Περιεχόμενα

Πρόλογος *xxv*

Κεφάλαιο 0 Σημειώσεις για τον αναγνώστη 1

0.1 Η δομή του βιβλίου 2
0.1.1 Η γενική προσέγγιση 3
0.1.2 Ασκήσεις, ευκαιρίες εκπαίδευσης και τα συναφή 4
0.1.3 Τι ακολουθεί μετά από το βιβλίο; 5
0.2 Μία φιλοσοφία διδασκαλίας και εκμάθησης 6
0.2.1 Θεματική οργάνωση του βιβλίου 9
0.2.2 Προγραμματισμός και γλώσσα προγραμματισμού 10
0.2.3 Η έννοια του μεταφερτού προγράμματος 11
0.3 Προγραμματισμός και επιστήμη υπολογιστών 12
0.4 Δημιουργικότητα και επίλυση προβλημάτων 12
0.5 Έκκληση για σχολισαμό 12
0.6 Βιβλιογραφικές παραπομπές 13
0.7 Βιογραφίες 13
Bjarne Stroustrup 14
Lawrence "Pete" Petersen 15

Κεφάλαιο 1 Υπολογιστές, άνθρωποι και προγραμματισμός 17

1.1 Εισαγωγή 18
1.2 Λογισμικό 19
1.3 Άνθρωποι 21
1.4 Επιστήμη υπολογιστών 24
1.5 Ο πανταχού παρών υπολογιστής 25
1.5.1 Και με οθόνη, και χωρίς 26
1.5.2 Ναυτιλία 26
1.5.3 Τηλεπικοινωνίες 28
1.5.4 Ιατρική 30

1.5.5 Πληροφορία 31
1.5.6 Μία κάθετη άποψη 33
1.5.7 Και τώρα λοιπόν τι; 34
1.6 Ιδανικά για προγραμματιστές 34

Μέρος I Τα Βασικά 41

Κεφάλαιο 2 Γεια σου κόσμε! 43
2.1 Προγράμματα 44
2.2 Το κλασικό, πρώτο πρόγραμμα 45
2.3 Μεταγλώττιση 47
2.4 Σύνδεση 51
2.5 Περιβάλλοντα προγραμματισμού 52

Κεφάλαιο 3 Αντικείμενα, τύποι και τιμές 59
3.1 Είσοδος 60
3.2 Μεταβλητές 62
3.3 Είσοδος και τύποι δεδομένων 64
3.4 Πράξεις και τελεστές 66
3.5 Εκχώρηση και αρχικοποίηση 69
3.5.1 Παράδειγμα: διαγραφή επαναλαμβανόμενων λέξεων 71
3.6 Τελεστές σύνθετης εκχώρησης 73
3.6.1 Παράδειγμα: εύρεση επαναλαμβανόμενων λέξεων 73
3.7 Ονόματα 74
3.8 Τύποι και αντικείμενα 77
3.9 Τύποι δεδομένων και ζητήματα ασφάλειας 78
3.9.1 Ασφαλείς μετατροπές 79
3.9.2 Μη-ασφαλείς μετατροπές 80

Κεφάλαιο 4 Υπολογισμός 89
4.1 Εκτέλεση υπολογισμών 90
4.2 Στόχοι και εργαλεία 92
4.3 Εκφράσεις 94
4.3.1 Εκφράσεις με σταθερές 95
4.3.2 Τελεστές 97
4.3.3 Μετατροπές 99
4.4 Εντολές 100
4.4.1 Επιλογή 102
4.4.2 Επανάληψη 109
4.5 Συναρτήσεις 113
4.5.1 Γιατί είναι σημαντικές οι συναρτήσεις; 115
4.5.2 Δηλώσεις συναρτήσεων 117

4.6 Η κλάση vector 117
4.6.1 Διάσχιση ενός vector 119
4.6.2 Ανάπτυξη ενός vector 119
4.6.3 Ένα παράδειγμα με αριθμούς 120
4.6.4 Ένα παράδειγμα με κείμενο 123
4.7 Χαρακτηριστικά της γλώσσας 125

Κεφάλαιο 5 Περί σφαλμάτων 133
5.1 Εισαγωγή 134
5.2 Αίτια πρόκλησης σφαλμάτων 136
5.3 Σφάλματα χρόνου μεταγλώττισης 136
5.3.1 Συντακτικά σφάλματα 137
5.3.2 Σφάλματα σχετιζόμενα με τύπους 138
5.3.3 Μη-σφάλματα 139
5.4 Σφάλματα χρόνου σύνδεσης 139
5.5 Σφάλματα χρόνου εκτέλεσης 140
5.5.1 Ο καλών αντιμετωπίζει τα σφάλματα 142
5.5.2 Ο καλούμενος αντιμετωπίζει τα σφάλματα 143
5.5.3 Αναφορά σφαλμάτων 145
5.6 Εξαιρέσεις 146
5.6.1 Εσφαλμένα ορίσματα 147
5.6.2 Σφάλματα σχετιζόμενα με το εύρος τιμών 148
5.6.3 Εσφαλμένη είσοδος 150
5.6.4 Σφάλματα περιορισμού 153
5.7 Λογικά σφάλματα 154
5.8 Εκτίμηση 157
5.9 Αποσφαλμάτωση 158
5.9.1 Πρακτική συμβουλή αποσφαλμάτωσης 159
5.10 Προ- και μετα- συνθήκες 163
5.10.1 Μετασυνθήκες 165
5.11 Έλεγχος 166

Κεφάλαιο 6 Σύνταξη προγράμματος 173
6.1 Ένα υπό επίλυση πρόβλημα 174
6.2 Μελέτη του προβλήματος 175
6.2.1 Στάδια ανάπτυξης 176
6.2.2 Στρατηγική 176
6.3 Επιστροφή στην αριθμομηχανή! 178
6.3.1 Πρώτη απόπειρα 179
6.3.2 Λεκτικές μονάδες 181
6.3.3 Υλοποίηση λεκτικών μονάδων 183
6.3.4 Χρήση λεκτικών μονάδων 185
6.3.5 Επιστροφή στον πίνακα σχεδίασης 186

6.4 Γραμματικές 188
6.4.1 Γραμματική και φυσικές γλώσσες 193
6.4.2 Σύνταξη γραμματικής 194
6.5 Μετατροπή γραμματικής σε κώδικα 195
6.5.1 Υλοποίηση κανόνων γραμματικής 196
6.5.2 Εκφράσεις 197
6.5.3 Όροι 200
6.5.4 Εκφράσεις πρωταρχικών όρων 202
6.6 Δοκιμή της πρώτης έκδοσης 203
6.7 Δοκιμή της δεύτερης έκδοσης 208
6.8 Ροές λεκτικών μονάδων 209
6.8.1 Υλοποίηση του Token_stream 211
6.8.2 Ανάγνωση λεκτικών μονάδων 212
6.8.3 Ανάγνωση αριθμών 214
6.9 Δομή προγράμματος 215

Κεφάλαιο 7 Ολοκλήρωση ενός προγράμματος 221

7.1 Εισαγωγή 222
7.2 Είσοδος και έξοδος 222
7.3 Χειρισμός σφαλμάτων 224
7.4 Αρνητικοί αριθμοί 229
7.5 Υπόλοιπο % 230
7.6 Τακτοποίηση του κώδικα 232
7.6.1 Συμβολικές σταθερές 232
7.6.2 Χρήση συναρτήσεων 234
7.6.3 Διάταξη κώδικα 235
7.6.4 Προσθήκη σχολίων 237
7.7 Ανάκαμψη από σφάλματα 239
7.8 Μεταβλητές 242
7.8.1 Μεταβλητές και ορισμοί 242
7.8.2 Εισαγωγή ονομάτων 247
7.8.3 Προκαθορισμένα ονόματα 250
7.8.4 Κοντεύουμε; 250

Κεφάλαιο 8 Αναγκαίες λεπτομέρειες: Συναρτήσεις και τα συναφή 255

8.1 Λεπτομέρειες, πλην όμως απαραίτητες 256
8.2 Δηλώσεις και ορισμοί 257
8.2.1 Είδη δηλώσεων 261
8.2.2 Δηλώσεις μεταβλητών και σταθερών 262
8.2.3 Αρχικοποίηση με προεπιλογές 263

8.3 Header αρχεία 264
8.4 Η έννοια της εμβέλειας 266
8.5 Κλήση συνάρτησης και επιστροφή από συνάρτηση 272
8.5.1 Δήλωση ορισμάτων και επιστραφόμενου τύπου 272
8.5.2 Επιστροφή τιμής 274
8.5.3 Πέρασμα διά τιμής 275
8.5.4 Πέρασμα δι' αναφοράς const 276
8.5.5 Πέρασμα δι' αναφοράς 279
8.5.6 Πέρασμα διά τιμής έναντι περάσματος δι' αναφοράς 281
8.5.7 Έλεγχος ορισμάτων και μετατροπές 284
8.5.8 Υλοποίηση κλήσης συνάρτησης 285
8.5.9 Συναρτήσεις constexpr 290
8.6 Σειρά αποτίμησης 291
8.6.1 Αποτίμηση έκφρασης 292
8.6.2 Καθολική αρχικοποίηση 293
8.7 Χώροι ονομάτων 294
8.7.1 Δηλώσεις και ντιρεκτίβες using 296

Κεφάλαιο 9 Αναγκαίες λεπτομέρειες: Κλάσεις και τα συναφή 303
9.1 Καθοριζόμενοι από τον χρήστη τύποι 304
9.2 Κλάσεις και μέλη 305
9.3 Διεπαφή και υλοποίηση 306
9.4 Εξέλιξη μιας κλάσης 308
9.4.1 struct και συναρτήσεις 308
9.4.2 Συναρτήσεις-μέλη και constructor 310
9.4.3 Οι λεπτομέρειες παραμένουν ιδιωτικές 312
9.4.4 Ορισμός συναρτήσεων-μελών 314
9.4.5 Αναφορές στο τρέχον αντικείμενο 317
9.4.6 Αναφορά σφαλμάτων 317
9.5 Απαριθμήσεις 318
9.5.1 "Απλές" απαριθμήσεις 320
9.6 Υπερφόρτωση τελεστών 321
9.7 Διεπαφές κλάσεων 323
9.7.1 Τύποι ορισμάτων 324
9.7.2 Αντιγραφή 326
9.7.3 Προκαθορισμένες συναρτήσεις constructor 327
9.7.4 Συναρτήσεις-μέλη const 330
9.7.5 Μέλη και "βοηθητικές" συναρτήσεις 332
9.8 Η κλάση Date 334

Μέρος II Είσοδος και έξοδος 343

Κεφάλαιο 10 Ροές εισόδου και εξόδου 345

10.1 Είσοδος και έξοδος (E/E) 346
10.2 Το μοντέλο ροών E/E 347
10.3 Αρχεία 349
10.4 Άνοιγμα αρχείων 350
10.5 Ανάγνωση σε και εγγραφή από αρχεία 352
10.6 Χειρισμός σφαλμάτων E/E 354
10.7 Ανάγνωση μεμονωμένης τιμής 358
10.7.1 Ανάλυση του προβλήματος σε διαχειρίσιμα μέρη 359
10.7.2 Διαχωρισμός διαλόγου από τη λειτουργικότητα 362
10.8 Καθοριζόμενοι από τον χρήστη τελεστές εξόδου 363
10.9 Καθοριζόμενοι από τον χρήστη τελεστές εισόδου 365
10.10 Ένας τυπικός βρόχος ανάγνωσης εισόδου 365
10.11 Ανάγνωση από δομημένα αρχεία 367
10.11.1 Αναπαράσταση στη μνήμη 368
10.11.2 Ανάγνωση δομημένων τιμών 370
10.11.3 Αλλαγή αναπαραστάσεων 374

Κεφάλαιο 11 Εξειδίκευση εισόδου και εξόδου 379

11.1 Κανονικότητα και η έλλειψή της 380
11.2 Μορφοποίηση της εξόδου 380
11.2.1 Έξοδος ακεραίων 381
11.2.2 Είσοδος ακεραίων 383
11.2.3 Έξοδος τιμών κινητής υποδιαστολής 384
11.2.4 Ακρίβεια 385
11.2.5 Πεδία 387
11.3 Άνοιγμα και εντοπισμός θέσης σε αρχεία 388
11.3.1 Καταστάσεις ανοίγματος αρχείων 388
11.3.2 Δυαδικά αρχεία 390
11.3.3 Εντοπισμός θέσης σε αρχεία 393
11.4 Ροές συμβολοσειρών 394
11.5 Είσοδος γραμμή προς γραμμή 395
11.6 Ταξινόμηση χαρακτήρων 396
11.7 Χρήση μη-τυπικών χαρακτήρων διαχωρισμού 398
11.8 Έχουμε πολύ δρόμο ακόμα… 406

Κεφάλαιο 12 Ένα μοντέλο για προβολή στην οθόνη 411

12.1 Γιατί τόσο μελάνι για τα γραφικά; 412
12.2 Ένα μοντέλο για προβολή γραφικών στην οθόνη 413
12.3 Ένα πρώτο παράδειγμα 414
12.4 Χρήση βιβλιοθήκης GUI 418

12.5 Συντεταγμένες 419
12.6 Σχήματα: η κλάση Shape 420
12.7 Χρήση βασικών σχημάτων 421
12.7.1 Header αρχεία γραφικών και main 421
12.7.2 Ένα σχεδόν κενό παράθυρο 422
12.7.3 Axis 424
12.7.4 Γραφική παράσταση συνάρτησης 426
12.7.5 Polygon 427
12.7.6 Rectangle 428
12.7.7 Γεμίσματα 431
12.7.8 Text 431
12.7.9 Image 433
12.7.10 Και πολλά άλλα… 434
12.8 Εκτέλεση του προγράμματος 435
12.8.1 Αρχεία πηγαίου κώδικα 437

Κεφάλαιο 13 Κλάσεις γραφικών 441

13.1 Επισκόπηση των κλάσεων γραφικών 442
13.2 Point και Line 444
13.3 Γραμμές με την Line 447
13.4 Χρώμα με την Color 450
13.5 Στιλ γραμμών με την Line_style 452
13.6 Open_polyline 455
13.7 Closed_polyline 456
13.8 Polygon 458
13.9 Rectangle 460
13.10 Διαχείριση ανώνυμων αντικειμένων 465
13.11 Text 467
13.12 Circle 470
13.13 Ellipse 472
13.14 Marked_polyline 474
13.15 Marks 476
13.16 Mark 478
13.17 Εικόνες με την Image 479

Κεφάλαιο 14 Σχεδίαση κλάσεων γραφικών 487

14.1 Αρχές σχεδίασης 488
14.1.1 Τύποι 488
14.1.2 Λειτουργίες 490
14.1.3 Ονομασία 491
14.1.4 Δυνατότητα μετάλλαξης 492
14.2 Shape 493

14.2.1 Μία αφηρημένη κλάση 495
14.2.2 Έλεγχος πρόσβασης 496
14.2.3 Σχεδίαση σχημάτων 500
14.2.4 Αντιγραφή και μετάλλαξη 503
14.3 Βασικές και παραγόμενες κλάσεις 504
14.3.1 Διάταξη αντικειμένων 506
14.3.2 Παραγόμενες κλάσεις και ορισμός εικονικών συναρτήσεων 507
14.3.3 Υπερίσχυση 508
14.3.4 Προσπέλαση 511
14.3.5 Αμιγώς εικονικές συναρτήσεις 512
14.4 Πλεονεκτήματα του αντικειμενοστραφούς προγραμματισμού 513

Κεφάλαιο 15 Γραφική αναπαράσταση συναρτήσεων και δεδομένων 519
15.1 Εισαγωγή 520
15.2 Γραφικές παραστάσεις απλών συναρτήσεων 520
15.3 Function 524
15.3.1 Προεπιλεγμένα ορίσματα 525
15.3.2 Επιπλέον παραδείγματα 527
15.3.3 Εκφράσεις λ (λάμδα) 528
15.4 Axis 529
15.5 Διαδικασία προσέγγισης 532
15.6 Γραφική αναπαράσταση δεδομένων 537
15.6.1 Ανάγνωση από αρχείο 539
15.6.2 Γενική διάταξη 541
15.6.3 Κλιμάκωση δεδομένων 542
15.6.4 Κατασκευή του γραφήματος 543

