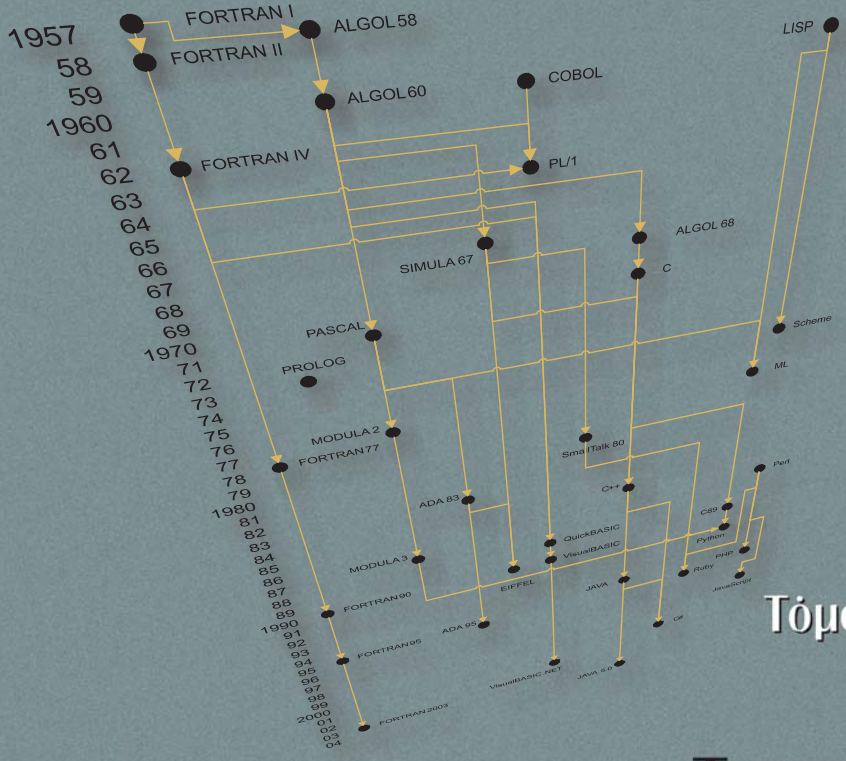
Ελληνικο Ανοικτο Πανεπιστημιο

Εισαγωγή στην Πληροφορική



Τόμος Β΄

Τεχνικές Προγραμματισμού

Αχιλλέας Δ. Καμέας
Επίκουρος Καθηγητής
Ελληνικού Ανοικτού
Πανεπιστημίου

Τεχνικές Προγραμματισμού

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Β'

Τεχνικές Προγραμματισμού

ΑΧΙΛΛΕΑΣ Δ. ΚΑΜΕΑΣ

Δρ Μηχανικός Η/Υ & Πληροφορικής

ΠΑΤΡΑ 2000

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Β'
Τεχνικές Προγραμματισμού

Συγγραφή
ΑΧΙΛΛΕΑΣ Δ. ΚΑΜΕΑΣ
Δρ Μηχανικός Η/Υ & Πληροφορικής

Κριτική Ανάγνωση
ΕΛΠΙΔΑ ΚΕΡΑΥΝΟΥ
Καθηγήτρια Τμήματος Πληροφορικής
Πανεπιστημίου Κύπρου

Ακαδημαϊκός Υπεύθυνος για την επιστημονική επιμέλεια του τόμου
ΠΑΝΑΓΙΩΤΗΣ ΠΙΝΤΕΛΑΣ
Καθηγητής Τμήματος Μαθηματικών Πανεπιστημίου Πατρών

Επιμέλεια στη μέθοδο της εκπαίδευσης από απόσταση
ΓΕΡΑΣΙΜΟΣ ΚΟΥΣΤΟΥΡΑΚΗΣ

Γλωσσική Επιμέλεια
ΙΩΑΝΝΗΣ ΘΕΟΦΙΛΑΣ

Τεχνική Επιμέλεια
ΕΣΠΙ ΕΚΔΟΤΙΚΗ Ε.Π.Ε.

Καλλιτεχνική Επιμέλεια, Σελιδοποίηση
TYPORAMA

Συντονισμός ανάπτυξης εκπαιδευτικού υλικού και γενική επιμέλεια των εκδόσεων
ΟΜΑΔΑ ΕΚΤΕΛΕΣΗΣ ΕΡΓΟΥ ΕΑΠ / 1997–2000

ISBN: 960–538–077–3
Κωδικός Έκδοσης: ΠΛΗ 10/2

Περιεχόμενα

Πρόλογος 11

ΚΕΦΑΛΑΙΟ 1
Εισαγωγή

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 15

1.1 Αντιμετώπιση προβλημάτων 17

1.2 Αντιμετώπιση προβλημάτων με τη χρήση υπολογιστή 18

1.2.1 Ο υπολογιστής ως σύστημα επεξεργασίας δεδομένων 20

1.2.2 Ο υπολογιστής ως μηχανή εκτέλεσης προγραμμάτων 22

Σύνοψη 24

Βιβλιογραφία Κεφαλαίου 1 24

ΚΕΦΑΛΑΙΟ 2
Αλγόριθμοι

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 25

2.1 Πρόβλημα και αλγόριθμος επίλυσης 27

2.2 Χαρακτηριστικά των αλγορίθμων 28

2.3 Αναπαράσταση αλγορίθμων με ψευδοκώδικα 31

2.4 Αναπαράσταση αλγορίθμων με διάγραμμα ροής 41

Σύνοψη 47

Βιβλιογραφία Κεφαλαίου 2 47

ΚΕΦΑΛΑΙΟ 3
Πρακτικές προγραμματισμού

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 49

3.1 Κατά βήμα εκλέπτυνση 51

3.2 Αρθρωτός προγραμματισμός ... 60
3.3 Τεχνοτροπίες (στυλ) προγραμματισμού ... 64
3.3.1 Προγραμματισμός για επαναχρησιμοποίηση ... 64
3.3.2 Προγραμματισμός με πλεονασμό ... 67
3.3.3 Αμυντικός προγραμματισμός ... 67
3.4 Παραδείγματα προγραμματισμού ... 69
Σύνοψη ... 72
Βιβλιογραφία Κεφαλαίου 3 ... 72

ΚΕΦΑΛΑΙΟ 4
Σχεδίαση προγράμματος

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις ... 73
4.1 Το κριτήριο της «σχεδιαστικής διαδικασίας» ... 78
4.2 Το κριτήριο της «σχεδιαστικής κατεύθυνσης» ... 80
4.2.1 Σχεδίαση «από πάνω προς τα κάτω» ... 80
4.2.2 Σχεδίαση «από κάτω προς τα πάνω» ... 82
4.2.3 Προγραμματισμός «από πάνω προς τα κάτω» ... 83
4.2.4 Προγραμματισμός «από κάτω προς τα πάνω» ... 85
4.2.5 Επίπεδα αφαιρετικότητας ... 86
4.2.6 Σχεδίαση «από τη μέση προς την άκρη» ... 86
4.3 Το κριτήριο της «μονάδας διάσπασης» ... 89
4.4 Είναι καλό το σχέδιό μου; ... 92
4.4.1 Συνοχή ... 92
4.4.2 Σύζευξη ... 94
Σύνοψη ... 97
Βιβλιογραφία Κεφαλαίου 4 ... 97

ΚΕΦΑΛΑΙΟ 5
Άλλα εργαλεία σχεδίασης

Σκοπός, Προσδοκώμενα αποτελέσματα,

Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 99
5.1 Γλώσσα σχεδίασης προγραμμάτων 101
5.2 Δομοδιαγράμματα 102
5.3 Διαγράμματα δομής 103
5.4 Διαγράμματα HIPO 104
5.5 Διαγράμματα WARNIER–ORR 106
5.6 Διαγράμμα Jackson 108
5.7 Ποιο εργαλείο να επιλέξω; 109
5.8 Αυτοματοποίηση των διαδικασιών προγραμματισμού 112
5.8.1 Case 112
5.8.2 Εργαλεία προγραμματισμού 113
Σύνοψη 114
Βιβλιογραφία Κεφαλαίου 5 114

ΚΕΦΑΛΑΙΟ 6
Αρχές δομημένου διαδικασιακού προγραμματισμού

Σκοπός, Προσδοκώμενα αποτελέσματα,
Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 115
6.1 Βασικές δομές προγραμματισμού 118
6.2 Μερικές ακόμη προγραμματιστικές δομές 119
6.2.1 Φωλιασμένες αποφάσεις 120
6.2.2 Επιλογή με πολλά ενδεχόμενα 120
6.2.3 Άλλα δύο είδη επαναλήψεων 122
6.3 Εισαγωγή στον προγραμματισμό 123
6.4 Πίνακες 127
6.4.1 Πρόληψη σφαλμάτων 140
6.5 Δυναμικές δομές δεδομένων 141
6.5.1 Δείκτες 142
6.5.2 Διασυνδεδεμένες λίστες 144
6.5.3 Άλλες δυναμικές δομές δεδομένων 153

6.5.4 Πρόληψη σφαλμάτων 155
6.6 Μπορώ να παραλάβω τους κανόνες; 157
Σύνοψη 162
Βιβλιογραφία Κεφαλαίου 6 162

ΚΕΦΑΛΑΙΟ 7

Προχωρημένα θέματα διαδικασιακού προγραμματισμού

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 163
7.1 Υπο–προγράμματα 166
7.1.1 Διαδικασίες και συναρτήσεις 167
7.1.2 Εμβέλεια 172
7.1.3 Παράμετροι 175
7.1.4 Πώς υλοποιείται η κλήση υπο–προγραμμάτων 179
7.2 Αναδρομή 182
7.3 Οπισθοδρόμηση 188
7.4 Ταξινόμηση και αναζήτηση 192
7.4.1 Ταξινόμηση με επιλογή 193
7.4.2 Ταξινόμηση με παρεμβολή 193
7.4.3 Ταξινόμηση φυσαλίδας 194
7.4.4 Γρήγορη ταξινόμηση 195
7.4.5 Δυαδική αναζήτηση 196
Σύνοψη 200
Βιβλιογραφία Κεφαλαίου 7 200

ΚΕΦΑΛΑΙΟ 8

Αρχές δομημένου διαδικασιακού προγραμματισμού

Σκοπός, Προσδοκώμενα αποτελέσματα, Έννοιες κλειδιά, Εισαγωγικές παρατηρήσεις 201
8.1 Τεκμηρίωση 203

8.1.1 Εξωτερική τεκμηρίωση 204
8.1.2 Εσωτερική τεκμηρίωση 207
8.2 Διαχείριση λαθών 210
8.3 Αποδοτικότητα 222
Σύνοψη 226
Βιβλιογραφία Κεφαλαίου 8 226

Επίλογος 227
Απαντήσεις ασκήσεων αυτοαξιολόγησης 228
Προτεινόμενη βιβλιογραφία για παραπέρα μελέτη 242
Γλωσσάρι όρων 245
Ελληνόγλωσση βιβλιογραφία 269
Ξενόγλωσση βιβλιογραφία 269

Στην Κατερίνα,

τη Φοίβη και τη Νεφέλη ...

Πρόλογος

Το βιβλίο που κρατάτε απαντά σε μερικές κοινές ερωτήσεις:

Ε: Τι χρειάζεται ένας υπολογιστής για να λειτουργήσει;

Α: Ηλεκτρικό ρεύμα, σίγουρα (και ένα κλιματιζόμενο δωμάτιο, ευχαριστώ)

Ε: Τι χρειάζεται ένας υπολογιστής για να είναι χρήσιμος;

Α: Λογισμικό (προγράμματα, δηλαδή)

Απαντά και σε κάποιες περισσότερο δύσκολες:

Ε: Γιατί είναι χρήσιμοι οι υπολογιστές;

Α: … (Από πού να αρχίσει κανείς …)

Λοιπόν, στο **Κεφάλαιο 1**, θα συζητήσουμε τη χρήση των υπολογιστών ως μηχανές επίλυσης προβλημάτων. Το σκεπτικό αυτό είναι θεμελιώδες για την κατανόηση των στόχων του παρόντος τόμου: οι υπολογιστές είναι μηχανές, οι οποίες μπορούν να εκτελέσουν με ακρίβεια και ταχύτητα όλα τα βήματα μιας διαδικασίας επίλυσης ενός προβλήματος.

Όμως, για ένα συγκεκριμένο πρόβλημα, ποιος έχει ανακαλύψει τη διαδικασία επίλυσής του; Σίγουρα όχι ο υπολογιστής, ο οποίος αποτελεί ένα σύστημα επεξεργασίας δεδομένων: δέχεται ένα σύνολο πληροφοριών, εφαρμόζει σε αυτά μία συγκεκριμένη επεξεργασία και παράγει το αποτέλεσμά της.

Στο **Κεφάλαιο 2**, θα δούμε ότι η σχεδίαση της διαδικασίας επίλυσης ενός προβλήματος είναι ευθύνη του προγραμματιστή. Αυτός πρέπει πρώτα απ᾽ όλα να περιγράψει τα βήματά της με τη μορφή ενός αλγο-

ρίθμου. Η περιγραφή μπορεί να είναι λεκτική (με τη χρήση ψευδοκώδικα), ή γραφική (με τη χρήση Διαγραμμάτων Ροής Προγράμματος).

Η ανάπτυξη ενός αλγορίθμου μπορεί να γίνει ακολουθώντας διάφορες πρακτικές, τεχνοτροπίες και παραδείγματα. Στο **Κεφάλαιο 3** παρουσιάζονται τα περισσότερο διαδεδομένα από αυτά (όπως η Κατά Βήμα Εκλέπτυνση). Να θυμάστε όμως ότι οι έμπειροι προγραμματιστές υιοθετούν τελικά ένα εντελώς προσωπικό στυλ προγραμματισμού.

Το **Κεφάλαιο 4** αναφέρεται στη σχεδίαση αλγορίθμων και προγραμμάτων και είναι ιδιαίτερα σημαντικό για την προσέγγιση που ακολουθείται στον τόμο αυτό. Σε όλο το βιβλίο, ο προγραμματισμός αντιμετωπίζεται ως φάση του κύκλου ζωής λογισμικού, μιας ευρύτερης δραστηριότητας, η οποία οδηγεί στην ανάπτυξη λογισμικού. Στον κύκλο ζωής λογισμικού περιλαμβάνονται και άλλες φάσεις, πριν και μετά τη φάση του προγραμματισμού, με τις οποίες η τελευταία έχει άμεση σχέση. Κάποιες από τις προηγούμενες φάσεις (π.χ. ανάλυση απαιτήσεων, σχεδίαση) παράγουν προϊόντα (π.χ. προδιαγραφές λογισμικού) που είναι απαραίτητα κατά τον προγραμματισμό. Αντίστοιχα, τα προϊόντα του προγραμματισμού (π.χ. κώδικας, τεκμηρίωση) είναι απαραίτητα σε επόμενες φάσεις (π.χ. έλεγχος, συντήρηση). Επιπλέον, εργασίες προγραμματισμού συνήθως περιέχονται και σε άλλες φάσεις (π.χ. έλεγχος, εκσφαλμάτωση).

Συνεπώς, ο προγραμματιστής πρέπει να είναι ικανός να σχεδιάσει έναν αλγόριθμο ή ένα πρόγραμμα, το οποίο στη συνέχεια πρέπει να μοιραστεί με όλες τις ενδιαφερόμενες πλευρές (διαχειριστές του έργου, τελικοί χρήστες, άλλοι προγραμματιστές κ.ά.). Αυτός είναι ο λόγος για τον οποίο στο **Κεφάλαιο 5** παρουσιάζονται μερικά ακόμη διαδεδομένα εργαλεία αναπαράστασης αλγορίθμων.

Όταν θα έχετε φτάσει στο σημείο αυτό, πιστεύω ότι θα έχετε δημιουργήσει ισχυρή άποψη υπέρ της ποιοτικής σχεδίασης και της ακριβούς αναπαράστασης των αλγορίθμων. Στο **Κεφάλαιο 6**, λοιπόν, είναι η κατάλληλη στιγμή για να μελετήσουμε τις αρχές του δομημένου διαδικασιακού προγραμματισμού. Η μελέτη γίνεται μέσα από ένα πλήθος αλγορίθμων, παραδειγμάτων και δραστηριοτήτων καθώς και μιας μελέτης περίπτωσης (κάθε ομοιότητα με υπάρχοντα πρόσωπα είναι εσκεμμένη).

Στο κεφάλαιο αυτό θα έχετε την ευκαιρία να μελετήσετε διαδεδομέ-

νους αλγορίθμους για στατικές και δυναμικές δομές δεδομένων (πίνακες και λίστες, αντίστοιχα), ενώ στο **Κεφάλαιο 7** παρουσιάζονται «προχωρημένες» τεχνικές προγραμματισμού, οι οποίες εφαρμόζονται σε περισσότερο σύνθετα προβλήματα. Πρόκειται για το πιο «δύσκολο» κεφάλαιο του βιβλίου, το οποίο όμως κρύβει όλη την ομορφιά του προγραμματισμού.

Ο στόχος του τόμου δεν είναι μόνο να σας κάνει ικανούς προγραμματιστές, παρουσιάζοντας όλες τις διαδεδομένες (ή τις έξυπνες) προγραμματιστικές τεχνικές, τις βασικές δομές προγραμματισμού και τους καλύτερους αλγορίθμους. Εξάλλου, για να γίνει κανείς καλός προγραμματιστής, εκτός από ένα καλό βιβλίο, απαιτείται και πολλή πρακτική εξάσκηση «πάνω» στον υπολογιστή.

Ο τόμος αυτός στοχεύει να σας κάνει συνειδητοποιημένους, «πραγματικούς» προγραμματιστές, ικανούς να ενταχθείτε σε μία ομάδα ανάπτυξης λογισμικού και να ανεβάσετε την ποιότητα και τις δυνατότητες της ομάδας. Για το λόγο αυτό επιμένουμε τόσο στη μετάδοση γενικότερων γνώσεων, όπως οι τεχνικές και τα εργαλεία σχεδίασης, αλλά και τα ειδικά ζητήματα (τεκμηρίωση, διαχείριση σφαλμάτων, αποδοτικότητα) που παρουσιάζονται στο **Κεφάλαιο 8**.

Το βιβλίο αυτό θέλει να σας βοηθήσει να κάνετε χρήσιμους τους υπολογιστές, αναπτύσσοντας προγράμματα. Θέλει επίσης να κάνει εσάς «χρήσιμους» προγραμματιστές, με την έννοια της θετικής συμβολής στο έργο της ανάπτυξης λογισμικού. Θέλει τέλος να σας βοηθήσει να γράψετε προγράμματα χρήσιμα για τους στοχευόμενους χρήστες τους, ώστε οι υπολογιστές να γίνουν χρήσιμοι και σε αυτούς.

Αντίθετα με την (πρόσφατα) διαδεδομένη πρακτική, στο βιβλίο παρουσιάζονται αλγόριθμοι και όχι προγράμματα. Το μεγάλο πλεονέκτημα των αλγορίθμων είναι ότι δεν εξαρτώνται από κάποια γλώσσα προγραμματισμού, οπότε και οι αναγνώστες του τόμου δεν χρειάζεται να γνωρίζουν κάποια τέτοια γλώσσα. Αντίθετα, οι αλγόριθμοι περιγράφονται με τη βοήθεια μιας ψευδογλώσσας, η οποία ορίζεται σταδιακά μέσα στον τόμο και είναι μακρινός συγγενής της Pascal. Να θυμάστε όμως ότι οι αλγόριθμοι δεν είναι απευθείας εκτελέσιμοι από κάποιον μεταγλωττιστή της γλώσσας Pascal.

Για την κατανόηση των περιεχομένων του τόμου δεν προαπαιτείται κάποια ειδική γνώση, εκτός ίσως από μία γενική αντίληψη του τρόπου λειτουργίας του ηλεκτρονικού υπολογιστή. Σε κάποια κεφάλαια του τόμου αναφέρονται συγγενείς Θ.Ε. του ΕΑΠ, από τις οποίες θα μπορούσε κανείς να αποκομίσει περισσότερη γνώση σχετικά με κάποιο συγκεκριμένο αντικείμενο, χωρίς όμως η ανάγνωση αυτών των βιβλίων να είναι απαραίτητη προϋπόθεση για την κατανόηση του τόμου που κρατάτε.

Εξάλλου, ο τόμος περιέχει πολυάριθμα παραδείγματα, δραστηριότητες και ασκήσεις αυτοαξιολόγησης. Οι απαντήσεις στις στις δραστηριότητες δίνονται μέσα στο κείμενο· οι απαντήσεις στις ασκήσεις αυτοαξιολόγησης έχουν συγκεντρωθεί στο τέλος του τόμου (σας συνιστώ να προσπαθήσετε να δώσετε τις δικές σας απαντήσεις πριν καταφύγετε στις απαντήσεις που περιέχονται στο βιβλίο). Στο τέλος του τόμου θα βρείτε και ένα γλωσσάριο όρων, όπου επεξηγούνται εν συντομία οι έννοιες κλειδιά του κάθε κεφαλαίου, τις βιβλιογραφικές πηγές που χρησιμοποίησα για τη συγγραφή του τόμου, αλλά και βιβλία που συστήνω σε όσους θέλουν να εμβαθύνουν στο αντικείμενο. Ακόμη, ο τόμος περιέχει μια εκτεταμένη μελέτη περίπτωσης, με πρωταγωνιστή το Βύρωνα, έναν ικανότατο νέο μηχανικό λογισμικού, ο οποίος στην προσπάθειά του να επιλύσει τα προβλήματα μηχανοργάνωσης μιας επιχείρησης, γίνεται οδηγός σας σε ένα πολύ ενδιαφέρον ταξίδι ...

Press any key to Start…

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Σκοπός

Ο στόχος του πρώτου αυτού κεφαλαίου είναι να σας παρουσιάσει την ανάπτυξη προγραμμάτων σε υπολογιστή σαν ένα τρόπο αντιμετώπισης μιας συγκεκριμένης κατηγορίας προβλημάτων και να περιγράψει εκείνα τα χαρακτηριστικά του υπολογιστή που τον κάνουν κατάλληλο για τέτοια χρήση.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο θα μπορείτε να:

- *αναφέρετε τα έξι στάδια της διαδικασίας αντιμετώπισης ενός προβλήματος*
- *εξηγήσετε γιατί όλα αυτά τα στάδια είναι απαραίτητα*
- *αντιστοιχίσετε τα γενικά αυτά στάδια σε συγκεκριμένα στάδια των μοντέλων κύκλου ζωής λογισμικού*
- *περιγράψετε τη λειτουργία του υπολογιστή ως σύστημα επεξεργασίας δεδομένων*
- *αναφέρετε τα τμήματα του υπολογιστή που χρησιμοποιούνται για είσοδο, έξοδο και επεξεργασία δεδομένων*
- *περιγράψετε τα πέντε βήματα εκτέλεσης μιας εντολής προγράμματος από τον υπολογιστή*

Έννοιες κλειδιά

- *Αντιμετώπιση προβλημάτων*
- *Δεδομένα του προβλήματος*
- *Αλγόριθμος επίλυσης*
- *Σχεδίαση της διαδικασίας επίλυσης*
- *Πρόγραμμα υπολογιστή*
- *Εντολές προγράμματος*
- *Σύστημα*
- *Είσοδος και έξοδος συστήματος*
- *Δεδομένα*
- *Επεξεργασία*
- *Κεντρική μονάδα επεξεργασίας*

- *Κύρια μνήμη*
- *Κανάλια διακίνησης πληροφοριών.*

Εισαγωγικές Παρατηρήσεις

Ο Douglas Adams στο βιβλίο του «Γυρίστε το Γαλαξία με Ωτο–στοπ» (Adams, 1979) διηγείται μια ιστορία για κάποια φυλή που κατασκεύασε τη Βαθιά Σκέψη, έναν πελώριο, καταπληκτικά έξυπνο σούπερ–κομπιούτερ για να τους λύσει όλα τα προβλήματα σχετικά με το νόημα της ζωής. Το απόσπασμα που ακολουθεί περιγράφει την εξέλιξη του εγχειρήματος:

«Σαράντα δύο!» ούρλιαξε ο Λούνκβωλ. «Αυτό είναι το μόνο που έχεις να πείς μετά από επτάμισι εκατομμύρια χρόνια εργασίας»

«Το έλεγξα από κάθε άποψη» είπε ο κομπιούτερ «και είναι σίγουρο πως αυτή είναι η απάντηση. Για να είμαι ειλικρινής μαζί σας, νομίζω πως το πρόβλημα είναι πως ποτέ δεν ξέρατε ποια είναι η ερώτηση».

«Αλλά ήταν το Μεγάλο Ερώτημα! Το Ύστατο Ερώτημα για τη Ζωή, το Σύμπαν και τα Πάντα» ούρλιαξε ο Λούνκβωλ.

«Ναι» είπε η Βαθιά Σκέψη με τον αέρα κάποιου που μπορεί άνετα να ανεχτεί κάποιον ανόητο, «αλλά ποια ακριβώς είναι»

Στην καθημερινή μας ζωή ερχόμαστε συνεχώς αντιμέτωποι με απλούστερα προβλήματα, τα οποία τις περισσότερες φορές πρέπει να λύσουμε σωστά σε λιγότερο χρόνο από αυτόν που είχε στη διάθεσή της η Βαθιά Σκέψη. Επειδή αυτό συμβαίνει από «καταβολής κόσμου», ο άνθρωπος συνειδητοποίησε τελικά ότι, εκτός από εκείνα τα ελάχιστα προβλήματα που απαιτούν έμπνευση, τα υπόλοιπα μπορούν να αντιμετωπιστούν αποτελεσματικά μόνο με μεθοδικό τρόπο.

Το εισαγωγικό αυτό κεφάλαιο θα σας παρουσιάσει στην ενότητα 1.1 μια γενική διαδικασία αντιμετώπισης προβλημάτων, πριν επικεντρωθεί στην ενότητα 1.2 σε διαδικασίες αντιμετώπισης προβλημάτων με τη χρήση υπολογιστή και τον τρόπο με τον οποίο η περιγραφή ενός προβλήματος μετασχηματίζεται σε αλγόριθμο και έπειτα σε πρόγραμμα. Στην ίδια ενότητα θα περιγραφεί και η δυνατότητα επεξεργασίας (με τη χρήση προγράμματος εντολών) των δεδομένων ενός προβλήματος για την παραγωγή της λύσης του.

1.1 Αντιμετώπιση προβλημάτων

Μπορεί να θεωρηθεί ότι η **αντιμετώπιση ενός προβλήματος** περιλαμβάνει τα εξής στάδια:

1. Εντοπισμός / προσδιορισμός του προβλήματος
2. Αναλυτική διατύπωση του προβλήματος
3. Ανάλυση και σχεδίαση της επίλυσης του προβλήματος
4. Εφαρμογή της διαδικασίας επίλυσης
5. Επαλήθευση ότι το πρόβλημα έχει αντιμετωπιστεί επιτυχώς
6. Καταγραφή εμπειρίας για μελλοντική χρήση

Δραστηριότητα 1.1

Αν και δεν φαίνεται από την πρώτη ματιά, και τα έξι στάδια είναι απαραίτητα. Συμφωνείτε με τον ισχυρισμό αυτό; Προσπαθήστε να τεκμηριώσετε την άποψή σας σε 150 περίπου λέξεις, πριν διαβάσετε το κείμενο που ακολουθεί. Εάν η απάντησή σας διαφέρει σημαντικά, τότε καλύτερα να τη συζητήσετε με το διδάσκοντα της Θεματικής Ενότητας.

Συνηθίζουμε να επιλύουμε τα καθημερινά μας προβλήματα χωρίς να ακολουθούμε συνειδητά τα έξι στάδια επίλυσης προβλημάτων που αναφέρθηκαν πιο πάνω. Όμως όλα τους είναι απαραίτητα για την ορθή και πλήρη (και όχι προσωρινή) επίλυση ενός προβλήματος. Για παράδειγμα, η **αναλυτική διατύπωση** πρέπει να ακολουθήσει τον εντοπισμό του προβλήματος, ώστε να διασφαλίσουμε ότι έχουμε κατανοήσει πλήρως όλες τις διαστάσεις του προβλήματος. Συνήθως, αυτό απαιτεί τη διάσπαση του προβλήματος σε μικρότερα και πιό απλά προβλήματα, ώστε λύνοντας ένα προς ένα τα μικρότερα προβλήματα να μπορούμε να συνθέσουμε τη λύση του αρχικού προβλήματος. Εάν η σύνθεση της τελικής λύσης από τις λύσεις των επί μέρους προβλημάτων δεν είναι εφικτή, αυτό σημαίνει είτε ότι δεν έχουμε επιτύχει τη σωστή διάσπαση, είτε ότι το πρόβλημα δεν είναι αρκετά σύνθετο ώστε να διασπαστεί.

Η **διαδικασία επίλυσης** πρέπει πρώτα να **σχεδιαστεί** και έπειτα να **εφαρμοστεί**, για να αποφύγουμε την εφαρμογή μιας πρόχειρης και μη

αποτελεσματικής διαδικασίας (σε πολλές περιπτώσεις δεν έχουμε καν τη δυνατότητα να δοκιμάσουμε μια δεύτερη διαδικασία επίλυσης). Συνήθως, βασίζεται στο σύνολο των πληροφοριών που έχουν συλλεχθεί για το πρόβλημα, από τις οποίες επιδιώκεται η επίτευξη των αποτελεσμάτων που αποτελούν τη λύση του προβλήματος. Καλό είναι, λύνοντας ένα πρόβλημα να φροντίζουμε να αντιμετωπίσουμε μια ολόκληρη κατηγορία παρόμοιων προβλημάτων. Πολλές φορές, αυτό απαιτεί διευκρινίσεις επί των αρχικών πληροφοριών, οπότε επαναλαμβάνονται κάποιες ενέργειες ανάλυσης του προβλήματος. Τέλος, σημαντικό ρόλο παίζει και ο τρόπος ή το μέσο εφαρμογής της διαδικασίας επίλυσης.

Μετά την εφαρμογή της διαδικασίας επίλυσης, πρέπει να **εξετάσουμε εάν πραγματικά λύθηκε το πρόβλημα**. Στην πραγματικότητα πρέπει να εξετάσουμε εάν το πρόβλημα λύθηκε σωστά και αν έχουμε λύσει το σωστό πρόβλημα. Εάν αυτό έχει συμβεί, καλό είναι με κάποιον τρόπο να **καταγράψουμε την αποκτηθείσα εμπειρία** (π.χ με καταγραφή και τεκμηρίωση της διαδικασίας επίλυσης), ώστε να διευκολυνθούμε στην αντιμετώπιση του ίδιου προβλήματος εάν αυτό παρουσιαστεί ξανά στο μέλλον. Πρέπει να έχουμε υπόψη μας ότι ένα πρόβλημα δεν παραμένει στατικό, αλλά συνήθως εξελίσσεται μέσα στο χρόνο (γι' αυτό και δεν μπορούμε να διασφαλίσουμε ότι μια μορφή του ίδιου προβλήματος δε θα μας απασχολήσει ξανά στο μέλλον). Εάν όμως το πρόβλημα δεν έχει λυθεί, πρέπει να διαγνώσουμε τι δεν πήγε καλά (ένα άλλο πρόβλημα από μόνο του) και ανάλογα να επαναλάβουμε τα βήματα αναλυτικής περιγραφής, σχεδίασης κλπ.

1.2 Αντιμετώπιση προβλημάτων με τη χρήση υπολογιστή

Τρία από τα θέματα που αναφέρθηκαν στην προηγούμενη ενότητα, έχουν ξεχωριστό ενδιαφέρον σε σχέση με τους στόχους του παρόντος κεφαλαίου: η περιγραφή του προβλήματος, η διαδικασία επίλυσης και το μέσο που θα επιλεγεί για την υλοποίησή της.

Αν και καθημερινά αντιμετωπίζουμε και επιλύουμε πολυάριθμα προβλήματα με διάφορους τρόπους και τεχνικές, στο παρόν κεφάλαιο μας ενδιαφέρουν μόνο εκείνα τα προβλήματα που μπορούν να επιλυθούν με τη χρήση ηλεκτρονικού υπολογιστή. Για να επιλύσουμε τέτοια προβλήματα, απαιτείται πρώτα η συλλογή των δεδομένων του προβλή-

ματος, έπειτα η περιγραφή της διαδικασίας επίλυσης και τέλος ο προγραμματισμός των υπολογιστών για να εκτελέσουν συγκεκριμένα καθήκοντα. Η περιγραφή της διαδικασίας επίλυσης ενός προβλήματος καλείται **αλγόριθμος**. Ο μετασχηματισμός του αλγορίθμου σε μορφή κατανοητή από έναν υπολογιστή καλείται **προγραμματισμός**. Το αποτέλεσμα του προγραμματισμού είναι η **ανάπτυξη προγραμμάτων**, τα οποία αποτελούν το **λογισμικό (software)** των υπολογιστών. Ένα πρόγραμμα αποτελείται από ένα σύνολο **εντολών (commands)**, οι οποίες αποτελούν «οδηγίες» προς τον υπολογιστή για την εκτέλεση συγκεκριμένων ενεργειών.

