίνητα (αριθμός κυκλοφορίας, χρώμα, κατασκευαστής, κυβικά, ιπποδύναμη). Να επιλέξετε εσείς τον σωστό τύπο δομής που πρέπει να χρησιμοποιήσετε. Η διαδικασία καταχώρισης θα πρέπει να διακόπτεται όταν δοθεί κενό για τον αριθμό κυκλοφορίας. ★★★
- 13.3** Έστω ότι έχουμε έναν πίνακα δομών με τη διπλανή μορφή. Στον πίνακα αυτόν είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Να γραφεί μια συνάρτηση η οποία να εμφανίζει τη λίστα των μαθητών. Η κάθε γραμμή να περιέχει το επώνυμο, την τάξη, και τον μέσο όρο του κάθε μαθητή (με ένα δεκαδικό ψηφίο). ★★
- 13.4** Να γραφεί ακόμη μια συνάρτηση, σύμφωνα με τα δεδομένα της προηγούμενης άσκησης, η οποία να επιστρέφει ως τιμή τη μεγαλύτερη από τις ηλικίες των 100 μαθητών. ★★
- 13.5** Εντοπίστε τα λάθη στο επόμενο πρόγραμμα: ★

```
struct stoxeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];
```

```
#include <stdio.h>
enum days {mon,tue,wed,thu,fri,sat,sun} birth_day,today;
typedef int meres;
int main(void)
{
    meres a,b;
    days my_date;
    a=1;
    my_date=mon;
    wed=5;
    meres=23;
    today=3;
    birth_day=fri;
    return 0;
}
```

13.14 Γράψτε ένα πρόγραμμα το οποίο να ορίζει τη δομή **mathima** του διπλανού πλαισίου, ενώ μέσα στη συνάρτηση **main()** θα πρέπει να συμβαίνουν τα εξής: ★★

```
struct mathima
{
    char onomasia[20];
    float bathmos;
    char examino;
};
```

- Θα δηλώνονται δύο μεταβλητές του τύπου δομής **mathima** με ονόματα **m1** και **m2**, και δύο μεταβλητές δείκτη προς δομές **mathima** με ονόματα **p1** και **p2**.
- Η μεταβλητή δείκτη **p1** θα πρέπει να δείχνει στη μεταβλητή δομής **m1**.
- Τα πεδία της μεταβλητής **m1** θα πρέπει να συμπληρωθούν με στοιχεία της επιλογής σας, χωρίς να χρησιμοποιηθεί ο δείκτης **p1**.
- Θα εμφανίζονται στην οθόνη τα περιεχόμενα των πεδίων της μεταβλητής **m1** με χρήση του δείκτη **p1**, χωρίς να χρησιμοποιηθεί καθόλου το όνομα της μεταβλητής **m1**.
- Η μεταβλητή δείκτη **p2** θα πρέπει να δείχνει στη μεταβλητή δομής **m2**.
- Με χρήση του δείκτη **p2**, να συμπληρώνονται τα πεδία της μεταβλητής **m2** με στοιχεία τα οποία θα πληκτρολογεί ο χρήστης. Δεν πρέπει να χρησιμοποιηθεί καθόλου το όνομα της μεταβλητής **m2**.
- Θα εμφανίζονται στην οθόνη τα περιεχόμενα των πεδίων της μεταβλητής **m2** χωρίς να γίνει χρήση του δείκτη **p2**.

13.15 Δίνεται ο ακόλουθος ορισμός δομής: ★★

```
struct mm
{
    unsigned int m1:2;
    int m2:3;
    unsigned int m3:1;
    int m4:4;
    int m5:15;
};
```

- Ποιο είναι το εύρος τιμών που μπορούν να καταχωριστούν στα πεδία **m1**, **m2** και **m3** της παραπάνω δομής;
- Ποιο είναι το συνολικό μέγεθος σε byte μιας μεταβλητής της δομής **mm**;
- Σε μια μεταβλητή της δομής **mm**, υπάρχει χώρος που δεν χρησιμοποιείται; Αν ναι, πόσο είναι το μέγεθος αυτού του χώρου σε bit;

13.16 Υποθέστε ότι θέλουμε να αποθηκεύουμε τα αποτελέσματα που παίρνουμε από μια μετεωρολογική συσκευή η οποία επιστρέφει τις ακόλουθες διαφορετικές α-κέραιες τιμές: ★★

Κεφάλαιο

Ρεύματα επικοινωνίας και χειρισμός αρχείων



Ένα πρόγραμμα μπορεί να επικοινωνεί με τις τυπικές περιφερειακές συσκευές, όπως η οθόνη και το πληκτρολόγιο, με διάφορες άλλες περιφερειακές μονάδες και με αρχεία στις μονάδες περιφερειακής μνήμης.

Στο κεφάλαιο αυτό αναλύεται ο τρόπος με τον οποίο γίνεται αυτή η επικοινωνία στη γλώσσα C, με το μεγαλύτερο βάρος να δίνεται στον χειρισμό αρχείων.

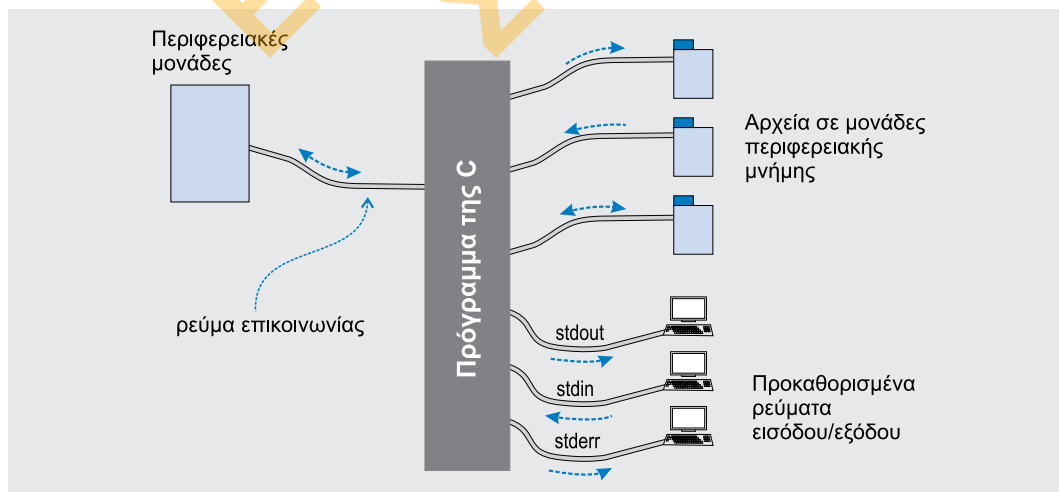
Εξετάζονται τα διαφορετικά είδη αρχείων καθώς και οι διαφορετικοί τρόποι προσπέλασής τους. Παρουσιάζονται οι συναρτήσεις για τον χειρισμό αρχείων κειμένου και δυαδικών αρχείων, τόσο για σειριακή όσο και για τυχαία προσπέλαση.

Ακολουθούν απλά παραδείγματα, καθώς και παραδείγματα που καλύπτουν πολύ εξειδικευμένες απαιτήσεις.

Εισαγωγή

Στη C, η επικοινωνία του προγράμματός μας με τις περιφερειακές συσκευές του H/Y επιτυγχάνεται μέσω των **ρευμάτων** (streams). Ένα ρεύμα αποτελεί μια λογική διασύνδεση του προγράμματος με κάποια από τις συσκευές του H/Y. Στη C, τόσο η είσοδος όσο και η έξοδος υλοποιείται με ακολουθίες από byte. Στην πραγματικότητα, το ρεύμα είναι ένα κανάλι μέσω του οποίου μπορούμε να διαβάσουμε ή να γράψουμε (στέλνουμε) ακολουθίες από byte από ή προς μια περιφερειακή συσκευή (Σχήμα 14.1). Ένα **ρεύμα εισόδου** δέχεται byte από μια περιφερειακή συσκευή, όπως από ένα αρχείο στον δίσκο, από το πληκτρολόγιο, ή από μια άλλη συσκευή εισόδου. Ένα **ρεύμα εξόδου** στέλνει μια σειρά από byte σε μια συσκευή εξόδου, όπως στην οθόνη, σε ένα αρχείο, σε έναν εκτυπωτή κ.λπ. Υπάρχουν επίσης τα **ρεύματα εισόδου/εξόδου**, τα οποία μπορούν και να εισάγουν και να εξάγουν πληροφορίες από και προς τη συσκευή με την οποία είναι συνδεδεμένα (π.χ. αρχείο στον δίσκο, δικτυακή συσκευή, κ.λπ).

Μπορούμε να φανταστούμε τα ρεύματα της C σαν «σωλήνες» μέσω των οποίων το πρόγραμμά μας ανταλλάσσει πληροφορίες με τις περιφερειακές συσκευές του συστήματός μας. Κάποιες τυπικές περιφερειακές συσκευές του H/Y μας, όπως η οθόνη και το πληκτρολόγιο, είναι μόνιμα συνδεδεμένες μέσω τέτοιων ρευμάτων (σωλήνων) τα οποία λέγονται **προκαθορισμένα ρεύματα εισόδου/εξόδου**. Αν όμως θέλουμε να συνδέσουμε κάποια άλλη συσκευή, όπως για παράδειγμα ένα αρχείο στο σκληρό μας δίσκο, θα πρέπει να δημιουργήσουμε ένα νέο ρεύμα (σωλήνα) επικοινωνίας και να το συνδέσουμε με αυτή την περιφερειακή συσκευή. Αυτό γίνεται με την εκτέλεση συγκεκριμένων εντολών του προγράμματός μας. Αντίθετα, με την έναρξη της εκτέλεσης ενός προγράμματος της C, τα προκαθορισμένα ρεύματα εισόδου/εξόδου δημιουργούνται και συνδέονται αυτόματα με τις τυπικές περιφερειακές συσκευές.



Σχήμα 14.1 Ρεύματα εισόδου και εξόδου της C

Προκαθορισμένα ρεύματα

Στη C υπάρχουν τρία προκαθορισμένα ρεύματα τα οποία συνδέουν τις τυπικές (*standard*) συσκευές ενός Η/Υ. Τα ρεύματα αυτά ορίζονται στο αρχείο κεφαλίδας stdio.h και είναι τα εξής:

stdout	Τυπική έξοδος (οθόνη)
stdin	Τυπική είσοδος (πληκτρολόγιο)
stderr	Τυπική έξοδος για μηνύματα σφαλμάτων (οθόνη)

Ένα πρόγραμμα της C χρησιμοποιεί ως προεπιλεγμένη (*default*) συσκευή εξόδου την οθόνη και ως προεπιλεγμένη συσκευή εισόδου το πληκτρολόγιο²³.

Εκτός από τα προκαθορισμένα ρεύματα εισόδου/εξόδου, τα ρεύματα που θα δημιουργούμε στα προγράμματά μας μπορούν να συνδεθούν με οποιαδήποτε περιφερειακή μονάδα, η οποία μπορεί να είναι ένα αρχείο στον δίσκο μας, ο εκτυπωτής μας, μια δικτυακή συσκευή, κ.λπ.

Επίπεδα ρευμάτων

Η C διαχειρίζεται τα ρεύματα σε δύο επίπεδα: Σε **χαμηλό επίπεδο** (low-level) και σε **υψηλό επίπεδο** (high level). Κατά τον χειρισμό σε χαμηλό επίπεδο, το ρεύμα αντιμετωπίζεται ως μια απλή σειρά από byte, και η διαχείρισή του περιορίζεται στην εισαγωγή ή εξαγωγή συγκεκριμένου πλήθους byte από ή προς το συγκεκριμένο ρεύμα. Κατά τον χειρισμό σε υψηλό επίπεδο, ο οποίος ονομάζεται επίσης «μορφοποιημένη είσοδος/έξοδος» (formatted I/O), τα byte ενός ρεύματος αντιμετωπίζονται ως χαρακτήρες.

Τα ρεύματα χαμηλού επιπέδου αποκαλούνται **δυναδικά ρεύματα** (binary streams), ενώ τα υψηλού επιπέδου αποκαλούνται **ρεύματα χαρακτήρων ή κειμένου** (text streams).

Τα προκαθορισμένα ρεύματα **stdout**, **stdin**, και **stderr**, είναι ρεύματα χαρακτήρων. Η διαφορά μεταξύ δυναδικών ρευμάτων και ρευμάτων χαρακτήρων θα γίνει πλήρως κατανοητή στη συνέχεια του κεφαλαίου.

Στη C όλα τα ρεύματα αντιμετωπίζονται με τον ίδιο τρόπο, ανεξάρτητα από τη φυσική συσκευή του συστήματός μας με την οποία είναι συνδεδεμένα. Επομένως, με τον ίδιο τρόπο με τον οποίο διαβάζουμε χαρακτήρες από ένα αρχείο, μπορούμε να διαβάζουμε και από το πληκτρολόγιο ή από μια σειριακή θύρα.

²³ Οι προκαθορισμένες συσκευές εξόδου και εισόδου του Η/Υ μπορούν να ανακατευθυνθούν από το λειτουργικό σύστημα, και να μην είναι πλέον η οθόνη και το πληκτρολόγιο.

Χειρισμός αρχείων

Ένα αρχείο δεν είναι τίποτα περισσότερο από μια σειρά από byte αποθηκευμένα σε μια μονάδα περιφερειακής μνήμης (σκληρό δίσκο, δισκέτα, CD, κ.λπ.). Για τον χειρισμό ενός αρχείου μέσα από ένα πρόγραμμα της C εκτελούνται δύο ενέργειες: η εγγραφή πληροφοριών στο αρχείο και η ανάγνωση πληροφοριών από αυτό.

Αρχεία κειμένου και δυαδικά αρχεία

Η εγγραφή πληροφοριών σε ένα αρχείο μπορεί να γίνει με δύο τρόπους:

- Οι πληροφορίες γράφονται σε μορφή χαρακτήρων, όπως δηλαδή θα εμφανίζονταν στην οθόνη.
- Οι πληροφορίες γράφονται με την εσωτερική αναπαράστασή τους, δηλαδή γράφονται τα byte στα οποία έχει μετατραπεί η συγκεκριμένη πληροφορία.

Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να γράψουμε τον ακέραιο αριθμό 12 σε ένα αρχείο. Στην πρώτη περίπτωση θα γραφούν δύο χαρακτήρες, το 1 και το 2, δηλαδή συνολικά δύο byte. Στη δεύτερη περίπτωση, όμως, θα γραφεί η εσωτερική αναπαράσταση του αριθμού 12, δηλαδή τα τέσσερα byte που απαρτίζουν οποιονδήποτε ακέραιο αριθμό. Ακόμη και αν ο αριθμός ήταν ο 123456789, πάλι θα καταλάμβανε 4 byte σε αυτόν τον τύπο αρχείου.

Στην πρώτη περίπτωση, το αρχείο αποκαλείται αρχείο κειμένου (text file), ενώ στη δεύτερη δυαδικό αρχείο (binary file).

Επίσης, όταν η C γράφει σε ένα αρχείο κειμένου, ο χαρακτήρας αλλαγής γραμμής (new line – '\n') μετατρέπεται σε δύο χαρακτήρες: στον χαρακτήρα προώθησης γραμμής (line feed με κωδικό ASCII 10) και στον χαρακτήρα επαναφοράς κεφαλής (carriage return με κωδικό ASCII 13).

Το αντίθετο γίνεται στην περίπτωση που η C διαβάζει από ένα αρχείο κειμένου. Όταν εντοπίσει τους δύο χαρακτήρες αλλαγής γραμμής (τον χαρακτήρα ASCII 10 και τον χαρακτήρα ASCII 13), τους μετατρέπει σε έναν χαρακτήρα, τον χαρακτήρα αλλαγής γραμμής ('\n'). Στην περίπτωση, όμως, που η C χειρίζεται ένα δυαδικό αρχείο, δεν γίνεται καμία τέτοια μετατροπή.

Για να μπορέσουμε να γράψουμε ή να διαβάσουμε πληροφορίες από κάποιο αρχείο, θα πρέπει πρώτα να το συνδέσουμε με κάποιο ρεύμα (κανάλι) εισόδου/εξόδου.

Η επικοινωνία του προγράμματός μας με ένα αρχείο κειμένου επιτυγχάνεται μέσω ενός ρεύματος χαρακτήρων, ενώ με ένα δυαδικό αρχείο η επικοινωνία επιτυγχάνεται μέσω ενός δυαδικού ρεύματος.

Σειριακή και τυχαία προσπέλαση

Φανταστείτε ότι θέλουμε να γράψουμε σε ένα τετράδιο ονόματα και κωδικούς (αριθμούς μητρώου) διάφορων φοιτητών. Παράλληλα θέλουμε να εκμεταλλευτούμε όλο τον διαθέσιμο χώρο των σελίδων του τετραδίου, ώστε σε κάθε σελίδα να μπορούμε να καταχωρίσουμε όσο το δυνατόν περισσότερους φοιτητές.

Έτσι, σε κάθε γραμμή του τετραδίου μας θα γράφουμε τα ονόματα και τους κωδικούς των φοιτητών, τον έναν μετά τον άλλον, μέχρι να γεμίσει η γραμμή, μετά θα συνεχίζουμε στην επόμενη γραμμή, κ.ο.κ (Σχήμα 14.2α). Με αυτόν τον τρόπο εκμεταλλευόμαστε όλο τον διαθέσιμο χώρο της κάθε γραμμής. Κάθε εγγραφή στο τετράδιο αποτελείται από τα πεδία του ονόματος και του κωδικού. Από τη στιγμή που κάθε όνομα και κάθε κωδικός μπορεί να αποτελείται από διαφορετικό πλήθος χαρακτήρων, κάθε εγγραφή έχει διαφορετικό μέγεθος. Το κενό διάστημα ανάμεσα στα πεδία χρησιμοποιείται για τον διαχωρισμό τους.

Με αυτόν τον τρόπο, όμως, για να μπορέσουμε να εντοπίσουμε μια εγγραφή μέσα στο τετράδιό μας (π.χ. την ενδέκατη) θα πρέπει να διαβάσουμε πρώτα όλες τις προηγούμενες (στην περίπτωση μας, τις προηγούμενες δέκα), ξεκινώντας από την αρχή του τετραδίου. Αυτό είναι αναγκαίο επειδή απλούστατα δεν γνωρίζουμε σε ποιο σημείο (π.χ. σε ποια γραμμή) βρίσκεται η ενδέκατη εγγραφή.

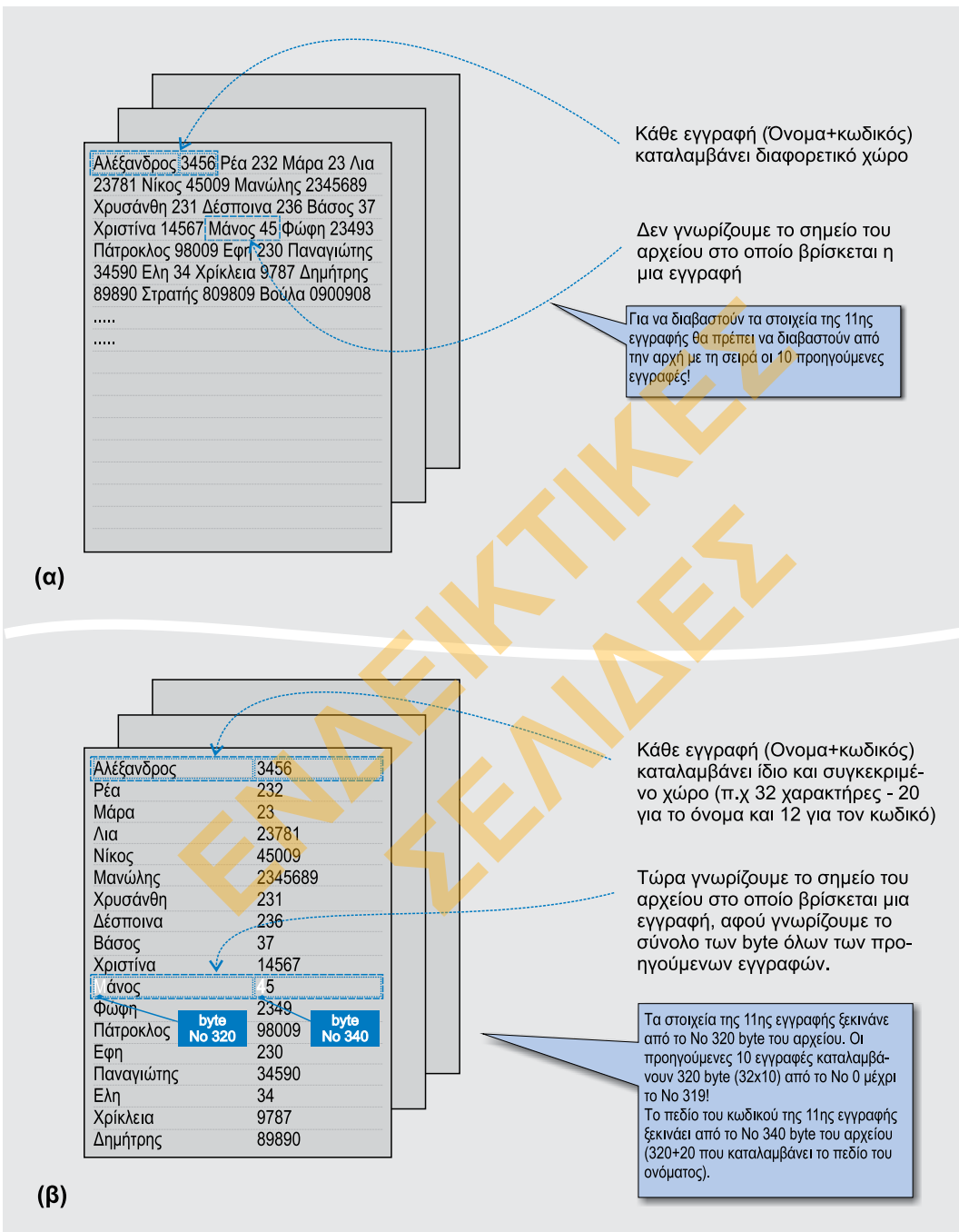
Αυτός ο τρόπος προσπέλασης (εντοπισμού) μιας εγγραφής λέγεται **σειριακός**, διότι πρέπει πάντα να ξεκινάμε την ανάγνωση από την αρχή και να διαβάζουμε με τη σειρά μία προς μία τις εγγραφές μέχρι να φτάσουμε σε αυτή που μας ενδιαφέρει.

Τώρα αποφασίζουμε ότι σε κάθε γραμμή του τετραδίου μας θα γράφουμε μία μόνο εγγραφή (με την παραδοχή ότι η κάθε γραμμή χωράει 32 χαρακτήρες), αφιερώνοντας συγκεκριμένο χώρο για το όνομα (έστω 20 χαρακτήρες) και συγκεκριμένο χώρο για τον κωδικό (π.χ. τους υπόλοιπους 12 χαρακτήρες), όπως παρουσιάζεται στο Σχήμα 14.2β.

Με αυτόν τον τρόπο, ο εντοπισμός μιας εγγραφής είναι πολύ εύκολη υπόθεση. Αν υποθέσουμε ότι κάθε σελίδα του τετραδίου μας διαθέτει 17 γραμμές, η 11η εγγραφή σίγουρα βρίσκεται στην 11η γραμμή της πρώτης σελίδας ενώ η 170ή εγγραφή βρίσκεται στη τελευταία γραμμή της 10ης σελίδας. Και στις δύο περιπτώσεις, οι εγγραφές μπορούν να εντοπιστούν άμεσα χωρίς να χρειάζεται να διαβαστεί καμία από τις προηγούμενες εγγραφές.

Αυτός ο τρόπος προσπέλασης μιας εγγραφής λέγεται **τυχαίος**, διότι μπορούμε να διαβάσουμε απευθείας όποια εγγραφή θέλουμε.

Όπως παρατηρούμε, στην περίπτωση που το όνομα ή ο κωδικός έχουν λιγότερους από 20 και 12 χαρακτήρες αντίστοιχα, χάνουμε πολύτιμο χώρο από το τετράδιο μας.



Σχήμα 14.2 Παράδειγμα σειριακής και τυχαίας προσπέλασης

Αρχεία κειμένου

Τα αρχεία κειμένου (text files) είναι αρχεία τα οποία έχουν ανοίξει σε κατάσταση κειμένου με τη συνάρτηση `fopen()`. Στα αρχεία κειμένου οι πληροφορίες γράφονται σε μορφή χαρακτήρων, όπως δηλαδή εμφανίζονται στην οθόνη. Η εγγραφή και η ανάγνωση πληροφοριών σε αρχεία κειμένου γίνεται με σειριακό τρόπο, μέσω των συναρτήσεων του πίνακα που ακολουθεί.

Συνάρτηση	Λειτουργία
<code>fputc()</code>	Γράφει έναν χαρακτήρα σε ένα αρχείο
<code>fgetc()</code>	Διαβάζει έναν χαρακτήρα από κάποιο αρχείο
<code>fprintf()</code>	Πραγματοποιεί εγγραφή σε ένα αρχείο, όπως η <code>printf</code> στην οθόνη
<code>fscanf()</code>	Πραγματοποιεί ανάγνωση από ένα αρχείο, όπως η <code>scanf</code> από το πληκτρολόγιο
<code>fgets()</code>	Διαβάζει ένα σύνολο χαρακτήρων (αλφαριθμητικό) από ένα αρχείο
<code>fputs()</code>	Γράφει ένα σύνολο χαρακτήρων (αλφαριθμητικό) σε ένα αρχείο

Κάθε μια από τις παραπάνω συναρτήσεις διαθέτει και μια παράμετρο (`fp`), έναν δείκτη τύπου `FILE`, στην οποία μεταβιβάζεται ο δείκτης του ρεύματος το οποίο είναι συνδεδεμένο με το συγκεκριμένο αρχείο.

fputc()

Η συνάρτηση `fputc()` γράφει έναν χαρακτήρα μέσα σε ένα αρχείο. Το πρωτότυπό της είναι:

int fputc(int *ch*, FILE **fp*);

όπου *ch* είναι ο χαρακτήρας που θα γραφεί στο αρχείο. Η συνάρτηση, στην περίπτωση που η εγγραφή είναι επιτυχής, επιστρέφει τον κωδικό ASCII του χαρακτήρα που έγραψε, διαφορετικά επιστρέφει τον χαρακτήρα EOF. Ο δείκτης *fp* προσδιορίζει το ρεύμα που είναι συνδεδεμένο με το αρχείο.

Το πρόγραμμα που ακολουθεί γράφει σε ένα αρχείο κειμένου με όνομα **text.txt** τους κεφαλαίους λατινικούς χαρακτήρες (από το A μέχρι το Z). Αν το αρχείο υπάρχει ήδη, αντικαθίστανται όλες οι πληροφορίες που περιέχει, διαφορετικά το αρχείο δημιουργείται.

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

14_write_text_file.c

```
int main(void)
```

```
{
    FILE *fp;
    char ch;
    fp=fopen("text.txt","w");
    if (fp==NULL) exit(2);
    for (ch='A';ch<='Z';ch++) fputc(ch,fp);
    fclose(fp);
    return 0;
}
```

Ανοίγει το αρχείο **text.txt** ως αρχείο κειμένου για εγγραφή.

Γράφει στο αρχείο τους λατινικούς χαρακτήρες από το 'A' μέχρι το 'Z'.

Κλείνει το αρχείο.

fgetc()

Η συνάρτηση **fgetc()** διαβάζει έναν χαρακτήρα από ένα αρχείο. Το πρωτότυπό της είναι:

```
int fgetc(FILE *fp);
```

Ο δείκτης **fp** προσδιορίζει το ρεύμα που είναι συνδεδεμένο με το αρχείο. Η συνάρτηση επιστρέφει ως τιμή τον κωδικό ASCII του χαρακτήρα που διάβασε. Αν φτάσει στο τέλος του αρχείου, η συνάρτηση επιστρέφει τον χαρακτήρα «τέλους αρχείου» (end-of-file – EOF). Η τιμή του χαρακτήρα «τέλους αρχείου» ορίζεται στο αναγνωριστικό **EOF** στο αρχείο κεφαλίδας **stdio.h**. Όταν διαβάζουμε ένα αρχείο κειμένου χαρακτήρα προς χαρακτήρα, ελέγχουμε πάντα για τον χαρακτήρα EOF ο οποίος σηματοδοτεί το τέλος του αρχείου. Για παράδειγμα, το επόμενο πρόγραμμα διαβάζει έναν προς έναν τους χαρακτήρες του αρχείου **text.txt** και τους εμφανίζει στην οθόνη.

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

```
int main(void)
```

```
{
```

```
    FILE *fp;
```

```
    char ch=' ';
```

```
    fp=fopen("text.txt","r");
```

```
    if (fp==NULL) exit(2);
```

```
    while (ch!=EOF)
```

```
    {
```

```
        ch=fgetc(fp);
```

```
        if (ch!=EOF) putchar(ch);
```

```
    }
```

```
    fclose(fp);
```

```
    return 0;
```

```
}
```

14_read_text_file.c

Ανοίγει το αρχείο **text.txt** ως αρχείο κειμένου για ανάγνωση.

Διαβάζει έναν προς έναν τους χαρακτήρες από το αρχείο μέχρι να διαβάσει τον χαρακτήρα EOF.

Εμφανίζει τον χαρακτήρα στην οθόνη (στην περίπτωση που δεν είναι ο EOF).

Κλείνει το αρχείο.

fprintf()

Η συνάρτηση **fprintf()** συμπεριφέρεται όπως η **printf()**, με τη διαφορά ότι, αντί να εμφανίσει τα αποτελέσματά της στην οθόνη, τα γράφει σε ένα αρχείο. Το πρωτότυπό της, παρόμοιο με της **printf()**, είναι το ακόλουθο:

```
int fprintf(FILE *fp, const char *αλφ_μορφ, παρ1, παρ2, ...);
```

Τα αποτελέσματα της **fprintf()** γράφονται στο αρχείο που καθορίζεται από τον δείκτη **fp**. Η **fprintf()** επιστρέφει το πλήθος των χαρακτήρων που έγραψε στο αρχείο.

Το πρόγραμμα που ακολουθεί ανοίγει ένα αρχείο κειμένου με όνομα **tyxaioi.txt** για εγγραφή και στη συνέχεια γράφει μέσα στο αρχείο 100 τυχαίους αριθμούς.

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

```
int main(void)
```

```
{
```

```
    FILE *fp;
```

14_write_randoms.c

Τυχαία προσπέλαση

Όπως έχει ήδη αναφερθεί, με την τυχαία προσπέλαση μπορούμε να μετακινούμε τον δείκτη θέσης σε όποιο σημείο του αρχείου θέλουμε, και να αρχίσουμε να διαβάζουμε ή να γράφουμε από αυτό το σημείο.

Η C διαθέτει τρεις συναρτήσεις για τον χειρισμό του δείκτη θέσης ενός αρχείου, τις **fseek()**, **rewind()** και **ftell()**.

fseek()

Η συνάρτηση **fseek()** τοποθετεί τον δείκτη σε κάποιο σημείο του αρχείου. Το πρωτότυπο της είναι:

int fseek(FILE *fp, long int apostasi, int thesi);

όπου ο **fp** καθορίζει το αρχείο και η παράμετρος **apostasi** δηλώνει τη σχετική απόσταση (σε byte) από τη θέση που καθορίζει η παράμετρος **thesi**. Η παράμετρος **thesi** μπορεί να πάρει τις παρακάτω τιμές:

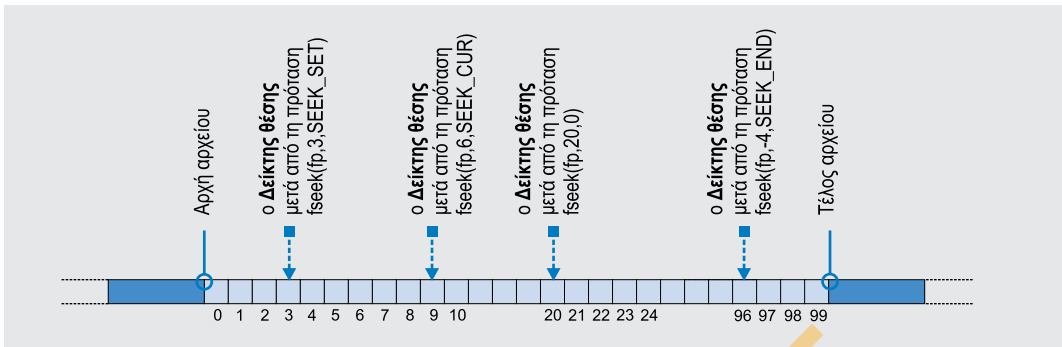
Τιμή παραμέτρου	Θέση
0	SEEK_SET αρχή αρχείου
1	SEEK_CUR τρέχουσα θέση
2	SEEK_END τέλος αρχείου

Τα αναγνωριστικά **SEEK_SET**, **SEEK_CUR**, και **SEEK_END** ορίζονται στο αρχείο κεφαλίδας **stdio.h**, αλλά μπορούν να χρησιμοποιηθούν και οι τιμές 0, 1, και 2 αντίστοιχα. Η συνάρτηση επιστρέφει τιμή 0 όταν εκτελεστεί επιτυχώς, διαφορετικά επιστρέφει μια μη μηδενική τιμή.

Όταν ως θέση αναφοράς ορίζεται το τέλος του αρχείου, τότε κατά την ανάγνωση του αρχείου στην παράμετρο **apostasi** έχουν νόημα μόνο οι αρνητικές τιμές. Όταν όμως γράφουμε στο αρχείο, μια θετική τιμή έχει ως αποτέλεσμα την εγγραφή των δεδομένων σε κάποια απόσταση μετά από το τέλος του αρχείου με ταυτόχρονη αύξηση του μεγέθους του.

Η συνάρτηση **fseek()**, σε συνδυασμό με τις συναρτήσεις **fread()** και **fwrite()**, δίνουν στη C τη δυνατότητα τυχαίας προσπέλασης ενός αρχείου.

Στο παράδειγμα του Σχήματος 14.6 ο δείκτης θέσης μετακινείται με χρήση της συνάρτησης **fseek()**, η οποία καλείται με διαφορετικά ορίσματα κάθε φορά. Υποθέτουμε ότι ο δείκτης **fp** αφορά στο ρεύμα επικοινωνίας που είναι συνδεδεμένο με το συγκεκριμένο αρχείο. Η πρώτη κλήση της **fseek()** μετακινεί τον δείκτη θέσης στο byte 3 του αρχείου, η δεύτερη κλήση της το μετακινεί 6 θέσεις από την τρέχουσα (οπότε μετακινείται στο byte 3+6=9), η επόμενη κλήση τον τοποθετεί στη θέση 20 και η τελευταία τον τοποθετεί στο τέταρτο byte πριν από το τέλος του αρχείου.



Σχήμα 14.6 Μετακίνηση του δείκτη θέσης με χρήση της συνάρτησης `fseek()`

rewind()

Η συνάρτηση `rewind()` τοποθετεί τον δείκτη θέσης στην αρχή του αρχείου. Η συνάρτηση έχει το ακόλουθο πρωτότυπο:

```
viod rewind(FILE *fp);
```

ftell()

Η συνάρτηση `ftell()` επιστρέφει την τιμή του δείκτη θέσης ενός αρχείου. Το πρωτότυπό της είναι:

```
long int fseek(FILE *fp);
```

όπου η παράμετρος `fp` καθορίζει το ρεύμα επικοινωνίας που είναι συνδεδεμένο με το συγκεκριμένο αρχείο.

Το πρόγραμμα που ακολουθεί ανοίγει ένα αρχείο κειμένου για ανάγνωση (κατάσταση "r") και διαβάζει χαρακτήρες από διαφορετικά σημεία του αρχείου μετακινώντας κατάλληλα τον δείκτη θέσης.

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

14_char_rand1.c

```
int main(void)
```

```
{
    FILE *fp;
    char ch;
    fp=fopen("ch1.txt","r");
    if (fp ==NULL ) exit(1);
    ch=fgetc(fp);           // Διαβάζει τον πρώτο χαρακτήρα
    printf("%c\n",ch);
    fseek(fp,4,SEEK_SET);   // Τοποθετεί τον δείκτη θέσης στο byte 4 του αρχείου
    ch=fgetc(fp);           // Διαβάζει τον χαρακτήρα 'E' (είναι ο 5ος χαρακτήρας)
    printf("%c\n",ch);
    fseek(fp,3,SEEK_CUR);   // Τοποθετεί τον δείκτη θέσης 3 θέσεις μετά την τρέχουσα
```

ch1.txt

ΑΒΓΔΕΖ
ΗΘΙΚΛ
ΜΝΞΟ
ΠΡΣ

Ε
Η
Ρ
Γ

Παραδείγματα

Π14.1 Το επόμενο πρόγραμμα ζητάει λέξεις και τις καταχωρίζει σε ένα αρχείο κειμένου με όνομα **lexeis.txt**. Το πρόγραμμα σταματάει όταν δοθεί ως λέξη το κενό.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char lex[40];
    FILE *fp;
    fp=fopen("lexeis.txt","w");
    while (1)
    {
        printf("Δώσε λέξη:");
        gets(lex);
        if (strcmp(lex,"")==0)
            break;
        fputs(lex,fp); // Γράφει τη λέξη στο αρχείο
        fputc('\n',fp); // Επιβάλλει αλλαγή γραμμής στο αρχείο
    }
    fclose(fp);
    return 0;
}
```

14_p14_1.c

Αν αντί για λέξη πληκτρολογήσουμε ένα κενό (δηλαδή πατήσουμε απλώς το <ENTER>), η επαναληπτική διαδικασία διακόπτεται.

Π14.2 Υποθέτουμε ότι το αρχείο κειμένου **data1.txt** περιέχει ένα άγνωστο πλήθος ακέραιων αριθμών. Το παρακάτω πρόγραμμα βρίσκει τον μικρότερο και τον μεγαλύτερο αριθμό του αρχείου.

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

int main(void)
{
    int ar,min,max,pl;
    FILE *fp;
    fp=fopen("data1.txt","r");
    pl=0;
    while (1)
    {
        fscanf(fp,"%d",&ar);
        if (feof(fp)) break;
        pl++;
        if (pl==1)
            if (pl==1) min=max=ar;
        if (ar>max) max=ar;
        if (ar<min) min=ar;
    }
}
```

14_p14_2.c

Ανοίγει το αρχείο **data1.txt** για ανάγνωση. Ο χειρισμός του αρχείου γίνεται μέσω του δείκτη **fp**.

