

**Μέθοδος διδασκαλίας
από τον ειδικό της C++**

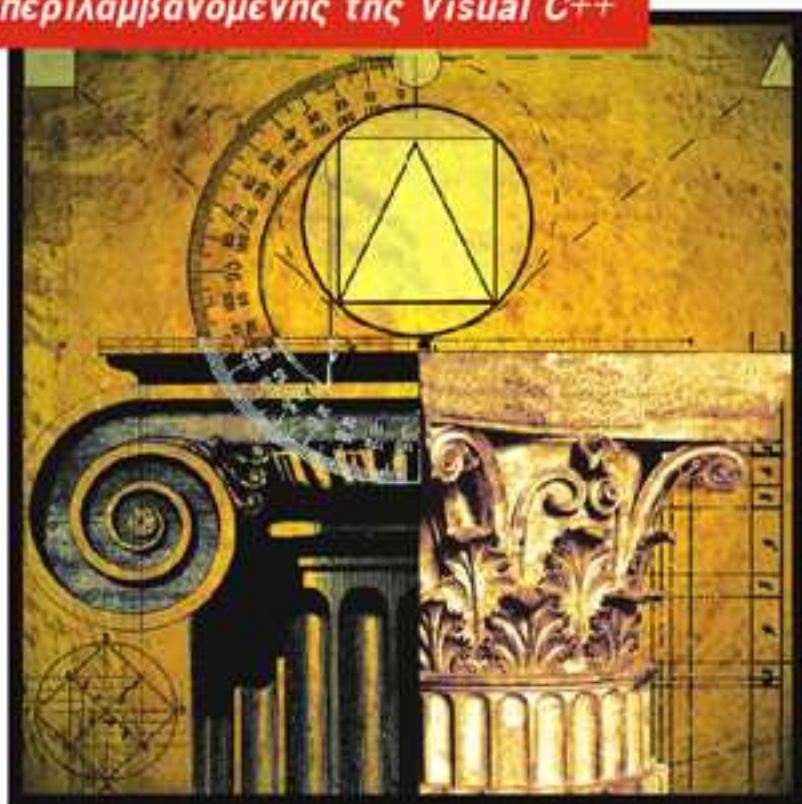
Μάθετε τη C++



ΠΕΡΙΧΕΙ
ΔΩΡΕΑΝ CD-ROM

από το ΜΗΔΕΝ

Για όλους τους μεταγλωττιστές της C++
συμπεριλαμβανομένης της Visual C++



3η ΑΜΕΡΙΚΑΝΙΚΗ ΕΚΔΟΣΗ

**Καλύπτει τα Διεθνή Πρότυπα
για τη C++**

Διδάσκει ολόκληρη τη γλώσσα C++,
από τα βασικά μέχρι τις προηγμένες
δυνατότητες

Γεμάτο συμβουλές και τεχνικές από
ειδικούς, καθώς και εκατοντάδες
παραδείγματα

Περιλαμβάνει την Standard Template
Library (STL) — καθιερωμένη
βιβλιοθήκη προτύπων

Το συνοδευτικό CD περιέχει και το
μεταγλωττιστή Borland C++ 5.5

Herbert Schildt

Συγγραφέας πολλών μπεστ-σέλερ για τον προγραμματισμό, με πωλήσεις άνω των 3 εκατομμυρίων βιβλίων



Περιεχόμενα

<i>Πρόλογος</i>	<i>xvii</i>
1 ■■■ Η ιστορία της C++	27
Η καταγωγή της C++	28
Η δημιουργία της C	28
Κατανόηση της ανάγκης της C++	30
Η γέννηση της C++	31
Η εξέλιξη της C++	32
Τι είναι ο αντικειμενοστρεφής προγραμματισμός	33
Ενθυλάκωση	34
Πολυμορφισμός	34
Κληρονομικότητα	35
Υλοποίηση του αντικειμενοστρεφούς προγραμματισμού στη C++	35
Η σχέση της C++ με την Java και τη C#	36
2 ■■■ Μια επισκόπηση της C++	37
Το πρώτο σας πρόγραμμα C++	38
Καταχώριση του προγράμματος	39
Μεταγλώττιση του προγράμματος	39
Εκτέλεση του προγράμματος	40
Εξήγηση του προγράμματος γραμμή προς γραμμή	41
Χειρισμός συντακτικών σφαλμάτων	43
Ένα δεύτερο απλό πρόγραμμα	44
Ένα πιο πρακτικό παράδειγμα	45
Ένας νέος τύπος δεδομένων	46

Μια γρήγορη ανασκόπηση.....	47
Συναρτήσεις.....	48
Ένα πρόγραμμα με δύο συναρτήσεις.....	48
Ορίσματα συναρτήσεων.....	50
Συναρτήσεις που επιστρέφουν τιμές.....	52
Η συνάρτηση main().....	53
Γενική μορφή των συναρτήσεων της C++.....	54
Μερικές επιλογές εξόδου.....	54
Δύο απλές εντολές.....	55
Η εντολή if.....	55
Ο βρόχος for.....	56
Τμήματα (μπλοκ) κώδικα.....	57
Ελληνικά ερωτηματικά και θέση εντολών.....	58
Τοποθέτηση εντολών σε εσοχές.....	59
Δεσμευμένες λέξεις της C++.....	59
Αναγνωριστικά της C++.....	60
Η καθιερωμένη βιβλιοθήκη της C++.....	60
3 ■■■ Βασικοί τύποι δεδομένων	63
Δήλωση μεταβλητών.....	65
Τοπικές μεταβλητές.....	65
Τυπικές παράμετροι.....	66
Καθολικές μεταβλητές.....	67
Μερικοί τροποποιητές τύπων.....	68
Κυριολεκτικά.....	72
Δεκαεξαδικά και οκταδικά κυριολεκτικά.....	73
Αλφαριθμητικά κυριολεκτικά.....	74
Ακολουθίες χαρακτήρων διαφυγής.....	74
Απόδοση αρχικών τιμών σε μεταβλητές.....	75
Τελεστές.....	77
Αριθμητικοί τελεστές.....	77
Αύξηση και μείωση.....	78
Πώς πήρε η C++ το όνομά της.....	80
Σχεσιακοί και λογικοί τελεστές.....	80
Παραστάσεις.....	83
Μετατροπή τύπων σε παραστάσεις.....	83
Μετατροπή από και προς τον τύπο bool.....	84
Ρητές μετατροπές.....	84

Κενά διαστήματα και παρενθέσεις.....	85
4 ■■■ Εντολές ελέγχου προγράμματος	87
Η εντολή if.....	88
Η παράσταση υπό συνθήκη.....	89
Ένθετες εντολές if.....	90
Η κλίμακα if-else-if.....	91
Ο βρόχος for.....	93
Μερικές παραλλαγές του βρόχου for	95
Απόντα τμήματα.....	96
Ατέρμων βρόχος.....	97
Βρόχοι χρονικής καθυστέρησης.....	97
Η εντολή switch	97
Ένθετες εντολές switch	101
Ο βρόχος while	101
Ο βρόχος do-while	103
Χρήση της continue	104
Χρήση της break για έξοδο από βρόχους.....	105
Ένθετοι βρόχοι.....	106
Χρήση της εντολής goto	107
Συναρμολόγηση όλων των κομματιών	108
5 ■■■ Πίνακες και αλφαριθμητικά.....	111
Μονοδιάστατοι πίνακες	112
Χωρίς έλεγχο ορίων	114
Ταξινόμηση ενός πίνακα	115
Αλφαριθμητικά	116
Ανάγνωση αλφαριθμητικού από το πληκτρολόγιο	117
Μερικές συναρτήσεις βιβλιοθήκης αλφαριθμητικών	119
strcpy()	119
strcat()	120
strcmp()	120
strlen()	121
Χρήση του χαρακτήρα τερματισμού.....	123
Διδιάστατοι πίνακες	124
Πολυδιάστατοι πίνακες	126
Απόδοση αρχικών τιμών σε πίνακα	126
Απόδοση αρχικών τιμών σε πίνακα απροσδιόριστου μεγέθους.....	130

Πίνακες αλφαριθμητικών.....	131
Ένα παράδειγμα χρήσης πινάκων αλφαριθμητικών.....	132
6 ■■■ Δείκτες	135
Τι είναι οι δείκτες;.....	136
Οι τελεστές δεικτών	137
Η σημασία του βασικού τύπου.....	138
Ανάθεση τιμών μέσω δείκτη	140
Παραστάσεις δεικτών.....	140
Αριθμητική δεικτών	141
Συγκρίσεις δεικτών.....	142
Δείκτες και πίνακες.....	143
Χρήση αριθμοδεικτών με δείκτες	146
Είναι πραγματικά εναλλάξιμοι οι δείκτες και οι πίνακες;	146
Δείκτες και κυριολεκτικά αλφαριθμητικά	147
Ένα παράδειγμα σύγκρισης	148
Πίνακες δεικτών.....	149
Η σύμβαση του "μηδενικού" δείκτη	152
Πολλαπλή έμμεση προσπέλαση.....	152
Δείκτες και 16μπιτα περιβάλλοντα	153
Προβλήματα με δείκτες	155
Δείκτες χωρίς αρχική τιμή.....	155
Μη έγκυρες συγκρίσεις δεικτών	156
Μη επαναφορά δείκτη στην αρχική του κατάσταση.....	157
7 ■■■ Συναρτήσεις, μέρος πρώτο: βασικές έννοιες	159
Κανόνες εμβέλειας συναρτήσεων.....	160
Τοπικές μεταβλητές.....	160
Τυπικές παράμετροι	166
Καθολικές μεταβλητές	166
Μεταβίβαση δεικτών και πινάκων.....	169
Κλήση συναρτήσεων με δείκτες	169
Κλήση συναρτήσεων με πίνακες.....	170
Μεταβίβαση αλφαριθμητικών.....	173
Οι παράμετροι argc και argv: ορίσματα της main().....	175
Μεταβίβαση αριθμητικών ορισμάτων γραμμής διαταγών	177
Μετατροπή αριθμητικών αλφαριθμητικών σε αριθμούς.....	178
Η εντολή return.....	179
Επιστροφή από συνάρτηση	179
Επιστροφή τιμών.....	180