- Συνοψίζοντας, λοιπόν, μπορούμε να πούμε ότι για να επιλύσουμε ένα πρόβλημα με τη χρήση υπολογιστή πρέπει πρώτα να σχεδιάσουμε τον αλγόριθμο, αφού λάβουμε υπόψη μας όλα τα δεδομένα του προβλήματος και, ακολούθως, να τον μετασχηματίσουμε σε ένα πρόγραμμα υπολογιστή αποτελούμενο από εντολές, το οποίο θα επεξεργάζεται τα δεδομένα του προβλήματος για να παράγει τη λύση του.

Δραστηριότητα 1.2

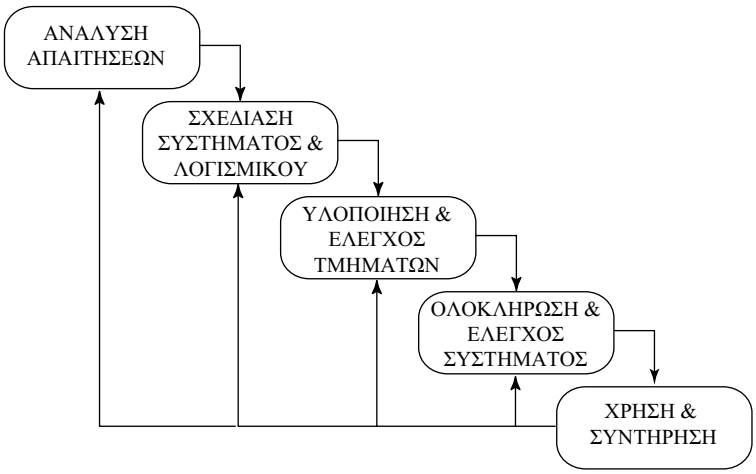
Η διαδικασία ανάπτυξης λογισμικού περιλαμβάνει όλα τα στάδια αντιμετώπισης ενός προβλήματος που περιγράφηκαν στην προηγούμενη ενότητα (στο κάτω–κάτω, η ανάπτυξη λογισμικού είναι ένα αρκετά σύνθετο πρόβλημα) και αποτελεί μέρος της Τεχνολογίας Λογισμικού (Software Engineering).

Προσπαθήστε να ανατρέξετε σε βιβλία της Τεχνολογίας Λογισμικού (μερικά καλά βιβλία προτείνονται στο τέλος του βιβλίου αυτού) και να καταγράψετε διάφορους τρόπους με τους οποίους αναφέρονται και περιγράφονται τα διάφορα στάδια ανάπτυξης λογισμικού (συνήθως αναφέρονται σαν «κύκλος ζωής λογισμικού»). Μια ενδεικτική απάντηση δίνεται στη συνέχεια.

Εάν ανατρέξει κανείς σε βιβλία της Τεχνολογίας Λογισμικού, θα συναντήσει διάφορους τρόπους υλοποίησης της διαδικασίας επίλυσης του προβλήματος της ανάπτυξης λογισμικού (κάθε τέτοιος τρόπος συνήθως λέγεται «**μοντέλο κύκλου ζωής λογισμικού**»). Κάθε τέτοιο μοντέλο περιλαμβάνει στάδια όπως: ανάλυση απαιτήσεων, σχεδίαση,

υλοποίηση λογισμικού, επαλήθευση και επικύρωση κ.ά. Είναι φανερή η άμεση αντιστοιχία των σταδίων αυτών με τα βήματα επίλυσης ενός προβλήματος.

Οι τεχνικές προγραμματισμού που θα περιγραφούν σε αυτό το μάθημα χρησιμοποιούνται σε όλες τις φάσεις που περιλαμβάνουν δραστηριότητες υλοποίησης λογισμικού. Για παράδειγμα, στο Σχήμα 1.1 φαίνεται το μοντέλο κύκλου ζωής που είναι γνωστό ως «καταρράκτης». Στις φάσεις «Υλοποίηση και έλεγχος τμημάτων», «Ολοκλήρωση και έλεγχος συστήματος» και «Συντήρηση» περιλαμβάνονται δραστηριότητες προγραμματισμού. Προηγούνται οι φάσεις της ανάλυσης, όπου συλλέγονται τα δεδομένα του προβλήματος, και της σχεδίασης, όπου περιγράφεται η διαδικασία επίλυσης του προβλήματος (Sommerville, 1996). Το παρόν βιβλίο θα αναφερθεί στο στάδιο σχεδίασης ενός προγράμματος υπολογιστή, αφού πρώτα παρουσιαστούν οι πιο διαδεδομένες πρακτικές προγραμματισμού.



Σχήμα 1.1

Η θέση των δραστηριοτήτων υλοποίησης λογισμικού στο μοντέλο ανάπτυξης λογισμικού «καταρράκτης»

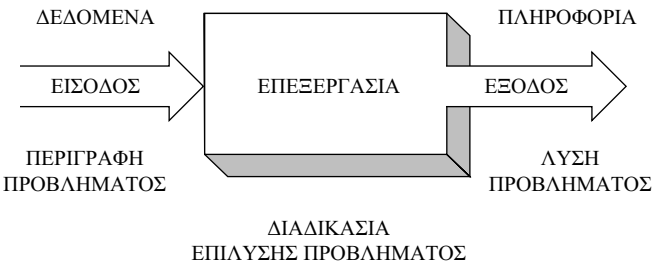
1.2.1 Ο υπολογιστής ως σύστημα επεξεργασίας δεδομένων

Ένα κατασκεύασμα το οποίο επεξεργάζεται εισόδους για να παράγει εξόδους καλείται «**σύστημα**». Αφού ο υπολογιστής δέχεται **δεδομένα** στην είσοδο για να παράγει **πληροφορίες** στην έξοδο, καλείται «**σύστημα επεξεργασίας δεδομένων**».

Συνεπώς, ένας απλός τρόπος αναπαράστασης της λειτουργίας ενός υπολογιστή είναι σαν ένα κουτί, το οποίο έχει εισόδους και εξόδους

(Σχήμα 1.2). Στις **εισόδους (inputs)**, οι χρήστες του υπολογιστή του παρέχουν τις πληροφορίες που χρειάζεται (πρόκειται για τα δεδομένα του προβλήματος). Στις **εξόδους (outputs)**, ο υπολογιστής παράγει την απάντηση στο πρόβλημα που του τέθηκε.

Μέσα στο κουτί συντελείται η **επεξεργασία (processing)** των δεδομένων ώστε να παραχθεί η πληροφορία εξόδου. Ουσιαστικά πρόκειται για εφαρμογή των μεθόδων επίλυσης του προβλήματος που «γνωρίζει» από πριν ο υπολογιστής. Στο σημείο αυτό πρέπει να παρατηρήσουμε ότι η μέθοδος επίλυσης είναι ανεξάρτητη από τα δεδομένα ενός προβλήματος: η ίδια μέθοδος επίλυσης μπορεί να χρησιμοποιηθεί πολλές φορές με διαφορετικά δεδομένα και ενδεχομένως να δώσει και διαφορετική πληροφορία εξόδου. Η έξοδος όμως του συστήματος δεν είναι ανεξάρτητη από την είσοδο, αλλά στην ουσία αποτελεί μετασχηματισμό της εισόδου. Συνεπώς, ο ρόλος της εισόδου, αλλά και της επεξεργασίας, είναι κρίσιμος για την ορθότητα της εξόδου.



Σχήμα 1.2
Ο υπολογιστής ως σύστημα επεξεργασίας δεδομένων

Δραστηριότητα 1.3

Όλοι έχετε δει υπολογιστές και βέβαια δε μοιάζουν καθόλου με το Σχήμα 1.2. Προσπαθήστε να αναφέρετε τουλάχιστον οκτώ (8) τμήματα των υπολογιστών που αντιστοιχούν στα τμήματα «είσοδος», «επεξεργασία» και «έξοδος» του Σχήματος 1.2. Μια ενδεικτική απάντηση δίνεται στη συνέχεια.

Ένας υπολογιστής αποτελείται από διάφορες συσκευές ή τμήματα, τα οποία επιτελούν τις λειτουργίες εισόδου, επεξεργασίας και εξόδου. Παραδείγματα τέτοιων συσκευών είναι:

Εισόδου: πληκτρολόγιο, ποντίκι, φωτογραφίδα, μικρόφωνο, σαρωτής, οθόνη αφής, κάρτα δικτύου κ.ά.

Εξόδου: οθόνη, εκτυπωτής, σχεδιογράφος, ηχεία, κάρτα δικτύου κ.ά.

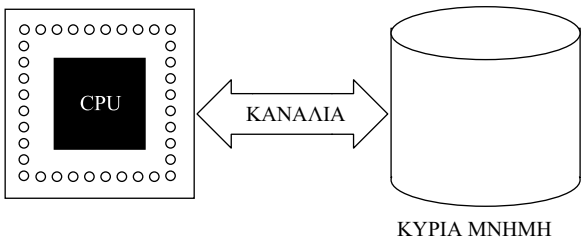
Επεξεργασίας: κεντρική μονάδα επεξεργασίας, επεξεργαστής γραφικών κ.ά.

Ακόμη, υπάρχουν και κάποιες συσκευές, οι οποίες λειτουργούν είτε ως συσκευές εισόδου, είτε ως συσκευές εξόδου. Πρόκειται για τις περιφερειακές μονάδες αποθήκευσης, δηλαδή τη μαγνητική ταινία, τη δισκέτα, το σκληρό δίσκο και το CD.

1.2.2 Ο υπολογιστής ως μηχανή εκτέλεσης προγραμμάτων

Στο Σχήμα 1.3 φαίνονται αναλυτικότερα τα συστατικά μέρη του τμήματος επεξεργασίας ενός υπολογιστή. Εκτός από τις μονάδες εισόδου και εξόδου, ένας υπολογιστής αποτελείται από:

- την **κεντρική μονάδα επεξεργασίας (central processing unit – CPU)** ή **επεξεργαστή (processor)**, η οποία εκτελεί τις πράξεις πάνω στα δεδομένα εισόδου που υπαγορεύονται από τις εντολές του προγράμματος και παράγει τα δεδομένα εξόδου
- την **κύρια μνήμη (main memory)**, στην οποία αποθηκεύονται τα δεδομένα εισόδου και εξόδου, καθώς και οι εντολές του προγράμματος
- τα **κανάλια διακίνησης πληροφοριών (buses)**, τα οποία είναι στην πραγματικότητα αγώγιμοι δρόμοι που συνδέουν τα διάφορα τμήματα του υπολογιστή, ώστε με την ανταλλαγή ηλεκτρικών σημάτων να είναι δυνατή η επικοινωνία τους.



Σχήμα 1.3
Ανάλυση του τμήματος επεξεργασίας ενός υπολογιστή

Τώρα που γνωρίζετε σε γενικές γραμμές την εσωτερική δομή ενός υπολογιστή, θα μπορέσετε πιο εύκολα να κατανοήσετε τον τρόπο που αυτός εκτελεί τις εντολές ενός προγράμματος (Tanenbaum, 1984). Να

θυμάστε ότι ένας επεξεργαστής μπορεί να εκτελεί μία εντολή κάθε φορά. Συνεπώς, οι σύγχρονοι υπολογιστές που αποτελούνται από έναν επεξεργαστή εκτελούν μία προς μία τις εντολές ενός προγράμματος, με τη σειρά που αυτές είναι γραμμένες (γι' αυτό λέγονται και ακολουθιακοί υπολογιστές).

Όπως έχουμε αναφέρει, τόσο οι εντολές του προγράμματος, όσο και τα δεδομένα που αυτό θα χρησιμοποιήσει, βρίσκονται αποθηκευμένα στην κύρια μνήμη (εάν βρίσκονται σε κάποια βοηθητική συσκευή αποθήκευσης, π.χ. στο σκληρό δίσκο, θα πρέπει πρώτα να μεταφερθούν στην κύρια μνήμη). Ο επεξεργαστής κάθε φορά αποθηκεύει εσωτερικά τη διεύθυνση της θέσης μνήμης στην οποία βρίσκεται η επόμενη προς εκτέλεση εντολή. Η διαδικασία εκτέλεσης μιας εντολής περιλαμβάνει τα εξής πέντε βήματα:

Βήμα 1. Ανάκληση επόμενης εντολής από τη μνήμη

Βήμα 2. Ερμηνεία εντολής και ανάκληση δεδομένων

Βήμα 3. Εκτέλεση εντολής

Βήμα 4. Μεταφορά του αποτελέσματος της εκτέλεσης στη μνήμη

Βήμα 5. Εάν υπάρχουν και άλλες εντολές στο πρόγραμμα, επαναλαμβάνεται το Βήμα 1.

Για παράδειγμα, έστω ότι ο υπολογιστής ανακαλεί μια εντολή της μορφής ADD (200), (201) (οι αριθμοί σε παρένθεση αποτελούν διευθύνσεις μνήμης). Στο βήμα 2, η εντολή αποκωδικοποιείται (πρόκειται για εντολή πρόσθεσης δύο αριθμών) και διαβάζονται τα περιεχόμενα των θέσεων μνήμης 200 και 201. Έπειτα ο επεξεργαστής προσθέτει τους δύο αριθμούς και στο βήμα 4 γράφει το αποτέλεσμα στην πρώτη κενή θέση μνήμης (π.χ. 202).

Σύνοψη

Στο πρώτο κεφάλαιο του βιβλίου είδαμε ότι η κυριότερη εφαρμογή του υπολογιστή είναι ως μηχανή επίλυσης προβλημάτων. Επειδή η ανάγκη επίλυσης προβλημάτων είναι γενικότερη, έχουν αναπτυχθεί γενικές διαδικασίες που βοηθούν προς τον σκοπό αυτό. Όμως, ο υπολογιστής μπορεί να εφαρμοστεί σε ειδικές κατηγορίες προβλημάτων, εξαιτίας της ικανότητάς του να λειτουργεί ως ένα σύστημα επεξεργασίας δεδομένων. Ένα τέτοιο σύστημα δέχεται στην είσοδό του τα δεδομένα ενός προβλήματος, τα οποία επεξεργάζεται και παράγει στην έξοδο τη λύση του.

Η επεξεργασία που θα εφαρμόσει στα δεδομένα περιγράφεται στον υπολογιστή με τη χρήση προγραμμάτων, τα οποία αποτελούνται από εντολές. Τα προγράμματα αποτελούν ένα συγκεκριμένο τρόπο έκφρασης της διαδικασίας επίλυσης ενός προβλήματος, η οποία καλείται αλγόριθμος. Ο υπολογιστής χρησιμοποιεί την κεντρική μονάδα επεξεργασίας για να εκτελέσει με καθορισμένο τρόπο καθεμία εντολή ενός προγράμματος.

Πριν όμως προχωρήσουμε στην ανάπτυξη ενός προγράμματος, καλό είναι να σχεδιάσουμε τα βήματα της διαδικασίας επίλυσης που θα ακολουθηθεί. Η περιγραφή της διαδικασίας αυτής γίνεται με τη βοήθεια των αλγορίθμων, οι οποίοι παρουσιάζονται στο επόμενο κεφάλαιο.

Βιβλιογραφία Κεφαλαίου 1

[1] Adams, D. (1979), *The Hitch–hiker's guide to the Galaxy*. Ελλ. Μετ. εκδ. Ars–Longa.

[2] Sommerville, I. (1996), *Software Engineering*. Addison–Wesley.

[3] Tanenbaum, A. (1984), *Structured Computer Organization*. Prentice Hall: New Jersey

ΚΕΦΑΛΑΙΟ 2

Αλγόριθμοι

Σκοπός

Το δεύτερο κεφάλαιο του βιβλίου περιγράφει την έννοια των αλγορίθμων, τις ιδιότητές τους και τους πιο διαδεδομένους τρόπους αναπαράστασης αυτών. Το κεφάλαιο αυτό είναι βασικό για την κατανόηση του τρόπου σχεδίασης προγραμμάτων γι' αυτό και περιλαμβάνει πολλά παραδείγματα και δραστηριότητες.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε διαβάσει αυτό το κεφάλαιο, θα μπορείτε να:

- *εξηγήσετε τι είναι ένας αλγόριθμος επίλυσης ενός προβλήματος*
- *αναφέρετε τα πέντε χαρακτηριστικά ενός αλγορίθμου*
- *περιγράψετε έναν αλγόριθμο χρησιμοποιώντας λεκτική ή συμβολική (γραφική) αναπαράσταση (δηλαδή ψευδοκώδικα ή διαγράμματα ροής προγράμματος)*

Έννοιες κλειδιά

- *Αλγόριθμος*
- *Ψευδοκώδικας*
- *Διάγραμμα Ροής Προγράμματος*

Εισαγωγικές Παρατηρήσεις

Έχουμε ήδη αναφέρει στο κεφάλαιο 1 του βιβλίου ότι η περιγραφή της διαδικασίας επίλυσης ενός προβλήματος καλείται ***αλγόριθμος****. Η περιγραφή αυτή είναι κατανοητή από τον άνθρωπο μόνο (οι υπολογιστές δεν αναγνωρίζουν τους αλγορίθμους) και ανεξάρτητη από κάποιο συγκεκριμένο πρόβλημα (δηλαδή, αναφέρεται σε μια κατηγορία προβλημάτων). Για παράδειγμα, εάν θέλουμε να βρούμε τις ρίζες του τριωνύμου* $x^2 - 3x + 2$*, θα κατασκευάσουμε έναν αλγόριθμο που θα βρίσκει τις ρίζες οποιουδήποτε τριωνύμου.*

Στο κεφάλαιο αυτό θα ασχοληθούμε ιδιαίτερα με τους αλγορίθμους. Αρχικά, στην ενότητα 2.1 περιγράφονται οι ιδιότητες των αλγορίθμων. Στην ενότητα 2.2 παρουσιάζεται ο ψευδοκώδικας ως μέσο λεκτικής

αναπαράστασης των αλγορίθμων. Ταυτόχρονα, εισάγονται η έννοια της οδηγίας, οι βασικές οδηγίες εισόδου/εξόδου και οι τρεις βασικές προγραμματιστικές δομές (ακολουθία, απόφαση, επανάληψη) μέσα από πολυάριθμα παραδείγματα διαφορετικού βαθμού πολυπλοκότητας. Στην ενότητα 2.3 παρουσιάζονται τα διαγράμματα ροής προγράμματος ως η πιο διαδεδομένη γραφική τεχνική σχεδίασης και αναπαράστασης αλγορίθμων. Για την ευκολότερη κατανόηση των αλγορίθμων που παρουσιάζονται, θα ήταν καλό να είστε εξοικειωμένοι με βασικές δομές αναπαράστασης δεδομένων, όπως π.χ. είναι οι πίνακες. Τέτοια ζητήματα αποτελούν το αντικείμενο της Θ.Ε. «Δομές Δεδομένων», στο βιβλίο της οποίας σας συνιστώ να ανατρέξετε εάν συναντήσετε δυσκολίες στην κατανόηση των αλγορίθμων. Μια σύντομη παρουσίασή τους θα γίνει και στο κεφάλαιο 6 του παρόντος τόμου.

2.1 Πρόβλημα και αλγόριθμος επίλυσης

Στην πραγματικότητα, εμείς οι άνθρωποι χρησιμοποιούμε συνεχώς αλγορίθμους για την επίλυση προβλημάτων στην καθημερινή μας ζωή. Όταν τα προβλήματα είναι απλά και τα συναντούμε συνεχώς, δε χρειάζεται να σχεδιάσουμε τους αλγόριθμους που χρησιμοποιούμε. Έτσι, όταν διψούμε, μας είναι πολύ εύκολο να λύσουμε το πρόβλημα πίνοντας ένα ποτήρι νερό, χωρίς να χρειάζεται να σχεδιάσουμε όλες τις επί μέρους κινήσεις μας (κάποιες από τις οποίες, εάν το καλοσκεφτεί κανείς, είναι αρκετά πολύπλοκες). Για την επίλυση σύνθετων προβλημάτων, όμως, είναι απαραίτητο να σχεδιάσουμε τα βήματα που θα ακολουθήσουμε.

- Η επίλυση κάθε προβλήματος απαιτεί την εκτέλεση μιας σειράς ενεργειών, οι οποίες δίνουν στο τέλος ένα συγκεκριμένο αποτέλεσμα. Η επιτυχής επίλυση του προβλήματος δεν εξαρτάται μόνο από την εφαρμογή σωστών βημάτων στη σωστή σειρά, αλλά και από την εκμετάλλευση όλων των πληροφοριών που μας έχουν δοθεί για το πρόβλημα.

Αν και η χρησιμότητα μιας τέτοιας διαδικασίας επίλυσης προβλημάτων φαίνεται να είναι μεγαλύτερη σε προβλήματα θετικών επιστημών ή μηχανικής, στην πραγματικότητα μπορεί να εφαρμοστεί σε οποιοδήποτε πρόβλημα.

Δραστηριότητα 2.1

Πριν προχωρήσετε, προσπαθήστε να περιγράψετε ένα πρόβλημα θετικών επιστημών ή μηχανικής και τον αντίστοιχο αλγόριθμο που χρησιμοποιείτε για να το λύσετε. Μπορείτε να κάνετε το ίδιο για ένα πρόβλημα της καθημερινής σας ζωής;

Όσον αφορά τις θετικές επιστήμες, ας προσπαθήσουμε να περιγράψουμε τη λύση του προβλήματος της εύρεσης του Μέγιστου Κοινού Διαιρέτη (ΜΚΔ) δύο θετικών ακεραίων Χ και Ψ (ο ΜΚΔ δύο αριθμών Χ και Ψ είναι ο μεγαλύτερος από τους θετικούς ακέραιους που διαιρούν ακριβώς και τους δύο αριθμούς). Η επιλογή του προβλήματος δεν είναι τυχαία: μέχρι το 1950, η λέξη «αλγόριθμος» τις περισσότερες φορές αναφερόταν στον «αλγόριθμο του Ευκλείδη», μια διαδικασία επίλυσης αυτού ακριβώς του προβλήματος που περιγράφει ο

Ευκλείδης στο περίφημο βιβλίο του «Στοιχεία». Ο αλγόριθμος του Ευκλείδη περιγράφεται είναι ο ακόλουθος:

Βήμα 1. ΕΣΤΩ ζ = ΥΠΟΛΟΙΠΟ (χ/ψ)

Βήμα 2. ΕΑΝ ζ = 0, ΤΟΤΕ

Βήμα 2.1 ΜΚΔ = ψ. ΤΕΡΜΑΤΙΣΜΟΣ

Βήμα 3. ΑΛΛΙΩΣ ΕΠΑΝΑΛΗΨΗ ΑΠΟ βήμα 1 ΜΕ (χ = ψ ΚΑΙ ψ = ζ)

Από την καθημερινή ζωή, διαλέγουμε το παράδειγμα της χρήσης Αυτόματης Μηχανής Συναλλαγών σε μια τράπεζα. Τα βήματα που πρέπει να ακολουθήσουμε για μια απλή συναλλαγή περιγράφονται στη συνέχεια:

Βήμα 1. ΕΙΣΑΓΩΓΗ κάρτας συναλλαγών

Βήμα 2. ΕΙΣΑΓΩΓΗ κωδικού χρήστη

Βήμα 3. ΕΠΙΛΟΓΗ συναλλαγής ΑΠΟ σύνολο συναλλαγών

Βήμα 4. ΕΑΝ συναλλαγή = ανάληψη ΤΟΤΕ

Βήμα 4.1 ΕΙΣΑΓΩΓΗ ποσού ανάληψης

Βήμα 4.2 ΠΑΡΑΛΑΒΗ χρημάτων

Βήμα 4.3 ΠΑΡΑΛΑΒΗ απόδειξης

Βήμα 5. ΑΛΛΙΩΣ ΕΑΝ συναλλαγή= κατάθεση ΤΟΤΕ

Βήμα 5.1 ΕΙΣΑΓΩΓΗ ποσού κατάθεσης

Βήμα 5.2 ΑΠΟΘΕΣΗ χρημάτων

Βήμα 5.3 ΠΑΡΑΛΑΒΗ απόδειξης

Βήμα 6. ΠΑΡΑΛΑΒΗ κάρτας συναλλαγών. ΤΕΡΜΑΤΙΣΜΟΣ

2.2 Χαρακτηριστικά των αλγορίθμων

Από όσα αναφέρθηκαν στην ενότητα 2.1 μπορούμε τώρα να εξάγουμε τα κυριότερα χαρακτηριστικά των αλγορίθμων. Ενας αλγόριθμος, λοιπόν, είναι η περιγραφή της διαδικασίας επίλυσης ενός προβλήματος. Αποτελείται από ξεχωριστά και αυτόνομα βήματα, τα οποία πρέπει να εκτελεστούν ακολουθιακά με μια συγκεκριμένη σειρά. Ο αριθμός των βημάτων είναι πεπερασμένος. Κάθε βήμα αρχίζει, εκτελείται

με συγκεκριμένο τρόπο και τελειώνει μετά από καθορισμένο χρόνο. Συνεπώς, κάθε αλγόριθμος αρχίζει και τελειώνει: δεν υπάρχουν αλγόριθμοι που εκτελούνται ατελείωτα.

Κάθε αλγόριθμος χρησιμοποιεί ως είσοδο τα δεδομένα του προβλήματος και παράγει ως έξοδο τη λύση του προβλήματος. Δεν υπάρχουν αλγόριθμοι που παράγουν έξοδο χωρίς να χρησιμοποιήσουν δεδομένα εισόδου, ούτε υπάρχουν αλγόριθμοι που δεν παράγουν έξοδο. Ο αλγόριθμος είναι ανεξάρτητος των δεδομένων: κάθε αλγόριθμος επιλύει μια ολόκληρη κλάση προβλημάτων, όχι ένα πρόβλημα μόνο.

Για κάθε πρόβλημα υπάρχουν σωστοί και λάθος αλγόριθμοι επίλυσης (όπως καταλαβαίνετε, οι τελευταίοι δεν λύνουν το πρόβλημα). Ένας αλγόριθμος είναι σωστός όταν κάθε βήμα του είναι σωστό και τα βήματα έχουν διαταχθεί στη σωστή σειρά. Οι λάθος αλγόριθμοι είτε δεν λύνουν σωστά το πρόβλημα (δηλαδή κάποια από τα βήματα τους είναι λάθος), είτε δεν λύνουν το σωστό πρόβλημα. Στην πρώτη περίπτωση, ο αλγόριθμος έχει λεκτικά λάθη και για να τα διορθώσουμε αρκεί να εξετάσουμε εάν κάθε βήμα είναι σωστό και εάν η ακολουθία εκτέλεσης είναι η σωστή. Στη δεύτερη περίπτωση, ο αλγόριθμος έχει λογικά λάθη. Η διόρθωση τέτοιων λαθών είναι αρκετά δύσκολη και απαιτεί επανασχεδιασμό του αλγορίθμου.

Ανάμεσα στους σωστούς αλγορίθμους επίλυσης ενός προβλήματος, κάποιοι είναι καλύτεροι από τους άλλους. Μέτρο της ποιότητας ενός αλγορίθμου αποτελεί συνήθως η αποδοτικότητα (efficiency) η οποία αναφέρεται στην ποσότητα του χώρου και του χρόνου που απαιτεί η εκτέλεση του αλγορίθμου, ανεξάρτητα από το πρόγραμμα που τον υλοποιεί. Ένας αποδοτικός αλγόριθμος περιλαμβάνει όσο το δυνατόν λιγότερα βήματα εκτέλεσης και χρησιμοποιεί την ελάχιστη απαραίτητη μνήμη.

Δραστηριότητα 2.2

Πριν προχωρήσετε, προσπαθήστε να συνοψίσετε τα κυριότερα χαρακτηριστικά των αλγορίθμων. Έπειτα διαβάστε το πλαίσιο κειμένου που ακολουθεί.

- Τα πέντε κυριότερα χαρακτηριστικά ενός σωστού αλγόριθμου συνοψίζονται στα εξής (Knuth, 1973):

  Είναι πεπερασμένος. Αυτό δεν σημαίνει μόνο ότι ένας αλγόριθμος αποτελείται από πεπερασμένο αριθμό βημάτων, αλλά και ότι ένας αλγόριθμος πρέπει πάντα να τερματίζει μετά από ένα πεπερασμένο αριθμό βημάτων.

  Είναι καλά ορισμένος. Κάθε βήμα ενός αλγορίθμου πρέπει να είναι επακριβώς ορισμένο και οι ενέργειες που θα εκτελεστούν σε αυτό πρέπει να περιγράφονται ρητά και αδιαμφισβήτητα.

  Είσοδος. Κάθε αλγόριθμος δέχεται μία ή περισσότερες εισόδους. Πρόκειται για δεδομένα που εισέρχονται σε αυτόν πριν αρχίσουν οι κυρίως υπολογισμοί.

  Εξοδος. Κάθε αλγόριθμος παράγει μία ή περισσότερες εξόδους. Πρόκειται για πληροφορίες που έχουν μια καθορισμένη σχέση με τις εισόδους.

  Αποτελεσματικότητα. Για να χαρακτηριστεί ένας αλγόριθμος ως αποτελεσματικός, πρέπει όλες οι ενέργειες που περιλαμβάνει να είναι αρκετά απλές ώστε να μπορούν κατ' αρχήν να εκτελεστούν με ακρίβεια σε πεπερασμένο χρόνο από έναν άνθρωπο που χρησιμοποιεί χαρτί και μολύβι.

Μια γενικά αποδεκτή δομή του αλγορίθμου τον χωρίζει σε τρία τμήματα (Μπεμ, 1991):

- την **κεφαλή (header)** του αλγόριθμου, η οποία περιλαμβάνει το όνομα του αλγόριθμου και τον κατάλογο των δεδομένων εισόδου / εξόδου. Δηλώνεται με τη λέξη ΑΛΓΟΡΙΘΜΟΣ
- το **τμήμα δηλώσεων των δεδομένων** (δηλώνεται με τη λέξη ΔΕΔΟΜΕΝΑ) με τα οποία εκτελούνται οι πράξεις που περιγράφει ο αλγόριθμος. Το τμήμα αυτό περιλαμβάνει τα δεδομένα εκείνα που ο αλγόριθμος χρησιμοποιεί εσωτερικά. Αυτά διαφέρουν από τα δεδομένα εισόδου / εξόδου
- το **κυρίως τμήμα του αλγορίθμου (body)**, στο οποίο περιγράφονται οι πράξεις που εκτελούνται πάνω στα δεδομένα. Η αρχή του τμήματος δηλώνεται με τη λέξη ΑΡΧΗ, ενώ το τέλος του με τη λέξη ΤΕΛΟΣ.

2.3 Αναπαράσταση αλγορίθμων με ψευδοκώδικα

Ένας αλγόριθμος, εκτός από τρόπος περιγραφής της διαδικασίας επίλυσης ενός προβλήματος, αποτελεί το βασικό εργαλείο επικοινωνίας μεταξύ των ατόμων που προσπαθούν να λύσουν το ίδιο πρόβλημα ή να κατανοήσουν τη λύση του. Από τον αλγόριθμο θα προκύψουν οι εντολές του προγράμματος που θα επιλύει το πρόβλημα με τη χρήση υπολογιστή.

- Ως εργαλείο επικοινωνίας, ο αλγόριθμος πρέπει να έχει συγκεκριμένη δομή, να εκφράζεται σε μορφή κατανοητή από όλους τους πιθανούς αποδέκτες και σε καθορισμένη σύνταξη ώστε να μην είναι διφορούμενη η ερμηνεία του.

Διακρίνουμε δύο μεγάλες κατηγορίες εργαλείων περιγραφής αλγορίθμων: τα εργαλεία που βασίζονται στη γλώσσα και αυτά που βασίζονται σε σχήματα. Το σημαντικότερο εργαλείο **λεκτικής περιγραφής** αλγορίθμων είναι ο **ψευδοκώδικας (pseudocode)**. Εάν ερμηνεύσουμε το όνομά του, ο ψευδοκώδικας είναι κάτι που μοιάζει με κώδικα προγραμματισμού, αλλά δεν είναι. Αληθώς, ο ψευδοκώδικας είναι μια δομημένη γλώσσα που χρησιμοποιεί στοιχεία (συντακτικές δομές) από τις γλώσσες προγραμματισμού, συμβολισμούς (σημασιολογικά στοιχεία) από τα Μαθηματικά και τη Μαθηματική Λογική και στοιχεία (λεκτικές περιγραφές) από τη φυσική γλώσσα. Αποτελεί την υλοποίηση της πρότασης για την ανάπτυξη ενός εργαλείου περιγραφής αλγορίθμων που θα συνδυάζει ταυτόχρονα την αυστηρότητα και την ακρίβεια που έχουν οι γλώσσες προγραμματισμού με την εκφραστική δύναμη της φυσικής γλώσσας (Davies, 1983, Σκορδαλάκης, 1991).