Κεφάλαιο 16 Βασιζόμενες σε γραφικά διεπαφές χρήστη 551
16.1 Επιλογές για διεπαφές χρήστη 552
16.2 Το κουμπί "Next" 553
16.3 Ένα απλό παράθυρο 554
16.3.1 Μία συνάρτηση επανάκλησης 556
16.3.2 Ένας βρόχος αναμονής 559
16.3.3 Μία έκφραση-λ ως συνάρτηση επανάκλησης 560
16.4 Κουμπιά και άλλοι μηχανισμοί επί της οθόνης 561
16.4.1 Widget 561
16.4.2 Button 563
16.4.3 In_box και Out_box 563
16.4.4 Menu 564

16.5 Ένα παράδειγμα 565
16.6 Αντιστροφή ελέγχου εκτέλεσης 569
16.7 Προσθήκη μενού 570
16.8 Αποσφαλμάτωση κώδικα GUI 575

Μέρος III Δεδομένα και αλγόριθμοι 581

Κεφάλαιο 17 Διανύσματα και χώρος ελεύθερης αποθήκευσης 583

17.1 Εισαγωγή 584
17.2 Τα βασικά της κλάσης vector 586
17.3 Μνήμη, διευθύνσεις και δείκτες 588
17.3.1 Ο τελεστής sizeof 590
17.4 Χώρος ελεύθερης αποθήκευσης και δείκτες 591
17.4.1 Δέσμευση χώρου ελεύθερης αποθήκευσης 593
17.4.2 Προσπέλαση μέσω δεικτών 594
17.4.3 Περιοχές 595
17.4.4 Αρχικοποίηση 596
17.4.5 Ο δείκτης null 598
17.4.6 Αποδέσμευση χώρου ελεύθερης αποθήκευσης 598
17.5 Συναρτήσεις destructor 601
17.5.1 Παραγόμενες συναρτήσεις destructor 603
17.5.2 Συναρτήσεις destructor και χώρος ελεύθερης αποθήκευσης 604
17.6 Προσπέλαση στοιχείων 605
17.7 Δείκτες σε αντικείμενα κλάσεων 606
17.8 Μπερδέματα με τύπους: void* και μετατροπές 608
17.9 Δείκτες και αναφορές 610
17.9.1 Παράμετροι δεικτών και αναφορών 611
17.9.2 Δείκτες, αναφορές και κληρονομικότητα 612
17.9.3 Παράδειγμα: Λίστες 613
17.9.4 Λειτουργίες για λίστες 615
17.9.5 Χρήση λιστών 616
17.10 Ο δείκτης this 618
17.10.1 Περισσότερα για τον χειρισμό λιστών 620

Κεφάλαιο 18 Διανύσματα και διατάξεις 627

18.1 Εισαγωγή 628
18.2 Αρχικοποίηση 629
18.3 Αντιγραφή 631
18.3.1 Συναρτήσεις constructor για δημιουργία αντιγράφων 633
18.3.2 Εκχωρήσεις αντιγράφων 634

18.3.3 Ζητήματα ορολογίας 636
18.3.4 Μετακινήσεις 637
18.4 Βασικές λειτουργίες 640
18.4.1 Ρητές συναρτήσεις constructor 642
18.4.2 Αποσφαλμάτωση συναρτήσεων constructor και destructor 643
18.5 Προσπέλαση στοιχείων ενός vector 646
18.5.1 Υπερφόρτωση για την περίπτωση των const 647
18.6 Διατάξεις 648
18.6.1 Δείκτες προς στοιχεία διατάξεων 650
18.6.2 Δείκτες και διατάξεις 652
18.6.3 Αρχικοποίηση διατάξεων 654
18.6.4 Προβλήματα με τους δείκτες 656
18.7 Παράδειγμα: Παλίνδρομα 659
18.7.1 Παλίνδρομα με χρήση string 659
18.7.2 Παλίνδρομα με χρήση διατάξεων 660
18.7.3 Παλίνδρομα με χρήση δεικτών 661

Κεφάλαιο 19 Διανύσματα, πρότυπα και εξαιρέσεις 667

19.1 Τα προβλήματα 668
19.2 Αλλαγή μεγέθους 671
19.2.1 Αναπαράσταση 671
19.2.2 reserve και capacity 673
19.2.3 resize 674
19.2.4 push_back 674
19.2.5 Εκχώρηση 675
19.2.6 Η κατάσταση του vector μας έως τώρα 677
19.3 Πρότυπα 678
19.3.1 Τύποι ως παράμετροι προτύπων 679
19.3.2 Γενικευμένος προγραμματισμός 681
19.3.3 Βασικές έννοιες 683
19.3.4 Περιέκτες και κληρονομικότητα 686
19.3.5 Ακέραιοι ως παράμετροι προτύπων 687
19.3.6 Απομείωση ορισμάτων 689
19.3.7 Γενίκευση του vector 690
19.4 Έλεγχος εύρους τιμών και εξαιρέσεις 693
19.4.1 Σύντομη παράκαμψη: ζητήματα σχεδίασης 694
19.4.2 Μία ομολογία: μακροεντολές 696
19.5 Πόροι και εξαιρέσεις 697
19.5.1 Πιθανά προβλήματα διαχείρισης πόρων 698
19.5.2 Απόκτηση πόρων ίσον αρχικοποίηση 700
19.5.3 Εγγυήσεις 701
19.5.4 unique_ptr 703

19.5.5 Επιστροφή μέσω μετακίνησης 704
19.5.6 Πόροι και αρχικοποίηση για την κλάση vector 705

Κεφάλαιο 20 Περιέκτες και επαναλήπτες 711

20.1 Αποθήκευση και επεξεργασία δεδομένων 712
20.1.1 Χειρισμός δεδομένων 713
20.1.2 Γενίκευση του κώδικα 714
20.2 Ιδανικά της STL 717
20.3 Ακολουθίες και επαναλήπτες 720
20.3.1 Επιστροφή στο παράδειγμα 723
20.4 Διασυνδεδεμένες λίστες 724
20.4.1 Λειτουργίες για λίστες 726
20.4.2 Επανάληψη 727
20.5 Γενίκευση της κλάσης vector, ξανά 729
20.5.1 Διάσχιση περιέκτη 732
20.5.2 auto 732
20.6 Παράδειγμα: απλός επεξεργαστής κειμένου 734
20.6.1 Γραμμές 736
20.6.2 Επανάληψη 737
20.7 vector, list, και string 741
20.7.1 insert και erase 742
20.8 Προσαρμογή του vector μας στις απαιτήσεις της STL 745
20.9 Προσαρμογή εγγενών διατάξεων στις απαιτήσεις της STL 747
20.10 Περιέκτες, συντομη επισκόπηση 749
20.10.1 Κατηγορίες επαναληπτών 751

Κεφάλαιο 21 Αλγόριθμοι και χάρτες αντιστοίχισης 757

21.1 Αλγόριθμοι της βιβλιοθήκης STL 758
21.2 Ο απλούστερος δυνατός αλγόριθμος: find() 759
21.2.1 Ενδεικτικές, γενικού σκοπού χρήσεις 761
21.3 Γενικευμένη μορφή αναζήτησης: find_if() 763
21.4 Αντικείμενα συναρτήσεων 765
21.4.1 Αφαιρετική θεώρηση των αντικειμένων συναρτήσεων 766
21.4.2 Κατηγορήματα σε μέλη κλάσεων 767
21.4.3 Εκφράσεις-λ 769
21.5 Αριθμητικοί αλγόριθμοι 770
21.5.1 Συσσώρευση 770
21.5.2 Γενίκευση του accumulate() 772
21.5.3 Εσωτερικό γινόμενο: Inner product 774
21.5.4 Γενίκευση του inner_product() 775
21.6 Συσχετιστικοί περιέκτες 776

21.6.1 map: χάρτες αντιστοίχισης 776
21.6.2 Επισκόπηση της κλάσης map 779
21.6.3 Ένα επιπλέον παράδειγμα 782
21.6.4 unordered_map 785
21.6.5 Σύνολα με την κλάση set 787
21.7 Αντιγραφές 789
21.7.1 Δημιουργία αντιγράφων 789
21.7.2 Εφαρμογή επαναληπτών σε ροές 790
21.7.3 Χρήση ενός set για τήρηση σειράς 793
21.7.4 copy_if : αντιγραφή υπό συνθήκη 794
21.8 Ταξινόμηση και αναζήτηση 794
21.9 Αλγόριθμοι για περιέκτες 797

Μέρος IV Μία Ευρύτερη Θεώρηση των Πραγμάτων 803

Κεφάλαιο 22 Ιδανικά και ιστορία 805
22.1 Ιστορία, ιδανικά και επαγγελματισμός 806
22.1.1 Στόχοι και φιλοσοφία των γλωσσών προγραμματισμού 807
22.1.2 Τα ιδανικά του προγραμματισμού 808
22.1.3 Στιλ και πρότυπα προγραμματισμού 815
22.2 Ιστορική επισκόπηση των γλωσσών προγραμματισμού 818
22.2.1 Οι πρώτες γλώσσες 819
22.2.2 Η προέλευση των σύγχρονων γλωσσών 821
22.2.3 Η οικογένεια Algol 826
22.2.4 Η Simula 833
22.2.5 Η C 836
22.2.6 Η C++ 839
22.2.7 Η κατάσταση σήμερα 842
22.2.8 Πηγές πληροφοριών 844

Κεφάλαιο 23 Χειρισμός κειμένου 849
23.1 Κείμενο 850
23.2 Συμβολοσειρές 850
23.3 Ροές E/E 855
23.4 Χάρτες αντιστοίχισης 855
23.4.1 Λεπτομέρειες υλοποίησης 861
23.5 Ένα πρόβλημα 864
23.6 Το σκεπτικό των υποδειγματικών εκφράσεων 866
23.6.1 "Ακατέργαστες" συμβολοσειρές 868
23.7 Αναζήτηση με υποδειγματικές εκφράσεις 869
23.8 Σύνταξη υποδειγματικών εκφράσεων 872
23.8.1 Χαρακτήρες, τυπικοί και ειδικοί 872

23.8.2 Κλάσεις για χαρακτήρες 873
23.8.3 Επαναλήψεις 874
23.8.4 Ομαδοποίηση 876
23.8.5 Εναλλαγή 876
23.8.6 Σύνολα χαρακτήρων 877
23.8.7 Σφάλματα κατά τη χρήση υποδειγματικών εκφράσεων 878
23.9 Χρήση υποδειγματικών εκφράσεων για ταυτίσεις 880
23.10 Βιβλιογραφικές παραπομπές 885

Κεφάλαιο 24 Αριθμοί και υπολογισμοί 889

24.1 Εισαγωγή 890
24.2 Μέγεθος, ακρίβεια και υπερχείλιση 890
24.2.1 Όρια αριθμητικών δεδομένων 894
24.3 Διατάξεις 895
24.4 Πολυδιάστατες διατάξεις με το στιλ της C 896
24.5 Η βιβλιοθήκη Matrix 897
24.5.1 Διαστάσεις και προσπέλαση 898
24.5.2 1D Matrix 901
24.5.3 2D Matrix 904
24.5.4 Είσοδος/Έξοδος σε Matrix 907
24.5.5 3D Matrix 907
24.6 Παράδειγμα: επίλυση γραμμικών εξισώσεων 908
24.6.1 Κλασική απαλοιφή κατά Gauss 910
24.6.2 Αναδιάταξη με περιστροφή 911
24.6.3 Έλεγχος 912
24.7 Τυχαίοι αριθμοί 914
24.8 Τυπικές μαθηματικές συναρτήσεις 917
24.9 Μιγαδικοί αριθμοί 919
24.10 Βιβλιογραφικές παραπομπές 920

Κεφάλαιο 25 Προγραμματισμός ενσωματωμένων συστημάτων 925

25.1 Εισαγωγή στα ενσωματωμένα συστήματα 926
25.2 Βασικές έννοιες 929
25.2.1 Προβλεψιμότητα 932
25.2.2 Ιδανικά 932
25.2.3 Το αναπόφευκτο της αποτυχίας 933
25.3 Διαχείριση μνήμης 935
25.3.1 Προβλήματα με τον χώρο ελεύθερης αποθήκευσης 936
25.3.2 Εναλλακτικές για τον χώρο ελεύθερης αποθήκευσης 939
25.3.3 Δομή δεξαμενής 940
25.3.4 Δομή στοίβας 942
25.4 Διευθύνσεις, δείκτες και διατάξεις 943

25.4.1 Μετατροπές χωρίς τους αναγκαίους ελέγχους 943
25.4.2 Πρόβλημα: δυσλειτουργικές διεπαφές 944
25.4.3 Λύση: μία κλάση για τη διεπαφή 947
25.4.4 Κληρονομικότητα και περιέκτες 951
25.5 Bits, bytes και words 954
25.5.1 Bits και πράξεις με bits 955
25.5.2 bitset 959
25.5.3 Με και χωρίς πρόσημο 961
25.5.4 Επεξεργασία σε επίπεδο bit 965
25.5.5 Πεδία bit 967
25.5.6 Παράδειγμα: απλή κρυπτογράφηση 969
25.6 Πρότυπα κωδικοποίησης 974
25.6.1 Τι θα πρέπει να είναι ένα πρότυπο κωδικοποίησης; 975
25.6.2 Δείγματα κανόνων 977
25.6.3 Πραγματικά πρότυπα κωδικοποίησης 983

Κεφάλαιο 26 Έλεγχοι και δοκιμές 989

26.1 Τι θέλουμε 990
26.1.1 Μία προειδοποίηση 991
26.2 Αποδείξεις 992
26.3 Έλεγχοι 992
26.3.1 Έλεγχοι παλινδρόμησης 993
26.3.2 Έλεγχοι μονάδων 994
26.3.3 Αλγόριθμοι και μη-αλγόριθμοι 1001
26.3.4 Έλεγχοι σε επίπεδο συστήματος 1009
26.3.5 Εντοπισμός αβάσιμων υποθέσεων 1009
26.4 Σχεδιασμός για ελέγχους/δοκιμές 1011
26.5 Αποσφαλμάτωση 1012
26.6 Απόδοση 1012
26.6.1 Χρονομέτρηση 1015
26.7 Βιβλιογραφικές παραπομπές 1016

Κεφάλαιο 27 Η γλώσσα προγραμματισμού C 1021

27.1 C και C++: Μία αδελφική σχέση 1022
27.1.1 Συμβατότητα μεταξύ των C και C++ 1024
27.1.2 Χαρακτηριστικά της C++ που λείπουν από τη C 1025
27.1.3 Η βιβλιοθήκη προτύπων της C 1027
27.2 Συναρτήσεις 1028
27.2.1 Δεν επιτρέπεται υπερφόρτωση ονομάτων συναρτήσεων 1028
27.2.2 Έλεγχοι τύπων για ορίσματα συναρτήσεων 1029
27.2.3 Ορισμοί συναρτήσεων 1031
27.2.4 Κλήση κώδικα C από τη C++ και το αντίστροφο 1032

27.2.5 Δείκτες προς συναρτήσεις 1034
27.3 Ήσσονος σημασίας διαφορές των δύο γλωσσών 1036
27.3.1 struct και χώροι ονομάτων 1036
27.3.2 Δεσμευμένες λέξεις 1037
27.3.3 Ορισμοί 1038
27.3.4 Μετατροπές μεταξύ τύπων με το στιλ της C 1040
27.3.5 Μετατροπή του void* 1041
27.3.6 enum 1042
27.3.7 Χώροι ονομάτων 1042
27.4 Χώρος ελεύθερης αποθήκευσης 1043
27.5 Συμβολοσειρές με το στιλ της C 1045
27.5.1 Συμβολοσειρές με το στιλ της C και const 1047
27.5.2 Λειτουργίες σε bytes 1048
27.5.3 Παράδειγμα:: strcpy() 1049
27.5.4 Ένα ζήτημα στιλ 1049
27.6 Είσοδος/Έξοδος: stdio 1050
27.6.1 Έξοδος 1050
27.6.2 Είσοδος 1052
27.6.3 Αρχεία 1053
27.7 Σταθερές και μακροεντολές 1054
27.8 Μακροεντολές 1055
27.8.1 Μακροεντολές που ενεργούν ως συναρτήσεις 1056
27.8.2 Μακροεντολές σύνταξης 1058
27.8.3 Μεταγλώττιση υπό συνθήκη 1058
27.9 Παράδειγμα: διεξοδικοί περιέκτες 1059