Συναρτήσεις void	182
Συναρτήσεις που επιστρέφουν δείκτες.....	183
Πρωτότυπα συναρτήσεων	185
Κεφαλίδες: μια αναλυτική εξέταση	186
Σύγκριση του σύγχρονου τρόπου δήλωσης παραμέτρων συνάρτησης με το παλιό στυλ	186
Αναδρομή.....	187
8 ■■■ Συναρτήσεις, μέρος δεύτερο: αναφορές, υπερφόρτωση, και προεπιλεγμένα ορίσματα	191
Δύο προσεγγίσεις στη μεταβίβαση ορισμάτων	192
Μεταβίβαση ορισμάτων στη C++	192
Χρήση δείκτη για τη δημιουργία κλήσης κατ' αναφορά	193
Αναφορικές παράμετροι	195
Δήλωση αναφορικών παραμέτρων.....	198
Επιστροφή αναφορών	199
Δημιουργία πίνακα καθορισμένων ορίων	202
Ανεξάρτητες αναφορές.....	203
Μερικοί περιορισμοί κατά τη χρήση αναφορών	204
Υπερφόρτωση συναρτήσεων	205
Ο αναχρονισμός της εντολής overload.....	208
Προεπιλεγμένα ορίσματα συνάρτησης	208
Σύγκριση προεπιλεγμένων ορισμάτων και υπερφόρτωσης.....	210
Σωστή χρήση των προεπιλεγμένων ορισμάτων	212
Υπερφόρτωση συναρτήσεων και ασάφεια.....	212
9 ■■■ Περισσότεροι τύποι δεδομένων και τελεστές.....	217
Τα προσδιοριστικά const και volatile	218
const	218
volatile.....	220
Προσδιοριστικά τάξης αποθήκευσης.....	221
auto.....	222
extern.....	222
Στατικές μεταβλητές	224
Μεταβλητές καταχωρητή	227
Η ιστορία του τροποποιητή register	229
Απαριθμήσεις.....	231
typedef.....	234
Περισσότεροι τελεστές	234
Τελεστές χειρισμού bit.....	235

AND, OR, XOR, και NOT	235
Οι τελεστές ολίσθησης	239
Ο τελεστής ?	241
Σύνθετες αναθέσεις	242
Ο τελεστής κόμμα	243
Πολλαπλές αναθέσεις	244
Χρήση του τελεστή sizeof	244
Δυναμική κατανομή με τους τελεστές new και delete	245
Απόδοση αρχικής τιμής σε μνήμη που έχει κατανεμηθεί δυναμικά	247
Κατανομή πινάκων	248
Η προσέγγιση της C στη δυναμική κατανομή: οι συναρτήσεις malloc() και free()	249
Ανασκόπηση προτεραιότητας τελεστών	251
10 ■■■ Δομές και ενώσεις	253
Δομές	254
Προσπέλαση των μελών μιας δομής	256
Πίνακες δομών	257
Ένα απλό παράδειγμα διαχείρισης αποθήκης	257
Μεταβίβαση δομών σε συναρτήσεις	264
Εκχώρηση δομών	265
Δείκτες σε δομές και ο τελεστής βέλος	266
Αναφορές σε δομές	270
Πίνακες και δομές στο εσωτερικό δομών	271
Σύγκριση δομών της C και της C++	272
Πεδία bit	273
Ενώσεις	275
Ανώνυμες ενώσεις	279
Χρήση του τελεστή sizeof για εξασφάλιση της φορητότητας	280
Μετάβαση στον αντικειμενοστρεφή προγραμματισμό	281
11 ■■■ Εισαγωγή στις τάξεις	283
Βασικές έννοιες της τάξης	284
Η γενική μορφή μιας τάξης	288
Μια προσεκτικότερη ματιά στην προσπέλαση των μελών μιας τάξης	288
Συναρτήσεις δόμησης και αποδόμησης	291
Παραμετρικές συναρτήσεις δόμησης	294
Μια εναλλακτική μέθοδος απόδοσης αρχικής τιμής	298
Η σχέση μεταξύ τάξεων και δομών	299

Σύγκριση δομών και τάξεων	300
Η σχέση μεταξύ ενώσεων και τάξεων.....	301
Εμβόλιμες συναρτήσεις	302
Δημιουργία εμβόλιμων συναρτήσεων σε μια τάξη	304
Πίνακες αντικειμένων	305
Απόδοση αρχικών τιμών σε πίνακες αντικειμένων	307
Δείκτες σε αντικείμενα.....	309
Αναφορές σε αντικείμενα	311
12 ■■■ Μια προσεκτικότερη ματιά στις τάξεις	313
Φύλλες συναρτήσεις	314
Υπερφόρτωση συναρτήσεων δόμησης	318
Δυναμική απόδοση αρχικής τιμής.....	320
Εφαρμογή της δυναμικής απόδοσης αρχικής τιμής σε συναρτήσεις δόμησης.....	320
Εκχώρηση αντικειμένων	322
Μεταβίβαση αντικειμένων σε συναρτήσεις	324
Μεταβίβαση αντικειμένων σε συναρτήσεις δόμησης και αποδόμησης	324
Ένα πιθανό πρόβλημα κατά τη μεταβίβαση αντικειμένων	326
Επιστροφή αντικειμένων.....	329
Ένα πιθανό πρόβλημα κατά την επιστροφή αντικειμένων.....	330
Δημιουργία και χρήση συνάρτησης δόμησης αντιγράφου.....	332
Συναρτήσεις δόμησης αντιγράφων και παράμετροι.....	333
Συναρτήσεις δόμησης αντιγράφων και απόδοση αρχικών τιμών.....	335
Χρήση της συνάρτησης δόμησης αντιγράφου κατά την επιστροφή αντικειμένου	336
Συναρτήσεις δόμησης αντιγράφου — δεν υπάρχει ευκολότερος τρόπος;	337
Η δεσμευμένη λέξη this	338
13 ■■■ Υπερφόρτωση τελεστών	341
Υπερφόρτωση τελεστή με χρήση συναρτήσεων-μελών	342
Χρήση συναρτήσεων-μελών για την υπερφόρτωση μοναδιαίων τελεστών	345
Συμβουλές και περιορισμοί για την υπερφόρτωση τελεστών	350
Θέματα προτεραιότητας.....	351
Συναρτήσεις τελεστών μη-μέλη της τάξης	352
Χρήση φίλιας συνάρτησης για την υπερφόρτωση μοναδιαίου τελεστή.....	355
Υπερφόρτωση σχεσιακών και λογικών τελεστών.....	358
Μια προσεκτικότερη ματιά στον τελεστή ανάθεσης τιμής	359
Υπερφόρτωση του τελεστή []	363

Υπερφόρτωση του τελεστή ()	366
Υπερφόρτωση άλλων τελεστών	368
Ένα άλλο παράδειγμα υπερφόρτωσης τελεστή.....	368
14 ■■■ Κληρονομικότητα	375
Εισαγωγή στην κληρονομικότητα.....	376
Έλεγχος πρόσβασης της βασικής τάξης.....	379
Χρήση προστατευμένων μελών	381
Χρήση του προσδιοριστικού protected για την κληρονομικότητα μιας βασικής τάξης.....	385
Ανασκόπηση των προσδιοριστικών public, protected, και private	386
Κληρονόμηση πολλών βασικών τάξεων.....	387
Συναρτήσεις δόμησης και αποδόμησης, και κληρονομικότητα.....	387
Πότε εκτελούνται οι συναρτήσεις δόμησης και αποδόμησης.....	388
Μεταβίβαση παραμέτρων σε συναρτήσεις δόμησης βασικής τάξης.....	391
Παραχώρηση πρόσβασης.....	394
Γραφήματα κληρονομικότητας της C++	397
Εικονικές βασικές τάξεις	398
15 ■■■ Εικονικές συναρτήσεις και πολυμορφισμός.....	403
Δείκτες σε παράγωγους τύπους.....	404
Αναφορές σε παράγωγους τύπους.....	407
Εικονικές συναρτήσεις.....	407
Οι εικονικές συναρτήσεις κληρονομούνται	410
Γιατί να χρησιμοποιούμε εικονικές συναρτήσεις;.....	412
Μια απλή εφαρμογή εικονικών συναρτήσεων	413
Γνήσιες εικονικές συναρτήσεις και αφηρημένες τάξεις.....	417
Πρώιμη και όψιμη σύνδεση	419
Πολυμορφισμός και οπαδοί της "καθαρότητας"	420
16 ■■■ Πρότυπα	421
Γενικές συναρτήσεις	422
Συνάρτηση με δύο γενικούς τύπους	425
Ρητή υπερφόρτωση γενικής συνάρτησης	425
Υπερφόρτωση πρότυπης συνάρτησης.....	427
Χρήση καθιερωμένων παραμέτρων σε συναρτήσεις προτύπων	428
Περιορισμοί των γενικών συναρτήσεων	429
Δημιουργία μιας γενικής συνάρτησης abs().....	429
Γενικές τάξεις.....	430
Ένα παράδειγμα με δύο γενικούς τύπους δεδομένων	434
Δημιουργία μιας γενικής τάξης πίνακα	434