Ο ψευδοκώδικας είναι ανεξάρτητος από οποιαδήποτε γλώσσα προγραμματισμού. Όμως, ένας αλγόριθμος που περιγράφεται με ψευδοκώδικα μπορεί άμεσα να «μεταφραστεί» σε ένα πρόγραμμα (άρα, όπως θα αναμένατε, ένας τέτοιος αλγόριθμος αποτελεί ένα «ομοίωμα» του τελικού προγράμματος!).

Το μεγάλο πλεονέκτημα της περιγραφής ενός αλγορίθμου με ψευδοκώδικα είναι ότι έτσι γίνεται κατανοητός τόσο από προγραμματιστές όσο και από μη προγραμματιστές. Επιπλέον, οι πρώτοι μπορούν άμεσα να υλοποιήσουν το πρόγραμμα από την περιγραφή του αλγορίθμου.

Άλλα πλεονεκτήματα της χρήσης ψευδοκώδικα είναι η εύκολη εισαγωγή μεγαλύτερης λεπτομέρειας στην περιγραφή του αλγορίθμου (μια πρακτική προγραμματισμού που εκμεταλλεύεται αυτή τη δυνατότητα είναι η Κατά Βήμα Εκλέπτυνση που περιγράφεται στην ενότητα 3.1) και η φθηνή παραγωγή τέτοιων περιγραφών, αφού απαιτείται μόνο ένας επεξεργαστής κειμένου και ένας εκτυπωτής. Μειονεκτήματα αποτελούν οι περιορισμένες εκφραστικές δυνατότητες (σε σύγκριση με τα διαγράμματα ροής που περιγράφονται στην αμέσως επόμενη ενότητα), η εξάρτηση από τη φυσική γλώσσα και η έλλειψη μιας τυποποιημένης διαλέκτου.

Αφού δεν υπάρχει τυποποιημένη μορφή ψευδοκώδικα, απαιτείται, όλοι όσοι εμπλέκονται στην ανάπτυξη του λογισμικού να συμφωνήσουν σε ένα σύνολο λέξεων και δομών ελέγχου που θα χρησιμοποιούν. Οι βασικές δομές και λέξεις που θα χρησιμοποιούμε σε αυτό το βιβλίο συνοψίζονται στον Πίνακα 2.1 (τα τμήματα ανάμεσα σε [και] είναι προαιρετικά). Μια απλή δομημένη γλώσσα μπορεί να περιέχει τρεις δομές ελέγχου: ακολουθία, απόφαση, επανάληψη, ενώ υποστηρίζει και τον ορισμό ολοκληρωμένων τμημάτων.

Ως **ακολουθία** ορίζεται οποιαδήποτε σειρά οδηγιών (προτιμούμε να χρησιμοποιήσουμε εδώ τη λέξη "οδηγία" αντί της λέξης "εντολή", για να τονίσουμε το γεγονός ότι η δομημένη γλώσσα δεν είναι γλώσσα προγραμματισμού). Εάν μια σειρά οδηγιών πρόκειται να χρησιμοποιηθεί πολλές φορές μέσα στην περιγραφή, καλό είναι να την ορίσουμε ως **ομάδα οδηγιών (block).**

Πίνακας 2.1

Δομές και λέξεις ψευδοκώδικα που θα χρησιμοποιούμε

Αρχή ομάδας οδηγιών	ΑΡΧΗ
Τέλος ομάδας οδηγιών	ΤΕΛΟΣ
Καταχώρηση σε μεταβλητή	:=
Διαχωριστής εντολών	;
Είσοδος δεδομένων από συσκευή (προαιρετικά)	ΔΙΑΒΑΣΕ (λίστα ονομάτων μεταβλητών εισόδου) [ΑΠΟ πηγή δεδομένων]
Εξοδος δεδομένων στην οθόνη	ΤΥΠΩΣΕ (μεταβλητές εξόδου, “κείμενο”)
Εξοδος δεδομένων στον εκτυπωτή	ΕΚΤΥΠΩΣΕ (μεταβλητές, “κείμενο”)

Εξοδος δεδομένων σε άλλη συσκευή	ΓΡΑΨΕ (μεταβλητές, "κείμενο") ΣΕ συσκευή
Απόφαση	ΕΑΝ (συνθήκη) ΤΟΤΕ οδηγία ή ομάδα οδηγιών [ΑΛΛΙΩΣ οδηγία ή ομάδα οδηγιών] ΕΑΝ-ΤΕΛΟΣ
Επανάληψη με μετρητή / δείκτη	ΓΙΑ μετρητής = αρχική τιμή ΕΩΣ τελική τιμή ΕΠΑΝΕΛΑΒΕ οδηγία ή ομάδα οδηγιών ΓΙΑ-ΤΕΛΟΣ
Επανάληψη με συνθήκη στην αρχή	ΕΝΟΣΩ (συνθήκη) ΕΠΑΝΕΛΑΒΕ οδηγία ή ομάδα οδηγιών ΕΝΟΣΩ-ΤΕΛΟΣ
Επανάληψη με συνθήκη στο τέλος	ΕΠΑΝΕΛΑΒΕ οδηγία ή ομάδα οδηγιών ΜΕΧΡΙ (συνθήκη)
Κλήση υποπρογράμματος με μεταβλητές εισόδου (προαιρετικά)	ΥΠΟΛΟΓΙΣΕ όνομα υποπρογράμματος [(λίστα μεταβλητών)]

Δεν πρέπει να ξεχνάμε ότι ο ψευδοκώδικας αποτελεί ουσιαστικά ένα εργαλείο σχεδίασης και περιγραφής περισσότερο αλγορίθμων παρά προγραμμάτων. Όμως, σε μια περιγραφή με ψευδοκώδικα μπορεί να περιέχονται οδηγίες εισόδου/εξόδου, προσπέλασης δομών δεδομένων και άλλες λεπτομέρειες, οι οποίες ξεφεύγουν από την περιοχή της σχεδίασης και εμπίπτουν περισσότερο στις αρμοδιότητες των προγραμματιστών (είναι γνωστό πόσο ευέξαπτοι γίνονται οι προγραμματιστές, όταν κάποιος προσπαθεί να τους υποδείξει πώς θα υλοποιήσουν ένα πρόγραμμα, αντί του τι θα κάνει το πρόγραμμα!). Ακόμη, προγραμματιστές που είναι συνηθισμένοι στη χρήση μιας συγκεκριμένης γλώσσας προγραμματισμού συνήθως γράφουν ψευδοκώδικα που αποτελεί συντετμημένη έκδοση των εντολών της γλώσσας αυτής.

Στη συνέχεια περιγράφεται ο αλγόριθμος του Ευκλείδη με ψευδοκώδικα (MOD είναι ο τελεστής που δίνει το υπόλοιπο της ακέραιας διαίρεσης δύο αριθμών, και < > σημαίνει "όχι ίσο με"). Στη δραστηριότητα 1 του κεφαλαίου 2 χρησιμοποιήθηκε αρχικά φυσική γλώσσα και

στη συνέχεια δομημένη φυσική γλώσσα για την περιγραφή του αλγορίθμου. Όπως βλέπετε, η αναπαράσταση με ψευδοκώδικα πλεονεκτεί της αναπαράστασης ενός αλγορίθμου με φυσική γλώσσα, κυρίως επειδή η τελευταία χρησιμοποιεί ένα τεράστιο λεξιλόγιο, πολλές λέξεις του οποίου δεν επιδέχονται σαφώς καθορισμένη ερμηνεία. Εξάλλου, μια περιγραφή ενός αλγορίθμου σε φυσική γλώσσα σχεδόν πάντα δημιουργεί παρανοήσεις ανάμεσα στους σχεδιαστές. Ακόμη, ο ψευδοκώδικας, επειδή πλησιάζει πολύ τον πραγματικό κώδικα, πλεονεκτεί της δομημένης φυσικής γλώσσας, η οποία συνήθως, δημιουργεί παρανοήσεις ανάμεσα στους προγραμματιστές.

```
ΑΛΓΟΡΙΘΜΟΣ ΜΚΔ

ΔΕΔΟΜΕΝΑ
    X, Y, Z, MKD: INTEGER;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ (X,Y) ;
    Z := X MOD Y ;
    ΕΑΝ (Z < > 0) ΤΟΤΕ
        X := Y ;
        Y := Z
    ΑΛΛΙΩΣ
        ΜΚΔ := Y
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Υπάρχουν και άλλες τεχνικές λεκτικής περιγραφής, όπως ο μετακώδικας (metacode), η γλώσσα σχεδίασης προγραμμάτων (program design language – PDL), η γλώσσα σχεδίασης διεργασιών (process design language) κλπ. Κάποιες από αυτές θα περιγραφούν στο κεφάλαιο 3.

Μερικά απλά παραδείγματα ...

Ας προσπαθήσουμε πρώτα να περιγράψουμε έναν αλγόριθμο που μετατρέπει βαθμούς Φαρενάιτ (F) σε βαθμούς Κελσίου (C) χρησιμοποιώντας την εξίσωση: C=5/9 * (F–32):

```
ΑΛΓΟΡΙΘΜΟΣ F-ΣΕ-C
ΔΕΔΟΜΕΝΑ
    C,F: REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(F);
    C:=5*(F-32)/9;
    ΤΥΠΩΣΕ(C)
ΤΕΛΟΣ
```

Ο αλγόριθμος αυτός είναι αρκετά απλός και δείχνει τι ακριβώς είναι μια ακολουθία οδηγιών. Προσέξτε το σύμβολο ";" που χρησιμοποιείται για να διαχωρίσει τις οδηγίες (κάτι συνηθισμένο στις γλώσσες προγραμματισμού), εκτός από την οδηγία που ακολουθείται από κάποιας μορφής ΤΕΛΟΣ. Ας δούμε τώρα έναν αλγόριθμο που βρίσκει τον μεγαλύτερο από δύο αριθμούς Χ και Υ:

```
ΑΛΓΟΡΙΘΜΟΣ MAX-XY
ΔΕΔΟΜΕΝΑ
    X,Y,MAX: REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(X,Y);
    ΕΑΝ (X > Y) ΤΟΤΕ
        MAX:=X
    ΑΛΛΙΩΣ
        MAX:=Y
    ΕΑΝ-ΤΕΛΟΣ;
    ΤΥΠΩΣΕ(MAX)
ΤΕΛΟΣ
```

Απλά, ο αλγόριθμός μας συγκρίνει τους δύο αριθμούς και τυπώνει το μεγαλύτερο!

Δραστηριότητα 2.3

(α) Προσπαθήστε να εξηγήσετε τί κάνει ο αλγόριθμος όταν Χ=Υ

(β) Προσπαθήστε να γράψετε έναν αλγόριθμο MAX-XYZ που να βρίσκει τον μεγαλύτερο από τρεις αριθμούς

Έπειτα, συγκρίνετε τις απαντήσεις σας με αυτές που ακολουθούν.

Όταν X=Y, ο αλγόριθμος εκτελεί το τμήμα του ΑΛΛΙΩΣ και τυπώνει τελικά τον Y. Στην πραγματικότητα, όμως, δεν έχει σημασία ποιος αριθμός θα τυπωθεί, έτσι δεν είναι;

Χρησιμοποιώντας παρόμοια σκέψη, ο αλγόριθμος MAX-XYZ έχει ως εξής:

```
ΑΛΓΟΡΙΘΜΟΣ MAX-XYZ

ΔΕΔΟΜΕΝΑ
    X,Y,Z: REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(X,Y,Z);
    ΕΑΝ (X > Y) ΤΟΤΕ
        ΕΑΝ (X > Z) ΤΟΤΕ
            ΤΥΠΩΣΕ(X)
        ΑΛΛΙΩΣ
            ΤΥΠΩΣΕ(Z)
        ΕΑΝ-ΤΕΛΟΣ
    ΑΛΛΙΩΣ
        ΕΑΝ (Y > Z) ΤΟΤΕ
            ΤΥΠΩΣΕ(Y)
        ΑΛΛΙΩΣ
            ΤΥΠΩΣΕ(Z)
        ΕΑΝ-ΤΕΛΟΣ
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Παρατηρήστε τον τρόπο που ενσωματώνουμε μια δομή απόφασης μέσα σε μια άλλη δομή απόφασης (λέγεται και «φωλιασμένο εάν» – nested if). Η οδηγία ΕΑΝ-ΤΕΛΟΣ βοηθά στην επισήμανση του τέλους της κάθε δομής απόφασης. Ακόμη, δεν χρειάζεται να χρησιμοποιήσουμε σύμβολα ομάδας εντολών, αφού κάθε δομή απόφασης θεωρείται ως μια εντολή.

Ο αλγόριθμος MAX χρησιμοποιεί οδηγίες απόφασης, ενώ ο επόμενος αλγόριθμος χρησιμοποιεί οδηγία επανάληψης για να υπολογίσει το μέσο όρο των στοιχείων ενός πίνακα 1xN (ο οποίος έχει μια γραμμή και N στήλες), αφού πρώτα τυπώσει όλα τα στοιχεία του:

ΑΛΓΟΡΙΘΜΟΣ ΜΟ-ΠΙΝΑΚΑ-1ΧΝ

```
ΔΕΔΟΜΕΝΑ
    P: ARRAY[1..N] OF INTEGER;
    N, X, I: INTEGER;
    MO: REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N);
    X:=0;
    ΓΙΑ I:=1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΤΥΠΩΣΕ(P[I]) ;
        X:=X+P[I]
    ΓΙΑ-ΤΕΛΟΣ
    MO:=X/N;
    ΤΥΠΩΣΕ(MO)
ΤΕΛΟΣ
```

Παρατηρήστε ότι:

- για την ορθή λειτουργία του αλγορίθμου πρέπει το Ν να είναι καθορισμένο,
- ο αλγόριθμος λειτουργεί για διαφορετικά Ν, αφού την τιμή του Ν τη διαβάζει κάθε φορά,
- στη μεταβλητή Χ αποθηκεύουμε στην Κ επανάληψη το άθροισμα των Κ πρώτων θέσεων του πίνακα,
- η μεταβλητή αυτή αρχικοποιείται στην τιμή 0,
- αν και ο πίνακας αποθηκεύει ακέραιους αριθμούς, ο μέσος όρος δηλώνεται ως πραγματικός αριθμός.

Δραστηριότητα 2.4

Προσπαθήστε να προσαρμόσετε τον παραπάνω αλγόριθμο, ώστε να βρίσκει το μέσο όρο από τα στοιχεία ενός πίνακα δύο διαστάσεων (ΜxΝ). Έπειτα, ρίξτε μια ματιά στον αλγόριθμο που σας προτείνουμε.

ΑΛΓΟΡΙΘΜΟΣ ΜΟ-ΠΙΝΑΚΑ-ΜΧΝ

```
ΔΕΔΟΜΕΝΑ
    P: ARRAY[1..M,1..N] OF INTEGER;
    M,N,X,I,K: INTEGER;
```

```
    MO:REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(M,N);
    X:=0;
    ΓΙΑ K:=1 ΕΩΣ M ΕΠΑΝΕΛΑΒΕ
        ΓΙΑ I:=1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
            ΤΥΠΩΣΕ(P[K,I]);
            X:=X+P[K,I]
        ΓΙΑ-ΤΕΛΟΣ ;
        ΤΥΠΩΣΕ(EOL)
    ΓΙΑ-ΤΕΛΟΣ ;
    MO:=X / (M*N);
    ΤΥΠΩΣΕ(MO)
ΤΕΛΟΣ
```

Για να τυπώσουμε όλα τα στοιχεία ενός πίνακα δύο διαστάσεων, χρησιμοποιούμε δύο φωλιασμένες εντολές επανάληψης (nested for): μια για να διαπερνά τις γραμμές και μια για τις στήλες. Η τοποθέτηση της μιας εντολής μέσα στην άλλη σημαίνει ότι τυπώνουμε όλες τις στήλες κάθε γραμμής μέχρι να τελειώσουν οι γραμμές. Κάθε φορά που αλλάζουμε γραμμή, αλλάζουμε και γραμμή στην εκτύπωση (EOL είναι ο ειδικός χαρακτήρας αλλαγής γραμμής).

... και ένα σύνθετο παράδειγμα

Ας υποθέσουμε ότι θέλετε να αγοράσετε μια καινούρια φωτογραφική μηχανή. Ψάχνετε λοιπόν στην αγορά και καταλήγετε σε καμιά εικοσαριά μοντέλα, όλα με διαφορετικές δυνατότητες και τιμές. Φυσικά, έχετε φτιάξει έναν πίνακα στον υπολογιστή σας, όπου αποθηκεύετε κάθε μοντέλο που σας ενδιαφέρει. Μετά το τέλος της έρευνας, θα θέλατε να ταξινομήσετε τα μοντέλα που βρήκατε με βάση την τιμή τους, ώστε να αποφασίσετε ποιο θα αγοράσετε. Υπάρχουν πολλοί αλγόριθμοι ταξινόμησης για να διαλέξετε. Εδώ, θα εξετάσουμε τον πιο απλό, την ταξινόμηση με επιλογή (selection sort):

ΑΛΓΟΡΙΘΜΟΣ ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ

```
ΔΕΔΟΜΕΝΑ
    I, K, N, MAX, POS, TEMP: INTEGER;
```

```
    P: ARRAY [1..N] OF INTEGER;
ΑΡΧΗ
    ΓΙΑ K:=1 ΕΩΣ N-1 ΕΠΑΝΕΛΑΒΕ
        MAX:=P[K];
        POS := K;
        ΓΙΑ I:=K+1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
            ΕΑΝ (P[I] > P[K]) ΤΟΤΕ
                MAX:=P[I];
                POS :=I
            ΕΑΝ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
        TEMP:=P[K];
        P[K]:=MAX;
        P[POS]:=TEMP
    ΓΙΑ-ΤΕΛΟΣ ;
    ΓΙΑ I:=1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΤΥΠΩΣΕ(P[I])
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Δραστηριότητα 2.5

Ο αλγόριθμος ταξινόμησης με επιλογή συνδυάζει και τις τρεις βασικές δομές οδηγιών που έχουμε έως τώρα συναντήσει (ακολουθία, απόφαση, επανάληψη). Τα δεδομένα εισόδου βρίσκονται στον πίνακα Ρ. Στον ίδιο πίνακα αποθηκεύονται και τα δεδομένα εξόδου (που είναι τα δεδομένα εισόδου ταξινομημένα). Προσπαθήστε να περιγράψετε πώς λειτουργεί ο αλγόριθμος πριν διαβάσετε το υπόλοιπο κείμενο (για παράδειγμα, δοκιμάστε να ταξινομήσετε με τον αλγόριθμο αυτό τους αριθμούς 3, 1, 2, 4).

Η ιδέα πίσω από την ταξινόμηση με επιλογή είναι σε κάθε πέρασμα του πίνακα, να βρίσκει ο αλγόριθμος το μεγαλύτερο στοιχείο του και να το τοποθετεί στο αριστερό άκρο του πίνακα που εξετάσθηκε. Τότε, στο επόμενο πέρασμα, δε χρειάζεται να εξεταστεί πάλι το στοιχείο αυτό, οπότε ο αλγόριθμος διαπερνά τον υπόλοιπο πίνακα. Στο τέλος, ο πίνακας βρίσκεται ταξινομημένος κατά φθίνουσα σειρά.

Αναλυτικότερα, για να ταξινομήσουμε τα στοιχεία ενός πίνακα εφαρ-

μόζοντας τη μέθοδο ταξινόμησης με επιλογή, σκεφτόμαστε ως εξής:

- Διαβάζουμε το πρώτο στοιχείο του πίνακα και το συγκρίνουμε με όλα τα υπόλοιπα, με στόχο να βρούμε το μεγαλύτερο. Αυτό το στοιχείο αλλάζει αμοιβαία θέση με το πρώτο στοιχείο. Μετά το βήμα αυτό, ο πίνακας του παραδείγματος θα γίνει 4, 1, 2, 3.
- Έπειτα, διαβάζουμε το δεύτερο στοιχείο του πίνακα και το συγκρίνουμε με όλα τα υπόλοιπα (εκτός του πρώτου). Το μεγαλύτερο από αυτά αλλάζει αμοιβαία θέση με το δεύτερο στοιχείο. Μετά το βήμα αυτό, ο πίνακας του παραδείγματος θα γίνει 4, 3, 1, 2.
- Επαναλαμβάνουμε αυτή τη διαδικασία μέχρι το προτελευταίο στοιχείο του πίνακα (δηλαδή, συνολικά N-1 φορές για N στοιχεία). Μετά το τρίτο και τελευταίο βήμα, ο πίνακας του παραδείγματος θα γίνει 4, 3, 2, 1.

Δραστηριότητα 2.6

Προσπαθήστε να γράψετε έναν αλγόριθμο, ο οποίος θα διαβάζει στην είσοδο μία ακολουθία 100 χαρακτήρων και θα μετρά πόσα άρτια, περιττά και μηδενικά ψηφία περιέχονται σ' αυτή. Έπειτα, συγκρίνετε τον αλγόριθμό σας με αυτόν που ακολουθεί

ΑΛΓΟΡΙΘΜΟΣ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ-ΨΗΦΙΩΝ

```
ΔΕΔΟΜΕΝΑ
    N: CHAR ;
    I, MA, MP, M0, MX: INTEGER ;
ΑΡΧΗ
    MA=MP=MX=M0 := 0 ;
    ΓΙΑ I := 1 ΕΩΣ 100 ΕΠΑΝΕΛΑΒΕ
        ΔΙΑΒΑΣΕ(N);
        ΕΑΝ (N = 0) ΤΟΤΕ
                M0 := M0+1
        ΑΛΛΙΩΣ ΕΑΝ (N in {1,3,5,7,9}) ΤΟΤΕ
                MP := MP+1
            ΑΛΛΙΩΣ ΕΑΝ (N in {2,4,6,8}) ΤΟΤΕ
                    MA := MA+1
                ΑΛΛΙΩΣ
```