Μέρος V Οδηγοί Αναφοράς 1071

Κεφάλαιο 28: Συνοπτική παρουσίαση της γλώσσας 1073

28.1 Γενικά 1074
28.1.1 Ορολογία 1075
28.1.2 Έναρξη και τερματισμός προγράμματος 1075
28.1.3 Σχόλια 1076
28.2 Literals 1077
28.2.1 Literal ακεραίων 1077
28.2.2 Literal τιμών κινητής υποδιαστολής 1079
28.2.3 Literal τύπου Boole 1079
28.2.4 Literal χαρακτήρων 1079
28.2.5 Literal συμβολοσειρών 1080
28.2.6 Literal δεικτών 1081
28.3 Αναγνωριστικά 1081
28.3.1 Δεσμευμένες λέξεις 1081

28.4 Εμβέλεια, αποθήκευση και διάρκεια ζωής 1082
28.4.1 Εμβέλεια 1082
28.4.2 Κλάση αποθήκευσης 1083
28.4.3 Διάρκεια ζωής 1085
28.5 Εκφράσεις 1086
28.5.1 Καθοριζόμενοι από τον χρήστη τελεστές 1091
28.5.2 Έμμεσες μετατροπές τύπων 1091
28.5.3 Εκφράσεις με σταθερές 1093
28.5.4 Ο τελεστής sizeof 1093
28.5.5 Λογικές εκφράσεις 1094
28.5.6 new και delete 1094
28.5.7 Μετατροπές μεταξύ τύπων 1095
28.6 Εντολές 1096
28.7 Δηλώσεις 1098
28.7.1 Ορισμοί 1098
28.8 Εγγενείς τύποι δεδομένων 1099
28.8.1 Δείκτες 1100
28.8.2 Διατάξεις 1101
28.8.3 Αναφορές 1102
28.9 Συναρτήσεις 1103
28.9.1 Χειρισμός υπερφόρτωσης 1104
28.9.2 Προεπιλεγμένα ορίσματα 1105
28.9.3 Μη-προσδιοριζόμενα ορίσματα 1105
28.9.4 Προδιαγραφές σύνδεσης 1106
28.10 Καθοριζόμενοι από τον χρήστη τύποι δεδομένων 1106
28.10.1 Υπερφόρτωση τελεστών 1107
28.11 Απαριθμήσεις 1107
28.12 Κλάσεις 1108
28.12.1 Προσπέλαση μελών 1108
28.12.2 Ορισμοί μελών κλάσης 1112
28.12.3 Κατασκευή, καταστροφή και αντιγραφή 1112
28.12.4 Παραγόμενες κλάσεις 1116
28.12.5 Πεδία bit 1120
28.12.6 Ενώσεις 1121
28.13 Πρότυπα 1121
28.13.1 Ορίσματα προτύπων 1122
28.13.2 Δημιουργία συγκεκριμένων προτύπων 1123
28.13.3 Τύποι μελών προτύπων 1124
28.14 Εξαιρέσεις 1125
28.15 Χώροι ονομάτων 1127
28.16 Ψευδώνυμα 1128
28.17 Ντιρεκτίβες του προεπεξεργαστή 1128

28.17.1 #include 1128
28.17.2 #define 1129

Κεφάλαιο 29: Συνοπτική παρουσίαση της βιβλιοθήκης προτύπων 1131

29.1 Επισκόπηση 1132
29.1.1 Header αρχεία 1133
29.1.2 Χώρος ονομάτων std 1136
29.1.3 Στιλ περιγραφής 1136
29.2 Χειρισμός σφαλμάτων 1137
29.2.1 Εξαιρέσεις 1138
29.3 Επαναλήπτες 1139
29.3.1 Μοντέλο επαναληπτών 1140
29.3.2 Κατηγορίες επαναληπτών 1142
29.4 Υποδοχείς 1144
29.4.1 Επισκόπηση 1146
29.4.2 Τύποι μελών 1147
29.4.3 Συναρτήσεις constructor/destructor και εκχωρήσεις 1148
29.4.4 Επαναλήπτες 1148
29.4.5 Προσπέλαση στοιχείων 1149
29.4.6 Λειτουργίες για στοίβες και ουρές 1149
29.4.7 Λειτουργίες για χειρισμό λιστών 1150
29.4.8 Μέγεθος και χωρητικότητα 1150
29.4.9 Άλλες λειτουργίες 1151
29.4.10 Λειτουργίες για συσχετιστικούς περιέκτες 1151
29.5 Αλγόριθμοι 1152
29.5.1 Μη-τροποποιητικοί αλγόριθμοι ακολουθίας 1153
29.5.2 Τροποποιητικοί αλγόριθμοι ακολουθίας 1154
29.5.3 Αλγόριθμοι βοηθητικών λειτουργιών 1156
29.5.4 Αλγόριθμοι ταξινόμησης και αναζήτησης 1157
29.5.5 Αλγόριθμοι για σύνολα 1159
29.5.6 Δομές σωρού 1160
29.5.7 Αντιμεταθέσεις 1160
29.5.8 min και max 1161
29.6 Βοηθητικά εργαλεία της STL 1162
29.6.1 Εισαγωγείς (Inserters) 1162
29.6.2 Αντικείμενα συναρτήσεων 1163
29.6.3 pair και tuple 1165
29.6.4 initializer_list 1166
29.6.5 Δείκτες και διαχείριση πόρων 1167
29.7 Ροές E/E 1168
29.7.1 Ιεραρχία ροών E/E 1170

29.7.2 Χειρισμός σφαλμάτων 1171
29.7.3 Λειτουργίες εισόδου 1172
29.7.4 Λειτουργίες εξόδου 1173
29.7.5 Μορφοποίηση 1173
29.7.6 Τυπικοί χειριστές 1173
29.8 Χειρισμός συμβολοσειρών 1175
29.8.1 Ταξινόμηση χαρακτήρων 1175
29.8.2 Ο τύπος String 1176
29.8.3 Ταύτιση με υποδειγματικές εκφράσεις 1177
29.9 Αριθμοί και υπολογισμοί 1180
29.9.1 Όρια αριθμητικών τιμών 1180
29.9.2 Βασικές μαθηματικές συναρτήσεις 1181
29.9.3 Μιγαδικοί αριθμοί 1182
29.9.4 valarray 1183
29.9.5 Γενικευμένοι αλγόριθμοι αριθμητικών πράξεων/λειτουργιών 1183
29.9.6 Τυχαίοι αριθμοί 1184
29.10 Χρονομέτρηση 1185
29.11 Συναρτήσεις της βιβλιοθήκης προτύπων της C 1185
29.11.1 Αρχεία 1186
29.11.2 Η οικογένεια printf() 1186
29.11.3 Συμβολοσειρές με το στιλ της C 1191
29.11.4 Μνήμη 1192
29.11.5 Ημερομηνία και ώρα 1193
29.10.6 Και τα λοιπά… 1194
29.12 Άλλες βιβλιοθήκες 1195

Κεφάλαιο 30: Εισαγωγή στο Visual Studio 1197

30.1 Εκτέλεση ενός προγράμματος 1198
30.2 Εγκατάσταση του Visual Studio 1198
30.3 Δημιουργία και εκτέλεση ενός προγράμματος 1199
30.3.1 Δημιουργία νέου έργου 1199
30.3.2 Χρήση του header αρχείου std_lib_facilities.h 1199
30.3.3 Προσθήκη πηγαίου κώδικα C++ στο έργο 1200
30.3.4 Εισαγωγή του πηγαίου κώδικα 1200
30.3.5 Κατασκευή εκτελέσιμου προγράμματος 1200
30.3.6 Εκτέλεση του προγράμματος 1201
30.3.7 Αποθήκευση του προγράμματος 1201
30.4 Επόμενα βήματα 1201

Κεφάλαιο 31: Εγκατάσταση του FLTK 1203

31.1 Εισαγωγή 1204
31.2 Λήψη του FLTK 1204

31.3 Εγκατάσταση του FLTK 1205
31.4 Χρήση του FLTK στο Visual Studio 1205
31.5 Έλεγχος ορθής λειτουργίας 1206

Κεφάλαιο 32: Υλοποίηση GUI 1207

32.1 Υλοποίηση επανάκλησης 1208
32.2 Υλοποίηση μηχανισμών με την κλάση Widget 1209
32.3 Υλοποίηση παραθύρων με την κλάση Window 1210
32.4 Η κλάση Vector_ref 1212
32.5 Παράδειγμα: χειρισμός των Widget 1213

Γλωσσάριο Όρων 1217
Βιβλιογραφία 1223
Ευρετήριο 1227

3

Αντικείμενα, τύποι και τιμές

"Η τύχη ευνοεί τον προετοιμασμένο νου"...

– Λουί Παστέρ

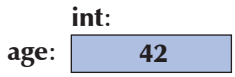
Αυτό το κεφάλαιο παρουσιάζει τη βασική θεωρία για την αποθήκευση και τη χρήση δεδομένων σε ένα πρόγραμμα. Για να γίνει αυτό, επικεντρώνουμε αρχικά το ενδιαφέρον μας στην ανάγνωση δεδομένων από το πληκτρολόγιο. Αφού εδραιωθούν οι θεμελιώδεις έννοιες των αντικειμένων, των τύπων, των τιμών και των μεταβλητών, παρουσιάζουμε αρκετούς τελεστές και δίνουμε πολλά παραδείγματα όπου χρησιμοποιούνται μεταβλητές των τύπων **char**, **int**, **double** και **string**.

3.1 Είσοδος
3.2 Μεταβλητές
3.3 Είσοδος και τύπος
3.4 Πράξεις και τελεστές
3.5 Εκχώρηση και αρχικοποίηση
3.5.1 Ένα παράδειγμα: διαγραφή επαναλαμβανόμενων λέξεων
3.6 Σύνθετοι τελεστές εκχώρησης
3.6.1 Ένα παράδειγμα: μέτρημα επαναλαμβανόμενων λέξεων
3.7 Ονόματα
3.8 Τύποι και αντικείμενα
3.9 Ασφάλεια τύπων
3.9.1 Ασφαλείς μετατροπές
3.9.2 Επισφαλείς μετατροπές

3.1 Είσοδος

Το πρόγραμμα "Γεια σου κόσμε!" απλά γράφει στην οθόνη. Παράγει έξοδο. Δε διαβάζει τίποτα και δε λαμβάνει είσοδο από το χρήστη του. Όλα αυτά σημαίνουν ότι είναι κάπως βαρετό. Τα πραγματικά προγράμματα συνήθως παράγουν αποτελέσματα που βασίζονται σε κάποια είσοδο που τους δίνουμε και δεν κάνουν το ίδιο πράγμα κάθε φορά που τα εκτελούμε.

Για να διαβάσουμε κάτι, χρειαζόμαστε τη θέση όπου θα διαβάζουμε, δηλαδή χρειαζόμαστε ένα μέρος στη μνήμη του υπολογιστή όπου θα τοποθετήσουμε αυτό που θα διαβάσουμε. Ονομάζουμε αυτό το "μέρος" αντικείμενο. Ένα *αντικείμενο* (object) είναι μια περιοχή μνήμης με έναν *τύπο* (type) που καθορίζει το είδος πληροφοριών που μπορούν να τοποθετηθούν σ' αυτό. Ένα αντικείμενο που έχει συγκεκριμένο όνομα ονομάζεται *μεταβλητή* (variable). Για παράδειγμα, οι συμβολοσειρές χαρακτήρων τοποθετούνται σε μεταβλητές **string** και οι ακέραιοι τοποθετούνται σε μεταβλητές **int**. Μπορείτε να σκεφτείτε ότι ένα αντικείμενο είναι ένα "κουτί" στο οποίο μπορείτε να τοποθετήσετε μια τιμή του τύπου του αντικειμένου:



Αυτό θα αναπαριστούσε ένα αντικείμενο τύπου **int** που ονομάζεται **age** και περιέχει την ακέραια τιμή **42**. Χρησιμοποιώντας μια μεταβλητή συμβολοσειράς, μπορούμε να διαβάσουμε μια συμβολοσειρά από είσοδο και να τη γράψουμε ξανά ως εξής:

```
// διάβασε και γράψε ένα όνομα
#include "std_lib_facilities.h"

int main()
{
    cout << "Παρακαλώ εισαγάγετε το μικρό όνομά σας (και μετά πατήστε 'Enter'):\n";
    string first_name;      // το first_name είναι μεταβλητή τύπου string
    cin >> first_name;      // διάβασε χαρακτήρες από το first_name
    cout << "Γεια σου " << first_name << "!\n";
}
```

Τα **#include** και **main()** είναι γνωστά από το Κεφάλαιο 2. Επειδή το **#include** είναι απαραίτητο σε όλα τα προγράμματά μας (έως το Κεφάλαιο 12), δε θα το παρουσιάσουμε, ώστε να μη σας αποσπάσουμε από αυτά που μας ενδιαφέρουν. Ομοίως, μερικές φορές θα παρουσιάζουμε κώδικα που θα λειτουργεί μόνο αν τοποθετηθεί μέσα στη συνάρτηση **main()** ή σε κάποια άλλη συνάρτηση μόνη της, όπως εδώ:

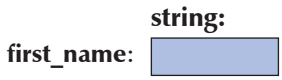
```
cout << "Παρακαλώ εισαγάγετε το μικρό όνομά σας (και μετά πατήστε 'Enter'):\n";
```

Υποθέτουμε ότι μπορείτε να καταλάβετε πώς θα τοποθετείτε τέτοιον κώδικα σε ένα ολοκληρωμένο πρόγραμμα για δοκιμή.

Η πρώτη γραμμή της **main()** απλά εμφανίζει ένα μήνυμα που ενθαρρύνει το χρήστη να εισάγει το μικρό όνομά του. Ένα τέτοιο μήνυμα συνήθως ονομάζεται προτροπή (*prompt*), επειδή προτρέπει το χρήστη να αναλάβει κάποια δράση. Οι επόμενες γραμμές ορίζουν μια μεταβλητή τύπου **string**, η οποία ονομάζεται **first_name**, διαβάζουν είσοδο από το πληκτρολόγιο σ' αυτή τη μεταβλητή και εμφανίζουν ένα χαιρετισμό. Ας εξετάσουμε αυτές τις τρεις γραμμές με τη σειρά:

```
string first_name; // το first_name είναι μεταβλητή τύπου string
```

Αυτή η γραμμή δεσμεύει μια περιοχή μνήμης για μια συμβολοσειρά χαρακτήρων και της δίνει το όνομα **first_name**:

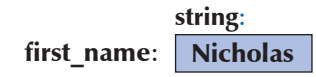


Μια εντολή που εισάγει ένα νέο όνομα σε ένα πρόγραμμα και δεσμεύει μνήμη για μια μεταβλητή ονομάζεται *ορισμός* (definition).

Η επόμενη γραμμή διαβάζει χαρακτήρες από την είσοδο (το πληκτρολόγιο) στη μεταβλητή:

```
cin >> first_name;      // διάβασε χαρακτήρες από το first_name
```

Το όνομα **cin** αναφέρεται στην τυπική ροή εισόδου (προφέρεται "σι-ιν", από το "**c**haracter **in**put" που σημαίνει είσοδος χαρακτήρων), όπως ορίζεται στη βιβλιοθήκη προτύπων. Ο δεύτερος τελεστέος του τελεστή **>>** ("πάρε από") καθορίζει πού πηγαίνει αυτή η είσοδος. Έτσι, αν πληκτρολογήσουμε κάποιο μικρό όνομα, όπως **Nicholas**, και ακολουθήσει μια νέα γραμμή, η συμβολοσειρά **"Nicholas"** γίνεται η τιμή της μεταβλητής **first_name**:



Η νέα γραμμή είναι απαραίτητη, ώστε να αποσπάσουμε την προσοχή του υπολογιστή. Μέχρι να εισαχθεί μια νέα γραμμή (μέχρι δηλαδή να πατήσουμε το πλήκτρο Enter), ο υπολογιστής απλά συλλέγει χαρακτήρες. Αυτή η "καθυστέρηση" σας δίνει την ευκαιρία να αλλάξετε γνώμη, να διαγράψετε μερικούς χαρακτήρες και να τους

αντικαταστήσετε με άλλους, πριν πατήσετε το Enter. Η νέα γραμμή δε θα είναι μέρος της συμβολοσειράς που αποθηκεύεται στη μνήμη.