Χρήση άτυπων ορισμάτων σε γενικές τάξεις.....	436
Χρήση προεπιλεγμένων ορισμάτων σε πρότυπες τάξεις.....	438
Ρητές εξειδικεύσεις τάξης	439
17 ■■■ Χειρισμός εξαιρέσεων	441
Βασικά στοιχεία του χειρισμού εξαιρέσεων	442
Οι συναρτήσεις exit() και abort()	444
Σύλληψη τύπων τάξεων	447
Χρήση πολλών εντολών catch	449
Επιλογές χειρισμού εξαιρέσεων.....	451
Σύλληψη όλων των εξαιρέσεων	451
Περιορισμός εξαιρέσεων που μεταβιβάζονται από συνάρτηση	453
Εκ νέου μεταβίβαση εξαίρεσης.....	455
Χειρισμός εξαιρέσεων που μεταβιβάζονται με τον τελεστή new	456
Η εναλλακτική λύση nothrow	457
Υπερφόρτωση των τελεστών new και delete.....	458
Υπερφόρτωση της μορφής nothrow του τελεστή new	462
18 ■■■ Το σύστημα εισόδου/εξόδου της C++.....	465
Σύγκριση παλαιού και νέου συστήματος εισόδου/εξόδου της C++.....	466
Ρεύματα της C++	467
Προκαθορισμένα ρεύματα της C++	468
Τάξεις ρευμάτων της C++.....	468
Υπερφόρτωση των τελεστών εισόδου/εξόδου	469
Δημιουργία συναρτήσεων εισαγωγής	470
Χρήση φίλιων συναρτήσεων για την υπερφόρτωση συναρτήσεων εισαγωγής.....	472
Υπερφόρτωση συναρτήσεων εξαγωγής	473
Σύγκριση εισόδου/εξόδου στη C και τη C++	475
Μορφοποίηση εισόδου/εξόδου	475
Μορφοποίηση με συναρτήσεις-μέλη της τάξης ios.....	476
Χρήση χειριστών εισόδου/εξόδου.....	481
Δημιουργία δικών σας συναρτήσεων χειρισμού ρευμάτων	483
Λειτουργίες εισόδου/εξόδου σε αρχεία.....	485
Άνοιγμα και κλείσιμο αρχείου	485
Ανάγνωση και εγγραφή αρχείων κειμένου	488
Μη μορφοποιημένη δυαδική είσοδος/έξοδος.....	489
Ανάγνωση και εγγραφή μπλοκ δεδομένων	491
Ανίχνευση τέλους αρχείου (EOF)	492
Ένα παράδειγμα σύγκρισης αρχείων.....	493
Άλλες δυαδικές συναρτήσεις εισόδου/εξόδου	495

Τυχαία προσπέλαση	497
Έλεγχος της κατάστασης εισόδου/εξόδου	499
Προσαρμοσμένη είσοδος/έξοδος και αρχεία	501
19 ■ Αναγνώριση τύπου κατά το χρόνο εκτέλεσης και τελεστές ρητής μετατροπής τύπων.....	503
Αναγνώριση τύπου κατά το χρόνο εκτέλεσης (RTTI)	504
Μια απλή εφαρμογή της αναγνώρισης τύπου κατά το χρόνο εκτέλεσης	508
Εφαρμογή του τελεστή typeid σε πρότυπες τάξεις	510
Τελεστές ρητής μετατροπής τύπων	514
dynamic_cast	514
const_cast	519
static_cast	520
reinterpret_cast	521
Σύγκριση της παραδοσιακής μετατροπής και των τεσσάρων τελεστών ρητής μετατροπής τύπων	522
20 ■ Χώροι ονομάτων και άλλα προχωρημένα θέματα	523
Χώροι ονομάτων	524
Βασικές έννοιες των χώρων ονομάτων	524
Η εντολή using	528
Ανώνυμοι χώροι ονομάτων	530
Ο χώρος ονομάτων std	531
Δείκτες σε συναρτήσεις	532
Εύρεση της διεύθυνσης μιας υπερφορτωμένης συνάρτησης	535
Στατικά μέλη τάξης.....	537
Συναρτήσεις-μέλη τύπου const και mutable	539
Συναρτήσεις δόμησης τύπου explicit.....	541
Ένα ενδιαφέρον όφελος από την έμμεση μετατροπή των συναρτήσεων δόμησης.....	542
Σύνταξη της απόδοσης αρχικής τιμής σε μέλος.....	543
Χρήση της εντολής asm	546
Προδιαγραφή σύνδεσης	547
Οι τελεστές δεικτών σε μέλη .* και ->*	548
Δημιουργία συναρτήσεων μετατροπής τύπων	550
21 ■ Εισαγωγή στην Καθιερωμένη Βιβλιοθήκη Προτύπων	553
Μια επισκόπηση της βιβλιοθήκης STL.....	554
Οι τάξεις αποδεκτών	557

Διανύσματα	559
Προσπέλαση διανύσματος μέσω επαναλήπτη	564
Εισαγωγή και διαγραφή στοιχείων διανύσματος	565
Αποθήκευση αντικειμένων τάξης σε διάνυσμα	566
Η δύναμη των επαναληπτών	568
Λίστες	570
Ταξινόμηση λίστας	575
Συγχώνευση μιας λίστας με άλλη	576
Αποθήκευση αντικειμένων τάξης σε λίστα	577
Χάρτες	579
Αποθήκευση αντικειμένων τάξης σε χάρτη	584
Αλγόριθμοι	585
Καταμέτρηση	589
Αφαίρεση και αντικατάσταση στοιχείων	590
Αντιστροφή ακολουθίας	592
Μετασχηματισμός ακολουθίας	592
Εξερεύνηση των αλγορίθμων	594
Η τάξη string	594
Μερικές συναρτήσεις-μέλη της τάξης string	598
Τοποθέτηση αλφαριθμητικών σε άλλους αποδέκτες	603
Τελικές σκέψεις για τη βιβλιοθήκη STL	603
22 ■■■ Ο προεπεξεργαστής της C++	605
#define	606
Μακροεντολές-συναρτήσεις	608
#error	610
#include	610
Οδηγίες μεταγλώττισης υπό συνθήκη	611
#if, #else, #elif, και #endif	611
#ifdef και #ifndef	613
#undef	614
Χρήση του τελεστή defined	615
Ο φθίνων ρόλος του προεπεξεργαστή	615
#line	616
#pragma	617
Οι τελεστές προεπεξεργαστή # και ##	617
Προκαθορισμένα ονόματα μακροεντολών	618
Τελικές σκέψεις	619

A ■■■ Το σύστημα εισόδου/εξόδου της C	621
Το σύστημα εισόδου/εξόδου της C	623
Κατανόηση των συναρτήσεων printf() και scanf()	623
printf()	623
scanf()	626
Το σύστημα αρχείων της C	630
fopen()	631
fputc()	632
fgetc()	633
feof()	633
fclose()	633
Χρήση των συναρτήσεων fopen(), fgetc(), fputc(), και fclose()	634
ferror() και rewind()	635
fread() και fwrite()	635
Η συνάρτηση fseek() — είσοδος/έξοδος τυχαίας προσπέλασης	637
Οι συναρτήσεις fprintf() και fscanf()	638
Διαγραφή αρχείων	638
B ■■■ Χρήση παλαιότερου μεταγλωττιστή C++	639
Δύο απλές αλλαγές	641
C ■■■ Επεκτάσεις .NET της C++	643
Οι επεκτάσεις δεσμευμένων λέξεων .NET	644
__abstract	645
__box	645
__delegate	645
__event	645
__finally	645
__gc	645
__identifier	645
__interface	645
__nogc	646
__pin	646
__property	646
__sealed	646
__try_cast	646
__typeof	646
__value	646

<i>Περιεχόμενα</i>	19
Επεκτάσεις προεπεξεργαστή.....	646
Το χαρακτηριστικό attribute	647
Μεταγλώττιση διαχειριζόμενου κώδικα C++	647
■ Ευρετήριο	649



ΚΕΦΑΛΑΙΟ 7

**Συναρτήσεις, μέρος
πρώτο: βασικές έννοιες**

Σε αυτό το κεφάλαιο ξεκινά μια σε βάθος εξέταση των συναρτήσεων. Οι συναρτήσεις αποτελούν τα δομικά υλικά της C++ και η καλή κατανόησή τους είναι θεμελιώδης αν θέλετε να γίνετε επιτυχημένος προγραμματιστής της C++. Εκτός από τη σύντομη παρουσίασή τους στο Κεφάλαιο 2, μέχρι τώρα έχετε χρησιμοποιήσει συναρτήσεις λίγο-πολύ διαισθητικά. Σε αυτό το κεφάλαιο θα τις μελετήσετε σε βάθος. Στα θέματα που θα εξετάσουμε περιλαμβάνονται οι κανόνες εμβέλειας των συναρτήσεων, οι αναδρομικές συναρτήσεις, μερικές ειδικές ιδιότητες της συνάρτησης `main()`, η εντολή `return`, και τα πρωτότυπα συναρτήσεων.