```
                    ΜΧ := ΜΧ+1
                ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ(«ΕΜΦΑΝΙΣΤΗΚΑΝ», Μ0, «ΜΗΔΕΝ», ΜΑ,
    «ΑΡΤΙΑ ΨΗΦΙΑ», ΜΡ, «ΠΕΡΙΤΤΑ ΨΗΦΙΑ ΚΑΙ», ΜΧ,
    «ΑΛΛΟΙ ΧΑΡΑΚΤΗΡΕΣ»)
ΤΕΛΟΣ
```

Παρατηρήστε ότι χρησιμοποιούμε τέσσερις μετρητές, Μ0 για τα μηδενικά ψηφία, ΜΑ για τα άρτια ψηφία, ΜΡ για τα περιττά ψηφία και ΜΧ για τους άλλους χαρακτήρες. Σε κάθε επανάληψη διαβάζουμε στη μεταβλητή Ν το επόμενο στοιχείο (π.χ. από το πληκτρολόγιο, ή από ένα αρχείο), το αξιολογούμε με τη δομή ΕΑΝ-ΑΛΛΙΩΣ και αυξάνουμε τον αντίστοιχο μετρητή (η συνθήκη ΕΑΝ Ν in {2 ,4, 6, 8} ελέγχει εάν το στοιχείο Ν ανήκει στο σύνολο {2, 4, 6, 8}, δηλαδή εάν είναι άρτιος αριθμός).

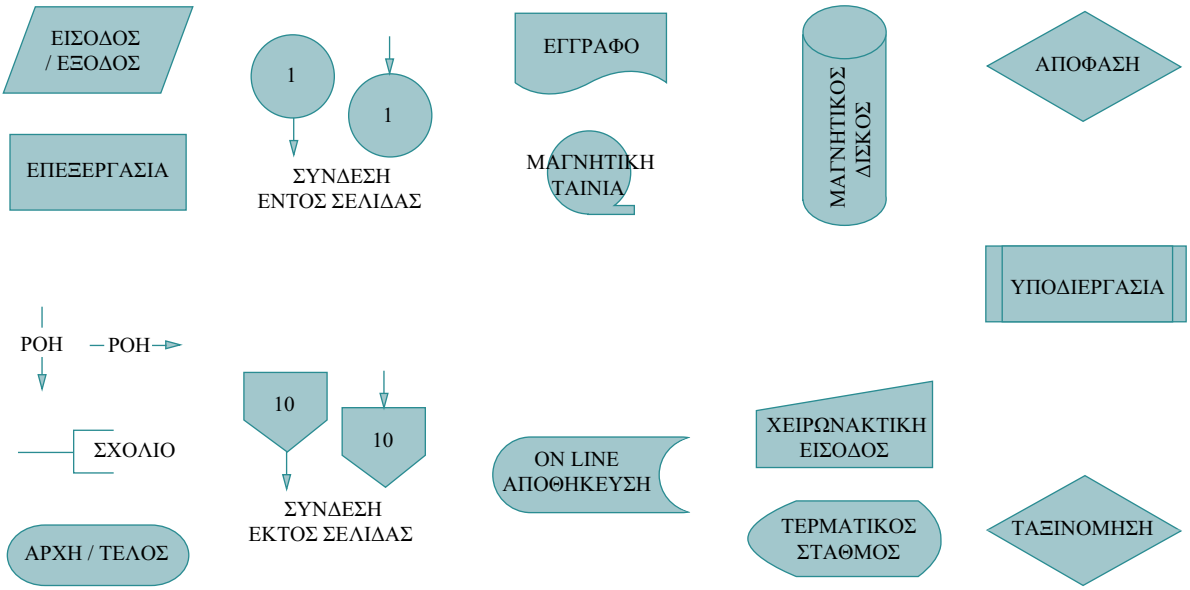
2.4 Αναπαράσταση αλγορίθμων με διάγραμμα ροής

Εκτός από τη λεκτική αναπαράσταση των αλγορίθμων, διαδεδομένη είναι και η συμβολική (γραφική) αναπαράσταση, η οποία βασίζεται κυρίως στα **Διαγράμματα Ροής Προγράμματος - ΔΡΠ (Program Flowcharts)** (Shooman, 1983). Πρόκειται για το παλαιότερο, περισσότερο χρησιμοποιημένο, διαδεδομένο, αλλά και περισσότερο αμφισβητημένο εργαλείο αναπαράστασης.

Τα ΔΡΠ θεωρούν έναν αλγόριθμο σαν ένα σύνολο από απλά ή σύνθετα τμήματα κώδικα. Επειδή για κάθε τέτοιο τμήμα υπάρχει και ένα ειδικό σύμβολο, τελικά ο αλγόριθμος αναπαρίσταται ως ένα ΔΡΠ. Τα βασικότερα από το σύνολο των κατά ANSI τυποποιημένων συμβόλων περιλαμβάνονται στο Σχήμα 2.1. Το βασικό σύμβολο είναι η ροή ελέγχου, η οποία αναπαριστά την ακολουθία εκτέλεσης των τμημάτων του αλγορίθμου. Σημειώστε ότι μέσα στα υπόλοιπα σύμβολα πρέπει να γράψουμε και το αντικείμενο της οδηγίας ή ενέργειας που αυτό περιγράφει (π.χ., μεταβλητές, συνθήκες, εκφράσεις κλπ.).

Τα ΔΡΠ συνήθως, χρησιμοποιούνται ως μια υψηλού επιπέδου αναπαράσταση ενός αλγορίθμου, η οποία πλεονεκτεί έναντι της λεκτικής αναπαράστασης στο ότι είναι απλή, αυστηρή και ανεξάρτητη από

φυσική γλώσσα. Εξάλλου, ας μην ξεχνούμε ότι μια εικόνα αξίζει όσο χίλιες λέξεις! Εκτός από τεκμηρίωση, τα ΔΡΠ προσφέρουν και εύκολη συντήρηση, γι' αυτό και αποτελούν ένα από τα περισσότερο διαδεδομένα παραδοτέα ενός έργου ανάπτυξης λογισμικού.

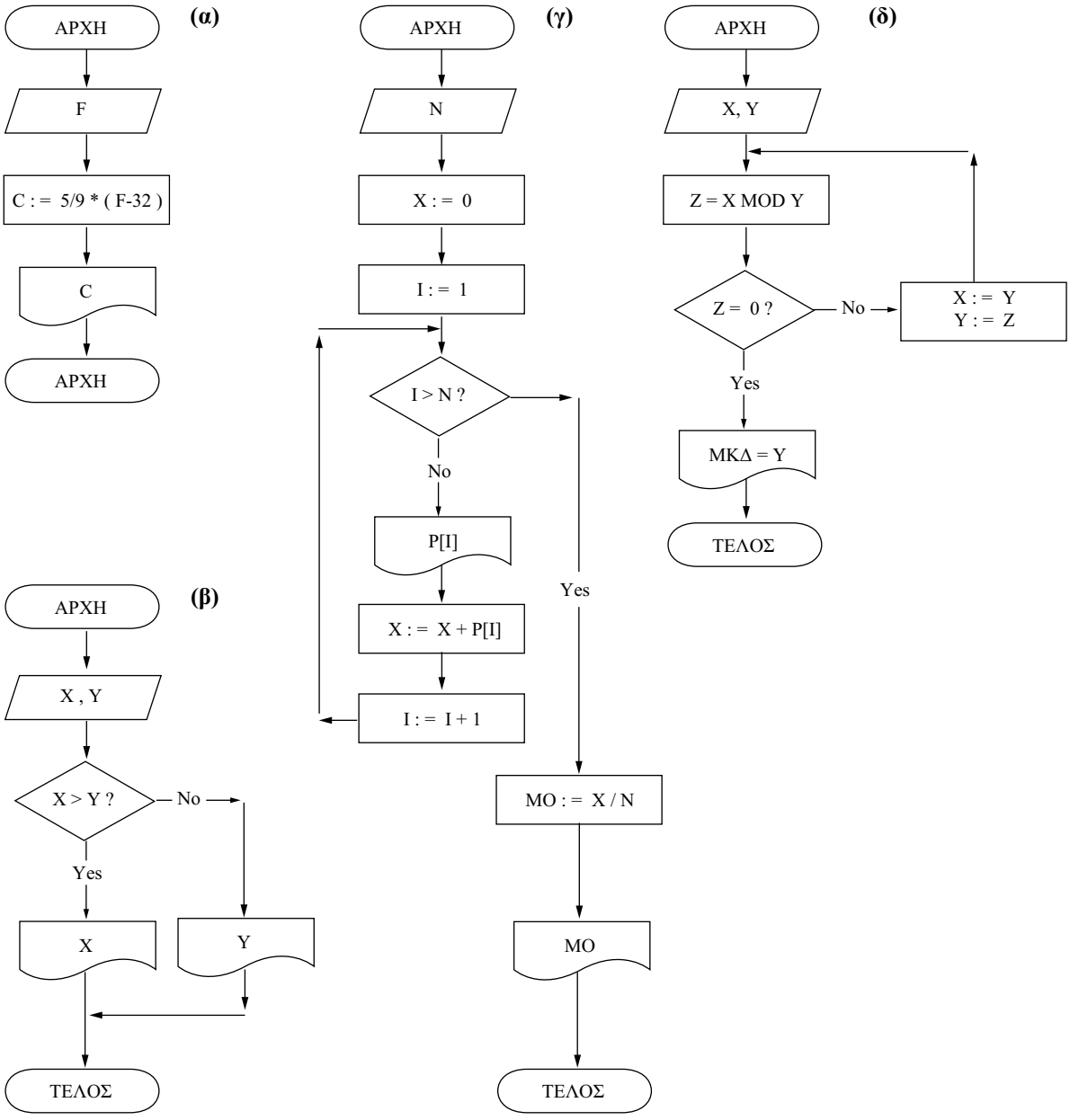


Σχήμα 2.1
Μερικά από τα βασικά κατά ANSI σύμβολα ενός ΔΡΠ

Στο Σχήμα 2.2 παρατίθενται τα ΔΡΠ για τους αλγορίθμους (α) μετατροπής βαθμών Φαρενάιτ σε Κελσίου, (β) εύρεσης του μεγαλύτερου από δύο αριθμούς, (γ) υπολογισμού του μέσου όρου των στοιχείων ενός μονοδιάστατου πίνακα, και (δ) του Ευκλείδη. Αφού όλοι αυτοί οι αλγόριθμοι έχουν ήδη περιγραφεί και με ψευδοκώδικα στην ενότητα 2.3, θα σας ωφελούσε αρκετά να συγκρίνετε τα δύο εργαλεία αναπαράστασης.

Κατά την κατασκευή ενός ΔΡΠ καλό είναι να ακολουθούμε κάποιους βασικούς κανόνες. Γενικά, η ροή σ' ένα σύστημα που αναπαρίσταται με ΔΡΠ έχει κατεύθυνση από πάνω προς τα κάτω και από αριστερά προς τα δεξιά. Καλό είναι να δίνουμε σε κάθε σύμβολο του διαγράμματος ένα και μοναδικό όνομα και να μην έχουμε σύμβολα χωρίς όνομα (δε χρειάζεται να ονοματίσουμε τις ροές, αφού δείχνουν απλά

την ακολουθία εκτέλεσης). Επίσης, πρέπει να φροντίζουμε ένα διάγραμμα να περιέχει τόσα σύμβολα, ώστε να χωρά σε μια σελίδα. Στην πραγματικότητα, εάν το ΔΡΠ είναι πολύπλοκο, μπορούμε να χρησιμοποιήσουμε τα σύμβολα διασύνδεσης, είτε μέσα στην ίδια σελίδα, είτε



Σχήμα 2.2
Μερικά ΔΡΠ που αναπαριστούν σχετικά απλούς αλγόριθμους

σε άλλη σελίδα. Επειδή όμως δεν είναι καλή πρακτική να απλώσουμε ένα ΔΡΠ σε πολλές σελίδες, μπορούμε να αναλύσουμε κάποιο σύμβολο (συνήθως μιας διεργασίας) σε ένα ΔΡΠ χαμηλότερου επιπέδου.

Βέβαια, ένα ΔΡΠ μπορεί να αναπαραστήσει έναν αλγόριθμο σε οποι-

ΑΛΓΟΡΙΘΜΟΣ ΑΤΜ

ΔΕΔΟΜΕΝΑ

ΚΑΡΤΑ, ΚΩΔΙΚΟΣ, ΣΥΝΑΛΛΑΓΗ, ΠΟΣΟ,
ΧΡΗΜΑΤΑ, ΑΠΟΔΕΙΞΗ

ΑΡΧΗ

ΔΙΑΒΑΣΕ (ΚΑΡΤΑ) ΑΠΟ ΣΧΙΣΜΗ-ΚΑΡΤΑΣ
ΔΙΑΒΑΣΕ (ΚΩΔΙΚΟΣ) ΑΠΟ ΚΑΡΤΑ
ΔΙΑΒΑΣΕ (ΣΥΝΑΛΛΑΓΗ) ΑΠΟ ΜΕΝΟΥ

ΕΑΝ (ΣΥΝΑΛΛΑΓΗ=ανάληψη) ΤΟΤΕ
    ΔΙΑΒΑΣΕ (ΠΟΣΟ) ΑΠΟ ΜΕΝΟΥ
    ΥΠΟΛΟΓΙΣΕ ΕΠΕΞΕΡΓΑΣΙΑ-ΣΤΟΙΧΕΙΩΝ
    ΓΡΑΨΕ (ΧΡΗΜΑΤΑ) ΣΕ ΣΧΙΣΜΗ-ΧΡΗΜΑΤΩΝ
    ΓΡΑΨΕ (ΑΠΟΔΕΙΞΗ) ΣΕ ΣΧΙΣΜΗ- ΑΠΟΔΕΙΞΗΣ

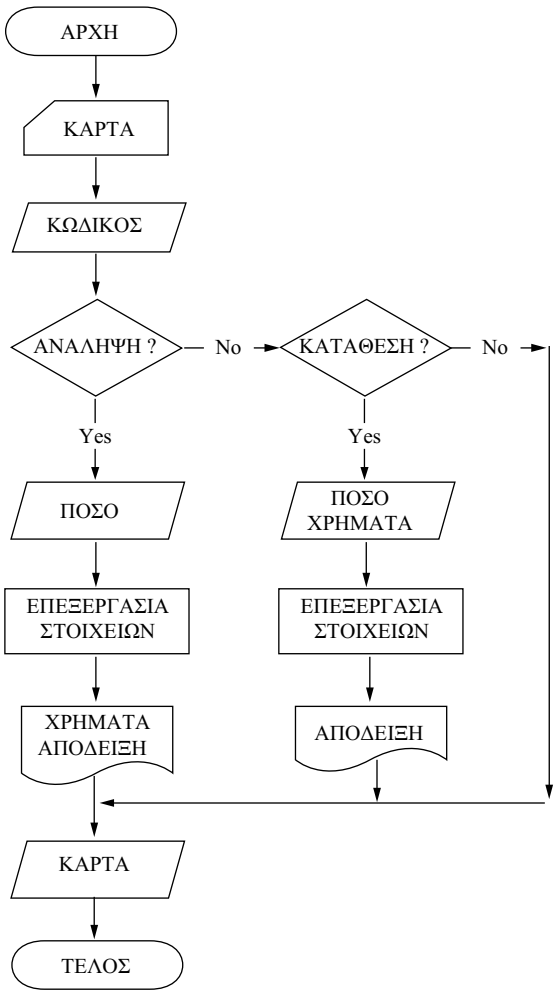
ΑΛΛΙΩΣ

    ΕΑΝ (ΣΥΝΑΛΛΑΓΗ=κατάθεση) ΤΟΤΕ
        ΔΙΑΒΑΣΕ (ΠΟΣΟ, ΧΡΗΜΑΤΑ) ΑΠΟ ΜΕΝΟΥ
        ΥΠΟΛΟΓΙΣΕ ΕΠΕΞΕΡΓΑΣΙΑ-ΣΤΟΙΧΕΙΩΝ
        ΓΡΑΨΕ (ΑΠΟΔΕΙΞΗ) ΣΕ ΣΧΙΣΜΗ–ΑΠΟΔΕΙΞΗΣ

    ΕΑΝ-ΤΕΛΟΣ

ΕΑΝ-ΤΕΛΟΣ

ΓΡΑΨΕ (ΚΑΡΤΑ) ΣΕ ΣΧΙΣΜΗ–ΚΑΡΤΑΣ
ΤΕΛΟΣ



Σχήμα 2.3

Ψευδοκώδικας & ΔΡΠ χρήσης μιας Αυτόματης Μηχανής Συναλλαγών

οδήποτε βαθμό λεπτομέρειας, ακόμη και με τρόπο ώστε να υπάρχει μία προς μία αντιστοιχία συμβόλων και εντολών του προγράμματος. Όμως, ένα τέτοιο ΔΡΠ για ένα πρόγραμμα μεσαίου μεγέθους θα καταλαμβάνει δεκάδες σελίδες, οπότε γίνεται εξαιρετικά δύσχρηστο (εκτός και αν χρησιμοποιείται μέσα από κάποιο αυτόματο εργαλείο, το οποίο αναλαμβάνει και τη συντήρησή του).

Δραστηριότητα 2.7

Στο Σχήμα 2.3 φαίνεται ψευδοκώδικας και διάγραμμα ροής για τον αλγόριθμο χρήσης μιας Αυτόματης Μηχανής Συναλλαγών. Προσπαθήστε να «διαβάσετε» τις δύο αναπαραστάσεις και μετά ρίξτε ξανά μια ματιά στην περιγραφή που ακολουθεί.

Αφού λοιπόν εισάγουμε την κάρτα συναλλαγών και πληκτρολογήσουμε τον (σωστό) κωδικό αριθμό, πρέπει να διαλέξουμε τη συναλλαγή από ένα σύνολο επιλογών (στην περίπτωση μόνο δύο επιλογές περιγράφονται: ανάληψη και κατάθεση χρημάτων). Εάν διαλέξουμε ανάληψη, τότε πρέπει να πληκτρολογήσουμε το ποσό της ανάληψης, να παραλάβουμε τα χρήματα και την απόδειξη, και να παραλάβουμε την κάρτα για να τελειώσει η συναλλαγή. Εάν διαλέξουμε κατάθεση, τότε πρέπει να πληκτρολογήσουμε το ποσό της κατάθεσης, να καταθέσουμε τα χρήματα, να παραλάβουμε την απόδειξη, και να παραλάβουμε την κάρτα για να τελειώσει η συναλλαγή.

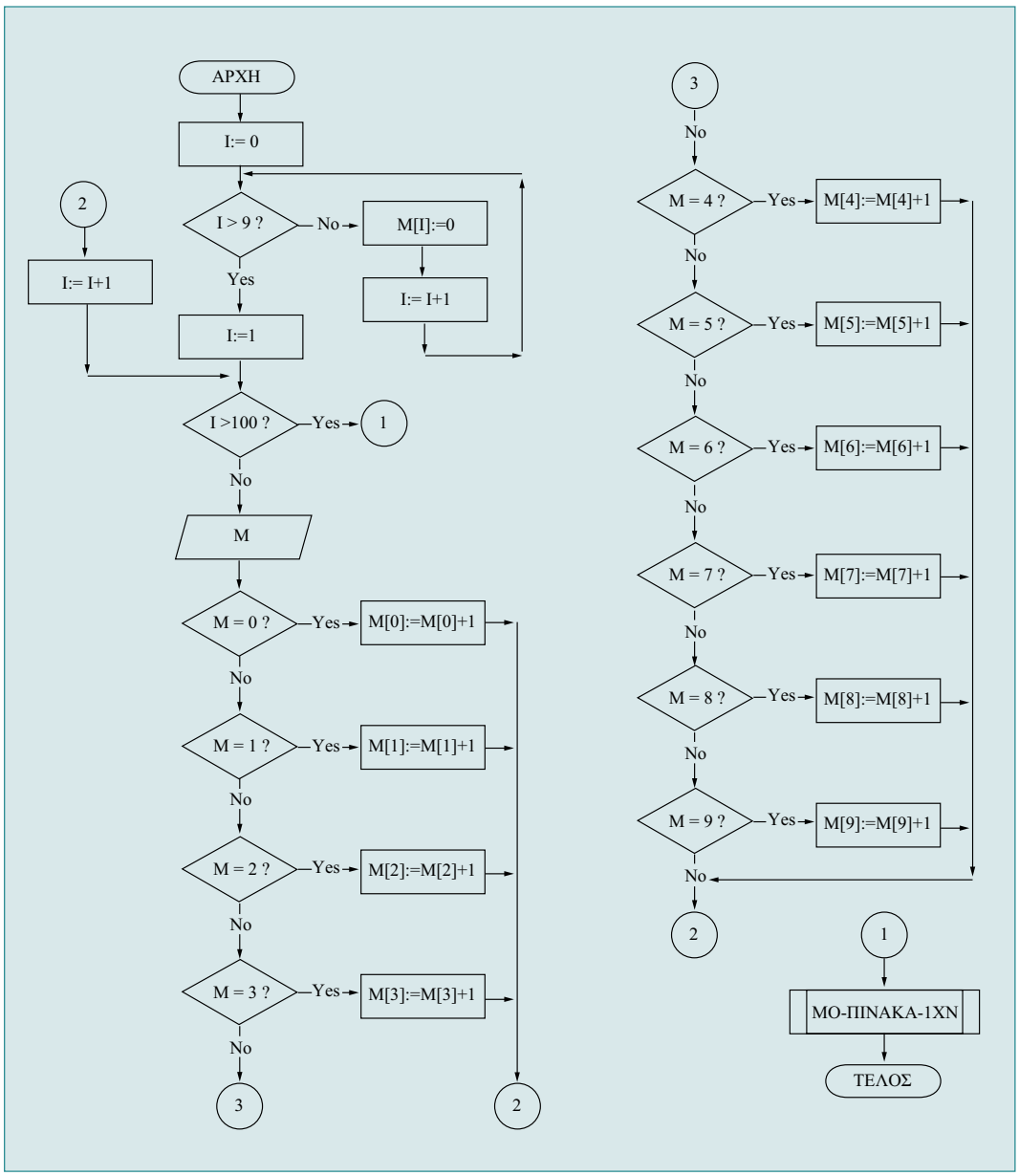
Άσκηση Αυτοαξιολόγησης 2.1

Προσπαθήστε να κατασκευάσετε ΔΡΠ για τους αλγορίθμους: (α) εύρεσης του μεγαλύτερου από τρεις αριθμούς, (β) ταξινόμησης με επιλογή, (γ) κατηγοριοποίησης των ψηφίων μιας ακολουθίας χαρακτήρων.

Υπόδειξη: Δοκιμάστε να αντιστοιχίσετε κάθε οδηγία ή δομή του ψευδοκώδικα με σύμβολα των ΔΡΠ.

Άσκηση Αυτοαξιολόγησης 2.2

Δοκιμάστε να περιγράψετε με ψευδοκώδικα τον αλγόριθμο που ακολουθεί (βλ. σχήμα στην επόμενη σελίδα).


ΑΡΧΗ
I:= 0
I > 9 ?
No
M[I]:=0
I:= I+1
Yes
I:=1
2
I:= I+1
I >100 ?
Yes
1
No
M
M = 0 ?
Yes
M[0]:=M[0]+1
No
M = 1 ?
Yes
M[1]:=M[1]+1
No
M = 2 ?
Yes
M[2]:=M[2]+1
No
M = 3 ?
Yes
M[3]:=M[3]+1
No
3
2
3
No
M = 4 ?
Yes
M[4]:=M[4]+1
No
M = 5 ?
Yes
M[5]:=M[5]+1
No
M = 6 ?
Yes
M[6]:=M[6]+1
No
M = 7 ?
Yes
M[7]:=M[7]+1
No
M = 8 ?
Yes
M[8]:=M[8]+1
No
M = 9 ?
Yes
M[9]:=M[9]+1
No
2
1
ΜΟ-ΠΙΝΑΚΑ-1ΧΝ
ΤΕΛΟΣ

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε η έννοια του αλγορίθμου και δόθηκαν παραδείγματα αλγορίθμων για την επίλυση προβλημάτων τόσο από τις επιστήμες, όσο και από την καθημερινή ζωή. Με βάση αυτά, συζητήσαμε τις ιδιότητες των αλγορίθμων και συμφωνήσαμε στο ότι ο καλός αλγόριθμος είναι πεπερασμένος και καλά ορισμένος, διαβάζει δεδομένα στην είσοδο και γράφει αποτελέσματα στην έξοδο, ενώ τα βήματα επεξεργασίας που ακολουθεί είναι απλά και αποτελεσματικά.

Στη συνέχεια, παρουσιάστηκαν αναλυτικά δύο εναλλακτικοί τρόποι περιγραφής αλγορίθμων, εκτός από τη φυσική γλώσσα, ο ψευδοκώδικας και το Διάγραμμα Ροής Προγράμματος. Ο ψευδοκώδικας συνδυάζει τις συντακτικές δομές των γλωσσών προγραμματισμού με λέξεις μιας φυσικής γλώσσας για να δώσει μια λεκτική περιγραφή του αλγορίθμου. Τα ΔΡΠ χρησιμοποιούν ειδικά γραφικά σύμβολα για κάθε τμήμα (δομή) κώδικα που συμμετέχει στον αλγόριθμο.

Αμέσως μετά, θα συνεχίσουμε παρουσιάζοντας τις δύο βασικές πρακτικές προγραμματισμού, την κατά βήμα εκλέπτυνση και τον αρθρωτό προγραμματισμό, ενώ στο κεφάλαιο 5 θα γνωρίσετε και άλλες τεχνικές αναπαράστασης αλγορίθμων.

Βιβλιογραφία Κεφαλαίου 2

[1] Davis, W. S. (1983), *Systems Analysis and Design: a structured approach.* Addison–Wesley

[2] Knuth, D. E. (1973), *Fundamental Algorithms*. Addison–Wesley: Reading, Mas.

[3] Μπεμ. Α. και Καραμπατζός, Γ. (1991), *Εισαγωγή στην πληροφορική*. Εκδ. Συμμετρία, Αθήνα.

[4] Shooman, M. L. (1983), *Software Engineering.* McGraw–Hill: Tokyo, Japan.

[5] Σκορδαλάκης, Ε. (1991), *Εισαγωγή στην Τεχνολογία Λογισμικού*. Εκδ. Συμμετρία, Αθήνα

ΚΕΦΑΛΑΙΟ 3

Πρακτικές προγραμματισμού

Σκοπός

Ο στόχος του κεφαλαίου είναι να σας εισαγάγει στον προγραμματισμό μέσα από δύο πολύ διαδεδομένες πρακτικές και τρία συνηθισμένα στυλ προγραμματισμού.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *αναφέρετε τέσσερις αρχές ανάπτυξης προγραμμάτων*
- *σχεδιάσετε έναν αλγόριθμο χρησιμοποιώντας την Κατά Βήμα Εκλέπτυνση*
- *σχεδιάσετε έναν αλγόριθμο με την πρακτική του Αρθρωτού Προγραμματισμού*
- *αναφέρετε και να περιγράψετε τρία διαδεδομένα στυλ προγραμματισμού*
- *περιγράψετε τρία διαδεδομένα παραδείγματα προγραμματισμού.*

Έννοιες κλειδιά

- *Αρχή της διάσπασης*
- *Αρχή της αφαιρετικότητας*
- *Αρχή της δομής*
- *Αρχή της τυπικότητας*
- *Κατά Βήμα Εκλέπτυνση*
- *Αρθρωτός Προγραμματισμός, Αμυντικός προγραμματισμός, Προγραμματισμός για επαναχρησιμοποίηση, Προγραμματισμός με πλεονασμό, Παράδειγμα προγραμματισμού, Διαδικασιακός προγραμματισμός, Δηλωτικός προγραμματισμός, Αντικειμενοστραφής προγραμματισμός.*

Εισαγωγικές παρατηρήσεις

Η εμπειρία έχει δείξει ότι η περιγραφή του τρόπου επίλυσης προβλημάτων μέσω της ανάπτυξης προγραμμάτων θα πρέπει να ακολουθεί

τις εξής τέσσερις βασικές αρχές:

- ***αρχή της διάσπασης:*** *κάθε σύνθετο πρόβλημα μπορεί να επιλυθεί ευκολότερα εάν το διασπάσουμε σε μικρότερα και απλούστερα προβλήματα. Κατ' επέκταση, κάθε πολύπλοκο πρόγραμμα μπορεί να διαιρεθεί σε απλούστερα προγράμματα, τα οποία θα αναπτυχθούν ανεξάρτητα.*
- ***αρχή της αφαιρετικότητας:*** *η λύση σε ένα σύνθετο πρόβλημα μπορεί να περιγράφεται σε διαφορετικά επίπεδα λεπτομέρειας. Αντίστοιχα, ένα σύνθετο πρόγραμμα μπορεί αρχικά να περιγράφεται ως ένα σύνολο υπηρεσιών (υψηλού επιπέδου αφηρημένη περιγραφή) και μέσα από μια διαδικασία εισαγωγής λεπτομερειών, να περιγράφεται τελικά ως ένα σύνολο λειτουργιών που πρέπει να υλοποιηθούν (χαμηλού επιπέδου συγκεκριμένη περιγραφή).*
- ***αρχή της δομής:*** *τα τμήματα που αποτελούν ένα σύνθετο πρόγραμμα προκύπτουν προχωρώντας από την αφηρημένη περιγραφή των λειτουργιών προς τη συγκεκριμένη περιγραφή της υλοποίησης και πρέπει να τακτοποιούνται (αλλιώς δομούνται, ιεραρχούνται) με τρόπο ώστε να συνθέτουν το «μεγάλο» πρόγραμμα.*
- ***αρχή της τυπικότητας:*** *όπως η επίλυση προβλημάτων απαιτεί την εφαρμογή μιας συστηματικής μεθοδολογίας (η οποία περιγράφεται στο κεφάλαιο 1), έτσι και η ανάπτυξη προγραμμάτων πρέπει να ακολουθεί κάποια μεθοδολογία που αποδεδειγμένα οδηγεί σε ποιοτικά αποτελέσματα. Μια τέτοια μεθοδολογία ονομάζεται «μοντέλο κύκλου ζωής λογισμικού» και, όπως έχουμε ήδη αναφέρει, αποτελείται από διαδοχικά και διακριτά βήματα, καθένα από τα οποία παράγει ένα αποτέλεσμα συγκεκριμένο και συγκρίσιμο ως προς ένα σύνολο προδιαγραφών.*

Στο κεφάλαιο 4 θα συναντήσουμε ξανά αυτές τις έννοιες. Μερικές θα τις συζητήσουμε σε μεγαλύτερη λεπτομέρεια, καθώς έχουν οδηγήσει στην ανάπτυξη διαφορετικών προσεγγίσεων σχεδίασης λογισμικού. Προς το παρόν, θα χρησιμοποιήσουμε αυτές τις αρχές για να περιγράψουμε δύο σημαντικές πρακτικές προγραμματισμού που έχουν αναπτυχθεί για να ανταποκριθούν στα χαρακτηριστικά των τεχνικών επίλυσης προβλημάτων:

- *την **κατά βήμα εκλέπτυνση** (ενότητα 3.1), η οποία παράγει διαδο-*

χικές «εκδόσεις» του προγράμματος. Κάθε έκδοση είναι ορθή και αυτάρκης, προκύπτει άμεσα από την προηγούμενη, και περιγράφει το πρόγραμμα σε μεγαλύτερο βαθμό λεπτομέρειας και

- *την **τμηματοποίηση** (ενότητα 3.2), η οποία υλοποιεί τη διάσπαση ενός προβλήματος σε απλούστερα, συνθέτοντας ένα πολύπλοκο πρόγραμμα από απλούστερα προγράμματα.*

Στην ενότητα 3.3 περιγράφονται τρία διαδεδομένα στυλ προγραμματισμού. Ένα στυλ προγραμματισμού υλοποιεί ένα σύνολο κανόνων προγραμματισμού που υιοθετεί ο προγραμματιστής. Η τελευταία ενότητα του κεφαλαίου είναι κυρίως ενημερωτική και παρουσιάζει τρία διαδεδομένα παραδείγματα προγραμματισμού.

3.1 Κατά βήμα εκλέπτυνση

Η **Κατά Βήμα Εκλέπτυνση - KBE (Stepwise Refinement)** είναι μια τεχνική που αναπτύχθηκε από τον Wirth (Wirth, 1971) για τη διάσπαση της υψηλού επιπέδου περιγραφής ενός συστήματος λογισμικού σε επίπεδα με μεγαλύτερη λεπτομέρεια. Όπως περιγράφεται αρχικά, η ΚΒΕ περιλαμβάνει τις εξής δραστηριότητες:

- Διάσπαση των σχεδιαστικών αποφάσεων σε στοιχειώδη επίπεδα.
- Απομόνωση σχεδιαστικών τμημάτων που στην πραγματικότητα δεν εξαρτώνται από άλλα.
- Αναβολή όσο το δυνατόν περισσότερο των αποφάσεων αναπαράστασης.
- Προσεκτική απόδειξη ότι κάθε διαδοχικό βήμα εκλέπτυνσης είναι πιστή επέκταση προηγούμενων βημάτων.
- Βελτίωση του προκύπτοντος αλγορίθμου.

Η αρχική περιγραφή του συστήματος αποτελείται από λίγα βασικά βήματα επεξεργασίας, τα οποία αποδεδειγμένα λύνουν το πρόβλημα. Έπειτα, η διαδικασία επαναλαμβάνεται για κάθε τμήμα του συστήματος, μέχρι αυτό να διασπασθεί σε ικανοποιητικό επίπεδο λεπτομέρειας, ώστε να είναι δυνατή η απευθείας υλοποίησή του με κάποια γλώσσα προγραμματισμού.

Η προσθήκη όλο και περισσότερης λεπτομέρειας σε κάθε διαδοχικό

βήμα της ΚΒΕ αναβάλλει όσο περισσότερο γίνεται τη λήψη οριστικών αποφάσεων σχεδίασης ενώ επιτρέπει στον σχεδιαστή να τεκμηριώσει πειστικά ότι το παραγόμενο πρόγραμμα είναι συμβατό με τις αρχικές προδιαγραφές.

Η ΚΒΕ δεν συμπεριλαμβάνει κάποια συγκεκριμένη τεχνική αναπαράστασης. Συνήθως, στα αρχικά στάδια χρησιμοποιείται ψευδοκώδικας, ο οποίος γίνεται συνεχώς πιο ακριβής, καθώς προχωρά η εκλέπτυνση του προγράμματος. Άλλες τεχνικές που μπορούν να χρησιμοποιηθούν είναι τα Διαγράμματα Ροής Προγράμματος, τα Διαγράμματα Δομής κλπ. Το τελικό σχέδιο μοιάζει πολύ με πρόγραμμα και πολλές φορές περιλαμβάνει και εντολές από κάποια γλώσσα προγραμματισμού.

Παράδειγμα 3.1

Ο αλγόριθμος που ακολουθεί είναι μια ελαφρά τροποποιημένη έκδοση του παραδείγματος που παρατίθεται στο Wirth, 1971 και δείχνει την εφαρμογή της ΚΒΕ στο πρόβλημα υπολογισμού και εκτύπωσης στην οθόνη των αρχικών Ν πρώτων αριθμών (ένας πρώτος αριθμός διαιρείται μόνο με τον εαυτό του και τον αριθμό 1).

Η αρχική έκδοση είναι σχετικά απλή και γενική (δηλαδή μπορεί να εφαρμοστεί σχεδόν σε όλα τα προβλήματα): ο αλγόριθμος διαβάζει κάποια δεδομένα στην είσοδο (το πλήθος Ν των πρώτων αριθμών), εκτελεί κάποιας μορφής επεξεργασία (βρίσκει τον επόμενο πρώτο αριθμό Χ), και παράγει κάποια πληροφορία στην έξοδο (τυπώνει τον Χ και ένα μήνυμα τέλους μόλις υπολογίσει Ν πρώτους αριθμούς). Κατά την επεξεργασία, ο αλγόριθμος βρίσκει διαδοχικά τους πρώτους αριθμούς και τους εκτυπώνει στην οθόνη, μέχρι το πλήθος τους να γίνει Ν.

Πριν προχωρήσετε στην ανάγνωση του αλγορίθμου, παρατηρήστε ότι (όπως ήταν αναμενόμενο) αυτός είναι δομημένος όπως ακριβώς ένα σύστημα επεξεργασίας στοιχείων (ενότητα 1.2.1)

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ0

```
ΔΕΔΟΜΕΝΑ
    N, X, I: INTEGER ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
    X := 1 ;
```

```
    ΓΙΑ I:= 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        X := "ΕΠΟΜΕΝΟΣ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ" ;
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ ("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
ΤΕΛΟΣ
```

Στην πρώτη εκλέπτυνση, αναλύουμε την οδηγία X := "ΕΠΟΜΕΝΟΣ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ", στην οποία ουσιαστικά ο αλγόριθμος πρέπει να αποφασίσει εάν ένας αριθμός X είναι πρώτος ή όχι. Έτσι, εισάγουμε τη Boolean μεταβλητή PRIME, η οποία είναι αληθής (TRUE), όταν ο X είναι πρώτος, και ψευδής (FALSE), όταν δεν είναι (αντί των εκφράσεων PRIME = TRUE και PRIME = FALSE, χρησιμοποιούνται οι προγραμματιστικά ορθότερες εκφράσεις PRIME και NOT PRIME). Ο αλγόριθμος γίνεται (σε κάθε βήμα εκλέπτυνσης, οι προσθήκες ή διορθώσεις δείχνονται με *πλάγια γράμματα*):

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ1

```
ΔΕΔΟΜΕΝΑ
    N, X, I: INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
    X := 1 ;
    ΓΙΑ I:= 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΕΠΑΝΕΛΑΒΕ
            X := X + 1 ;
            PRIME := "X ΕΙΝΑΙ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ"
        ΜΕΧΡΙ (PRIME) ;
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ ("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
ΤΕΛΟΣ
```

Πρώτος στον κατάλογο των πρώτων αριθμών είναι το 2. Όλοι οι υπόλοιποι πρώτοι αριθμοί είναι περιττοί (γιατί;). Έτσι, μπορούμε να θεωρήσουμε το 2 σαν ειδική περίπτωση και, στη συνέχεια, να υπολογί-

σουμε τους υπόλοιπους αριθμούς αυξάνοντας το βήμα της επανάληψης κατά 2. Το επόμενο βήμα εκλέπτυνσης του αλγορίθμου έχει ως εξής:

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ2

```
ΔΕΔΟΜΕΝΑ
    N, X, I: INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
    X := 1 ;
    ΕΑΝ (N >= 1) ΤΟΤΕ
        ΤΥΠΩΣΕ ("2")
    ΕΑΝ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 2 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΕΠΑΝΕΛΑΒΕ
            X := X + 2 ;
            PRIME := "Χ ΕΙΝΑΙ ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ"
        ΜΕΧΡΙ (PRIME) ;
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ ("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
ΤΕΛΟΣ
```

Στη συνέχεια, περιγράφουμε τον τρόπο με τον οποίο ο αλγόριθμος υπολογίζει εάν ο Χ είναι πρώτος αριθμός: ένας πρώτος αριθμός διαιρείται ακριβώς (δηλαδή αφήνοντας υπόλοιπο 0) μόνο με τον εαυτό του και τη μονάδα. Συνεπώς, για κάθε αριθμό Χ, εξετάζουμε το υπόλοιπο της διαίρεσής του με όλους τους αριθμούς από 2 έως Χ-1: Εάν αυτό ποτέ δεν είναι 0, τότε ο αριθμός είναι πρώτος. Χρησιμοποιούμε τη μεταβλητή Κ για να μετρήσουμε κάθε φορά αυτούς τους αριθμούς. Η δομή επανάληψης τερματίζεται είτε μόλις η PRIME γίνει FALSE (δηλαδή το υπόλοιπο της διαίρεσης της Χ με κάποιον από τους αριθμούς γίνει 0) είτε μόλις τελειώσουν οι αριθμοί. Η σύνθετη συνθήκη εξόδου από την επανάληψη ορίζεται με τον τελεστή OR που συνδέει τις επί μέρους συνθήκες και σημαίνει ότι η επανάληψη θα τερματιστεί μόλις ισχύσει κάποια από τις δύο συνθήκες.

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ3

```
ΔΕΔΟΜΕΝΑ
    N, X, I, K: INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
    X := 1 ;
    ΕΑΝ (N >= 1) ΤΟΤΕ
        ΤΥΠΩΣΕ ("2")
    ΕΑΝ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 2 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΕΠΑΝΕΛΑΒΕ
            X := X + 2 ;
            K := 2 ;
            ΕΠΑΝΕΛΑΒΕ
                K := K + 1 ;
                PRIME := "Χ ΔΕΝ ΔΙΑΙΡΕΙΤΑΙ ΜΕ Κ"
            ΜΕΧΡΙ ((NOT PRIME) OR (K = X - 1))
        ΜΕΧΡΙ (PRIME) ;
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ ("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
ΤΕΛΟΣ
```

Έφτασε η ώρα να αποφασίσουμε πώς ακριβώς βρίσκουμε εάν ένας αριθμός είναι πρώτος ή όχι. Έχουμε αναβάλλει αυτή τη σχεδιαστική απόφαση έως την τελευταία στιγμή, ώστε να επιλέξουμε την καλύτερη λύση, που είναι η χρήση του τελεστή mod, ο οποίος δίνει το υπόλοιπο της διαίρεσης δύο αριθμών:

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ4

```
ΔΕΔΟΜΕΝΑ
    N, X, I, K: INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
```

```
    X := 1 ;
    ΕΑΝ (N >= 1) ΤΟΤΕ
        ΤΥΠΩΣΕ ("2")
    ΕΑΝ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 2 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΕΠΑΝΕΛΑΒΕ
            X := X + 2 ;
            K := 2 ;
            ΕΠΑΝΕΛΑΒΕ
                K := K + 1 ;
                ΕΑΝ ((X mod K) = 0) ΤΟΤΕ
                    PRIME := FALSE
                ΑΛΛΙΩΣ
                    PRIME := TRUE
                ΕΑΝ-ΤΕΛΟΣ
            ΜΕΧΡΙ ((NOT PRIME) OR (K = X - 1))
        ΜΕΧΡΙ (PRIME) ;
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ ;
    ΤΥΠΩΣΕ ("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
ΤΕΛΟΣ
```

Αυτός είναι ο τελικός αλγόριθμος που υπολογίζει τους αρχικούς Ν πρώτους αριθμούς και τους τυπώνει στην οθόνη. Είναι τεκμηριωμένα ορθός, αφού η κάθε νέα έκδοση προκύπτει με σωστό τρόπο από την προηγούμενη. Είναι πλήρης γιατί κάθε οδηγία μπορεί άμεσα να μεταφραστεί σε μια εντολή προγράμματος. Δεν είναι όμως ο πιο αποδοτικός αλγόριθμος που θα μπορούσαμε να βρούμε.

Για παράδειγμα, μπορούμε να βελτιώσουμε τον αλγόριθμο, εάν παρατηρήσουμε ότι για κάθε αριθμό Χ δεν χρειάζεται να ψάχνουμε πέραν του μεγαλύτερου φυσικού αριθμού, ο οποίος δεν υπερβαίνει την τετραγωνική ρίζα του Χ. Έτσι, εάν η συνάρτηση SQRT(X) επιστρέφει την τετραγωνική ρίζα ενός θετικού αριθμού Χ και η συνάρτηση INT(X) επιστρέφει το ακέραιο μέρος ενός πραγματικού αριθμού Χ, τότε η συνθήκη τερματισμού της εσωτερικής επανάληψης μπορεί να γραφεί ως:

```
(NOT PRIME) OR (K > INT(SQRT(X)))
```

Μία άλλη βελτίωση βασίζεται στο γεγονός ότι κάθε μη πρώτος αριθ-

μός μπορεί να εκφραστεί ως γινόμενο πρώτων αριθμών. Έτσι, εάν ένας αριθμός Χ διαιρείται ακριβώς με ένα μη πρώτο αριθμό Κ, τότε διαιρείται ακριβώς και με τους πρώτους αριθμούς που αποτελούν παράγοντες του Κ. Συνεπώς, δεν χρειάζεται να ελέγχουμε εάν κάθε αριθμός Χ διαιρείται ακριβώς με κάποιον αριθμό Κ ανάμεσα στο 3 και INT(SQRT(X)), αλλά αρκεί για κάθε αριθμό Χ να ελέγχουμε εάν διαιρείται ακριβώς με κάποιον από τους πρώτους αριθμούς που έχουμε ήδη βρει, τους οποίους φροντίζουμε να διατηρούμε σε ένα πίνακα Ρ. Σε τέτοια περίπτωση, μπορούμε να τυπώσουμε στο τέλος του προγράμματος τα στοιχεία του πίνακα, αντί να τυπώνουμε κάθε πρώτο αριθμό μόλις τον βρίσκουμε. Έτσι, στο επόμενο βήμα προκύπτει η ακόλουθη βελτιωμένη έκδοση του αλγορίθμου:

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ5

```
ΔΕΔΟΜΕΝΑ
    N, X, I, K, LIM: INTEGER ;
    P: ARRAY [1..N] OF INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(N) ;
    X := 1 ;
    LIM := 1 ;
    ΕΑΝ (N >= 1) ΤΟΤΕ
        P[1] := 2
    ΕΑΝ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 2 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΕΠΑΝΕΛΑΒΕ
            X := X + 2 ;
            K := 2 ;
            PRIME := TRUE ;
            ΕΝΟΣΩ ((PRIME) AND (K < LIM)) ΕΠΑΝΕΛΑΒΕ
                ΕΑΝ ((X mod P[K]) = 0) ΤΟΤΕ
                    PRIME := FALSE
                ΕΑΝ-ΤΕΛΟΣ ;
                K := K + 1
            ΕΝΟΣΩ-ΤΕΛΟΣ
        ΜΕΧΡΙ (PRIME) ;
```

```
        P[I] := X ;
        LIM := LIM + 1
    ΓΙΑ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        ΤΥΠΩΣΕ P[I]
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Με τη χρήση της ΚΒΕ, ένα πρόβλημα διασπάται σε μικρότερα κομμάτια, τα οποία μπορούμε να λύσουμε πιο εύκολα, ενώ, με την κατά το δυνατό αναβολή των τελικών σχεδιαστικών αποφάσεων, ελαχιστοποιείται η ποσότητα λεπτομέρειας που πρέπει κάθε φορά να αντιμετωπίσουμε. Έτσι, η σκέψη μας επικεντρώνεται κάθε φορά σε ένα επιμέρους τμήμα του προβλήματος, ενώ μπορούμε να τεκμηριώσουμε ότι η συνολική λύση έχει προκύψει με ορθό τρόπο από τον εμπλουτισμό των επιμέρους λύσεων. Η σχεδίαση λογισμικού που χρησιμοποιεί την ΚΒΕ καλείται «σχεδίαση από πάνω προς τα κάτω» και θα καλυφθεί στο κεφάλαιο 4.