Αφού πάρουμε τη συμβολοσειρά εισόδου στη μεταβλητή **first_name**, μπορούμε να τη χρησιμοποιήσουμε:

```
cout << "Γεια σου " << first_name << "!\n";
```

Θα εμφανιστεί έτσι ο χαιρετισμός **Γεια σου**, και κατόπιν θα εισάγεται το όνομα, **Nicholas** (η τιμή του **first_name**), θα ακολουθεί ένα **!** και μια νέα γραμμή (**'\n'**) στην οθόνη:

```
Γεια σου Nicholas!
```

Εάν σας αρέσουν η επανάληψη και η πληκτρολόγηση, θα μπορούσατε να γράψετε τρεις διαφορετικές εντολές εξόδου:

```
cout << "Γεια σου ";
cout << first_name;
cout << "!\n";
```

Ωστόσο, δεν είμαστε δακτυλογράφοι και —το πιο σημαντικό— απεχθανόμαστε την επανάληψη (επειδή η επανάληψη δίνει ευκαιρίες στα λάθη), γι' αυτό και συνδυάζουμε αυτές τις τρεις ενέργειες εξόδου σε μια εντολή.

Παρατηρήστε πώς χρησιμοποιούμε εισαγωγικά γύρω από τους χαρακτήρες στη φράση **"Γεια σου "**, αλλά όχι στη μεταβλητή **first_name**. Χρησιμοποιούμε εισαγωγικά όταν θέλουμε μια αυτολεξεί συμβολοσειρά, δηλαδή μια παράθεση. Όταν δε θέλουμε παράθεση, παραπέμπουμε στην τιμή κάποιου αντικειμένου με ένα όνομα. Δείτε το παρακάτω:

```
cout << "first_name" << " is " << first_name;
```

Εδώ, το **"first_name"** μας δίνει τους δέκα χαρακτήρες **first_name** (μικρό όνομα) και το απλό **first_name** μας δίνει την τιμή της μεταβλητής **first_name**, εν προκειμένω το **Nicholas**. Αυτό που θα παίρνουμε ως έξοδο είναι το εξής:

```
first_name is Nicholas
```

3.2 Μεταβλητές

Στην πραγματικότητα, δεν μπορούμε να κάνουμε τίποτα ενδιαφέρον με έναν υπολογιστή αν δεν αποθηκεύσουμε δεδομένα στη μνήμη, όπως κάναμε με τη συμβολοσειρά εισόδου στο πιο πάνω παράδειγμα. Τα "μέρη" στα οποία αποθηκεύουμε δεδομένα ονομάζονται *αντικείμενα* (objects). Για να προσπελάσουμε ένα αντικείμενο, χρειαζόμαστε ένα *όνομα* (name). Ένα αντικείμενο με όνομα ονομάζεται *μεταβλητή* και έχει συγκεκριμένο *τύπο* (όπως **int** ή **string**) που προσδιορίζει τι μπορεί να τοποθετηθεί στο αντικείμενο (π.χ., το **123** μπορεί να τοποθετηθεί σε ένα **int** και το **"Γεια σου κόσμε!\n"** μπορεί να τοποθετηθεί σε ένα **string**), αλλά και ποιες πράξεις μπορούν να γίνουν (π.χ., μπορούμε να πολλαπλασιάσουμε **int** χρησιμοποιώντας τον τελεστή ***** και να συγκρίνουμε **string** χρησιμοποιώντας τον τελεστή **<=**). Τα στοιχεία δεδομένων που τοποθετούμε σε μεταβλητές ονομάζονται *τιμές* (values). Μια εντολή που ορίζει

μια μεταβλητή ονομάζεται *ορισμός* (definition) και ένας ορισμός μπορεί (και συνήθως πρέπει) να παρέχει μια αρχική τιμή. Δείτε αυτό:

```
string name = "Annemarie";
int number_of_steps = 39;
```

Μπορείτε να φανταστείτε αυτές τις μεταβλητές έτσι:

Δεν μπορείτε να τοποθετήσετε τιμές με το λάθος τύπο σε μια μεταβλητή:

```
string name2 = 39;                    // σφάλμα: το 39 δεν είναι string
int number_of_steps = "Annemarie";    // σφάλμα: το "Annemarie" δεν είναι int
```

Ο μεταγλωττιστής θυμάται τον τύπο κάθε μεταβλητής και βεβαιώνει ότι τον χρησιμοποιείτε σύμφωνα με τον τύπο του, όπως ορίζεται στον ορισμό του.

Η C++ παρέχει αρκετά μεγάλο αριθμό τύπων (ανατρέξτε στην §A.8). Ωστόσο, μπορείτε να γράψετε πολύ καλά προγράμματα χρησιμοποιώντας μόνο πέντε απ' αυτούς:

```
int number_of_steps = 39;          // int για ακεραίους
double flying_time = 3.5;          // double για αριθμούς κινητής υποδιαστολής
char decimal_point = '.';          // char για μεμονωμένους χαρακτήρες
string name = "Annemarie";         // string για συμβολοσειρές χαρακτήρων
bool tap_on = true;                // bool για λογικές μεταβλητές
```

Ο λόγος για τη χρήση του ονόματος **double** είναι ιστορικός: το **double** είναι η συντομογραφία του "doubleprecision floating point", που σημαίνει *κινητή υποδιαστολή διπλής ακρίβειας*. Η κινητή υποδιαστολή είναι η προσέγγιση των υπολογιστών στη μαθηματική έννοια ενός πραγματικού αριθμού.

Παρατηρήστε ότι καθένας από αυτούς τους τύπους έχει το δικό του χαρακτηριστικό στιλ για απόλυτες τιμές:

```
39            // int: ακέραιος
3.5           // double: αριθμός κινητής υποδιαστολής
'.'           // char: μεμονωμένος χαρακτήρας που περικλείεται σε μονά εισαγωγικά
"Annemarie"   // string: μια ακολουθία χαρακτήρων που οριοθετείται από διπλά εισαγωγικά
true          // bool: true ή false, δηλαδή σωστό ή λάθος
```

Κατά συνέπεια, μια ακολουθία ψηφίων (όπως **1234**, **2** ή **976**) δηλώνει έναν ακέραιο, ένας χαρακτήρας σε μονά εισαγωγικά (όπως **'1'**, **'@'** ή **'x'**) δηλώνει ένα χαρακτήρα, μια ακολουθία ψηφίων με ένα δεκαδικό σημείο (όπως **1.234**, **0.12** ή **.98**) δηλώνει μια τιμή κινητής υποδιαστολής και μια ακολουθία χαρακτήρων που περικλείεται σε διπλά εισαγωγικά (όπως **"1234"**, **"Howdy!"** ή **"Annemarie"**) δηλώνει μια συμβολοσειρά. Για πιο αναλυτική περιγραφή των απόλυτων, κατά γράμμα, τιμών ανατρέξτε στην §A.2.

3.3 Είσοδος και τύπος

Η πράξη εισόδου **>>** ("πάρε από") είναι ευαίσθητη στον τύπο, που σημαίνει ότι διαβάζει σύμφωνα με τον τύπο της μεταβλητής από την οποία διαβάζετε. Για παράδειγμα:

```
// διάβασε όνομα και ηλικία
int main()
{
    cout << "Παρακαλώ εισαγάγετε το μικρό όνομά σας και την ηλικία σας\n";
    string first_name;          // μεταβλητή συμβολοσειράς
    int age;                    // μεταβλητή ακέραιου
    cin >> first_name;          // διάβασε μια μεταβλητή
    cin >> age;                 // διάβασε έναν ακέραιο
    cout << "Γεια σου " << first_name << " (ηλικία " << age << ")\n";
}
```

Αν λοιπόν πληκτρολογήσετε **Carlos 22**, ο τελεστής **>>** θα διαβάσει **Carlos** στο **first_name**, **22** στο **age** και θα παραγάγει την εξής έξοδο:

```
Γεια σου Carlos (ηλικία 22)
```

Γιατί δεν διαβάστηκε όλο το **Carlos 22** στη μεταβλητή **first_name**; Επειδή, λόγω συμβάσεων, η ανάγνωση συμβολοσειρών τερματίζεται από αυτό που ονομάζεται *κενός χώρος (whitespace)*, δηλαδή κενό διάστημα, νέα γραμμή και χαρακτήρας tab. Διαφορετικά, ο κενός χώρος αγνοείται, εξ ορισμού, από το **>>**. Για παράδειγμα, μπορείτε να προσθέσετε όσα κενά θέλετε πριν από έναν αριθμό που θα διαβαστεί και το **>>** απλά θα τα αγνοήσει και θα διαβάσει τον αριθμό.

Εάν εισαγάγετε **22 Carlos**, θα δείτε κάτι που ίσως σας εκπλήξει, μέχρι να το σκεφτείτε καλύτερα. Το **22** θα διαβαστεί στο **first_name** επειδή, τελικά, το **22** είναι μια ακολουθία χαρακτήρων. Από την άλλη πλευρά, το **Carlos** δεν είναι ακέραιος, επομένως δε θα διαβαστεί. Η έξοδος θα είναι **22** και θα ακολουθεί κάποιος τυχαίος αριθμός, όπως **–96739** ή **0**. Γιατί; Δε δώσατε στο **age** μια αρχική τιμή και δεν καταφέρατε να διαβάσετε μια τιμή σ' αυτό. Επομένως, παίρνετε μια άχρηστη τιμή που έτυχε να βρίσκεται σ' αυτό το κομμάτι μνήμης, όταν ξεκινήσατε την εκτέλεση. Στην §10.6, μελετάμε τρόπους χειρισμού "σφαλμάτων μορφής εισόδου". Προς το παρόν, ας αρχικοποιήσουμε απλά το **age**, ώστε να πάρουμε μια προβλέψιμη τιμή αν αποτύχει η είσοδος:

```
// διάβασε όνομα και ηλικία (2η έκδοση)
int main()
{
    cout << "Παρακαλώ εισαγάγετε το μικρό όνομά σας και την ηλικία σας\n";
```

```
    string first_name = ";;;";      // μεταβλητή συμβολοσειράς
                                    // (";;;" σημαίνει "δε γνωρίζω το όνομα")
    int age = –1;   // μεταβλητή ακέραιου (το –1 σημαίνει "δε γνωρίζω την ηλικία")
    cin >> first_name >> age;  // διάβασε μια συμβολοσειρά και μετά έναν ακέραιο
    cout << "Γεια σου " << first_name << " (ηλικία " << age << ")\n";
}
```

Τώρα, η είσοδος **22 Carlos** θα εξαγάγει

```
Γεια σου 22 (ηλικία –1)
```

Παρατηρήστε ότι μπορούμε να διαβάσουμε αρκετές τιμές σε μια μόνο εντολή εισόδου, όπως μπορούμε να γράψουμε αρκετές τιμές σε μια μόνο εντολή εξόδου. Παρατηρήστε επίσης ότι το << είναι ευαίσθητο στους τύπους, όπως είναι και το >>, οπότε, μπορούμε να εξαγάγουμε την τύπου **int** μεταβλητή **age** και το χαρακτήρα **'\n'** όπως και την τύπου **string** μεταβλητή **first_name** και τις συμβολοσειρές **"Γεια σου "** και **" (ηλικία "**.

Ένα **string** που διαβάζεται χρησιμοποιώντας το >> (εξ ορισμού) τερματίζεται από κενό χώρο, που σημαίνει ότι διαβάζει μια μόνο λέξη. Αλλά μερικές φορές θέλουμε να διαβάσουμε περισσότερες από μια λέξεις. Υπάρχουν ασφαλώς πολλοί τρόποι να το κάνουμε. Για παράδειγμα, μπορούμε να διαβάσουμε ένα όνομα που αποτελείται από δύο λέξεις ως εξής:

```
int main()
{
    cout << "Παρακαλώ εισαγάγετε το πρώτο και το δεύτερο μικρό όνομά σας\n";
    string first;
    string second;
    cin >> first >> second;       // διάβασε δύο συμβολοσειρές
    cout << "Γεια σου " << first << ' ' << second << '\n';
}
```

Απλά χρησιμοποιήσαμε το >> δύο φορές, μια για κάθε όνομα. Όταν θέλουμε να γράψουμε τα ονόματα στην έξοδο, πρέπει να εισαγάγουμε ένα κενό μεταξύ τους.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Τρέξτε το παράδειγμα "όνομα και ηλικία". Κατόπιν τροποποιήστε το, ώστε να γράψετε την ηλικία σε μήνες: Διαβάστε την είσοδο σε χρόνια και πολλαπλασιάστε (χρησιμοποιώντας τον τελεστή *) επί 12. Διαβάστε την ηλικία σε ένα **double**, ώστε να προνοήσετε και για τα παιδιά που μπορεί να είναι πολύ περήφανα που είναι 5,5 ετών και όχι μόνο 5.

3.4 Πράξεις και τελεστές

Εκτός από τον καθορισμό του είδους των τιμών που μπορούν να αποθηκευτούν σε μια μεταβλητή, ο τύπος μιας μεταβλητής καθορίζει και ποιες πράξεις μπορούν να γίνουν σ' αυτήν και τι σημαίνουν. Για παράδειγμα:

```
int count;
cin >> count;                  // το >> διαβάζει έναν ακέραιο στο count
string name;
cin >> name;                   // το >> διαβάζει μια συμβολοσειρά στο name

int c2 = count+2;              // το + προσθέτει ακεραίους
string s2 = name + " Jr. ";    // το + προσαρτά χαρακτήρες

int c3 = count–2;              // το – αφαιρεί ακεραίους
string s3 = name – "Jr. ";     // σφάλμα: το – δεν ορίζεται για συμβολοσειρές
```

Με τη λέξη "σφάλμα" εννοούμε ότι ο μεταγλωττιστής θα απορρίψει ένα πρόγραμμα που προσπαθεί να αφαιρέσει συμβολοσειρές. Ο μεταγλωττιστής γνωρίζει ακριβώς ποιες πράξεις μπορούν να γίνουν σε κάθε μεταβλητή, και έτσι μπορεί να αποτρέψει πολλά σφάλματα. Ωστόσο, ο μεταγλωττιστής δε γνωρίζει ποιες πράξεις έχουν νόημα για σας και για ποιες τιμές, οπότε με χαρά θα δεχτεί νομιμόφρονες πράξεις που δίνουν αποτελέσματα, τα οποία ωστόσο μπορεί να σας φαίνονται παράδοξα. Για παράδειγμα:

```
int age = –100;
```

Μπορεί σε σας να φαίνεται παράλογο να υπάρχει αρνητική ηλικία (γιατί όχι;), αλλά κανείς δεν το είπε στο μεταγλωττιστή, επομένως θα προχωρήσει στην παραγωγή κώδικα γι' αυτό τον ορισμό.

Ακολουθεί ένας πίνακας χρήσιμων τελεστών για μερικούς κοινούς και χρήσιμους τύπους:

	bool	char	int	double		string
εκχώρηση	=	=	=	=	=	=
πρόθεση			+	+		
αλληλουχία						+
αφαίρεση			–	–		
πολλαπλασιασμός			*	*		
διαίρεση			/	/		
υπόλοιπο (από ακέραια διαίρεση)			%			
αύξηση κατά 1			++	++		
μείωση κατά 1			– –	– –		
αύξηση κατά **n**			+= **n**	+= **n**		

	bool	char	int	double	string
πρόσθεση στο τέλος					+=
μείωση κατά n			-= n	-= n	
πολλαπλασιασμός και εκχώρηση			*=	*=	
διαίρεση και εκχώρηση			/=	/=	
υπόλοιπο και εκχώρηση			%=		
διάβασε από το s στο x	s >> x	s >> x	s >> x	s >> x	s >> x
γράψε x στο s	s << x	s << x	s << x	s << x	s << x
ίσο με	==	==	==	==	==
όχι ίσο	!=	!=	!=	!=	!=
μεγαλύτερο από	>	>	>	>	>
μεγαλύτερο από ή ίσο	>=	>=	>=	>=	>=
μικρότερο από	<	<	<	<	<
μικρότερο από ή ίσο	<=	<=	<=	<=	<=

Ένα κενό τετράγωνο υποδεικνύει μια πράξη που δεν είναι άμεσα διαθέσιμη για έναν τύπο (αν και μπορεί να υπάρχουν έμμεσοι τρόποι χρήσης αυτής της πράξης, όπως μπορείτε να δείτε στην §3.7). Θα εξηγήσουμε αυτές τις πράξεις και άλλες στην πορεία. Τα κύρια σημεία εδώ είναι ότι υπάρχουν πολλοί χρήσιμοι τελεστές και ότι το νόημά τους συνήθως είναι το ίδιο για παρόμοιους τύπους.