Κανόνες εμβέλειας συναρτήσεων

Οι κανόνες εμβέλειας καθορίζουν τον τρόπο με τον οποίο μπορεί να προσπελαστεί ένα αντικείμενο και τη διάρκεια ζωής του.

Οι κανόνες εμβέλειας (scope rules) μιας γλώσσας είναι οι κανόνες που ορίζουν τον τρόπο με τον οποίο μπορεί να προσπελαστεί ένα αντικείμενο από τα διάφορα τμήματα ενός προγράμματός σας. Με άλλα λόγια, οι κανόνες εμβέλειας καθορίζουν ποιος κώδικας έχει πρόσβαση σε μια μεταβλητή. Επίσης, οι κανόνες εμβέλειας καθορίζουν και τη διάρκεια ζωής μιας μεταβλητής. Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο, υπάρχουν τρία είδη μεταβλητών: οι τοπικές μεταβλητές (local variables), οι τυπικές παράμετροι (formal parameters), και οι καθολικές μεταβλητές (global variables). Αυτή τη φορά θα εξετάσουμε τους κανόνες εμβέλειας πολύ πιο προσεκτικά, δίνοντας έμφαση στη σχέση τους με τις συναρτήσεις.

Τοπικές μεταβλητές

Όπως γνωρίζετε, οι μεταβλητές που δηλώνονται στο εσωτερικό μιας συνάρτησης ονομάζονται *τοπικές μεταβλητές* (local variables). Ωστόσο, η C++ υποστηρίζει μια πολύ πιο "εξειδικευμένη" έννοια τοπικής μεταβλητής, την οποία δεν έχετε συναντήσει μέχρι τώρα. Στη C++, οι μεταβλητές μπορούν να είναι τοπικές για ένα μπλοκ κώδικα. Δηλαδή, μια μεταβλητή μπορεί να δηλωθεί στο εσωτερικό ενός μπλοκ κώδικα και είναι τοπική ως προς αυτό. (Θυμηθείτε ότι ένα μπλοκ αρχίζει με ένα αριστερό άγκιστρο και τελειώνει με ένα δεξιό άγκιστρο.) Στην πραγματικότητα, οι τοπικές μεταβλητές μιας συνάρτησης δεν είναι παρά μια ειδική περίπτωση της γενικότερης έννοιας των τοπικών μεταβλητών.

Μια τοπική μεταβλητή μπορεί να χρησιμοποιηθεί μόνο από τις εντολές που βρίσκονται στο εσωτερικό του μπλοκ στο οποίο έχει δηλωθεί. Με άλλα λόγια, μια τοπική μεταβλητή δεν είναι γνωστή έξω από το δικό της μπλοκ κώδικα. Επομένως, οι εντολές που βρίσκονται έξω από το μπλοκ δεν μπορούν να προσπελάσουν ένα αντικείμενο που έχει οριστεί στο εσωτερικό του μπλοκ.

Ένα από τα πλέον σημαντικά πράγματα που πρέπει να καταλάβετε για τις τοπικές μεταβλητές είναι ότι υπάρχουν μόνο όσο εκτελείται το μπλοκ κώδικα στο οποίο έχουν δηλωθεί. Αυτό σημαίνει ότι μια τοπική μεταβλητή δημιουργείται κατά την έναρξη εκτέλεσης του μπλοκ στο οποίο ανήκει και καταστρέφεται κατά την έξοδο από αυτό. Όπως είναι προφανές, επειδή ακριβώς η τοπική μεταβλητή καταστρέφεται κατά την έξοδο από το μπλοκ στο οποίο ανήκει, η τιμή της χάνεται.

Το πιο συνηθισμένο μπλοκ κώδικα είναι η συνάρτηση. Στη C++, κάθε συνάρτηση ορίζει ένα μπλοκ κώδικα το οποίο αρχίζει με το αριστερό άγκιστρο της συνάρτησης και τελειώνει με το δεξιό της άγκιστρο. Ο κώδικας και τα δεδομένα μιας συνάρτησης είναι ιδιωτικά (private) για τη συγκεκριμένη συνάρτηση, και δεν είναι δυνατό να προσπελαστούν από καμία εντολή άλλης συνάρτησης, εκτός και αν γίνει κλήση της συνάρτησης. (Δεν μπορείτε, για παράδειγμα, να χρησιμοποιήσετε μια εντολή `goto` για να μεταπηδήσετε στο εσωτερικό μια άλλης συνάρτησης.) Το

"σώμα" μιας συνάρτησης είναι κρυμμένο από το υπόλοιπο πρόγραμμα και δεν μπορεί ούτε να επηρεάσει ούτε να επηρεαστεί από άλλα τμήματα του προγράμματος, εκτός και αν χρησιμοποιεί καθολικές μεταβλητές. Έτσι, τα περιεχόμενα μιας συνάρτησης είναι εντελώς ξεχωριστά από τα περιεχόμενα μιας άλλης. Για να το θέσουμε διαφορετικά, ο κώδικας και τα δεδομένα που έχουν οριστεί στο εσωτερικό μιας συνάρτησης δεν μπορούν να αλληλεπιδράσουν με τον κώδικα ή τα δεδομένα μιας άλλης συνάρτησης γιατί οι δύο συναρτήσεις έχουν διαφορετική εμβέλεια.

Επειδή κάθε συνάρτηση ορίζει τη δική της εμβέλεια, οι μεταβλητές που έχουν δηλωθεί στο εσωτερικό μιας συνάρτησης δεν έχουν καμία επίδραση σε μεταβλητές που έχουν δηλωθεί σε μια άλλη συνάρτηση — ακόμη και αν αυτές οι μεταβλητές έχουν ίδια ονόματα. Για παράδειγμα, δείτε το παρακάτω πρόγραμμα:

```
#include <iostream>
using namespace std;

void f1();

int main()
{
    char str[]="this is str in main()";

    cout << str << '\n';
    f1();
    cout << str << '\n';

    return 0;
}

void f1()
{
    char str[80];

    cout << "Enter something: ";
    cin >> str;
    cout << str << '\n';
}
```

Σε αυτό το πρόγραμμα, ένας πίνακας χαρακτήρων με το όνομα **str** δηλώνεται δύο φορές, μία στη συνάρτηση **main()** και μία στην **f1()**. Ωστόσο, ο πίνακας **str** της συνάρτησης **main()** δεν έχει καμία σχέση με αυτόν της **f1()**. Ο λόγος γι' αυτό είναι ότι κάθε πίνακας **str** είναι γνωστός μόνο στο μπλοκ κώδικα στο οποίο έχει δηλωθεί. Για να το επιβεβαιώσετε αυτό, δοκιμάστε να εκτελέσετε το πρόγραμμα. Όπως θα δείτε, παρόλο που ο πίνακας **str** δέχεται στο εσωτερικό της **f1()** ένα αλφαριθμητικό το οποίο καταχωρίζει ο χρήστης από το πληκτρολόγιο, τα περιεχόμενα του πίνακα **str** της συνάρτησης **main()** παραμένουν αναλλοίωτα.

Η γλώσσα C++ περιλαμβάνει τη δεσμευμένη λέξη **auto** που μπορεί να χρησιμοποιηθεί για τη δήλωση τοπικών μεταβλητών. Επειδή, όμως, όλες οι μη καθολικές μεταβλητές είναι εξ ορισμού τύπου **auto**, αυτή η δεσμευμένη λέξη δεν χρησιμοποιείται ουσιαστικά ποτέ. Επομένως, δεν θα τη δείτε ούτε και στα παραδείγματα του βιβλίου. Αν επιλέξετε, πάντως, να τη χρησιμοποιήσετε, θα πρέπει να την τοποθετήσετε πριν από τον τύπο της μεταβλητής, όπως φαίνεται εδώ:

```
auto char ch;
```

Είναι συνήθης πρακτική η δήλωση όλων των μεταβλητών που απαιτούνται σε μια συνάρτηση να γίνεται στην αρχή του μπλοκ κώδικα της συνάρτησης. Ο κύριος λόγος γι' αυτό είναι ώστε να μπορεί όποιος διαβάσει τον κώδικα να προσδιορίσει εύκολα ποιες μεταβλητές χρησιμοποιούνται. Ωστόσο, η αρχή του μπλοκ μιας συνάρτησης δεν είναι και η μοναδική θέση όπου μπορεί να γίνει δήλωση τοπικών μεταβλητών. Μια τοπική μεταβλητή μπορεί να δηλωθεί οπουδήποτε στο εσωτερικό ενός μπλοκ κώδικα. Μια μεταβλητή που δηλώνεται στο εσωτερικό ενός μπλοκ είναι τοπική για το συγκεκριμένο μπλοκ. Αυτό σημαίνει ότι η μεταβλητή δεν υπάρχει μέχρι να αρχίσει η εκτέλεση του μπλοκ και καταστρέφεται κατά την έξοδο από αυτό. Επιπλέον, κανένα τμήμα κώδικα έξω από αυτό το μπλοκ — συμπεριλαμβανομένου και άλλου κώδικα της συνάρτησης — δεν μπορεί να προσπελάσει τη συγκεκριμένη μεταβλητή.