Διαβάζουμε έναν προς έναν όλους τους αριθμούς.

Όταν διαβάσουμε τον πρώτο αριθμό (**pl==1**), τον εκχωρούμε ως αρχική τιμή στις μεταβλητές **min** και **max**.

Αν ο αριθμός είναι μεγαλύτερος από το **max**, τότε καταχωρίζουμε στη **max** αυτόν τον αριθμό. Αν είναι μικρότερος από το **min**, τότε τον καταχωρίζουμε στη **min**.


```

fclose(fp);
printf("min=%d max=%d\n",min,max);
return 0;
}

```

Ο χειρισμός του αρχείου γίνεται μέσω του δείκτη `fp`. Η μεταβλητή `pl` χρησιμοποιείται για την καταμέτρηση των αριθμών που διαβάζονται. Στις μεταβλητές `min` και `max` τίθεται ως αρχική τιμή ο πρώτος αριθμός που διαβάζεται. Η επαναληπτική διαδικασία τερματίζεται όταν φτάσουμε στο τέλος του αρχείου και η συνάρτηση `fEOF()` επιστρέφει τιμή `true`.

Π14.3 Έστω ότι έχουμε έναν πίνακα δομών με τη διπλανή μορφή. Στον πίνακα αυτό είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Το επόμενο τμήμα κώδικα αποθηκεύει τα στοιχεία του πίνακα σε ένα αρχείο με όνομα `sxoleio`.

```

.....
FILE *fp;
fp=fopen("sxoleio","wb");
fwrite(mathites,sizeof(struct stoixeia),100,fp);
fclose(fp);

```

```

struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];

```

Ας εξετάσουμε προσεκτικά την κλήση της `fwrite()`:

```

fwrite(mathites,sizeof(struct stoixeia),100,fp);

```

- Η παράμετρος `mathites` είναι ένας δείκτης στην αρχή του πίνακα δομών που περιέχει τα δεδομένα μας. Από αυτή τη θέση μνήμης θα αρχίσει η ανάγνωση των προς εγγραφή δεδομένων.
- Η παράσταση `sizeof(struct stoixeia)` αποδίδει το μέγεθος σε byte του τύπου δεδομένων `stoixeia`. Στο συγκεκριμένο παράδειγμα, το μέγεθος σε byte του τύπου `stoixeia` είναι 43 byte (30+5+4+4).
- Ο αριθμός `100` υποδεικνύει το πλήθος των τμημάτων μνήμης μεγέθους 43 byte που θα γραφούν στο αρχείο.
- Η παράμετρος `fp` προσδιορίζει το αρχείο.

Π14.4 Έστω ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε από το προηγούμενο παράδειγμα, και το οποίο περιέχει τα στοιχεία των 100 μαθητών.

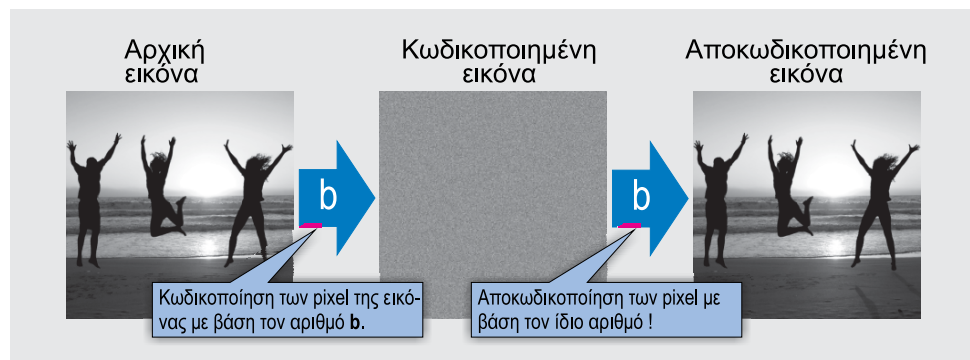
Το επόμενο πρόγραμμα διαβάζει από το αρχείο τα στοιχεία των μαθητών και τα καταχωρίζει στον πίνακα `mathites`, τύπου δομής `stoixeia`. Το πρόγραμμα στο τέλος εμφανίζει τα ονόματα των 100 μαθητών που διάβασε από το αρχείο.

Π14.9 Το παράδειγμα αυτό κωδικοποιεί και αποκωδικοποιεί ένα αρχείο εικόνας τύπου BMP. Θα πρέπει να έχουμε κατανοήσει πλήρως τον τρόπο με τον οποίο αποθηκεύεται μια εικόνα αυτού του τύπου, όπως αναλύθηκε διεξοδικά στο Παράδειγμα Π14.8, επειδή ο κώδικας του παρόντος παραδείγματος βασίζεται σε αυτό. Το ζητούμενο του παραδείγματος είναι να μπορέσουμε να κωδικοποιήσουμε τα pixel μιας εικόνας και να παραγάγουμε μια άλλη εικόνα η οποία δεν θα έχει καμία σχέση με την αρχική. Με το ίδιο πρόγραμμα θα πρέπει να μπορούμε να αποκωδικοποιήσουμε αυτή την κωδικοποιημένη εικόνα ώστε να προκύψει πάλι η αρχική. Τόσο η κωδικοποίηση όσο και η αποκωδικοποίηση θα πρέπει να βασίζονται στον ίδιο κωδικό αριθμό που δίνει ο χρήστης (Σχήμα 14.10).

Η τεχνική που θα χρησιμοποιήσουμε βασίζεται στη χρήση του δυαδικού τελεστή XOR (^). Εκμεταλλευόμαστε μια ιδιότητα του τελεστή η οποία γίνεται κατανοητή από το ακόλουθο παράδειγμα: Αν το αποτέλεσμα της πράξης $a \wedge b$ είναι c , το αποτέλεσμα της πράξης $c \wedge b$ είναι πάλι a .

Ας υποθέσουμε ότι θέλουμε να στέλνουμε σε κάποιον φίλο μας αριθμούς κινητών τηλεφώνων χωρίς όμως να υπάρχει η δυνατότητα υποκλοπής τους. Θα πρέπει επομένως να τους κωδικοποιήσουμε. Έστω ότι θέλουμε να στείλουμε τον αριθμό 6937068069. Επιλέγουμε ως κωδικό κωδικοποίησης/αποκωδικοποίησης κάτι που θα το θυμόμαστε εύκολα, για παράδειγμα τον αριθμό 123456789 τον οποίο γνωρίζουμε μόνο εμείς και ο φίλος μας. Αν κάνουμε τη πράξη XOR $6937068069 \wedge 123456789$ προκύπτει ο αριθμός 6880792368 τον οποίο στέλνουμε στον φίλο μας. Ο αριθμός αυτός, ακόμα και αν υποκλαπεί, δεν έχει καμία σχέση με τον αρχικό αριθμό τηλεφώνου. Αν τώρα ο φίλος μας εκτελέσει την ίδια πράξη XOR με τον αριθμό που του στείλαμε και τον κωδικό κωδικοποίησης, δηλαδή $6880792368 \wedge 123456789$, θα προκύψει ο αρχικός αριθμός 6937068069!

Αν κωδικοποιήσουμε κάθε χρώμα του κάθε pixel της εικόνας με τον παραπάνω τρόπο, θα προκύψει μια εικόνα με διαφορετικά χρώματα αλλά κατά βάση ίδια με την αρχική. Αυτό είναι πρόβλημα το οποίο θα λύσουμε με έναν έξυπνο τρόπο!



Σχήμα 14.10 Κωδικοποίηση/αποκωδικοποίηση εικόνας

Θα κωδικοποιήσουμε το χρώμα κάθε pixel της εικόνας με βάση διαφορετικό αριθμό! Εδώ εκμεταλλευόμαστε τη συνάρτηση `rand()` η οποία αποδίδει ψευδο-τυχαίους αριθμούς. Η ακολουθία των αριθμών που επιστρέφει η συνάρτηση είναι πάντα η ίδια, δηλαδή πάντα πρώτος θα είναι ο 41, δεύτερος ο 18477, τρίτος ο 6334, κ.ο.κ. Η ακολουθία αυτή αλλάζει μόνο αν ορίσουμε κάποιον διαφορετικό αριθμό βάσης, με χρήση της συνάρτησης `srand()`. Όμως ακόμα και σε αυτή την περίπτωση, η ακολουθία των αριθμών θα είναι μεν διαφορετική αλλά σταθερή. Για παράδειγμα, μετά από μια πρόταση `srand(10)` η ακολουθία των αριθμών που επιστρέφει η `rand()` θα είναι πάντα: 71, 16899, 3272, κ.ο.κ. Έτσι, αντί να έχουμε έναν κρυφό κωδικό για κάθε χρώμα καθενός pixel, πράγμα που θα ήταν αδύνατο, θα έχουμε έναν μόνο κρυφό κωδικό κωδικοποίησης/αποκωδικοποίησης ο οποίος θα χρησιμοποιείται ως βάση για τη γεννήτρια τυχαίων αριθμών, δηλαδή ως όρισμα της `srand()`. Αν υποθέσουμε ότι ο κωδικός κωδικοποίησης/αποκωδικοποίησης που δίνει ο χρήστης είναι ο `ar`, τότε η πρόταση `srand(ar)` δημιουργεί μια συγκεκριμένη σειρά ψευδοτυχαίων αριθμών που επιστρέφει η συνάρτηση `rand()`.

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

14_p14_9.c

```
struct pixel
{
    unsigned char blue;
    unsigned char green;
    unsigned char red;
};
```

```
int main(void)
{
    FILE *fp1,*fp2;
    struct pixel p;
    char onin[30],onout[30];
    int platos,ypsos,offset,g,size,bytes_per_raw,padding;
    int s,count=0,i,cnt,ar;
    short int bits;
    unsigned char ch;
    printf("Δώσε κωδικό κωδικοποίησης/αποκωδικοποίησης -> ");
    scanf("%d",&ar);
    fflush(stdin);
    srand(ar);
    printf("Δώσε όνομα αρχείου εισόδου χωρίς την προέκταση .bmp -> ");
    gets(onin);
    strcat(onin,".bmp");
    printf("Δώσε όνομα αρχείου εξόδου χωρίς την προέκταση .bmp -> ");
    gets(onout);
```

Ο κωδικός που δίνει ο χρήστης χρησιμοποιείται ως βάση για τη γεννήτρια τυχαίων αριθμών της `rand()`.

```

strcat(onout, ".bmp");
fp1=fopen(onin, "rb");
fp2=fopen(onout, "wb");
// Τμήμα κώδικα το οποίο διαβάζει και εμφανίζει τα στοιχεία της κεφαλίδας της εικόνας
// .....
// Ο κώδικας έχει παραλειφθεί διότι είναι ίδιος με τον αντίστοιχο του Παραδείγματος 14.8
// .....
// Το παρακάτω τμήμα επεξεργάζεται τις τιμές των pixel της εικόνας
fseek(fp1, 0, SEEK_SET); // Μετακινεί τον δείκτη θέσης στην αρχή του αρχείου εισόδου
for (i=0; i<offset; i++)
{
    fread(&p, 1, 1, fp1);
    fwrite(&p, 1, 1, fp2);
}
fseek(fp1, offset, SEEK_SET);
fseek(fp2, offset, SEEK_SET);
for (g=0; g<yposos; g++)
{
    for (s=0; s<platos; s++)
    {
        count++;
        fread(&p, 3, 1, fp1);
        p.red=p.red^(rand()%256);
        p.green=p.green^(rand()%256);
        p.blue=p.blue^(rand()%256);
        fwrite(&p, 3, 1, fp2);
    }
    fseek(fp1, padding, SEEK_CUR);
    ch=0;
    for (i=1; i<=padding; i++) fwrite(&ch, 1, 1, fp2);
    putchar('#');
}
fclose(fp1);
fclose(fp2);
printf("\nΗ κωδικοποιημένη/αποκωδικοποιημένη εικόνα βρίσκεται στο
        αρχείο -> %s", onout);
return 0;
}

```

Αντιγράφει όλα τα περιεχόμενα της κεφαλίδας της εικόνας του αρχείου εισόδου στο αρχείο εξόδου. Δηλαδή, αντιγράφει ένα προς ένα όλα τα byte μέχρι το **offset**, μετά από το οποίο αρχίζουν τα δεδομένα των pixel της εικόνας.

Μετακινεί τον δείκτη θέσης στο σημείο που αρχίζουν τα δεδομένα των pixel και στα δύο αρχεία.

Διαβάζει τα δεδομένα ενός pixel και τα καταχωρίζει στη μεταβλητή δομής **p**.

Κωδικοποιεί κάθε χρώμα του pixel με χρήση του δυαδικού τελεστή XOR. Η πράξη γίνεται με έναν τυχαίο ακέραιο μεταξύ 0 και 255!

Προσπερνά τα byte συμπλήρωσης της εικόνας εισόδου.

Γράφει τόσα μηδενικά byte συμπλήρωσης στο αρχείο της εικόνας εξόδου, όσα υπήρχαν στο αρχείο της εικόνας εισόδου.

🔗 Προσοχή πρέπει να δοθεί στις προτάσεις **p.red=p.red^(rand()%256)**; κ.λπ., οι οποίες κάνουν όλη τη δουλειά. Κάθε χρώμα ενός pixel κωδικοποιείται/αποκωδικοποιείται με χρήση της πράξης XOR με έναν τυχαίο αριθμό. Για να γίνει σωστά η κωδικοποίηση και η αποκωδικοποίηση θα πρέπει ο αριθμός αυτός να είναι ο ίδιος.

🔗 Από τη στιγμή που η βάση της γεννήτριας τυχαίων αριθμών (η οποία καθορίζεται από την **srand()**) είναι ίδια, η συνάρτηση **rand()** επιστρέφει πάντα την ίδια ακολουθία τυχαίων αριθμών, οπότε η παραπάνω απαίτηση ικανοποιείται.

4η Εργασία – Κρυφό μήνυμα σε εικόνα

Ένας άδικα φυλακισμένος πολιτικός κρατούμενος θέλει να επικοινωνεί με τους συντρόφους του, αλλά ότι και να στέλνει, με ηλεκτρονικό ή άλλο τρόπο, περνάει από αυστηρό έλεγχο. Το μόνο που του επιτρέπεται να στέλνει είναι αθώες καθημερινές φωτογραφίες, μέσω του ηλεκτρονικού του ταχυδρομείου. Θέλουμε να τον βοηθήσετε ώστε να μπορεί να στέλνει κρυμμένα μηνύματα μέσα σε αυτές τις φωτογραφίες! Για να το κάνετε αυτό, θα πρέπει να εκμεταλλευτείτε τα αχρησιμοποίητα byte συμπλήρωσης που υπάρχουν σε αρχεία εικόνων τύπου bmp. Για αυτό πρέπει να μελετήσετε με προσοχή το Παράδειγμα Π14.8 στο οποίο αναλύεται πλήρως ο τρόπος αποθήκευσης μιας εικόνας σε ένα αρχείο bmp με βάθος χρώματος 24 bit. Στο τέλος κάθε σειράς από pixel υπάρχουν από ένα έως τρία κενά byte συμπλήρωσης (padding bytes), ανάλογα με τις διαστάσεις της εικόνας. Εκεί μέσα θα κρύψουμε το κείμενο που θέλουμε. Είναι προφανές ότι θα προτιμήσουμε εικόνες που διαθέτουν byte συμπλήρωσης.



Στο πλαίσιο της εργασίας θα πρέπει να φτιάξετε δύο προγράμματα. Ένα που θα κρύβει ένα κείμενο μέσα σε ένα αρχείο εικόνας και ένα που θα το εξάγει. Μπορείτε να χρησιμοποιήσετε τμήματα από τον κώδικα των Παραδειγμάτων Π14.8 και Π14.9, στα δύο προγράμματα που θα φτιάξετε.

Στο φάκελο ΕΡΓΑΣΙΕΣ/Ε4, θα βρείτε τα εκτελέσιμα αρχεία των προγραμμάτων, καθώς και εικόνες για τις δοκιμές σας!

Οδηγίες – Βοήθεια:

- Το πρόγραμμα κωδικοποίησης θα πρέπει να ζητάει από τον χρήστη να πληκτρολογήσει το όνομα του αρχείου εικόνας, καθώς και το όνομα του αρχείου κειμένου το οποίο θα κρυφτεί μέσα στην εικόνα.
- Το πρόγραμμα θα πρέπει να ελέγχει αν η εικόνα είναι κατάλληλη για αυτή τη δουλειά. Δηλαδή, αν είναι τύπου bmp, με βάθος χρώματος 24 bit, και αν διαθέτει byte συμπλήρωσης. Επίσης θα πρέπει να ενημερώνει τον χρήστη με το μέγιστο πλήθος χαρακτήρων που μπορούν να κρυφτούν στη εικόνα. Το πρόγραμμα θα πρέπει να εντοπίζει τη θέση των byte συμπλήρωσης σε κάθε γραμμή ώστε να μετακινεί στο σημείο αυτό τον δείκτη θέσης και να γράφει τους χαρακτήρες προς απόκρυψη (δείτε το Παράδειγμα Π14.8).
- Το πρόγραμμα αποκωδικοποίησης θα πρέπει να ζητάει από τον χρήστη να πληκτρολογήσει μόνο το όνομα του αρχείου εικόνας. Το κείμενο που θα εξαχθεί θα πρέπει να γραφεί σε ένα αρχείο κειμένου με όνομα `out.txt`. Η διαδικασία είναι παρόμοια με αυτή του προγράμματος κωδικοποίησης, με τη διαφορά ότι αντί να γράφει τους χαρακτήρες στα byte συμπλήρωσης, θα τους διαβάζει.

Έχετε υπόψη ότι byte συμπλήρωσης υπάρχουν μόνο όταν η τιμή της παράστασης `3*platos` (πλάτος της εικόνας σε pixel) δεν είναι πολλαπλάσια του 4. Σε αυτή την περίπτωση, το πλήθος τους υπολογίζεται από την παράσταση: `4-(3*platos)%4`.

Κεφάλαιο

15

Προχωρημένα θέματα

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό, όπως δηλώνει και ο τίτλος του, καλύπτει κάποια εξειδικευμένα θέματα. Οι δείκτες προς συναρτήσεις, οι αναδρομικές και οι εμβόλιμες συναρτήσεις, καθώς και οι συναρτήσεις με μεταβλητό πλήθος ορισμάτων, είναι μερικά από αυτά.

Η αναδρομή είναι ένα από τα θέματα που καλύπτεται εκτενώς, αφού αρκετά από τα κλασικά προβλήματα της επιστήμης των υπολογιστών επιλύονται με αναδρομικές διαδικασίες.

Στο κεφάλαιο περιγράφεται επίσης ο τρόπος μεταγλώττισης προγραμμάτων τα οποία αποτελούνται από πολλά αρχεία πηγαίου κώδικα, καθώς και η δυνατότητα κλήσης ενός προγράμματος από τη γραμμή εντολών, με χρήση ορισμάτων.

Αναδρομή

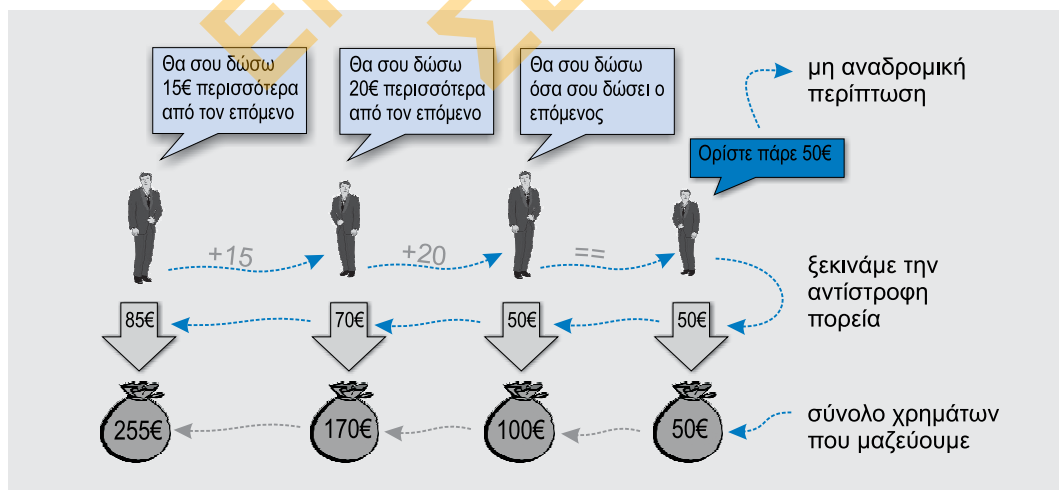
Αναδρομή (recursion) είναι η διαδικασία κατά την οποία ορίζουμε μια λειτουργία σε σχέση με τον εαυτό της. Αν και είναι λίγο δυσνόητο, ας δούμε ένα απλό παράδειγμα:

Ας υποθέσουμε ότι κάνουμε έναν έρανο στον δρόμο. Αν ζητήσουμε χρήματα από τον πρώτο που θα συναντήσουμε μπορεί να μας απαντήσει ως εξής: «Θα σου δώσω 15€ περισσότερα από όσα θα σου δώσει ο επόμενος», πάμε στον επόμενο και αυτός απαντάει με τον ίδιο τρόπο: «Θα σου δώσω 20€ περισσότερα από όσα θα σου δώσει ο επόμενος» ή «Θα σου δώσω όσα θα σου δώσει ο επόμενος» κ.ο.κ. Έστω ότι αυτό συνεχίζεται και με άλλους πολίτες, μέχρι να βρούμε κάποιον ο οποίος δεν θα απαντήσει έτσι και θα μας δώσει τα χρήματα. Τότε αρχίζουμε την αντίστροφη πορεία και πάμε σε όλους όσους μας υποσχέθηκαν χρήματα, και τους ζητάμε να μας δώσουν τα χρήματα που τους αναλογούν.

Το Σχήμα 15.1 που ακολουθεί δείχνει ένα παράδειγμα αναδρομικής διαδικασίας, όπως αναφέρθηκε παραπάνω. Οι πρώτοι τρεις πολίτες που συναντάμε μας υπόσχονται ένα χρηματικό ποσό ανάλογο με το ποσό που θα μας δώσει ο επόμενος. Όμως, ο τέταρτος δεν ακολουθεί την ίδια τακτική και μας δίνει 50€. Σε εκείνο το σημείο ξεκαθαρίζουν όλα και πλέον μπορούμε να μαζέψουμε τα χρήματα που μας υποσχέθηκαν οι προηγούμενοι ακολουθώντας αντίστροφη πορεία.

Κάθε αναδρομική διαδικασία πρέπει να έχει μία **μη αναδρομική περίπτωση**, διαφορετικά θα συνεχίζεται επ' άπειρο.

Στη C, η διαδικασία της αναδρομής μπορεί να υλοποιηθεί όταν μια συνάρτηση καλεί τον εαυτό της!



Σχήμα 15.1 Παράδειγμα αναδρομικής διαδικασίας

Αν στο σώμα μιας συνάρτησης μια πρόταση καλεί την ίδια τη συνάρτηση, τότε η συνάρτηση λέγεται **αναδρομική** (recursive).

Ας δούμε την επόμενη συνάρτηση, η οποία επιστρέφει ως τιμή το παραγοντικό ενός αριθμού n ($1*2*3 \dots * n$). Για παράδειγμα, το παραγοντικό του 3 είναι 6 ($1*2*3$) και του 8 είναι 40320 ($1*2*3*4*5*6*7*8$).

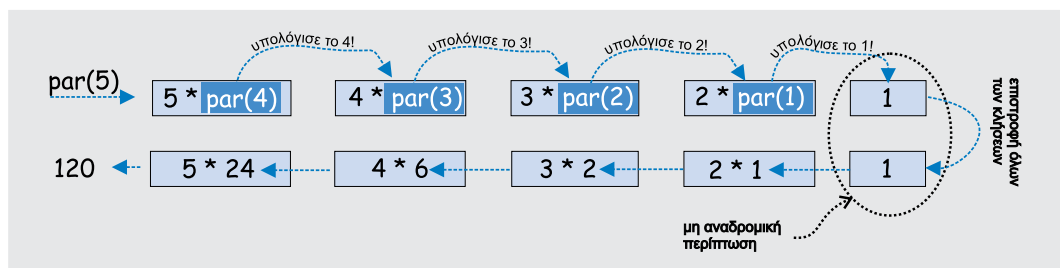
```
int par(int n)
{
    int i,p;
    p=1;
    for (i=1;i<=n;i++) p=p*i;
    return p;
}
```

Η προηγούμενη συνάρτηση υπολογίζει (με τον κλασικό τρόπο) και επιστρέφει ως τιμή το παραγοντικό ενός αριθμού n . Το παραγοντικό ($n!$) ενός αριθμού n είναι το γινόμενο όλων των ακέραιων αριθμών από το 1 μέχρι το n . Επίσης, το παραγοντικό ($n!$) ενός αριθμού n ορίζεται ως το γινόμενο του n και του παραγοντικού του προηγούμενου αριθμού, δηλαδή $n!=n*(n-1)!$, οπότε το παραγοντικό του 6 είναι $6 * 5!$. Έτσι, ο υπολογισμός του παραγοντικού μπορεί να υλοποιηθεί με αναδρομική διαδικασία.

```
int par(int n)
{
    int p;
    if (n==1) return 1;
    p=n*par(n-1);
    return p;
}
```

Έλεγχος για μη αναδρομική περίπτωση

Για παράδειγμα, αν η παραπάνω συνάρτηση κληθεί με παράμετρο 5, δηλαδή $a=\text{par}(5)$, θα ακολουθήσει τη διαδικασία του Σχήματος 15.2 σύμφωνα με την οποία η συνάρτηση θα επιστρέψει την τιμή της που τελικά θα αποθηκευτεί στη μεταβλητή a . Οι αναδρομικές κλήσεις της συνάρτησης $\text{par}()$ θα σταματήσουν όταν ζητηθεί το παραγοντικό του 1, το οποίο αποτελεί τη μη αναδρομική περίπτωση.



Σχήμα 15.2 Παράδειγμα αναδρομικής κλήσης συνάρτησης

Εμβόλιμες συναρτήσεις

Από το κεφάλαιο των συναρτήσεων γνωρίζουμε ότι ο κώδικας μιας συνάρτησης εμφανίζεται μόνο μία φορά μέσα στο μεταγλωττισμένο πρόγραμμα.

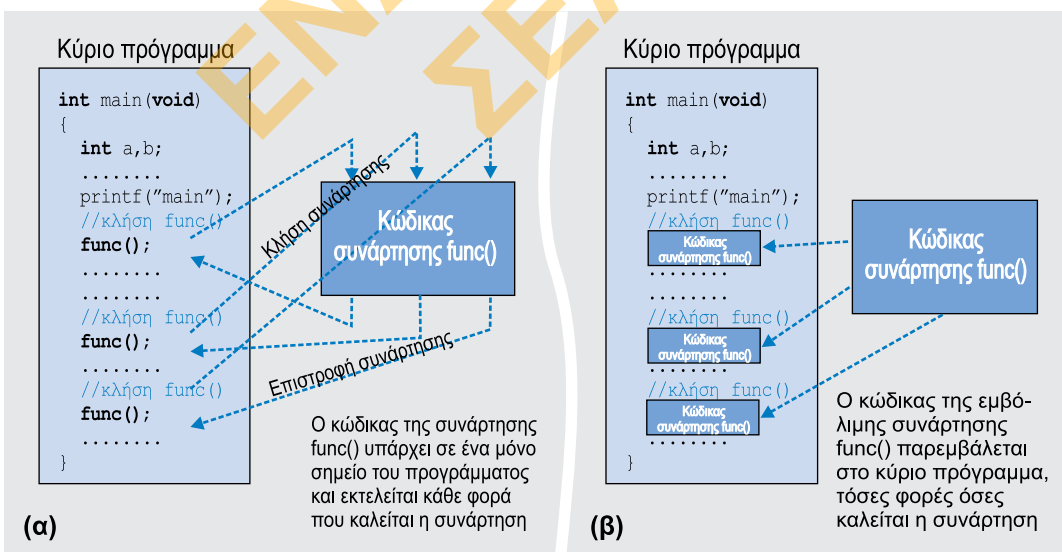
Κάθε φορά που καλείται μια συνάρτηση, ο έλεγχος του προγράμματος μεταφέρεται σε αυτό το τμήμα κώδικα, και με την ολοκλήρωσή του επιστρέφει πίσω αμέσως μετά από το σημείο της κλήσης (Σχήμα 15.5α).

Για τις διαδικασίες της κλήσης και της επιστροφής της συνάρτησης, ο μεταγλωττιστής δημιουργεί τον κατάλληλο κώδικα ο οποίος όμως επιφέρει κάποια, έστω και μικρή, καθυστέρηση. Αν το μέγεθος μιας συνάρτησης είναι μικρό, τότε ο κώδικας που απαιτείται για την κλήση και την επιστροφή της ίσως να είναι περισσότερος από τον κώδικα της ίδιας της συνάρτησης! Σε αυτές τις περιπτώσεις η καθυστέρηση είναι αναλογικά μεγάλη, και όταν θέλουμε βέλτιστη απόδοση από το πρόγραμμά μας τότε ακόμη και οι μικρές καθυστερήσεις έχουν συνέπειες.

Για αυτές τις περιπτώσεις, η C διαθέτει τις **εμβόλιμες** (inline) συναρτήσεις.

Όταν ο μεταγλωττιστής συναντήσει την κλήση μιας εμβόλιμης συνάρτησης, τότε παρεμβάλλει τον κώδικα της συνάρτησης στο σημείο της κλήσης. Αν η συνάρτηση καλείται περισσότερες φορές, σε κάθε σημείο της κλήσης παρεμβάλλεται ολόκληρος ο κώδικας της συνάρτησης (Σχήμα 15.5β).

Μια εμβόλιμη συνάρτηση δηλώνεται όπως και οποιαδήποτε άλλη, με τη διαφορά ότι πριν από τον τύπο της τοποθετείται το πρόθεμα **inline**.



Σχήμα 15.5 Εμβόλιμες (inline) συναρτήσεις

```
inline int func(int x, int y)
{
    return x+y;
}
```

Η συνάρτηση **func()**, δηλώνεται ως εμφύλιμη. Αυτό είναι μια **προτροπή** προς το μεταγλωττιστή να παρεμβάλει τον κώδικα της συνάρτησης σε κάθε σημείο κλήσης της.

Η δήλωση μιας συνάρτησης ως εμφύλιμης είναι απλώς μια **προτροπή** προς τον μεταγλωττιστή. Ο μεταγλωττιστής μπορεί να αποφασίσει να τη μεταγλωττίσει ως κανονική συνάρτηση. Για παράδειγμα, ίσως θεωρήσει ότι η συνάρτηση έχει πολύ μεγάλο μέγεθος για να γίνει εμφύλιμη, και να την καλέσει με τον συμβατικό τρόπο.

Για να μπορέσει ο μεταγλωττιστής να κάνει μια συνάρτηση εμφύλιμη, θα πρέπει να έχει συναντήσει τον ορισμό της (όχι τη δήλωσή της) πριν από την πρώτη κλήση της. Για παράδειγμα, στην επόμενη περίπτωση η συνάρτηση **func()** δεν θα αντιμετωπιστεί ως εμφύλιμη, διότι όταν ο μεταγλωττιστής συναντήσει την κλήση της μέσα στη **main()**, δεν γνωρίζει ακόμη τον κώδικα της συνάρτησης (η οποία ορίζεται μετά) ώστε να τον παρεμβάλει σε αυτό το σημείο.

```
inline int func(int x, int y);
```

Προκαταβολική δήλωση της συνάρτησης **func()**.

```
int main(void)
{
    int a=4,b=8;
    printf("%d\n", func(a,b));
    return 0;
}
```

Στο σημείο αυτό ο μεταγλωττιστής συναντά την κλήση της **func()**, αλλά δεν γνωρίζει τον κώδικα της συνάρτησης (η οποία ορίζεται μετά) ώστε να τον παρεμβάλει σε αυτό το σημείο. Επομένως, η συνάρτηση καλείται με τον συμβατικό τρόπο.

```
inline int func(int x, int y)
{
    return x+y;
}
```

Ορισμός της συνάρτησης **func()**. Το πρόθεμα **inline** προτρέπει τον μεταγλωττιστή να τη χρησιμοποιήσει ως εμφύλιμη.

Στον παραπάνω κώδικα, για να αντιμετωπιζόταν η συνάρτηση **func()** ως εμφύλιμη, θα έπρεπε να έχει οριστεί πριν από τη **main()**. Σε αυτή την περίπτωση θα ήταν περιττή η προκαταβολική δήλωσή της.

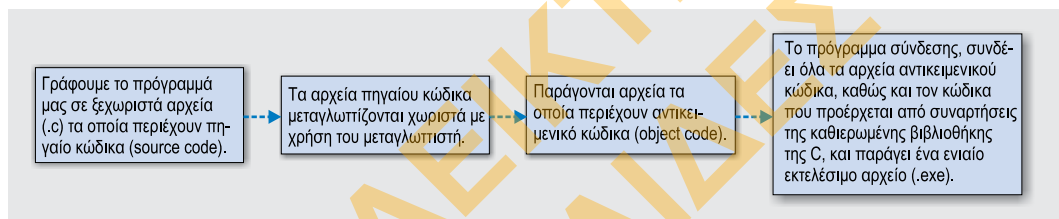
Για να συνοψίσουμε, ισχύουν τα παρακάτω βασικά θέματα που αφορούν τις εμφύλιμες συναρτήσεις:

- Το πρόθεμα **inline** είναι απλώς μια προτροπή προς τον μεταγλωττιστή να προσπαθήσει να αντιμετωπίσει τη συνάρτηση ως εμφύλιμη.
- Οι εμφύλιμες συναρτήσεις πρέπει να είναι σχετικά μικρές.
- Για να αντιμετωπιστεί μια συνάρτηση ως εμφύλιμη, θα πρέπει να δίνεται ο ορισμός της πριν από την πρώτη κλήση της.
- Τελικά, ο μεταγλωττιστής θα αποφασίσει αν θα χειριστεί ως εμφύλιμη μια συνάρτηση που δηλώνεται με το πρόθεμα **inline**.

Προγράμματα με πολλά πηγαία αρχεία

Μέχρι τώρα, όλα τα προγράμματα με τα οποία ασχοληθήκαμε αποτελούνταν από ένα μόνο πηγαίο αρχείο. Όμως, ένα πρόγραμμα της C μπορεί να αποτελείται από περισσότερα πηγαία αρχεία, κάθε ένα από τα οποία μεταγλωττίζεται ξεχωριστά. Η τεχνική αυτή βοηθάει στη καλύτερη δομή του προγράμματος, και επίσης κάνει πιο εύκολη την αποσφαλμάτωση καθώς και τη συντήρησή του. Για παράδειγμα, η αλλαγή στον κώδικα μιας συνάρτησης θα οδηγήσει στη μεταγλώττιση μόνο του αρχείου που περιέχει τη συγκεκριμένη συνάρτηση, και όχι ολόκληρου του προγράμματος.

Η δημιουργία ενός προγράμματος στη C περνάει από δύο στάδια: Κατά το στάδιο της **μεταγλώττισης**, ο μεταγλωττιστής μεταφράζει τον πηγαίο κώδικα σε αντικειμενικό κώδικα (object code), δηλαδή σε κώδικα γλώσσας μηχανής ο οποίος μπορεί να εκτελεστεί άμεσα από τον επεξεργαστή. Κατά το δεύτερο στάδιο, το στάδιο της **σύνδεσης**, το πρόγραμμα σύνδεσης (linker) συνενώνει όλα τα αρχεία αντικειμενικού κώδικα, μαζί με κώδικα που προέρχεται από την καθιερωμένη βιβλιοθήκη της γλώσσας, σε ένα ενιαίο εκτελέσιμο αρχείο. Το σχήμα που ακολουθεί δείχνει αυτά τα διαφορετικά στάδια:



Ο μεταγλωττιστής GCC

Το ολοκληρωμένο περιβάλλον Code::Blocks, το οποίο συνοδεύει το βιβλίο και πιθανώς να το έχετε ήδη χρησιμοποιήσει για τη συγγραφή των προγραμμάτων σας, χρησιμοποιεί τον μεταγλωττιστή **GCC**. Ο μεταγλωττιστής GCC ουσιαστικά είναι δύο προγράμματα σε ένα: Μπορεί να χρησιμοποιηθεί ως μεταγλωττιστής αλλά και ως πρόγραμμα σύνδεσης! Το Σχήμα 15.6 δείχνει τη διαδικασία μεταγλώττισης ενός προγράμματος το οποίο έχει γραφεί σε τρία ξεχωριστά αρχεία πηγαίου κώδικα. Δυστυχώς, για να χρησιμοποιήσετε με άμεσο τρόπο τον μεταγλωττιστή GCC, θα πρέπει να εγκαταλείψετε το εύχρηστο γραφικό περιβάλλον του Code::Blocks και να ανοίξετε ένα παράθυρο γραμμής εντολών (command line). Η χρήση του μεταγλωττιστή από τη γραμμή εντολών θα μας βοηθήσει να κατανοήσουμε πλήρως τα διαφορετικά στάδια της μεταγλώττισης και της σύνδεσης.

Το εκτελέσιμο αρχείο **gcc.exe** του μεταγλωττιστή GCC βρίσκεται μέσα στον φάκελο **CodeBlocksFolder\MinGW\bin**. Για να μπορούμε να το εκτελούμε από οποιονδήποτε φάκελο, θα πρέπει να προσθέσουμε στη μεταβλητή **path** του περιβάλλοντός μας τον συγκεκριμένο φάκελο. Η επόμενη εντολή της γραμμής εντολών κάνει ακριβώς αυτό.

```
c:\>path %path%;CodeBlocksFolder\MinGW\bin
```

Εδώ, το **CodeBlocksFolder** αντικαθίσταται με την απόλυτη διαδρομή του φακέλου εγκατάστασης του Code::Blocks, η οποία ανάλογα με το λειτουργικό μας σύστημα μπορεί να είναι: C:\Program Files\CodeBlocks ή C:\Αρχεία Εφαρμογών\CodeBlocks, κλπ.

Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων

Ας υποθέσουμε ότι έχουμε χωρίσει το πρόγραμμά μας σε τρία διαφορετικά αρχεία πηγαίου κώδικα. Το πρώτο αρχείο **intro.c** περιέχει τη συνάρτηση **book()**, η οποία εμφανίζει στην οθόνη το κείμενο «Η Γλώσσα C σε βάθος».