Ας δοκιμάσουμε ένα παράδειγμα όπου χρησιμοποιούνται αριθμοί κινητής υποδιαστολής:

```
// απλό πρόγραμμα για εξάσκηση στους τελεστές
int main()
{
    cout << "Παρακαλώ εισαγάγετε μια τιμή κινητής υποδιαστολής: ";
    double n;
    cin >> n;
    cout << "n == " << n
         << "\nn+1 == " << n+1
         << "\nthree times n == " << 3*n
         << "\ntwice n == " << n+n
         << "\nn squared == " << n*n
         << "\nhalf of n == " << n/2
         << "\nsquare root of n == " << sqrt(n)
         << endl;          // άλλο όνομα για νέα γραμμή ("end of line")
}
```

Προφανώς, οι συνήθεις αριθμητικές πράξεις έχουν το συνήθη τρόπο γραφής τους και το σύνηθες νόημά τους, όπως τους ξέρουμε από το δημοτικό. Όπως είναι φυσικό, δεν είναι διαθέσιμα όλα όσα θα θέλαμε να κάνουμε με έναν αριθμό κινητής υποδια-

στολής, όπως η εύρεση της τετραγωνικής ρίζας του, με τη μορφή τελεστή. Πολλές πράξεις αναπαρίστανται ως συναρτήσεις με όνομα. Σ' αυτή την περίπτωση, χρησιμοποιούμε την **sqrt()** από τη βιβλιοθήκη προτύπων προκειμένου να πάρουμε την τετραγωνική ρίζα του **n**: **sqrt(n)**. Ο τρόπος γραφής είναι οικείος από τα μαθηματικά. Θα χρησιμοποιήσουμε συναρτήσεις σε όλο το βιβλίο και θα τις εξετάσουμε σε βάθος στις §4.5 και §8.5.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Τρέξτε αυτό το μικρό πρόγραμμα. Κατόπιν, τροποποιήστε το, ώστε να διαβάζει έναν ακέραιο **int** αντί για ένα **double**. Παρατηρήστε ότι η **sqrt()** δεν ορίζεται για ένα **int**, επομένως πρέπει να εκχωρήσετε το **n** σε ένα **double** και να πάρετε την **sqrt()** αυτού. Επίσης, "εκτελέστε" μερικές ακόμα πράξεις. Παρατηρήστε ότι για τα **int**, το / είναι ακέραια διαίρεση και το **%** είναι το υπόλοιπο (ακέραια διαίρεση), ώστε το **5/2** δίνει **2** (και όχι **2.5** ή **3**) και το **5%2** δίνει **1**. Οι ορισμοί των ακέραιων πράξεων *, / και **%** εγγυώνται ότι για δύο θετικά **int**, **a** και **b**, παίρνουμε **a/b * b + a%b == a**.

Οι συμβολοσειρές έχουν λιγότερους τελεστές, αλλά, όπως θα δούμε στο Κεφάλαιο 23, έχουν πολλές πράξεις με όνομα. Ωστόσο, οι τελεστές που έχουν μπορούν να χρησιμοποιηθούν με συμβατικό τρόπο. Για παράδειγμα:

```
// διάβασε πρώτο και δεύτερο μικρό όνομα
int main()
{
    cout << "Παρακαλώ εισαγάγετε το πρώτο και το δεύτερο μικρό όνομά σας\n";
    string first;
    string second;
    cin >> first >> second;              // διάβασε δύο συμβολοσειρές
    string name = first + ' ' + second;  // σύνδεσε αλυσιδωτά τις δύο συμβολοσειρές
    cout << "Γεια σου " << name << '\n';
}
```

Για τις συμβολοσειρές, το + σημαίνει αλληλουχία ή αλυσιδωτή σύνδεση, που σημαίνει ότι όταν τα **s1** και **s2** είναι συμβολοσειρές, το **s1+s2** είναι μια συμβολοσειρά όπου οι χαρακτήρες από το **s1** ακολουθούνται από τους χαρακτήρες του **s2**. Για παράδειγμα, εάν το **s1** έχει τιμή **"Γεια σου"** και το **s2** έχει τιμή **"κόσμε"**, τότε το **s1+s2** θα έχει τιμή **"Γεια σου κόσμε"**. Η σύγκριση μεταξύ **string**s είναι ιδιαίτερα χρήσιμη:

```
// διάβασε και σύγκρινε ονόματα
int main()
{
    cout << "Παρακαλώ εισαγάγετε δύο ονόματα\n";
    string first;
    string second;
    cin >> first >> second;     // διάβασε δύο συμβολοσειρές
    if (first == second) cout << "αυτό είναι το ίδιο όνομα δύο φορές\n";
```

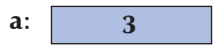
```
    if (first < second)
        cout << first << " είναι, αλφαβητικά, πριν το " << second <<'\n';
    if (first > second)
        cout << first << " είναι, αλφαβητικά, μετά το " << second <<'\n';
}
```

Εδώ, χρησιμοποιήσαμε μια εντολή **if**, την οποία θα εξηγήσουμε αναλυτικά στην §4.4.1.1, προκειμένου να επιλέξουμε ενέργειες που βασίζονται σε συνθήκες.

3.5 Εκχώρηση και αρχικοποίηση

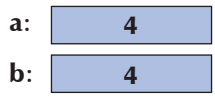
Από πολλές απόψεις, ο πιο ενδιαφέρων τελεστής είναι η εκχώρηση, που αναπαρίσταται ως **=**. Δίνει σε μια μεταβλητή μια νέα τιμή. Για παράδειγμα:

```
int a = 3;        // το a ξεκινάει με την τιμή 3
```

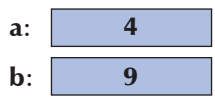



```
a = 4; // το a παίρνει την τιμή 4 ("γίνεται 4")
```

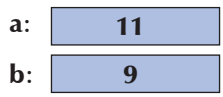

```
int b = a;        // το b ξεκινάει με ένα αντίγραφο της τιμής του a (δηλαδή 4)
```




```
b = a+5;          // το b παίρνει την τιμή του a+5 (δηλαδή 9)
```




```
a = a+7;          // το a παίρνει την τιμή του a+7 (δηλαδή 11)
```



Η τελευταία πράξη εκχώρησης αξίζει περισσότερη προσοχή. Πρώτον, δείχνει καθαρά ότι το **=** δε σημαίνει "ίσον" – είναι σαφές ότι το **a** δεν ισούται με **a+7**. Σημαίνει εκχώρηση, δηλαδή τοποθέτηση μιας νέας τιμής σε μια μεταβλητή. Αυτό που γίνεται με το **a=a+7** είναι το εξής:

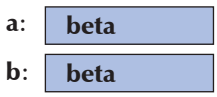
1. Πρώτα, πάρε την τιμή του **a** – ο ακέραιος 4.
2. Κατόπιν, πρόσθεσε 7 σ' αυτό το 4 και πάρε το αποτέλεσμα – τον ακέραιο 11.
3. Τέλος, τοποθέτησε αυτό το 11 στο **a**.

Μπορούμε επίσης να παρουσιάσουμε την πράξη εκχώρησης χρησιμοποιώντας συμβολοσειρές:

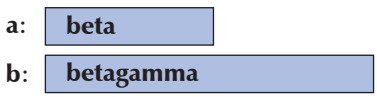
```
string a = "alpha";       // το a ξεκινάει με την τιμή "alpha"
```


```
a = "beta";     // το a παίρνει την τιμή "beta" (γίνεται "beta")
```

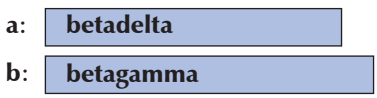

```
string b = a;  // το b ξεκινάει με ένα αντίγραφο της τιμής του a (δηλαδή "beta")
```




```
b = a+"gamma";        // το b παίρνει την τιμή a+"gamma" (δηλαδή "betagamma")
```




```
a = a+"delta"; // το a παίρνει την τιμή a+"delta" (δηλαδή "betadelta")
```



Εδώ, χρησιμοποιούμε τις φράσεις "ξεκινάει με" και "παίρνει" προκειμένου να διαχωρίσουμε τις δύο παρόμοιες, αλλά λογικά διακριτές, πράξεις:

- Αρχικοποίηση (δίνει σε μια μεταβλητή την αρχική τιμή της)
- Εκχώρηση (δίνει σε μια μεταβλητή μια νέα τιμή)

Αυτές οι πράξεις είναι τόσο παρόμοιες που η C++ μας επιτρέπει να χρησιμοποιήσουμε τον ίδιο τρόπο γραφής (το =) και για τις δύο:

```
int y = 8;              // αρχικοποίησε το y με το 8
x = 9;                  // εκχώρησε το 9 στο x

string t = "howdy!";    // αρχικοποίησε με το "howdy!"
s = "G'day";            // εκχώρησε το "G'day" στο s
```

Ωστόσο, λογικά, η εκχώρηση και η αρχικοποίηση είναι διαφορετικές. Μπορείτε να ξεχωρίσετε τις δύο ενέργειες από τον προσδιορισμό του τύπου (όπως **int** ή **string**) που ξεκινά πάντα με μια αρχικοποίηση, ενώ η εκχώρηση δεν το κάνει αυτό. Κατ' αρχάς, μια αρχικοποίηση βρίσκει πάντα τη μεταβλητή κενή. Από την άλλη πλευρά, μια εκχώρηση (κατ' αρχάς) πρέπει να διαγράψει την παλιά τιμή από τη μεταβλητή

πριν τοποθετήσει τη νέα τιμή. Μπορείτε να φανταστείτε μια μεταβλητή ως ένα είδος μικρού κουτιού και την τιμή ως κάτι στέρεο, όπως ένα κέρμα, το οποίο τοποθετείτε μέσα στο κουτί. Πριν την αρχικοποίηση, το κουτί είναι άδειο, αλλά μετά την αρχικοποίηση έχει πάντα μέσα του ένα κέρμα, και για να τοποθετήσετε μέσα ένα νέο κέρμα (με τον τελεστή εκχώρησης), πρέπει πρώτα να αφαιρέσετε το παλιό ("κατέστρεψε την παλιά τιμή") — και δεν μπορείτε να αφήσετε το κουτί άδειο. Τα πράγματα δεν είναι τόσο κυριολεκτικά στη μνήμη του υπολογιστή, αλλά κάπως έτσι συμβαίνει τα πράγματα.

3.5.1 Ένα παράδειγμα: διαγραφή επαναλαμβανόμενων λέξεων

Η εκχώρηση είναι απαραίτητη όταν θέλουμε να τοποθετήσουμε μια νέα τιμή σε ένα αντικείμενο. Όταν το σκεφτείτε, είναι προφανές ότι η εκχώρηση είναι ιδιαίτερα χρήσιμη όταν κάνετε κάτι πολλές φορές. Χρειαζόμαστε μια εκχώρηση όταν θέλουμε να κάνουμε κάτι ξανά, με διαφορετική τιμή. Ας δούμε το παρακάτω μικρό πρόγραμμα, το οποίο εντοπίζει γειτονικές επαναλαμβανόμενες λέξεις σε μια ακολουθία λέξεων. Αυτός ο κώδικας εμπεριέχεται στα περισσότερα προγράμματα γραμματικού ελέγχου:

```
int main()
{
    string previous = " ";          // προηγούμενη λέξη, αρχικοποιημένη με κενό
    string current;                 // τρέχουσα λέξη
    while (cin>>current) {          // διάβασε μια ροή από λέξεις
        if (previous == current)    // έλεγξε αν η λέξη είναι η ίδια με την προηγούμενη
            cout << "επαναλαμβανόμενη λέξη: " << current << '\n';
        previous = current;
    }
}
```

Αυτό το πρόγραμμα δεν είναι πολύ χρήσιμο, επειδή δε μας λέει πού εμφανίστηκε η επαναλαμβανόμενη λέξη στο κείμενο, αλλά προς το παρόν είναι αρκετό. Θα εξετάσουμε αυτό το πρόγραμμα γραμμή προς γραμμή, ξεκινώντας από τη γραμμή

```
string current;          // τρέχουσα λέξη
```

Αυτή είναι η μεταβλητή συμβολοσειράς στην οποία διαβάζουμε άμεσα την τρέχουσα (δηλαδή αυτήν που διαβάσαμε πιο πρόσφατα) λέξη, χρησιμοποιώντας την εντολή

```
while (cin>>current)
```

Αυτή η δομή, η οποία ονομάζεται εντολή **while**, είναι από μόνη της ενδιαφέρουσα και θα μας απασχολήσει στην §4.4.2.1. Το **while** λέει ότι η εντολή after **(cin>>current)** πρέπει να εκτελείται επανειλημμένα για όσο ικανοποιείται η πράξη εισόδου **cin>>current**, και η **cin>>current** θα ικανοποιείται για όσο υπάρχουν χαρακτήρες που θα διαβάζονται στην κανονική είσοδο. Θυμηθείτε ότι για ένα **string**, το **>>** διαβάζει λέξεις

που χωρίζονται από κενά. Μπορείτε να τερματίσετε αυτόν το βρόγχο δίνοντας στο πρόγραμμα ένα χαρακτήρα τέλους εισόδου (συνήθως ονομάζεται *end of file*, τέλος αρχείου). Σε έναν υπολογιστή με Windows, αυτός ο χαρακτήρας είναι ο συνδυασμός Ctrl+Z (ταυτόχρονο πάτημα των πλήκτρων Control και Z) που ακολουθείται από το πάτημα του Enter. Σε έναν υπολογιστή με Unix ή Linux, αυτός ο χαρακτήρας είναι ο συνδυασμός Ctrl+D (ταυτόχρονο πάτημα των πλήκτρων Control και D).

Επομένως, αυτό που κάνουμε είναι να διαβάσουμε μια λέξη στο **current** και κατόπιν να τη συγκρίνουμε με την προηγούμενη λέξη (που αποθηκεύεται στο **previous**). Εάν είναι ίδιες, το λέμε:

```
if (previous == current)          // έλεγξε αν η λέξη είναι η ίδια με την προηγούμενη
cout << "επαναλαμβανόμενη λέξη: " << current << '\n';
```

Στη συνέχεια πρέπει να προετοιμαστούμε, ώστε να κάνουμε το ίδιο για την επόμενη λέξη. Θα το κάνουμε αντιγράφοντας την τρέχουσα λέξη **current** στην προηγούμενη λέξη **previous**:

```
previous = current;
```

Με την εντολή αυτή αντιμετωπίζονται όλες οι περιπτώσεις, εφόσον μπορούμε να ξεκινήσουμε. Τι θα πρέπει να κάνει αυτός ο κώδικας στην πρώτη λέξη όταν δεν υπάρχουν προηγούμενες λέξεις με τις οποίες θα γίνει η σύγκριση; Αυτό το πρόβλημα αντιμετωπίζεται με τον ορισμό του **previous**:

```
string previous = " ";     // προηγούμενη λέξη, αρχικοποιημένη με κενό
```

Το " " περιέχει ένα χαρακτήρα (το χαρακτήρα κενού διαστήματος). Ο τελεστής εισόδου >> παραλείπει τον κενό χώρο, που σημαίνει ότι δε θα μπορούσαμε να τον διαβάσουμε από την είσοδο. Κατά συνέπεια, την πρώτη φορά που εκτελείται η εντολή **while**, η δοκιμή

```
if (previous == current)
```

αποτυγχάνει (όπως θέλαμε).