Δραστηριότητα 3.1

Οι αριθμοί Fibonacci είναι μια ακολουθία αριθμών οι οποίοι προκύπτουν ως εξής: Οι πρώτοι δύο αριθμοί είναι οι 0 και 1 και κάθε επόμενος αριθμός προκύπτει αθροίζοντας τους δύο προηγούμενους αριθμούς. Ενδεικτικά, οι πρώτοι πέντε αριθμοί της σειράς είναι: 0, 1, 1, 2, 3. Δοκιμάστε χρησιμοποιώντας ΚΒΕ να περιγράψετε έναν αλγόριθμο, ο οποίος θα υπολογίζει τους Ν πρώτους αριθμούς Fibonacci. Η δική μας πρόταση παρατίθεται στη συνέχεια.

Η πρώτη έκδοση του αλγορίθμου είναι σχετικά απλή:

```
ΑΛΓΟΡΙΘΜΟΣ FIB-ΕΚΔ0
ΔΕΔΟΜΕΝΑ
    I, N, X: INTEGER ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ (N);
    ΓΙΑ I:=1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        X:= "ΕΠΟΜΕΝΟΣ ΑΡΙΘΜΟΣ FIBONACCI";
        ΤΥΠΩΣΕ (X)
```

```
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Όπως έχουμε αναφέρει, οι δύο πρώτοι αριθμοί της σειράς είναι οι 0 και 1 (η σειρά Fibonacci δεν έχει έννοια με λιγότερα από τρία μέλη). Συνεπώς, προσαρμόζουμε τον αλγόριθμό μας ως εξής:

```
ΑΛΓΟΡΙΘΜΟΣ FIB-ΕΚΔ1
ΔΕΔΟΜΕΝΑ
    I, N, X: INTEGER;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ (N);
    ΤΥΠΩΣΕ ("0");
    ΤΥΠΩΣΕ ("1");
    ΓΙΑ I := 3 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        X := "ΕΠΟΜΕΝΟΣ ΑΡΙΘΜΟΣ FIBONACCI";
        ΤΥΠΩΣΕ (X)
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Ας προσπαθήσουμε τώρα να περιγράψουμε πώς υπολογίζεται ο επόμενος αριθμός στη σειρά. Αφού αυτός προκύπτει από το άθροισμα των δύο τελευταίων αριθμών στη σειρά, πρέπει κάθε φορά να αποθηκεύουμε αυτούς τους αριθμούς. Για το σκοπό αυτό χρησιμοποιούμε τις μεταβλητές LAST (η οποία αποθηκεύει τον προ-τελευταίο αριθμό) και PREV (η οποία στην αρχή κάθε επανάληψης αποθηκεύει τον τελευταίο αριθμό). Προσέξτε ότι στην αρχή κάθε επανάληψης υπολογίζουμε τον επόμενο αριθμό της σειράς και τον αποθηκεύουμε σε μια μεταβλητή προσωρινής αποθήκευσης (TEMP). Έπειτα, αλλάζουμε τις τιμές των LAST και PREV, ώστε να είμαστε έτοιμοι για τον υπολογισμό του επόμενου αριθμού (εάν βέβαια χρειαστεί).

```
ΑΛΓΟΡΙΘΜΟΣ FIB-ΕΚΔ2
ΔΕΔΟΜΕΝΑ
    I, N, PREV, LAST, TEMP: INTEGER;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ (N);
```

```
    ΤΥΠΩΣΕ ("0");
    ΤΥΠΩΣΕ ("1");
    LAST := 0;
    PREV := 1;
    ΓΙΑ I:=3 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        TEMP := LAST + PREV;
        LAST := PREV;
        PREV := TEMP;
        ΤΥΠΩΣΕ (PREV)
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

3.2 Αρθρωτός προγραμματισμός

Κατά τον **αρθρωτό προγραμματισμό (modular programming),** ένα σύστημα λογισμικού συντίθεται από τμήματα λογισμικού (software modules). Με τον τρόπο αυτό υλοποιείται η αρχή της διάσπασης ενός πολύπλοκου προβλήματος σε μικρότερα και απλούστερα προβλήματα.

> Ένα τμήμα λογισμικού είναι ένα μετρίου μεγέθους υπο-πρόγραμμα το οποίο εκτελεί μια συγκεκριμένη λειτουργία. Ένα τμήμα είναι ανεξάρτητο και αυτόνομο σε σχέση με το υπόλοιπο σύστημα λογισμικού, ώστε η αφαίρεσή του από το σύστημα να απενεργοποιεί τη συγκεκριμένη λειτουργία μόνο, χωρίς άλλες επιπτώσεις

Ένα τμήμα λογισμικού έχει τα ακόλουθα χαρακτηριστικά (Fairley, 1985):

- Περιέχει εντολές και δομές δεδομένων
- Μπορεί να μεταγλωττιστεί ανεξάρτητα και να αποθηκευθεί σε μια βιβλιοθήκη
- Μπορεί να συμπεριλαμβάνεται σε ένα πρόγραμμα, το οποίο θα χρησιμοποιεί τις λειτουργίες του τμήματος καλώντας τις με κάποιο όνομα και μια λίστα παραμέτρων
- Μπορεί να χρησιμοποιεί άλλα τμήματα

Τα πλεονεκτήματα και τα μειονεκτήματα του αρθρωτού προγραμματισμού συνοψίζονται στον Πίνακα 3.1 (Shooman, 1983). Η σχεδίαση λογισμικού με αυτά τα χαρακτηριστικά καλείται τμηματοποιημένη (αρθρωτή) σχεδίαση και θα καλυφθεί στο κεφάλαιο 4.

Πίνακας 3.1

Πλεονεκτήματα και μειονεκτήματα του αρθρωτού προγραμματισμού

Πλεονεκτήματα	Μειονεκτήματα
Η χρήση αρθρωτού προγραμματισμού προάγει την επαναχρησιμοποίηση κώδικα και οδηγεί στη μείωση του κόστους ανάπτυξης.	Θα χρειαστεί να καταβάλουμε μεγαλύτερο κόπο και προσοχή κατά τη σχεδίαση
Είναι πιο εύκολο και φθηνό να αλλάξουμε ή να βελτιώσουμε το πρόγραμμα μετά την παράδοσή του.	Είναι δύσκολη η εκμάθηση και η εφαρμογή της τεχνικής, παρόλο που οι αρχές της είναι ξεκάθαρες.
Είναι πιο εύκολο να κατανοήσουμε τη λειτουργία ενός αρθρωτού προγράμματος διαβάζοντας τον κώδικά του.	Η συσχέτιση τμήμα-προγραμματιστής δεν είναι πάντα επιθυμητή, ιδιαίτερα όταν χρησιμοποιείται ομάδα ικανών προγραμματιστών.
Είναι πιο εύκολο να διαχειριστούμε την ανάπτυξη του προγράμματος (π.χ., δίνοντας τα δυσκολότερα τμήματα σε πιο έμπειρους προγραμματιστές).	Ένα αρθρωτό πρόγραμμα συνήθως απαιτεί περισσότερο χώρο και χρόνο για την εκτέλεσή του.
Είναι πιο εύκολο να αναπτύξουμε και να δοκιμάσουμε/ διορθώσουμε το πρόγραμμα.	
Μπορούμε να διασπάσουμε ένα μεγάλο, πολύπλοκο πρόβλημα σε ένα αριθμό τμημάτων μικρότερης πολυπλοκότητας.	
Ο αρθρωτός προγραμματισμός ακολουθεί «φυσιολογικά» την από πάνω προς τα κάτω σχεδίαση, αλλά η τυποποιημένη περιγραφή του τρόπου κλήσης των τμημάτων είναι εξαιρετικά χρήσιμη όταν ακολουθείται σχεδίαση από κάτω προς τα πάνω ή από τη μέση προς την άκρη.	

Παράδειγμα 3.2

Επιστρέφουμε στο πρόβλημα του υπολογισμού των αρχικών Ν πρώτων αριθμών. Εάν, αντί της ΚΒΕ που χρησιμοποιήσαμε στην ενότητα 3.1, αποφασίσουμε να χρησιμοποιήσουμε αρθρωτό προγραμματισμό, τότε από τον αλγόριθμο ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-ΕΚΔ0 συμπεραί-

νουμε ότι το πρόγραμμα αποτελείται από τρία τμήματα: Το πρώτο διαβάζει τα δεδομένα εισόδου, το δεύτερο υπολογίζει εάν κάποιος αριθμός είναι πρώτος και το τρίτο τυπώνει τον κατάλογο με τους πρώτους αριθμούς. Έτσι, ο αλγόριθμος γίνεται ως εξής:

ΑΛΓΟΡΙΘΜΟΣ ΠΡΩΤΟΙ-ΑΡΙΘΜΟΙ-MOD

```
ΔΕΔΟΜΕΝΑ
    N, X, I: INTEGER ;
    P: ARRAY [1..N] OF INTEGER ;
    PRIME: BOOLEAN ;
ΑΡΧΗ
    ΥΠΟΛΟΓΙΣΕ ΔΙΑΒΑΣΕ-ΔΕΔΟΜΕΝΑ (N) ;
    X := 1 ;
    ΕΑΝ (N >= 1) ΤΟΤΕ
        P[1] := 2
    ΕΑΝ-ΤΕΛΟΣ ;
    ΓΙΑ I:= 2 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
        PRIME := FALSE ;
        ΕΠΑΝΕΛΑΒΕ
            X := X + 2 ;
            ΥΠΟΛΟΓΙΣΕ ΕΙΝΑΙ-ΠΡΩΤΟΣ (X, PRIME);
        ΜΕΧΡΙ (PRIME) ;
        P[I] := X
    ΓΙΑ-ΤΕΛΟΣ
    ΥΠΟΛΟΓΙΣΕ ΤΥΠΩΣΕ-ΑΡΙΘΜΟΥΣ
ΤΕΛΟΣ ;
```

Κατ' αρχήν, παρατηρήστε ότι εξακολουθεί να υφίσταται ένας κυρίως αλγόριθμος, ο οποίος χρησιμοποιεί τρία τμήματα, καλώντας τα με το όνομά τους και ένα σύνολο παραμέτρων (για να δειχθεί η κλήση τμήματος χρησιμοποιείται η οδηγία ΥΠΟΛΟΓΙΣΕ).

Το τμήμα ΔΙΑΒΑΣΕ-ΔΕΔΟΜΕΝΑ(N) επιστρέφει το πλήθος των αρχικών πρώτων αριθμών που πρέπει να υπολογίσει το πρόγραμμα. Χρησιμοποιώντας τμήμα αντί για μια εντολή, το πρόγραμμά μας γίνεται ανεξάρτητο από τον τρόπο εισόδου του N (μπορεί να γίνει από το πληκτρολόγιο, από αρχείο δεδομένων κλπ).

Το τμήμα ΕΙΝΑΙ-ΠΡΩΤΟΣ(X, PRIME) δέχεται ως είσοδο έναν αριθμό

Χ και επιστρέφει τη Boolean μεταβλητή PRIME με τιμή TRUE εάν αυτός είναι πρώτος ή FALSE εάν δεν είναι. Με τον τρόπο αυτό το πρόγραμμά μας είναι ανεξάρτητο από τον τρόπο υπολογισμού των πρώτων αριθμών, για τον οποίο θα μπορούσαν να χρησιμοποιηθούν διάφοροι αλγόριθμοι. Έτσι, εάν ανακαλύψουμε έναν καλύτερο αλγόριθμο υπολογισμού πρώτων αριθμών, τότε στο κυρίως πρόγραμμά μας δεν θα αλλάξει τίποτε.

Το τμήμα ΤΥΠΩΣΕ-ΑΡΙΘΜΟΥΣ τυπώνει τους πρώτους αριθμούς διαβάζοντας εσωτερικά τον πίνακα P. Χρησιμοποιώντας τμήμα, το πρόγραμμά μας γίνεται ανεξάρτητο από το μέσο εξόδου (μπορεί να είναι η οθόνη, ο εκτυπωτής, ένα αρχείο κλπ), ενώ αποκρύπτεται και ο τρόπος προσπέλασης της εσωτερικής δομής αποθήκευσης των αριθμών (του πίνακα P).

Δραστηριότητα 3.2

Πριν προχωρήσετε, προσπαθήστε να γράψετε τον αλγόριθμο που βρίσκει το μεγαλύτερο από τρεις αριθμούς (ενότητα 2.3) χρησιμοποιώντας ως τμήμα τον αλγόριθμο που βρίσκει το μεγαλύτερο από δύο αριθμούς.

Αντίστοιχα με το προηγούμενο παράδειγμα, ο αλγόριθμος που βρίσκει το μεγαλύτερο από τρεις αριθμούς (έχει παρουσιαστεί στην ενότητα 2.3) μπορεί να ξαναγραφεί χρησιμοποιώντας ως τμήμα τον αλγόριθμο που βρίσκει το μεγαλύτερο από δύο αριθμούς (μόνο που αυτός θα χρειαστεί μια τροποποίηση ώστε να διαβάζει ως παραμέτρους εισόδου τους δύο αριθμούς προς σύγκριση):

```
ΑΛΓΟΡΙΘΜΟΣ MAX-XYZ-MOD
ΔΕΔΟΜΕΝΑ
    X,Y,Z: REAL;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ (X,Y,Z);
    ΕΑΝ (X > Y) ΤΟΤΕ
        ΥΠΟΛΟΓΙΣΕ MAX-XY (X,Z)
    ΑΛΛΙΩΣ
        ΥΠΟΛΟΓΙΣΕ MAX-XY (Y,Z)
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

3.3 Τεχνοτροπίες (στυλ) προγραμματισμού

Συνήθως λέμε ότι ένας προγραμματιστής έχει το δικό του στυλ προγραμματισμού όταν με συνέπεια ακολουθεί ένα συγκεκριμένο «χάρτη προγραμματιστικών επιλογών» ανάμεσα σε ένα σύνολο εναλλακτικών τρόπων υλοποίησης ενός προγράμματος, για κάθε πρόγραμμα στο οποίο συμμετέχει.

Το στυλ προγραμματισμού μπορεί να περιγράφεται από μια λίστα με οδηγίες της μορφής «κάνε ...» ή «μην κάνεις ...», ή μπορεί να διέπεται από αρχές που υλοποιούν κάποια συγκεκριμένη προσέγγιση στην ανάπτυξη λογισμικού (στη βιβλιογραφία – π.χ. (Fairley, 1985– Shooman, 1983) – μπορείτε να βρείτε ένα μεγάλο σύνολο προτρεπτικών και αποτρεπτικών οδηγιών – ευτυχώς οι συγγραφείς συμφωνούν στις βασικές οδηγίες!). Αν και οι οδηγίες αυτές αποτελούν έναν δοκιμασμένο κατάλογο ελέγχου για κάθε πρόγραμμα και ένα καλό σημείο αφετηρίας για οποιοδήποτε προγραμματιστή, οι έμπειροι προγραμματιστές αναπτύσσουν τους δικούς τους καταλόγους, ως μέρος του προσωπικού στυλ προγραμματισμού.

Όταν το στυλ προγραμματισμού εφαρμοστεί αρκετά στην πράξη, ο προγραμματιστής κατασταλάζει σε μια συγκεκριμένη προσέγγιση στην ανάπτυξη προγραμμάτων. Ο στόχος τέτοιων προσεγγίσεων είναι να διασφαλίσουν υψηλότερη ποιότητα του παραγόμενου λογισμικού με αντίστοιχη μείωση του κόστους. Οι τρεις περισσότερο διαδεδομένες προσεγγίσεις περιγράφονται στη συνέχεια.

3.3.1 Προγραμματισμός για επαναχρησιμοποίηση

Εάν πιστεύετε ότι το λογισμικό που αναπτύσσετε πρέπει να είναι ταυτόχρονα οικονομικό και αξιόπιστο, τότε καλύτερα να εφαρμόσετε **συστηματική επαναχρησιμοποίηση (reuse)** των «κεφαλαίων λογισμικού» που διαθέτετε. Σε αυτά περιλαμβάνονται ολόκληρες εφαρμογές, συστατικά τμήματα (ψηφίδες) λογισμικού, αλλά και άλλα προϊόντα του κύκλου ανάπτυξης λογισμικού, όπως προδιαγραφές και σχέδια.

Περισσότερο συνηθισμένη (και εμπορικά ενδιαφέρουσα) είναι η επαναχρησιμοποίηση ψηφίδων λογισμικού (software components). Πολλές φορές όμως, η επαναχρησιμοποίηση προδιαγραφών και σχεδίων λογισμικού αποδεικνύεται αρκετά χρήσιμη, αφού τα προϊόντα αυτά

είναι από κατασκευής γενικά και ανεξάρτητα από την τελική υλοποίηση του λογισμικού.

Τέτοιες ψηφίδες, συνήθως, προκύπτουν από τμήματα κώδικα που είχαν αρχικά αναπτυχθεί για κάποια συγκεκριμένη εφαρμογή. Ο κώδικας αυτός χρησιμοποιήθηκε με τις απαραίτητες προσαρμογές και σε άλλες εφαρμογές, οπότε η αξιοπιστία του έχει αποδειχθεί στην πράξη. Για να μετατραπεί αυτός ο κώδικας σε επαναχρησιμοποιήσιμη ψηφίδα λογισμικού, πρέπει να γενικευθούν κάποια χαρακτηριστικά του, ώστε να μπορεί να ενσωματωθεί σε οποιαδήποτε εφαρμογή. Σε αυτά περιλαμβάνονται τα ονόματα των μεταβλητών, ο τρόπος δήλωσης και κλήσης των ρουτινών, και η διαχείριση των σφαλμάτων.

Βέβαια, λίγες εφαρμογές θα χρησιμοποιήσουν όλες τις δυνατότητες μιας ψηφίδας λογισμικού. Επειδή όμως οι ανάγκες μιας εφαρμογής δεν μπορούν να προβλεφθούν, ο σχεδιαστής ψηφίδων πρέπει να βρει τρόπο ώστε να παράσχει ένα ελάχιστο αλλά ταυτόχρονα αποτελεσματικό σύνολο λειτουργιών.

Το κόστος της μετατροπής υπάρχοντος κώδικα σε επαναχρησιμοποιήσιμη ψηφίδα λογισμικού είναι γενικά υψηλό, γι' αυτό και δεν προτιμάται από τους διαχειριστές των έργων λογισμικού. Από την άλλη πλευρά, η εξαρχής σχεδίαση ψηφίδων λογισμικού αυξάνει το βραχυπρόθεσμο κόστος ανάπτυξης, αλλά αποφέρει μακροπρόθεσμα οφέλη, κυρίως όταν ευνοείται η ανάπτυξη εφαρμογών με τη χρήση επαναχρησιμοποιούμενων ψηφίδων λογισμικού. Σε αυτά συμπεριλαμβάνονται η αύξηση της αξιοπιστίας του συστήματος, η μείωση του ρίσκου κατά την ανάπτυξη, η ενσωμάτωση προτύπων στο λογισμικό, η ελάττωση του χρόνου ανάπτυξης και η εκμετάλλευση της εμπειρίας (Sommerville, 1996). Σε κάθε περίπτωση, η υιοθέτηση των αρχών του αρθρωτού προγραμματισμού διευκολύνει την επαναχρησιμοποίηση κώδικα και τη δημιουργία ψηφίδων

Δραστηριότητα 3.3

Υποθέστε ότι ένα μετεωρολογικό εργαστήριο διαθέτει ένα σύστημα αισθητήρων με το οποίο μετρά κάθε ημέρα τη μέγιστη θερμοκρασία που σημειώνεται στην περιοχή σας. Το πρόγραμμα που χειρίζεται τους αισθητήρες καταχωρεί τις θερμοκρασίες σε βαθμούς Φαρενάιτ σε ένα πίνακα 366 θέσεων. Με βάση τα προγράμματα που υλοποιούν τους αλγορίθμους που παρουσιάστηκαν στο κεφάλαιο 2, προσπαθήστε να συνθέσετε ένα πρόγραμμα (μπορείτε να χρησιμοποιήσετε ψευδοκώδικα για την περιγραφή του), το οποίο στο τέλος ενός έτους:

- θα διαβάζει τη μέγιστη θερμοκρασία που σημειώθηκε σε κάθε ημέρα του έτους
- θα τη μετατρέπει σε βαθμούς Κελσίου και θα την καταχωρεί στην ίδια θέση του πίνακα, και
- στο τέλος θα ταξινομεί τις θερμοκρασίες και θα βρίσκει τη μέση θερμοκρασία του έτους.

Ποιες ψηφίδες θα χρησιμοποιήσετε ;

Χρησιμοποιούμε τρεις ψηφίδες: (α) για τη μετατροπή των βαθμών Φαρενάιτ σε Κελσίου, (β) για την ταξινόμηση του πίνακα των θερμοκρασιών και (γ) για την εύρεση της μέσης θερμοκρασίας του έτους. Τα σχετικά προγράμματα θα πρέπει να γραφούν έτσι, ώστε να δέχονται τις αντίστοιχες μεταβλητές εισόδου και εξόδου και να κάνουν εσωτερική διαχείριση σφαλμάτων (η επικοινωνία του κυρίως προγράμματος με τις ψηφίδες περιγράφεται στις δηλώσεις του τμήματος ΔΙΕΠΑΦΗ).

ΑΛΓΟΡΙΘΜΟΣ ΘΕΡΜΟΚΡΑΣΙΕΣ-ΕΤΟΥΣ

```
ΔΕΔΟΜΕΝΑ
    MO: REAL
    T: ARRAY [1..366] OF REAL;
ΑΡΧΗ
    ΨΗΦΙΔΑ F-ΣΕ-C
    ΔΙΕΠΑΦΗ
            ΕΙΣΟΔΟΣ:   T ;
            ΕΞΟΔΟΣ:    T ;
```

```
ΨΗΦΙΔΑ ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ:   Τ ;
        ΕΞΟΔΟΣ:    Τ ;

ΨΗΦΙΔΑ ΜΟ-ΠΙΝΑΚΑ-1ΧΝ
ΔΙΕΠΑΦΗ
    ΕΙΣΟΔΟΣ:   Τ ;
    ΕΞΟΔΟΣ:    ΜΟ ;

ΤΕΛΟΣ
```

3.3.2 Προγραμματισμός με πλεονασμό

Εάν πιστεύετε ότι το λογισμικό πρέπει σε κάθε περίπτωση να παραμένει αξιόπιστο και ποτέ να μην σταματά να λειτουργεί, τότε πρέπει να σκεφτείτε την υιοθέτηση του **προγραμματισμού με πλεονασμό (redundant programming),** ο οποίος προτείνει την ανάπτυξη και χρήση τουλάχιστον δύο διαφορετικών προγραμμάτων για την επίλυση του ίδιου προβλήματος (Shooman, 1983).

Για παράδειγμα, για τον υπολογισμό των πραγματικών τιμών των ριζών μιας δευτεροβάθμιας εξίσωσης, αναπτύσσουμε δύο προγράμματα: ένα που χρησιμοποιεί τη μέθοδο της διακρίνουσας και ένα που υλοποιεί την προσεγγιστική μέθοδο Newton-Raphson. Σε κάποιο πραγματικό πρόβλημα εκτελούμε και τα δύο προγράμματα. Εάν οι απαντήσεις είναι πολύ κοντά, τότε ως απάντηση θεωρούμε τη μέση τιμή τους. Εάν απέχουν, τότε κάποιο από τα δύο προγράμματα έχει κάνει λάθος υπολογισμό, οπότε καλό είναι να επαληθεύσουμε το αποτέλεσμα με μια τρίτη μέθοδο (λογική της πλειοψηφίας).

Αν και θα περιμέναμε ότι το κόστος ανάπτυξης δύο προγραμμάτων για το ίδιο πρόβλημα θα ήταν διπλάσιο από αυτό της ανάπτυξης ενός, στην πραγματικότητα είναι λιγότερο από μιάμιση φορά μεγαλύτερο, αφού πολλές φάσεις της ανάπτυξης είναι κοινές και στα δύο προγράμματα (τμήμα της σχεδίασης, ο έλεγχος, κλπ).

3.3.3 Αμυντικός προγραμματισμός

Εάν πιστεύετε ότι το λογισμικό πάντα έχει λάθη που δεν ανιχνεύονται, τότε η **αμυντική προσέγγιση (defensive programming)** σας ταιριά-

ζει. Φροντίστε, λοιπόν, στα προγράμματα που αναπτύσσετε να ενσωματώνετε όσο το δυνατό περισσότερα επιπλέον τμήματα κώδικα, τα οποία είτε θα κάνουν έλεγχο για λάθη πριν αυτά συμβούν, είτε διαχείριση σφαλμάτων, όταν αυτά συμβούν (αφού έτσι κι αλλιώς, κάποτε θα συμβούν!).

Στην πρώτη περίπτωση πρόκειται για πρόληψη αστοχίας (failure prevention), ενώ στη δεύτερη μιλάμε για ανάνηψη μετά από αστοχία (failure recovery). Ενδιάμεσα, μεσολαβεί μια φάση εκτίμησης των ζημιών (damage assessment), κατά την οποία εκτιμάται η έκταση της αστοχίας και τα τμήματα του προγράμματος που έχουν επηρεαστεί, ώστε να σχεδιαστεί η διαδικασία ανάνηψης (Shooman, 1983, Sommerville, 1996).

Οι τεχνικές αμυντικού προγραμματισμού είναι ενεργητικές ή παθητικές. Στη δεύτερη κατηγορία ανήκουν τεχνικές που ελέγχουν για λάθος, όταν το πρόγραμμα φτάσει σε σημείο εκτέλεσης του κώδικα ελέγχου (π.χ., όταν σε ένα πρόγραμμα δανειστικής βιβλιοθήκης καταχωρείται η ημερομηνία επιστροφής για κάποιο βιβλίο, αυτό πρέπει να ελέγχει ότι ο αριθμός που αναπαριστά το μήνα είναι ανάμεσα στους 1 και 12, ότι η ημέρα, εάν ο μήνας είναι Ιανουάριος, είναι ανάμεσα στο 1 και 31, κ.λπ.).

Οι τεχνικές της πρώτης κατηγορίας ενσωματώνουν στο πρόγραμμα κώδικα ο οποίος εκτελείται με πρωτοβουλία του προγράμματος (σε καθορισμένα χρονικά διαστήματα ή όταν ο φόρτος εκτέλεσης είναι μικρός) και ελέγχει την ορθότητα των μεταβλητών κατάστασης, των συνθηκών, των δεδομένων κ.λπ.

Βέβαια, η ενσωμάτωση επιπλέον κώδικα ελέγχου μεγαλώνει το μέγεθος του προγράμματος, καθυστερεί την παράδοσή του, χειροτερεύει τις επιδόσεις του και γενικά κάνει την ανάπτυξη του προγράμματος ακριβότερη. Έτσι, ο αμυντικός προγραμματισμός είναι καλός μόνο όταν μπορούν να διατεθούν οι αναγκαίοι πόροι, ενώ ταυτόχρονα, η εμφάνιση λαθών κατά τη χρήση του προγράμματος θα κοστίσει ακριβότερα από τη διόρθωσή τους. Σε τέτοια περίπτωση, είναι καλό ο κώδικας άμυνας να συμπεριλαμβάνεται στο πρόγραμμα από τις αρχικές φάσεις ανάλυσης και σχεδίασης.

Ένα άλλο σημαντικό ζήτημα αφορά στο τι ελέγχουμε στο πρόγραμμα.

Εάν ελέγχουμε κάθε υπολογισμό, τότε ουσιαστικά ακολουθούμε προγραμματισμό με πλεονασμό. Για να συμπεράνουμε τι χρειάζεται έλεγχο, καλό είναι να κατασκευάσουμε έναν πίνακα με όλα τα πιθανά λάθη και τις συνέπειές τους και έναν πίνακα με όλους τους πιθανούς ελέγχους και το κόστος τους από πλευράς υλοποίησης και εκτέλεσης. Εάν το πρόγραμμα που αναπτύσσουμε είναι συνηθισμένο, τέτοιους πίνακες μπορεί να βρούμε στη βιβλιογραφία.

Παράδειγμα 3.3

Εάν γράφαμε προγράμματα για τους αλγόριθμους που παρουσιάστηκαν στο κεφάλαιο 2, θα μπορούσαμε να προσθέσουμε τις εξής εντολές αμυντικού προγραμματισμού:

- Στο πρόγραμμα για τον αλγόριθμο ΜΚΔ, πρέπει να ελέγχουμε στην αρχή εάν Y = 0, γιατί σε τέτοια περίπτωση, στη συνέχεια θα γίνει διαίρεση με 0!
- Στο πρόγραμμα για τον αλγόριθμο ΜΟ-ΠΙΝΑΚΑ, πρέπει πάλι να ισχύει N > 0, αλλιώς, θα γίνει διαίρεση με 0.
- Στο πρόγραμμα για τον αλγόριθμο ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ, θα ήταν χρήσιμο να προσθέσουμε μια εντολή εκτύπωσης των περιεχομένων του πίνακα στο τέλος κάθε επανάληψης
- Σε όλα τα προγράμματα πρέπει να προσέχουμε να αρχικοποιούμε κατάλληλα όλες τις μεταβλητές που χρησιμοποιούμε.

Δραστηριότητα 3.4

Συνοψίστε τα πλεονεκτήματα και τα μειονεκτήματα των τριών στυλ προγραμματισμού που μελετήσατε στην ενότητα 3.3. Έπειτα, συγκρίνετε την απάντησή σας με τον Πίνακα 3.2.