// αρχείο πηγαίου κώδικα intro.c

```
#include <stdio.h>
#include <stdlib.h>
void book()
{
    printf("Η Γλώσσα C σε βάθος\n");
}
```

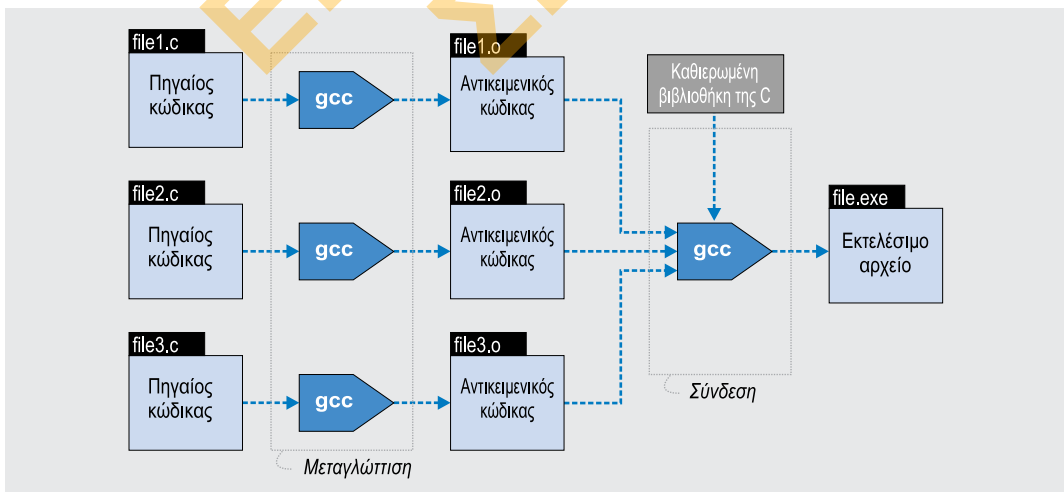
15_separate1/intro.c

Το δεύτερο αρχείο **bye.c** περιέχει τη συνάρτηση **thanks()**, η οποία δέχεται ως όρισμα μια συμβολοσειρά και εμφανίζει στην οθόνη το κείμενο «Ευχαριστώ ##!», όπου το ## είναι η συμβολοσειρά.

// αρχείο πηγαίου κώδικα bye.c

```
#include <stdio.h>
#include <stdlib.h>
void thanks(char *name)
{
    printf("Ευχαριστώ %s!\n", name);
}
```

15_separate1/bye.c



Σχήμα 15.6 Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων με χρήση του μεταγλωττιστή GCC

Συναρτήσεις με μεταβλητό πλήθος ορισμάτων

Συνήθως, μια συνάρτηση της C πρέπει να καλείται με ένα συγκεκριμένο πλήθος υποχρεωτικών ορισμάτων. Δηλαδή, στην πρόταση ορισμού της καθορίζεται τόσο το πλήθος όσο και ο τύπος της κάθε παραμέτρου. Όμως, υπάρχουν περιπτώσεις που το πλήθος των ορισμάτων κατά τη κλήση μιας συνάρτησης δεν θέλουμε να είναι συγκεκριμένο. Οι συναρτήσεις βιβλιοθήκης `printf()` και `scanf()` είναι παραδείγματα συναρτήσεων με μεταβλητό πλήθος ορισμάτων (variadic functions).

Στον ορισμό μιας συνάρτησης με μεταβλητό πλήθος ορισμάτων πρέπει να δίνεται τουλάχιστον μια υποχρεωτική παράμετρος. Κατά τη δήλωση μιας τέτοιας συνάρτησης, μετά από τις υποχρεωτικές παραμέτρους ακολουθεί μια ειδική παράμετρος χωρίς τύπο με όνομα τρεις τελείες (...), η οποία υποδηλώνει ότι η συνάρτηση θα δέχεται μεταβλητό πλήθος προαιρετικών ορισμάτων.

Για παράδειγμα, τα ακόλουθα πρωτότυπα δηλώνουν συναρτήσεις με μεταβλητό πλήθος ορισμάτων:

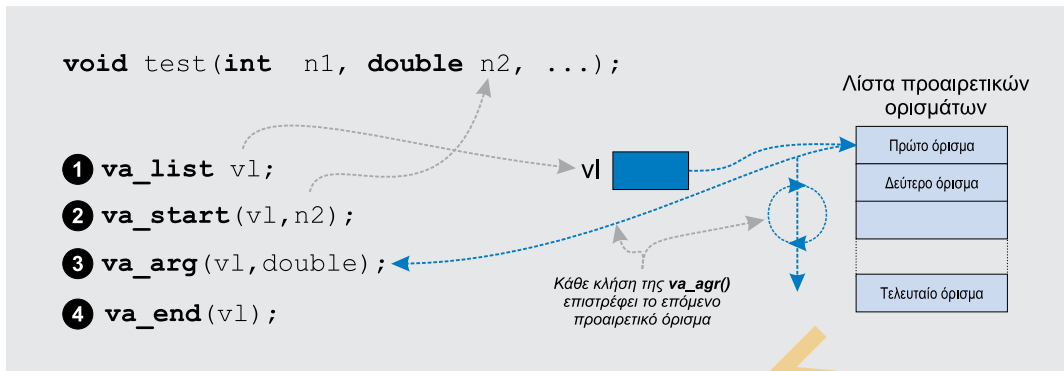
```
void PrintNumbers(int num, ...);
double mesos(int num, ...);
void PrintStrings(char *first, ...);
void test(int n1, double n2, ...);
```

Οι τρεις πρώτες διαθέτουν μια υποχρεωτική παράμετρο και η τέταρτη δύο.

Η διαχείριση των προαιρετικών ορισμάτων γίνεται από τη C μέσω του τύπου `va_list`, καθώς και από ένα πλήθος μακροεντολών²⁶. Κάθε συνάρτηση που μπορεί να κληθεί με μεταβλητό πλήθος ορισμάτων μπορεί να προσπελάζει μια λίστα με τα προαιρετικά όρισματα με τα οποία καλείται. Για την προσπέλαση χρησιμοποιείται μια μεταβλητή τύπου `va_list`. Η μεταβλητή αυτή στην πραγματικότητα λειτουργεί ως δείκτης μέσω του οποίου διαχειριζόμαστε τη λίστα των προαιρετικών ορισμάτων.

Το Σχήμα 15.8 δείχνει τα βήματα και τις μακροεντολές που χρησιμοποιούνται για την προσπέλαση των προαιρετικών ορισμάτων. Στο παράδειγμα του σχήματος, ως συνάρτηση με μεταβλητό πλήθος ορισμάτων χρησιμοποιείται μια συνάρτηση τύπου `void`, με όνομα `test`, με δύο υποχρεωτικές παραμέτρους `n1` και `n2`. Πρώτα πρέπει να δηλωθεί μια μεταβλητή τύπου `va_list`, στο παράδειγμα μας η `vl` (❶). Η αρχικοποίηση αυτής της μεταβλητής γίνεται μέσω της μακροεντολής `va_start()`, της οποίας το πρώτο όρισμα πρέπει να είναι ο χειριστής της λίστας `vl` και το δεύτερο πρέπει να είναι το όνομα της τελευταίας υποχρεωτικής παραμέτρου της συνάρτησης – στην περίπτωσή μας, το `n2` (❷). Η μακροεντολή `va_arg()` χρησιμοποιείται για την εξαγωγή των προαιρετικών ορισμάτων της λίστας, ένα προς ένα. Το πρώτο όρισμα πρέπει να είναι ο χειριστής της λίστας, στην περίπτωση μας το `vl`, και το δεύτερο καθορίζει τον τύπο δεδομένων που αναμένει να βρει στη λίστα των προαιρετικών ορισμάτων (❸).

²⁶ Οι μακροεντολές θα συζητηθούν αργότερα, στο Κεφάλαιο 19. Μέχρι τότε, θα τις αντιμετωπίζουμε ως συναρτήσεις



Σχήμα 15.8 Χρήση μακροεντολών για τη διαχείριση των προαιρετικών ορισμάτων

Κάθε φορά που καλείται η `va_arg()`, επιστρέφει ως τιμή το αμέσως επόμενο όρισμα της λίστας. Όταν τελειώσει η διαδικασία εξαγωγής των προαιρετικών ορισμάτων, πρέπει να κληθεί η μακροεντολή `va_end()` με όρισμα τον χειριστή της λίστας `vl` (4). Η μακροεντολή αυτή σηματοδοτεί το τέλος της διαδικασίας, και επιτρέπει την προσπέλαση της λίστας από την αρχή, με τη χρήση μιας νέας `va_start()`. Οι μακροεντολές αυτές ορίζονται στο αρχείο κεφαλίδας `stdarg.h`.

Το πρόγραμμα που ακολουθεί υλοποιεί το παράδειγμα του Σχήματος 15.8 και δείχνει τη χρήση συναρτήσεων με μεταβλητό πλήθος ορισμάτων.

```
#include <stdio.h>
#include <stdarg.h>
void test(int n1, double n2, ...)
{
    va_list vl;
    double ar;
    int i;
    printf("Πρώτο υποχρεωτικό: %d\n", n1);
    printf("Δεύτερο υποχρεωτικό: %f\n", n2);
    va_start(vl, n2);
    printf("\nΠροαιρετικά ορίσματα\n");
    for (i=1; i<=n1; i++)
    {
        ar=va_arg(vl, double);
        printf("%d -> %f\n", i, ar);
    }
    va_end(vl);
}

int main(void)
{
    test(5, 3.2, 4.4, 5.0, 6.7, 7.5, 8.9);
    return 0;
}
```

15_variadic_func.c

Αρχείο κεφαλίδας στο οποίο ορίζονται οι απαραίτητες μακροεντολές για την υλοποίηση συναρτήσεων με μεταβλητό πλήθος ορισμάτων.

Δήλωση του χειριστή της λίστας των προαιρετικών ορισμάτων.

Πρώτο υποχρεωτικό: 5
Δεύτερο υποχρεωτικό: 3.2

Προαιρετικά ορίσματα:

1 -> 4.4
2 -> 5.0
3 -> 6.7
4 -> 7.5
5 -> 8.9

Σύνοψη

- Η C ενθαρρύνει την ανάπτυξη προγραμμάτων σε ξεχωριστά αρχεία πηγαίου κώδικα. Αυτό οδηγεί σε προγράμματα που αναπτύσσονται, αποσφαλματώνονται, και συντηρούνται πιο εύκολα.
- Η μεταγλώττιση των αρχείων γίνεται ξεχωριστά, οπότε προκύπτουν αρχεία αντικειμενικού κώδικα (object code).
- Κατά τη διαδικασία της σύνδεσης (link), το πρόγραμμα σύνδεσης (linker) συνενώνει τον κώδικα των αντικειμενικών αρχείων σε ένα ενιαίο εκτελέσιμο αρχείο (executable file). Επίσης, στο στάδιο αυτό το πρόγραμμα σύνδεσης ενσωματώνει στον εκτελέσιμο κώδικα και τον κώδικα των συναρτήσεων που έχουμε χρησιμοποιήσει. Τον κώδικα αυτόν τον εντοπίζει μέσα σε αρχεία βιβλιοθήκης (.a), είτε αυτά περιλαμβάνονται στην καθιερωμένη βιβλιοθήκη της C, είτε σε εξωτερικά αρχεία τρίτων κατασκευαστών.
- Αν όμως θέλουμε να χρησιμοποιήσουμε εξωτερικές βιβλιοθήκες, θα πρέπει κατά το στάδιο της σύνδεσης να ενημερώσουμε το πρόγραμμα σύνδεσης ώστε να τις συμπεριλάβει. Σε αντίθετη περίπτωση, το πρόγραμμα σύνδεσης θα συμπεριλάβει μόνο τα αρχεία της καθιερωμένης βιβλιοθήκης.
- Μπορούμε να φτιάξουμε τις δικές μας βιβλιοθήκες, συνενώνοντας διαφορετικά αρχεία αντικειμενικού κώδικα σε ένα αρχείο βιβλιοθήκης. Αυτό γίνεται με ένα ειδικό εργαλείο το οποίο μας παρέχεται από το περιβάλλον του μεταγλωττιστή που χρησιμοποιούμε. Ο μεταγλωττιστής GCC διαθέτει τον «βιβλιοθηκάριο» (archiver) **ar**!
- Η χρήση εξωτερικών καθολικών μεταβλητών και συναρτήσεων μας δίνει τη δυνατότητα να επεκτείνουμε την εμβέλειά τους σε όλα τα αρχεία πηγαίου κώδικα ενός προγράμματος.
- Το πρόθεμα **static** στον ορισμό μιας καθολικής μεταβλητής ή μιας συνάρτησης περιορίζει τη χρήση της μόνο μέσα στο αρχείο στο οποίο ορίστηκε.
- Τα επίπεδα εμβέλειας είναι πλέον τέσσερα. Από το χαμηλότερο στο υψηλότερο, είναι τα εξής: σύνθετη πρόταση, συνάρτηση, αρχείο, ολόκληρο το πρόγραμμα (όλα τα αρχεία κώδικα από τα οποία αποτελείται).
- Η C μας δίνει τη δυνατότητα να ορίσουμε συναρτήσεις με μεταβλητό πλήθος ορισμάτων. Οι συναρτήσεις αυτές πρέπει να διαθέτουν τουλάχιστον μία υποχρεωτική παράμετρο. Μια τελευταία παράμετρος με όνομα τρεις τελείες (...) υποδηλώνει το μεταβλητό πλήθος ορισμάτων που δέχεται η συνάρτηση. Η διαχείριση των προαιρετικών ορισμάτων γίνεται μέσω ειδικών μακροεντολών και ενός ειδικού δείκτη ο οποίος χρησιμοποιείται για τον χειρισμό της λίστας των προαιρετικών ορισμάτων.

Παραδείγματα

Π15.1 Η επόμενη αναδρομική συνάρτηση δέχεται ως όρισμα έναν αριθμό και τον εμφανίζει αντίστροφα. Για παράδειγμα, αν δοθεί ο 4672 εμφανίζει τον 2764.

```
int reverse(int n)
{
    if (n==0)
        return;
    else
    {
        printf("%d",n%10);
        return reverse(n/10);
    }
}
```

Μη αναδρομική περίπτωση.

Εμφανίζει το υπόλοιπο της διαίρεσης με το 10, δηλαδή το λιγότερο σημαντικό ψηφίο του αριθμού.

Καλεί αναδρομικά τη συνάρτηση με παράμετρο το πηλίκο της ακεραίας διαίρεσης του αριθμού με το 10.

- 👉 Η συνάρτηση εμφανίζει το υπόλοιπο της διαίρεσης της τιμής της παραμέτρου της με το 10 ($n\%10$). Αυτό πρακτικά είναι το τελευταίο ψηφίο του αριθμού της παραμέτρου.
- 👉 Η συνάρτηση καλείται αναδρομικά με το πηλίκο της ακεραίας διαίρεσης της τιμής της παραμέτρου της με το 10 ($n/10$).
- 👉 Κάθε αναδρομική συνάρτηση πρέπει να έχει μία τουλάχιστον μη αναδρομική περίπτωση. Στη συγκεκριμένη περίπτωση, αυτό συμβαίνει όταν το $n==0$.

Το πρόγραμμα που ακολουθεί ζητάει έναν αριθμό από το πληκτρολόγιο και τον εμφανίζει αντίστροφα χρησιμοποιώντας τη συνάρτηση `reverse()`.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int ar;
    printf("Δώσε αριθμό:");
    // Διαβάζει έναν αριθμό από το πληκτρολόγιο
    scanf("%d",&ar);
    // Εμφάνιση του αριθμού με αντίστροφη σειρά των ψηφίων του
    reverse(ar);
    printf("\n-----\n");
    return 0;
}
```

Π15.2 Ο επόμενος τύπος, που προέρχεται από τον Leonard Euler (1707-1783), υπολογίζει προσεγγιστικά το π . Το πρόγραμμα που ακολουθεί χρησιμοποιεί αυτή τη σχέση για να υπολογίσει την τιμή του π . Το πρόγραμμα χρησιμοποιεί αναδρομική συνάρτηση για τον υπολογισμό του δεξιού μέλους της σχέσης.

$$\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$$

Η συνάρτηση δέχεται ως όρισμα το n . Για τον υπολογισμό χρησιμοποιούνται οι 1000 πρώτοι όροι.

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

15_p15_2.c

```
double melos();
```

```
int main(void)
{
    double p;
    p=sqrt(melos(1000)*6);
    printf("p=%f\n",p);
    return 0;
}
```

Επιστρέφει την τιμή του δεξιού μέλους της σχέσης για τους 1000 πρώτους όρους.

Υπολογίζει την τιμή του p (π) επιλύοντας την παραπάνω σχέση ως προς p .

```
double melos(int n)
{
    double res;
    if (n==1)
        return 1.0;
    else
    {
        res=1.0/(n*n)+melos(n-1);
        return res;
    }
}
```

Μη αναδρομική περίπτωση.

Αναδρομική κλήση της συνάρτησης.

Η συνάρτηση θα κληθεί αναδρομικά 999 φορές! Αυτό σημαίνει ότι θα εισαγάγει τις τιμές των τοπικών της μεταβλητών στη στοίβα 999 φορές. Αν η συνάρτηση κληθεί με όρισμα κάποια μεγάλη τιμή (μεγαλύτερη από το 1000 του παραδείγματος), τότε μπορεί να προκύψει σφάλμα υπερχείλισης στοίβας (stack overflow). Δοκιμάστε τη με τιμή 100000.

Π15.3 Στα μαθηματικά, η ακολουθία Fibonacci (Φιμπονάτσι)²⁷ είναι η ακολουθία των ακέραιων αριθμών: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,... Εξ ορισμού, οι δύο πρώτοι αριθμοί Fibonacci είναι το 0 και το 1, και κάθε επόμενος αριθμός είναι το άθροισμα των δύο προηγούμενων. Ο πίνακας που ακολουθεί δείχνει τους δώδεκα πρώτους όρους της ακολουθίας. Ο πρώτος όρος είναι ο 0 και συμβολίζεται με f_0 .

n	0	1	2	3	4	5	6	7	8	9	10	11
f_n	0	1	1	2	3	5	8	13	21	34	55	89

Κάθε όρος της ακολουθίας Fibonacci υπολογίζεται από την ακόλουθη μαθηματική σύμβαση:

$$f_0 = 0$$

$$f_1 = 1 \text{ και για } n > 1$$

$$f_n = f_{n-1} + f_{n-2}$$

Παρατηρούμε ότι ο υπολογισμός του n-οστού όρου της ακολουθίας έχει καθαρά αναδρομικό χαρακτήρα. Το πρόγραμμα που ακολουθεί χρησιμοποιεί την αναδρομική συνάρτηση `fibonacci()` η οποία επιστρέφει τον όρο της ακολουθίας ανάλογα με την τιμή που της μεταβιβάζεται ως όρισμα.

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

15_p15_3.c

```
int fibonacci(int oros);
```

```
int main(void)
{
    int i;
    for (i=0; i<=11; i++)
        printf("%d ", fibonacci(i));
    return 0;
}
```

Καλείται η συνάρτηση `fibonacci()` για τους δώδεκα πρώτους όρους της ακολουθίας ($n=0-11$).

```
int fibonacci(int oros)
```

```
{
    if (oros==0)
        return 0;
    else if (oros==1)
        return 1;
    else
        return fibonacci(oros-1)+fibonacci(oros-2);
}
```

Μη-αναδρομικές περιπτώσεις κλήσης της συνάρτησης

Αναδρομικές κλήσεις της συνάρτησης

²⁷ Η ακολουθία Fibonacci ονομάστηκε έτσι από τον Λεονάρντο της Πίζα, γνωστό και ως Fibonacci. Στο βιβλίο που έγραψε το 1202 με τίτλο *Liber Abaci*, εισήγαγε την ακολουθία στα Μαθηματικά της Δυτικής Ευρώπης, όμως η ακολουθία είχε περιγραφεί πιο πριν από τους Ινδούς.

Συχνά λάθη

- ❖ Δεν διασφαλίζουμε την ύπαρξη μη αναδρομικής περίπτωσης σε αναδρομικές συναρτήσεις.
- ❖ Ξεχνάμε ότι το στοιχείο `argv[0]` περιέχει έναν δείκτη προς τη συμβολοσειρά του ονόματος του εκτελέσιμου αρχείου, και όχι της πρώτης από τις παραμέτρους της γραμμής εντολών.

Ανασκόπηση Κεφαλαίου 15

- Αναδρομική είναι μια διαδικασία η οποία μπορεί να οριστεί σε σχέση με την ίδια τη διαδικασία.
- Αναδρομική συνάρτηση είναι η συνάρτηση η οποία καλεί ξανά τον εαυτό της. Κάθε αναδρομική συνάρτηση έχει τουλάχιστον μία μη αναδρομική περίπτωση.
- Στις παραμέτρους της συνάρτησης `main()` μεταβιβάζονται τα ορίσματα της γραμμής εντολών με τα οποία εκτελούμε το πρόγραμμα.
- Ένας δείκτης προς συνάρτηση είναι μια μεταβλητή δείκτη η οποία περιέχει τη διεύθυνση μιας συνάρτησης.
- Μπορούμε να καλέσουμε μια συνάρτηση χρησιμοποιώντας μια μεταβλητή δείκτη η οποία περιέχει τη διεύθυνσή της.
- Αν μια συνάρτηση δηλωθεί ως `inline` (ένθετη), ο μεταγλωττιστής προσπαθεί να παρεμβάλει τον κώδικα της συνάρτησης στο σημείο της κλήσης της τόσες φορές όσες την εντοπίσει μέσα στο πρόγραμμα το οποίο την καλεί.
- Η C ενθαρρύνει την ανάπτυξη προγραμμάτων σε ξεχωριστά αρχεία πηγαίου κώδικα. Η μεταγλώττιση των αρχείων γίνεται χωριστά μέσω του μεταγλωττιστή, και η σύνδεσή τους σε ένα ενιαίο εκτελέσιμο αρχείο γίνεται μέσω του προγράμματος σύνδεσης.
- Η C δίνει επίσης τη δυνατότητα να δημιουργήσουμε βιβλιοθήκες με τις δικές μας συναρτήσεις. Τις βιβλιοθήκες αυτές μπορούμε να τις συνδέουμε με οποιοδήποτε δικό μας πρόγραμμα, και να χρησιμοποιούμε τις συναρτήσεις μας όπως τις συναρτήσεις της καθιερωμένης βιβλιοθήκης της C.
- Η C επιτρέπει τη δημιουργία συναρτήσεων με μεταβλητό πλήθος ορισμάτων. Η διαχείριση αυτών των ορισμάτων γίνεται μέσω του τύπου `va_list`, καθώς και από ένα πλήθος μακροεντολών.
- Με το προσδιοριστικό `extern` μπορούμε να επεκτείνουμε την εμβέλεια μιας καθολικής μεταβλητής σε περισσότερα αρχεία πηγαίου κώδικα. Με το προσδιοριστικό `static` περιορίζουμε την εμβέλεια καθολικών μεταβλητών και συναρτήσεων μόνο μέσα στο αρχείο στο οποίο ορίζονται.

Ασκήσεις Κεφαλαίου 15

- 15.1** Να γραφεί μια αναδρομική συνάρτηση η οποία να υπολογίζει το άθροισμα της σειράς $1/1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$.

Η συνάρτηση θα δέχεται ως όρισμα τον αριθμό n . ★★

- 15.2** Να εντοπιστεί το λάθος στην επόμενη αναδρομική συνάρτηση. ★

```
int par(int n)
{
    int p;
    p=n+par(n-1);
    return p;
}
```

- 15.3** Τι κάνει η επόμενη συνάρτηση; ★

```
int par(int n)
{
    int p;
    if (n==0) return 0;
    p=n+par(n/2);
    return p;
}
```

Τι τιμή θα επιστρέψει η παράσταση $\text{par}(20)$;

- 15.4** Ο επόμενος τύπος υπολογίζει προσεγγιστικά το π . Να γραφεί ένα πρόγραμμα το οποίο να χρησιμοποιεί αυτόν τον τύπο και να υπολογίζει την τιμή του π . Για τον υπολογισμό του δεξιού μέλους να χρησιμοποιηθούν αναδρομικές συναρτήσεις. Να χρησιμοποιηθούν οι 1000 πρώτοι όροι. ★★★

$$\pi = 2 \times \frac{2}{\sqrt{2}} \times \frac{2}{\sqrt{2+\sqrt{2}}} \times \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2}}}} \times \dots$$

- 15.5** Να γραφεί μια συνάρτηση η οποία να επιστρέφει το πλήθος των χαρακτήρων διαστήματος που περιέχονται σε μια συμβολοσειρά. Στη συνάρτηση θα μεταβιβάζεται μόνο η διεύθυνση της συμβολοσειράς. Η συνάρτηση θα πρέπει να είναι αναδρομική. Να γραφεί επίσης ένα πρόγραμμα που να δείχνει τη λειτουργία της συνάρτησης. ★★★

```
int main(void)
{
    printf("%d\n",mx(4,2,8,4,5));
    printf("%d\n",mx(1,5));
    printf("%d\n",mx(3,5,10,6,15,20,12));
    return 0;
}
```

Σχολιάστε το αποτέλεσμα της επόμενης κλήσης της συνάρτησης `mx()`:

```
mx(4,2,8.5,4.2,5)
```

- 15.16** Σχολιάστε τη διαφορά της παρακάτω συνάρτησης από αυτή της προηγούμενης άσκησης. Γνωρίζει η συνάρτηση το πλήθος των προαιρετικών ορισμάτων που της μεταβιβάζονται; ★★

```
int mx(int n,...)
{
    va_list v;
    int mm;
    int i,ar;
    va_start(v,n);
    mm=n;
    if (mm<0) return 0;
    do
    {
        ar=va_arg(v,int);
        if (ar>mm) mm=ar;
    } while (ar>=0);
    va_end(v);
    return mm;
}
```

Σημειώστε την τιμή που επιστρέφει η συνάρτηση `mx()` στις παρακάτω περιπτώσεις κλήσης της:

```
mx(2,8,4,5,-2)
mx(2,-2)
mx(-8)
mx(3,5,10,6,-15,20,12)
mx(2,10,20,3,4)
```

- 15.17** Να γραφεί μια συνάρτηση με μεταβλητό πλήθος ορισμάτων η οποία να δέχεται ως προαιρετικά ορίσματα συμβολοσειρές. Στη συνάρτηση θα πρέπει να μεταβιβάζεται επίσης το πλήθος των προαιρετικών ορισμάτων. Η συνάρτηση θα πρέπει να εμφανίζει σε μια γραμμή τους πρώτους χαρακτήρες των συμβολοσειρών που τις μεταβιβάζονται. Να γραφεί ένα πρόγραμμα που να δείχνει τη χρήση της συνάρτησης. ★★★

Κεφάλαιο**Αναζήτηση και
ταξινόμηση**ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ

Το κεφάλαιο αυτό ασχολείται με τις «κλασικές» λειτουργίες της αναζήτησης και της ταξινόμησης σε πίνακες βασικών αλλά και προσαρμοσμένων τύπων δεδομένων.

Παρουσιάζονται διάφοροι αλγόριθμοι αναζήτησης και ταξινόμησης, καθώς και οι συναρτήσεις βιβλιοθήκης της C που εκτελούν τις παραπάνω λειτουργίες.

Ιδιαίτερη αναφορά γίνεται στις συναρτήσεις σύγκρισης, μια δυνατότητα της C που επιτρέπει στις έτοιμες συναρτήσεις βιβλιοθήκης αναζήτησης και ταξινόμησης να συμπεριφέρονται με τρόπο που καθορίζει ο προγραμματιστής.

Εισαγωγή

Η αναζήτηση και η ταξινόμηση αποτελούν τις μεγαλύτερες προκλήσεις στη διαχείριση δεδομένων.

Η αναζήτηση αφορά τον εντοπισμό μιας συγκεκριμένης τιμής σε ένα σύνολο δεδομένων, ενώ η ταξινόμηση αφορά τη διάταξη ενός συνόλου δεδομένων σε αύξουσα ή φθίνουσα σειρά.

Τα σύνολα δεδομένων σε έναν H/Y είναι αποθηκευμένα με έναν συγκεκριμένο τρόπο και καλούνται *δομές δεδομένων*. Η πιο απλή δομή δεδομένων είναι ένας πίνακας. Η δομή δεδομένων μπορεί να είναι εσωτερική (αποθηκευμένη στην κεντρική μνήμη του H/Y) –όπως είναι οι πίνακες, οι συνδεδεμένες λίστες, τα δυαδικά δένδρα κ.λπ.– ή εξωτερική, οπότε τα δεδομένα είναι αποθηκευμένα σε αρχεία. Μια δομή δεδομένων μπορεί να είναι στατική (όπως ένας πίνακας) ή δυναμική όπως οι λίστες, τα δένδρα και οι γράφοι²⁸.

Είτε πρόκειται για εσωτερικές είτε για εξωτερικές δομές, οι αλγόριθμοι που χρησιμοποιούνται για την αναζήτηση και την ταξινόμηση είναι ίδιοι. Αυτό που διαφέρει είναι ο τρόπος υλοποίησής τους για κάθε περίπτωση.

Βασικός στόχος κάθε αλγόριθμου αναζήτησης και ταξινόμησης είναι η ταχύτητα. Οι διάφοροι αλγόριθμοι που έχουν σχεδιαστεί διαφοροποιούνται μόνο ως προς την απόδοσή τους. Συνήθως οι πιο απλοί στη σύλληψη και στην υλοποίηση είναι οι λιγότερο αποδοτικοί. Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι πιο «κλασικές» μέθοδοι αναζήτησης και ταξινόμησης. Σε όλες τις περιπτώσεις, τα δεδομένα είναι αποθηκευμένα σε πίνακες. Ωστόσο, με μικρές παραλλαγές, οι ίδιες μέθοδοι μπορούν να χρησιμοποιηθούν και σε δεδομένα που είναι αποθηκευμένα σε αρχεία περιφερειακών μονάδων αποθήκευσης.

Αν έχουμε έναν πίνακα από δομές, τότε η αναζήτηση εντοπίζει ένα στοιχείο με συγκεκριμένη τιμή σε ένα ή περισσότερα πεδία, ενώ με την ταξινόμηση τα στοιχεία του πίνακα διατάσσονται με βάση κάποιο από τα επιμέρους πεδία του.

Όταν τα δεδομένα είναι αποθηκευμένα σε πιο πολύπλοκες δομές δεδομένων, όπως οι λίστες και τα δένδρα που θα συναντήσουμε στο Κεφάλαιο 18, η αναζήτηση και η ταξινόμηση υλοποιούνται με διαφορετικούς τρόπους που ποικίλουν ανάλογα με την εκάστοτε δομή δεδομένων.

Στο παρόν κεφάλαιο όλοι οι αλγόριθμοι αναζήτησης και ταξινόμησης αφορούν σε δεδομένα που είναι αποθηκευμένα σε πίνακες.

²⁸ Στο Κεφάλαιο 18 θα αναφερθούμε διεξοδικά στις δυναμικές δομές δεδομένων.

Ταξινόμηση

Κατά καιρούς έχουν αναπτυχθεί διάφοροι αλγόριθμοι ταξινόμησης (sorting) οι οποίοι βασίζονται σε διαφορετικές τεχνικές. Στην παρούσα ενότητα παρουσιάζονται οι τέσσερις πιο συχνά χρησιμοποιούμενοι αλγόριθμοι.

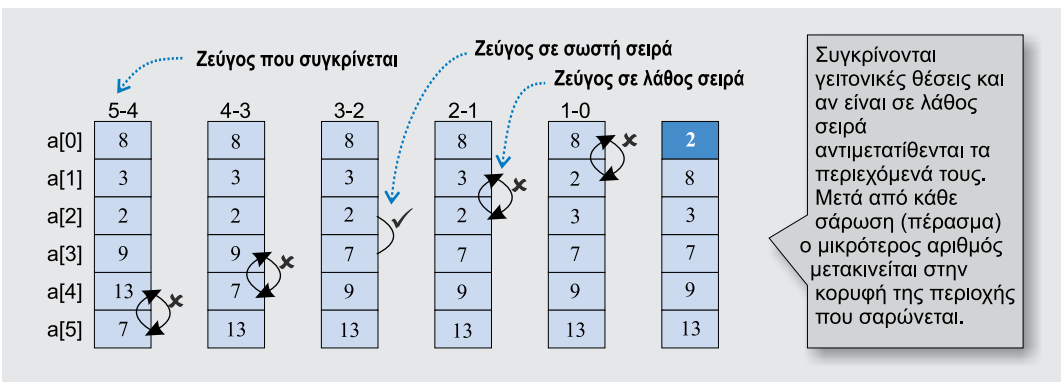
Ταξινόμηση φυσαλίδας

Η ταξινόμηση φυσαλίδας (bubble sort) είναι ίσως ο πιο απλός αλγόριθμος ταξινόμησης ο οποίος υλοποιείται με την ακόλουθη λογική:

Ξεκινάμε από το τέλος του πίνακα και ελέγχουμε ανά ζεύγη τα γειτονικά κελιά του (το κάθε κελί με το προηγούμενό του). Αν είναι σε διαφορετική σειρά από αυτήν που θέλουμε (υποθέτουμε αύξουσα), αντιμεταθέτουμε τα περιεχόμενά τους, διαφορετικά τα αφήνουμε όπως είναι. Μετά το πρώτο πέρασμα, στη κορυφή του πίνακα θα βρεθεί η μικρότερη τιμή. Το επόμενο πέρασμα (σάρωση) γίνεται για τις υπόλοιπες θέσεις του πίνακα (πλην της πρώτης,) και στο τέλος θα τοποθετηθεί στη δεύτερη θέση του πίνακα η επόμενη μικρότερη τιμή. Κάθε φορά ο έλεγχος εφαρμόζεται σε ολοένα μικρότερο τμήμα του πίνακα, με αποτέλεσμα η μικρότερη τιμή να μετακινείται στη κορυφή του τμήματος που ελέγχεται. Η διαδικασία θα σταματήσει μόλις το τμήμα που ελέγχεται αποτελείται από τις δύο τελευταίες θέσεις του πίνακα, και τότε πια ο πίνακας θα είναι ταξινομημένος. Στην παραπάνω περίπτωση πραγματοποιούσαμε τον έλεγχο $a[n] < a[n-1]$, δηλαδή ελέγχαμε αν η θέση n είχε τιμή μικρότερη από τη προηγούμενή της, και τότε κάναμε την αντιμετάθεση. Αυτό οδηγεί σε αύξουσα ταξινόμηση. Αν θέλουμε η ταξινόμηση να είναι φθίνουσα, πρέπει να αντιστρέψουμε τον έλεγχο, δηλαδή $a[n] > a[n-1]$.

Το Σχήμα 16.2 δείχνει μια τέτοια σάρωση ενός πίνακα έξι θέσεων. Στο τέλος της σάρωσης, η μικρότερη τιμή του πίνακα, το 2, μετακινείται στη κορυφή του πίνακα.

Η συνάρτηση που ακολουθεί υλοποιεί τη μέθοδο ταξινόμησης «φυσαλίδας» (bubblesort) για την αύξουσα ταξινόμηση ενός πίνακα x μίας διάστασης με n θέσεις μνήμης.



```

void bubblesort(int x[], int n)
{
    int i,k,temp;
    for (i=1;i<n;i++)
    {
        for (k=n-1;k>=i;k--)
        {
            if (x[k]<x[k-1])
            {
                temp=x[k];
                x[k]=x[k-1];
                x[k-1]=temp;
            }
        }
    }
}

```

Αν επιθυμούσαμε φθίνουσα ταξινόμηση, η συνθήκη θα ήταν $x[k]>x[k-1]$.

Αντιμέταθεση των γειτονικών θέσεων μνήμης όταν βρίσκονται σε λάθος σειρά.

Στο παράδειγμα του Σχήματος 16.3 της επόμενης σελίδας απεικονίζεται η διαδικασία ταξινόμησης ενός πίνακα [a](#) έξι θέσεων. Παρατηρούμε ότι στο τέταρτο πέρασμα του πίνακα δεν εντοπίζεται ούτε ένα ζεύγος σε λάθος σειρά. Αυτό σημαίνει ότι ο πίνακας έχει ήδη ταξινομηθεί, οπότε το πέμπτο πέρασμα είναι περιττό. Σε μεγάλους πίνακες, αυτό μπορεί να συμβεί αρκετά νωρίς στη διαδικασία ταξινόμησης, οπότε η συνέχιση της διαδικασίας μετά από αυτό το σημείο είναι εντελώς άσκοπη. Η επόμενη συνάρτηση είναι μια βελτιωμένη έκδοση της προηγούμενης, η οποία λαμβάνει υπόψη και αυτό το ενδεχόμενο. Με άλλα λόγια, αν σε κάποιο πέρασμα δεν βρεθεί ούτε ένα ζεύγος σε λάθος σειρά, ο πίνακας έχει ήδη ταξινομηθεί και η διαδικασία σταματάει.