Ένας τρόπος να καταλάβουμε τη ροή του προγράμματος είναι να "το παίξουμε υπολογιστής", δηλαδή να ακολουθήσουμε το πρόγραμμα γραμμή προς γραμμή, κάνοντας ό,τι λέει. Απλά σχεδιάστε κουτιά σε ένα φύλλο χαρτί και γράψτε τις τιμές τους μέσα στα κουτιά. Αλλάξτε τις τιμές που αποθηκεύονται σύμφωνα με το πρόγραμμα.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Εκτελέστε αυτό το πρόγραμμα μόνοι σας, χρησιμοποιώντας ένα φύλλο χαρτί. Χρησιμοποιήστε την είσοδο "**Η γάτα γάτα πήδηξε**". Ακόμα και πεπειραμένοι προγραμματιστές χρησιμοποιούν αυτή την τεχνική, ώστε να δουν στην πράξη τις ενέργειες μικρών αποσπασμάτων κώδικα, οι οποίες, με κάποιον τρόπο, δεν είναι εντελώς προφανείς.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Τρέξτε το "πρόγραμμα εντοπισμού επαναλαμβανόμενων λέξεων". Δοκιμάστε το με την πρόταση "**Αυτή Αυτή γέλασε επειδή Αυτός Αυτός Αυτός δε φαινόταν πολύ πολύ καλά καλά**". Πόσες επαναλαμβανόμενες λέξεις υπάρχουν; Γιατί; Ποιος είναι ο ορισμός της *λέξης* που χρησιμοποιούμε εδώ; Ποιος είναι ο ορισμός της *επαναλαμβανόμενης λέξης*; (Για παράδειγμα, το "**Αυτή Αυτή**" είναι επανάληψη;)

3.6 Σύνθετοι τελεστές εκχώρησης

Η αύξηση μιας μεταβλητής (δηλαδή η προσθήκη του 1 σ' αυτήν) είναι τόσο κοινή σε προγράμματα, ώστε η C++ παρέχει ειδικό συντακτικό γι' αυτήν. Για παράδειγμα, το:

```
++counter
```

σημαίνει

```
counter = counter + 1
```

Υπάρχουν και άλλοι, πολλοί και συνηθισμένοι τρόποι να αλλάξουμε την τιμή μιας μεταβλητής βασιζόμενοι στην τρέχουσα τιμή της. Για παράδειγμα, μπορεί να θέλουμε να προσθέσουμε το 7 σ' αυτήν, να αφαιρέσουμε το 9 απ' αυτήν ή να την πολλαπλασιάσουμε επί 2. Τέτοιες πράξεις υποστηρίζονται επίσης ευθέως από τη C++. Για παράδειγμα:

```
a += 7;      // σημαίνει a = a+7
b -= 9;      // σημαίνει b = b–9
c *= 2;      // σημαίνει c = c*2
```

Γενικά, για ένα δυαδικό τελεστή **oper**, το **a oper= b** σημαίνει **a = a oper b** (§A.5). Για αρχή, αυτός ο κανόνας μας δίνει τους τελεστές **+=**, **-=**, ***=**, **/=** και **%=**. Έχουμε έτσι έναν ευτυχώς συμπυκνωμένο τρόπο γραφής, ο οποίος εκφράζει άμεσα τις ιδέες μας. Για παράδειγμα, σε πολλές εφαρμογές, τα **/=** και **%=** ονομάζονται "κλιμάκωση".

3.6.1 Ένα παράδειγμα: μέτρημα επαναλαμβανόμενων λέξεων

Θα χρησιμοποιήσουμε το πιο πάνω παράδειγμα που εντοπίζει τις επαναλαμβανόμενες γειτονικές λέξεις. Θα μπορούσαμε να το βελτιώσουμε, δίνοντας μια ιδέα για το πού μέσα στην ακολουθία βρίσκεται η επαναλαμβανόμενη λέξη. Μια απλή παραλλαγή αυτής της ιδέας μετράει τις λέξεις και εξάγει το μετρητή για την επαναλαμβανόμενη λέξη:

```
int main()
{
```

```
    int number_of_words = 0;
    string previous = " ";          // όχι λέξη
    string current;
    while (cin>>current) {
        ++number_of_words;     // αύξησε το μετρητή λέξεων
        if (previous == current)
            cout << "αριθμός λέξεων " << number_of_words
                 << " που επαναλαμβάνονται: "<< current << '\n';
        previous = current;
    }
}
```

Ξεκινάμε με το μετρητή λέξεων να έχει τιμή 0. Κάθε φορά που βλέπουμε μια λέξη, αυξάνουμε αυτόν το μετρητή:

```
++number_of_words;
```

Με αυτό τον τρόπο, η πρώτη λέξη γίνεται ο αριθμός 1, η επόμενη γίνεται ο αριθμός 2 και ούτω καθεξής. Θα μπορούσαμε να πετύχουμε το ίδιο με την εντολή

```
number_of_words += 1;
```

ή ακόμα και με την εντολή

```
number_of_words = number_of_words+1;
```

αλλά το **++number_of_words** είναι πιο σύντομο και εκφράζει την έννοια της αύξησης άμεσα.

Παρατηρήστε πόσο αυτό το πρόγραμμα μοιάζει με αυτό στην §3.5.1. Προφανώς, απλά πήραμε το πρόγραμμα από την §3.5.1 και το τροποποιήσαμε λίγο, ώστε να εξυπηρετήσει το νέο σκοπό μας. Πρόκειται για μια ιδιαίτερα κοινή τεχνική: Όταν πρέπει να λύσουμε ένα πρόβλημα, ψάχνουμε για ένα παρόμοιο πρόβλημα και χρησιμοποιούμε τη λύση γι' αυτό, με κατάλληλη τροποποίηση. Μην ξεκινάτε από την αρχή, εκτός αν πραγματικά πρέπει να το κάνετε. Η χρήση μιας προηγούμενης έκδοσης ενός προγράμματος ως βάση για τροποποίηση συχνά είναι σημαντική εξοικονόμηση χρόνου και αποσβένουμε μέρος της προσπάθειας που κάναμε για το αρχικό πρόγραμμα.

3.7 Ονόματα

Ονομάζουμε τις μεταβλητές μας, ώστε να μπορούμε να τις θυμόμαστε και να παραπέμπουμε σ' αυτές από διάφορα μέρη ενός προγράμματος. Τι μπορεί να γίνει όνομα στη C++; Σε ένα πρόγραμμα C++, ένα όνομα ξεκινά με ένα γράμμα και περιέχει μόνο γράμματα, ψηφία και χαρακτήρες underscore (κάτω παύλα). Για παράδειγμα:

```
x
number_of_elements
Fourier_transform
z2
Polygon
```

Τα παρακάτω δεν είναι ονόματα:

```
2x                  // ένα όνομα πρέπει να ξεκινάει με ένα γράμμα
time$to$market      // το $ δεν είναι γράμμα, ψηφίο ή underscore
Start menu          // το κενό διάστημα δεν είναι γράμμα, ψηφίο ή underscore
```

Όταν λέμε "όχι ονόματα", εννοούμε ότι ένας μεταγλωττιστής C++ δε θα τα δεχτεί ως ονόματα.

Εάν διαβάζετε κώδικα συστήματος ή κώδικα που παράγεται από μηχανή, μπορείτε να δείτε ονόματα που ξεκινούν με underscore, όπως το **_foo**. Δε θα πρέπει ωστόσο να τα γράφετε μόνοι σας έτσι, επειδή τέτοια ονόματα δεσμεύονται για οντότητες υλοποίησης και συστήματος. Εάν αποφεύγετε τα αρχικά underscore, ποτέ δε θα διαπιστώσετε σύγκρουση των ονομάτων σας με κάποιο όνομα που παρήγαγε η υλοποίηση.

Τα ονόματα διακρίνουν τους πεζούς από τους κεφαλαίους χαρακτήρες, που σημαίνει ότι τα **x** και **X**, για παράδειγμα, είναι διαφορετικά ονόματα. Αυτό το μικρό πρόγραμμα έχει τουλάχιστον τέσσερα σφάλματα:

```
#include "std_lib_facilities.h"

int Main()
{
    String s = "Αντίο σκληρέ κόσμε! ";
    cOut << S << '\n';
}
```

Συνήθως δεν είναι καλή ιδέα να ορίζουμε ονόματα που διαφέρουν μόνο σε ένα χαρακτήρα που τη μια φορά είναι πεζός και την άλλη κεφαλαίος, όπως στο **one** και **One**. Κάτι τέτοιο δε θα προκαλέσει σύγχυση στο μεταγλωττιστή, αλλά θα το κάνει εύκολα σε έναν προγραμματιστή.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Μεταγλωττίστε το πρόγραμμα "Αντίο σκληρέ κόσμε!" και εξετάστε τα μηνύματα σφάλματος. Ο μεταγλωττιστής βρήκε όλα τα σφάλματα; Τι πρόκρινε ως προβλήματα; Ο μεταγλωττιστής μπερδεύτηκε και διέγνωσε περισσότερα από τέσσερα σφάλματα; Αφαιρέστε τα σφάλματα ένα προς ένα, ξεκινώντας πρώτα με τα λεξικολογικά και δείτε πώς αλλάζουν (και βελτιώνονται) τα μηνύματα σφάλματος.

Η γλώσσα C++ δεσμεύει αρκετά (περίπου 70) ονόματα ως "keywords". Τα αναφέρουμε όλα στην §A.3.1. Δεν μπορείτε να τα χρησιμοποιήσετε για να ονομάσετε δικές σας μεταβλητές, τύπους, συναρτήσεις και άλλα. Για παράδειγμα:

```
int if = 7;      // σφάλμα: το "if" είναι keyword
```

Μπορείτε να χρησιμοποιήσετε ονόματα υπηρεσιών της βιβλιοθήκης προτύπων, όπως το **string**, αλλά δε θα πρέπει να το κάνετε. Η χρήση ενός τόσο κοινού ονόματος θα προκαλέσει προβλήματα, αν ποτέ προκύψει η ανάγκη να χρησιμοποιήσετε τη βιβλιοθήκη προτύπων:

```
int string = 7;  // αυτό θα οδηγήσει σε προβλήματα
```

όταν επιλέγετε ονόματα για μεταβλητές, συναρτήσεις, τύπους και άλλα, επιλέξτε ονόματα που σημαίνουν κάτι, δηλαδή ονόματα που θα βοηθήσουν άλλους να κατανοήσουν το πρόγραμμά σας. Ακόμα και εσείς θα έχετε προβλήματα να καταλάβετε το πρόγραμμά σας και τι υποτίθεται ότι κάνει, αν το έχετε γεμίσει με μεταβλητές που έχουν ονόματα που μπορείτε να πληκτρολογήσετε γρήγορα, όπως **x1**, **x2**, **s3** και **p7**. Οι συντομογραφίες και τα ακρωνύμια συνήθως προκαλούν σύγχυση και γι' αυτό συνιστούμε να τα χρησιμοποιείτε με φειδώ. Αυτά τα ακρωνύμια ήταν προφανή σε μας όταν τα γράψαμε, αλλά είμαστε βέβαιοι ότι σε σας θα προκαλέσουν πολλές απορίες:

mtbf
TLA
myw
NBV

Σε μερικούς μήνες μάλιστα είναι βέβαιο ότι και εμείς θα έχουμε απορίες.

Τα σύντομα ονόματα, όπως **x** και **i**, έχουν νόημα μόνο όταν χρησιμοποιούνται συμβατικά, δηλαδή το **x** θα πρέπει να είναι μια τοπική μεταβλητή ή παράμετρος (ανατρέξτε στις §4.5 και §8.4) και το **i** θα πρέπει να είναι ένας μετρητής βρόγχου (ανατρέξτε στην §4.4.2.3).

Μη χρησιμοποιείτε υπερβολικά μεγάλα ονόματα, καθώς δεν είναι εύκολη η πληκτρολόγησή τους, μεγαλώνουν τις γραμμές τόσο, ώστε δε χωρούν στην οθόνη και δεν μπορούν να διαβαστούν γρήγορα. Τα παρακάτω είναι λογικά και ικανοποιητικά:

partial_sum
element_count
stable_partition

Ενώ αυτά είναι μάλλον μεγάλα:

the_number_of_elements
remaining_free_slots_in_symbol_table

Το δικό μας στιλ είναι να χρησιμοποιούμε χαρακτήρες underscore για να διαχωρίζουμε λέξεις σε μια μεταβλητή, όπως **element_count**, αντί για εναλλακτικές λύσεις, όπως **elementCount** και **Element-Count**. Δε χρησιμοποιούμε ποτέ ονόματα μόνο με κεφαλαία γράμματα, όπως **ALL_CAPITAL_LETTERS**, επειδή εκ συμβάσεως, αυτό το στιλ χρησιμοποιείται για μακροεντολές (§27.8 και §A.17.2), τις οποίες αποφεύγουμε. Χρησιμοποιούμε ένα αρχικό κεφαλαίο γράμμα για τύπους που ορίζουμε εμείς, όπως **Square** και **Graph**. Η γλώσσα και η βιβλιοθήκη προτύπων C++ δε χρησιμοποιούν κεφαλαία

γράμματα, γι' αυτό και βλέπετε **int** αντί για **Int** και **string** αντί για **String**. Έτσι, η σύμβασή μας μας βοηθάει να ελαχιστοποιούμε τη σύγχυση μεταξύ των δικών μας τύπων και των προτύπων της γλώσσας.

Αποφύγετε επίσης ονόματα που μπορούν να γραφτούν και να διαβαστούν με λάθος τρόπο ή να προκαλέσουν σύγχυση πολύ εύκολα. Για παράδειγμα:

Name	**names**	**nameS**
foo	**f00**	**fl**
f1	**fI**	**fi**

Οι χαρακτήρες **0**, **o**, **O**, **1**, **l**, **I** είναι ιδιαίτερα ευάλωτοι σε σφάλματα.

3.8 Τύποι και αντικείμενα

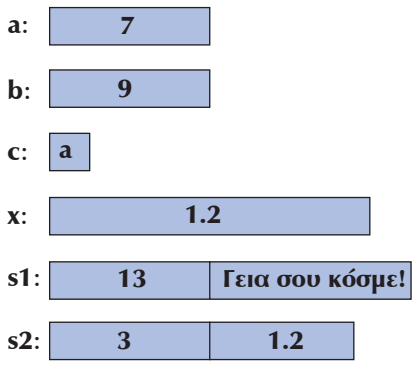
Η έννοια τύπου είναι πολύ σημαντική στη C++ και στις περισσότερες γλώσσες προγραμματισμού.

Ας μελετήσουμε τώρα πιο αναλυτικά και πιο τεχνικά τους τύπους, συγκεκριμένα τους τύπους των αντικειμένων στους οποίους αποθηκεύουμε δεδομένα ενώ κάνουμε υπολογισμούς. Θα σας βοηθήσει μακροπρόθεσμα, ως προς το χρόνο και ως προς την κατανόηση.

- Ένας *τύπος* ορίζει ένα σύνολο από πιθανές τιμές και ένα σύνολο πράξεων (για ένα αντικείμενο).
- Ένα *αντικείμενο* είναι κάποια μνήμη που διατηρεί μια τιμή ενός δεδομένου τύπου.
- Μια *τιμή* είναι ένα σύνολο από bits στη μνήμη, τα οποία ερμηνεύονται σύμφωνα με έναν τύπο.
- Μια *μεταβλητή* είναι ένα αντικείμενο με όνομα.
- Μια *δήλωση* είναι μια εντολή που δίνει όνομα σε ένα αντικείμενο.
- Ένας *ορισμός* είναι μια δήλωση που δεσμεύει μνήμη για ένα αντικείμενο.