Για να το καταλάβετε, δοκιμάστε το παρακάτω πρόγραμμα:

```
/* Αυτό το πρόγραμμα δείχνει με ποιον τρόπο μπορούν
   οι μεταβλητές να είναι τοπικές σε ένα μπλοκ.
*/

#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    int choice;

    cout << "(1) add numbers or ";
    cout << "(2) concatenate strings?: ";

    cin >> choice;
    if(choice == 1) {
        int a, b; /* ενεργοποίηση δύο ακέραιων μεταβλητών*/
        cout << "Enter two numbers: ";
        cin >> a >> b;
        cout << "Sum is " << a+b << '\n';
    }
    else {
        char s1[80], s2[80]; /* ενεργοποίηση δύο αλφαριθμητικών */
        cout << "Enter two strings: ";
        cin >> s1;
        cin >> s2;
        strcat(s1, s2);
        cout << "Concatenation is " << s1 << '\n';
    }

    return 0;
}
```

Αυτό το πρόγραμμα είτε προσθέτει δύο αριθμούς είτε συνενώνει δύο αλφαριθμητικά, ανάλογα με την επιλογή του χρήστη. Παρατηρήστε τις δηλώσεις των μεταβλητών **a** και **b** στο μπλοκ της **if** και τις δηλώσεις των **s1** και **s2** στο μπλοκ της **else**. Αυτές οι μεταβλητές θα αρχίσουν να υπάρ-

χουν μόνο όταν αρχίσει η εκτέλεση των αντίστοιχων μπλοκ κώδικα, και θα πάνε να υπάρχουν κατά την έξοδο από αυτά. Αν ο χρήστης επιλέξει να προσθέσει αριθμούς, θα δημιουργηθούν οι μεταβλητές **a** και **b**. Αν ο χρήστης επιλέξει να συνενώσει αλφαριθμητικά, θα δημιουργηθούν οι μεταβλητές **s1** και **s2**. Επίσης, δεν μπορεί να γίνει καμία αναφορά σε αυτές τις μεταβλητές έξω από τα δικά τους μπλοκ κώδικα — ούτε και από άλλα μέρη της ίδιας συνάρτησης. Για παράδειγμα, αν επιχειρήσετε να μεταγλωττίσετε την παρακάτω (λανθασμένη) έκδοση αυτού του προγράμματος, θα δείτε ένα μήνυμα σφάλματος:

```
/* Αυτό το πρόγραμμα είναι λανθασμένο. */

#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    int choice;

    cout << "(1) add numbers or ";
    cout << "(2) concatenate strings?: ";

    cin >> choice;
    if(choice == 1) {
        int a, b; /* ενεργοποίηση δύο ακέραιων μεταβλητών*/
        cout << "Enter two numbers: ";
        cin >> a >> b;
        cout << "Sum is " << a+b << '\n';
    }
    else {
        char s1[80], s2[80]; /* ενεργοποίηση δύο αλφαριθμητικών */
        cout << "Enter two strings: ";
        cin >> s1;
        cin >> s2;
        strcat(s1, s2);
        cout << "Concatenation is " << s1 << '\n';
    }

    a = 10; // *** Σφάλμα *** -- η a δεν είναι γνωστή εδώ!

    return 0;
}
```

Στην προκειμένη περίπτωση, η μεταβλητή **a** δεν είναι γνωστή έξω από το μπλοκ **else**. Επομένως, είναι λάθος να προσπαθήσετε να τη χρησιμοποιήσετε.

Όταν μια τοπική μεταβλητή που έχει δηλωθεί σε ένα "εσωτερικό" μπλοκ έχει το ίδιο όνομα με μια άλλη μεταβλητή που έχει δηλωθεί στο "εξωτερικό" μπλοκ, τότε, στην περιοχή εμβέλειας του εσωτερικού μπλοκ, η μεταβλητή του εσωτερικού μπλοκ υπερισχύει της μεταβλητής του εξωτερικού μπλοκ. Για παράδειγμα:

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i, j;

    i = 10;
    j = 100;

    if(j > 0) {
        int i; // αυτή η i είναι διαφορετική από την εξωτερική i

        i = j / 2;
        cout << "inner i: " << i << '\n';
    }

    cout << "outer i: " << i << '\n';

    return 0;
}
```

Η έξοδος αυτού του προγράμματος είναι η εξής:

```
inner i: 50
outer i: 10
```

Η μεταβλητή **i** που δηλώθηκε στο εσωτερικό του μπλοκ **if** υποσκελίζει ("κρύβει") την εξωτερική **i**. Τυχόν αλλαγές που θα γίνουν στην εσωτερική **i** δεν θα έχουν καμία επίδραση στην εξωτερική **i**. Από την άλλη μεριά, έξω από το μπλοκ **if** η εσωτερική **i** είναι άγνωστη, οπότε επανέρχεται σε ισχύ η εξωτερική **i**.

Μια τοπική μεταβλητή δεν διατηρεί την τιμή της μεταξύ δύο διαδοχικών ενεργειών.

Επειδή οι τοπικές μεταβλητές δημιουργούνται και καταστρέφονται κατά την είσοδο και την έξοδο στα και από τα μπλοκ στα οποία έχουν δηλωθεί, μια τοπική μεταβλητή δεν διατηρεί την τιμή της μεταξύ δύο διαδοχικών ενεργοποιήσεων του μπλοκ στο οποίο έχει δηλωθεί. Αυτό είναι ιδιαίτερα σημαντικό και πρέπει να το θυμάστε όταν έχετε να κάνετε με κλήσεις συναρτήσεων. Όταν καλείται μια συνάρτηση, δημιουργούνται οι τοπικές μεταβλητές της· κατά την επιστροφή της, καταστρέφονται. Αυτό σημαίνει ότι οι τοπικές μεταβλητές δεν μπορούν να διατηρήσουν τις τιμές τους μεταξύ των διαδοχικών κλήσεων της συνάρτησης. (Υπάρχει, πάντως, ένας τρόπος με τον οποίο παρακάμπτεται αυτός ο περιορισμός — θα περιγραφεί αργότερα.)

Αν δεν ορίζεται διαφορετικά, η αποθήκευση των τοπικών μεταβλητών γίνεται στη στοίβα (stack). Το γεγονός ότι η στοίβα είναι δυναμική — κάτι που σημαίνει ότι "μετακινείται" από τη μια περιοχή της μνήμης στην άλλη — εξηγεί και το γιατί οι τοπικές μεταβλητές δεν μπορούν, τουλάχιστον σε γενικές γραμμές, να διατηρήσουν την τιμή τους μεταξύ διαδοχικών κλήσεων της συνάρτησης στην οποία έχουν δηλωθεί.

Όπως αναφέρθηκε νωρίτερα, αν και οι τοπικές μεταβλητές δηλώνονται συνήθως στην αρχή του μπλοκ στο οποίο ανήκουν, αυτό δεν είναι υποχρεωτικό. Μια τοπική μεταβλητή μπορεί να δηλωθεί οπουδήποτε στο εσωτερικό ενός μπλοκ, αρκεί να γίνει αυτό πριν χρησιμοποιηθεί για πρώτη φορά. Για παράδειγμα, το παρακάτω πρόγραμμα είναι απολύτως έγκυρο:

```
#include <iostream>
using namespace std;
```

```
int main()
{
    cout << "Enter a number: ";
    int a; // δήλωση μεταβλητής
    cin >> a;

    cout << "Enter a second number: ";
    int b; // δήλωση άλλης μεταβλητής
    cin >> b;

    cout << "Product: " << a*b << '\n';

    return 0;
}
```

Σε αυτό το παράδειγμα, οι μεταβλητές **a** και **b** δηλώνονται ακριβώς τη στιγμή που χρειάζονται. Στην πράξη, οι περισσότεροι προγραμματιστές δηλώνουν όλες τις τοπικές μεταβλητές στην αρχή του μπλοκ στο οποίο τις χρησιμοποιούν, αλλά αυτό είναι θέμα προσωπικού στυλ.

Δήλωση μεταβλητών στο εσωτερικό εντολών επανάληψης και επιλογής

Μπορείτε να δηλώσετε μια μεταβλητή στο τμήμα απόδοσης αρχικών τιμών ενός βρόχου **for** ή στην παράσταση υπό συνθήκη μιας εντολής **if**, **switch**, ή **while**. Η εμφάνιση μιας μεταβλητής που θα δηλωθεί σε κάποια από αυτές τις θέσεις περιορίζεται στο μπλοκ κώδικα που ελέγχεται από τη συγκεκριμένη εντολή.

Έτσι, μια μεταβλητή που δηλώνεται σε μια εντολή **for** θα είναι τοπική στο βρόχο, όπως φαίνεται και στο πρόγραμμα που ακολουθεί:

```
#include <iostream>
using namespace std;

int main()
{
    // Η i είναι τοπική στο βρόχο for
    for(int i = 0; i<10; i++) {
        cout << i << " ";
        cout << "squared is " << i * i << "\n";
    }

    // i = 10; // *** Λάθος *** -- Η i δεν είναι γνωστή εδώ!

    return 0;
}
```

Εδώ, η **i** δηλώνεται στο τμήμα απόδοσης αρχικών τιμών της εντολής **for** και χρησιμοποιείται για τον έλεγχο του βρόχου. Έξω από το βρόχο, η **i** είναι άγνωστη.

Γενικά, όταν η μεταβλητή ελέγχου του βρόχου **for** δεν χρειάζεται έξω από το βρόχο, καλό είναι να γίνεται η δήλωσή της στο εσωτερικό της εντολής **for**, όπως στο παράδειγμα, γιατί αυτό κάνει τη μεταβλητή τοπική για το βρόχο και δεν επιτρέπει τη λανθασμένη χρήση της κάπου αλλού. Σε επαγγελματικά γραμμένο κώδικα C++, θα δείτε πολύ συχνά τη δήλωση της μεταβλητής ελέγχου

του βρόχου να γίνεται στο εσωτερικό της εντολής **for**. Φυσικά, αν η μεταβλητή είναι αναγκαία και στον κώδικα που βρίσκεται έξω από το βρόχο, δεν μπορεί να δηλωθεί στην εντολή **for**.



ΣΥΜΒΟΥΛΗ: Το αν μια μεταβλητή που δηλώνεται στο τμήμα απόδοσης αρχικών τιμών ενός βρόχου **for** θα είναι τοπική στο βρόχο έχει αλλάξει με την πάροδο του χρόνου. Αρχικά, η μεταβλητή ήταν διαθέσιμη και μετά τη **for**. Όμως, η Standard C++ περιορίζει τη μεταβλητή στην εμβέλεια του βρόχου **for**. Παρόλα αυτά, οι διάφοροι μεταγλωττιστές εξακολουθούν να αντιμετωπίζουν διαφορετικά αυτό το ζήτημα.

Αν ο μεταγλωττιστής σας συμμορφώνεται πλήρως με τις προδιαγραφές της Standard C++, τότε μπορείτε να δηλώσετε μια μεταβλητή και στο εσωτερικό των παραστάσεων υπό συνθήκη των εντολών **if**, **switch**, και **while**. Για παράδειγμα, στο παρακάτω απόσπασμα κώδικα

```
if(int x = 20) {
    cout << "This is x: ";
    cout << x;
}
```

δηλώνεται η μεταβλητή **x** και της αποδίδεται η τιμή 20. Επειδή αυτή η τιμή είναι αληθής, οι εντολές **cout** θα εκτελεστούν. Η εμβέλεια των μεταβλητών που δηλώνονται σε μια παράσταση υπό συνθήκη περιορίζεται στο μπλοκ κώδικα που ελέγχεται από την εντολή. Έτσι, σε αυτή την περίπτωση, η **x** δεν είναι γνωστή έξω από την εντολή **if**. Για να είμαι ειλικρινής, δεν θεωρούν όλοι οι προγραμματιστές ότι η δήλωση μεταβλητών σε παραστάσεις υπό συνθήκη είναι καλή πρακτική — γι' αυτό και δεν θα χρησιμοποιηθεί στα παραδείγματα του βιβλίου.

Τυπικές παράμετροι

Όπως γνωρίζετε, αν μια συνάρτηση δέχεται ορίσματα, πρέπει σε αυτή τη συνάρτηση να γίνει δήλωση των μεταβλητών που θα "υποδεχτούν" τις τιμές αυτών των ορισμάτων. Αυτές οι μεταβλητές ονομάζονται *τυπικές παράμετροι* (formal parameters) της συνάρτησης. Αν εξαιρέσουμε τη λήψη των ορισμάτων κατά την κλήση της συνάρτησης, οι τυπικές παράμετροι συμπεριφέρονται όπως και οποιαδήποτε άλλη τοπική μεταβλητή που έχει δηλωθεί στο εσωτερικό της συνάρτησης. Η εμβέλεια μιας παραμέτρου είναι τοπική στη συνάρτησή της.

Θα πρέπει να εξασφαλίζετε ότι οι τυπικές παράμετροι που δηλώνετε είναι του ίδιου τύπου με τα ορίσματα που θα μεταβιβάσετε στη συνάρτηση. Επίσης, παρά το γεγονός ότι αυτές οι μεταβλητές εκτελούν την ειδική εργασία της λήψης των τιμών των ορισμάτων, μπορούν να χρησιμοποιηθούν όπως και κάθε άλλη τοπική μεταβλητή. Για παράδειγμα, στο εσωτερικό της συνάρτησης μπορείτε να εκχωρήσετε νέα τιμή σε μια παράμετρο.

Καθολικές μεταβλητές

Οι καθολικές μεταβλητές (global variables) είναι, από πολλές πλευρές, το αντίθετο των τοπικών μεταβλητών. Είναι γνωστές σε ολόκληρο το πρόγραμμα, μπορούν να χρησιμοποιηθούν από οποιοδήποτε μπλοκ κώδικα, και διατηρούν την τιμή τους σε όλη τη διάρκεια της εκτέλεσης του προγράμματος. Επομένως, η εμβέλειά τους εκτείνεται σε ολόκληρο το πρόγραμμα. Για να δημιουργήσετε καθολικές μεταβλητές αρκεί να τις δηλώσετε έξω από οποιαδήποτε συνάρτηση.

Επειδή ακριβώς είναι καθολικές, μπορούν να προσπελαστούν από οποιαδήποτε παράσταση, ανεξάρτητα από τη συνάρτηση στην οποία βρίσκεται η παράσταση.

Όταν μια καθολική και μια τοπική μεταβλητή έχουν το ίδιο όνομα, η τοπική μεταβλητή έχει προτεραιότητα. Για να το θέσουμε διαφορετικά, μια τοπική μεταβλητή "σκεπάζει" μια καθολική μεταβλητή με το ίδιο όνομα. Έτσι, αν και οι καθολικές μεταβλητές μπορούν να προσπελαστούν από οποιοδήποτε τμήμα κώδικα του προγράμματός σας, αυτό θα συμβεί μόνο αν δεν υπάρχει τοπική μεταβλητή με το ίδιο όνομα με την καθολική μεταβλητή.

Η χρήση των καθολικών μεταβλητών επιδεικνύεται στο πρόγραμμα που ακολουθεί. Όπως μπορείτε να δείτε, οι μεταβλητές **count** και **num_right** δηλώνονται έξω από όλες τις συναρτήσεις· επομένως, είναι καθολικές μεταβλητές. Η συνήθης πρακτική καθορίζει ότι είναι καλύτερο να δηλώνονται οι καθολικές μεταβλητές κοντά στην αρχή του προγράμματος. Ωστόσο, από τεχνικής άποψης, αρκεί απλώς να δηλωθούν πριν από την πρώτη τους χρήση. Το πρόγραμμα που ακολουθεί είναι μια απλή άσκηση πρόσθεσης. Πρώτα σας ρωτά πόσα προβλήματα θέλετε να λύσετε και, μετά, για κάθε πρόβλημα, καλεί τη συνάρτηση **drill()** η οποία παράγει δύο τυχαίους αριθμούς μεταξύ του 0 και του 99. Στη συνέχεια, το πρόγραμμα ζητά, και κατόπιν ελέγχει, την απάντησή σας. Έχετε τρεις προσπάθειες για κάθε πρόβλημα. Στο τέλος, το πρόγραμμα εμφανίζει το πλήθος των σωστών απαντήσεων που δώσατε. Προσέξτε ιδιαίτερα τις καθολικές μεταβλητές που χρησιμοποιούνται στο πρόγραμμα:

// Ένα απλό πρόγραμμα άσκησης πρόσθεσης.