```

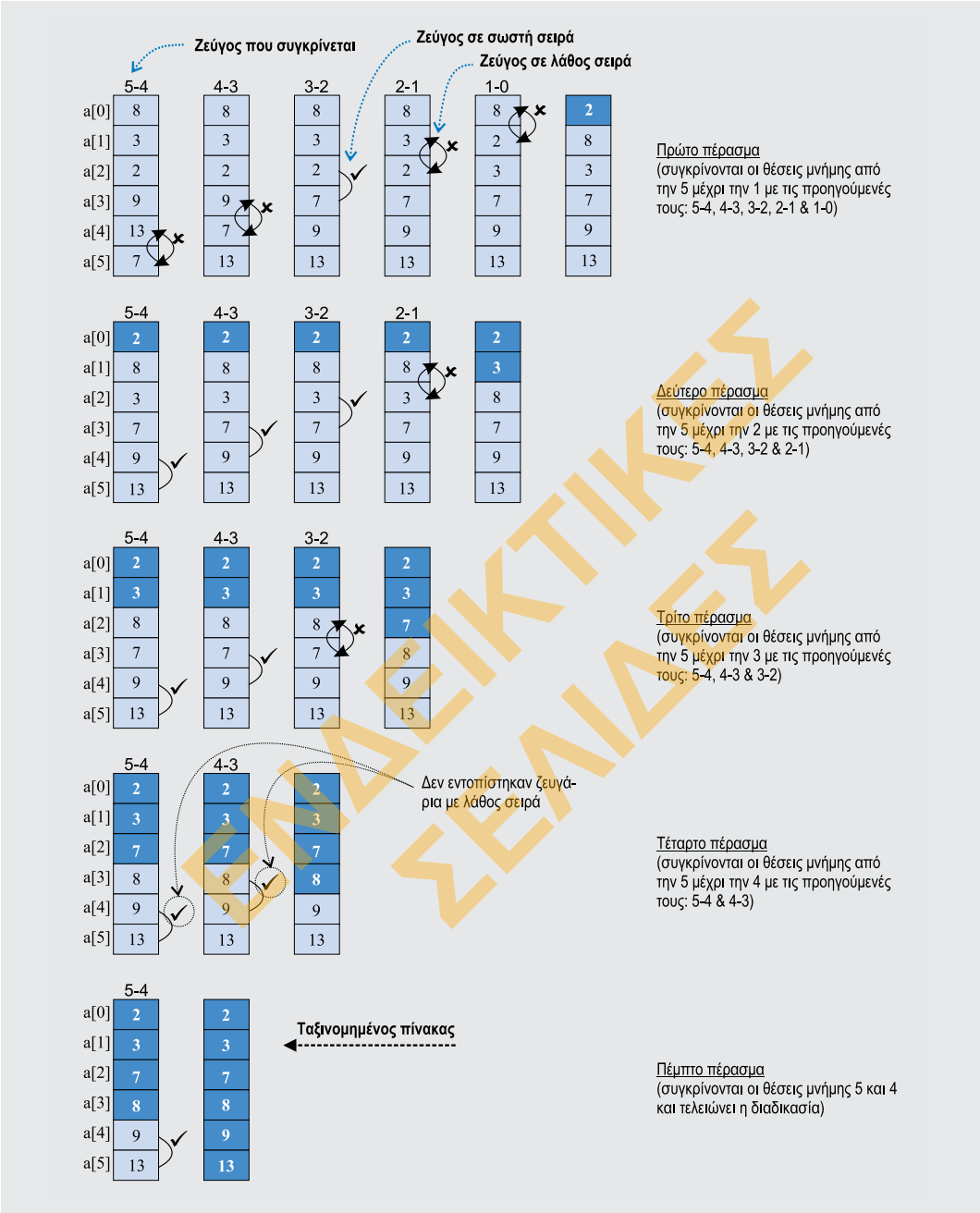
void bubblesort2(int x[],int n)
{
    int i,k,temp;
    bool found;
    for (i=1;i<n;i++)
    {
        found=false;
        for (k=n-1;k>=i;k--)
        {
            if (x[k]<x[k-1])
            {
                temp=x[k];
                x[k]=x[k-1];
                x[k-1]=temp;
                found=true;
            }
        }
        if (!found) break;
    }
}

```

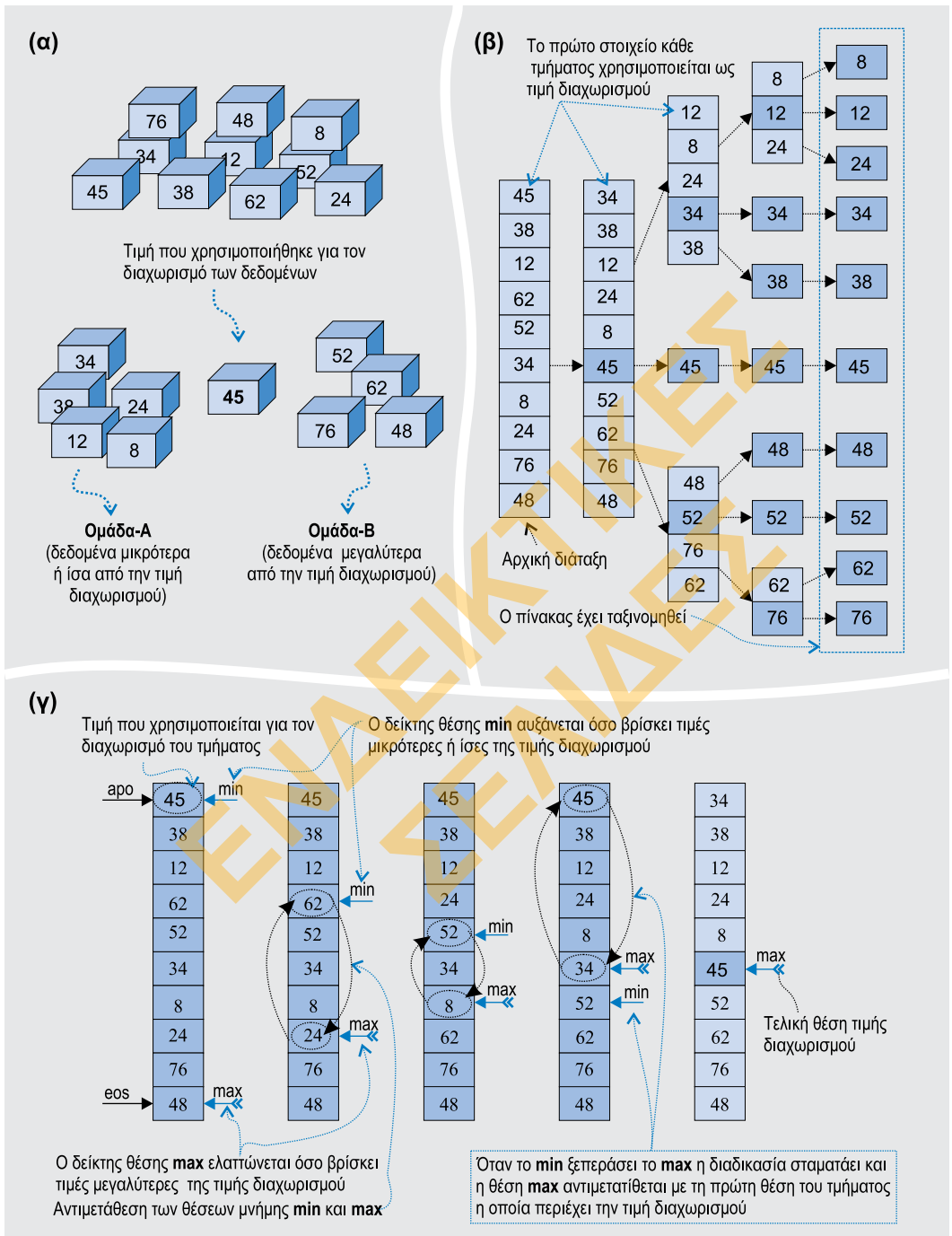
Αρχικά, στη λογική μεταβλητή **found** καταχωρίζεται η τιμή **false**.

Όταν βρεθεί έστω και ένα ζεύγος σε λάθος σειρά, στη **found** καταχωρίζεται η τιμή **true**.

Αν η **found** εξακολουθεί να έχει τιμή **false**, αυτό σημαίνει ότι στο τελευταίο πέρασμα δεν βρέθηκε κανένα ζεύγος σε λάθος θέση, οπότε ο πίνακας έχει ταξινομηθεί και δεν γίνεται άλλο πέρασμα.



Σχήμα 16.3 Ταξινόμηση πίνακα με τη μέθοδο «φυσαλίδας» (bubble sort)



Σχήμα 16.5 Η διαδικασία που ακολουθεί ο αλγόριθμος ταξινόμησης quicksort

Η συνάρτηση lfind()

Η συνάρτηση `lfind()` εντοπίζει ένα στοιχείο μέσα σε έναν πίνακα, με τη μέθοδο της σειριακής αναζήτησης. Η συνάρτηση επιστρέφει ως τιμή έναν δείκτη προς το στοιχείο που εντόπισε, διαφορετικά επιστρέφει τιμή `NULL`. Το πρωτότυπό της είναι το ακόλουθο:

void *lfind (const void *k, const void *p, size_t *n, size_t s, comparison_fn_t cf)

Οι παράμετροι της συνάρτησης έχουν ως εξής:

- Η παράμετρος `k` είναι δείκτης προς την οντότητα την οποία θέλουμε να εντοπίσουμε μέσα στον πίνακα. Στην παράμετρο αυτή μεταβιβάζεται η διεύθυνση αυτής της οντότητας. Η οντότητα αυτή πρέπει να είναι του ίδιου τύπου με τα στοιχεία του πίνακα.
- Η παράμετρος `p` είναι δείκτης προς τη θέση μνήμης από την οποία ξεκινάει ο πίνακας. Στην παράμετρο αυτή μεταβιβάζεται η διεύθυνση του πίνακα.
- Η παράμετρος `n` είναι δείκτης προς μια θέση μνήμης η οποία περιέχει το πλήθος των στοιχείων του πίνακα. Στην παράμετρο αυτή μεταβιβάζεται η διεύθυνση αυτής της θέσης μνήμης.
- Η παράμετρος `s` είναι τύπου `size_t` και σε αυτή μεταβιβάζεται το μέγεθος του τύπου των στοιχείων του πίνακα.
- Η παράμετρος `cf` είναι δείκτης (τύπου `comparison_fn`) προς τη συνάρτηση σύγκρισης που θα χρησιμοποιήσει η `lfind()` για τον εντοπισμό της οντότητας `*k`. Στην παράμετρο αυτή μεταβιβάζεται η διεύθυνση της συνάρτησης σύγκρισης.

Παρατηρούμε ότι οι δείκτες `k` και `p`, όπως και ο δείκτης που επιστρέφει η συνάρτηση, είναι τύπου `void`. Έτσι η συνάρτηση `lfind()` μπορεί να χρησιμοποιηθεί με οποιονδήποτε βασικό ή προσαρμοσμένο τύπο δεδομένων. Βέβαια η τιμή που επιστρέφει η συνάρτηση θα πρέπει να μετατραπεί σε δείκτη ίδιου τύπου με τον τύπο του πίνακα `p`, στον οποίο εκτελούμε την αναζήτηση.

Το πρόγραμμα που ακολουθεί δείχνει τη χρήση της συνάρτησης `lfind()`. Όπως και το πρόγραμμα του προηγούμενου παραδείγματος `16_comp_func2.c`, δημιουργεί έναν πίνακα ακεραίων 30 θέσεων και τον συμπληρώνει με τυχαίους αριθμούς. Ακολουθώς ζητάει από τον χρήστη έναν ακέραιο αριθμό και χρησιμοποιεί τη συνάρτηση `lfind()` για να τον εντοπίσει μέσα στον πίνακα. Ανάλογα με το αποτέλεσμα, εμφανίζει και ένα αντίστοιχο μήνυμα!

```
#include <stdio.h>
#include <stdlib.h>
// Συμπλήρωση πίνακα n θέσεων με τυχαίους αριθμούς
void fill_it(int a[], int n)
{
    ..... // ίδιος κώδικας με το παράδειγμα 16_comp_func2.c
}
```

16_lfind.c

Η συνάρτηση `qsort()`

Η συνάρτηση `qsort()` ταξινομεί έναν πίνακα. Η ταξινόμηση γίνεται με βάση τη συνάρτηση σύγκρισης που της μεταβιβάζεται. Το πρωτότυπο της είναι το ακόλουθο:

```
void qsort (void *p, size_t n, size_t s, comparison_fn_t cf)
```

Οι παράμετροι της συνάρτησης έχουν ως εξής:

- Στην παράμετρο `p` μεταβιβάζεται η διεύθυνση του πίνακα.
- Στη παράμετρο `n` μεταβιβάζεται το πλήθος των στοιχείων του πίνακα.
- Στη παράμετρο `s` μεταβιβάζεται το μέγεθος του τύπου των στοιχείων του πίνακα.
- Η παράμετρος `cf` είναι δείκτης προς τη συνάρτηση σύγκρισης που θα χρησιμοποιήσει η `qsort()` και από την οποία θα καθοριστεί ο τρόπος ταξινόμησης των στοιχείων του πίνακα.

Το πρόγραμμα που ακολουθεί δείχνει τη χρήση της συνάρτησης `qsort()`. Όπως και το πρόγραμμα του προηγούμενου παραδείγματος, δημιουργεί έναν πίνακα ακεραίων 30 θέσεων και τον συμπληρώνει με τυχαίους αριθμούς. Ακολούθως, με χρήση της `qsort()`, ταξινομεί τα στοιχεία του πίνακα, πρώτα με αύξουσα σειρά και μετά με βάση το άθροισμα των ψηφίων τους. Μετά από κάθε ταξινόμηση εμφανίζει τα περιεχόμενα του πίνακα.

Στο πρόγραμμα ορίζονται δύο συναρτήσεις σύγκρισης. Η πρώτη, `compfunc1()`, συγκρίνει δύο ακέραιους αριθμούς με βάση τη τιμή τους ενώ η δεύτερη, `compfunc2()`, συγκρίνει τους αριθμούς με βάση το άθροισμα των ψηφίων τους. Δηλαδή, ο 234 θεωρείται μικρότερος από τον αριθμό 64, διότι έχει άθροισμα ψηφίων 9 (2+3+4) ενώ ο 64 έχει άθροισμα 10 (6+4). Οι αριθμοί 64, 1234, 55, 73 και 11233 θεωρούνται ίσοι επειδή το άθροισμα των ψηφίων τους είναι ίσο με 10!

Η συνάρτηση `qsort()` καλείται δύο φορές. Σε κάθε κλήση της μεταβιβάζεται διαφορετική συνάρτηση σύγκρισης, με αποτέλεσμα η συμπεριφορά της και το αποτέλεσμα της ταξινόμησης να είναι διαφορετικό.

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

16_qsort.c

```
// Συμπλήρωση πίνακα n θέσεων με τυχαίους αριθμούς  
void fill_it(int a[], int n)  
{  
    ..... // ίδιος κώδικας με το παράδειγμα 16_comp_func2.c  
}  
  
// Εμφάνιση περιεχομένων πίνακα n θέσεων σε γραμμές των 10 στηλών  
void print_it(int a[], int n)  
{  
    ..... // ίδιος κώδικας με το παράδειγμα 16_comp_func2.c  
}
```

Παραδείγματα

Π16.1 Το επόμενο πρόγραμμα συμπληρώνει έναν πίνακα δύο διαστάσεων 100 γραμμών και 4 στηλών με τυχαίους αριθμούς και μετά τον ταξινομεί ως προς την τελευταία στήλη του σε φθίνουσα σειρά. Τέλος, εμφανίζει τα περιεχόμενα του πίνακα διατεταγμένα σε στήλες.

```
#include <stdio.h>
#include <stdlib.h>
void bubblesort2d();
int main(void)
{
    int a[100][4];
    int i, j;
    for (i=0; i<100; i++)
        for (j=0; j<4; j++)
            a[i][j]=rand();
    bubblesort2d(a, 3);
    for (i=0; i<100; i++)
    {
        for (j=0; j<4; j++)
        {
            printf("%10d", a[i][j]);
        }
        putchar('\n');
    }
    return 0;
}
void bubblesort2d(int x[][4], int m)
{
    int i, j, k, temp;
    for (i=1; i<100; i++)
    {
        for (k=99; k>=i; k--)
        {
            if (x[k][m]>x[k-1][m])
            {
                for (j=0; j<4; j++)
                {
                    temp=x[k][j];
                    x[k][j]=x[k-1][j];
                    x[k-1][j]=temp;
                }
            }
        }
    }
}
```

16_p16_1.c

Συμπλήρωση του πίνακα **a** με τυχαίους αριθμούς.

Κλήση της συνάρτησης **bubblesort2d()** που ακολουθεί, με παραμέτρους τον πίνακα και το 3 (τελευταία στήλη).

Εμφάνιση του πίνακα. Κάθε αριθμός του πίνακα εμφανίζεται σε χώρο 10 θέσεων στην οθόνη.

Αλλαγή γραμμής μετά από την εμφάνιση κάθε σειράς αριθμών.

Το **>** για φθίνουσα ταξινόμηση

Αντιμέταθεση των γειτονικών θέσεων μνήμης στην περίπτωση που είναι σε λάθος σειρά.

Συχνά λάθη

- ❖ Προσπαθούμε να εκτελέσουμε δυαδική αναζήτηση σε πίνακα που δεν είναι ταξινομημένος.
- ❖ Η ταξινόμηση συμβολοσειρών προϋποθέτει τη σύγκρισή τους. Κατά λάθος χρησιμοποιούμε τους συγκριτικούς τελεστές `>` και `<`, αντί της συνάρτησης `strcmp()` που αποτελεί τον μοναδικό τρόπο σύγκρισης δύο αλφαριθμητικών.
- ❖ Μεγάλη προσοχή πρέπει να δοθεί κατά την κλήση των συναρτήσεων βιβλιοθήκης `lfind()`, `bsearch()` και `qsort()` ώστε τα ορίσματα να είναι σύμφωνα με τα πρωτότυπά τους.

Ανασκόπηση Κεφαλαίου 16

- Αναζήτηση είναι η διαδικασία εντοπισμού ενός στοιχείου σε μια δομή δεδομένων.
- Στη σειριακή αναζήτηση, ο έλεγχος των δεδομένων γίνεται με τη σειρά από το πρώτο μέχρι το τελευταίο, έως ότου εντοπιστεί το ζητούμενο στοιχείο.
- Η δυαδική αναζήτηση μπορεί να εφαρμοστεί μόνο σε ταξινομημένα δεδομένα.
- Ταξινόμηση είναι η διαδικασία κατά την οποία ένα σύνολο δεδομένων διατάσσεται σε αύξουσα ή φθίνουσα σειρά. Η ταξινόμηση αριθμητικών δεδομένων σε αύξουσα σειρά τα διατάσσει από το μικρότερο προς το μεγαλύτερο, ενώ σε φθίνουσα σειρά τα διατάσσει αντίστροφα. Η ταξινόμηση αλφαριθμητικών δεδομένων διατάσσει τα δεδομένα με απόλυτη αλφαβητική σειρά.
- Υπάρχουν διάφοροι αλγόριθμοι ταξινόμησης, με διαφορετική πολυπλοκότητα και απόδοση.
- Ο αλγόριθμος φυσαλίδας (bubble sort) είναι ο δημοφιλέστερος και απλούστερος αλγόριθμος ταξινόμησης, ιδανικός για μικρό πλήθος δεδομένων.
- Ο αλγόριθμος γρήγορης ταξινόμησης (quick sort) είναι ο πιο πολύπλοκος, αλλά και ο πιο αποδοτικός αλγόριθμος ταξινόμησης, ιδιαίτερα όταν έχουμε μεγάλο πλήθος δεδομένων με τυχαία διασπορά τιμών.
- Η C διαθέτει τις συναρτήσεις βιβλιοθήκης `lfind()` και `bsearch()` για σειριακή και δυαδική αναζήτηση αντίστοιχα, καθώς και τη συνάρτηση `qsort()` για ταξινόμηση. Οι συναρτήσεις αυτές είναι πολυμορφικές και μπορούν να χρησιμοποιηθούν σε πίνακες οποιουδήποτε βασικού ή προσαρμοσμένου τύπου. Στις συναρτήσεις αυτές μεταβιβάζονται **συναρτήσεις σύγκρισης**, οι οποίες καθορίζουν τον τρόπο με τον οποίο θα γίνεται η αναζήτηση και η σύγκριση των στοιχείων του πίνακα.

Ασκήσεις Κεφαλαίου 16

- 16.1** Να γραφεί ένα πρόγραμμα το οποίο να ζητάει μια λέξη, να ταξινομεί τους χαρακτήρες της λέξης σε αλφαβητική σειρά, και να την εμφανίζει στην οθόνη. Για παράδειγμα, αν πληκτρολογηθεί η λέξη ΑΝΑΝΑΣ θα πρέπει να εμφανίσει τη λέξη ΑΑΑΝΝΣ. ★★
- 16.2** Να γραφεί ένα πρόγραμμα (ή μια συνάρτηση) που να ταξινομεί σε αύξουσα αλφαβητική σειρά, **ως προς τον δεύτερο χαρακτήρα τους**, τις συμβολοσειρές ενός πίνακα χαρακτήρων δύο διαστάσεων με 100 γραμμές και 40 στήλες. ★★★
- 16.3** Αν υποθέσουμε ότι στον πίνακα χαρακτήρων `lex` υπάρχει καταχωρισμένη η λέξη «ΧΑΡΟΥΜΕΝΟΣ», ποιο θα είναι το περιεχόμενο του πίνακα `lex` μετά από την εκτέλεση του επόμενου τμήματος κώδικα; ★★

```
int i=0,p1,p2;
char ch1,ch2,temp;
ch1=ch2=lex[0];
p1=p2=0;
while (lex[i]!='\0')
{
    if (ch1>lex[i])
    {
        ch1=lex[i];
        p1=i;
    }
    if (ch2<lex[i])
    {
        ch2=lex[i];
        p2=i;
    }
    i++;
}
temp=lex[0];
lex[0]=lex[p2];
lex[p2]=temp;
temp=lex[i-1];
lex[i-1]=lex[p1];
lex[p1]=temp;
```

- 16.4** Να γραφεί μια συνάρτηση η οποία θα προσθέτει έναν χαρακτήρα, παρεμβάλλοντάς τον στη σωστή του θέση, μέσα σε έναν ταξινομημένο πίνακα χαρακτήρων. Για παράδειγμα, αν ο πίνακας περιέχει τους χαρακτήρες «ΑΓΓΔΚΜΧ» και ο χαρακτήρας που πρόκειται να προστεθεί είναι ο «Ε», μετά την εκτέλεση της συνάρτησης ο πίνακας θα πρέπει να περιέχει τους χαρακτήρες «ΑΓΓΔΕΚΜΧ». Η συ-

- 16.17** Με δεδομένο το ακόλουθο τμήμα προγράμματος, να συμπληρωθεί κατάλληλα ώστε με χρήση της συνάρτησης **qsort()** να ταξινομεί τον πίνακα συμβολοσειρών ως προς το μικρό όνομα σε αύξουσα σειρά. Μετά από την ταξινόμηση να εμφανίζει τα περιεχόμενα του πίνακα. Χρησιμοποιήστε λατινικούς χαρακτήρες στο πρόγραμμά σας. ★★★

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

int main(void)
{
    char onom[5][30]={ "ΜΠΑΡΟΒΙΟΣ ΚΩΣΤΑΣ", "ΠΙΝΕΖΑ ΜΑΡΙΑ", "ΤΕΜΠΕΛΧΑΝΑΣ
                        ΒΑΣΙΛΗΣ", "ΑΡΑΧΤΟΣ ΝΙΚΟΣ", "ΣΠΑΣΙΚΛΑ ΑΝΝΑ"};
    .....
}
```

- 16.18** Με δεδομένο το ακόλουθο τμήμα προγράμματος, να συμπληρωθεί κατάλληλα ώστε με χρήση της συνάρτησης **lfind()** να αναζητάει μέσα στον πίνακα δομών μια θέση με όνομα και αριθμό μητρώου που θα πληκτρολογεί ο χρήστης. Στην περίπτωση που εντοπιστεί η θέση να εμφανίζει το πεδίο του βαθμού, διαφορετικά να εμφανίζει ένα κατάλληλο μήνυμα. Χρησιμοποιήστε λατινικούς χαρακτήρες στο πρόγραμμά σας. ★★★

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

struct stoiceia
{
    char onoma[15];
    char ar_mitr[5];
    float bathmos;
};

int main(void)
{
    .....
    struct stoiceia foitites[5]={
        "Νίκος", "ct102", 8.5,
        "Τάκης", "sa345", 9.0,
        "Αννα", "ct123", 4.5,
        "Μαρία", "sa561", 6.0,
        "Βασίλης", "ct001", 3.0};
    .....
    .....
}
```

Κεφάλαιο

Δυναμική διαχείριση μνήμης



Πολλές φορές δεν γνωρίζουμε από πριν το μέγεθος της μνήμης που θα απαιτηθεί για την αποθήκευση των δεδομένων μας, αλλά αυτό προκύπτει κατά το χρόνο εκτέλεσης του προγράμματος.

Είναι πολύ χρήσιμο επομένως να μπορούμε να δεσμεύουμε μνήμη όταν τη χρειαζόμαστε και να την αποδεσμεύουμε όταν δεν μας είναι πλέον απαραίτητη. Αυτό ονομάζεται δυναμική διαχείριση μνήμης και είναι το θέμα με το οποίο ασχολείται το παρόν κεφάλαιο.

Παρουσιάζεται η μεθοδολογία και οι συναρτήσεις της C που χρησιμοποιούνται για τη δυναμική κατανομή μνήμης, καθώς και ο τρόπος με τον οποίο χειριζόμαστε αυτή τη μνήμη.

Ιδιαίτερη αναφορά γίνεται στον τρόπο με τον οποίο δημιουργούμε και χειριζόμαστε δυναμικούς πίνακες μίας και δύο διαστάσεων.

Εισαγωγή

Μέχρι τώρα είδαμε ότι ο χειρισμός της μνήμης από τη C γίνεται είτε με τη χρήση μεταβλητών είτε με τη χρήση πινάκων. Οι μεταβλητές και οι πίνακες μπορεί να είναι τριών κατηγοριών, ανάλογα με τον τρόπο και τη θέση που δηλώνονται:

- Καθολικές
- Στατικές
- Τοπικές

Οι καθολικές και οι στατικές μεταβλητές καταλαμβάνουν συγκεκριμένο χώρο στη μνήμη και διατηρούν τα περιεχόμενά τους σε όλη τη διάρκεια του προγράμματος.

Η κατανομή μνήμης για καθολικές και στατικές μεταβλητές λέγεται **στατική κατανομή**, και οι θέσεις μνήμης δεσμεύονται κατά τη στιγμή της μεταγλώττισης.

Οι τοπικές μεταβλητές (που λέγονται επίσης *αυτόματες* μεταβλητές) δεσμεύουν χώρο στη μνήμη και διατηρούν το περιεχόμενό τους όσο διαρκεί η εμβέλειά τους. Για παράδειγμα, όταν μια τοπική μεταβλητή δηλωθεί μέσα σε μια συνάρτηση, δεσμεύεται ο κατάλληλος χώρος μνήμης ανάλογα με τον τύπο της. Όταν η συνάρτηση επιστρέφει, ο χώρος που δέσμευε η μεταβλητή αποδεσμεύεται και τα περιεχόμενά της χάνονται.

Η κατανομή μνήμης για τοπικές μεταβλητές λέγεται **αυτόματη κατανομή**, και οι θέσεις μνήμης δεσμεύονται προσωρινά (κατά τη δήλωσή τους) και αποδεσμεύονται μόλις λήξει η εμβέλειά τους.

Έτσι, οι προτάσεις

```
char a,b;  
int num[100];
```

δεσμεύουν δύο byte για τις μεταβλητές **a** και **b**, και 400 byte (4×100) για τον πίνακα **num**. Οι θέσεις αυτές δεσμεύονται τη στιγμή της δήλωσης και αποδεσμεύονται μόλις λήξει η εμβέλεια των μεταβλητών.

Αν όμως, για παράδειγμα, θέλουμε έναν πίνακα στον οποίο θα καταχωρίζουμε τους αριθμούς των αυτοκινήτων που διέρχονται από ένα σημείο παρατήρησης, το ερώτημα είναι πόσο μεγάλος πρέπει να είναι αυτός ο πίνακας.

Αν οριστεί μικρό μέγεθος πίνακα (π.χ. με 100 θέσεις), είναι πιθανό να μην επαρκεί επειδή ενδέχεται να διέλθουν περισσότερα αυτοκίνητα. Αν οριστεί αρκετά μεγάλο μέγεθος πίνακα, με πρόβλεψη (αν αυτό είναι δυνατόν) της μέγιστης κυκλοφορίας αυτοκινήτων (π.χ. με 1000 θέσεις), το πιθανότερο είναι να σπαταλήσουμε άσκοπα μνήμη την οποία δεν θα χρησιμοποιήσουμε ποτέ.

Πρέπει να έχουμε υπόψη ότι δεν είναι δυνατό να μεταβάλουμε το μέγεθος ενός πίνακα. Ο πίνακας έχει πάντα το μέγεθος που ορίζουμε με την πρόταση δήλωσής του.

Ένα άλλο πρόβλημα προκύπτει όταν δεν γνωρίζουμε το μέγεθος του πίνακα κατά τη συγγραφή του κώδικα, αλλά το γνωρίζουμε κατά την ώρα εκτέλεσης του προγράμματος. Για παράδειγμα, ίσως το πρόγραμμα να ζητάει το πλήθος των μαθητών μιας τάξης και έπειτα να δημιουργείται ένας πίνακας με τόσες θέσεις όσο είναι το πλήθος των μαθητών. Στις αρχικές εκδόσεις της C, δεν ήταν δυνατόν να δηλώσουμε έναν πίνακα που οι διαστάσεις του να καθορίζονται από μεταβλητές. Όμως, με το πρότυπο C99 και τους πίνακες μεταβλητού μήκους που ήδη έχουμε συναντήσει, προτάσεις όπως η επόμενη `int num[a];`

είναι έγκυρες, και μάλιστα υπάρχει η δυνατότητα καθορισμού των διαστάσεων, και επομένως του μεγέθους ενός πίνακα, από τιμές μεταβλητών κατά τη στιγμή της δήλωσης του πίνακα (δείτε την ενότητα *Πίνακες μεταβλητού μήκους* στη σελίδα 610). Και πάλι, όμως, από τη στιγμή που θα δηλωθεί ο πίνακας με συγκεκριμένες διαστάσεις, δεν μπορούμε να μεταβάλουμε το μέγεθός του αργότερα. Η λύση βρίσκεται στον μηχανισμό δυναμικής κατανομής μνήμης που διαθέτει η C.

Η C διαθέτει έναν μηχανισμό **δυναμικής κατανομής μνήμης** και σχετικές συναρτήσεις για την υλοποίησή του. Μπορούμε να δεσμεύουμε περισσότερη μνήμη όταν την έχουμε ανάγκη, ή να αποδεσμεύουμε μνήμη την οποία δεν χρειαζόμαστε πλέον.

Δυναμική κατανομή μνήμης

Στο διπλανό σχήμα απεικονίζεται ο τρόπος διαχείρισης της μνήμης από τη C. Ένα τμήμα μνήμης χρησιμοποιείται για την αποθήκευση του κώδικα του προγράμματος, και ένα γειτονικό τμήμα καταλαμβάνουν οι καθολικές και οι στατικές μεταβλητές.

Όπως έχει ήδη αναφερθεί, η **στοίβα** (stack) είναι ένας χώρος μνήμης στον οποίο αποθηκεύονται οι τοπικές μεταβλητές, καθώς και οι πληροφορίες που αφορούν κλήσεις των συναρτήσεων, όπως η διεύθυνση επιστροφής στον κώδικα από τον οποίον κλήθηκε μια συνάρτηση.

Η στοίβα αυξάνεται ή μειώνεται ανάλογα με τις ανάγκες του προγράμματος.

Ο υπόλοιπος αδιάθετος χώρος μνήμης μεταξύ της στοίβας και του χώρου των γενικών μεταβλητών λέγεται **σωρός** (heap).

Κατά τον χρόνο εκτέλεσης (run-time) του προγράμματος, μπορούν να διατεθούν τμήματα μνήμης (memory blocks) από τον σωρό με δυναμική κατανομή,



να χρησιμοποιηθούν, και να αποδεσμευθούν όταν δεν χρειάζονται. Οι συναρτήσεις που χρησιμοποιούνται για δυναμική κατανομή μνήμης είναι (σύμφωνα με το πρότυπο ANSI) οι `malloc()`, `calloc()`, `free()`, και `realloc()`. Οι συναρτήσεις αυτές δηλώνονται στο αρχείο κεφαλίδας `stdlib.h`.

malloc() ⇒ δεσμεύει δυναμικά συγκεκριμένο πλήθος από byte.

calloc() ⇒ δεσμεύει δυναμικά ένα πλήθος από τμήματα μνήμης και τα μηδενίζει.

free() ⇒ αποδεσμεύει χώρο που δεσμεύτηκε με δυναμικό τρόπο.

realloc() ⇒ αλλάζει το μέγεθος ενός τμήματος μνήμης που έχει δεσμευθεί με δυναμικό τρόπο.

Να θυμάστε ότι ο μηχανισμός και οι συναρτήσεις που χρησιμοποιούνται για τη δυναμική κατανομή μνήμης κάνουν εκτενή χρήση των δεικτών.

Η συνάρτηση `malloc()`

`#include <stdlib.h>`

`void *malloc(size_t size)`

Η συνάρτηση `malloc()` δεσμεύει από τον σωρό ένα τμήμα (μπλοκ) μνήμης, με μέγεθος (σε byte) ίσο με την τιμή της παραμέτρου `size`. Η `malloc()` επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του μπλοκ που δέσμευσε.

Η συνάρτηση `malloc()` διαθέτει μία παράμετρο τύπου `size_t` και η τιμή που επιστρέφει είναι ένας δείκτης (pointer) τύπου `void` στο τμήμα το οποίο δέσμευσε.

Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το μέγεθος του τμήματος, η `malloc()` επιστρέφει έναν δείκτη NULL. **Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.**

Στο παρακάτω τμήμα κώδικα φαίνεται ο συνήθης τρόπος χρήσης της `malloc()`. Στο συγκεκριμένο παράδειγμα δεσμεύεται με δυναμικό τρόπο ένας χώρος μνήμης μεγέθους 8 byte. Η διεύθυνση του χώρου που δεσμεύτηκε καταχωρίζεται στον δείκτη `p` (Σχήμα 17.1α).