Πληροφοριακά, θεωρούμε ένα αντικείμενο ως ένα κουτί στο οποίο μπορούμε να τοποθετήσουμε τιμές συγκεκριμένου τύπου. Ένα κουτί **int** μπορεί να διατηρεί ακέραιους, όπως **7**, **42** και **–399**. Ένα κουτί **string** μπορεί να διατηρεί τιμές συμβολοσειρών χαρακτήρων, όπως **"Interoperability"**, **"tokens: !@#$%·&*"** και **"Old McDonald had a farm"**. Γραφικά, μπορούμε να το σκεφτούμε ως εξής:

```
int a = 7;
int b = 9;
char c = 'a';
double x = 1.2;
string s1 = "Γεια σου κόσμε!";
string s2 = "1.2";
```



Η αναπαράσταση ενός **string** είναι λίγο περισσότερο περίπλοκη από την αναπαράσταση ενός **int**, επειδή ένα **string** καταγράφει τον αριθμό των χαρακτήρων που διατηρεί. Σημειώστε ότι ένα **double** αποθηκεύει έναν αριθμό, ενώ ένα **string** αποθηκεύει χαρακτήρες. Για παράδειγμα, το **x** αποθηκεύει τον αριθμό **1.2**, ενώ το **s2** τους τρεις χαρακτήρες **'1'**, **'.'** και **'2'**. Τα εισαγωγικά για τις τιμές χαρακτήρων και συμβολοσειρών δεν αποθηκεύονται.

Κάθε **int** έχει το ίδιο μέγεθος, δηλαδή ο μεταγλωττιστής δεσμεύει την ίδια σταθερή ποσότητα μνήμης για κάθε **int**. Σε ένα συνηθισμένο υπολογιστή, αυτή η ποσότητα είναι 4 bytes (32 bits). Ομοίως, τα **bool**, **char** και **double** έχουν σταθερό μέγεθος. Θα διαπιστώσετε ότι ένας υπολογιστής χρησιμοποιεί ένα byte (8 bits) για ένα **bool** ή ένα **char** και 8 bytes για ένα **double**. Σημειώστε ότι διαφορετικοί τύποι αντικειμένων καταλαμβάνουν διαφορετικές ποσότητες χώρου. Συγκεκριμένα, ένα **char** καταλαμβάνει λιγότερο χώρο από ένα **int** και το **string** διαφέρει από τα **double**, **int** και **char** υπό την έννοια ότι διαφορετικές συμβολοσειρές καταλαμβάνουν διαφορετικές ποσότητες χώρου.

Η σημασία των bits στη μνήμη εξαρτάται απόλυτα από τον τύπο που χρησιμοποιείται για την πρόσβασή της. Μπορείτε να το φανταστείτε ως εξής: Η μνήμη των υπολογιστών δε γνωρίζει για τύπους – είναι απλά μνήμη. Τα bits μνήμης αποκτούν νόημα μόνο όταν αποφασίσουμε πώς θα πρέπει να ερμηνεύσουμε τη μνήμη. Αυτό μοιάζει κάπως με ό,τι κάνουμε κάθε μέρα όταν χρησιμοποιούμε αριθμούς. Τι σημαίνει **12.5**; Δε γνωρίζουμε. Θα μπορούσε να είναι **$12.5** ή **12.5cm** ή **12.5gallons**. Μόνο όταν δίνουμε τη μονάδα, ο αριθμός **12.5** αποκτά νόημα.

Για παράδειγμα, τα ίδια bits μνήμης που αναπαριστούν την τιμή **120**, όταν τα θεωρήσουμε ως **int**, θα είναι **'x'** όταν τα θεωρήσουμε ως **char**. Εάν τα θεωρήσουμε ως **string**, δε θα είχε κανένα νόημα και θα παρουσιαζόταν ένα σφάλμα εκτέλεσης αν επιχειρούσαμε να το χρησιμοποιήσουμε έτσι. Μπορούμε να παρουσιάσουμε αυτή την έννοια με γραφικό τρόπο, χρησιμοποιώντας 1 και 0 για να δείξουμε την τιμή των bits στη μνήμη:

```
00000000 00000000 00000000 01111000
```

Αυτή είναι η ρύθμιση των bits μιας περιοχής μνήμης (μια λέξη) που θα μπορούσε να διαβαστεί ως **int** (**120**) ή ως **char** (**'x'**, αν δούμε τα 8 πιο δεξιά bits μόνο). Ένα **bit** είναι μια μονάδα μνήμη υπολογιστή που θα μπορούσε να διατηρήσει την τιμή 0 ή 1. Για την ερμηνεία των *δυαδικών* αριθμών, ανατρέξτε στην §A.2.1.1.

3.9 Ασφάλεια τύπων

Κάθε αντικείμενο λαμβάνει έναν τύπο όταν γίνεται ο ορισμός του. Ένα πρόγραμμα —ή μέρος ενός προγράμματος— είναι ασφαλές ως προς τους τύπους, όταν τα αντικείμενα χρησιμοποιούνται σύμφωνα με τους κανόνες που υπάρχουν για τον τύπο τους. Δυστυχώς, υπάρχουν τρόποι να γίνουν πράξεις που δεν είναι ασφαλείς ως προς τους τύπους. Για παράδειγμα, η χρήση μιας μεταβλητής πριν την αρχικοποίησή της δε θεωρείται ασφαλής χρήση τύπων:

```
int main()
{
    double x;          // "ξεχάσαμε" να αρχικοποιήσουμε:
                       // η τιμή του x δεν έχει οριστεί
```

```
    double y = x;          // η τιμή του y δεν έχει οριστεί
    double z = 2.0+x;      // το νόημα του + και η τιμή του z δεν έχουν οριστεί
}
```

Μια υλοποίηση επιτρέπεται ακόμα και για να δώσει ένα σφάλμα υλικού όταν χρησιμοποιείται το μη αρχικοποιημένο **x**. Θα πρέπει να αρχικοποιείτε τις μεταβλητές σας πάντα! Υπάρχουν μερικές —πολύ λίγες— εξαιρέσεις σ' αυτό τον κανόνα, όπως μια μεταβλητή που χρησιμοποιούμε άμεσα ως στόχο μιας πράξης εισόδου, αλλά θα πρέπει να συνηθίσετε από τώρα να αρχικοποιείτε για να γλιτώσετε από προβλήματα.

Η πλήρης ασφάλεια τύπων είναι το ιδανικό και, επομένως, είναι ο γενικός κανόνας για τη γλώσσα. Δυστυχώς, ένας μεταγλωττιστής C++ δεν μπορεί να εγγυηθεί την πλήρη ασφάλεια τύπων, αλλά μπορούμε να αποφύγουμε παραβιάσεις ασφάλειας τύπων μέσω ενός συνδυασμού καλής πρακτικής κωδικοποίησης και ελέγχων εκτέλεσης. Το ιδανικό είναι να μη χρησιμοποιούμε ποτέ χαρακτηριστικά γλώσσας που ο μεταγλωττιστής δεν μπορεί να αποδείξει ότι είναι ασφαλή: στατική ασφάλεια τύπων. Δυστυχώς, αυτό δημιουργεί πολλούς περιορισμούς για τις πιο ενδιαφέρουσες χρήσεις του προγραμματισμού. Η προφανής λύση ο μεταγλωττιστής να παράγει με δυναμικό τρόπο κώδικα που κάνει έλεγχο για παραβιάσεις της ασφάλειας τύπων και τις εντοπίζει όλες, δε γίνεται στη C++. Όταν αποφασίσουμε να κάνουμε πράγματα που είναι επισφαλή (ως προς τους τύπους), πρέπει να κάνουμε τον έλεγχο μόνοι μας. Θα τονίσουμε τέτοιες καταστάσεις στην πορεία.

Το ιδανικό της ασφάλειας τύπων είναι απίστευτα σημαντικό όταν γράφουμε κώδικα. Γι' αυτό και ασχολούμαστε με αυτό το θέμα στην αρχή του βιβλίου. Προσέξτε τις παγίδες και αποφύγετέ τες.

3.9.1 Ασφαλείς μετατροπές

Στην §3.4 είδαμε ότι δε θα μπορούσαμε να προσθέσουμε απευθείας **char** ή να συγκρίνουμε ένα **double** με ένα **int**. Ωστόσο, η C++ παρέχει έναν έμμεσο τρόπο να το κάνουμε. Όταν χρειάζεται, ένα **char** μετατρέπεται σε ένα **int** και ένα **int** μετατρέπεται σε ένα **double**. Για παράδειγμα:

```
char c = 'x';
int i1 = c;
int i2 = 'x';
```

Εδώ αμφότερα τα **i1** και **i2** λαμβάνουν την τιμή **120**, η οποία είναι η ακέραια τιμή του χαρακτήρα **'x'** στο πιο δημοφιλές σύνολο χαρακτήρων των 8 bits, ASCII. Αυτός είναι ένας απλός και ασφαλής τρόπος να πάρουμε την αριθμητική αναπαράσταση ενός χαρακτήρα. Ονομάζουμε αυτή τη μετατροπή από **char** σε **int** ασφαλή, επειδή δε χάνονται πληροφορίες, δηλαδή μπορούμε να αντιγράψουμε το τελικό **int** σε ένα **char** και να πάρουμε την αρχική τιμή:

```
char c2 = i1;
cout << c << ' ' << i1 << ' ' << c2 << '\n';
```

Αυτό θα δώσει

```
x 120 x
```

Υπό αυτή την έννοια, δηλαδή ότι μια τιμή μετατρέπεται πάντα σε μια ίση τιμή ή (για τα **double**) κατά προσέγγιση σε μια ίση τιμή — αυτές οι μετατροπές είναι ασφαλείς:

bool σε **char**
bool σε **int**
bool σε **double**
char σε **int**
char σε **double**
int σε **double**

Η πιο χρήσιμη μετατροπή είναι **int** σε **double**, επειδή μας επιτρέπει να συνδυάζουμε **int** και **double** σε εκφράσεις:

```
double d1 = 2.3;
double d2 = d1+2;      // το 2 μετατρέπεται σε 2.0 πριν την πρόθεση
if (d1 < 0)            // το 0 μετατρέπεται σε 0.0 πριν τη σύγκριση
    error("d1 is negative");
```

Για ένα πραγματικά μεγάλο **int**, μπορούμε (για μερικούς υπολογιστές) να έχουμε κάποια απώλεια ακρίβειας κατά τη μετατροπή σε **double**. Αυτό ωστόσο είναι σπάνιο.

3.9.2 Επισφαλείς μετατροπές

Οι ασφαλείς μετατροπές είναι συνήθως πλεονέκτημα για τον προγραμματιστή και απλοποιούν τη σύνταξη κώδικα. Δυστυχώς, η C++ επιτρέπει επίσης τις (ενδεχόμενες) επισφαλείς μετατροπές. Με τη λέξη επισφαλής, εννοούμε ότι μια τιμή μπορεί να μετατραπεί με δυναμικό τρόπο σε μια τιμή άλλου τύπου που δεν ισούται με την αρχική τιμή. Για παράδειγμα:

```
int main()
{
    int a = 20000;
    char c = a;          // προσπάθησε να συμπιέσεις ένα μεγάλο int σε ένα μικρό
char
    int b = c;
    if (a != b)  // το != σημαίνει "όχι ίσο"
        cout << "οχ!: " << a << "!=" << b << '\n';
    else
        cout << "Τρομερό! Έχουμε μεγάλους χαρακτήρες\n";
}
```

Τέτοιες μετατροπές ονομάζονται επίσης μετατροπές "περιορισμού", επειδή τοποθετούν μια τιμή σε ένα αντικείμενο που μπορεί να είναι πολύ μικρό ("περιορισμένου μεγέθους"). Δυστυχώς, λίγοι μεταγλωττιστές προειδοποιούν για την επισφαλή αρχικοποίηση του **char** με ένα **int**. Το πρόβλημα είναι ότι ένα **int** συνήθως είναι πολύ μεγαλύτερο από ένα **char** για να μπορεί (και εν προκειμένω το κάνει) να διατηρήσει

μια τιμή **int** που δεν μπορεί να αναπαρασταθεί ως **char**. Δοκιμάστε το παρακάτω, προκειμένου να δείτε αν η τιμή **b** θα φτάσει στον υπολογιστή σας (**32** είναι το αποτέλεσμα), ή ακόμα καλύτερα πειραματιστείτε:

```
int main()
{
    double d =0;
    while (cin>>d) {          // επανάλαβε τις πιο κάτω εντολές
                              // για όσο εισάγουμε αριθμούς
    int i = d;                // προσπάθησε να συμπιέσεις ένα double σε ένα int
    char c = i;               // προσπάθησε να συμπιέσεις ένα int σε ένα char
    int i2 = c;                         // πάρε την ακέραια τιμή του χαρακτήρα
    cout << "d==" << d                  // το αρχικό double
         << " i==" << i                 // που μετατράπηκε σε int
         << " i2==" << i2               // η τιμή int του char
         << " char(" << c << ")\n";     // το char
    }
}
```

Η εντολή **while** που χρησιμοποιούμε για να επιτραπεί η δοκιμή πολλών τιμών θα εξηγηθεί στην §4.4.2.1.

ΔΟΚΙΜΑΣΤΕ ΑΥΤΟ

Τρέξτε αυτό το πρόγραμμα με διάφορες εισόδους. Δοκιμάστε μικρές τιμές (π.χ., **2** και **3**), δοκιμάστε μεγαλύτερες τιμές (μεγαλύτερες από **127**, μεγαλύτερες από **1.000**), δοκιμάστε αρνητικές τιμές, δοκιμάστε το **56**, το **89**, το **128**, δοκιμάστε μη ακέραιες τιμές (π.χ., **56.9** και **56.2**). Εκτός από το να δείχνει πώς γίνονται στον υπολογιστή σας οι μετατροπές από **double** σε **int** και από **int** σε **char**, αυτό το πρόγραμμα σας δείχνει ποιο χαρακτήρα θα εμφανίσει ο υπολογιστής σας για μια δεδομένη ακέραια τιμή (αν εμφανίσει κάποιον).

Θα διαπιστώσετε ότι πολλές τιμές εισόδου παράγουν "παράλογα" αποτελέσματα. Ουσιαστικά, προσπαθούμε να βάλουμε ένα λίτρο σε μια μικρή κούπα καφέ. Όλες οι μετατροπές

double σε **int**
double σε **char**
double σε **bool**
int σε **char**

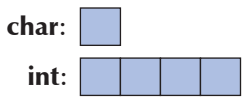
int σε **bool**
char σε **bool**

γίνονται αποδεκτές από το μεταγλωττιστή, ακόμα κι αν είναι επισφαλείς. Είναι επισφαλείς με την έννοια ότι η τιμή που αποθηκεύεται μπορεί να διαφέρει από την τιμή που δίνεται. Γιατί όμως αυτό μπορεί να αποδειχθεί πρόβλημα; Επειδή συχνά δεν υποπτευόμαστε ότι μια επισφαλής μετατροπή λαμβάνει χώρα. Δείτε το παρακάτω:

```
double x = 2.7;
// πολύς κώδικας
int y = x;       // το y γίνεται 2
```

Μέχρι να ορίσουμε το **y**, μπορεί να έχουμε ξεχάσει ότι το **x** ήταν **double** ή μπορεί να ξεχάσουμε προσωρινά ότι η μετατροπή **double** σε **int** περικόπτει (πάντα με στρογγυλοποίηση προς τα κάτω) και δε χρησιμοποιεί τη συμβατική στρογγυλοποίηση των 4/5. Αυτό που συμβαίνει είναι απόλυτα προβλέψιμο, αλλά δεν υπάρχει τίποτα στο **int y = x;** που να μας θυμίζει ότι χάνοντας πληροφορίες (το **.7**).

Οι μετατροπές από **int** σε **char** δεν έχουν προβλήματα με την περικοπή — ούτε το ένα ούτε το άλλο δεν μπορούν να αναπαραστήσουν κλάσμα ακεραίου. Ωστόσο, ένα **char** μπορεί να διατηρήσει πολύ μικρές ακέραιες τιμές. Σε ένα PC ένα **char** είναι 1 byte, ενώ ένα **int** είναι 4 bytes:



Επομένως, δεν μπορούμε να τοποθετήσουμε ένα μεγάλο αριθμό, όπως το 1.000, σε ένα **char** χωρίς απώλεια πληροφοριών: η τιμή "περιορίζεται". Για παράδειγμα:

```
int a = 1000;
char b = a;    // το b γίνεται –24 (σε μερικούς υπολογιστές)
```

Δεν έχουν όλες οι τιμές **int** ισοδύναμα **char** και το ακριβές εύρος των τιμών **char** εξαρτάται από τη συγκεκριμένη υλοποίηση. Σε ένα PC το εύρος των τιμών **char** είναι [–128:127], αλλά μόνο το [0,127] μπορεί να χρησιμοποιηθεί και σε άλλους υπολογιστές, επειδή υπάρχει η πιθανότητα κάποιος άλλος υπολογιστής να έχει διαφορετικό εύρος για τις τιμές του **char**, όπως [0:255].