```
#include <iostream>
#include <cstdlib>
using namespace std;

void drill();

int count; // οι count και num_right είναι καθολικές
int num_right;

int main()
{
    cout << "How many practice problems: ";
    cin >> count;
    num_right = 0;
    do {
        drill();
        count--;
    } while(count);
    cout << "You got " << num_right << " right.\n";

    return 0;
}

void drill()
{
    int count; /* Αυτή η μεταβλητή count είναι τοπική και
                δεν έχει καμία σχέση με την καθολική.
                */
```

```

int a, b, ans;

// Παραγωγή δύο αριθμών μεταξύ 0 και 99.
a = rand() % 100;
b = rand() % 100;

// Ο χρήστης έχει 3 προσπάθειες να βρει τη σωστή απάντηση.
for(count=0; count<3; count++) {
    cout << "What is " << a << " + " << b << "? ";
    cin >> ans;
    if(ans==a+b) {
        cout << "Right\n";
        num_right++;
        return;
    }
}
cout << "You've used up all your tries.\n";
cout << "The answer is " << a+b << '\n';
}

```

Αν εξετάσετε προσεκτικά αυτό το πρόγραμμα, θα πρέπει να σας είναι ξεκάθαρο ότι και η **main()** και η **drill()** προσπελάζουν την καθολική μεταβλητή **num_right**. Η περίπτωση της **count**, όμως, είναι λίγο πιο περίπλοκη. Η αναφορά στην **count** που γίνεται στη συνάρτηση **main()** είναι για την καθολική μεταβλητή **count**. Ωστόσο, στη συνάρτηση **drill()** έχει δηλωθεί μια τοπική μεταβλητή με το όνομα **count**. Όταν η **drill()** χρησιμοποιεί την **count**, αναφέρεται στην τοπική της μεταβλητή και όχι στην καθολική. Θυμηθείτε ότι, αν στο εσωτερικό μιας συνάρτησης μια καθολική και μια τοπική μεταβλητή έχουν το ίδιο όνομα, όλες οι χρήσεις της μεταβλητής θα αναφέρονται στην τοπική μεταβλητή και όχι στην καθολική.

Η αποθήκευση των καθολικών μεταβλητών γίνεται σε μια σταθερή περιοχή της μνήμης, η οποία δεσμεύεται γι' αυτόν το σκοπό από το πρόγραμμά σας. Οι καθολικές μεταβλητές είναι χρήσιμες στις περιπτώσεις που τα ίδια δεδομένα χρησιμοποιούνται από πολλές συναρτήσεις του προγράμματός σας ή όταν μια μεταβλητή πρέπει να διατηρεί την τιμή της σε όλη τη διάρκεια της εκτέλεσης του προγράμματος. Ωστόσο, θα πρέπει να αποφεύγετε την άσκοπη χρήση καθολικών μεταβλητών για τρεις κυρίως λόγους:

- ◆ Καταλαμβάνουν μνήμη σε όλη τη διάρκεια εκτέλεσης του προγράμματός σας και όχι μόνο όταν είναι απαραίτητες.
- ◆ Η χρήση μιας καθολικής μεταβλητής εκεί που θα αρκούσε μια τοπική κάνει τη συνάρτηση λιγότερο "γενική", επειδή βασίζεται σε κάτι το οποίο πρέπει να οριστεί έξω από αυτή.
- ◆ Η χρήση πολλών καθολικών μεταβλητών μπορεί να οδηγήσει σε σφάλματα εξαιτίας άγνωστων και ανεπιθύμητων παρενεργειών. Ένα σημαντικό πρόβλημα κατά την ανάπτυξη μεγάλων προγραμμάτων είναι η ακούσια τροποποίηση της τιμής μιας μεταβλητής λόγω της χρήσης της σε κάποιο άλλο σημείο του προγράμματος. Αυτό μπορεί να συμβεί στη C++ όταν χρησιμοποιείτε πολλές καθολικές μεταβλητές στα προγράμματά σας.

Μεταβίβαση δεικτών και πινάκων

Μέχρι τώρα, στα παραδείγματα του βιβλίου μεταβιβάσαμε σε συναρτήσεις απλές μεταβλητές. Ωστόσο, θα υπάρξουν περιπτώσεις που θα χρειάζεται να χρησιμοποιήσετε ως ορίσματα δείκτες και πίνακες. Αν και η μεταβίβαση αυτού του είδους των ορισμάτων είναι μια διαδικασία σχετικά ξεκάθαρη, θα πρέπει να εξετάσουμε ορισμένα ειδικά ζητήματα.

Κλήση συναρτήσεων με δείκτες

Η C++ σάς επιτρέπει να μεταβιβάσετε ένα δείκτη σε μια συνάρτηση. Για να το κάνετε, αρκεί να δηλώσετε την παράμετρο ως τύπου δείκτη. Ακολουθεί ένα παράδειγμα:

```
// Μεταβίβαση δείκτη σε συνάρτηση.
#include <iostream>
using namespace std;

void f(int *j);

int main()
{
    int i;
    int *p;

    p = &i; // Η μεταβλητή p δείχνει τώρα στην i

    f(p);

    cout << i; // η i έχει τώρα την τιμή 100

    return 0;
}

void f(int *j)
{
    *j = 100; // στη μεταβλητή στην οποία δείχνει η j εκχωρείται η τιμή 100
}
```

Μελετήστε αυτό το πρόγραμμα προσεκτικά. Όπως μπορείτε να δείτε, η **f()** δέχεται μία παράμετρο: ένα δείκτη σε ακέραιο. Στο εσωτερικό της **main()**, στο δείκτη **p** εκχωρείται η διεύθυνση της **i**. Κατόπιν, καλείται η **f()** με όρισμα την **p**. Όταν η παράμετρος **j**, τύπου δείκτη, δεχτεί την τιμή του δείκτη **p**, θα δείχνει και αυτή στη μεταβλητή **i** της **main()**. Έτσι, η εντολή ανάθεσης

```
*j = 100;
```

θα έχει ως αποτέλεσμα να πάρει η **i** την τιμή 100, οπότε και το πρόγραμμα θα εμφανίσει στην οθόνη **100**. Για να γενικεύσουμε, η **f()** εκχωρεί την τιμή 100 στην όποια διεύθυνση που θα της μεταβιβαστεί.

Στο προηγούμενο παράδειγμα, δεν είναι πραγματικά αναγκαία η χρήση της μεταβλητής δείκτη **p**. Κατά την κλήση της **f()**, θα μπορούσαμε απλώς να χρησιμοποιήσουμε τον τελεστή **&** πριν από

τη μεταβλητή **i** (κάτι που θα δημιουργούσε τη διεύθυνση της **i**). Το αναθεωρημένο πρόγραμμα φαίνεται στη συνέχεια:

```
// Μεταβίβαση δείκτη σε συνάρτηση -- αναθεωρημένη έκδοση.
#include <iostream>
using namespace std;

void f(int *j);

int main()
{
    int i;

    f(&i);

    cout << i;

    return 0;
}

void f(int *j)
{
    *j = 100; // στη μεταβλητή στην οποία δείχνει η j εκχωρείται η τιμή 100
}
```

Είναι αποφασιστικής σημασίας να καταλάβετε ένα πολύ σημαντικό πράγμα σχετικά με τη μεταβίβαση δεικτών σε συναρτήσεις: Όταν εκτελείτε στο εσωτερικό της συνάρτησης οποιαδήποτε πράξη στην οποία χρησιμοποιείται ο δείκτης, στην πραγματικότητα χρησιμοποιείτε τη μεταβλητή στην οποία δείχνει ο δείκτης. Έτσι, η συνάρτηση μπορεί να αλλάξει την τιμή του αντικειμένου στο οποίο δείχνει η παράμετρος.

Κλήση συναρτήσεων με πίνακες

Όταν χρησιμοποιείται κάποιος πίνακας ως όρισμα σε μια συνάρτηση, στη συνάρτηση μεταβιβάζεται μόνο η διεύθυνση του πρώτου του στοιχείου και όχι ένα αντίγραφο ολόκληρου του πίνακα. (Θυμηθείτε ότι, στη C++, το όνομα ενός πίνακα χωρίς αριθμοδείκτη είναι ένας δείκτης προς το πρώτο στοιχείο του πίνακα.) Αυτό σημαίνει ότι η δήλωση της παραμέτρου θα πρέπει να είναι συμβατού τύπου. Υπάρχουν τρεις τρόποι για να δηλώσετε μια παράμετρο που πρόκειται να δεχτεί ένα δείκτη σε πίνακα. Πρώτον, μπορεί να δηλωθεί ως πίνακας ίδιου τύπου και μεγέθους με τον πίνακα που θα χρησιμοποιηθεί κατά την κλήση της συνάρτησης, όπως φαίνεται εδώ:

```
#include <iostream>
using namespace std;

void display(int num[10]);

int main()
{
    int t[10], i;

    for(i=0; i<10; ++i) t[i]=i;
```

```

    display(t); // μεταβίβαση του πίνακα t στη συνάρτηση

    return 0;
}

// Εμφάνιση ορισμένων αριθμών στην οθόνη.
void display(int num[10])
{
    int i;

    for(i=0; i<10; i++) cout << num[i] << ' ';
}

```

Παρά το γεγονός ότι η παράμετρος **num** έχει δηλωθεί ως πίνακας ακεραίων με 10 στοιχεία, ο μεταγλωττιστής της C++ θα τη μετατρέψει αυτόματα σε δείκτη προς ακέραιο. Αυτό είναι αναγκαίο γιατί, στην πραγματικότητα, καμία παράμετρος δεν μπορεί να δεχτεί έναν ολόκληρο πίνακα. Επειδή θα μεταβιβαστεί μόνο ένας δείκτης στον πίνακα, θα πρέπει να υπάρχει μια παράμετρος-δείκτης για να "υποδεχτεί" την τιμή.

Ο δεύτερος τρόπος για να δηλώσετε μια παράμετρο πίνακα, είναι να την ορίσετε ως πίνακα απροσδιόριστου μεγέθους, όπως στη συνέχεια:

```

void display(int num[])
{
    int i;

    for(i=0; i<10; i++) cout << num[i] << ' ';
}

```

Εδώ, η **num** έχει δηλωθεί ως πίνακας ακεραίων αγνώστου μεγέθους. Επειδή η C++ δεν κάνει έλεγχο των ορίων ενός πίνακα, το πραγματικό μέγεθος του πίνακα δεν ενδιαφέρει την παράμετρο (αλλά ενδιαφέρει, φυσικά, το πρόγραμμα). Και σε αυτή τη μέθοδο δήλωσης γίνεται αυτόματη μετατροπή του πίνακα σε δείκτη από το μεταγλωττιστή.

Ο τελευταίος τρόπος με τον οποίο μπορείτε να δηλώσετε τη **num** είναι ως δείκτη. Αυτή είναι και η πιο συνηθισμένη μέθοδος που θα δείτε σε επαγγελματικά προγράμματα της C++. Ακολουθεί ένα παράδειγμα:

```

void display(int *num)
{
    int i;

    for(i=0; i<10; i++) cout << num[i] << ' ';
}

```

Ο λόγος για τον οποίο είναι δυνατό να δηλωθεί η **num** ως δείκτης είναι το γεγονός ότι οποιοσδήποτε δείκτης μπορεί να δεικτοδοτηθεί με τη χρήση των αγκυλών [], σαν να ήταν πίνακας. Θα πρέπει να έχετε διαπιστώσει ότι και οι τρεις μέθοδοι δήλωσης μιας παραμέτρου πίνακα έχουν το ίδιο αποτέλεσμα: ένα δείκτη.

Από την άλλη μεριά, όταν ως όρισμα χρησιμοποιείται ένα *στοιχείο* πίνακα, αντιμετωπίζεται σαν να ήταν οποιαδήποτε απλή μεταβλητή. Για παράδειγμα, το προηγούμενο πρόγραμμα θα μπορούσε να έχει γραφεί χωρίς να μεταβιβάζεται ολόκληρος ο πίνακας, όπως φαίνεται εδώ:

```
#include <iostream>
using namespace std;

void display(int num);

int main()
{
    int t[10],i;

    for(i=0; i<10; ++i) t[i]=i;
    for(i=0; i<10; i++) display(t[i]);

    return 0;
}

// Εμφάνιση ορισμένων αριθμών στην οθόνη.
void display(int num)
{
    cout << num << ' ';
}
```

Όπως μπορείτε να δείτε, η παράμετρος της συνάρτησης **display()** είναι τύπου **int**. Δεν έχει σχέση το γεγονός ότι η **display()** καλείται με ένα στοιχείο πίνακα, επειδή μεταβιβάζεται μόνο αυτή η τιμή του πίνακα.

Είναι σημαντικό να θυμάστε ότι, όταν χρησιμοποιείται ένας πίνακας ως όρισμα συνάρτησης, στη συνάρτηση μεταβιβάζεται η διεύθυνσή του. Αυτό σημαίνει ότι ο κώδικας στο εσωτερικό της συνάρτησης θα ενεργεί — και θα μπορεί να τροποποιήσει — τα πραγματικά περιεχόμενα του πίνακα που χρησιμοποιήθηκε στην κλήση της συνάρτησης. Για παράδειγμα, δείτε τη συνάρτηση **cube()** του επόμενου προγράμματος, η οποία υψώνει την τιμή κάθε στοιχείου του πίνακα στην τρίτη δύναμη. Για να καλέσετε την **cube()**, αρκεί να της μεταβιβάσετε τη διεύθυνση του πίνακα ως πρώτο όρισμα και το μέγεθός του ως δεύτερο.

```
#include <iostream>
using namespace std;

void cube(int *n, int num);

int main()
{
    int i, nums[10];

    for(i=0; i<10; i++) nums[i] = i+1;
    cout << "Original contents: ";
    for(i=0; i<10; i++) cout << nums[i] << ' ';
    cout << '\n';

    cube(nums, 10); // υπολογισμός του κύβου

    cout << "Altered contents: ";
    for(i=0; i<10; i++) cout << nums[i] << ' ';
```

```

    return 0;
}

void cube(int *n, int num)
{
    while(num) {
        *n = *n * *n * *n;
        num--;
        n++;
    }
}

```

Να ποια είναι η έξοδος που παράγεται από το πρόγραμμα:

```

Original contents: 1 2 3 4 5 6 7 8 9 10
Altered contents: 1 8 27 64 125 216 343 512 729 1000

```

Όπως μπορείτε να δείτε, μετά την κλήση της **cube()**, τα περιεχόμενα του πίνακα **nums** στη συνάρτηση **main()** θα είναι η τρίτη δύναμη των αρχικών του τιμών. Με άλλα λόγια, οι τιμές των στοιχείων του πίνακα **nums** έχουν τροποποιηθεί από τις εντολές στο εσωτερικό της συνάρτησης **cube()**, γιατί ο δείκτης **n** δείχνει στον πίνακα **nums**.

Μεταβίβαση αλφαριθμητικών

Όπως γνωρίζετε, τα αλφαριθμητικά δεν είναι παρά πίνακες χαρακτήρων που τερματίζονται με το μηδενικό χαρακτήρα (**null**). Έτσι, όταν μεταβιβάζετε σε μια συνάρτηση ένα αλφαριθμητικό, στην πραγματικότητα μεταβιβάζεται ένας δείκτης προς την αρχή του αλφαριθμητικού. Αυτός ο δείκτης είναι τύπου **char ***. Για παράδειγμα, εξετάστε το πρόγραμμα που ακολουθεί· περιλαμβάνει τη συνάρτηση **stringupper()**, η οποία μετατρέπει τους χαρακτήρες ενός αλφαριθμητικού σε κεφαλαίους.

```

// Μεταβίβαση αλφαριθμητικού σε συνάρτηση.
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;

void stringupper(char *str);

int main()
{
    char str[80];

    strcpy(str, "this is a test");

    stringupper(str);
    cout << str; // εμφάνιση του αλφαριθμητικού με κεφαλαία
    return 0;
}

void stringupper(char *str)
{
    while(*str) {

```

Μάθετε τη **C++** από το **ΜΗΔΕΝ** 3η ΑΜΕΡΙΚΑΝΙΚΗ ΕΚΔΟΣΗ

Ο Γρήγορος Οδηγός σας για επιτυχία στη C++

Κανένας δεν διδάσκει τη C++ καλύτερα από τον Herb Schildt — γι' αυτό και τα βιβλία του τα επέλεξαν εκατομμύρια άνθρωποι σε ολόκληρο τον κόσμο. Τώρα μπορείτε και εσείς να μάθετε γρήγορα τη C++ χρησιμοποιώντας τη δοκιμασμένη του μέθοδο. Σε αυτή την πλήρως ανανεωμένη νέα έκδοση του βιβλίου, ο Herb εξηγεί τη γλώσσα C++ από τις βασικές της έννοιες μέχρι τα πιο προηγμένα χαρακτηριστικά της. Θα ξεκινήσετε με μια γενική επισκόπηση της C++ και τη γενική μορφή ενός προγράμματος C++, και μετά θα γνωρίσετε βασικές έννοιες όπως είναι οι εντολές ελέγχου, οι τελεστές, οι μεταβλητές, οι τάξεις, και τα αντικείμενα. Στη συνέχεια, θα μάθετε για τις εξαιρέσεις, τα πρότυπα, τους χώρους ονομάτων, την αναγνώριση τύπων κατά το χρόνο εκτέλεσης, και τη Standard Template Library (STL) — καθιερωμένη βιβλιοθήκη προτύπων. Θα βρείτε ακόμη και μια σύνοψη των επεκτεταμένων δεσμευμένων λέξεων που χρησιμοποιούνται για τον προγραμματισμό στο πλαίσιο εργασίας .NET — και όλα αυτά με το ξεκάθαρο, σαφές, και ασυμβίβαστο στυλ του Herb.



Ο συγγραφέας:

Ο Herbert Schildt είναι ο κορυφαίος συγγραφέας για τον προγραμματισμό παγκοσμίως. Είναι αυθεντία στις γλώσσες προγραμματισμού C, C++, Java, και C#, και ειδικός στον προγραμματισμό για τα Windows. Ήταν μέλος της επιτροπής ANSI/ISO η οποία προτυποποίησε τη C++.



ΚΛΕΙΔΑΡΙΘΜΟΣ

Σολωμού 57, 10432, ΑΘΗΝΑ, Τηλ. 210-5237635

Επισκεφθείτε μας στο Internet
<http://www.klidarithmos.gr>



Θέματα που καλύπτονται:

- Γενική μορφή των προγραμμάτων C++
- Τύποι δεδομένων, τελεστές, και παραστάσεις
- Εντολές ελέγχου
- Συναρτήσεις
- Δείκτες και αναφορές
- Τάξεις και αντικείμενα
- Υπερφόρτωση συναρτήσεων και τελεστών
- Συναρτήσεις δόμησης και αποδόμησης
- Πρότυπα
- Χειρισμός εξαιρέσεων
- Κληρονομικότητα
- Εικονικές συναρτήσεις
- Σύστημα εισόδου/εξόδου της C++
- Αναγνώριση τύπων κατά το χρόνο εκτέλεσης
- Χώροι ονομάτων
- Καθιερωμένη βιβλιοθήκη προτύπων (STL)
- Αρχές αντικειμενοστρεφούς προγραμματισμού (OOP)
- Και πολλά, πολλά ακόμη!

Το συνοδευτικό CD περιέχει το μεταγλωττιστή Borland C++ 5.5, το ολοκληρωμένο περιβάλλον ανάπτυξης Dev-C++ (εκδόσεις 4 και 5), και τον κώδικα των παραδειγμάτων του βιβλίου.

ISBN 960-209-731-0



9 789602 097311