```
char *p;
p=(char *)malloc(8);
if (p==NULL)
{
    puts("Δεν υπάρχει αρκετή μνήμη");
    exit(2);
}
gets(p);
putchar(p[3]);
```

Μετατροπή του δείκτη `void` που επιστρέφει η `malloc()` σε δείκτη προς δεδομένα τύπου `char`. Όπως θα δούμε αργότερα, η μετατροπή τύπου μπορεί να παραλειφθεί.

Έλεγχος του δείκτη που επέστρεψε η `malloc()`.

Καταχώριση των χαρακτήρων που πληκτρολογούνται στον χώρο μνήμης που δείχνει ο δείκτης `p`.

Εμφάνιση του 4ου χαρακτήρα από αυτούς που πληκτρολογήθηκαν. Ο χώρος που δεσμεύτηκε χρησιμοποιείται ως πίνακας, με όνομα το όνομα του δείκτη `p`!

Αν θυμηθούμε λίγο τη φιλοσοφία των δεικτών και τη σχέση τους με τους πίνακες, οι παραστάσεις `p[3]` και `*(p+3)` είναι ισοδύναμες και αναφέρονται στο τέταρτο από τα δεδομένα που ξεκινάνε από τη θέση όπου δείχνει ο δείκτης `p`.

Επομένως, μπορούμε να χρησιμοποιούμε τον χώρο μνήμης που δεσμεύτηκε ως έναν μονοδιάστατο πίνακα που έχει το όνομα του δείκτη στον οποίο έχει καταχωριστεί η διεύθυνση που επέστρεψε η `malloc()`. Παρατηρούμε ότι γίνεται μετατροπή του τύπου του δείκτη που επιστρέφει η `malloc()` σε δείκτη του τύπου δεδομένων που πρόκειται να καταχωριστούν μέσα στον χώρο μνήμης που δεσμεύτηκε. Όπως θα δούμε αργότερα, αυτό **δεν είναι απαραίτητο**, αλλά το κάνουμε για λόγους συμβατότητας με τη C++. Επίσης, θα συναντούμε τέτοιες μετατροπές τύπων πολύ συχνά σε έτοιμο κώδικα.

Η συνάρτηση `calloc()`

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

```
void *calloc(size_t num, size_t size);
```

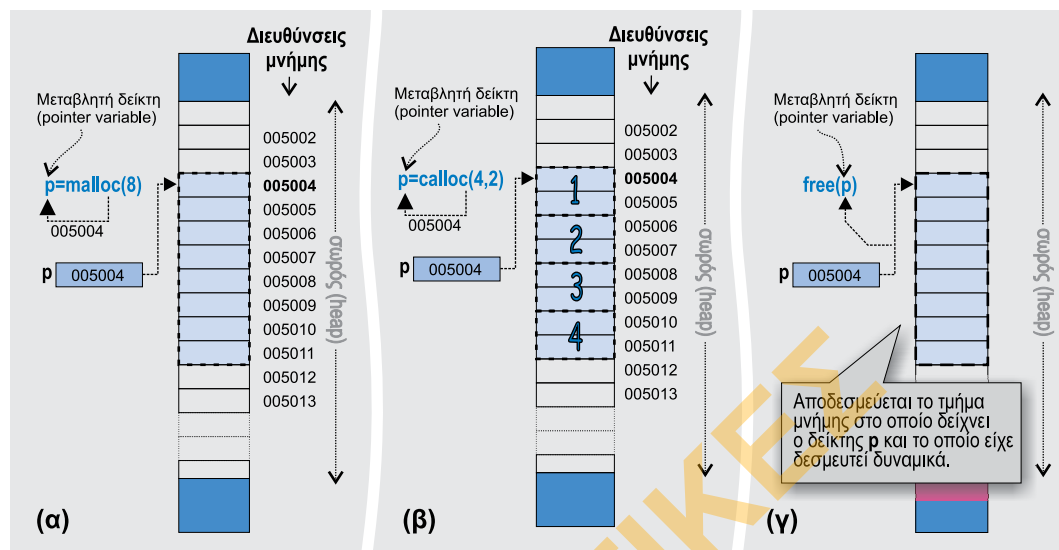
Η συνάρτηση `calloc()` δεσμεύει από τον σωρό *num* τμήματα μνήμης με μέγεθος *size* (σε byte) το καθένα. Δηλαδή, δεσμεύει χώρο μνήμης σε byte ίσο με το γινόμενο της πρώτης παραμέτρου *num* και της δεύτερης παραμέτρου *size*: *num*size*.

Η `calloc()` επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του χώρου μνήμης που δέσμευσε. Στην περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για το μέγεθος του τμήματος, η `calloc()` επιστρέφει έναν δείκτη NULL. **Πρέπει πάντα να ελέγχουμε αυτή την περίπτωση.**

Στο Σχήμα 17.1β η `calloc()` δεσμεύει χώρο μεγέθους 8 byte (4x2). Η διεύθυνση που επιστρέφει καταχωρίζεται στον δείκτη `p`.

- ☞ Με άλλα λόγια, η `calloc()` δεσμεύει την απαραίτητη μνήμη για ένα σύνολο (π.χ. ένα πίνακα) από *num* αντικείμενα μεγέθους *size*.
- ☞ Μια διαφορά μεταξύ των `malloc()` και `calloc()` είναι στον τρόπο υπολογισμού του μεγέθους του τμήματος της μνήμης που θα δεσμεύσουν. Ενώ στη `malloc()` το μέγεθος δίνεται έτοιμο σε byte, η `calloc()` το υπολογίζει πολλαπλασιάζοντας τις δύο παραμέτρους της (*num*size*).
- ☞ Επίσης, η `calloc()` **μηδενίζει** τον χώρο μνήμης που δεσμεύει, καταχωρίζοντας το 0 σε όλες τις θέσεις του.

Η συνάρτηση `calloc()` διαθέτει δύο παραμέτρους τύπου `size_t`, και η τιμή που επιστρέφει είναι ένας δείκτης τύπου `void` προς το τμήμα το οποίο δέσμευσε. Η `calloc()` μηδενίζει τον δεσμευμένο χώρο καταχωρίζοντας ως αρχική τιμή το 0 σε όλες τις θέσεις του.



Σχήμα 17.1 Δυναμική κατανομή μνήμης

Η συνάρτηση free()

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

```
void free(void *ptr);
```

Η συνάρτηση `free()` αποδεσμεύει από τον σωρό το τμήμα μνήμης στο οποίο δείχνει ο δείκτης `ptr`. Η `free()` πρέπει να κληθεί με όρισμα έναν δείκτη στον οποίο έχει ανατεθεί τιμή από προηγούμενες κλήσεις συναρτήσεων δυναμικής κατανομής, όπως η `malloc()` και η `calloc()`. Η χρήση της `free()` με έναν αυθαίρετο δείκτη θα έχει καταστροφικά αποτελέσματα. Στο παράδειγμα του Σχήματος 17.1γ, η συνάρτηση `free()` αποδεσμεύει τον χώρο των 8 byte που είχε δεσμευτεί προηγουμένως, είτε με τη συνάρτηση `malloc()` είτε με τη συνάρτηση `calloc()`, και στον οποίο δείχνει ο δείκτης `p`.

Η συνάρτηση `free()` διαθέτει μία παράμετρο που είναι δείκτης τύπου `void`, και δεν επιστρέφει τιμή. Η συνάρτηση αποδεσμεύει χώρο μνήμης που έχει δεσμευτεί με δυναμικό τρόπο.

Η συνάρτηση realloc()

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

```
void *realloc(void *ptr, size_t size);
```

Η συνάρτηση `realloc()` αυξάνει ή μειώνει το μέγεθος ενός τμήματος μνήμης που έχει ήδη δεσμευτεί από προηγούμενες κλήσεις συναρτήσεων δυναμικής κατανομής, όπως η

Συχνά λάθη

- ❖ Δεν ελέγχουμε αν η δυναμική κατανομή μνήμης ήταν επιτυχής. Πρέπει πάντα να ελέγχουμε τον δείκτη που επιστρέφει η αντίστοιχη συνάρτηση. Αν είναι NULL, αυτό σημαίνει ότι δεν υπάρχει ο απαιτούμενος χώρος μνήμης.
- ❖ Νομίζουμε ότι η συνάρτηση `realloc()` αυξάνει ή μειώνει το μέγεθος ενός δεσμευμένου χώρου μνήμης, διατηρώντας τον στην αρχική θέση του. Η αλήθεια είναι ότι η `realloc()` μπορεί να μετακινήσει τον δεσμευμένο χώρο σε τελείως διαφορετική θέση στη μνήμη.

Ανασκόπηση Κεφαλαίου 17

- Η εκχώρηση μνήμης μέσω μεταβλητών και πινάκων δεν επιτρέπει την αλλαγή του μεγέθους της δεσμευμένης μνήμης κατά τον χρόνο εκτέλεσης του προγράμματος.
- Ο μηχανισμός δυναμικής κατανομής μνήμης της C επιτρέπει στον προγραμματιστή να δεσμεύει και να αποδεσμεύει τμήματα μνήμης κατά τον χρόνο εκτέλεσης του προγράμματος.
- Όλες οι συναρτήσεις δυναμικής κατανομής μνήμης χρησιμοποιούν δείκτες προς τα τμήματα μνήμης που χειρίζονται. Η διαχείριση των τμημάτων αυτών γίνεται είτε με τη λογική των δεικτών, είτε με τη λογική των πινάκων.
- Οι συναρτήσεις δυναμικής κατανομής μνήμης χρησιμοποιούν τον σωρό (heap) για τη δέσμευση του απαιτούμενου χώρου μνήμης. Ο σωρός, όπως και γενικότερα η μνήμη ενός H/Y, έχει περιορισμένο μέγεθος, επομένως υπάρχει περίπτωση η συνάρτηση δυναμικής κατανομής μνήμης να μην μπορέσει να δεσμεύσει το απαιτούμενο μέγεθος μνήμης.
- Οι πίνακες μεταβλητού μήκους (VLA), που έχουμε συναντήσει στο Κεφάλαιο 12, είναι μια μορφή δυναμικής κατανομής μνήμης, με τη διαφορά ότι δεν μπορούμε ούτε να τροποποιήσουμε το μέγεθός τους, ούτε να αποδεσμεύσουμε τον χώρο που δεσμεύουν. Όμως, όπως και οι συναρτήσεις δυναμικής κατανομής μνήμης, δεσμεύουν χώρο κατά τον χρόνο εκτέλεσης του προγράμματος (run time), και όχι κατά τη μεταγλώττισή του (compile time).
- Για να δημιουργήσουμε δυναμικούς πίνακες δύο ή περισσότερων διαστάσεων, στη πραγματικότητα δημιουργούμε πίνακες δεικτών προς δυναμικά δεσμευμένους χώρους. Οι χώροι αυτοί δεν είναι συνεχόμενοι, όπως συμβαίνει με πίνακες που δημιουργούνται με στατική κατανομή μνήμης!

Ασκήσεις Κεφαλαίου 17

- 17.1** Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει το πλήθος των αριθμών που πρόκειται να καταχωριστούν σε έναν πίνακα ακεραίων, να δημιουργεί έναν πίνακα με τόσες θέσεις μνήμης ώστε να χωράει **ακριβώς** το πλήθος των δεδομένων, και μετά να διαβάζει και να καταχωρίζει αυτούς τους αριθμούς μέσα στον πίνακα. Να χρησιμοποιηθεί η τεχνική της δυναμικής κατανομής μνήμης. ★★
- 17.2** Να συμπληρωθεί το πρόγραμμα της προηγούμενης άσκησης ώστε μετά από την καταχώριση των αριθμών να αυξάνει το μέγεθος του πίνακα στο διπλάσιο του αρχικού του μεγέθους. ★
- 17.3** Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει το πλήθος των μαθητών ενός σχολείου και να δεσμεύει τόσο χώρο μνήμης όσο χρειάζεται για την καταχώριση των στοιχείων των μαθητών, όπως φαίνεται στη δομή **stoxeia**. Αμέσως μετά να ζητάει και να αποθηκεύει τα **στοιχεία** για τον 1ο μαθητή. ★★★
- ```
struct stoxeia
{
 char onoma[15];
 char address[20];
 char thl[13];
 int ilikia;
};
```
- 17.4** Να συμπληρωθεί το πρόγραμμα του παραδείγματος Π17.2 ώστε μετά από την καταχώριση των 10 αριθμών να αποδεσμεύει τον μισό από τον δεσμευμένο χώρο μνήμης (τις 50 από τις 100 θέσεις), χωρίς όμως να χαθούν τα δεδομένα που έχουμε καταχωρίσει. ★
- 17.5** Ποια από τα επόμενα αληθεύουν; ★
- ☐ Ο πίνακας αποτελεί μια περίπτωση δυναμικής κατανομής μνήμης.
  - ☐ Στη δυναμική κατανομή μνήμης υπάρχει περίπτωση να μην είναι δυνατή η δέσμευση της ζητούμενης ποσότητας μνήμης.
  - ☐ Η **malloc(5,100)** δεσμεύει 500 byte μνήμης.
  - ☐ Η **free(100)** αποδεσμεύει 100 byte από τη μνήμη.
  - ☐ Η **calloc()** και η **malloc()** επιτελούν σχεδόν την ίδια λειτουργία.
- 17.6** Μέσα σε ένα αρχείο με όνομα **times.txt** είναι καταχωρισμένες διάφορες τιμές προϊόντων. Ο πρώτος αριθμός του αρχείου υποδεικνύει το πλήθος των τιμών που ακολουθούν. Να γραφεί ένα πρόγραμμα το οποίο να δεσμεύει τον απαραίτητο χώρο για την καταχώριση των τιμών του αρχείου, να διαβάζει τις τιμές από το αρχείο, και να τις καταχωρίζει στον χώρο που δέσμευσε.
- Τέλος, το πρόγραμμα να εμφανίζει τον μέσο όρο των τιμών, καθώς και όλες τις τιμές που διάβασε σε **φθίνουσα** σειρά. ★★★

**17.7** Να γραφεί ένα πρόγραμμα το οποίο αρχικά να διαβάζει το πλήθος των φράσεων που πρόκειται να καταχωριστούν σε έναν πίνακα χαρακτήρων 2Δ. Στη συνέχεια να δημιουργεί δυναμικά τον πίνακα χαρακτήρων για τη καταχώριση αυτών των φράσεων, με την προϋπόθεση ότι καμία από τις φράσεις δεν θα διαθέτει περισσότερους από 50 χαρακτήρες (συμπεριλαμβανομένου και του χαρακτήρα τερματισμού \0). Το πρόγραμμα θα πρέπει να ζητάει από τον χρήστη να πληκτρολογήσει τις φράσεις και να τις καταχωρίζει στον πίνακα. Τέλος, θα πρέπει να εμφανίζει τις φράσεις που πληκτρολογήθηκαν και να αποδεσμεύει τον χώρο που καταλάμβανε ο δυναμικός πίνακας. ★ ★ ★

**17.8** Εξηγήστε τη λειτουργία του παρακάτω προγράμματος. ★ ★

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
 char *k;
 int i,a=0;
 k=calloc(20,sizeof(char));
 fgets(k,20,stdin);
 for (i=0;i<strlen(k);i++)
 if (k[i]!='\n') a++;
 puts(k);
 printf("%d\n",a);
 free(k);
 return 0;
}
```

**17.9** Μελετήστε προσεκτικά το πρόγραμμα του παραδείγματος Π17.5. Συμπληρώστε το κατάλληλα ώστε να ζητάει από τον χρήστη τον κωδικό ενός φοιτητή και να απαντάει σε ποια ή σε ποιες ομάδες είναι μέλος ο συγκεκριμένος φοιτητής. Στην περίπτωση που ο φοιτητής δεν ανήκει σε κάποια ομάδα, να εμφανίζεται κατάλληλο μήνυμα. ★ ★

**17.10** Διάφοροι ερευνητές εργάζονται κάποιες μέρες ο καθένας και λαμβάνουν μετρήσεις θερμοκρασίας για ένα πείραμα. Δεν γνωρίζουμε το πλήθος των ερευνητών, των ημερών που εργάζεται ο καθένας, και των μετρήσεων που λαμβάνει κάθε ημέρα. Να γραφεί ένα πρόγραμμα το οποίο θα ζητάει από τον χρήστη το πλήθος των ερευνητών, τις μέρες που εργάστηκε ο κάθε ερευνητής, και το πλήθος των μετρήσεων που έκανε κάθε ημέρα. Στη συνέχεια πρέπει να ζητάει τις μετρήσεις ανά ερευνητή και ημέρα και να τις καταχωρίζει σε κατάλληλο χώρο μνήμης. Τέλος, το πρόγραμμα πρέπει να εμφανίζει τη μέση θερμοκρασία ανά ερευνητή και ημέρα. ★ ★ ★

## Κεφάλαιο

# 18

### Δυναμικές δομές δεδομένων

ΕΝΔΕΙΚΤΙΚΕΝ  
ΣΕΛΙΔΕΝ



Οι δυναμικές δομές δεδομένων δίνουν τη δυνατότητα αποθήκευσης δεδομένων σε δομές που βασίζονται στη δυναμική κατανομή μνήμης.

Στο κεφάλαιο αυτό αναλύονται οι πιο συχνά χρησιμοποιούμενες δυναμικές δομές, όπως οι συνδεδεμένες λίστες, τα δυαδικά δένδρα αναζήτησης και οι γράφοι.

Παρουσιάζονται αναλυτικά και υλοποιούνται διάφοροι αλγόριθμοι διάσχισης και επεξεργασίας αυτών των δομών.

## Εισαγωγή

Μέχρι στιγμής, η μοναδική δομή δεδομένων που έχουμε γνωρίσει είναι ο πίνακας. Οι πίνακες αποτελούν **στατικές** δομές καταχώρισης δεδομένων και έχουν ένα μεγάλο μειονέκτημα: πρέπει να καθορίζουμε το μέγεθός τους στην πρόταση δήλωσής τους, χωρίς να έχουμε τη δυνατότητα να το αλλάξουμε (αυξήσουμε ή μειώσουμε) κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Για τον λόγο αυτό, συνήθως χρησιμοποιούμε πίνακες μεγάλου μεγέθους ώστε να είμαστε σίγουροι ότι θα μπορούμε να αποθηκεύσουμε το μεγαλύτερο πιθανό πλήθος δεδομένων. Όμως η λύση αυτή οδηγεί σε υπερκατανάλωση μνήμης, η οποία τις περισσότερες φορές δεν είναι απαραίτητη. Επίσης, αν θέλουμε να διατηρούμε τα δεδομένα του πίνακα συνεχώς ταξινομημένα, η διαδικασία επαναταξινόμησης (ή παρεμβολής) μετά από την προσθήκη κάθε νέου στοιχείου είναι αρκετά χρονοβόρα, ιδιαίτερα σε πίνακες μεγάλου μεγέθους. Η λύση στα προηγούμενα προβλήματα είναι η χρήση δυναμικών δομών, οι οποίες δεσμεύουν μνήμη όταν τη χρειάζονται και την αποδεσμεύουν όταν δεν τους είναι πια απαραίτητη. Η χρήση δυναμικών δομών, εκτός από το ότι λύνουν το πρόβλημα της κατανομής μνήμης, ορισμένες από αυτές μας δίνουν τη δυνατότητα να διατηρούμε τα δεδομένα συνεχώς διατεταγμένα, ώστε να μην είναι πια απαραίτητες οι χρονοβόρες διαδικασίες αναζήτησης και ταξινόμησης.

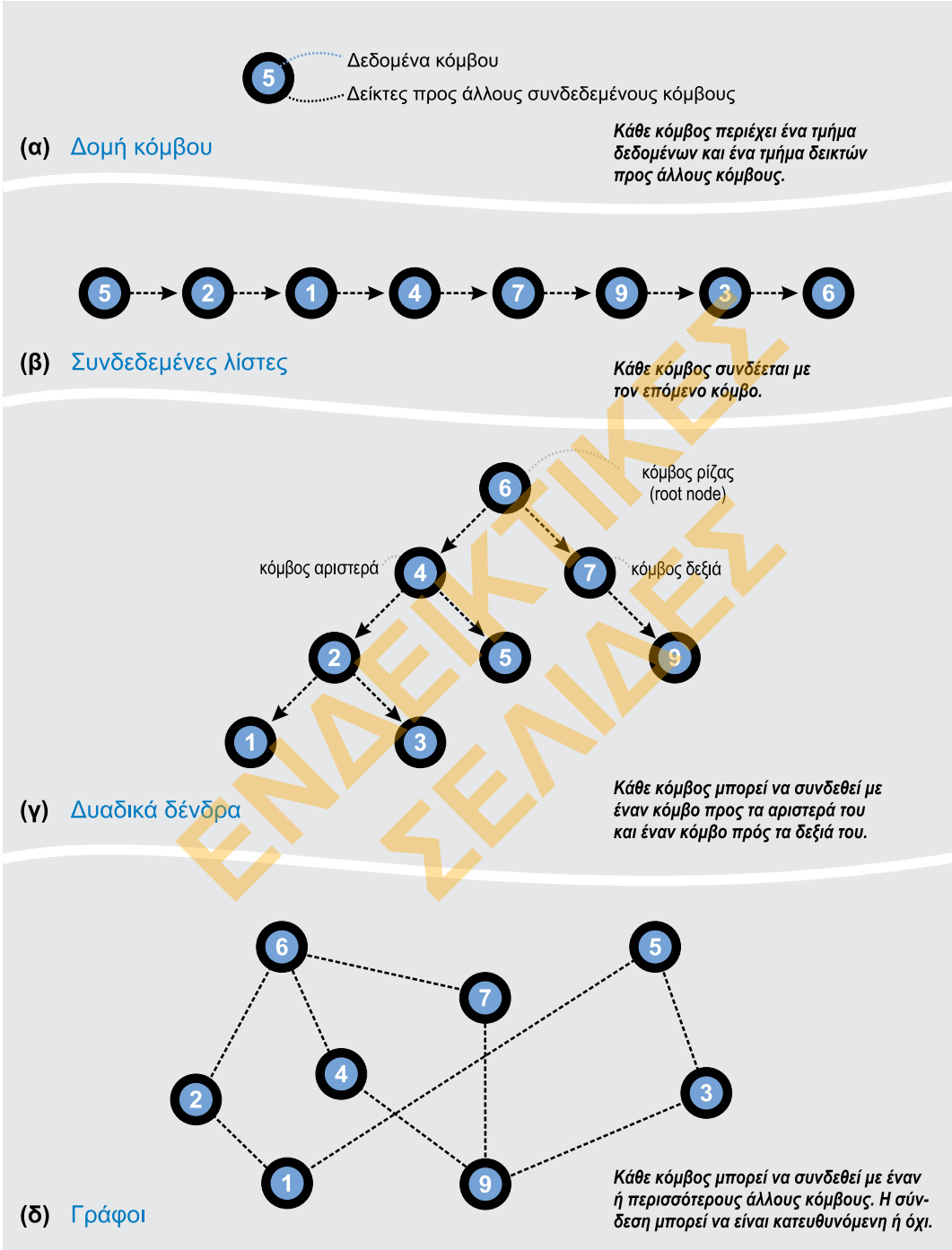
Το κεφάλαιο αυτό δεν φιλοδοξεί να καλύψει όλες τις δυνατότητες των δυναμικών δομών δεδομένων, οι οποίες από μόνες τους θα απαιτούσαν ένα ολόκληρο βιβλίο για να εξηγηθούν επαρκώς. Θα εξετάσουμε όμως την υλοποίηση και τη διαχείριση των πιο συχνά χρησιμοποιούμενων δυναμικών δομών δεδομένων, οι οποίες είναι:

- Οι συνδεδεμένες λίστες (linked lists)
- Τα δυαδικά δένδρα αναζήτησης (binary search trees)
- Οι γράφοι (graphs)

Το δομικό στοιχείο των δυναμικών δομών δεδομένων είναι ο **κόμβος**. Μια δομή δεδομένων αποτελείται από πολλούς συνδεδεμένους κόμβους. Κάθε κόμβος είναι ένα σύνολο πληροφοριών που περιέχει τόσο τα ίδια τα δεδομένα, όσο και πληροφορίες για τον επόμενο ή τους επόμενους κόμβους (Σχήμα 18.1α). Για τη δέσμευση της μνήμης που απαιτεί ο κάθε κόμβος χρησιμοποιείται ο μηχανισμός δυναμικής διαχείρισης μνήμης. Η διάταξη των κόμβων διαφέρει ανάλογα με το είδος της δομής δεδομένων (συνδεδεμένη λίστα, δυαδικό δένδρο, ή γράφος).

Στις συνδεδεμένες λίστες, κάθε κόμβος έχει έναν προηγούμενο και έναν επόμενο κόμβο, εκτός φυσικά από τον πρώτο και τον τελευταίο (Σχήμα 18.1β). Κάθε κόμβος της λίστας, πέρα από το τμήμα δεδομένων του, διαθέτει και έναν δείκτη προς τον επόμενο κόμβο. Η δομή των δυαδικών δένδρων διαφέρει. Κάθε κόμβος του δένδρου μπορεί να συνδεθεί με έναν κόμβο προς τα αριστερά του και έναν κόμβο προς τα δεξιά του (Σχήμα 18.1γ).



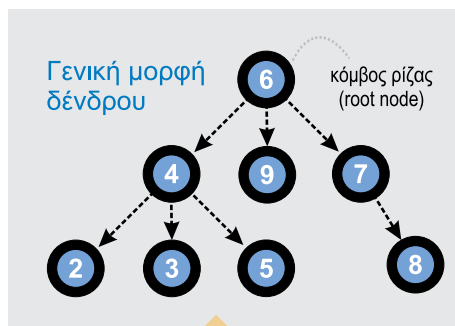


Σχήμα 18.1 Δυναμικές δομές δεδομένων

Ο πρώτος κόμβος του δένδρου λέγεται κόμβος «ρίζας». Κάθε κόμβος ενός δυαδικού δένδρου, πέρα από το τμήμα δεδομένων του, διαθέτει και δύο δείκτες: έναν προς τον επόμενο κόμβο αριστερά και έναν προς τον επόμενο κόμβο δεξιά.

Τα δυαδικά δένδρα είναι μια υποπερίπτωση της γενικής μορφής δομής δένδρου. Σε αυτή τη γενική μορφή, κάθε κόμβος μπορεί να συνδέεται με κανέναν, έναν, ή και περισσότερους κόμβους χαμηλότερα στην ιεραρχία. Η γενική αυτή μορφή δεν εξετάζεται ξεχωριστά στο βιβλίο, αλλά καλύπτεται από τη δομή γράφου.

Σε έναν γράφο, κάθε κόμβος μπορεί να είναι συνδεδεμένος με κανέναν, έναν, ή περισσότερους άλλους κόμβους (Σχήμα 18.1δ). Κάθε κόμβος ενός γράφου, πέρα από το τμήμα δεδομένων του, διαθέτει και έναν ή περισσότερους δείκτες προς τους άλλους κόμβους με τους οποίους είναι συνδεδεμένος. Στους γράφους δεν υπάρχει κάποια ιεραρχική δομή αλλά κάθε κόμβος μπορεί να συνδέεται με οποιονδήποτε άλλο κόμβο.



### Πότε χρησιμοποιούνται οι διαφορετικές δυναμικές δομές

Η δομή των δεδομένων μας καθορίζει και το είδος της δυναμικής δομής δεδομένων που είναι η καταλληλότερη για την αποθήκευση και τη διαχείρισή τους.

Όταν χειριζόμαστε δεδομένα που δεν έχουν κάποια ιδιαίτερη συσχέτιση μεταξύ τους και θα μπορούσαμε να τα αποθηκεύσουμε και σε έναν απλό πίνακα, τότε η καταλληλότερη δυναμική δομή είναι η συνδεδεμένη λίστα. Για παράδειγμα, τα βιβλία μιας βιβλιοθήκης, τα στοιχεία μιας ομάδας ατόμων, κ.λπ.

Όταν τα δεδομένα μας έχουν μια ιεραρχική δομή, όπως για παράδειγμα η δομή μιας εταιρείας ή ένα οικογενειακό δένδρο, τότε η δυναμική δομή δένδρου είναι η καταλληλότερη. Η ιδιαίτερη περίπτωση των δυαδικών δένδρων χρησιμοποιείται συνήθως για τη δημιουργία ευρετηρίων αναζήτησης, και γι' αυτό τον λόγο ονομάζονται δυαδικά δένδρα αναζήτησης.

Όταν τα δεδομένα μας συνδέονται μεταξύ τους, αλλά όχι με γραμμικό ή ιεραρχικό τρόπο, τότε η δυναμική δομή που ενδείκνυται είναι ο γράφος. Ένα οδικό δίκτυο μεταξύ πόλεων, ο παγκόσμιος Ιστός (web), οι σταθμοί του μετρό, είναι παραδείγματα γράφων. Αν παρατηρήσετε τη δομή ενός γράφου θα διαπιστώσετε ότι μπορεί να καλύψει και τη γενική μορφή της δομής δένδρου, με τον περιορισμό ότι κάθε κόμβος θα συνδέεται μόνο με κόμβους που βρίσκονται χαμηλότερα στην ιεραρχία.

Η δομή και η συσχέτιση των δεδομένων μας καθορίζει το είδος της καταλληλότερης δυναμικής δομής δεδομένων για την αποθήκευση και διαχείρισή τους

## Μια διαφορετική υλοποίηση στοίβας και ουράς

Στις υλοποιήσεις που είδαμε στις προηγούμενες παραγράφους, οι χειριστές τόσο μιας στοίβας όσο και μιας ουράς ήταν καθολικές μεταβλητές στις οποίες είχαν πρόσβαση όλες οι συναρτήσεις που τις διαχειρίζονται. Αυτό δημιουργεί προβλήματα ιδιαίτερα στην περίπτωση που θέλουμε να διατηρούμε ταυτόχρονα περισσότερες από μία στοίβες ή ουρές. Στην υλοποίηση που παρουσιάζεται στην παρούσα ενότητα, η στοίβα και η ουρά αποτελούν αυτοτελείς οντότητες οι οποίες προσδιορίζονται από ξεχωριστές δομές που περιέχουν τους δείκτες των χειριστών τους.

Το πρόγραμμα που ακολουθεί υλοποιεί μια στοίβα με τη λογική που αναφέρθηκε. Μια στοίβα προσδιορίζεται από τη δομή **stack** η οποία περιέχει μόνο ένα πεδίο, τον χειριστή της στοίβας, δηλαδή έναν δείκτη προς τον κόμβο κορυφής της. Η δημιουργία μιας στοίβας γίνεται τώρα ως εξής: Δημιουργούμε έναν δείκτη προς δομές τύπου **stack** και σε αυτόν καταχωρίζουμε τη διεύθυνση μιας μεταβλητής δομής **stack** την οποία δεσμεύουμε δυναμικά:

```
struct stack *st=(struct stack*) malloc(sizeof(struct stack));
```

Τώρα η διαχείριση της δομής γίνεται μέσω αυτού του δείκτη. Για παράδειγμα, η πρόταση **st->top=NULL** καταχωρίζει την τιμή **NULL** στον χειριστή της στοίβας στην οποία δείχνει ο δείκτης **st**, γεγονός που υποδηλώνει μια κενή στοίβα.

Στο πρόγραμμα υλοποίησης, η στοίβα δημιουργείται μέσω της συνάρτησης **create\_stack()**, με τη μέθοδο που μόλις αναφέραμε, η οποία επιστρέφει ως τιμή έναν δείκτη σε αυτή τη δομή στοίβας. Στις υπόλοιπες συναρτήσεις που διαχειρίζονται τη στοίβα μεταβιβάζεται αυτός ο δείκτης μέσω του οποίου οι συναρτήσεις αυτές μπορούν να προσπελάζουν τον χειριστή της στοίβας.

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

18\_new\_stack.c

```
// Η δομή των κόμβων της στοίβας
// Κάθε κόμβος περιέχει το τμήμα δεδομένων του και έναν δείκτη προς τον επόμενο κόμβο
```

```
struct stack_node
{
 int data;
 struct stack_node *next;
};
```

```
// Η δομή stack αναπαριστά μια στοίβα
// Το μόνο που περιέχει είναι ένα πεδίο-δείκτης προς τον κόμβο κορυφής της στοίβας
// Ο δείκτης αυτός είναι ο χειριστής της στοίβας
```

```
struct stack
{
 struct stack_node *top;
};
```

```

void pop_from_stack(struct stack *s);
bool push_in_stack(struct stack *s, int d);
bool is_stack_empty(struct stack *s);
int get_stack_top(struct stack *s);
struct stack *create_stack();
void display_stack(struct stack *s);

```

```

int main(void)

```

```

{
 char ch;
 int timi;
 struct stack *mystack=create_stack();
 while (1)
 {
 printf("\n0->Εξοδος 1->Ωθηση 2->Απόθηση 3->Εμφάνιση κορυφής
 4->Εμφάνιση όλων\n");
 ch=getch();
 switch (ch)
 {
 case '0':
 exit(0);
 case '1':
 printf("Δώσε έναν αριθμό:");
 scanf("%d",&timi);
 if (!push_in_stack(mystack,timi)) puts("Δεν υπάρχει μνήμη");
 break;
 case '2':
 pop_from_stack(mystack);
 break;
 case '3':
 if (!is_stack_empty(mystack))
 printf("%d\n",get_stack_top(mystack));
 else
 printf("Κενή στοίβα\n");
 break;
 case '4':
 if (!is_stack_empty(mystack))
 display_stack(mystack);
 else
 printf("Κενή στοίβα\n");
 break;
 default:
 puts("Λάθος πλήκτρο");
 break;
 }
 }
 return 0;
}

```

Δημιουργεί μια κενή στοίβα. Ο δείκτης **mystack** δείχνει σε αυτή τη δομή στοίβας. Μέσω του δείκτη αυτού μπορούμε να προσπελάσουμε τον χειριστή της στοίβας (**mystack->top**)!

Ωθεί στη στοίβα που προσδιορίζεται από τον δείκτη **mystack**, την τιμή που έδωσε ο χρήστης. Ο δείκτης **mystack** μεταβιβάζεται σε όλες τις συναρτήσεις χειρισμού της στοίβας.

Απωθεί το κορυφαίο στοιχείο από τη στοίβα που προσδιορίζεται από τον δείκτη **mystack**.

Εμφανίζει το κορυφαίο στοιχείο της στοίβας.

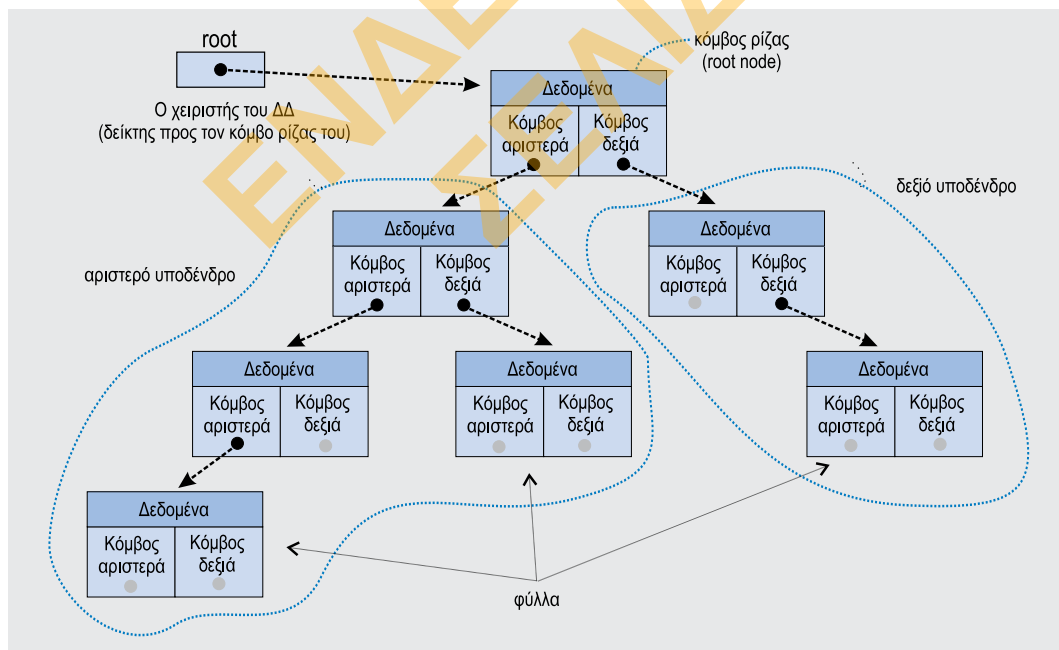
Εμφανίζει όλα τα στοιχεία της στοίβας που προσδιορίζεται από τον δείκτη **mystack**.

## Δυαδικά δένδρα

Το δυαδικό δένδρο περιλαμβάνει έναν κόμβο που αποτελεί τη **ρίζα** του δένδρου (root node), ένα **υποδένδρο αριστερά**, και ένα **υποδένδρο δεξιά**. Κάθε υποδένδρο είναι και αυτό ένα δυαδικό δένδρο, με τον κόμβο ρίζας του και το αριστερό και το δεξί υποδένδρο του (Σχήμα 18.6). Κάθε κόμβος του δυαδικού δένδρου περιέχει, εκτός από το τμήμα δεδομένων, έναν δείκτη προς το αριστερό υποδένδρο του και έναν δείκτη προς το δεξί. Οι κόμβοι οι οποίοι δεν έχουν υποδένδρο ονομάζονται **φύλλα** του δυαδικού δένδρου.

Η πρόσβαση σε ένα δυαδικό δένδρο γίνεται μέσω ενός δείκτη ο οποίος περιέχει τη διεύθυνση του κόμβου ρίζας (root node) του δυαδικού δένδρου.

- Ένα δυαδικό δένδρο μπορεί να είναι κενό, δηλαδή να μην περιέχει κανέναν κόμβο. Τότε ο δείκτης ο οποίος δείχνει στον κόμβο ρίζας θα πρέπει να έχει τιμή NULL.
- Όταν ένα δυαδικό δένδρο αποτελείται από έναν μόνο κόμβο, τότε αυτός είναι ο κόμβος ρίζας.
- Κάθε κόμβος του δένδρου προσπελάζεται μέσω μιας **μοναδικής** διαδρομής, η οποία ξεκινάει από τον κόμβο ρίζας του.
- Η διαχείριση ενός δυαδικού δένδρου γίνεται μέσω ενός δείκτη προς τον κόμβο ρίζας του (root), ο οποίος αποτελεί και τον χειριστή του δυαδικού δένδρου.

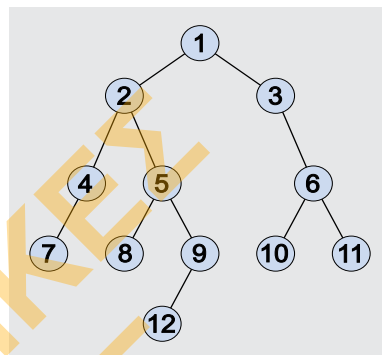


Σχήμα 18.6 Η δομή ενός δυαδικού δένδρου

- Ένα δυαδικό δένδρο έχει **μέγεθος**, το οποίο ορίζεται από το πλήθος των κόμβων που περιέχει.
- Κάθε κόμβος του δένδρου έχει **βάθος**, το οποίο ορίζεται από το πλήθος των κόμβων που απέχει από τον κόμβο ρίζας του δένδρου.
- Ένα δυαδικό δένδρο έχει **ύψος**, το οποίο είναι το βάθος του βαθύτερου κόμβου του, δηλαδή του κόμβου που απέχει περισσότερο από τη ρίζα του δένδρου.

Ας μετετήσουμε το δυαδικό δένδρο του διπλανού σχήματος, με κόμβο ρίζας τον κόμβο 1.

- Το **μέγεθος** του είναι 12, διότι αποτελείται από 12 κόμβους.
- Το **ύψος** του είναι 4, διότι ο πλέον απομακρυσμένος κόμβος του (ο 12) απέχει τέσσερις κόμβους από τη ρίζα του (1).
- Το **βάθος** του κόμβου 1 είναι 0, ενώ του κόμβου 6 είναι 2 (απέχει δύο κόμβους από τη ρίζα).



### Δυαδικά δένδρα αναζήτησης (ΔΔΑ)

Τα δυαδικά δένδρα αναζήτησης (binary search trees – BST) είναι μια συγκεκριμένη κατηγορία δυαδικών δένδρων, και είναι πολύ χρήσιμα στον προγραμματισμό επειδή είναι εξαιρετικά αποδοτικά στη διαδικασία αναζήτησης των κόμβων που περιέχουν. Οι τεχνικές και οι αλγόριθμοι που θα αναπτυχθούν στη συνέχεια του κεφαλαίου αφορούν δυαδικά δένδρα αναζήτησης.

Η λογική με την οποία είναι διατεταγμένοι οι κόμβοι ενός δυαδικού δένδρου αναζήτησης είναι η εξής:

- Ένα ή περισσότερα από τα δεδομένα του κόμβου αποτελούν το **κλειδί** με βάση το οποίο θα είναι διατεταγμένο το δυαδικό δένδρο.
- Ο κόμβος στον οποίο δείχνει ο **δείκτης αριστερά** έχει τιμή κλειδιού μικρότερη (αριθμητικά ή αλφαβητικά) από την τιμή του κλειδιού του τρέχοντος κόμβου.
- Ο κόμβος στον οποίο δείχνει ο **δείκτης δεξιά** έχει τιμή κλειδιού μεγαλύτερη ή ίση από την τιμή του κλειδιού του τρέχοντος κόμβου.

Το παράδειγμα που ακολουθεί δείχνει ένα δυαδικό δένδρο αναζήτησης (ΔΔΑ) στο οποίο είναι καταχωρισμένα τα στοιχεία κάποιων ατόμων. Το **κλειδί** που χρησιμοποιείται για τη διάταξη του συγκεκριμένου δυαδικού δένδρου είναι το **όνομα** του ατόμου (Σχήμα 18.7).

Ο πρώτος κόμβος που προστέθηκε στο δυαδικό δένδρο είναι ο κόμβος με τιμή κλειδιού ΝΙΚΟΣ, ο οποίος αποτελεί τη ρίζα του δυαδικού δένδρου.

## Γράφοι

Αν και η γενική θεωρία των γράφων βασίζεται σε μαθηματικά μοντέλα, η ενότητα αυτή αποτελεί μια μικρή εισαγωγή στους γράφους χωρίς να εμπλέκεται καθόλου με μαθηματικές έννοιες. Για μια ολοκληρωμένη μελέτη των γράφων θα χρειαζόταν ένα ολόκληρο βιβλίο, όμως κάτι τέτοιο ξεφεύγει από τον σκοπό του παρόντος βιβλίου. Η ενότητα αυτή καλύπτει κάποιες από τις κατηγορίες γράφων και τις βασικές δυνατότητες επεξεργασίας τους. Δίνει μια πρώτη εικόνα για τον τρόπο αναπαράστασης και υλοποίησης απλών γράφων, χρησιμοποιώντας δυναμική κατανομή μνήμης. Όπως αναφέρθηκε στην αρχή του κεφαλαίου, ένας γράφος είναι μια διάταξη κόμβων όπου κάθε κόμβος μπορεί να είναι συνδεδεμένος με κανέναν, έναν, ή περισσότερους άλλους κόμβους. Οι κόμβοι ενός γράφου λέγονται **κορυφές** (vertices) και οι γραμμές σύνδεσης μεταξύ των κορυφών λέγονται **ακμές** (edges). Ένας γράφος μπορεί να είναι μη κατευθυνόμενος ή κατευθυνόμενος.