3.4 Παραδείγματα προγραμματισμού

Το **παράδειγμα προγραμματισμού (programming paradigm)** καθορίζει τον τρόπο με τον οποίο θα περιγράψουμε στον υπολογιστή τις λειτουργίες που θέλουμε να εκτελέσει για μας, δηλαδή, με άλλα λόγια, ορίζει τον τρόπο με τον οποίο θα συνθέσουμε ένα πρόγραμμα.

Κατά την εξέλιξη του προγραμματισμού έχουν ανακύψει διάφορα παραδείγματα προγραμματισμού, τα οποία είναι δυνατό να κατηγοριοποιηθούν κατά διάφορους τρόπους: εάν απαιτούν ή όχι τη ρητή

Πίνακας 3.2

Συγκριτική παράθεση των τριών στυλ προγραμματισμού

ΣΤΥΛ	ΥΠΕΡ	ΚΑΤΑ
Επαναχρησιμοποίηση	Οικονομία στην ανάπτυξη. Χρήση δοκιμασμένων τμημάτων λογισμικού. Αύξηση αξιοπιστίας. Μείωση ρίσκου. Εκμετάλλευση εμπειρίας.	Υψηλό κόστος μετατροπής υπάρχοντος κώδικα. Αυξημένο βραχυπρόθεσμο κόστος ανάπτυξης. Ειδική σχεδίαση που απαιτεί εμπειρία.
Πλεονασμός	Διασφάλιση συνεχούς λειτουργίας λογισμικού.	Αύξηση κόστους και όγκου κώδικα. Δύσκολα εφαρμόσιμο στυλ.
Αμυντικός	Αύξηση αξιοπιστίας. Διασφάλιση συνεχούς λειτουργίας λογισμικού. Μειωμένες απώλειες μετά από αστοχία.	Αύξηση κόστους και όγκου κώδικα. Χειρότερες επιδόσεις. Προσεκτική σχεδίαση που απαιτεί εμπειρία.

περιγραφή των βημάτων του αλγορίθμου, εάν δίνουν έμφαση στα δεδομένα ή τις διεργασίες, εάν επιτρέπουν ή όχι απόκρυψη δεδομένων, κ.α. Επειδή κάθε πρόγραμμα γράφεται σε μια γλώσσα προγραμματισμού, όλες οι γλώσσες προγραμματισμού υιοθετούν ένα τουλάχιστον προγραμματιστικό παράδειγμα.

Για παράδειγμα, σχετικά με το πρώτο κριτήριο, οι γλώσσες προγραμματισμού κατηγοριοποιούνται σε προστακτικές (imperative) ή δηλωτικές (declarative), ενώ σε σχέση με το τελευταίο, έχουμε τις διαδικαστικές (procedural) και τις αντικειμενοστραφείς (object-oriented) γλώσσες.

Τα τρία περισσότερο διαδεδομένα παραδείγματα προγραμματισμού είναι τα εξής:

- Το **διαδικασιακό παράδειγμα,** το οποίο σε συνδυασμό με το προστακτικό παράδειγμα, θα παρουσιαστεί στο κεφάλαιο 6 στη μορφή του δομημένου προγραμματισμού. Ένα δομημένο πρόγραμμα, όπως θα δούμε στο κεφάλαιο 6, συντίθεται από δομικά τμήματα

(blocks ή schemas (Pintelas, 1978)), καθένα από τα οποία μπορεί να περιλαμβάνει μία ή περισσότερες εντολές. Κάθε εντολή περιγράφει κάποιας μορφής επεξεργασία δεδομένων. Στις διαδικασιακές γλώσσες πρέπει εμείς να περιγράψουμε τα βήματα που πρέπει να ακολουθήσει ο υπολογιστής για να επιλύσει ένα πρόβλημα (δηλαδή, περιγράφουμε τη διαδικασία επίλυσης χρησιμοποιώντας εντολές προγράμματος). Η ανάπτυξη τέτοιων προγραμμάτων διευκολύνεται, εάν υιοθετήσουμε τις τεχνικές λειτουργικής σύνθεσης ή διάσπασης των εργασιών επίλυσης του προβλήματος που θα συναντήσουμε στο κεφάλαιο 4 (δηλ. προγραμματισμό από-πάνω-προς-τα-κάτω ή από-κάτω-προς-τα-πάνω).

- Το **δηλωτικό παράδειγμα,** κυριότερος εκπρόσωπος του οποίου είναι η λογική γλώσσα Prolog. Οι λογικές γλώσσες έχουν μια σημαντική διαφορά από τις διαδικασιακές: αντί να περιγράψουμε τη διαδικασία επίλυσης, χρειάζεται μόνο να περιγράψουμε το στόχο μας (goal), δηλαδή το πρόβλημα που επιχειρούμε να επιλύσουμε, δίνοντας ταυτόχρονα όσες πληροφορίες έχουμε γι' αυτό. Ο μηχανισμός διερμηνείας της γλώσσας είναι αυτός που αναλαμβάνει να συνδυάσει αυτές τις πληροφορίες με άλλες πληροφορίες που ίσως κατέχει και μέσα από μια πολύπλοκη αλλά καλά ορισμένη διαδικασία να μας επιστρέψει την απάντηση.
- Το **αντικειμενοστραφές παράδειγμα** όπως υλοποιείται από διαδεδομένες αντικειμενοστραφείς γλώσσες (C++, Ada, κ.ά.). Οι αντικειμενοστραφείς γλώσσες, παρόλο που είναι προστακτικές, δεν είναι διαδικασιακές, δηλαδή δεν επικεντρώνονται στις λειτουργίες. Αντίθετα, στις γλώσσες αυτές, τα δεδομένα είναι εσωτερικά (τοπικά) σε κάθε τμήμα κώδικα που τα χρησιμοποιεί (λέγεται αντικείμενο). Το συνολικό σύστημα λογισμικού αποτελείται από πολλά τέτοια τμήματα, καθένα από τα οποία περιλαμβάνει δεδομένα και μεθόδους χειρισμού των δεδομένων. Σημειώστε ότι κανένα τμήμα κώδικα δεν μπορεί να έχει απ' ευθείας πρόσβαση στα εσωτερικά δεδομένα άλλου τμήματος.

Σύνοψη

Στο κεφάλαιο αυτό γνωρίσατε δύο πολύ διαδεδομένες πρακτικές προγραμματισμού (την εκλέπτυνση κατά βήμα και τον αρθρωτό προγραμματισμό) και τρεις τεχνοτροπίες υλοποίησης προγραμμάτων (τον προγραμματισμό για επαναχρησιμοποίηση, τον προγραμματισμό με πλεονασμό, και τον αμυντικό προγραμματισμό). Στο επόμενο κεφάλαιο, θα δείτε ότι οι πρακτικές αυτές αποτελούν υλοποίηση ευρύτερων σχεδιαστικών προσεγγίσεων.

Ακόμη, ενημερωθήκατε και για τα περισσότερο διαδεδομένα παραδείγματα προγραμματισμού: το διαδικασιακό παράδειγμα, το δηλωτικό παράδειγμα και το αντικειμενοστραφές παράδειγμα (με το πρώτο θα ασχοληθούμε στα κεφάλαια 6 έως 8).

Βιβλιογραφία Κεφαλαίου 3

[1] Fairley, R. (1985), *Software Engineering Concepts*. McGraw–Hill: Singapore.

[2] Pintelas, P.E. (1978), *Lecture notes on program schemas*. Computer Education journal, June 78.

[3] Sommerville, I. (1996), *Software Engineering*. Addison–Wesley, USA

[4] Shooman, M.L. (1983), *Software Engineering*. McGraw–Hill: Tokyo.

[5] Wirth, N. (1971), *Program development with Stepwise Refinement*. Communications of the ACM, 14(4).

ΚΕΦΑΛΑΙΟ 4

Σχεδίαση προγράμματος

Σκοπός

Το κεφάλαιο αυτό στοχεύει στην εξοικείωσή σας με την αρχή της «σχεδίασης πριν την υλοποίηση ενός προγράμματος», γι' αυτό και παρουσιάζει διάφορες μεθοδολογίες σχεδίασης και κριτήρια ελέγχου της ποιότητας της σχεδίασης. Κατά τη μελέτη να θυμάστε ότι οι μεθοδολογίες αυτές δεν είναι αμοιβαία αποκλειόμενες, αλλά αλληλοσυμπληρώνονται.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *εξηγήσετε γιατί είναι απαραίτητο να σχεδιάζουμε ένα πρόγραμμα πριν το υλοποιήσουμε*
- *εξηγήσετε γιατί η σχεδίαση είναι μια απαιτητική αλλά δημιουργική διαδικασία*
- *διακρίνετε τις δύο κατηγορίες μεθοδολογιών σχεδίασης (ακολουθιακές και επαναληπτικές) με βάση το κριτήριο της σχεδιαστικής διαδικασίας*
- *διακρίνετε τις τρεις κατηγορίες μεθοδολογιών σχεδίασης (από πάνω προς τα κάτω, από κάτω προς τα πάνω, και από τη μέση προς την άκρη) με βάση το κριτήριο της σχεδιαστικής κατεύθυνσης*
- *διακρίνετε τις τρεις κατηγορίες μεθοδολογιών σχεδίασης (βασισμένες στις διεργασίες, βασισμένες στα δεδομένα, και συνδυαστικές (αντικειμενοστραφείς) μεθοδολογίες) με βάση το κριτήριο της μονάδας διάσπασης*
- *περιγράψετε τις έννοιες της λειτουργικής ανεξαρτησίας, της σύζευξης και της συνοχής*
- *προσδιορίσετε το βαθμό σύζευξης και συνοχής που έχει ένα σχέδιο προγράμματος*

Έννοιες κλειδιά

- *Σχεδίαση προγράμματος,*
- *Ακολουθιακές μεθοδολογίες σχεδίασης,*

- *Επαναληπτικές μεθοδολογίες σχεδίασης*
- *Σχεδίαση από πάνω προς τα κάτω*
- *Σχεδίαση από κάτω προς τα πάνω*
- *Σχεδίαση από τη μέση προς την άκρη*
- *Μεθοδολογίες βασισμένες στις διεργασίες*
- *Μεθοδολογίες βασισμένες στα δεδομένα*
- *Συνδυαστικές (αντικειμενοστραφείς) μεθοδολογίες*
- *Λειτουργική ανεξαρτησία*
- *Σύζευξη*
- *Συνοχή*

Εισαγωγικές παρατηρήσεις

Πριν παρουσιάσουμε μεθοδολογίες, τεχνικές και εργαλεία σχεδίασης, θα ήταν καλό να αναρωτηθούμε για την αναγκαιότητα της ***σχεδίασης ενός προγράμματος****. Αφού, όπως έχουμε αναφέρει, χρειαζόμαστε τα προγράμματα για να λύνουμε προβλήματα χρησιμοποιώντας τον υπολογιστή, το ερώτημα αυτό γενικεύεται στο εξής: Αφού γνωρίζω αναλυτικά τα δεδομένα και τις παραμέτρους του προβλήματος, γιατί δεν μπορώ να αρχίσω κατευθείαν να το λύνω, αλλά πρέπει πρώτα να σχεδιάσω τη λύση του;*

Αυτό είναι ένα τόσο γενικό ερώτημα, που όποια απάντηση και να δώσουμε δεν θα έχει για εμάς πρακτική αξία. «Προβάλλουμε» λοιπόν το ερώτημα αυτό πάνω στο δικό μας πρόβλημα και έχουμε την εξής πιο συγκεκριμένη ερώτηση: Αφού γνωρίζω τις προδιαγραφές ενός συστήματος λογισμικού, γιατί χρειάζεται να το σχεδιάσω, ενώ μπορώ να αρχίσω κατ' ευθείαν να το προγραμματίζω; Έτσι, μια ερώτηση που έμοιαζε αποκομμένη έχει τοποθετηθεί στο συγκεκριμένο πλαίσιο της Τεχνολογίας Λογισμικού, μέσα στο οποίο είναι πιο εύκολο να αναζητήσουμε την απάντησή της.

Κατ' αρχήν, λοιπόν, η σχεδίαση ενός προγράμματος αυξάνει την αποτελεσματικότητα και την απόδοση της συνολικής διαδικασίας ανάπτυξης του συστήματος λογισμικού. Για παράδειγμα, οι περισσότερες από τις σύγχρονες μεθοδολογίες ανάπτυξης λογισμικού χρησιμοποιούν τη

φάση της σχεδίασης για να διασπάσουν το αρχικό πρόβλημα σε μικρότερα προβλήματα, επειδή αυτά λύνονται ευκολότερα και με μικρότερες πιθανότητες λάθους. Η προσέγγιση αυτή οδηγεί στην ανάπτυξη μικρών τμημάτων λογισμικού, τα οποία στη συνέχεια «συνενώνονται» σε ένα σύστημα, το οποίο λύνει το αρχικό πρόβλημα. Επιπλέον, τα τμήματα αυτά μπορεί να προγραμματίζονται παράλληλα από διαφορετικές ομάδες ανάπτυξης, οπότε επιταχύνεται η διαδικασία επίλυσης του προβλήματος.

Επιπλέον, η σχεδίαση του προγράμματος μας βοηθά στο να αποκτήσουμε μια γενική άποψη της λύσης πριν ακόμη την επιχειρήσουμε. Έτσι, γνωρίζουμε από πριν τόσο τα βήματα (δηλαδή τα τμήματα λογισμικού) που θα συνθέσουν τη λύση (δηλαδή το σύστημα λογισμικού), όσο και ποια από αυτά θα είναι δύσκολο ή κρίσιμο να υλοποιηθούν. Έτσι, πριν καταλήξουμε στο πρόγραμμα που θα αναπτύξουμε, έχουμε την «πολυτέλεια» να σχεδιάσουμε και να εξετάσουμε διαφορετικούς τρόπους επίλυσης του προβλήματος.

Ένας άλλος λόγος που καθιστά τη σχεδίαση μια απαραίτητη φάση κατά την ανάπτυξη λογισμικού είναι η ανάγκη διάδοσης και κατανόησης από άλλους της λύσης που υιοθετούμε ή του προγράμματος που θα αναπτύξουμε. Η περιγραφή αυτή γίνεται χρησιμοποιώντας ειδικά εργαλεία και είναι απαραίτητη επειδή το πρόγραμμά μας πρέπει να είναι κατανοητό τόσο από τον υπολογιστή που θα το εκτελέσει, όσο και από τους συναδέλφους μας που εργάζονται στο ίδιο ή σε παρόμοια προβλήματα.

Εντάξει λοιπόν. Ελπίζω ότι έχετε όλοι πειστεί για την αναγκαιότητα σχεδίασης ενός προγράμματος. Εάν όχι, τότε (δυστυχώς για εσάς) θα πειστείτε όταν θα προσπαθήσετε να αναπτύξετε το πρώτο σας πρόγραμμα. Εάν ναι, τότε το επόμενο ερώτημα που ίσως έρχεται στο μυαλό σας είναι: «πώς μπορώ να διασφαλίσω την ποιότητα και την επιτυχία της σχεδίασής μου;»

Λυπάμαι, αλλά η απάντηση είναι «δεν μπορείτε». Θα πρέπει να γνωρίζετε ότι δεν είναι ποτέ δυνατό να διασφαλίσετε εκ των προτέρων ότι θα επιτύχετε την καλύτερη ή την ορθότερη λύση, ούτε μπορείτε να αποδείξετε ότι το πρόγραμμα που πρόκειται να αναπτύξετε θα εκτελείται σωστά σε όλες τις περιπτώσεις. Μπορείτε απλά να αυξήσετε τις πιθανότητες για κάτι τέτοιο ακολουθώντας δοκιμασμένες και διαδεδομένες μεθοδολογίες ή τεχνικές σχεδίασης και ανάπτυξης του λογισμικού

(μια μεθοδολογία συνήθως είναι ένα σύνολο από τεχνικές που αλληλοσυμπληρώνονται).

Μετά από όλα αυτά θα αναρωτιέστε βέβαια: Μα είναι δυνατόν η ανάπτυξη λογισμικού να συνίσταται απλά στην «εφαρμογή των οδηγιών του βιβλίου»; Η απάντηση είναι πάλι «όχι». Στην πραγματικότητα, η σχεδίαση ενός προγράμματος περιλαμβάνει την προσαρμογή των «οδηγιών του βιβλίου» στις ιδιαίτερες απαιτήσεις του προς επίλυση προβλήματος. Είναι συνεπώς μια πολύ δημιουργική διαδικασία, αφού σπανίως δύο προβλήματα είναι ακριβώς τα ίδια. Μερικές φορές δεν υπάρχουν καν οδηγίες για ένα πρόβλημα. Τότε είναι που η σχεδίαση γίνεται μια πραγματικά καινοτόμος δραστηριότητα, γεμάτη προκλήσεις και ικανή να παράσχει μεγάλη ικανοποίηση στον μηχανικό λογισμικού.

Στη συνέχεια του κεφαλαίου περιγράφονται τρία πολύ σημαντικά κριτήρια επιλογής της σχεδιαστικής μεθοδολογίας: το κριτήριο της σχεδιαστικής διαδικασίας (ενότητα 4.1), το κριτήριο της σχεδιαστικής κατεύθυνσης (ενότητα 4.2) και το κριτήριο της μονάδας διάσπασης (ενότητα 4.3). Στην τελευταία ενότητα αναλύονται δύο κριτήρια εκτίμησης της ποιότητας της σχεδίασης: η συνοχή των τμημάτων του λογισμικού και η σύζευξη μεταξύ των τμημάτων του λογισμικού. Αν και η έκταση των θεμάτων αυτών είναι αρκετή για τις ανάγκες του τόμου, θα σας συμβούλευα να ανατρέξετε στο υλικό των Θ.Ε. «Τεχνολογία Λογισμικού Ι» και «Τεχνολογία Λογισμικού ΙΙ» για μια εκτενέστερη παρουσίαση

Δραστηριότητα 4.1

Πριν προχωρήσετε, προσπαθήστε με βάση όσα αναπτύχθηκαν στις εισαγωγικές παρατηρήσεις να απαντήσετε στις εξής ερωτήσεις (χρησιμοποιήστε περίπου 50 λέξεις για κάθε απάντηση):

(α) Γιατί η σχεδίαση ενός προγράμματος είναι απαραίτητη;

(β) Είναι η σχεδίαση πολύπλοκη διαδικασία;

(γ) Είναι η σχεδίαση δημιουργική διαδικασία;

Είναι λοιπόν η σχεδίαση απαραίτητη;

- Η σχεδίαση ενός προγράμματος πριν την υλοποίησή του μας επιτρέπει να επιλέξουμε την καλύτερη και αποτελεσματικότε-

ρη διαδικασία ανάπτυξης, διασφαλίζοντας ταυτόχρονα ότι θα αναπτυχθεί ένα πρόγραμμα που θα λειτουργεί σωστά και με συνέπεια προς τις αρχικές προδιαγραφές. Σε κάθε περίπτωση, χρησιμοποιούμε ένα σύνολο από εργαλεία σχεδίασης για να περιγράψουμε σε μορφή κατανοητή και από άλλους το πρόγραμμα που θα αναπτύξουμε.

Είναι η σχεδίαση πολύπλοκη;

- Μια από τις καλύτερες περιγραφές της σχεδίασης λογισμικού που έχω συναντήσει είναι ως «μια διανοητικά απαιτητική δραστηριότητα, η οποία απαιτεί εμπειρία τόσο στην περιοχή τού προς επίλυση προβλήματος όσο και στο αντικείμενο της ανάπτυξης λογισμικού»

Είναι η σχεδίαση δημιουργική;

- Στο βαθμό που «τέχνη είναι ο συνδυασμός της τεχνικής με την προσωπική σφραγίδα», η σχεδίαση είναι τέχνη. Μπορεί να χρησιμοποιούμε κάποια από τις υπάρχουσες μεθοδολογίες ή να ακολουθούμε ένα σύνολο τεχνικών που μας είναι γνωστές, για να σχεδιάσουμε ένα πρόγραμμα. Πρόκειται όμως για μια αρκετά προσωπική διαδικασία, η οποία συνδυάζει την εμπειρία και την έμπνευση, τις γνώσεις και τις ικανότητες πειραματισμού, αφού κάθε πρόβλημα απαιτεί τη δική του ιδιαίτερη λύση.

Οι έμπειροι μηχανικοί λογισμικού δεν ακολουθούν κάποια γενική σχεδιαστική μεθοδολογία. Αντίθετα, προτιμούν (χρησιμοποιώντας τα κριτήρια που περιγράφονται στη συνέχεια) να συνθέτουν τη δική τους, προσωπική μεθοδολογία από ένα σύνολο τεχνικών που θεωρούν αποδοτικές.

4.1 Το κριτήριο της «σχεδιαστικής διαδικασίας»

Σε όλες τις μεθοδολογίες ανάπτυξης λογισμικού (τις οποίες πρέπει να υιοθετούμε, σύμφωνα με την «αρχή της τυπικότητας» που συζητήσαμε στο κεφάλαιο 2), η σχεδίαση ενός προγράμματος αρχίζει αφού έχουν συλλεχθεί οι απαιτήσεις και έχουν παραχθεί οι προδιαγραφές του. Οι μεθοδολογίες χωρίζονται σε δύο μεγάλες κατηγορίες, ανάλογα με το εάν υιοθετούν μια ακολουθιακή ή μια επαναλαμβανόμενη σχεδιαστική διαδικασία (design process).

Οι **ακολουθιακές μεθοδολογίες** οδηγούν στην ανάπτυξη συστημάτων λογισμικού μέσα από τη διαδοχική εκτέλεση βημάτων που λέγονται «φάσεις». Κάθε φάση μπορεί να εκτελεστεί μια μόνο φορά και αφού τελειώσει η προηγούμενη. Επειδή η εκτέλεση πρέπει να είναι επιτυχημένη, κάθε φάση σχεδιάζεται καλά πριν εκτελεστεί. Αν και οι μεθοδολογίες αυτής της κατηγορίας είναι αρκετά οικονομικές, εν τούτοις σπάνια εφαρμόζονται στην πράξη, αφού οι περιορισμοί της μοναδικής εκτέλεσης κάθε φάσης και της τήρησης της ακολουθίας εκτέλεσης όλων των φάσεων τις καθιστούν ανελαστικές και ευαίσθητες στα σφάλματα.

Αντίθετα, στις **επαναληπτικές μεθοδολογίες** είτε επιτρέπεται η οπισθοδρόμηση για την επανάληψη συγκεκριμένων φάσεων, είτε επιτρέπεται η επανάληψη μιας ολόκληρης ακολουθίας από φάσεις.

Στις επαναληπτικές μεθοδολογίες της πρώτης κατηγορίας η επανάληψη των φάσεων συνήθως δεν είναι προσχεδιασμένη, αλλά επιτρέπεται να συμβεί όταν παραστεί ανάγκη (π.χ., εάν διαγνωσθεί κάποια λανθασμένη λειτουργία στο πρόγραμμα, πρέπει να ανακαλυφθεί εάν η πηγή του λάθους βρίσκεται στις απαιτήσεις, στις προδιαγραφές, στη σχεδίαση ή στην υλοποίηση και ανάλογα να επαναληφθεί η αντίστοιχη φάση). Αυτός είναι ο λόγος που η εφαρμογή τέτοιων μεθοδολογιών μπορεί να γίνει πολύ «ακριβή», εάν οι φάσεις δεν εκτελούνται με προσοχή (συνήθως η απόφαση για την επανάληψη κάποιων φάσεων λαμβάνεται συγκρίνοντας το κόστος της επανάληψης με το κόστος της διάθεσης λογισμικού που έχει λάθη λειτουργίας).

Στις επαναληπτικές μεθοδολογίες της δεύτερης κατηγορίας, κάθε ακολουθία φάσεων παραδίδει μια ολοκληρωμένη και σωστή έκδοση του λογισμικού (συνήθως καλείται «πρωτότυπο») την οποία ο επόμενος

κύκλος ανάπτυξης θα προσπαθήσει να βελτιώσει (συνήθως η απόφαση για την εκτέλεση ενός ακόμη κύκλου ανάπτυξης λαμβάνεται συγκρίνοντας το κόστος του κύκλου με το κόστος διάθεσης της τρέχουσας έκδοσης του λογισμικού).

Μελέτη περίπτωσης

Η κ. Ελένη Νικολοπούλου, μόλις ανέλαβε τη διεύθυνση της βιοτεχνίας παραγωγής παιδικών ρούχων CHILDWARE του πατέρα της. Στα πλαίσια της αναδιοργάνωσης της επιχείρησης, η Ελένη αποφάσισε τη μηχανοργάνωσή της (ξεκινώντας από το τμήμα μισθοδοσίας) και συνεργάζεται με την εταιρεία πληροφορικής Thundersoft. Αυτή ανέθεσε το έργο στον Βύρωνα, έναν νέο μηχανικό λογισμικού, ο οποίος άρχισε αμέσως την ανάλυση του προβλήματος, και συνέλεξε τα ακόλουθα στοιχεία:

Το τμήμα μισθοδοσίας διατηρεί τα πάγια στοιχεία μισθοδοσίας κάθε υπαλλήλου (π.χ., ημερομίσθιο, υπερωρίες, επιπλέον αμοιβές, επιδόματα κ.λπ.) και λαμβάνει από το τμήμα προσωπικού τις «κάρτες εργασίας» των υπαλλήλων, στις οποίες αναγράφονται, για κάθε μήνα, οι ημέρες που ο καθένας τους εργάστηκε. Με βάση αυτά, στο τέλος κάθε μήνα υπολογίζει το μεικτό μισθό κάθε υπαλλήλου, ως το άθροισμα του βασικού μισθού (που είναι το γινόμενο των ημερών εργασίας επί το ημερομίσθιο) συν τυχόν επιπλέον αμοιβές. Η καθαρή αμοιβή υπολογίζεται αφαιρώντας από τη μεικτή αμοιβή τις κρατήσεις. Το τμήμα μισθοδοσίας ενημερώνει το λογιστήριο, στέλνοντας μια κατάσταση μισθοδοσίας και διανέμει τις επιταγές στους υπαλλήλους.

Ο Βύρων έχει ήδη συλλέξει κάποιες αρχικές πληροφορίες για το σύστημα μισθοδοσίας. Όταν όμως προσπαθεί να σχεδιάσει τον αλγόριθμο υπολογισμού της καθαρής αμοιβής, ανακαλύπτει ότι χρειάζεται και άλλα δεδομένα (π.χ., ποιες είναι οι κρατήσεις και πώς υπολογίζονται). Πρέπει λοιπόν να επαναλάβει τη φάση της ανάλυσης, οπότε είναι αναγκασμένος να ακολουθήσει μια επαναληπτική μεθοδολογία της πρώτης κατηγορίας (π.χ., μεθοδολογία καταρράκτη με επαναλήψεις). Η προσέγγιση αυτή είναι αρκετή για τη μηχανοργάνωση του τμήματος μισθοδοσίας. Όμως, αφού η Ελένη σκοπεύει να μηχανοργανώσει πλήρως την επιχείρηση, με το τέλος της ανάπτυξης του συστήματος μισθοδοσίας, ο Βύρων θα πρέπει να εξετάσει πιθανές λύσεις για το πλήρες σύστημα λογισμικού ακολουθώντας μια επανα-

ληπτική μεθοδολογία της δεύτερης κατηγορίας (π.χ., σπειροειδή μεθοδολογία).

4.2 Το κριτήριο της «σχεδιαστικής κατεύθυνσης»

Η επίλυση ενός σύνθετου προβλήματος μέσα από τη διάσπασή του σε μικρότερα και απλούστερα στη λύση τους προβλήματα («αρχή της διάσπασης», κεφάλαιο 3) υλοποιείται από την «από πάνω προς τα κάτω» σχεδίαση (top-down design). Όπως περιγράφεται στη συνέχεια, αυτή δεν είναι η μοναδική σχεδιαστική κατεύθυνση που μπορεί κανείς να ακολουθήσει (Shooman, 1983), αλλά μπορεί εναλλακτικά να επιλέξει είτε τη σχεδίαση «από κάτω προς τα πάνω» (bottom-up design), ή ένα συνδυασμό των δύο που καλείται σχεδίαση «από τη μέση προς την άκρη» (middle-out design).

4.2.1 Σχεδίαση «από πάνω προς τα κάτω»

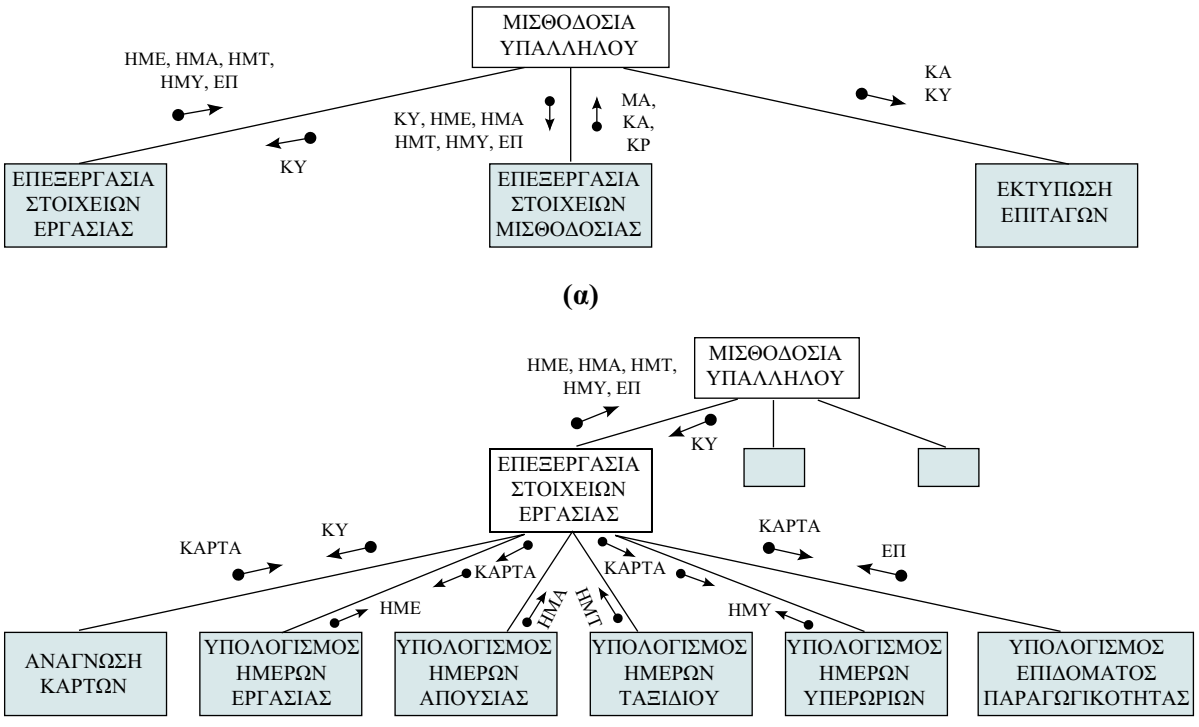
Η **σχεδίαση «από πάνω προς τα κάτω»** είναι η γνωστή διαδικασία διάσπασης, η οποία στα αρχικά στάδια ασχολείται με τη διάσπαση του προβλήματος και συνεχίζει με την τμηματοποιημένη σχεδίαση του συστήματος λογισμικού. Κάθε τμήμα του λογισμικού διασπάται με τη σειρά του σε άλλα μικρότερα και ανεξάρτητα τμήματα. Η διαδικασία επαναλαμβάνεται έως ότου σχεδιαστούν τμήματα που είναι αφ' ενός συνεκτικά και αφ' ετέρου αρκετά μικρά ώστε να είναι κατανοητή η λειτουργία τους.

Καθώς προχωρά η διαδικασία της «από πάνω προς τα κάτω» σχεδίασης, παρατηρούμε ότι σε κάθε επίπεδο σχεδίασης, είναι άγνωστες οι σχεδιαστικές λεπτομέρειες των τμημάτων που βρίσκονται σε ένα κατώτερο επίπεδο, όπως άλλωστε πρεσβεύει και η «αρχή της αφαιρετικότητας», που συναντήσαμε στο κεφάλαιο 3. Αυτό σημαίνει ότι εάν κατά τη σχεδίαση ενός τμήματος του λογισμικού Μ αποφασίσουμε ότι πρέπει να το διασπάσουμε σε μικρότερα τμήματα N_1 έως $N_κ$, εκείνη τη στιγμή δεν γνωρίζουμε (ούτε χρειάζεται να αποφασίσουμε) τον τρόπο που θα σχεδιάσουμε τα N_1 έως $N_κ$. Το μόνο που πρέπει να αποφασιστεί είναι ο τρόπος που το Μ θα συντεθεί από τα N_1 έως $N_κ$, όταν αυτά υλοποιηθούν, καθώς και ο τρόπος που το Μ θα ανταλλάσσει δεδομένα με τα N_1 και $N_κ$.