Γιατί οι άνθρωποι δέχονται το πρόβλημα του περιορισμού των μετατροπών; Ο βασικός λόγος είναι ιστορικός: Η C++ κληρονόμησε τις μετατροπές περιορισμού από τον πρόγονό της, τη C, που σημαίνει ότι από την πρώτη μέρα της C++ υπήρχε πολύς κώδικας που εξαρτιόταν από μετατροπές περιορισμού. Επίσης, πολλές τέτοιες μετατροπές δεν προκαλούν προβλήματα, επειδή οι τιμές που εμπλέκονται στη διαδικασία τυχαίνει να βρίσκονται εντός εύρους και πολλοί προγραμματιστές αντιστέκονται στους μεταγλωττιστές "που τους λένε τι πρέπει να κάνουν". Συγκεκριμένα, τα προβλήματα με τις επισφαλείς μετατροπές συχνά μπορούν να αντιμετωπιστούν σε

μικρά προγράμματα και από έμπειρούς προγραμματιστές. Μπορεί να αποτελούν πηγή σφαλμάτων σε μεγαλύτερα προγράμματα, αλλά και μια σημαντική αιτία προβλημάτων για νέους προγραμματιστές. Ωστόσο, οι μεταγλωττιστές μπορούν να προειδοποιούν για μετατροπές περιορισμού — και πολλοί το κάνουν.

Τι θα πρέπει λοιπόν να κάνετε αν σκέφτεστε ότι μια μετατροπή μπορεί να οδηγήσει σε λάθος τιμή; Απλά ελέγχετε την τιμή πριν την εκχωρήσετε, όπως κάναμε στο πρώτο παράδειγμα αυτής της ενότητας. Ανατρέξτε στις §5.6.4 και §7.5 για να δείτε πώς θα μπορείτε να κάνετε εύκολα αυτό τον έλεγχο.

Διαδικασία

Μετά από κάθε βήμα αυτής της διαδικασίας, τρέξτε το πρόγραμμά σας προκειμένου να βεβαιωθείτε ότι όλα γίνονται όπως θα περιμένατε. Καταγράψτε τα λάθη που κάνετε, ώστε να προσπαθήσετε να τα αποφύγετε στο μέλλον.

1. Αυτή η διαδικασία αφορά στη σύνταξη ενός προγράμματος που παράγει μια απλή επιστολή φόρμας, η οποία βασίζεται σε είσοδο χρήστη. Ξεκινήστε πληκτρολογώντας τον κώδικα από την §3.1, ο οποίος ζητά από ένα χρήστη να εισαγάγει το όνομά του και γράφει "Γεια σου **first_name**", όπου **first_name** είναι το όνομα που εισήγαγε ο χρήστης. Κατόπιν, τροποποιήστε τον κώδικά σας ως εξής: αλλάξτε την προτροπή σε "Εισαγάγετε το όνομα του προσώπου στο οποίο θέλετε να στείλετε επιστολή" και αλλάξτε την έξοδο σε "Αγαπητέ **first_name**". Μην ξεχάσετε να αλλάξετε το επίθετο "αγαπητέ" σε "αγαπητή" αν χρειάζεται.
2. Προσθέστε λίγες εισαγωγικές προτάσεις, όπως "Πώς είσαι; Εγώ είμαι καλά. Σε σκεφτόμουν". Η πρώτη γραμμή θα πρέπει να τοποθετείται σε εσοχή. Προσθέστε μερικές ακόμα γραμμές της επιλογής σας – άλλωστε, είναι δική σας επιστολή.
3. Ζητήστε τώρα από το χρήστη να σας δώσει το όνομα άλλου φίλου του και αποθηκεύστε το στη μεταβλητή **friend_name**. Προσθέστε την εξής γραμμή στην επιστολή σας: "Έχεις δει το/τη **friend_name** τελευταία;"
4. Δηλώστε μια μεταβλητή **char** με όνομα **friend_sex** και αρχικοποιήστε τη με την τιμή 0. Ζητήστε από το χρήστη να εισαγάγει ένα **m** αν ο φίλος είναι άντρας (male) και ένα **f** αν είναι γυναίκα (female). Δώστε την τιμή που εισάγει ο χρήστης στη μεταβλητή **friend_sex**. Κατόπιν, χρησιμοποιήστε δύο εντολές **if** για να γράψετε το εξής:

   Εάν ο φίλος είναι άντρας, γράψτε "Αν δεις το **friend_name** ζήτησέ του να μου τηλεφωνήσει".

   Αν ο φίλος είναι γυναίκα, γράψτε "Αν δεις τη **friend_name** ζήτησέ της να μου τηλεφωνήσει".

   Κάντε το ίδιο για τη φράση του βήματος 3, "Έχεις δει τον/τη **friend_name** τελευταία;".
5. Ζητήστε από το χρήστη να εισαγάγει την ηλικία του παραλήπτη και περάστε αυτή την τιμή στην **int** μεταβλητή **age**. Το πρόγραμμα θα πρέπει να γράψει "Έμαθα ότι είχες γενέθλια και ότι έκλεισες τα **age**". Εάν το **age** είναι 0 ή μικρότερο ή 110 ή μεγαλύτερο, καλέστε τη συνάρτηση **error ("αστειεύεσαι!")**.

6. Προσθέστε τα εξής στην επιστολή σας:

   Εάν ο φίλος σας είναι κάτω από 12, γράψτε "Την επόμενη χρονιά να κλείσεις τα **age+1**".

   Εάν ο φίλος σας είναι 17, γράψτε "Την επόμενη χρονιά θα μπορείς να ψηφίσεις".

   Εάν ο φίλος σας είναι πάνω από 70, γράψτε "Ελπίζω να απολαμβάνεις τη σύνταξη".

   Ελέγξτε το πρόγραμμά σας, προκειμένου να βεβαιωθείτε ότι ανταποκρίνεται κατάλληλα σε όλες τις τιμές.
7. Προσθέστε τη φράση "Με αγάπη", προσθέστε δύο κενές γραμμές για την υπογραφή σας και ολοκληρώστε με το όνομά σας.

Επανάληψη

1. Τι σημαίνει ο όρος *προτροπή*;
2. Ποιον τελεστή χρησιμοποιείτε για να διαβάσετε μια μεταβλητή;
3. Εάν θέλετε ο χρήστης να εισαγάγει μια ακέραια τιμή στο πρόγραμμά σας για τη μεταβλητή με το όνομα **number**, ποιες δύο γραμμές κώδικα θα μπορούσατε να γράψετε ώστε να ζητήσετε από το χρήστη να γράψει τον αριθμό και να εισαγάγετε την τιμή στο πρόγραμμά σας;
4. Πώς ονομάζεται το **\n** και ποιο σκοπό εξυπηρετεί;
5. Τι τερματίζει την είσοδο σε μια συμβολοσειρά;
6. Τι τερματίζει την είσοδο σε έναν ακέραιο;
7. Πώς θα γράφατε

   ```
   cout << "Γεια σου ";
   cout << first_name;
   cout << "!\n";
   ```

   ως μια γραμμή κώδικα;
8. Τι είναι ένα αντικείμενο;
9. Τι είναι μια κατά γράμμα τιμή (literal);
10. Ποια είδη τιμών κατά γράμμα υπάρχουν;
11. Τι είναι μια μεταβλητή;
12. Ποια είναι τα συνήθη μεγέθη των **char**, **int** και **double**;
13. Ποια μέτρα εφαρμόζουμε για το μέγεθος μικρών οντοτήτων στη μνήμη, όπως **int** και **string**;
14. Ποια είναι η διαφορά μεταξύ = και ==;
15. Τι είναι ορισμός;
16. Τι είναι αρχικοποίηση και σε τι διαφέρει από μια εκχώρηση;
17. Τι είναι αλληλουχία συμβολοσειράς και πώς λειτουργεί στη C++;
18. Ποια από τα παρακάτω είναι σωστά ονόματα στη C++; Εάν ένα όνομα είναι λάθος, γιατί είναι λάθος;

```
This_little_pig      This_1_is fine      2_For_1_special
latest thing         the_$12_method      _this_is_ok
MiniMineMine         number              correct;
```

19. Δώστε πέντε παραδείγματα σωστών ονομάτων που δε θα χρησιμοποιούσατε επειδή μπορεί να προκαλέσουν σύγχυση.
20. Αναφέρετε μερικούς καλούς κανόνες για την επιλογή ονομάτων.
21. Τι είναι η ασφάλεια τύπων και γιατί είναι σημαντική;
22. Γιατί η μετατροπή από **double** σε **int** είναι κάτι που δεν πρέπει να γίνεται;
23. Ορίστε έναν κανόνα που θα σας βοηθήσει να αποφασίσετε αν μια μετατροπή από έναν τύπο σε άλλον είναι ασφαλής ή επισφαλής.

Όροι

cin
αλληλουχία
αντικείμενο
αρχικοποίηση
ασφάλεια τύπων
αύξηση
δήλωση
εκχώρηση
μείωση
μεταβλητή
μετατροπή
όνομα
ορισμός
περιορισμός
πράξη
τελεστής
τιμή
τύπος

Ασκήσεις

1. Εάν δεν το έχετε κάνει ήδη, κάντε τις ασκήσεις **Δοκιμάστε αυτό** του κεφαλαίου.
2. Γράψτε ένα πρόγραμμα σε C++ που μετατρέπει μίλια σε χιλιόμετρα. Το πρόγραμμά σας θα πρέπει να έχει μια λογική προτροπή για το χρήστη, ώστε να γνωρίζει ότι εισάγει μίλια. Υπόδειξη: ένα μίλι ισούται με 1,609 χιλιόμετρα.
3. Γράψτε ένα πρόγραμμα που δεν κάνει τίποτα εκτός από το να δηλώνει κάποιες μεταβλητές με σωστά και λάθος ονόματα (όπως **int double = 0;**), ώστε να δείτε πώς αντιδρά ο μεταγλωττιστής.
4. Γράψτε ένα πρόγραμμα που ζητά από το χρήστη να εισαγάγει δύο ακέραιες τιμές. Αποθηκεύστε αυτές τις τιμές σε μεταβλητές **int** με ονόματα **val1** και **val2**. Γράψτε ένα πρόγραμμα που θα βρίσκει το μικρότερο, το μεγαλύτερο, το άθροισμα, τη διαφορά, το γινόμενο και το λόγο αυτών των τιμών και αναφέρετε τα αποτελέσματα στο χρήστη.
5. Τροποποιήστε το πιο πάνω πρόγραμμα για να ζητήσετε από το χρήστη να εισαγάγει τιμές κινητής υποδιαστολής και αποθηκεύστε τες σε μεταβλητές **double**. Συγκρίνετε τις εξόδους των δύο προγραμμάτων για κάποιες εισόδους της επιλογής σας. Τα αποτελέσματα είναι ίδια; Ποια θα έπρεπε να είναι; Ποια είναι η διαφορά;
6. Γράψτε ένα πρόγραμμα που ζητά από το χρήστη να εισαγάγει τρεις ακέραιες τιμές και μετά εξάγει τις τιμές σε αριθμητική ακολουθία που χωρίζεται με κόμμα. Αν λοιπόν ο χρήστης εισάγει τις τιμές 10 4 6, η έξοδος θα πρέπει να είναι 4, 6, 10. Εάν δύο τιμές είναι ίδιες, θα πρέπει να τοποθετηθούν η μια δίπλα στην άλλη. Για παράδειγμα, η είσοδος 4 5 4 θα πρέπει να δώσει 4, 4, 5.
7. Κάντε την άσκηση 6, αλλά με τρεις τιμές συμβολοσειρών. Αν λοιπόν ο χρήστης εισαγάγει τις τιμές "**Steinbeck**", "**Hemingway**", "**Fitzgerald**", η έξοδος θα πρέπει να είναι "**Fitzgerald, Hemingway, Steinbeck**".

8. Γράψτε ένα πρόγραμμα που θα ελέγχει μια ακέραια τιμή, ώστε να διαπιστώσει αν είναι άρτια ή περιττή. Όπως πάντα, η έξοδός σας θα πρέπει να είναι σαφής και πλήρης. Με άλλα λόγια, μην εξάγετε απλά "ναι" ή "όχι". Η έξοδός σας θα πρέπει να είναι ουσιαστική, όπως "Η τιμή 4 είναι άρτιος αριθμός". Υπόδειξη: Δείτε πάλι τον τελεστή υπόλοιπου (ακέραια διαίρεση) στην §3.4.
9. Γράψτε ένα πρόγραμμα που μετατρέπει αριθμούς που γράφονται ολογράφως, όπως "μηδέν" και "δύο" σε ψηφία, όπως 0 και 2. Όταν ο χρήστης εισαγάγει έναν αριθμό, το πρόγραμμα θα πρέπει να εμφανίζει το αντίστοιχο ψηφίο. Κάντε το για τις τιμές 0, 1, 2, 3 και 4 και γράψτε το μήνυμα "δε γνωρίζω αυτό τον αριθμό" αν ο χρήστης εισαγάγει κάτι που δεν αντιστοιχεί σ' αυτό που ζητάτε, όπως "χαζέ υπολογιστή!"
10. Γράψτε ένα πρόγραμμα που λαμβάνει μια πράξη που ακολουθείται από δύο τελεστέους και εξάγει το αποτέλεσμα. Για παράδειγμα:

    ```
    + 100 3.14
    * 4 5
    ```

    Διαβάστε την πράξη σε μια συμβολοσειρά, η οποία ονομάζεται **operation**, και χρησιμοποιήστε μια εντολή **if** ώστε να καταλάβετε ποια πράξη θέλει ο χρήστης, για παράδειγμα, **if (operation=="+")**. Διαβάστε τους τελεστέους σε μεταβλητές τύπου **double**. Υλοποιήστε το για τις πράξεις +, –, *, / (συν, πλην, επί και διά) με τις προφανείς ερμηνείες τους.
11. Γράψτε ένα πρόγραμμα που ζητά από το χρήστη να εισαγάγει ποσότητες κερμάτων (ένα λεπτό, δύο λεπτά, πέντε λεπτά, δέκα λεπτά, είκοσι λεπτά, πενήντα λεπτά, ένα ευρώ και δύο ευρώ). Ζητήστε από το χρήστη να γράψει ξεχωριστά το πλήθος κερμάτων για κάθε ποσό, π.χ., "πόσα κέρματα του λεπτού έχεις;" Το πρόγραμμά σας θα πρέπει να εμφανίσει κάτι σαν το παρακάτω:

    Έχεις 23 κέρματα του ενός λεπτού.

    Έχεις 12 κέρματα των δύο λεπτών.

    Έχεις 17 κέρματα των πέντε λεπτών.

    Έχεις 14 κέρματα των δέκα λεπτών.

    Έχεις 8 κέρματα των είκοσι λεπτών.

    Έχεις 7 κέρματα των πενήντα λεπτών.

    Έχεις 3 κέρματα του ενός ευρώ.

    Έχεις 2 κέρματα των δύο ευρώ.

    Έχεις συνολικά 1482 λεπτά.

    Θα πρέπει να σκεφτείτε πώς θα στοιχίσετε τους αριθμούς δεξιά. Στο τέλος, αναφέρετε το άθροισμα σε ευρώ και λεπτά και όχι μόνο σε λεπτά: 14 ευρώ και 82 λεπτά.

Υστερόγραφο

Μην υποτιμάτε τη σημασία της έννοιας της ασφάλειας τύπων. Οι τύποι βρίσκονται στο κέντρο του σωστού προγραμματισμού και των πιο αποτελεσματικών τεχνικών για κατασκευή προγραμμάτων που βασίζονται στη σχεδίαση και στη χρήση τύπων — όπως θα δείτε στα Κεφάλαια 6 και 9, στα Μέρη II, III και IV.