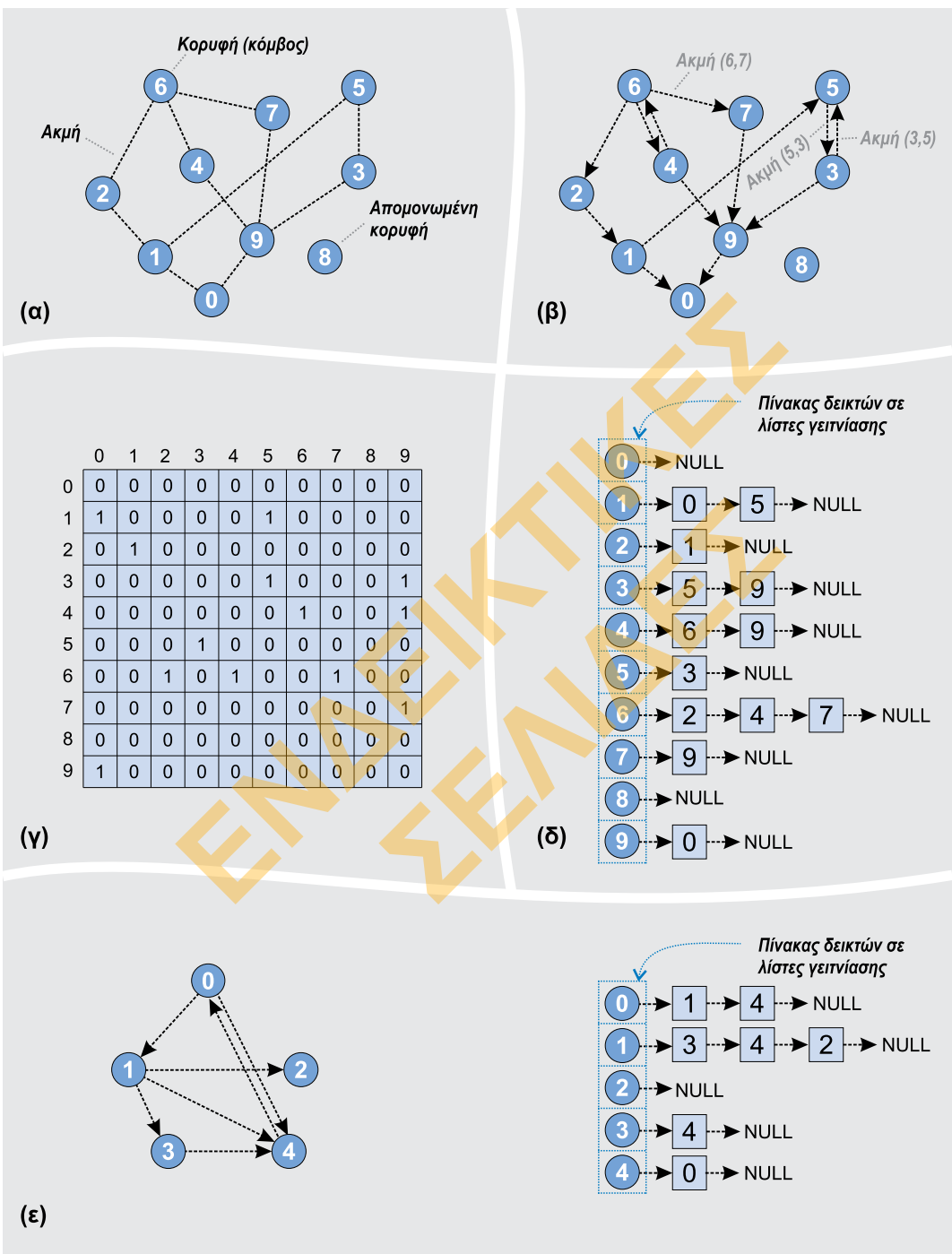
Το Σχήμα 18.13α δείχνει έναν μη κατευθυνόμενο γράφο (undirected graph). Αυτό σημαίνει ότι από τη στιγμή που υπάρχει μια ακμή μεταξύ δύο κορυφών, η σύνδεση τους είναι αμφίδρομη. Για παράδειγμα, από την κορυφή 4 μπορούμε να μεταβούμε στην κορυφή 9 και το αντίστροφο. Η κορυφή 8 δεν συνδέεται με καμία άλλη κορυφή, αν και είναι τμήμα του γράφου. Μια τέτοια κορυφή ονομάζεται **απομονωμένη** (isolated vertex).

Στο Σχήμα 18.13β εμφανίζεται ένας κατευθυνόμενος γράφος (directed graph). Κάθε ακμή ενός κατευθυνόμενου γράφου είναι μόνο μίας κατεύθυνσης. Για παράδειγμα, από την κορυφή 4 μπορούμε να μεταβούμε στην κορυφή 9 αλλά όχι και το αντίστροφο. Όταν σε έναν κατευθυνόμενο γράφο θέλουμε αμφίδρομη επικοινωνία μεταξύ δύο κορυφών, τότε αυτές συνδέονται με δύο ακμές αντιθέτων κατευθύνσεων. Για παράδειγμα, οι κορυφές 6 και 4, καθώς και οι κορυφές 5 και 3, συνδέονται μεταξύ τους με ένα ζεύγος ακμών διαφορετικής κατεύθυνσης. Ένας μη κατευθυνόμενος γράφος μπορεί να αναπαρασταθεί από έναν κατευθυνόμενο γράφο όπου οι συνδεδεμένες κορυφές συνδέονται με ζεύγη ακμών αντίθετης κατεύθυνσης.

Ένας γράφος είναι μια γενική περίπτωση ενός συνόλου κορυφών-κόμβων που μπορεί να συνδέονται μεταξύ τους με οποιονδήποτε τρόπο. Επομένως, οι λίστες και τα δένδρα μπορούν να θεωρηθούν ως ειδικές περιπτώσεις γράφων.

## Βασικές έννοιες γράφων

Στη θεωρία των γράφων, ένας γράφος  $G$  ορίζεται ως  $G=(V,E)$  όπου  $V$  είναι το σύνολο όλων των κορυφών  $\{v_1, v_2, \dots, v_n\}$  και  $E$  είναι το σύνολο όλων των ακμών του γράφου  $\{e_1, e_2, \dots, e_m\}$ . Κάθε ακμή του γράφου είναι ένα διμελές σύνολο κορυφών. Σε έναν μη κατευθυνόμενο γράφο, μια ακμή  $e=\{v_1, v_2\}$  συνδέει τις κορυφές  $v_1$  και  $v_2$  και είναι αμφίδρομη. Οι κορυφές  $v_1$  και  $v_2$  ονομάζονται **άκρα** της ακμής.



Σχήμα 18.13 Αναπαράσταση γράφων



Σε έναν κατευθυνόμενο γράφο, μια ακμή  $e=\{v_1,v_2\}$  αποτελεί ένα διατεταγμένο ζεύγος κορυφών και έχει κατεύθυνση από την κορυφή  $v_1$  προς την κορυφή  $v_2$ . Η κορυφή  $v_1$  ονομάζεται **ουρά** (tail) και η  $v_2$  **κεφαλή** (head) της ακμής. Επίσης, σε μια τέτοια ακμή η κορυφή  $v_1$  λέγεται **προκάτοχος** (predecessor) της  $v_2$  και η  $v_2$  **διάδοχος** (successor) της κορυφής  $v_1$ .

Δύο κορυφές που συνδέονται με μια ακμή ονομάζονται συνδεδεμένες ή γειτονικές. Επίσης, η γειτονιά (neighbourhood) μιας κορυφής είναι το σύνολο των γειτονικών της κορυφών. Για παράδειγμα, στο Σχήμα 18.13α, οι κορυφές 9 και 4 είναι γειτονικές και η γειτονιά της κορυφής 9 αποτελείται από τις κορυφές 0, 4, 7 και 3!

Κάθε κορυφή ενός γράφου χαρακτηρίζεται από τον **βαθμό** της (degree). Ο βαθμός μιας κορυφής προσδιορίζεται από το σύνολο των ακμών που συνδέονται με τη συγκεκριμένη κορυφή και συμβολίζεται με  $\text{deg}(V_n)$ , όπου  $V_n$  είναι μια οποιαδήποτε κορυφή του γράφου. Με βάση το Σχήμα 18.13α, σε έναν μη κατευθυνόμενο γράφο ο βαθμός μιας κορυφής  $V$  είναι το πλήθος των ακμών που συνδέονται με τη συγκεκριμένη κορυφή.

Σε έναν κατευθυνόμενο γράφο, κάθε κορυφή διαθέτει έναν **βαθμό εισόδου** (indegree) και έναν **βαθμό εξόδου** (outdegree). Ο βαθμός εισόδου μιας κορυφής  $V$  είναι το πλήθος των ακμών που καταλήγουν σε αυτή την κορυφή, δηλαδή των ακμών των οποίων η κορυφή  $V$  αποτελεί την κεφαλή τους, και συμβολίζεται με  $\text{deg}^+(V)$ . Ο βαθμός εξόδου μιας κορυφής  $V$  είναι το πλήθος των ακμών που αναχωρούν από αυτή την κορυφή, δηλαδή των οποίων η κορυφή  $V$  αποτελεί την ουρά τους και συμβολίζεται με  $\text{deg}^-(V)$ .

Ο πίνακας που ακολουθεί δείχνει τον βαθμό κάθε κορυφής για τα παραδείγματα γράφων των Σχημάτων 18.13α και β. Παρατηρούμε ότι μια απομονωμένη κορυφή έχει βαθμό 0.

| Κορυφή | Βαθμός<br>μη κατευθυνόμενου<br>γράφου 18.13α | Βαθμός εισόδου<br>κατευθυνόμενου<br>γράφου 18.13β | Βαθμός εξόδου<br>κατευθυνόμενου<br>γράφου 18.13β |
|--------|----------------------------------------------|---------------------------------------------------|--------------------------------------------------|
| 0      | 2                                            | 2                                                 | 0                                                |
| 1      | 3                                            | 1                                                 | 2                                                |
| 2      | 2                                            | 1                                                 | 1                                                |
| 3      | 2                                            | 1                                                 | 2                                                |
| 4      | 2                                            | 1                                                 | 2                                                |
| 5      | 2                                            | 2                                                 | 1                                                |
| 6      | 3                                            | 1                                                 | 3                                                |
| 7      | 2                                            | 1                                                 | 1                                                |
| 8      | 0                                            | 0                                                 | 0                                                |
| 9      | 4                                            | 3                                                 | 1                                                |

Μια **διαδρομή** (path) σε έναν γράφο αποτελείται από ένα σύνολο ακμών μέσω των οποίων μπορούμε να μεταβούμε από μια κορυφή του γράφου σε μια άλλη. Στη διαδρομή δεν επιτρέπεται να περάσουμε από μια κορυφή δύο φορές, εκτός αν η κορυφή αφετηρίας και η κορυφή άφιξης συμπίπτουν. Σε έναν κατευθυνόμενο γράφο, οι ακμές πρέπει να έχουν την ίδια φορά, από την κορυφή προέλευσης προς την κορυφή προορισμού. Για παράδειγμα, στον μη κατευθυνόμενο γράφο του Σχήματος 18.13α, μεταξύ των κορυφών 6 και 3 υπάρχουν οι πιθανές διαδρομές 6-2-1-5-3, 6-2-1-0-9-3, 6-4-9-3, 6-7-9-3, 6-7-9-0-1-5-3 και 6-4-9-0-1-5-3, ενώ στον κατευθυνόμενο γράφο του Σχήματος 18.13β υπάρχει μόνο η διαδρομή  $6 \rightarrow 2 \rightarrow 1 \rightarrow 5 \rightarrow 3$ !

**Απόσταση** (distance) μεταξύ δύο κορυφών ενός γράφου είναι το πλήθος των ακμών στη συντομότερη διαδρομή που τις συνδέει. Για παράδειγμα, στον γράφο του Σχήματος 18.13α, η απόσταση μεταξύ των κορυφών 6 και 3 είναι 3, ενώ η απόσταση μεταξύ των ίδιων κορυφών στον κατευθυνόμενο γράφο του Σχήματος 18.13β είναι 4.

Κατά τη **διάσχιση** ενός γράφου (graph traversal), επισκεπτόμαστε όλες τις προσπελάσιμες κορυφές του. Η διάσχιση ενός γράφου αναφέρεται και ως *αναζήτηση γράφου* (graph search) και ξεκινάει από μια κορυφή του.

### Αναπαράσταση γράφων με πίνακες γειτνίασης

Η αναπαράσταση ενός γράφου περιλαμβάνει κυρίως τη διασύνδεση των κορυφών του. Ο πιο εύκολος και άμεσος τρόπος αναπαράστασης είναι μέσω ενός πίνακα **γειτνίασης** (ή μήτρα γειτνίασης – adjacency matrix). Ένας πίνακας γειτνίασης είναι ένας τετραγωνικός πίνακας διαστάσεων  $N \times N$ , όπου  $N$  είναι το πλήθος των κορυφών του γράφου. Οι γραμμές και οι στήλες του πίνακα αντιστοιχούν στην κάθε κορυφή του. Μια θέση  $A[i,j]$  του πίνακα γειτνίασης έχει τιμή 1 στην περίπτωση που υπάρχει κάποια ακμή  $e = \{i,j\}$ , διαφορετικά έχει τιμή 0. Σε έναν μη κατευθυνόμενο γράφο, ο πίνακας γειτνίασης είναι συμμετρικός ως προς την κύρια διαγώνιό του, και επομένως ισχύει  $A[i,j] = A[j,i]$ . Σε έναν κατευθυνόμενο γράφο, οι γραμμές του πίνακα γειτνίασης συμβολίζουν τις κορυφές-ουρές των ακμών, ενώ οι στήλες τις κορυφές-κεφαλές των ακμών.

Ο πίνακας γειτνίασης του Σχήματος 18.13γ αναπαριστά τον κατευθυνόμενο γράφο του Σχήματος 18.13β. Για παράδειγμα, βλέπουμε ότι στη θέση  $A[4,9]$  υπάρχει η μονάδα, που υποδηλώνει μια κατευθυνόμενη ακμή με ουρά την κορυφή 4 και κεφαλή την κορυφή 9. Η θέση  $A[9,4]$  περιέχει το 0, υποδεικνύοντας ότι δεν υπάρχει αντίστροφη διαδρομή από την κορυφή 9 στην κορυφή 4. Οι θέσεις  $A[6,4]$  και  $A[4,6]$  περιέχουν και οι δύο τη μονάδα που υποδεικνύει την ύπαρξη αμφίδρομης διαδρομής μεταξύ των κορυφών 4 και 6, η οποία υλοποιείται με δύο ακμές αντίθετης φοράς. Για την απομονωμένη κορυφή 8, η αντίστοιχη γραμμή και στήλη του πίνακα γειτνίασης έχει όλο μηδενικά!

Παρόλο που η αναπαράσταση γράφων με πίνακες γειτνίασης μας επιτρέπει να προσπελάσουμε άμεσα και να χειριζόμαστε εύκολα τους γράφους, έχει πολλά μειονεκτήματα, ιδιαίτερα όταν οι γράφοι περιέχουν πολλές κορυφές και λίγες ακμές.

## Διάσχιση γράφων

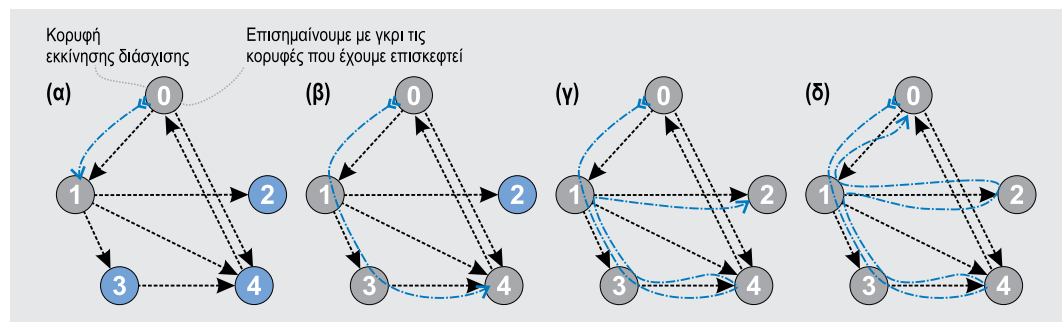
Η διάσχιση ενός γράφου (graph traversal) είναι η διαδικασία επίσκεψης όλων των προσπελάσιμων κορυφών του γράφου, με αφετηρία κάποια από τις κορυφές του. Είναι προφανές ότι δεν είναι δυνατόν να επισκεφτούμε μια απομονωμένη κορυφή ενός γράφου, εκτός αν η διάσχιση ξεκινάει από αυτήν. Επίσης, σε έναν κατευθυνόμενο γράφο, από μια κορυφή μπορούν να προσπελαστούν μόνο οι διάδοχες κορυφές της.

Η διάσχιση ενός γράφου λέγεται και αναζήτηση γράφου (graph search), και μπορεί να γίνει με δύο μεθόδους οι οποίες υλοποιούνται από αντίστοιχους αλγόριθμους: Τη **διάσχιση κατά βάθος** (Depth First Search – DFS) και τη **διάσχιση κατά πλάτος** (Breadth First Search – BFS).

### Διάσχιση κατά βάθος

Η διάσχιση πάντα ξεκινάει από κάποια κορυφή του γράφου, την οποία θα ονομάσουμε κορυφή Α. Η διάσχιση κατά βάθος ακολουθεί αρχικά κάποια ακμή του γράφου, ξεκινώντας από την κορυφή Α, και ακολουθεί τις ακμές με τη σωστή φορά μέχρι να φτάσει σε κάποια κορυφή του η οποία είτε δεν έχει διάδοχες κορυφές είτε τις έχουμε ήδη επισκεφτεί. Φανταστείτε ότι καταγράφουμε σε μια λίστα όλες τις κορυφές που επισκεφτόμαστε με τη σειρά. Μόλις φτάσουμε σε μια κορυφή η οποία δεν έχει διάδοχες κορυφές ή τις έχουμε ήδη επισκεφτεί, τότε τη διαγράφουμε από τη λίστα, γιατί δεν θα μας απασχολήσει στο μέλλον, και πηγαίνουμε μια κορυφή πίσω, δηλαδή στη τελευταία που έχουμε σημειώσει στη λίστα μας, και εκτελούμε την ίδια διαδικασία. Σταματάμε όταν δεν έχουμε καμία εκκρεμότητα στη λίστα, που σημαίνει ότι έχουμε επισκεφτεί όλες τις κορυφές.

Ας προσπαθήσουμε να κάνουμε αυτό που περιγράψαμε προηγουμένως, στον γνωστό γράφο του Σχήματος 18.14, ξεκινώντας από την κορυφή 0. Έτσι, ακολουθούμε μια τυχαία διαδρομή, για παράδειγμα ξεκινώντας από την πρώτη ακμή για να φτάσουμε στην κορυφή 1. Μέχρι τώρα έχουμε επισκεφτεί δύο κορυφές, την 0 και την 1, τις οποίες επισημαίνουμε χρωματίζοντάς τις με γκρι χρώμα (Σχήμα 18.14α) και τις σημειώνουμε στη λίστα μας με τη σειρά 0, 1. Συνεχίζουμε ακολουθώντας πάλι την πρώτη ακμή, μεταβαίνουμε στην κορυφή 3, και στη συνέχεια μεταβαίνουμε με την ίδια λογική στην κορυφή 4.

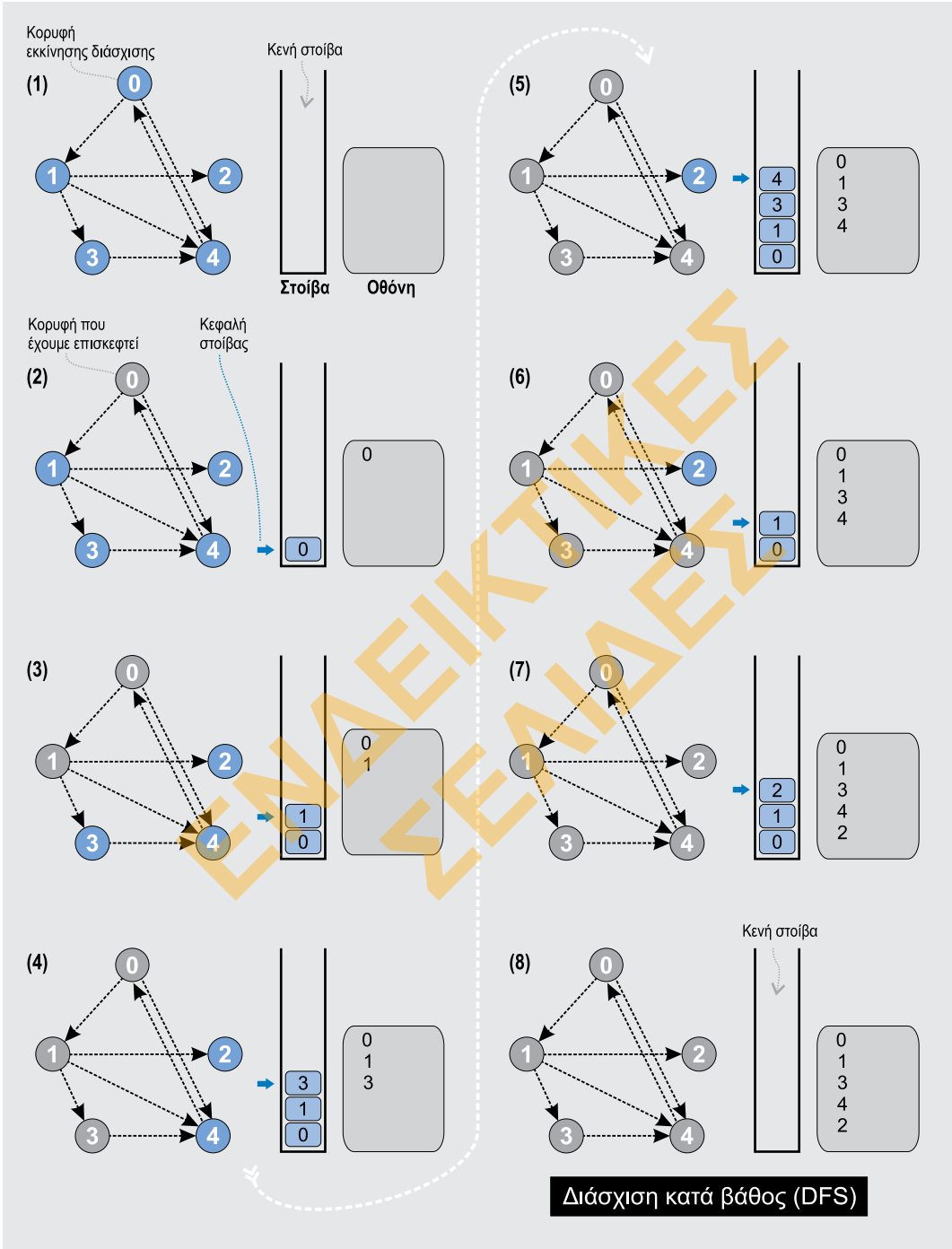


Σχήμα 18.14 Διάσχιση γράφου κατά βάθος

Επισημαίνουμε ότι έχουμε επισκεφτεί αυτές τις δύο κορυφές (Σχήμα 18.14β) και τις προσθέτουμε στη λίστα μας η οποία τώρα περιέχει με τη σειρά τις κορυφές 0, 1, 3 και 4. Τώρα, από την κορυφή 4 δεν μπορούμε να πάμε σε καμία άλλη κορυφή. Η μόνη διάδοχη κορυφή είναι η κορυφή 0, την οποία όμως έχουμε ήδη επισκεφτεί. Οπότε, διαγράφουμε την κορυφή 4 από τη λίστα, διότι δεν υπάρχει καμία εκκρεμότητα με αυτή, και μεταφερόμαστε μια κορυφή προς τα πίσω, στην τελευταία που υπάρχει στη λίστα, δηλαδή στην κορυφή 3, και ακολουθούμε την ίδια διαδικασία. Η κορυφή 3 έχει μόνο μία διάδοχη κορυφή, την 4, την οποία όμως έχουμε επισκεφτεί! Έτσι, αφού δεν έχουμε άλλη εκκρεμότητα με την 3, τη διαγράφουμε και αυτή από τη λίστα, και επιστρέφουμε στην τελευταία από αυτές που παρέμειναν, δηλαδή στην κορυφή 1. Τώρα η κορυφή 1 έχει τρεις διάδοχες κορυφές: την 3, την 4, και τη 2. Την 3 και την 4 τις έχουμε ήδη επισκεφτεί, οπότε μεταβαίνουμε στην κορυφή 2 (Σχήμα 18.14γ). Την προσθέτουμε και αυτή στη λίστα και την επισημαίνουμε ότι την έχουμε επισκεφτεί. Η λίστα μας τώρα περιέχει με τη σειρά τις κορυφές 0, 1, και 2. Τώρα η κορυφή 2 δεν έχει διάδοχη κορυφή, οπότε τη διαγράφουμε και αυτή από τη λίστα στην οποία παραμένουν μόνο οι κορυφές 0 και 1. Κάνουμε ένα βήμα πίσω στη τελευταία που έχουμε στη λίστα και πάμε πάλι στην κορυφή 1. Από την κορυφή 1 όμως δεν μπορούμε να μεταβούμε σε καμία άλλη κορυφή επειδή έχουμε επισκεφτεί και τις τρεις διάδοχες κορυφές της, έτσι τη διαγράφουμε και αυτή από τη λίστα η οποία τώρα περιέχει ως εκκρεμότητα μόνο την κορυφή 0. Οι διάδοχες κορυφές της κορυφής 0 είναι η 1 και η 4 που τις έχουμε επισκεφτεί και τις δύο (Σχήμα 18.14δ). Δεν υπάρχει λοιπόν εκκρεμότητα ούτε με την κορυφή 0, οπότε τη διαγράφουμε και αυτή από τη λίστα και η διαδικασία ολοκληρώνεται. Επομένως, η σειρά επίσκεψης των κορυφών κατά βάθος ήταν: 0-1-3-4-2. Τα αποτελέσματα της διάσχισης κατά βάθος εξαρτώνται από τη διάδοχη κορυφή που επιλέγουμε κάθε φορά για να προχωρήσουμε βαθύτερα στον γράφο. Για παράδειγμα, αν από την κορυφή 1 πηγαίναμε στην 4 και όχι στην 3, η σειρά θα ήταν 0-1-4-3-2 ή 0-1-4-2-3.

Ας προσεγγίσουμε τώρα το πρόβλημα προγραμματιστικά. Τον ρόλο της έντυπης λίστας, θα αναλάβει μια δομή στοίβας! Όταν θέλουμε να προσθέσουμε μια κορυφή απλά την ωθούμε στη στοίβα, όταν θέλουμε να ανακτήσουμε την τελευταία βλέπουμε την κεφαλή της στοίβας, και όταν θέλουμε να διαγράψουμε την τελευταία κάνουμε απώθηση. Συγκεκριμένα, ο αλγόριθμος που ακολουθείται περιγράφεται από τα παρακάτω βήματα:

1. Επισκέψου την κορυφή εκκίνησης. Επισήμανέ την ότι την έχεις επισκεφτεί, εμφάνισε την και ώθησέ την στη στοίβα.
2. Θεώρησε ως τρέχουσα κορυφή αυτή που βρίσκεται στην κεφαλή της στοίβας. Εντόπισε μια διάδοχη της τρέχουσας κορυφής, την οποία να μην έχεις επισκεφτεί. Επισήμανέ την ότι την έχεις επισκεφτεί, εμφάνισέ την και ώθησέ την στη στοίβα. Επανάλαβε το βήμα 2 μέχρι να φτάσεις σε μια κορυφή που να μην έχει διάδοχες κορυφές ή να μην τις έχεις επισκεφτεί.
3. Απώθησε μια κορυφή από τη στοίβα.
4. Αν η στοίβα δεν είναι κενή, επέστρεψε στο βήμα 2, διαφορετικά τερμάτισε την επαναληπτική διαδικασία – η διάσχιση κατά βάθος ολοκληρώθηκε.



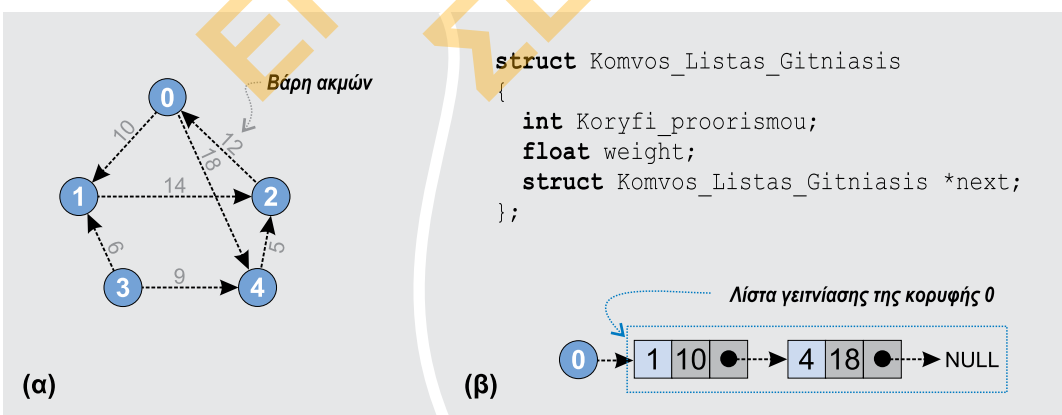
Σχήμα 18.15 Διάσχιση γράφου κατά βάθος με χρήση στοίβας

# Σταθμισμένοι γράφοι

Στους σταθμισμένους γράφους (weighted graphs), κάθε ακμή τους σχετίζεται και με μια αριθμητική τιμή. Η αριθμητική τιμή αυτή αποτελεί το κόστος που πρέπει να καταβάλουμε αν επιλέξουμε αυτή την ακμή στη διαδρομή μας. Το κόστος αυτό λέγεται βάρος (weight) της ακμής. Για παράδειγμα, το βάρος μιας ακμής μπορεί να είναι η απόσταση μεταξύ δύο πόλεων σε ένα γράφο οδικού δικτύου, ο χρόνος σε έναν γράφο του μετρό, η ταχύτητα μετάδοσης δεδομένων (bandwidth) σε έναν γράφο ευρυζωνικού δικτύου, κ.λπ.

Στον σταθμισμένο γράφο του Σχήματος 18.20α εμφανίζονται τα βάρη των ακμών του. Ας υποθέσουμε ότι τα βάρη αυτά αναφέρονται σε αποστάσεις (σε χιλιόμετρα) μεταξύ των κορυφών του. Είναι προφανές ότι οι κόμβοι των λιστών γειτνίασης του γράφου θα πρέπει να προσαρμοστούν ώστε να περιέχουν και το βάρος κάθε ακμής του. Το Σχήμα 18.20β δείχνει τη νέα μορφή που αποκτά η δομή ενός κόμβου, καθώς και την κατάλληλα διαμορφωμένη λίστα γειτνίασης της κορυφής 0. Το πιο χαρακτηριστικό πρόβλημα σε έναν σταθμισμένο γράφο είναι η εύρεση μιας διαδρομής με ελάχιστο κόστος (πρόβλημα συντομότερης διαδρομής – shortest path problem). Για παράδειγμα, στον γράφο του Σχήματος 18.20α, από την κορυφή 3 στη κορυφή 2 υπάρχουν οι δυνατές διαδρομές 3-1-2 και 3-4-2. Αυτή όμως που έχει το λιγότερο κόστος (μικρότερο άθροισμα βαρών) είναι η δεύτερη με συνολικό βάρος 14 (έναντι 20 της πρώτης).

Έχουν αναπτυχθεί διάφοροι αλγόριθμοι για την επίλυση του προβλήματος της συντομότερης διαδρομής. Ο πιο δημοφιλής είναι ο αλγόριθμός του **Dijkstra**, ο οποίος εντοπίζει είτε τη συντομότερη διαδρομή μεταξύ δύο κορυφών είτε τη συντομότερη διαδρομή μεταξύ μιας κορυφής αφετηρίας και όλων των υπόλοιπων κορυφών του γράφου<sup>31</sup>.



Σχήμα 18.20 Σταθμισμένος γράφος με βάρη στις ακμές του

<sup>31</sup> Στον αλγόριθμο αυτό βασίζεται το πρωτόκολλο OSPF (Open Shortest Path First – προτεραιότητα συντομότερης διαδρομής), το οποίο χρησιμοποιείται για τις δρομολογήσεις πακέτων σε δίκτυα δεδομένων.

## Παραδείγματα

**Π18.1** Έστω ότι έχουμε μια απλά συνδεδεμένη λίστα με την εξής δομή κόμβου:

```
struct node
{
 float temp;
 struct node *next;
}*list_head;
```

Ο χειριστής της λίστας `list_head` δείχνει στην κεφαλή της λίστας. Οι αριθμοί της λίστας είναι οι θερμοκρασίες που μετρήθηκαν κατά τη διάρκεια ενός πειράματος. Η επόμενη συνάρτηση δέχεται ως όρισμα τον χειριστή της λίστας και επιστρέφει ως τιμή τη μέγιστη θερμοκρασία που υπάρχει στη λίστα.

```
float max_temp(struct node *lh)
{
 float max;
 if (lh==NULL)
 {
 puts("Η λίστα είναι κενή");
 return;
 }
 else
 max=lh->temp;
 while (lh!=NULL)
 {
 if (lh->temp>max) max=lh->temp;
 lh=lh->next;
 }
 return max;
}
```

Αν η λίστα είναι κενή, εμφανίζει το κατάλληλο μήνυμα και επιστρέφει.

Καταχωρίζει ως αρχική τιμή της `max` τη θερμοκρασία του πρώτου κόμβου της λίστας.

Αν το πεδίο `temp` περιέχει τιμή μεγαλύτερη από τη `max`, καταχωρίζει στη `max` την τιμή της `temp`.

Στον δείκτη `lh` καταχωρίζεται η διεύθυνση του επόμενου κόμβου.

Μετά από τη σύγκριση του πεδίου της θερμοκρασίας (`lh->temp`) με τη τρέχουσα μέγιστη τιμή (`max`) μέσα στον βρόχο `while`, καταχωρίζεται στον δείκτη `lh` η διεύθυνση του επόμενου κόμβου. Αυτό επαναλαμβάνεται μέχρι να φτάσουμε στον τελευταίο κόμβο της λίστας, όταν δηλαδή καταχωριστεί στον δείκτη `lh` τιμή `NULL`. Τέλος, ως τιμή επιστρέφεται το περιεχόμενο της μεταβλητής `max`.

Μια πιο σωστή προσέγγιση θα ήταν η συνάρτηση να επιστρέφει έναν δείκτη προς τον κόμβο με τη μεγαλύτερη θερμοκρασία και, στην περίπτωση που η λίστα είναι κενή, να επιστρέφει `NULL`. Με λίγες μετατροπές, η συνάρτηση διαμορφώνεται ως εξής:

```
struct node *max_temp(struct node *lh)
{
 struct node *max_node;
```



```

float max;
if (lh==NULL)
{
 return NULL;
}
else
{
 max=lh->temp;
 max_node=lh;
}
while (lh!=NULL)
{
 if (lh->temp>max)
 {
 max=lh->temp;
 max_node=lh;
 }
 lh=lh->next;
}
return max_node;
}

```

Αν η λίστα είναι κενή επιστρέφει NULL.

Καταχωρίζει ως αρχική τιμή της **max** τη θερμοκρασία του πρώτου κόμβου της λίστας και στο δείκτη **max\_node** τη διεύθυνση του πρώτου κόμβου.

Αν η **temp** περιέχει τιμή μεγαλύτερη από τη **max**, καταχωρίζει στη **max** την τιμή της **temp** και στο δείκτη **max\_node** τη διεύθυνση του κόμβου.

Τέλος, επιστρέφει ως τιμή το περιεχόμενο του δείκτη **max\_node** που δείχνει στον κόμβο με τη μέγιστη θερμοκρασία.

Παρατηρούμε τώρα ότι, μέσα στον βρόχο **while**, εκτός από την τρέχουσα μέγιστη τιμή θερμοκρασίας (**max**), καταχωρίζεται στο δείκτη **max\_node** η διεύθυνση του κόμβου στον οποίο εντοπίστηκε η τρέχουσα μέγιστη τιμή. Αυτό επαναλαμβάνεται μέχρι να φτάσουμε στον τελευταίο κόμβο της λίστας. Τέλος, η συνάρτηση επιστρέφει ως τιμή το περιεχόμενο του δείκτη **max\_node**, δηλαδή τη διεύθυνση του κόμβου με τη μέγιστη θερμοκρασία. Στην περίπτωση που η λίστα είναι κενή, επιστρέφεται τιμή NULL.

Προσέξτε επίσης τον ορισμό της συνάρτησης, με τον οποίο δηλώνεται ότι επιστρέφεται ένας δείκτης προς δομές τύπου **node**.

**Π18.2** Το επόμενο πρόγραμμα διαβάζει τους ακέραιους αριθμούς που υπάρχουν σε ένα αρχείο κειμένου με όνομα ARITHMOI και τους καταχωρίζει σε μια απλά συνδεδεμένη λίστα:

```

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

```

18\_p18\_2.c

```

struct node
{
 int data;
 struct node *next;
}*list_head,*neof;
void add_node_to_list(int num);

```

Η δομή των κόμβων της συνδεδεμένης λίστας.

Προκαταβολική δήλωση της συνάρτησης **add\_node\_to\_list()**, η οποία ορίζεται μετά από τη **main()**.



```

int main(void)
{
 int ar;
 struct stack *mystack=create_stack();
 struct queue *myqueue=create_queue();
 FILE *fp;
 fp=fopen("ARITHMOI", "r");
 while (!feof(fp))
 {
 fscanf(fp,"%d",&ar);
 if (push_in_stack(mystack,ar)==false) puts("Δεν υπάρχει μνήμη");
 }
 fclose(fp);
 while (!is_stack_empty(mystack))
 {
 ar=get_stack_top(mystack);
 if (insert_in_queue(myqueue,ar)==false) puts("Δεν υπάρχει μνήμη");
 pop_from_stack(mystack);
 }
 display_queue(myqueue);
 return 0;
}

```

**Π18.7** Το πρόγραμμα που ακολουθεί δημιουργεί τους γράφους (α) και (β) του σχήματος. Ο γράφος (α) είναι μη κατευθυνόμενος, ενώ το (β) απεικονίζει ένα δένδρο το οποίο υλοποιείται από έναν κατευθυνόμενο γράφο. Οι αριθμοί στις κορυφές αποτελούν το προσδιοριστικό κάθε κορυφής.

Θα χρησιμοποιηθούν γράφοι με μια συνδεδεμένη λίστα κορυφών, των οποίων η δημιουργία και η διαχείριση γίνεται με συναρτήσεις που υπάρχουν στο αρχείο κεφαλίδας [mygraphLL.h](#).