Μελέτη περίπτωσης (συνέχεια)

Στο Σχήμα 4.1(α) δείχνεται η «από πάνω προς τα κάτω» σχεδίαση του προγράμματος μισθοδοσίας ενός υπαλλήλου που έφτιαξε ο Βύρων (ο συμβολισμός θα εξηγηθεί πιο αναλυτικά στην ενότητα 5.3). Σύμφωνα με αυτή, το πρόγραμμα μισθοδοσίας ενός υπαλλήλου (το λευκό παραλληλόγραμμο) αποτελείται από τρία κύρια υπο-προγράμματα (τα τρία σκιασμένα παραλληλόγραμμα): επεξεργασία στοιχείων εργασίας, επεξεργασία μισθοδοσίας, εκτύπωση επιταγών. Ο Βύρων έχει σχεδιάσει τη δομή του προγράμματος μισθοδοσίας (οι ακμές που συνδέουν τα παραλληλόγραμμα) και τον τρόπο επικοινωνίας με τα υπο-προγράμματα (τα βέλη που φαίνονται δίπλα στις ακμές).

Στο Σχήμα 4.1(β), ένα τμήμα του προγράμματος (επεξεργασία στοιχείων εργασίας) αναλύεται περισσότερο χρησιμοποιώντας την ίδια τεχνική. Στο στάδιο αυτό, ο Βύρων βρίσκεται ακόμη στη φάση της σχεδίασης και δεν ασχολείται με τον τρόπο υλοποίησης των διαφόρων τμημάτων. Απ' ό,τι φαίνεται, όμως, δε χρειάζεται άλλη ανάλυση των τμημάτων του Σχήματος 4.1(β).

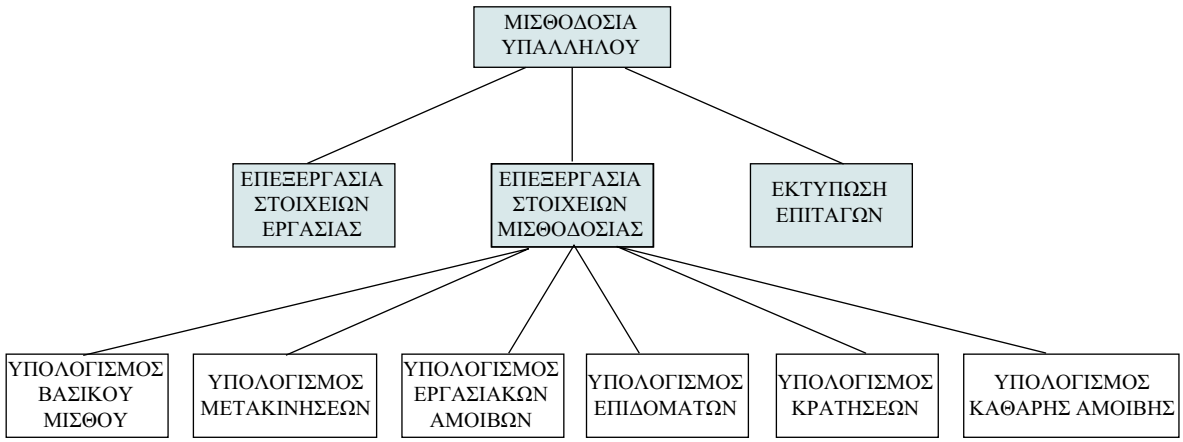


Σχήμα 4.1

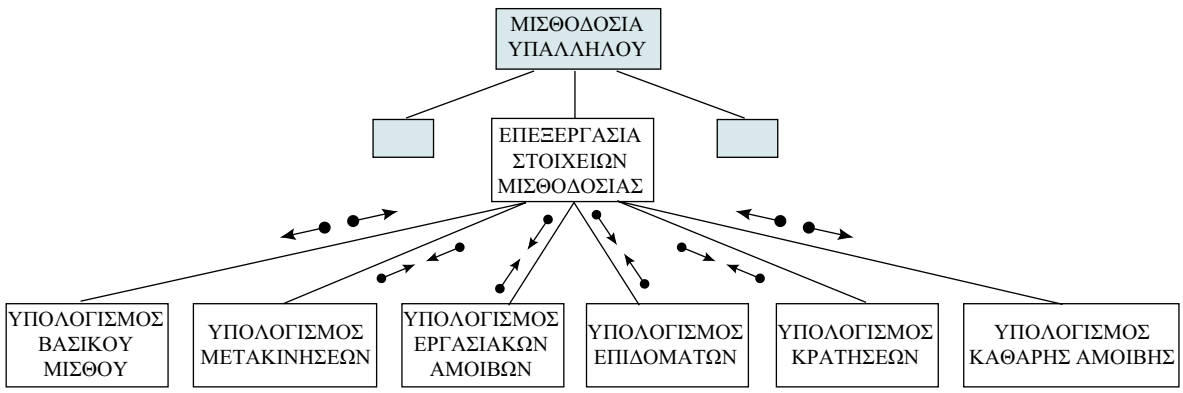
Από πάνω προς τα κάτω σχεδίαση του προγράμματος μισθοδοσίας

4.2.2 Σχεδίαση «από κάτω προς τα πάνω»

Εάν ακολουθήσουμε **σχεδίαση «από κάτω προς τα πάνω»**, τότε, αφού κάνουμε ένα σχέδιο του συστήματος, προχωρούμε στην αναλυτική διερεύνηση και σχεδίαση των τμημάτων εκείνων που είναι καθοριστικά, σημαντικά ή περιοριστικά για τη σχεδίαση του πλήρους συστήματος. Το υπόλοιπο σύστημα σχεδιάζεται έτσι, ώστε να «ταιριάζει» με τον τρόπο που σχεδιάστηκαν αυτά τα τμήματα.



(α)



(β)

Σχήμα 4.2

Από κάτω προς τα πάνω σχεδίαση του προγράμματος μισθοδοσίας

Μελέτη περίπτωσης (συνέχεια)

Στο Σχήμα 4.2(α) δείχνεται η εναλλακτική «από κάτω προς τα πάνω» σχεδίαση του προγράμματος μισθοδοσίας ενός υπαλλήλου που έφτιαξε ο Βύρων. Ξεκίνησε σχεδιάζοντας τα τμήματα υπολογισμού της μεικτής αμοιβής, των επιδομάτων, των κρατήσεων και της καθαρής αμοιβής, από τα οποία συνέθεσε το τμήμα επεξεργασίας της μισθοδοσίας ενός υπαλλήλου [Σχήμα 4.2(β)]. Στην απόφαση αυτή τον ώθησε το γεγονός ότι τα τμήματα αυτά είναι τα κρισιμότερα του προγράμματος και, επιπλέον, είχε αναπτύξει παρόμοια προγράμματα στο παρελθόν.

4.2.3 Προγραμματισμός «από πάνω προς τα κάτω»

Αντίστοιχα με τις μεθοδολογίες σχεδίασης, υπάρχουν και οι μεθοδολογίες του «από πάνω προς τα κάτω» και του «από κάτω προς τα πάνω» προγραμματισμού.

Εάν ακολουθηθεί η πρώτη μεθοδολογία, τότε, καθώς προχωρούμε σχεδιαστικά σε απλούστερα τμήματα λογισμικού, χρειάζεται να προγραμματίσουμε για το Μ τον κώδικα που θα ενοποιεί τα τμήματα αυτά, όταν θα υλοποιηθούν. Προς το παρόν, όμως, κάθε τέτοιο τμήμα θεωρείται ως «κλειστό κουτί» (λέγεται στέλεχος – stub), οπότε στον κώδικα του Μ απλώς υπάρχουν "θέσεις", σε καθεμιά από τις οποίες θα ενσωματωθεί ο κώδικας του αντίστοιχου τμήματος.

Όταν φτάσουμε στο τέλος της διάσπασης, για κάθε απλό τμήμα N_1 έως $N_κ$ που θα προκύψει, αναπτύσσεται ένα απλό πρόγραμμα που το υλοποιεί. Το πρόγραμμα αυτό τοποθετείται στη θέση που του αντιστοιχεί (και αντικαθιστά το υπάρχον στέλεχος) στον κώδικα του Μ, και ελέγχεται εάν επικοινωνεί σωστά με το ήδη υλοποιημένο τμήμα του Μ. Έτσι, για να υλοποιηθεί το πλήρες σύστημα λογισμικού χρειάζεται να ακολουθήσουμε μια διαδικασία σύνθεσης που είναι ακριβώς αντίστροφη από τη διαδικασία διάσπασης που είχαμε αρχικά ακολουθήσει: από απλά προγράμματα συνθέτουμε περισσότερο σύνθετα προγράμματα. Σε μια τέτοια περίπτωση βρίσκουν πλήρη εφαρμογή οι πρακτικές της ΚΒΕ και του αρθρωτού προγραμματισμού που γνωρίσαμε στο κεφάλαιο 3.

Μελέτη περίπτωσης (συνέχεια)

Στη συνέχεια φαίνεται ο τρόπος από πάνω προς τα κάτω προγραμματισμού που ακολούθησε ο Βύρων. Προσέξτε πως το κυρίως πρόγραμμα μισθοδοσίας περιέχει τρία τμήματα, τα οποία θα σχεδιαστούν έτσι ώστε να λύνουν συγκεκριμένα υπο–προβλήματα. Προς το παρόν,

ο Βύρων έχει απλά σημειώσει τη θέση και τον τρόπο επικοινωνίας των τμημάτων, χωρίς να γνωρίζει την εσωτερική τους λειτουργία.

Οι παράμετροι που χρησιμοποιούνται για την επικοινωνία είναι (σημειώνονται ως επιγραφές των βελών στο Σχήμα 4.1):

ΚΥ: Κωδικός Υπαλλήλου
ΗΜΕ: Ημέρες Εργασίας
ΗΜΑ: Ημέρες Απουσίας
ΗΜΤ: Ημέρες Ταξιδιού
ΗΜΥ: Ημέρες Υπερωριών
ΕΠ: Επίδομα παραγωγικότητας
ΜΑ: Μεικτή αμοιβή
ΚΑ: Καθαρή αμοιβή
ΚΡ: Κρατήσεις

ΑΛΓΟΡΙΘΜΟΣ ΜΙΣΘΟΔΟΣΙΑ-ΥΠΑΛΛΗΛΟΥ

ΔΕΔΟΜΕΝΑ
ΚΥ, ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ: INTEGER ;
ΕΠ, ΜΑ, ΚΑ, ΚΡ: REAL ;
ΑΡΧΗ
ΔΙΑΒΑΣΕ(ΚΥ) ;
ΥΠΟΛΟΓΙΣΕ ΣΤΟΙΧΕΙΑ-ΕΡΓΑΣΙΑΣ (ΚΥ, ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ, ΕΠ);
ΥΠΟΛΟΓΙΣΕ ΣΤΟΙΧΕΙΑ-ΜΙΣΘΟΔΟΣΙΑΣ (ΚΥ, ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ, ΕΠ, ΜΑ, ΚΑ, ΚΡ) ;
ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ-ΕΠΙΤΑΓΗΣ (ΚΥ, ΚΑ) ;
ΤΕΛΟΣ

Στη συνέχεια περιγράφεται ένα από τα τμήματα αυτά, το οποίο με τη σειρά του αποτελείται από άλλα τμήματα (ΚΑΡΤΑ είναι το σύνολο των στοιχείων εργασίας):

ΑΛΓΟΡΙΘΜΟΣ ΣΤΟΙΧΕΙΑ-ΕΡΓΑΣΙΑΣ

ΔΙΕΠΑΦΗ
ΕΙΣΟΔΟΣ: ΚΥ ;
ΕΞΟΔΟΣ: ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ, ΕΠ ;
ΔΕΔΟΜΕΝΑ
ΚΑΡΤΑ
ΑΡΧΗ
ΥΠΟΛΟΓΙΣΕ ΕΙΣΟΔΟΣ-ΚΑΡΤΑΣ (ΚΥ, ΚΑΡΤΑ) ;

ΓΙΑ όλες τις ημέρες του μήνα **ΜΕ ΒΗΜΑ** ημέρα **ΕΠΑΝΕΛΑΒΕ**
ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΕΡΓΑΣΙΑΣ(ΚΑΡΤΑ,ΗΜΕ) ;
ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΑΠΟΥΣΙΑΣ(ΚΑΡΤΑ,ΗΜΑ) ;
ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΤΑΞΙΔΙΟΥ(ΚΑΡΤΑ,ΗΜΤ) ;
ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΥΠΕΡΩΡΙΩΝ(ΚΑΡΤΑ,ΗΜΥ) ;
ΥΠΟΛΟΓΙΣΕ ΕΠΙΔΟΜΑΠΑΡΑΓΩΓΙΚΟΤΗΤΑΣ(ΚΑΡΤΑ,ΕΠ)
ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ

Η επικοινωνία μεταξύ των τμημάτων γίνεται με τις παραμέτρους που δηλώνονται στο τμήμα ΔΙΕΠΑΦΗ. Οι δηλώσεις για τα υπόλοιπα τμήματα του κυρίως προγράμματος είναι:

ΥΠΟΛΟΓΙΣΕ ΣΤΟΙΧΕΙΑ-ΜΙΣΘΟΔΟΣΙΑΣ
ΔΙΕΠΑΦΗ
ΕΙΣΟΔΟΣ: ΚΥ ;
ΕΞΟΔΟΣ: ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ, ΕΠ, ΜΑ, ΚΑ, ΚΡ;

ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ-ΕΠΙΤΑΓΗΣ
ΔΙΕΠΑΦΗ
ΕΙΣΟΔΟΣ: ΚΥ, ΚΑ;
ΕΞΟΔΟΣ:

4.2.4 Προγραμματισμός «από κάτω προς τα πάνω»

Όταν ακολουθούμε «από κάτω προς τα πάνω» προγραμματισμό, τότε προγραμματίζουμε κάθε τμήμα λογισμικού αμέσως μόλις το σχεδιάσουμε. Έτσι, τα σημαντικότερα τμήματα του λογισμικού υλοποιούνται και δοκιμάζονται πρώτα, οπότε μπορούμε να έχουμε νωρίς μια εικόνα των δυνατοτήτων και περιορισμών λειτουργίας του συστήματος.

Είναι πιθανό να αναμίξουμε ένα τρόπο σχεδίασης με ένα διαφορετικό τρόπο υλοποίησης. Μια πολύ συχνά ακολουθούμενη μεθοδολογία (που είναι και αρκετά «φυσική» στην εφαρμογή της) είναι ο συνδυασμός «από πάνω προς τα κάτω» σχεδίαση και από «κάτω προς τα πάνω» υλοποίηση. Σε τέτοια περίπτωση, αφού διασπάσουμε το σύστημα λογισμικού σε απλούστερα τμήματα, αρχίζουμε να προγραμματί-

ζουμε τα απλά τμήματα ευθύς μόλις τα σχεδιάσουμε. Μόλις έχουμε υλοποιήσει αρκετά τέτοια μικρά προγράμματα, υλοποιούμε και τον κώδικα που τα συνθέτει σε μεγαλύτερα τμήματα λογισμικού.

4.2.5 Επίπεδα αφαιρετικότητας

Πρόκειται για μια τεχνική που αρχικά χρησιμοποιήθηκε από τον Dijkstra (Dijkstra, 1968) για την περιγραφή ενός λειτουργικού συστήματος. Σύμφωνα με αυτή, ένα σύστημα λογισμικού χτίζεται από διαδοχικά επίπεδα αφαιρετικότητας (levels of abstraction), ξεκινώντας από την αναλυτική περιγραφή του πιο χαμηλού επιπέδου και προχωρώντας προς υψηλότερα και περισσότερο γενικά (αφηρημένα) επίπεδα. Κάθε επίπεδο περιλαμβάνει ένα σύνολο από σχετιζόμενες λειτουργίες (συναρτήσεις) και χρησιμοποιεί αποκλειστικά ένα σύνολο πόρων του συστήματος.

Κάποιες από τις λειτουργίες είναι ορατές εξωτερικά και επιτρέπεται να χρησιμοποιηθούν από άλλες λειτουργίες σε υψηλότερα επίπεδα αφαιρετικότητας, ενώ κάποιες άλλες είναι μόνο εσωτερικές, οπότε είναι κρυμμένες από λειτουργίες υψηλότερων επιπέδων και επιτρέπεται να χρησιμοποιηθούν μόνο από λειτουργίες του ίδιου επιπέδου.

Κάθε επίπεδο αφαιρετικότητας παρέχει ένα σύνολο υπηρεσιών στο αμέσως υψηλότερο επίπεδο αφαιρετικότητας: οι λειτουργίες του υψηλότερου επιπέδου καλούν απευθείας λειτουργίες του αμέσως χαμηλότερου επιπέδου (αλλά το αντίστροφο δεν επιτρέπεται σε καμία περίπτωση).

Επιπλέον, επειδή οι λειτουργίες κάθε επιπέδου έχουν αποκλειστική πρόσβαση σε πόρους του συστήματος, κάθε επίπεδο είναι στο σύνολό του αύταρκες, οπότε μπορεί να χρησιμοποιηθεί αναλλοίωτο σε διαφορετικά συστήματα λογισμικού (όπου φυσικά μπορεί να αλλάζουν οι λειτουργίες του υψηλότερου επιπέδου).

4.2.6 Σχεδίαση «από τη μέση προς την άκρη»

Στην πραγματικότητα, όταν έχουμε να λύσουμε ένα πολύ δύσκολο πρόβλημα, ίσως δεν μπορούμε να σχεδιάσουμε καλά το υψηλότερο επίπεδο του συστήματος, ώστε να εφαρμόσουμε «από πάνω προς τα κάτω» σχεδίαση, αλλά ούτε μπορούμε να ανακαλύψουμε τα κρίσιμα τμήματα του λογισμικού, ώστε να ακολουθήσουμε «από κάτω προς τα πάνω» σχεδίαση. Σε τέτοιες περιπτώσεις καλό είναι να ξεκινήσουμε

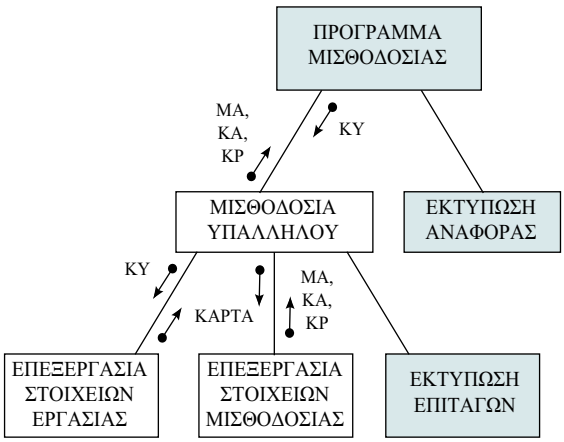
σχεδιάζοντας τη λύση για ένα ή περισσότερα τμήματα του προβλήματος που έχουν ένα μέσο βαθμό πολυπλοκότητας και, το κυριότερο, μας είναι γνωστά. Για τα τμήματα αυτά μπορεί να ακολουθήσουμε κάποια από τις δύο κατευθύνσεις σχεδίασης που αναφέρθηκαν. Έπειτα, προσπαθούμε να σχεδιάσουμε το υπόλοιπο σύστημα ξεκινώντας από τη σχεδίαση και ίσως την υλοποίηση αυτών των τμημάτων. Μια τέτοια προσέγγιση καλείται **σχεδίαση «από τη μέση προς την άκρη»**.

Μελέτη περίπτωσης (συνέχεια)

Ο Βύρων έχει έως τώρα σχεδιάσει το τμήμα εκείνο του προγράμματος που υπολογίζει τη μισθοδοσία ενός υπαλλήλου. Το πλήρες πρόγραμμα της μισθοδοσίας πρέπει να υπολογίζει τη μισθοδοσία όλων των υπαλλήλων και, στο τέλος, να τυπώνει μια συγκεντρωτική αναφορά για το λογιστήριο. Επειδή αυτό ήταν σχετικά πολύπλοκο, ο Βύρων προτίμησε να σχεδιάσει πλήρως τη μισθοδοσία ενός υπαλλήλου, και μετά να σχεδιάσει το πλήρες πρόγραμμα, ακολουθώντας ουσιαστικά από τη μέση προς την άκρη σχεδίαση και προγραμματισμό (Σχήμα 4.3)

ΑΛΓΟΡΙΘΜΟΣ ΜΙΣΘΟΔΟΣΙΑ

ΔΕΔΟΜΕΝΑ
KY: INTEGER
MA, KA, KP: REAL;
ΑΡΧΗ
ΕΝΟΣΩ υπάρχουν υπάλληλοι **ΕΠΑΝΕΛΑΒΕ**
ΥΠΟΛΟΓΙΣΕ Μισθοδοσία-Υπαλλήλου (KY, MA, KA, KP)
ΥΠΟΛΟΓΙΣΕ Εκτύπωση-Αναφοράς
ΕΝΟΣΩ-ΤΕΛΟΣ
ΤΕΛΟΣ



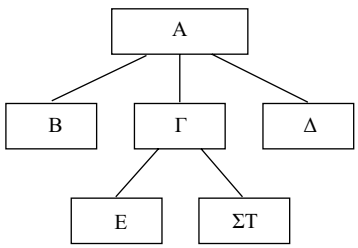
Σχήμα 4.3

Από τη μέση προς την άκρη σχεδίαση και προγραμματισμός

Άσκηση αυτοαξιολόγησης 4.1

Η άσκηση αναφέρεται στο Σχήμα 4.4. Μελετήστε το και προσπαθήστε να αντιστοιχίσετε τεχνικές σχεδιαστικής κατεύθυνσης (βρίσκονται στην αριστερή στήλη) με τη σειρά σχεδίασης των τμημάτων (δείχνεται στη δεξιά στήλη).

Από πάνω προς τα κάτω	Ε - ΣΤ - Γ
	Α - Β - Γ - Δ
Από κάτω προς τα πάνω	Γ- Ε - ΣΤ
	Γ - Ε - ΣΤ - Β - Δ - Α
Από μέσα προς τα έξω	Β - Γ - Δ - Α



Σχήμα 4.4

Δομή ενός προγράμματος για την άσκηση αυτοαξιολόγησης 1

Δραστηριότητα 4.2

Όταν χρειαστεί να αναπτύξετε ένα πρόγραμμα, το δυσκολότερο είναι να αποφασίσετε ποια μεθοδολογία θα ακολουθήσετε. Ένας ενδεικτικός κατάλογος κριτηρίων μπορεί να είναι ο εξής: πόσο νωρίς μπορεί να ελεγχθεί εάν η επιχειρούμενη λύση είναι εφικτή, σε ποιο βαθμό η σχεδίαση εξαρτάται από τις προδιαγραφές του λογισμικού, πόσο μεγάλη ευαισθησία έχει η μεθοδολογία σε λάθη σχεδίασης, ποιο είναι το αποδεκτό μέγεθος τμημάτων, ποια προβλήματα μπορεί να προκύψουν κατά την ολοκλήρωση του συστήματος, πότε πρέπει να σχεδιαστούν τα δεδομένα του συστήματος. Προσπαθήστε να συγκρίνετε τις μεθοδολογίες «από πάνω προς τα κάτω» και «από κάτω προς τα πάνω» προγραμματισμού με βάση τα κριτήρια αυτά, πριν μελετήσετε τον Πίνακα 4.1, στον οποίο γίνεται μια σύνοψη από τις απαντήσεις που δίνονται στη βιβλιογραφία.

Πίνακας 4.1

Σύγκριση των μεθοδολογιών προγραμματισμού «από πάνω προς τα κάτω» και «από κάτω προς τα πάνω»

Κριτήριο	Από πάνω προς τα κάτω	Από κάτω προς τα πάνω
Εφικτή λύση	Εάν άλλοι έχουν επιλύσει παρόμοια προβλήματα, τότε η λύση είναι αποδεδειγμένα εφικτή, και η από πάνω προς τα κάτω σχεδίαση βολεύει.	Εάν διερευνάται η εφικτή λύση, τότε είναι καλύτερα να ξεκινήσουμε από τα κρίσιμα τμήματα.
Εξάρτηση από τις προδιαγραφές	Απαιτεί πολύ καλά ορισμένες προδιαγραφές για όλο το σύστημα από την αρχή.	Λεπτομερείς προδιαγραφές απαιτούνται μόνο για τα κρίσιμα τμήματα.
Ευαισθησία σε σχεδιαστικά λάθη	Εάν γίνουν λάθη στα υψηλά επίπεδα σχεδίασης και δεν ανακαλυφθούν έγκαιρα, μπορεί να οδηγήσουν σε εκ νέου προγραμματισμό μεγάλου μέρους του κώδικα.	Τα αρχικά λάθη οδηγούν σε εκ νέου προγραμματισμό ενός τμήματος μόνο
Μέγεθος τμημάτων	Μικρό (συνήθως περιορίζεται σε μια σελίδα κώδικα). Εάν υπάρχουν μεγαλύτερα τμήματα, τότε διασπώνται.	Εξαρτάται από το τμήμα από το οποίο αρχίζει ο προγραμματισμός.
Ολοκλήρωση	Συνήθως δεν εμφανίζονται απρόσμενα προβλήματα	Εμφανίζονται προβλήματα υλοποίησης της επικοινωνίας μεταξύ των τμημάτων.
Δεδομένα	Η σχεδίαση των δεδομένων που χρησιμοποιούνται σε εσωτερικά τμήματα μπορεί να καθυστερήσει μέχρι την υλοποίηση αυτών	Πρέπει να σχεδιαστούν όλα τα δεδομένα που χρησιμοποιεί το κρίσιμο τμήμα που υλοποιείται πρώτο.

4.3 Το κριτήριο της μονάδας διάσπασης

Όλες οι μεθοδολογίες εξετάζουν ένα σύστημα από δύο όψεις: διεργασίες (processes) και δεδομένα (data). Οι διαφορετικές προσεγγίσεις οφείλονται στη «μονάδα διάσπασης» (unit of decomposition) που υιοθετεί η καθεμία, αν και τελικά όλες πρέπει να θεωρήσουν και τις δύο

όψεις. Ανάλογα με την αρχική προσέγγιση της διαδικασίας σχεδίασης, οι μεθοδολογίες αυτές διακρίνονται σε (Vessey, 1998):

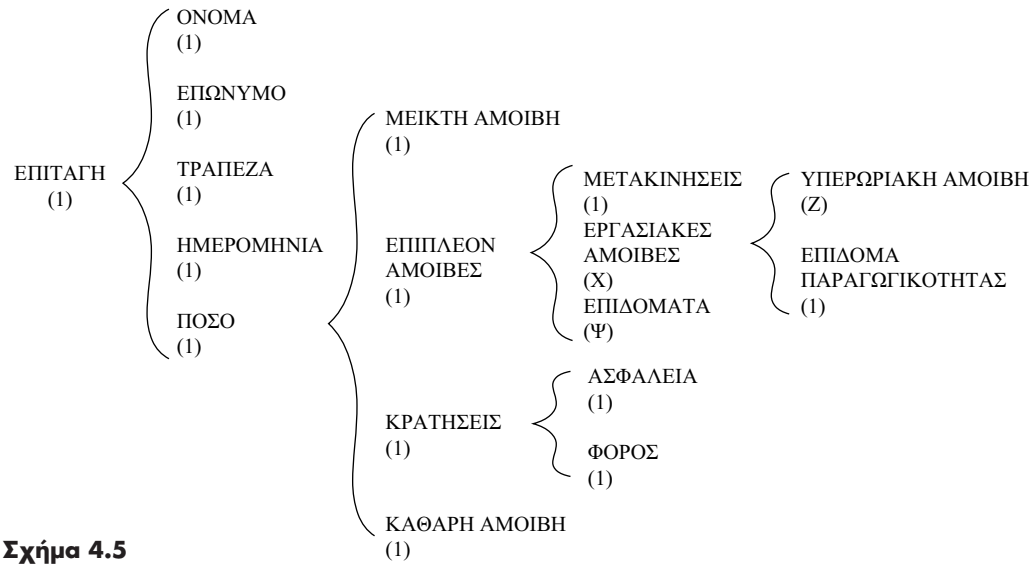
- **Μεθοδολογίες βασισμένες στις διεργασίες:** Πρόκειται για τις γνωστές «μεθοδολογίες δομημένης ανάλυσης και σχεδίασης». Όλες βασίζονται στην διάσπαση των διεργασιών (process decomposition) που αποτελούν το σύστημα, αν και οι περισσότερες από αυτές περιλαμβάνουν και την κανονικοποίηση των δεδομένων του συστήματος. Ο βασικός στόχος είναι να προδιαγραφούν και να σχεδιαστούν οι μικρότερες ανεξάρτητες διεργασίες. Στη συνέχεια, αυτές υλοποιούνται με μικρά προγράμματα, τα οποία συνδυάζονται κατάλληλα σε μεγαλύτερα προγράμματα, έως ότου υλοποιηθεί το πλήρες σύστημα λογισμικού. Πρόκειται για τις πιο παλιές και περισσότερο διαδεδομένες και δοκιμασμένες τεχνικές.
- **Μεθοδολογίες βασισμένες στα δεδομένα:** Βασίζονται στην αρχή ότι τα δεδομένα που χρησιμοποιεί μια επιχείρηση και γενικότερα ένα σύστημα είναι περισσότερο σταθερά από τις διεργασίες που επενεργούν σε αυτά. Αρχικά, λοιπόν, γίνεται διάσπαση των δεδομένων (data decomposition) με στόχο να προδιαγραφούν τα στοιχειώδη δεδομένα που ρέουν στο σύστημα και ο τρόπος παραγωγής από αυτά των σύνθετων δεδομένων. Σε δεύτερο επίπεδο, αναπτύσσονται διεργασίες που μεταχειρίζονται τα δεδομένα, ακολουθώντας τεχνικές δομημένης σχεδίασης και ανάπτυξης. Οι πιο παλιές τεχνικές αυτής της κατηγορίας είναι τα Διαγράμματα Ροής Δεδομένων και κυρίως τα Διαγράμματα Οντοτήτων Συσχετίσεων.
- **Συνδυαστικές μεθοδολογίες:** Συνήθως εξετάζουν με την ίδια βαρύτητα και τις δύο όψεις ενός συστήματος (διεργασίες και δεδομένα). Οι περισσότερο γνωστοί εκπρόσωποι είναι οι αντικειμενοστραφείς μεθοδολογίες (object-oriented methodologies). Σε αυτές, το σύστημα θεωρείται σαν ένα σύνολο αντικειμένων που αλληλεπιδρούν. Ένα αντικείμενο είναι μια συνεκτική συλλογή δεδομένων και των διεργασιών που επιδρούν σε αυτά ακριβώς τα δεδομένα. Η ανάπτυξη του συστήματος γίνεται σχεδιάζοντας και υλοποιώντας τα αντικείμενα και τους τρόπους αλληλεπίδρασής τους. Η διαδικασία περιλαμβάνει πολλούς κύκλους ταυτόχρονης σχεδίασης των δεδομένων που χρησιμοποιεί κάθε αντικείμενο και υλοποίησης των διεργασιών που τα μεταχειρίζονται.

Σε όλες τις μεθοδολογίες, μπορούμε να ξεκινήσουμε την ανάλυση των δεδομένων από δύο σημεία:

- μελετώντας τις εξόδους του συστήματος και σχεδιάζοντας «προς τα πίσω» τις διεργασίες εκείνες που τις παράγουν, μέχρι να φτάσουμε στη σχεδίαση της επεξεργασίας των εισόδων του συστήματος (front-to-back design)
- μελετώντας τις εισόδους του συστήματος και σχεδιάζοντας «προς τα εμπρός» τις διεργασίες που τις επεξεργάζονται ώστε, τελικά, να παραχθούν οι απαιτούμενες έξοδοι (back-to-front design)

Μελέτη περίπτωσης (συνέχεια)

Ο Βύρων λοιπόν, έφτασε σε αδιέξοδο κατά τη σχεδίαση του συστήματος, γιατί δεν έδωσε την πρέπουσα σημασία στα δεδομένα που χειρίζεται το πρόγραμμα. Αναγκάστηκε να μελετήσει τις εξόδους του συστήματος (επιταγές, κατάσταση μισθοδοσίας) ώστε να αναλύσει τα δεδομένα και να συμπεράνει το σημείο που αυτά παράγονται (όπως καταλαβαίνετε, πρόκειται για front-to-back design). Για παράδειγμα, πάνω σε κάθε επιταγή τυπώνεται το ονοματεπώνυμο του υπαλλήλου, το ποσό (που είναι η καθαρή αμοιβή του), η ημερομηνία και η τράπεζα εξόφλησης. Στο Σχήμα 4.5 δείχνεται ο τρόπος παραγωγής των δεδομένων που τυπώνονται πάνω σε μια επιταγή (ο συμβολισμός θα εξηγηθεί στην ενότητα 5.5)



Σχήμα 4.5

Δομή μέρους των δεδομένων του προγράμματος

4.4 Είναι καλό το σχέδιό μου;

Όλες οι αρχές και οι βασικές έννοιες που έχουμε έως τώρα συναντήσει χρησιμεύουν στην τμηματοποιημένη σχεδίαση του λογισμικού. Τα πλεονεκτήματά της είναι η μειωμένη πολυπλοκότητα, η διευκόλυνση της τροποποίησης, η αποφυγή ανεπιθύμητων αποτελεσμάτων και η ευκολότερη και γρηγορότερη υλοποίηση.