Το πρόγραμμα εμφανίζει τη δομή των δύο γράφων, τη διάσχιση κατά βάθος με αφετηρία την κορυφή 12, καθώς και τις δυνατές διαδρομές από την κορυφή 12 στην 9!

```

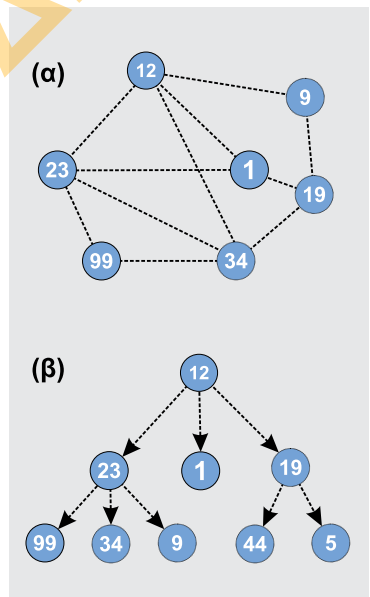
#include <stdio.h>
#include <stdlib.h>
#include "mygraphLL.h"

```

```

// Ορισμοί των συναρτήσεων, όπως αυτές δημιουργήθηκαν στον κώδικα
// του προγράμματος 18_graph_LL_all.c

```



```

int main(void)
{
 // Δημιούργησε έναν μη κατευθυνόμενο γράφο A
 struct Graph *graph_a=dimioyrgia_grafou(false);
 // Δημιούργησε έναν κατευθυνόμενο γράφο B
 struct Graph *graph_b=dimioyrgia_grafou(true);
 // Πρόσθεσε στον γράφο A τις παρακάτω κορυφές
 prosthiki_korifis(graph_a,1);
 prosthiki_korifis(graph_a,9);
 prosthiki_korifis(graph_a,12);
 prosthiki_korifis(graph_a,19);
 prosthiki_korifis(graph_a,23);
 prosthiki_korifis(graph_a,34);
 prosthiki_korifis(graph_a,99);
 // Εμφάνισε όλες τις κορυφές του γράφου A
 emfanisi_korifon(graph_a);
 // Πρόσθεσε στον γράφο A τις παρακάτω ακμές
 prosthiki_akmis(graph_a,1,12);
 prosthiki_akmis(graph_a,1,23);
 prosthiki_akmis(graph_a,1,19);
 prosthiki_akmis(graph_a,12,29);
 prosthiki_akmis(graph_a,12,9);
 prosthiki_akmis(graph_a,12,34);
 prosthiki_akmis(graph_a,23,99);
 prosthiki_akmis(graph_a,99,34);
 prosthiki_akmis(graph_a,34,19);
 prosthiki_akmis(graph_a,19,9);
 // Εμφάνισε τις λίστες γειτνίασης κάθε κορυφής του γράφου A
 emfanisi_grafou(graph_a);
 // Εμφάνισε τη διάσχιση του γράφου A ξεκινώντας από τη κορυφή 12 χρησιμοποιώντας
 // τον αλγόριθμο κατά βάθος (DepthFirstSearch)
 DFS(graph_a,12);
 // Εμφάνισε όλες τις δυνατές διαδρομές στον γράφο A από την κορυφή 12 στη 9
 DisplayPaths(graph_a,12,9);
 // Πρόσθεσε στον γράφο B τις παρακάτω κορυφές
 prosthiki_korifis(graph_b,1);
 prosthiki_korifis(graph_b,9);
 prosthiki_korifis(graph_b,12);
 prosthiki_korifis(graph_b,19);
 prosthiki_korifis(graph_b,23);
 prosthiki_korifis(graph_b,34);
 prosthiki_korifis(graph_b,99);
 prosthiki_korifis(graph_b,5);
 prosthiki_korifis(graph_b,44);
 // Εμφάνισε όλες τις κορυφές του γράφου B
 emfanisi_korifon(graph_b);
 // Πρόσθεσε στον γράφο B τις παρακάτω ακμές
 prosthiki_akmis(graph_b,12,23);

```

## Συχνά λάθη

- 🔊 Όταν **διαγράφουμε** έναν κόμβο από μια δυναμική δομή δεδομένων πρέπει πάντα να **αποδεσμεύουμε** τον χώρο μνήμης που καταλάμβανε. Σε διαφορετική περίπτωση, οι διαγραφμένοι κόμβοι θα εξακολουθούν να δεσμεύουν χώρο, παρά το γεγονός ότι δεν θα είναι πλέον προσπελάσιμοι.

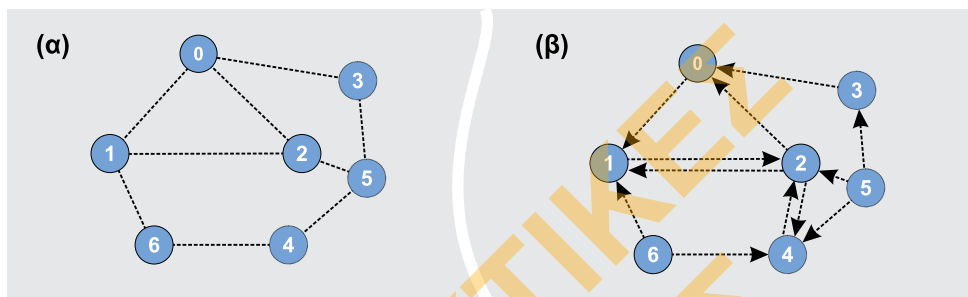
## Ανασκόπηση Κεφαλαίου 18

- Οι **δυναμικές δομές δεδομένων** επιτρέπουν τη διαχείριση δεδομένων με δυναμικό τρόπο. Δεσμεύουν μνήμη όταν τη χρειάζονται και την αποδεσμεύουν όταν δεν είναι πλέον απαραίτητη.
- Οι κατάλληλες δομές δεδομένων μπορεί να επιταχύνουν εντυπωσιακά την αναζήτηση και την ταξινόμηση.
- Οι **δομές δεδομένων** αποτελούνται από **κόμβους**. Κάθε κόμβος περιέχει δεδομένα και δείκτες για τη σύνδεσή του με την υπόλοιπη δομή δεδομένων.
- Μια **δομή** επιτρέπει την προσθήκη, διαγραφή, και αναζήτηση των κόμβων της.
- Οι **απλά συνδεδεμένες λίστες** αποτελούνται από κόμβους που ο καθένας περιέχει έναν δείκτη προς τον επόμενο.
- Οι **διπλά συνδεδεμένες λίστες** αποτελούνται από κόμβους που ο καθένας περιέχει δείκτες τόσο προς τον επόμενο όσο και προς τον προηγούμενο κόμβο.
- Οι **διατεταγμένες λίστες** διατηρούν τους κόμβους τους σε μια διατεταγμένη σειρά, είτε αύξουσα είτε φθίνουσα.
- Οι δομές **στοίβας** και **ουράς** μπορούν να υλοποιηθούν δυναμικά, με τη χρήση συνδεδεμένων λιστών.
- Τα **δυναμικά δένδρα αναζήτησης** αποτελούνται από κόμβους διατεταγμένους σε μια δενδροειδή δομή. Κάθε κόμβος του δυαδικού δένδρου περιέχει, εκτός από το τμήμα δεδομένων, και έναν δείκτη προς το αριστερό υποδένδρο του και έναν δείκτη προς το δεξιό. Κάθε αριστερό υποδένδρο περιέχει μικρότερες τιμές κλειδιού από τη ρίζα του και κάθε δεξιό υποδένδρο μεγαλύτερες (ή ίσες).
- Οι **γράφοι** αποτελούνται από κόμβους (κορυφές) και από ακμές που τους συνδέουν μεταξύ τους. Ένας γράφος μπορεί να είναι κατευθυνόμενος ή μη κατευθυνόμενος. Ο γράφος αποτελεί τη γενικότερη περίπτωση μιας δυναμικής δομής δεδομένων και επομένως μπορεί να υλοποιήσει και απλούστερες δομές όπως δένδρα και λίστες.
- Σε έναν δυναμικό γράφο, οι κορυφές αναπαρίστανται με έναν πίνακα ή μια συνδεδεμένη λίστα και οι ακμές με λίστες γειτνίασης.

**Ασκήσεις Κεφαλαίου 18**

- 18.1** Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί έναν πίνακα **a** 100 θέσεων και να τον συμπληρώνει με 100 τυχαίους ακέραιους αριθμούς. Στη συνέχεια, το πρόγραμμα να δημιουργεί μια απλά συνδεδεμένη λίστα στην οποία να καταχωρίζει τους 100 αριθμούς του πίνακα **a**. ★★
- 18.2** Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί έναν πίνακα **a** 100 θέσεων και να τον συμπληρώνει με 100 τυχαίους ακέραιους αριθμούς. Στη συνέχεια, το πρόγραμμα να δημιουργεί μια **ταξινομημένη** διπλά συνδεδεμένη λίστα στην οποία να καταχωρίζει τους 100 αριθμούς του πίνακα **a**. Τέλος, να εμφανίζει όλους τους αριθμούς της λίστας. ★★★
- 18.3** Ποια από τα επόμενα αληθεύουν; ★
- ☐ Οι συνδεδεμένες λίστες δεσμεύουν συγκεκριμένο μέγεθος μνήμης.
  - ☐ Σε μια δομή ουράς, το πρώτο στοιχείο που προστίθεται στην ουρά είναι το πρώτο που φεύγει από την ουρά.
  - ☐ Σε μια δομή στοίβας, το πρώτο στοιχείο που προστίθεται στη στοίβα είναι το πρώτο που φεύγει από τη στοίβα.
  - ☐ Σε μια απλά συνδεδεμένη λίστα δεν μπορούμε να εμφανίσουμε τα στοιχεία της λίστας με τη σειρά από το πρώτο που καταχωρίσαμε προς το τελευταίο.
  - ☐ Οι λίστες είναι η πλέον κατάλληλη δομή για τη μοντελοποίηση ενός γενεαλογικού δένδρου.
- 18.4** Έστω ότι έχουμε μια απλά συνδεδεμένη λίστα με τη δομή κόμβου του διπλανού πλαισίου. Ο χειριστής της λίστας **list\_head** δείχνει στην κεφαλή της λίστας. Οι αριθμοί που περιέχει η λίστα είναι τα σωματικά βάρη ενός δείγματος πληθυσμού. Να γραφεί μια συνάρτηση η οποία να υπολογίζει και να επιστρέφει τον μέσο όρο του σωματικού βάρους του δείγματος. Στη συνάρτηση να μεταβιβάζεται ο δείκτης κεφαλής της λίστας. ★★
- ```
struct node
{
    float varos;
    struct node *next;
}*list_head;
```
- 18.5** Να γραφεί ένα πρόγραμμα το οποίο να ζητάει ονόματα και να δημιουργεί μια στοίβα στην οποία να τα καταχωρίζει. Το πρόγραμμα να σταματάει όταν δοθεί μια κενή συμβολοσειρά (""). Τέλος, να εμφανίζει τα ονόματα της στοίβας και να τα απωθεί από αυτήν. Το πρόγραμμα πρέπει να χρησιμοποιεί τη λογική και τις συναρτήσεις του προγράμματος **18_new_stack.c**, οι οποίες πρέπει να τροποποιηθούν έτσι ώστε να υποστηρίζουν τη νέα δομή κόμβου (με πεδίο δεδομένων

- 18.18** Μελετήστε τους γράφους των σχημάτων (α) και (β). Ο γράφος του σχήματος (α) είναι μη κατευθυνόμενος. Σημειώστε τη διάσχιση κατά βάθος και κατά πλάτος ξεκινώντας από την κορυφή 2 και των δύο γράφων. Επειδή μπορεί να υπάρχουν πολλές δυνατές λύσεις, σημειώστε τουλάχιστον δύο για κάθε διάσχιση. Σημειώστε όλες τις δυνατές διαδρομές από την κορυφή 2 στην 5 και στους δύο γράφους. Τέλος, δημιουργήστε έναν πίνακα στον οποίο να εμφανίζεται ο βαθμός κάθε κορυφής και των δύο γράφων. Στον γράφο (β) να εμφανίζεται χωριστά ο βαθμός εξόδου και ο βαθμός εισόδου κάθε κορυφής. ★★



- 18.19** Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί τους γράφους των σχημάτων (α) και (β) της Άσκησης 18.18. Θα χρησιμοποιηθούν γράφοι με πίνακα δεικτών στις λίστες γειτνίασης, των οποίων η δημιουργία και η διαχείριση γίνεται με συναρτήσεις που υπάρχουν στο αρχείο κεφαλίδας mygraph.h. Το πρόγραμμα να εμφανίζει τη δομή των δύο γράφων, δηλαδή τις λίστες γειτνίασης κάθε κορυφής. ★★
- 18.20** Συμπληρώστε τον κώδικα της Άσκησης 18.19 ώστε το πρόγραμμα να εμφανίζει τη διάσχιση κατά βάθος και κατά πλάτος, ξεκινώντας από την κορυφή 2, και των δύο γράφων. Το πρόγραμμα πρέπει επίσης να εμφανίζει όλες τις δυνατές διαδρομές από την κορυφή 2 στην 5 και στους δύο γράφους³². ★★
- 18.21** Τροποποιήστε τον κώδικα του Παραδείγματος Π18_8 (αρχείο 18_p18_8.c), ώστε στο πρόγραμμα να προστεθεί η δυνατότητα διαγραφής κορυφών. Προσοχή, η διαγραφή μιας κορυφής θα έχει ως επακόλουθο και τη διαγραφή όλων των εισερχόμενων και εξερχόμενων ακμών της. ★★★
- 18.22** Τροποποιήστε τον κώδικα του Παραδείγματος Π18_8 (αρχείο 18_p18_8.c), ώστε στο πρόγραμμα να προστεθεί η δυνατότητα εμφάνισης του βαθμού (degree) κάθε κορυφής του γράφου. Στην περίπτωση που ο γράφος είναι κατευθυνόμενος να εμφανίζει χωριστά τον βαθμό εξόδου και τον βαθμό εισόδου κάθε κορυφής. ★★★

³² Για τη διάσχιση κατά βάθος και κατά πλάτος, καθώς και για την εμφάνιση των δυνατών διαδρομών, να χρησιμοποιηθούν οι κατάλληλες συναρτήσεις των προγραμμάτων 18_graph_BFS.c και 18_graph_Paths.c.

Κεφάλαιο

19

Ο προμεταγωγτι-
στής της C

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Το κεφάλαιο αυτό παρουσιάζει τις δυνατότητες του προμεταγλωττιστή της C. Ο προμεταγλωττιστής διαθέτει οδηγίες για τη χρήση βιβλιοθηκών, σταθερών, και μακροεντολών, στα προγράμματα που φτιάχνουμε.

Επίσης, κάποιες από τις οδηγίες του προμεταγλωττιστή επιτρέπουν τη μεταγλώττιση συγκεκριμένου κώδικα «υπό συνθήκη». Με αυτόν τον τρόπο μπορούν να μεταγλωττίζονται διαφορετικές εκδόσεις του προγράμματος, με ελάχιστες ή καθόλου αλλαγές στον κώδικά του.

Τέλος, παρουσιάζονται οι δυνατότητες που μας παρέχει ο προμεταγλωττιστής για την αποσφαλμάτωση των προγραμμάτων μας.

Εισαγωγή

Όπως έχει ήδη αναφερθεί σε προηγούμενα κεφάλαια, μέσα στον πηγαίο κώδικα του προγράμματός μας μπορούμε να συμπεριλάβουμε οδηγίες (compiler directives) προς τον προμεταγλωττιστή της C (C precompiler ή C preprocessor). Οι οδηγίες αυτές στην πραγματικότητα δεν είναι ενσωματωμένες στη γλώσσα, αλλά αποτελούν τμήμα του περιβάλλοντος μεταγλώττισης. Μέχρι τώρα έχουμε συναντήσει και χρησιμοποιήσει τις οδηγίες **#include** και **#define**.

Οι οδηγίες εκτελούνται από τον προμεταγλωττιστή της C, ένα στάδιο πριν ξεκινήσει η διαδικασία της μεταγλώττισης. Οι πιο συχνά χρησιμοποιούμενες οδηγίες του προμεταγλωττιστή της C είναι οι εξής:

```
#include
#define
#undef
#if
#else
#elif
#endif
#ifdef
#ifndef
#error
```

Οι παραπάνω οδηγίες ορίζονται στο αρχικό πρότυπο (ANSI) της γλώσσας και υποστηρίζονται από όλους τους μεταγλωττιστές. Τα πιο πρόσφατα πρότυπα, αλλά και ορισμένοι μεταγλωττιστές, υποστηρίζουν ένα εκτεταμένο σύνολο οδηγιών, στις οποίες όμως δεν θα αναφερθούμε.

#include

Σε όλα σχεδόν τα προγράμματα της C που έχουμε δημιουργήσει μέχρι τώρα χρησιμοποιούσαμε την οδηγία **#include**. Γνωρίζουμε ότι η οδηγία **#include** συντάσσεται με τον εξής τρόπο:

#include <αρχείο> ή **#include "όνομα_αρχείου"**

Ο πρώτος τρόπος χρησιμοποιείται για τη συμπερίληψη αρχείων κεφαλίδων της καθιερωμένης βιβλιοθήκης της C, ενώ ο δεύτερος τρόπος για δικά μας αρχεία κεφαλίδων. Και με τους δύο τρόπους ο μεταγλωττιστής αναγκάζεται να συμπεριλάβει κατά τη φάση της μεταγλώττισης το αρχείο το οποίο ορίζεται.

Για παράδειγμα, το αποτέλεσμα της πρότασης:

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

αναγκάζει τον μεταγλωττιστή να συμπεριλάβει το αρχείο κεφαλίδας **stdio.h** της καθιερωμένης βιβλιοθήκης, ενώ η πρόταση:

```
#include "mylib.h"
```


αναγκάζει τον μεταγλωττιστή να συμπεριλάβει το αρχείο `mylib.h`, από τον ίδιο φάκελο στον οποίο βρίσκεται και το πηγαίο αρχείο (.c) του προγράμματός μας. Αυτός ο τρόπος μάς δίνει τη δυνατότητα να καθορίσουμε και τη διαδρομή, απόλυτη ή σχετική, στην οποία βρίσκεται το αρχείο. Για παράδειγμα:

```
#include "c:/test/mylib.h" // όταν το αρχείο βρίσκεται στον φάκελο c:/test
#include "../mylib.h"      // όταν το αρχείο βρίσκεται στον προηγούμενο φάκελο
```

#define

Και η οδηγία `#define` μάς είναι γνώριμη από προγράμματα που έχουμε χρησιμοποιήσει μέχρι τώρα. Η οδηγία ορίζει ένα αναγνωριστικό και ένα σύνολο χαρακτήρων που θα αντικαταστήσει αυτό το αναγνωριστικό κατά τη διαδικασία της μεταγλώττισης του πηγαίου κώδικα. Η σύνταξή της είναι:

#define αναγνωριστικό χαρακτήρες

Ο προμεταγλωττιστής της C αντικαθιστά το **αναγνωριστικό** με τους **χαρακτήρες**, σε όποιο σημείο του πηγαίου κώδικα και αν το βρει. Συνήθως χρησιμοποιούμε την οδηγία `#define` για να ορίσουμε κάποιες σταθερές του προγράμματός μας. Για παράδειγμα, στο πρόγραμμα που ακολουθεί ανατίθεται στο αναγνωριστικό `SIZE` η τιμή 100. Ο προμεταγλωττιστής αντικαθιστά το αναγνωριστικό `SIZE` με το 100, σε όποιο σημείο του προγράμματος και αν το εντοπίσει. Αυτό γίνεται πριν αρχίσει η κανονική διαδικασία της μεταγλώττισης.

```
#include <stdio.h>
#define SIZE 100
int main(void)
{
    int ar[SIZE];
    for (i=0; i<SIZE; i++)
        scanf("%d", &ar[i]);
    ....
    return 0;
}
```

Ο προμεταγλωττιστής αντικαθιστά το αναγνωριστικό `SIZE` με το 100, σε όποιο σημείο του προγράμματος και αν το εντοπίσει.

Επομένως, το παραπάνω πρόγραμμα δημιουργεί έναν πίνακα 100 θέσεων και τον συμπληρώνει με αριθμούς που θα πληκτρολογήσει ο χρήστης. Αν αλλάξουμε την τιμή που ανατίθεται στο αναγνωριστικό `SIZE`, το πρόγραμμα θα προσαρμοστεί αυτομάτως ώστε να δουλεύει με το νέο μέγεθος πίνακα.

Αν και δεν είναι υποχρεωτικό, μια καλή προγραμματιστική πρακτική είναι να χρησιμοποιούμε **κεφαλαίους** χαρακτήρες στα αναγνωριστικά των οδηγιών `#define`. Με αυτό τον τρόπο ξεχωρίζουν στον κώδικά μας τόσο οι σταθερές όσο και οι μακροεντολές που θα συναντήσουμε στη συνέχεια.

Μακροεντολές

Η χρήση της `#define` δεν περιορίζεται μόνο στον ορισμό σταθερών. Η οδηγία χρησιμοποιείται επίσης για τον ορισμό **μακροεντολών** (macros). Οι μακροεντολές μοιάζουν με συναρτήσεις και μπορούν να έχουν παραμέτρους! Ο ορισμός μιας μακροεντολής είναι ίδιος με τον ορισμό μιας σταθεράς. Για παράδειγμα, η παρακάτω πρόταση δηλώνει μια μακροεντολή με όνομα `PRINT` η οποία διαθέτει μία παράμετρο:

```
#define PRINT(X) printf("%d\n",X)
```

Η πρώτη συμβολοσειρά του ορισμού (`PRINT`) καθορίζει το όνομα της μακροεντολής, το οποίο συνοδεύεται από ένα ζεύγος παρενθέσεων, μέσα στο οποίο υπάρχουν οι «τυπικές παράμετροι» της μακροεντολής. Στη συγκεκριμένη περίπτωση υπάρχει μία παράμετρος με όνομα `X`. Στο σύνολο χαρακτήρων που ακολουθεί το όνομα της μακροεντολής χρησιμοποιούνται οι τυπικές παράμετροι της μακροεντολής, στη συγκεκριμένη περίπτωση το `X`. Όταν ο προμεταγλωττιστής εντοπίσει τη μακροεντολή μέσα στον κώδικα, για παράδειγμα

```
PRINT(10);
```

αντικαθιστά το όνομα της μακροεντολής με το σύνολο χαρακτήρων που έχουμε ορίσει στην οδηγία `#define` και αντικαθιστά τις τυπικές παραμέτρους με τα ορίσματα της μακροεντολής. Επομένως, η παραπάνω πρόταση αντικαθίσταται με την πρόταση:

```
printf("%d\n",10);
```

και η πρόταση `PRINT(a+b);` αντικαθίσταται με την πρόταση `printf("%d\n",a+b);`

Η ακόλουθη πρόταση ορίζει μια μακροεντολή, με όνομα `TETR`, για τον υπολογισμό του τετραγώνου ενός αριθμού, επίσης με μία παράμετρο:

```
#define TETR(A) A*A
```

Ο προμεταγλωττιστής θα κάνει τις παρακάτω αντικαταστάσεις:

```
a=TETR(10); // αντικαθίσταται με τη πρόταση: a=10*10
```

```
b=TETR(a); // αντικαθίσταται με τη πρόταση: b=a*a
```

Θα πρέπει να δίνεται ιδιαίτερη προσοχή στην προτεραιότητα των πράξεων. Ο προμεταγλωττιστής κάνει μια απλή αντικατάσταση, επομένως μια πρόταση της μορφής `a=TETR(5+5);` θα αντικατασταθεί με την πρόταση `a=5+5 * 5+5;` η οποία θα έχει ως αποτέλεσμα την καταχώριση του αριθμού 35 ($5 + 5 * 5 + 5$) στη μεταβλητή `a`, αντί του 100 που θα περιμέναμε. Αυτό γίνεται επειδή ο πολλαπλασιασμός έχει μεγαλύτερη προτεραιότητα από την πρόσθεση. Αν θέλουμε να μη συμβαίνει αυτό, θα πρέπει να χρησιμοποιήσουμε παρενθέσεις κατά τον ορισμό της μακροεντολής. Έτσι, αν ορίσουμε τη μακροεντολή `TETR` ως εξής:

```
#define TETR(A) (A) * (A)
```

ο προμεταγωγτιστής θα αντικαταστήσει την πρόταση `a=TETR(5+5);` με την πρόταση `a=(5+5) * (5+5);` η οποία θα έχει το επιθυμητό αποτέλεσμα.

Η ακόλουθη πρόταση ορίζει μια μακροεντολή με όνομα `MAX` και δύο παραμέτρους:

```
#define MAX(A,B) (A>B)?A:B
```

Η μακροεντολή `MAX` αντικαθίσταται από μια παράσταση που υπολογίζει τη μεγαλύτερη τιμή από τις δύο παραμέτρους της `A` και `B`. Για παράδειγμα, η παράσταση `MAX(10,5)` αντικαθίσταται από την παράσταση `(10>5)?10:5` και επιστρέφει τιμή 10, ενώ η `MAX(k,l)` αντικαθίσταται με την `(k>l)?k:l` και επιστρέφει τη μεγαλύτερη τιμή μεταξύ των μεταβλητών `k` και `l`.

Στο πρόγραμμα που ακολουθεί ορίζονται και οι τρεις μακροεντολές που αναφέραμε ως παραδείγματα. Το πρόγραμμα εμφανίζει τα αποτελέσματα που φαίνονται στο πλαίσιο.

```
#include <stdio.h>
#define PRINT(X) printf("%d\n",X)
#define TETR(A) (A)*(A)
#define MAX(A,B) (A>B)?A:B
int main(void)
{
    int k=5,l=10;
    PRINT(20);           // αντικαθίσταται με: printf("%d\n",20); - εμφανίζει το 20
    PRINT(k+1);          // αντικαθίσταται με: printf("%d\n",k+1); - εμφανίζει το 15
    l=TETR(4);           // αντικαθίσταται με: l=(4)*(4); - καταχωρίζει στη μεταβλητή l το 16
    PRINT(TETR(k));      // αντικαθίσταται με: printf("%d\n", (5)*(5)); - εμφανίζει το 25
    printf("%d\n",MAX(k,l)); // αντικαθίσταται με: printf("%d\n", (k>l)?k:l); - εμφανίζει το 16
    PRINT(MAX(100,30));  // αντικαθίσταται με: printf("%d\n", (100>30)?100:30); - εμφανίζει το 100
    return 0;
}
```

19_compiler_directives_1.c

20
15
25
16
100

Μακροεντολές με περισσότερες προτάσεις

Οι μακροεντολές που συναντήσαμε μέχρι τώρα αναπτύσσονται σε μία μόνο πρόταση. Όμως, έχουμε τη δυνατότητα να ορίσουμε μακροεντολές με πολλές προτάσεις, ακόμα και σύνθετες. Για παράδειγμα, η παρακάτω μακροεντολή:

```
#define DISPLAY_MY_NAME(X) int i;for (i=1;i<=X;i++) puts("Νίκος")
```

εμφανίζει τη λέξη «Νίκος» τόσες φορές όσες υποδεικνύει η τιμή της παραμέτρου `X`.

Ο προμεταγωγτιστής μετατρέπει το παρακάτω πρόγραμμα στον κώδικα που φαίνεται στο διπλανό πλαίσιο.

```
int main(void)
{
    DISPLAY_MY_NAME(5);
    return 0;
}
```

```
int main(void)
{
    int i;
    for (i=1;i<=5;i++)
        puts("Νίκος");
    return 0;
}
```

Παραδείγματα

Π19.1 Στο πρόγραμμα που ακολουθεί χρησιμοποιείται η οδηγία **#define** για να ορίσει τόσο αναγνωριστικά-σταθερές, όσο και μακροεντολές. Μελετήστε το προσεκτικά:

```
#include <stdio.h>
#include <stdlib.h>
#define PROGRAMMA int main(void)
#define OLA_KALA return 0;
#define AKERAIOI int
#define ARXH {;
#define TELOS };
#define ALLAXE_GRAMMH printf("\n");
#define BIBLIO puts("C σε βάθος");
#define PRINT(X) printf("%d",X);
#define GET(X) scanf("%d",&X);
#define PROSTHESE(X,Y) (X)+(Y)

PROGRAMMA
ARXH
    AKERAIOI a,b,c;
    GET(a);
    GET(b);
    c=PROSTHESE(a,b);
    PRINT(c);
    ALLAXE_GRAMMH;
    BIBLIO;
    OLA_KALA;
TELOS
```

19_p19_1.c

Ορισμοί αναγνωριστικών-σταθερών

Ορισμοί αναγνωριστικών-μακροεντολών

Προμεταγλωττιστής

```
int main(void)
{
    int a,b,c;
    scanf("%d",&a);
    scanf("%d",&b);
    c=(a)+(b);
    printf("%d",c);
    printf("\n");
    puts("C σε βάθος");
    return 0;
}
```

Ο προμεταγλωττιστής της C, πριν από την έναρξη της μεταγλώττισης, θα αντικαταστήσει όλα τα αναγνωριστικά που έχουν οριστεί με την οδηγία **#define**, με τις συμβολοσειρές οι οποίες τους έχουν ανατεθεί. Στις μακροεντολές γίνεται επίσης αντικατάσταση των παραμέτρων. Ο κώδικας που θα προκύψει φαίνεται στο διπλανό πλαίσιο και είναι αυτός που τελικά θα μεταγλωττιστεί.

Παρατηρούμε ότι με χρήση της **#define** μπορούμε να αλλάξουμε ριζικά την εικόνα ενός προγράμματος χωρίς να επηρεάσουμε την ουσία του! Αυτό μερικές φορές (όμως όχι πάντα!) κάνει τον κώδικα πιο ευανάγνωστο.

Ασκήσεις Κεφαλαίου 19

- 19.1** Τι κάνει το παρακάτω πρόγραμμα; Τι εμφανίζει στην οθόνη; Πότε θα σταματήσει; ★★

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

#define TO_PROGRAMMA_MOY int main(void)
#define GYRNA return 0
#define BEGIN {
#define END }
#define TYXAIOS rand()%100
#define EMFANISE(X) puts("==");printf("%d\n",X)
#define PATA_ENTER_GIA_TON_EPOMENO getchar()
#define APO_EDO do {
#define MEXRI_EDO_KANTO_SYNEXEIA } while (1)

TO_PROGRAMMA_MOY
BEGIN
    APO_EDO
        EMFANISE(TYXAIOS);
        PATA_ENTER_GIA_TON_EPOMENO;
        MEXRI_EDO_KANTO_SYNEXEIA;
        GYRNA;
END
```

- 19.2** Δίνεται ο παρακάτω κώδικας:

```
A_RE_TI_FTIAXNO_SHMERA
KSEKINA_ORE
    KANTO_APO_EDO(FORES)
        GRAPSE("Η ΓΛΩΣΣΑ C ΣΕ ΒΑΘΟΣ");
        GRAPSE("-----");
    MEXRI_EDO
        PERIMENE_EDO_MEXRI_NA_PATHTHEI_ENTER;
        GRAPSE("ΤΕΛΟΣ");
        GYRNA_PISO;
TSOS
```

Προσθέστε, **πριν** από τον παραπάνω κώδικα, τις κατάλληλες οδηγίες **#include** και **#define** ώστε το πρόγραμμα να μεταγλωττιστεί σωστά και να εμφανίσει τη φράση «Η γλώσσα C σε βάθος», ακολουθούμενη από μια γραμμή με παύλες, 5 φορές. Στη συνέχεια, το πρόγραμμα θα περιμένει να πατήσετε το ENTER και μετά θα εμφανίζει τη λέξη «ΤΕΛΟΣ» και θα τερματίζεται. **Δεν θα πρέπει να κάνετε καμία αλλαγή στον παραπάνω κώδικα.** ★★

Κεφάλαιο

Από τη C,
στη C++



Το κεφάλαιο αυτό παρέχει μια συνοπτική και ομαλή μετάβαση από τη φιλοσοφία του «κλασικού» διαδικασιακού προγραμματισμού στον μαγικό κόσμο του αντικειμενοστρεφούς προγραμματισμού.

Παρουσιάζονται οι θεμελιώδεις έννοιες του αντικειμενοστρεφούς προγραμματισμού, καθώς και κάποια πολύ βασικά στοιχεία της γλώσσας C++.

Γίνεται επίσης αναφορά στις διαφορές των δύο γλωσσών.

Με λίγα λόγια, το κεφάλαιο αυτό σας προετοιμάζει για το επόμενο αναμενόμενο βήμα σας: την ενασχόλησή σας με τον αντικειμενοστρεφή προγραμματισμό!

Εισαγωγή

Μετά από τη σπουδή σας στη C, το επόμενο φυσικό βήμα θα είναι σίγουρα μια γλώσσα αντικειμενοστρεφούς προγραμματισμού. Με δεδομένη τη γνώση που αποκτήσατε μαθαίνοντας τη γλώσσα C, η C++ είναι το πιο εύκολο βήμα στον δύσκολο δρόμο του αντικειμενοστρεφούς προγραμματισμού³³.

Βασισμένη στη γλώσσα C, η C++ αναπτύχθηκε από τον Bjarne Stroustrup στα εργαστήρια Bell, την περίοδο 1983-1985. Πριν από το 1983, ο Bjarne Stroustrup δανείστηκε τα αντικειμενοστρεφή χαρακτηριστικά της Simula (η οποία θεωρείται η πρώτη αντικειμενοστρεφής γλώσσα προγραμματισμού) και, συνδυάζοντάς τα με την αποδοτικότητα και την ευελιξία της C, δημιούργησε μια έκδοση της C που ονομάστηκε «C with Classes» (C με κλάσεις). Αυτή αποτέλεσε την πρώτη «πειραματική» έκδοση της C++, η οποία απέκτησε το τελικό της όνομα το 1983. Οι συνεχείς βελτιώσεις και προσθήκες στη γλώσσα συνεχίστηκαν μέχρι το 1997, οπότε οριστικοποιήθηκε το πρότυπο της γλώσσας και πιστοποιήθηκε από διεθνείς οργανισμούς (ANSI, ISO κ.λπ.). Αρχικά η C++ σχεδιάστηκε για το λειτουργικό σύστημα UNIX. Σήμερα οι διαφορετικές εκδόσεις της γλώσσας καλύπτουν όλα τα λειτουργικά συστήματα και αποτελούν αναμφίβολα την επιλογή των περισσότερων επαγγελματιών προγραμματιστών για την ανάπτυξη πολύπλοκων εφαρμογών. Με τη C++ οι προγραμματιστές μπορούν να βελτιώσουν την ποιότητα των προγραμμάτων τους, και ο κώδικας που παράγουν μπορεί να επαναχρησιμοποιηθεί πιο εύκολα.

Η κληρονομιά από τη C

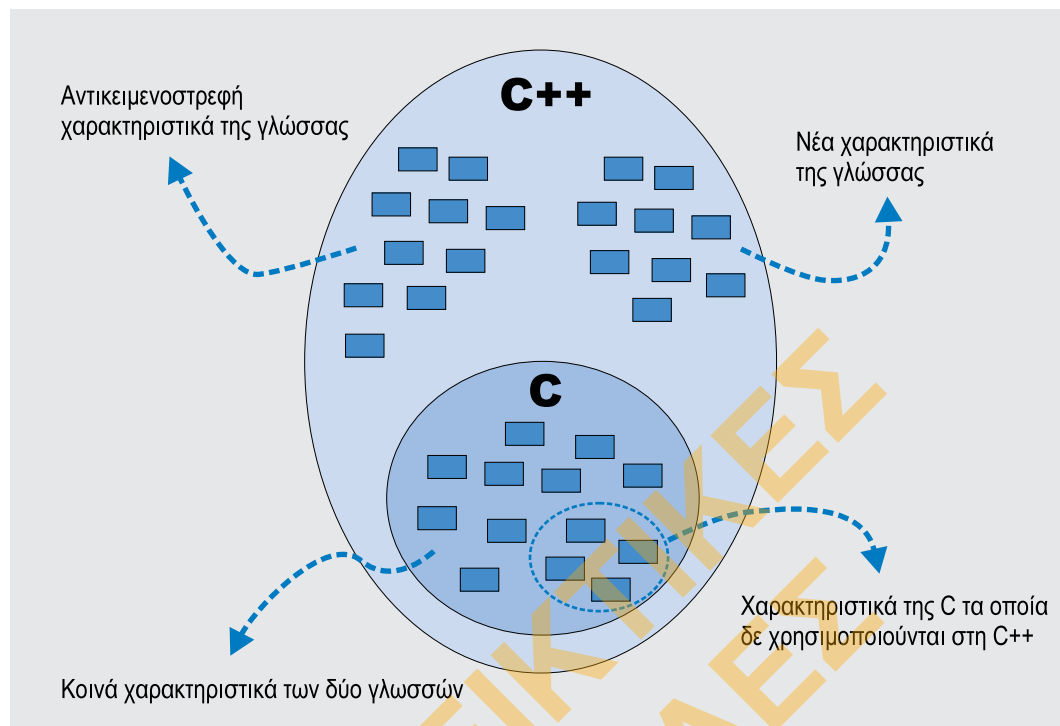
Η C++ ξεκίνησε ως μια επέκταση της C, προσθέτοντας στην πλέον δημοφιλή αυτή γλώσσα αντικειμενοστρεφείς δυνατότητες (Σχήμα 20.1). Όμως η βάση της γλώσσας και οι «παραδοσιακές» δομές της παραμένουν ίδιες όπως της C.

Μπορούμε λοιπόν να ισχυριστούμε ότι η C αποτελεί ένα υποσύνολο της C++. Ένα πρόγραμμα γραμμένο σε C, με ελάχιστες ή καθόλου αλλαγές, θα μεταγλωττιστεί χωρίς πρόβλημα από οποιονδήποτε μεταγλωττιστή της C++.

Μια αντικειμενοστρεφής γλώσσα

Η C++, μαζί με την Java και τη C#, αποτελούν τις πλέον δημοφιλείς και χαρακτηριστικές γλώσσες αντικειμενοστρεφούς προγραμματισμού (Object Oriented Programming, OOP). Οι αντικειμενοστρεφείς γλώσσες προγραμματισμού δίνουν περισσότερη έμφαση στα δεδομένα, παρά στον κώδικα. Το πρόγραμμα αναπτύσσεται γύρω από τα αντικείμενα που απαρτίζουν το πρόβλημα, και τις διαδικασίες (τον κώδικα) που επενεργούν σε αυτά τα αντικείμενα.

³³ Το μεγαλύτερο μέρος του παρόντος κεφαλαίου προέρχεται από το βιβλίο μου «Η γλώσσα C++ σε βάθος» – 2η έκδοση, εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ, 2014.



Σχήμα 20.1 Η σχέση των γλωσσών C και C++

Για παράδειγμα, αν θέλαμε να αναπτύξουμε ένα πρόγραμμα διαχείρισης ενοικιαζόμενων αυτοκινήτων, κάθε αυτοκίνητο προς ενοικίαση θα αποτελούσε ένα αντικείμενο. Η ενοικίαση ενός αυτοκινήτου θα γινόταν με την εφαρμογή μιας διαδικασίας (μεθόδου) επάνω σε αυτό το αντικείμενο. Η επιστροφή του αυτοκινήτου από τον πελάτη στην εταιρεία θα γινόταν με την εφαρμογή μιας άλλης διαδικασίας, κ.ο.κ.

Η C++ υποστηρίζει πλήρως τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού: Την **ενθυλάκωση** (encapsulation), τον **πολυμορφισμό** (polymorphism), και την **κληρονομικότητα** (inheritance).

Αυτά τα χαρακτηριστικά της γλώσσας, σε συνδυασμό με τις τεράστιες δυνατότητες που παρέχει, την ευελιξία της, και την απόδοσή της, την καθιστούν σήμερα την πρώτη επιλογή των προγραμματιστών για την κατασκευή μεγάλων και απαιτητικών εφαρμογών.

Ο αντικειμενοστρεφής προγραμματισμός συγκεντρώνει όλα τα στοιχεία του δομημένου και του τμηματικού προγραμματισμού, καταργώντας το χάσμα που υπάρχει μεταξύ δεδομένων και διαδικασιών, και εισάγοντας παράλληλα νέες έννοιες και μια διαφορετική φιλοσοφία για τον σχεδιασμό και την ανάπτυξη προγραμμάτων.

Αντικειμενοστρεφής προγραμματισμός

Μια από τις καθημερινές μου συνήθειες είναι να φτιάχνω τον πρωινό μου καφέ. Ας προσπαθήσουμε να αναλύσουμε λίγο αυτή τη διαδικασία.

Η πρώτη ύλη που διαθέτουμε (τα δεδομένα μας) είναι ο καφές, η ζάχαρη, το νερό, και το γάλα, τα οποία είναι αποθηκευμένα στα ντουλάπια και στο ψυγείο της κουζίνας μας. Εμείς παίρνουμε τα υλικά που χρειαζόμαστε, έχοντας απευθείας πρόσβαση τους αποθηκευτικούς μας χώρους, και ακολουθούμε όλα τα βήματα που χρειάζονται για να φτιάξουμε τον καφέ μας. Ο συγκάτοικός μας έχει και αυτός πρόσβαση στα ίδια υλικά και φτιάχνει και αυτός τον καφέ του ακολουθώντας την ίδια ή διαφορετική διαδικασία.

Παρατηρούμε ότι με την παραπάνω διαδικασία ...

- Τα υλικά μας (δεδομένα) είναι εκτεθειμένα σε όποιον έχει πρόσβαση στους αποθηκευτικούς μας χώρους (ντουλάπια, ψυγείο). Με αυτόν τον τρόπο μπορεί να γίνει αλόγιστη χρήση των υλικών ακόμη και για διαφορετικό σκοπό (π.χ. μπορεί να ξοδέψουμε ζάχαρη και γάλα για να φτιάξουμε ένα κέϊκ).
- Η επεξεργασία των υλικών για την παρασκευή του καφέ δεν είναι αυτοματοποιημένη, και ο καθένας μπορεί να ακολουθεί μια εντελώς διαφορετική διαδικασία.
- Αν αποθηκεύσουμε κάπου αλλού τα υλικά (σε άλλο ντουλάπι), πρέπει να αλλάξουμε και τη διαδικασία επεξεργασίας ώστε να τα παίρνουμε από το νέο αποθηκευτικό χώρο.
- Στην περίπτωση που θέλουμε να φτιάχνουμε διαφορετικά είδη καφέ (π.χ. καπουτσίνο, φραπέ, εσπρέσο), θα πρέπει κάθε φορά να χρησιμοποιούμε διαφορετική διαδικασία επεξεργασίας.
- Αν θελήσουμε να φτιάξουμε καφέ κάπου αλλού και όχι στο σπίτι μας, θα πρέπει κατ'αρχήν να μάθουμε πού βρίσκονται τα υλικά μας (τα δεδομένα μας), ενώ ίσως χρειαστεί να τροποποιήσουμε τη διαδικασία παρασκευής του καφέ ανάλογα με τις δυνατότητες του χώρου στον οποίο βρισκόμαστε. Για παράδειγμα, θα ακολουθήσουμε διαφορετική διαδικασία για να ζεστάνουμε το νερό ανάλογα με το αν ο χώρος διαθέτει υγραέριο, ηλεκτρική κουζίνα, ή φούρνο μικροκυμάτων.

Αν και έδειξα αρκετή υπομονή, η κατάσταση είχε φτάσει στο απροχώρητο. Υπήρχαν φορές που ανακάλυπτα την τελευταία στιγμή ότι τελείωσε ο καφές γιατί τον είχε χρησιμοποιήσει ο συγκάτοικός μου, άλλες φορές δεν έβρισκα ζάχαρη επειδή έφτιαχνε γλυκά η φίλη μου, και άλλωτε δεν έβρισκα καθόλου τα υλικά για καφέ επειδή η καθαρίστρια αποφάσιζε να τα τακτοποιήσει σε κάποιο διαφορετικό ντουλάπι. Και σαν να μην έφταναν αυτά, κάθε φορά ο καφές είχε διαφορετική γεύση, είτε γιατί έπεφτε λίγο περισσότερη ζάχαρη είτε γιατί δεν υπολόγιζα σωστά τη δόση του γάλακτος!

Σκέφτηκα, λοιπόν, μήπως υπάρχει κάποια συσκευή που να φτιάχνει καφέ και να με απαλλάξει από όλους αυτούς του μπελάδες. Έτσι αποφάσισα να αγοράσω μια μηχανή για καφέ (καφετέρια) και αμέσως όλα μου τα προβλήματα λύθηκαν.

- Τα υλικά πλέον βρίσκονταν μέσα στην καφετιέρα, και ήταν μάλλον απίθανο να ανοίξει κάποιος τη μηχανή για να βγάλει λίγο καφέ ή ζάχαρη.
- Ποτέ πια δεν χρειάστηκε να αναζητήσω τον καφέ, τη ζάχαρη, ή το γάλα· μάλιστα, η καφετέρια με ειδοποιούσε όταν κόντευαν να τελειώσουν τα υλικά που περιέχει.
- Κάθε φορά πατούσα απλώς το κουμπί και ο καφές ετοιμαζόταν χωρίς ιδιαίτερο κόπο και, το κυριότερο, πάντα με τις ίδιες αναλογίες.
- Η μηχανή είχε τη δυνατότητα να παρασκευάζει διάφορα είδη καφέ. Εγώ απλώς πατούσα το σωστό κουμπί, χωρίς να με απασχολεί η διαφορετική διαδικασία που ακολουθούσε η μηχανή για να φτιάξει τον καφέ που ζητήσα.
- Και φυσικά, όταν πήγαινα σε κάποιον άλλο χώρο, απλώς έπαιρνα την καφετέρια μαζί μου ή φρόντιζα να υπάρχει μια ακριβώς ίδια.

Ό,τι διαβάσατε πιο πάνω, αντιστοιχεί στις βασικές αρχές και στα πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού!

Κλάσεις και αντικείμενα

Παρατηρούμε ότι από τη στιγμή που αποφασίσαμε να κατασκευάσουμε (ή να αγοράσουμε) ένα αντικείμενο (καφετιέρα) το οποίο εκτελεί την εργασία που μέχρι τώρα υλοποιούσαμε με αποσπασματικές διαδικασίες, λύθηκαν τα περισσότερα προβλήματα που είχαμε.

Μήπως λοιπόν η καλύτερη προσέγγιση και στην κατασκευή των προγραμμάτων βρίσκεται στη χρήση **αντικειμένων**; Τα προγράμματα με τις παραδοσιακές, διαδικασιακές, τεχνικές προγραμματισμού βασίζονται περισσότερο στις διαδικασίες, παρά στα δεδομένα. Ας επισημάνουμε επίσης ότι ο φυσικός και ο τεχνητός κόσμος στον οποίο ζούμε είναι πιο κοντά στη φιλοσοφία του **αντικειμενοστρεφούς**, παρά του διαδικασιακού προγραμματισμού.

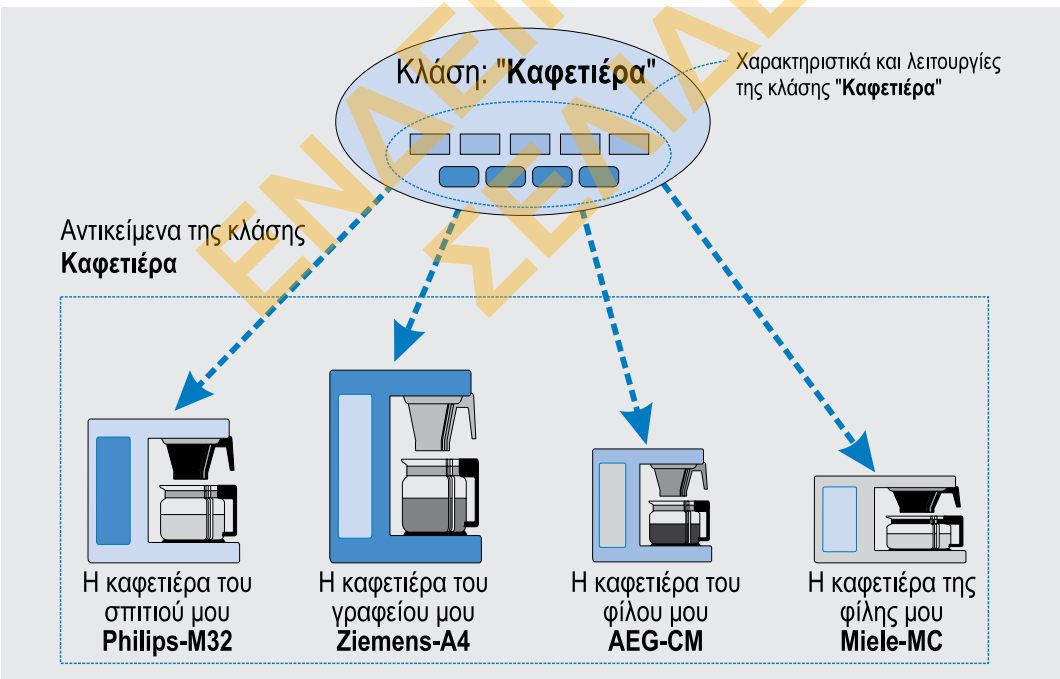
Με μια κλασική διαδικασιακή γλώσσα προγραμματισμού, όπως η C, αναγκαζόμαστε να προσαρμόσουμε το πρόβλημα που καλούμαστε να λύσουμε στις δομές και στις δυνατότητες που παρέχει η γλώσσα. Στις γλώσσες αντικειμενοστρεφούς προγραμματισμού κάνουμε ακριβώς το αντίθετο: Προσαρμόζουμε τη γλώσσα και τις δομές της ώστε να μας βοηθούν στη διαχείριση του συγκεκριμένου προβλήματος.

Στις αντικειμενοστρεφείς γλώσσες προγραμματισμού αντιμετωπίζουμε τις διαφορές οντότητες του προγράμματος ως αντικείμενα. Ο κώδικας του προγράμματος καθορίζει τις διαδικασίες που επενεργούν σε αυτά τα αντικείμενα.

Οι **κλάσεις** (classes) και τα **αντικείμενα** (objects) είναι έννοιες πάνω στις οποίες βασίζεται όλη η φιλοσοφία και τα χαρακτηριστικά των γλωσσών αντικειμενοστρεφούς προγραμματισμού.

Η **κλάση** (class) είναι μια έννοια η οποία προσδιορίζει μια κατηγορία αντικειμένων, και ταυτόχρονα περιγράφει τα κοινά χαρακτηριστικά τους. Με άλλα λόγια μπορούμε να πούμε ότι μια κλάση είναι το σύνολο των **προδιαγραφών** κατασκευής αντικειμένων αυτής της κλάσης.

Για παράδειγμα, η κλάση «Καφετιέρα» που εμφανίζεται στο Σχήμα 20.2 περιγράφει μια κατηγορία συσκευών για την παρασκευή καφέ. Όλα τα αντικείμενα αυτής της κλάσης έχουν κάποια κοινά χαρακτηριστικά και λειτουργίες. Όλες οι καφετιέρες έχουν χώρο για νερό, καφέ, γάλα, και ζάχαρη. Επίσης, όλες διαθέτουν λειτουργίες για να φτιάχνουν καφέ σκέτο, γλυκό, ή μέτριο. Ένα αντικείμενο-καφετιέρα έχει επομένως κάποια **χαρακτηριστικά** (χρώμα, υλικά, κ.λ.π.) και κάποιες **λειτουργίες**, οι οποίες διαχειρίζονται τα υλικά για να φτιάχνουν καφέ! Στο σχήμα εμφανίζονται επίσης τέσσερα αντικείμενα της κλάσης «Καφετιέρα». Κάθε ένα αποτελεί μια διαφορετική οντότητα με διαφορετικές τιμές χαρακτηριστικών (χρώμα, ποσότητες υλικών, κ.λπ) αλλά με τις ίδιες λειτουργίες.



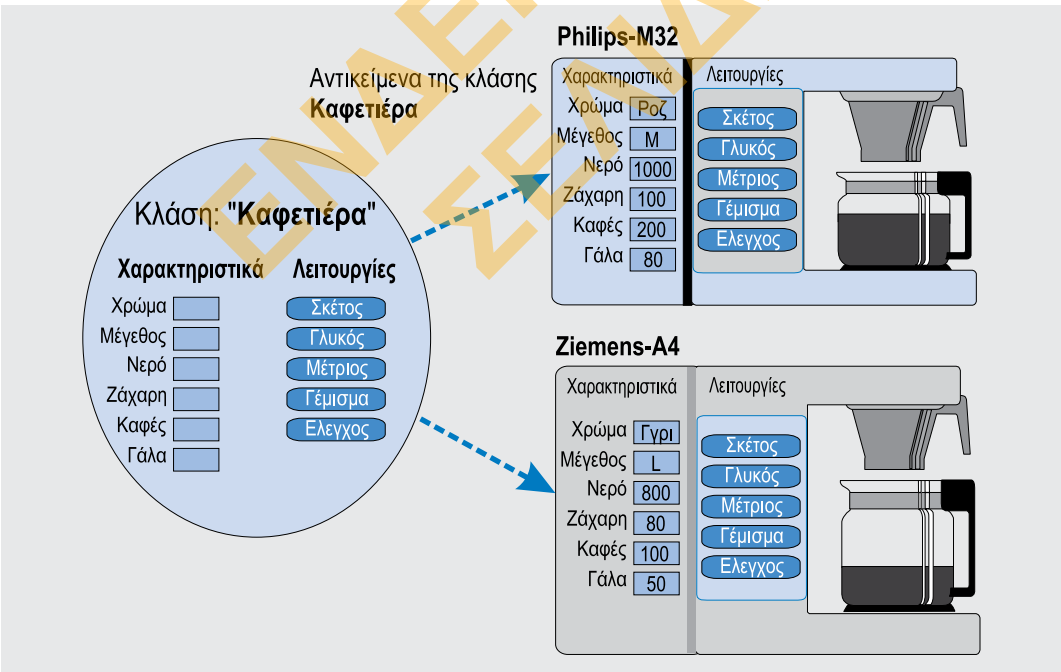
Σχήμα 20.2 Παράδειγμα κλάσης: η κλάση «Καφετιέρα»

Ένα **αντικείμενο** (object) μιας κλάσης έχει όλα τα χαρακτηριστικά και τις λειτουργίες της κλάσης στην οποία ανήκει. Το αντικείμενο λέμε ότι αποτελεί ένα **στιγμιότυπο** (instance) μιας κλάσης.

Μια κλάση δεν αναφέρεται σε κανένα συγκεκριμένο αντικείμενο, αλλά είναι μια γενική έννοια που προσδιορίζει μια κατηγορία αντικειμένων. Για παράδειγμα, η κλάση «Αυτοκίνητο» δεν αναφέρεται σε κανένα συγκεκριμένο αυτοκίνητο, αλλά αποτελεί μια έννοια που προσδιορίζει μια κατηγορία οχημάτων.

Στο Σχήμα 20.3 φαίνεται πιο αναλυτικά η κλάση «Καφετιέρα» και δύο αντικείμενα που ανήκουν σε αυτή την κλάση.

Στη συγκεκριμένη περίπτωση, η κλάση προσδιορίζει τα χαρακτηριστικά, όπως το χρώμα, το μέγεθος, και οι χωρητικότητες νερού, ζάχαρης, καφέ, και γάλακτος που θα έχουν τα αντικείμενα, χωρίς όμως να αναφέρονται συγκεκριμένες τιμές. Παρατηρούμε ότι τα δύο αντικείμενα της κλάσης έχουν όλα τα παραπάνω χαρακτηριστικά, **αλλά με συγκεκριμένες τιμές το καθένα**. Επίσης, στην κλάση «Καφετιέρα» ορίζονται και οι λειτουργίες που θα εκτελούν τα αντικείμενα αυτής της κλάσης: Παρασκευή τριών ειδών καφέ (σκέτος, μέτριος, γλυκός), συμπλήρωση της καφετιέρας με υλικά (καφέ, ζάχαρη, κ.λπ.), καθώς και έλεγχος για επάρκεια υλικών.



Σχήμα 20.3 Αντικείμενα της κλάσης «Καφετιέρα»

Ανασκόπηση Κεφαλαίου 20

- Η C++ είναι μια αντικειμενοστρεφής μεταγλωττιζόμενη γλώσσα, τα χαρακτηριστικά της οποίας ευνοούν τον τμηματικό (ή αρθρωτό) προγραμματισμό. Ο βασικός κορμός της C++ είναι ίδιος με της C. Κάθε πρόγραμμα της C, χωρίς σχεδόν καμία μετατροπή, μπορεί να μεταγλωττιστεί από οποιονδήποτε μεταγλωττιστή της C++.
- Στη C++, όπως και στη C, αρκετές λειτουργίες υλοποιούνται από τα αντικείμενα και τις συναρτήσεις βιβλιοθήκης που τη συνοδεύουν.
- Ο αντικειμενοστρεφής προγραμματισμός βασίζεται στη μοντελοποίηση του προβλήματος μέσω της χρήσης αντικειμένων.
- Ο ακρογωνιαίος λίθος του αντικειμενοστρεφούς προγραμματισμού είναι οι κλάσεις από τις οποίες δημιουργούνται τα αντικείμενα. Οι κλάσεις είναι μια μετεξέλιξη των δομών.
- Κάθε αντικείμενο ανήκει σε μια κλάση η οποία προσδιορίζει τα χαρακτηριστικά και τις λειτουργίες του.
- Μπορούμε να πούμε ότι μια κλάση ορίζει τις προδιαγραφές των χαρακτηριστικών και των λειτουργιών των αντικειμένων που ανήκουν σε αυτή την κλάση, ή πρόκειται να δημιουργηθούν από αυτή την κλάση.
- Τα κύρια χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού είναι η ενθυλάκωση, η κληρονομικότητα, και ο πολυμορφισμός.
- Η κληρονομικότητα είναι το βασικό χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού που επιτρέπει την επαναχρησιμοποίηση κώδικα.
- Το αντικείμενο `cout` χρησιμοποιείται για την εμφάνιση πληροφοριών στην οθόνη. Για την αποστολή πληροφοριών στο αντικείμενο `cout`, χρησιμοποιείται ο τελεστής εισαγωγής `<<`.
- Ο χειριστής `endl` του αντικειμένου `cout` επιβάλλει μια αλλαγή γραμμής στην οθόνη.
- Το αντικείμενο `cin` χρησιμοποιείται για την ανάγνωση δεδομένων (χαρακτήρων και αριθμών) από το πληκτρολόγιο. Για τη λήψη δεδομένων από το αντικείμενο `cin` χρησιμοποιείται ο τελεστής εξαγωγής `>>`.
- Η C++, πέρα από τα αντικειμενοστρεφή χαρακτηριστικά που διαθέτει, εισάγει νέα χαρακτηριστικά ακόμα και στις παραδοσιακές διαδικασιακές λειτουργίες τις οποίες κληρονόμησε από τη C.
- Σε κάθε περίπτωση, το επόμενο βήμα σας θα είναι σίγουρα προς την κατεύθυνση του αντικειμενοστρεφούς προγραμματισμού!

Ασκήσεις Κεφαλαίου 20

20.1 Εξηγήστε τη λειτουργία του επόμενου προγράμματος της C++: ★★

```
int main(void)
{
    int a,b;
    cin>>a>>b;
    if (a>b)
        cout<<a;
    else
        cout<<b;
    return 0;
}
```

20.2 Ποια από τα παρακάτω αληθεύουν; ★

- ☐ Η C++ είναι μια επέκταση της C, στην οποία έχουν προστεθεί αντικειμενοστρεφή χαρακτηριστικά.
- ☐ Όλα τα αντικείμενα μιας κλάσης έχουν τα ίδια χαρακτηριστικά και λειτουργίες.
- ☐ Η κληρονομικότητα είναι ένα χαρακτηριστικό του πολυμορφισμού.
- ☐ Με τη C++ μπορούμε να κατασκευάζουμε μόνο αντικειμενοστρεφή προγράμματα.
- ☐ Ένα πρόγραμμα σε C μπορεί να μεταγλωττιστεί από οποιονδήποτε μεταγλωττιστή της C++, με ελάχιστες ή καθόλου αλλαγές.
- ☐ Στη C++ χρησιμοποιείται το αντικείμενο `cout` για την έξοδο πληροφοριών στην οθόνη.
- ☐ Στη C++ η μεταβίβαση παραμέτρων γίνεται όπως και στη C, μόνο κατ' αξία.
- ☐ Στη C++, για την αποθήκευση συμβολοσειρών χρησιμοποιούμε την κλάση `string`, και όχι πίνακες χαρακτήρων.
- ☐ Στη C++ δεν μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις `printf()` και `scanf()`.
- ☐ Η C++ διαθέτει λογικό τύπο δεδομένων.

Παράρτημα

A

Συναρτήσεις
βιβλιοθήκης
της C

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Στο παράρτημα αυτό θα βρείτε τις πιο συχνά χρησιμοποιούμενες συναρτήσεις της C. Οι συναρτήσεις είναι ομαδοποιημένες σύμφωνα με το αρχείο κεφαλίδας στο οποίο ορίζονται.

Πέρα από το πρωτότυπο και την επεξήγηση της λειτουργίας κάθε συνάρτησης, παρέχονται χαρακτηριστικά παραδείγματα χρήσης της.

Αναφέρεται επίσης το κεφάλαιο του βιβλίου στο οποίο εξετάζεται αναλυτικά η κάθε συνάρτηση.

Η καθιερωμένη βιβλιοθήκη της C

Η καθιερωμένη βιβλιοθήκη (standard library) της C περιλαμβάνει κάθε είδους συναρτήσεις, οι οποίες είναι ομαδοποιημένες ανά κατηγορίες και δηλώνονται σε διαφορετικά αρχεία κεφαλίδων (header files). Θεωρήσαμε πιο χρήσιμο να αναφέρουμε τις συναρτήσεις ανάλογα με το αρχείο κεφαλίδας στο οποίο δηλώνονται. Για να χρησιμοποιήσουμε οποιαδήποτε από αυτές τις συναρτήσεις, θα πρέπει στον κώδικα του προγράμματος να συμπεριλάβουμε με την οδηγία **#include** το αρχείο κεφαλίδας όπου δηλώνεται.

Σε ορισμένα προγραμματιστικά περιβάλλοντα, αρκετές συναρτήσεις βιβλιοθήκης δηλώνονται και στα αρχεία κεφαλίδων **stdlib.h** ή **stdio.h** εκτός από το αρχείο κεφαλίδας το οποίο καθορίζει το πρότυπο ANSI. Σε αυτή την περίπτωση, αρκεί να συμπεριλάβουμε αυτά τα αρχεία στον κώδικά μας για να χρησιμοποιήσουμε τις περισσότερες συναρτήσεις βιβλιοθήκης.

<ctype.h>

int islower(int c);

Η συνάρτηση **islower()** επιστρέφει μη μηδενική τιμή αν ο χαρακτήρας που μεταβιβάζεται στην παράμετρο **c** είναι πεζός χαρακτήρας, διαφορετικά επιστρέφει τιμή 0.

Παράδειγμα:

Το επόμενο πρόγραμμα ζητάει συνεχώς χαρακτήρες μέχρι να πληκτρολογηθεί ένας κεφαλαίος χαρακτήρας.

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    char ch;
    do
    {
        ch=getch();
        putchar(ch);
    } while (islower(ch));
    return 0;
}
```

int isupper(int c);

Η συνάρτηση **isupper()** επιστρέφει μη μηδενική τιμή αν ο χαρακτήρας που μεταβιβάζεται στην παράμετρο **c** είναι κεφαλαίος χαρακτήρας, διαφορετικά επιστρέφει τιμή 0.

int isprint(int c);

Η συνάρτηση **isprint()** επιστρέφει μη μηδενική τιμή αν ο χαρακτήρας που μεταβιβάζεται στην παράμετρο **c** είναι εκτυπώσιμος χαρακτήρας (συμπεριλαμβανομένου και του διαστήματος), διαφορετικά επιστρέφει τιμή 0.

int tolower(int c);

Η συνάρτηση **tolower()** επιστρέφει τον αντίστοιχο πεζό χαρακτήρα από αυτόν που μεταβιβάζεται στην παράμετρο **c**. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

Παράδειγμα

Η πρόταση,

```
tolower('A')
```

επιστρέφει τον χαρακτήρα 'a'.

int toupper(int c);

Η συνάρτηση **toupper()** επιστρέφει τον αντίστοιχο κεφαλαίο χαρακτήρα από αυτόν που μεταβιβάζεται στην παράμετρο **c**. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

Παράδειγμα

Η πρόταση,

```
toupper('a')
```

επιστρέφει τον χαρακτήρα 'A'.

<math.h>

double exp(double x);

Η συνάρτηση **exp()** επιστρέφει τη βάση των φυσικών λογαρίθμων e υψωμένη στη δύναμη της παραμέτρου **x** (e^x).

Παράδειγμα

Η επόμενη πρόταση καταχωρίζει στη μεταβλητή **f** την τιμή e^4 .

```
f=exp(4)
```

double log(double x);

Η συνάρτηση **log()** επιστρέφει τον φυσικό λογάριθμο της παραμέτρου **x**. Η συνάρτηση επιστρέφει μήνυμα λάθους αν η τιμή της παραμέτρου **x** είναι μικρότερη ή ίση με 0.

Παράδειγμα

Το επόμενο πρόγραμμα εμφανίζει τους φυσικούς λογαρίθμους των αριθμών από το 1 μέχρι το 10.

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

```
int main(void)
{
    int i;
    for (i=1;i<=10;i++)
        printf("%d %f\n",i,log((double)i));
    return 0;
}
```

Όπως βλέπετε, γίνεται μετατροπή³⁴ του ορίσματος **i** σε τύπο **double** ώστε να συμφωνεί με τον τύπο της παραμέτρου που απαιτεί η συνάρτηση **log()**.

double log10(double x);

Η συνάρτηση **log10()** επιστρέφει τον δεκαδικό λογάριθμο της παραμέτρου **x**. Η συνάρτηση επιστρέφει μήνυμα λάθους αν το **x** είναι μικρότερο από ή ίσο με 0.

double pow(double x, double y);

Η συνάρτηση **pow()** επιστρέφει την τιμή της πρώτης παραμέτρου **x** υψωμένη στη δύναμη της δεύτερης παραμέτρου **y** (x^y).

Παράδειγμα

Το επόμενο πρόγραμμα εμφανίζει τις δυνάμεις του 2 από το 0 έως το 7 ($2^0, 2^1, \dots, 2^7$).

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    int i;
    for (i=0;i<=7;i++)
        printf("%d %f\n",i,pow(2,i));
    return 0;
}
```

double sqrt(double x);

Η συνάρτηση **sqrt()** επιστρέφει ως τιμή την τετραγωνική ρίζα της παραμέτρου **x**. Η συνάρτηση επιστρέφει μήνυμα λάθους αν το **x** είναι μικρότερο από 0.

Παράδειγμα

Το παρακάτω τμήμα κώδικα υπολογίζει και εμφανίζει τις ρίζες μιας εξίσωσης β' βαθμού $ax^2+bx+c=0$.

```
float a,b,c,d;
scanf("%f %f %f",&a,&b,&c);
d=b*b-4*a*c;
```

³⁴ Σε πολλούς μεταγλωττιστές η μετατροπή γίνεται αυτόματα, και δεν είναι απαραίτητη η χρήση ρητής μετατροπής τύπου (type casting).

Παράρτημα

Το ολοκληρωμένο περιβάλλον του Code::Blocks

B

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Στο παράρτημα αυτό παρουσιάζεται το ολοκληρωμένο περιβάλλον ανάπτυξης Code::Blocks.

Δίνονται οδηγίες για τη σωστή εγκατάσταση του Code::Blocks, αναφέρονται οι βασικές λειτουργίες του περιβάλλοντος, καθώς και ο τρόπος παραμετροποίησής του για τη χρήση ελληνικών χαρακτήρων.

ΕΝΔΕΙΚΤΙΚΑ
ΣΕΛΙΔΕΣ

Το ολοκληρωμένο περιβάλλον ανάπτυξης Code::Blocks

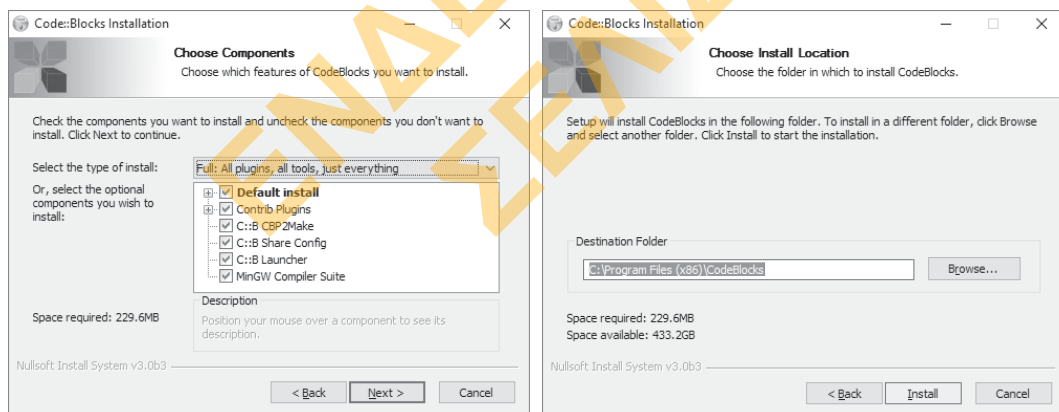
Το **Code::Blocks** είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης για τις γλώσσες C και C++. Χρησιμοποιεί τον μεταγλωττιστή GCC και παράγει 32μπιτες εφαρμογές (Win32) τόσο για τη γραμμή εντολών (console applications) όσο και για το περιβάλλον γραφικών των Windows (GUI applications). Το περιβάλλον **Code::Blocks** διατίθεται δωρεάν υπό τους όρους της γενικής άδειας χρήσης της GNU (GPL v3.0).

Στο συνοδευτικό υλικό θα βρείτε το πρόγραμμα εγκατάστασης του ολοκληρωμένου περιβάλλοντος **Code::Blocks**. Επίσης, συνδέσμους για τη λήψη του προγράμματος θα βρείτε στην ιστοσελίδα της εφαρμογής <http://www.codeblogs.org>, καθώς και στον διαδικτυακό τόπο αυτού του βιβλίου <http://c.bytes.gr>.

Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του **Code::Blocks**.

Εγκατάσταση του Code::Blocks

Για να εγκαταστήσετε το ολοκληρωμένο περιβάλλον του **Code::Blocks** σε περιβάλλον Windows, εκτελέστε το αρχείο **CodeBlocks_setup.exe** από τον φάκελο του συνοδευτικού υλικού. Αν θέλετε την πιο πρόσφατη έκδοση του περιβάλλοντος, ή κάποια άλλη έκδοσή του (π.χ. για Linux) επισκεφτείτε την ιστοσελίδα <http://www.codeblogs.org>, και κατεβάστε από εκεί το κατάλληλο πρόγραμμα εγκατάστασης.



- 1 Επιλέξτε τον τύπο «πλήρους» εγκατάστασης (**Full**).
- 2 Αποδεχθείτε όλα τα προτεινόμενα επιλεγμένα τμήματα της εφαρμογής.
- 3 Αποδεχθείτε τον προτεινόμενο φάκελο εγκατάστασης της εφαρμογής.
- 4 Πατήστε στο κουμπί **Install** για να ξεκινήσει η εγκατάσταση.

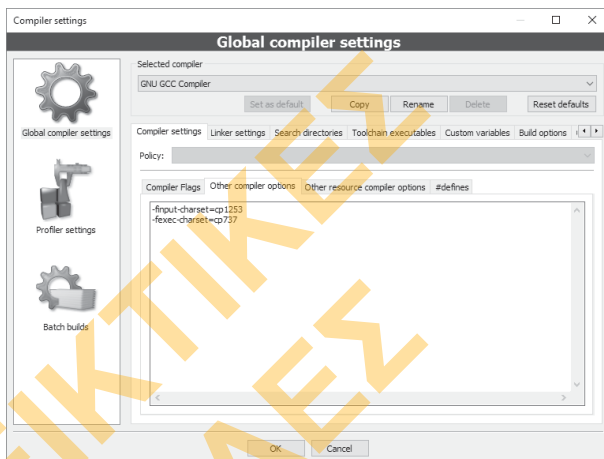
Οι βασικές λειτουργίες του Code::Blocks

Στη συνέχεια αναφέρουμε τις βασικές λειτουργίες του περιβάλλοντος ώστε να μπορείτε να δοκιμάζετε τα παραδείγματα του βιβλίου ή τα προγράμματα που δημιουργείτε μόνοι σας.

Παραμετροποίηση του μεταγλωτιστή για χρήση ελληνικών χαρακτήρων

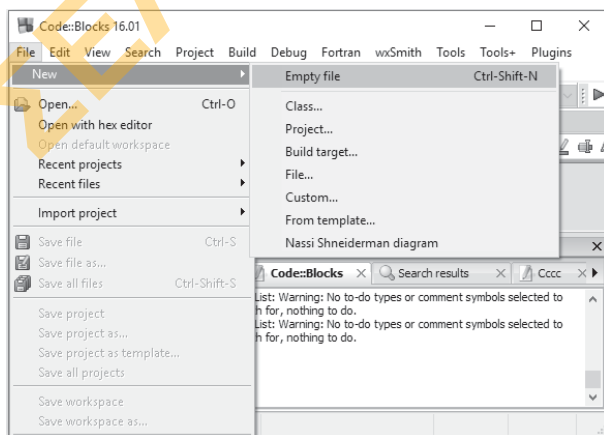
Ανοίξτε το μενού **Settings**, και πατήστε στην επιλογή **Compiler**. Επιλέξτε τη καρτέλα **Compiler Settings** και μέσα σε αυτήν επιλέξτε την καρτέλα **Other compiler options**. Στο πλαίσιο κειμένου που θα παρουσιαστεί πληκτρολογήστε τις ακόλουθες γραμμές:

-finput-charset=cp1253
-fexec-charset=cp737



Δημιουργία νέου αρχείου πηγαίου κώδικα ή έργου

Για τη δημιουργία ενός νέου αρχείου πηγαίου κώδικα της C, επιλέξτε **File->New->Empty file**. Αν θέλετε να δημιουργήσετε μια εφαρμογή με πολλά αρχεία (δηλαδή ένα έργο), επιλέξτε **File->New->Project**. Όλα τα παραδείγματα του βιβλίου χρησιμοποιούν μόνο ένα αρχείο πηγαίου κώδικα, οπότε αυτή η επιλογή θα σας χρειαστεί μόνο αν ασχοληθείτε με πιο πολύπλοκες εφαρμογές.



Αντιστοίχιση ελληνικών και αγγλικών όρων

ΕΝΔΕΙΚΤΙΚΕΝ
ΣΕΛΙΔΕΝ



Είναι αλήθεια ότι ο προγραμματισμός και η Αγγλική γλώσσα είναι άρρηκτα συνδεδεμένα.

Η ονοματολογία των εντολών όλων των γλωσσών προγραμματισμού, αλλά και η ορολογία της επιστήμης των υπολογιστών βασίζεται σε αυτήν.

Σε αυτό το βιβλίο χρησιμοποιείται ελληνική ορολογία η οποία όμως δεν θα σας φανεί χρήσιμη όταν αναζητήσετε πληροφορίες στη διεθνή βιβλιογραφία και στο Διαδίκτυο.

Έτσι θεώρησα σωστό να συμπεριλάβω ένα μικρό «λεξικό» των ελληνικών όρων και τους αντίστοιχους αγγλικούς.

Αντιστοίχιση ελληνικών και αγγλικών όρων

Αναφορά	Reference
Αγκύλη	Bracket
Αθροιστής	Accumulator
Αλφαριθμητικό	Character string
Αναγνωριστικό	Identifier
Αναδρομή	Recursion
Αναζήτηση	Searching
Αναφορική μεταβλητή	Reference variable
Αντικειμενοστρεφής	Object oriented
Αντικειμενοστρεφής Προγραμματισμός	Object oriented programming (OOP)
Αντικειμενοστρεφής σχεδιασμός	Object oriented design
Αντικείμενο	Object
Απαρίθμηση	Enumeration
Απλά συνδεδεμένη λίστα	Simply linked list
Αποσφαλμάτωση	Debugging
Αρχείο	File
Αρχείο κειμένου	Text file
Αρχική τιμή	Initial value
Βιβλιοθήκη	Library
Βρόχος	Loop
Γράφος	Graph
Δείκτης	Pointer
Δεκαεξαδικό	Hexadecimal
Δήλωση συνάρτησης	Function declaration
Δηλωτική πρόταση	Declaration statement
Διατεταγμένη συνδεδεμένη λίστα	Ordered linked list
Διεύθυνση	Address
Διεύθυνση μνήμης	Memory address
Διπλά συνδεδεμένη λίστα	Doubly linked list
Δομή	Structure
Δυαδική Αναζήτηση	Binary search