Ένα βασικό χαρακτηριστικό που προκύπτει ως συνέπεια της τμηματοποίησης και της υιοθέτησης των αρχών της αφαίρεσης και της απόκρυψης πληροφοριών είναι η **λειτουργική ανεξαρτησία (functional independence)**.

Αυτή επιτυγχάνεται όταν σχεδιάζουμε κάθε τμήμα προγράμματος έτσι, ώστε να αντιμετωπίζει ένα μικρό υποσύνολο των απαιτήσεων (π.χ., υλοποιεί μια μόνο συνάρτηση ή λειτουργία) και να μπορεί να χρησιμοποιηθεί από άλλα τμήματα του προγράμματος με απλό τρόπο. Η λειτουργική ανεξαρτησία μετρείται με δύο ποιοτικά κριτήρια: τη συνοχή και τη σύζευξη.

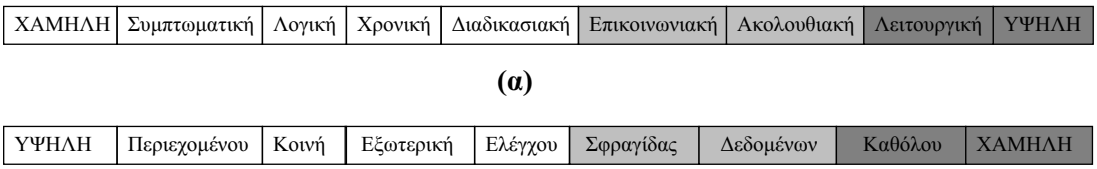
4.4.1 Συνοχή

Η **συνοχή (cohesion)** είναι άμεση συνέπεια της έννοιας της απόκρυψης πληροφοριών: Ένα συνεκτικό τμήμα επιτελεί μια μόνο λειτουργία μέσα σε ένα πρόγραμμα και χρειάζεται πολύ λίγη αλληλεπίδραση (interaction) με άλλα τμήματα του προγράμματος. Οι διάφοροι βαθμοί συνοχής είναι δυνατό να αναπαρασταθούν ως ένα «φάσμα» [Σχήμα 4.6(α)], το οποίο όμως είναι μη-γραμμικό: Ο χαμηλός βαθμός συνοχής είναι πολύ χειρότερος από το μέσο βαθμό, ο οποίος είναι περίπου τόσο καλός όσο και ο υψηλός.

- Πάντα επιδιώκουμε υψηλή συνοχή (στην ιδανική περίπτωση, «κάθε τμήμα κάνει μόνο ένα πράγμα») αλλά και ένας «μέσος» βαθμός συνοχής είναι αποδεκτός. Στην πράξη, δε χρειάζεται να κατηγοριοποιήσουμε ακριβώς το βαθμό συνοχής ενός τμήματος, αλλά να κατανοήσουμε την έννοια και να αποφύγουμε χαμηλά επίπεδα συνοχής.

Οι Constantine και Yourdon (Constantine, 1979) επισημαίνουν επτά επίπεδα συνοχής των μονάδων προγράμματος ενός τμήματος, όπως φαίνεται και στο Σχήμα 4.6(α):

- **Συμπτωματική** (coincidental): Το τμήμα συντίθεται από διάφορες μονάδες, οι οποίες δεν σχετίζονται λειτουργικά, αλλά απλά έχουν τοποθετηθεί στο ίδιο τμήμα.
- **Λογική** (logical): Οι μονάδες του προγράμματος που επιτελούν παρόμοιες λειτουργίες (π.χ., λειτουργίες εισόδου/εξόδου, λειτουργίες διαχείρισης σφαλμάτων κ.ά.) έχουν τοποθετηθεί στο ίδιο τμήμα.
- **Χρονική** (temporal): Οι μονάδες που ενεργοποιούνται την ίδια χρονική στιγμή (π.χ. κατά την εκκίνηση του προγράμματος) έχουν τοποθετηθεί στο ίδιο τμήμα.
- **Διαδικασιακή** (procedural): Οι μονάδες του τμήματος συνθέτουν μια μοναδική ακολουθία εκτέλεσης.
- **Επικοινωνιακή** (communicational): ΄Ολες οι μονάδες του τμήματος χρησιμοποιούν τα ίδια δεδομένα εισόδου, ή παράγουν τα ίδια δεδομένα εξόδου.
- **Ακολουθιακή** (sequential): Η έξοδος μιας μονάδας του τμήματος χρησιμοποιείται ως είσοδος σε μια άλλη μονάδα του τμήματος.
- **Λειτουργική** (functional): Κάθε μονάδα του τμήματος είναι απαραίτητη για την εκτέλεση μιας οποιαδήποτε λειτουργίας.



Σχήμα 4.6

Το φάσμα βαθμών (α) συνοχής και (β) σύζευξης

Οι ίδιοι περιγράφουν έναν απλό τρόπο για να προσδιορίσουμε το βαθμό συνοχής ενός τμήματος: Πρώτα γράφουμε μια φράση, η οποία περιγράφει ακριβώς τη λειτουργικότητα του τμήματος. Κατόπιν, εξετάζουμε τη φράση:

- Εάν η φράση είναι σύνθετη, ή περιλαμβάνει κόμμα, ή περισσότερα από ένα ρήματα, τότε το τμήμα πιθανότατα χαρακτηρίζεται από ακολουθιακή ή επικοινωνιακή συνοχή

- Εάν η πρόταση περιλαμβάνει χρονικούς προσδιορισμούς (π.χ., πρώτο, επόμενο, πριν, μετά κ.ά.), ή λέξεις όπως «αρχικοποίηση», «εκκαθάριση» κ.ά., τότε το τμήμα χαρακτηρίζεται από χρονική συνοχή.
- Εάν η φράση δεν περιλαμβάνει ένα συγκεκριμένο αντικείμενο για το ρήμα, τότε το τμήμα έχει λογική συνοχή (π.χ., διαγραφή όλων των εγγραφών).
- Εάν δεν μπορούμε να αποφύγουμε τη χρήση τέτοιων φράσεων για την πλήρη περιγραφή της λειτουργίας του τμήματος, τότε αυτό δεν είναι λειτουργικά συνεκτικό.

Είναι φανερό λοιπόν ότι η περισσότερο συνεκτική μορφή στην οποία μπορεί να βρεθεί κάποιο τμήμα ενός προγράμματος είναι σαν μια συνάρτηση που υλοποιεί μια μοναδική λειτουργία. Μπορούμε, επίσης, να συμπεράνουμε ότι τα αντικειμενοστραφή συστήματα έχουν από τη φύση τους υψηλό βαθμό συνοχής.

4.4.2 Σύζευξη

Η **σύζευξη (coupling)** αποτελεί μέτρο του βαθμού διασύνδεσης και αλληλεξάρτησης ανάμεσα στα τμήματα του προγράμματος. Εξαρτάται από την πολυπλοκότητα του τρόπου επικοινωνίας των τμημάτων ή κλήσης ενός τμήματος από άλλο, και από τα δεδομένα που ανταλλάσσονται. Όπως και με τη συνοχή, οι βαθμοί σύζευξης μπορούν να αναπαρασταθούν με ένα «φάσμα» [Σχήμα 4.6(β)].

> Κατά τη σχεδίαση και την υλοποίηση προγραμμάτων, επιδιώκουμε τη χαμηλότερη δυνατή σύζευξη, ώστε να αποφύγουμε τη διάδοση σε ολόκληρο το πρόγραμμα των λαθών που ίσως συμβούν σε ένα τμήμα (ripple effect).

Για να εντοπίσουμε το βαθμό σύζευξης δύο τμημάτων, πρέπει να προσδιορίσουμε τον τρόπο που αλληλεπιδρούν (Myers, 1978):

- Εάν το ένα τμήμα περιλαμβάνει εντολές που μεταφέρουν τον έλεγχο μέσα στο άλλο ή τροποποιούν μεταβλητές του άλλου τμήματος, τότε έχουμε σύζευξη περιεχομένου (η χειρότερη μορφή).
- Εάν και τα δύο προσπελαύνουν τα ίδια «σφαιρικά» (global) δεδομένα, τότε έχουμε κοινή σύζευξη. Αυτό δεν σημαίνει ότι δεν πρέπει να χρησιμοποιούνται σφαιρικά δεδομένα, αλλά ότι πρέπει να εξετάσουμε καλά τις πιθανές συνέπειες της χρήσης τους.

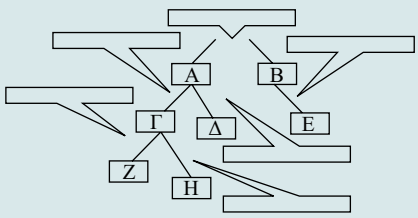
- Εάν υπάρχουν τμήματα που επικοινωνούν με εξωτερικές συσκευές ή προγράμματα, τότε, επειδή αυτά τα τμήματα πρέπει να γραφούν σύμφωνα με τις προδιαγραφές των εξωτερικών συσκευών ή προγραμμάτων, θεωρούμε ότι εξαρτώνται από αυτά και έχουν εξωτερική σύζευξη.
- Εάν το ένα τμήμα στέλνει δεδομένα με βάση τα οποία στο άλλο τμήμα καθορίζεται η ροή εκτέλεσης (π.χ., υπάρχουν εντολές απόφασης στο δεύτερο τμήμα με βάση την τιμή εισερχόμενης μεταβλητής που προέρχεται από το πρώτο τμήμα), τότε τα τμήματα βρίσκονται σε σύζευξη ελέγχου.
- Όταν, αντί μιας απλής μεταβλητής, τα δύο τμήματα ανταλλάσσουν ολόκληρες δομές δεδομένων (π.χ., πίνακες, εγγραφές, λίστες κ.λπ.) ή τμήματα αυτών, τότε βρίσκονται σε σύζευξη σφραγίδας
- Εάν τα δύο τμήματα απλά ανταλλάσσουν δεδομένα, τότε έχουμε σύζευξη δεδομένων, που είναι και ο χαμηλότερος βαθμός σύζευξης (προφανώς, όταν δεν ανταλλάσσουν τίποτα, δεν βρίσκονται καν σε σύζευξη!)

Είναι φανερό λοιπόν, ότι οι περισσότερο επιθυμητοί βαθμοί σύζευξης είναι η σύζευξη δεδομένων και η σύζευξη σφραγίδας, όπου τα τμήματα επικοινωνούν μόνο με την ανταλλαγή παραμέτρων.

Άσκηση αυτοαξιολόγησης 4.2

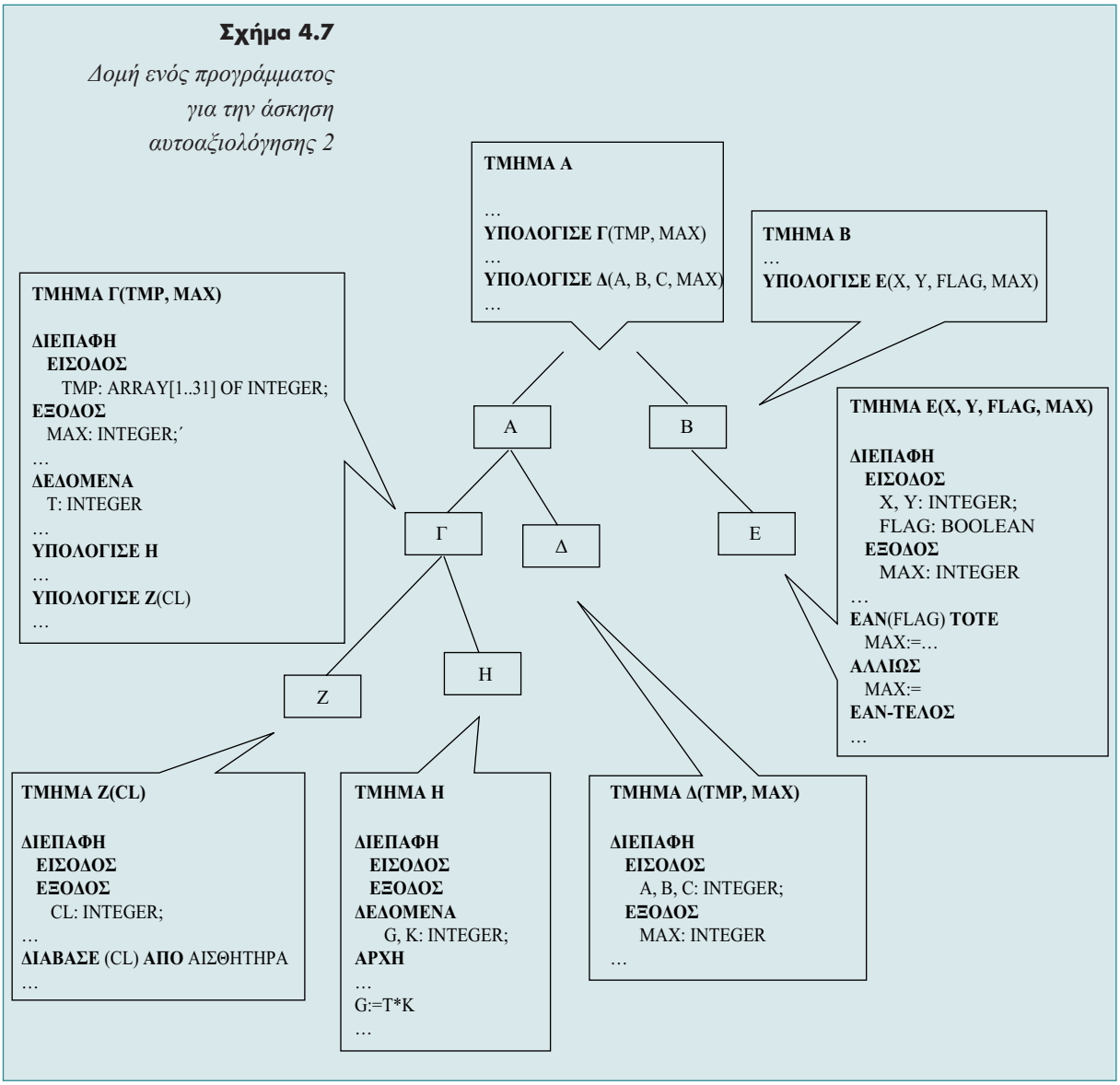
Στο Σχήμα 4.7, απεικονίζεται η δομή και η επικοινωνία μεταξύ επτά τμημάτων λογισμικού. Αφού μελετήσετε το σχήμα, προσπαθήστε να συμπληρώσετε στα έξι κουτάκια του παρακάτω διαγράμματος το είδος της σύζευξης που εμφανίζουν ανά δύο τα τμήματα λογισμικού.

Υπόδειξη: Κάντε την επιλογή σας από τους επτά βαθμούς σύζευξης που προαναφέρθηκαν, δηλαδή: καμμία σύζευξη, σύζευξη δεδομένων, σύζευξη σφραγίδας, σύζευξη ελέγχου, εξωτερική σύζευξη, κοινή σύζευξη, σύζευξη περιεχομένου.



Σχήμα 4.7

Δομή ενός προγράμματος για την άσκηση αυτοαξιολόγησης 2



Σύνοψη

Στο κεφάλαιο αυτό έγινε μια εισαγωγή στις βασικές έννοιες της σχεδίασης ενός προγράμματος. Αφού τεκμηριώθηκε η ανάγκη σχεδίασης πριν την υλοποίηση, παρουσιάστηκαν αναλυτικά τρία κριτήρια επιλογής της μεθοδολογίας σχεδίασης:

- *Το κριτήριο της σχεδιαστικής διαδικασίας, με το οποίο επιλέγουμε ανάμεσα σε μια ακολουθιακή ή επαναλαμβανόμενη μεθοδολογία ανάπτυξης.*
- *Το κριτήριο της σχεδιαστικής κατεύθυνσης, με το οποίο επιλέγουμε τον τρόπο διάσπασης του προβλήματος και σύνθεσης του συνολικού προγράμματος.*
- *Το κριτήριο της μονάδας διάσπασης, με το οποίο καθορίζουμε εάν η έμφαση της σχεδίασης θα δοθεί στα δεδομένα, στις διεργασίες ή τα αντικείμενα που συνθέτουν το πρόγραμμα.*

Τέλος, γνωρίσατε δύο κριτήρια εκτίμησης της ποιότητας της σχεδίασης:

- *Τη συνοχή των τμημάτων του λογισμικού, η οποία περιγράφει το βαθμό αυτάρκειας ενός τμήματος (επιδιώκουμε υψηλό βαθμό συνοχής).*
- *Τη σύζευξη μεταξύ των τμημάτων του λογισμικού, η οποία περιγράφει το βαθμό ανεξαρτησίας κάθε τμήματος (επιδιώκουμε χαμηλό βαθμό σύζευξης).*

Στο επόμενο κεφάλαιο θα γνωρίσετε μερικά ακόμη εργαλεία με τα οποία μπορείτε να περιγράψετε το σχέδιο ενός προγράμματος (ήδη γνωρίζετε τον ψευδοκώδικα και τα Διαγράμματα Ροής Προγράμματος).

Βιβλιογραφία Κεφαλαίου 4

[1] Constantine, L. and Yourdon, E. (1979), *Structured Design*. Prentice–Hall: Englewood Cliffs.

[2] Dijkstra, W. E. (1968), *GOTO Statement Considered Harmful*. Communications of the ACM, 11(3), σελ. 147–148.

[3] Myers, G. (1978), *Composite / Structured Design.* Van Nostrand Reinhold: New York.

[4] Shooman, L. M. (1983), *Software Engineering: design, reliability and management.* McGraw–Hill:Tokyo.

[5] Vessey, L. and Glass, R. (1998), *Strong vs. Weak approaches to systems development*. Communications of the ACM, 41(4), σελ. 99–102.

ΚΕΦΑΛΑΙΟ 5

Άλλα εργαλεία σχεδίασης

Σκοπός

Το κεφάλαιο στοχεύει να σας γνωρίσει μερικά λιγότερο διαδεδομένα, αλλά αρκετά ισχυρά, εργαλεία αναπαράστασης της σχεδίασης ενός προγράμματος. Δίνοντας ξεχωριστή σημασία στη σχεδίαση, θα αντιληφθείτε ότι ο προγραμματισμός δεν είναι κάποια αποκομμένη δραστηριότητα που ασκείται από εκκεντρικά άτομα σε στενά, ακατάστατα γραφεία τις μικρές πρωινές ώρες, αλλά αποτελεί μια φάση των μοντέλων κύκλου ζωής λογισμικού, η οποία, όπως όλες οι άλλες φάσεις, διεκπεραιώνεται μέσα σε καθορισμένα όρια από επαγγελματίες που συνδυάζουν καλά ένα «ρεπερτόριο» τεχνικών με την προσωπική τους εμπειρία.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *αναφέρετε έξι διαφορετικά εργαλεία σχεδίασης προγράμματος*
- *κατασκευάσετε και εξηγήσετε περιγραφές προγράμματος που έχουν κατασκευασθεί χρησιμοποιώντας καθένα από αυτά τα έξι εργαλεία*
- *επιλέγετε και χρησιμοποιείτε κάθε φορά τα κατάλληλα εργαλεία σχεδίασης προγράμματος*

Έννοιες κλειδιά

- *Γλώσσα Σχεδίασης Προγράμματος*
- *Δομοδιαγράμματα*
- *Διαγράμματα Δομής*
- *HIPO*
- *Διαγράμματα Warnier-Orr*
- *Διαγράμματα Jackson*
- *CASE*
- *Πρωτοτυποποίηση*
- *RAD*

Εισαγωγικές παρατηρήσεις

Στο κεφάλαιο 2 μελετήσατε αναλυτικά τις δύο περισσότερο διαδεδομένες τεχνικές αναπαράστασης αλγορίθμων. Μετά τα όσα αναφέρθηκαν στα κεφάλαια 3 και 4, έχετε πλέον καταλάβει ότι στην πραγματικότητα, ο ψευδοκώδικας και τα Διαγράμματα Ροής Προγράμματος (ΔΡΠ) αποτελούν εργαλεία αναπαράστασης της σχεδίασης διαδικασιακών προγραμμάτων και γενικότερα λογισμικού.

Στο κεφάλαιο αυτό θα παρουσιαστούν κι άλλα, ίσως λιγότερο διαδεδομένα, εργαλεία σχεδίασης. Η ύπαρξη πολλών τέτοιων εργαλείων και τεχνικών οφείλεται κυρίως στο ότι κανένα από αυτά δεν αναπαριστά πλήρως ή έστω ικανοποιητικά όλες τις απόψεις (views) ενός συστήματος λογισμικού. Η γνώση και η δυνατότητα χρήσης διαφορετικών τέτοιων εργαλείων θα σας βοηθήσει στην πληρέστερη περιγραφή, την ευκολότερη κατανόηση και τη φθηνότερη συντήρηση του λογισμικού.

Έτσι, ενώ π.χ., τα ΔΡΠ παρέχουν μια πολύ καλή λειτουργική αναπαράσταση του λογισμικού, δε δίνουν αρκετές πληροφορίες για τη δομή του ή για τον τρόπο που αυτό χειρίζεται τα δεδομένα. Αντίθετα, τέτοιες πληροφορίες δίνουν τα Δομοδιαγράμματα (ενότητα 5.2), τα Διαγράμματα Δομής (ενότητα 5.3) και HIPO (ενότητα 5.4), τα Διαγράμματα Warnier–Orr (ενότητα 5.5) και τα Διαγράμματα Jackson (ενότητα 5.6). Πριν από αυτά, στην ενότητα 5.1 παρουσιάζεται η Γλώσσα Σχεδίασης Προγράμματος, ως γενίκευση του ψευδοκώδικα. Περισσότερο αναλυτική παρουσίαση των εργαλείων σχεδίασης, μαζί με παραδείγματα εφαρμογής, θα βρείτε στις Θ.Ε. «Τεχνολογία Λογισμικού Ι» και «Τεχνολογία Λογισμικού ΙΙ».

Το κεφάλαιο κλείνει με μια σύγκριση των εργαλείων σχεδίασης (ενότητα 5.7) και μια σύντομη αναφορά σε σύγχρονα εργαλεία αυτοματοποιημένης σχεδίασης και παραγωγής λογισμικού (ενότητα 5.8).

5.1 Γλώσσα σχεδίασης προγραμμάτων

Πρόκειται για μια γλώσσα με την οποία μπορεί κανείς να περιγράψει οποιοδήποτε διαδικασιακό πρόγραμμα. Η **Γλώσσα Σχεδίασης Προγραμμάτων – ΓΣΠ (Program Design Language – PDL)** προτάθηκε αρχικά από τους Caine, Farber και Gordon (Caine, 1975). Αποτελεί το πιο γενικευμένο εργαλείο λεκτικής περιγραφής προγραμμάτων και σε πρώτη ματιά μοιάζει πολύ με μια υψηλού επιπέδου γλώσσα προγραμματισμού, όπως η Pascal ή η Ada.

Η βασική διαφορά με τέτοιες γλώσσες έγκειται στη δυνατότητα ενσωμάτωσης περιγραφικού κειμένου κατ' ευθείαν μέσα στις εντολές τής ΓΣΠ. Αν και δεν υπάρχουν (ακόμη) μεταγλωττιστές για ΓΣΠ, υπάρχουν όμως προ–επεξεργαστές, οι οποίοι μεταφράζουν περιγραφές με ΓΣΠ σε γραφικά μοντέλα (π.χ., ΔΡΠ) και ταυτόχρονα παράγουν ένα σύνολο πινάκων τεκμηρίωσης.

Στις βασικές συντακτικές δυνατότητες μιας ΓΣΠ περιλαμβάνονται ο ορισμός υποπρογραμμάτων, η περιγραφή διεπαφών μεταξύ προγραμμάτων, η δήλωση δεδομένων, τεχνικές για δόμηση ομάδων οδηγιών, οι βασικές δομές του δομημένου προγραμματισμού (ακολουθία, απόφαση και επανάληψη) και δομές εισόδου/εξόδου.

Πίνακας 5.1

Δομές μιας ΓΣΠ που συμπληρώνουν τον ψευδοκώδικα

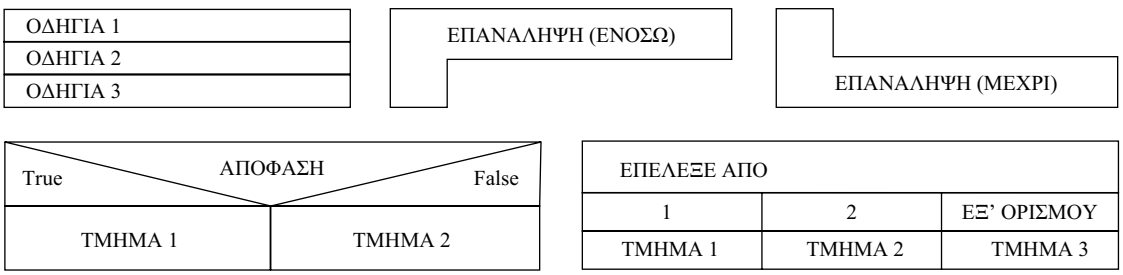
Περιγραφή διεπαφής	ΔΙΕΠΑΦΗ (λίστα μεταβλητών)
	ΕΙΣΟΔΟΣ: μεταβλητές εισόδου
	ΕΞΟΔΟΣ: μεταβλητές εξόδου
Δήλωση τύπου δεδομένων	ΤΥΠΟΣ
	όνομα τύπου ΕΙΝΑΙ περιγραφή
Δήλωση σταθερών	ΣΤΑΘΕΡΕΣ
	όνομα σταθεράς = τιμή
Πολλαπλή επιλογή	ΕΠΕΛΕΞΕ μεταβλητή ΑΠΟ
	ΠΕΡΙΠΤΩΣΗ (τιμή 1) ομάδα οδηγιών
	ΠΕΡΙΠΤΩΣΗ (τιμή 2) ομάδα οδηγιών
	ΕΞ' ΟΡΙΣΜΟΥ ομάδα οδηγιών
	ΕΠΙΛΟΓΗ-ΤΕΛΟΣ

Ο ψευδοκώδικας που χρησιμοποιούμε από την ενότητα 2.3 ήδη περιλαμβάνει τις περισσότερες από αυτές τις δυνατότητες. Στον Πίνακα 5.1 περιγράφονται οι υπόλοιπες.

5.2 Δομοδιαγράμματα

Τα **Δομοδιαγράμματα (box diagrams)** αποτελούν ένα εργαλείο σχεδίασης διαδικασιακών προγραμμάτων που δεν επιτρέπει την παραβίαση των κανόνων του δομημένου προγραμματισμού. Εισήχθησαν από τους Nassi και Shneiderman (Nassi, 1973) και αναπτύχθηκαν περισσότερο από τον Chapin (Chapin, 1974).

Η αναπαράσταση των βασικών δομών προγραμματισμού με Δομοδιαγράμματα φαίνεται στο Σχήμα 5.1. Όπως και ένα ΔΡΠ, ένα Δομοδιάγραμμα μπορεί να εκτείνεται σε πολλές σελίδες, καθώς κάθε τμήμα του λογισμικού περιγράφεται με μεγαλύτερη λεπτομέρεια (εξάλλου, υπάρχει ξεχωριστό σύμβολο για τη χρήση ενός τμήματος λογισμικού από ένα άλλο).



Σχήμα 5.1.

Ο βασικός συμβολισμός των Δομοδιαγραμμάτων

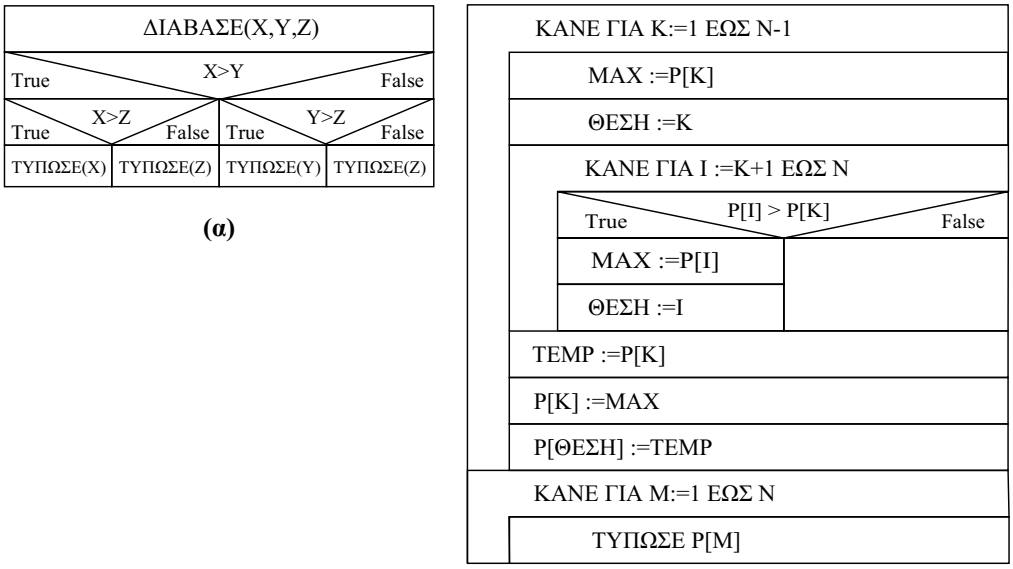
Δραστηριότητα 5.1

Προσπαθήστε να αναπαραστήσετε με Δομοδιάγραμμα τους αλγόριθμους εύρεσης του μεγαλύτερου από τρεις αριθμούς και ταξινόμησης με επιλογή που έχουν περιγραφεί στην ενότητα 2.3

Η ζητούμενη αναπαράσταση φαίνεται στο Σχήμα 5.2(α) για τον αλγόριθμο εύρεσης του μεγαλύτερου από τρεις αριθμούς και στο 5.2(β) για τον αλγόριθμο ταξινόμησης με επιλογή. Παρατηρήστε πόσο συνεκτική και δομημένη είναι η αναπαράσταση των αλγορίθμων.

Τα Δομοδιαγράμματα έχουν τα ακόλουθα χαρακτηριστικά (τα δύο τελευταία θα γίνουν περισσότερο κατανοητά αφού μελετήσετε το κεφάλαιο 7):

- Δεν είναι δυνατή η αυθαίρετη αλλαγή της ροής ελέγχου.
- Η έκταση ενός τμήματος προγράμματος ή μιας προγραμματιστικής δομής είναι εύκολα αναγνωρίσιμη.
- Μπορεί άμεσα να προσδιοριστεί η εμβέλεια των μεταβλητών.
- Η αναπαράσταση της αναδρομής είναι εύκολη.



Σχήμα 5.2

Αναπαράσταση δύο απλών αλγορίθμων με Δομοδιαγράμματα

5.3 Διαγράμματα δομής

Τα **Διαγράμματα Δομής – ΔΔ (Structure Charts)** αποτυπώνουν το αρχιτεκτονικό σχέδιο ενός συστήματος λογισμικού (Stevens, 1974). Κάθε ανεξάρτητο κομμάτι (π.χ., υποπρόγραμμα, διεργασία, τμήμα) του συστήματος αναπαρίσταται με ένα παραλληλόγραμμο. Δύο παραλληλόγραμμα συνδέονται όταν ένα κομμάτι καλεί (χρησιμοποιεί) ένα άλλο. Επιπλέον, η ανταλλαγή δεδομένων (παραμέτρων) ανάμεσα στα δύο τμήματα συμβολίζεται με ένα ειδικό βέλος, η φορά του οποίου δείχνει προς τον αποδέκτη των δεδομένων. Δίπλα σε κάθε βέλος, συνήθως, αναγράφονται οι παράμετροι που ανταλλάσσονται.

Μελέτη περίπτωσης (συνέχεια)

Μελετήστε τα ΔΔ που χρησιμοποίησε ο Βύρων στην ενότητα 4.2 για να περιγράψει το σύστημα του τμήματος μισθοδοσίας. Η λειτουργικότητα και τα δεδομένα που ανταλλάσσουν περιγράφονται με ψευδοκώδικα στην ίδια ενότητα.

Τα ΔΔ δεν περιέχουν σημεία λήψης αποφάσεων, ούτε αναπαριστούν με κάποιον τρόπο τη ροή εκτέλεσης (δηλαδή η παράθεση των τμημάτων δεν υπονοεί κάποια διάταξή τους ως προς το χρόνο). Το μεγάλο τους πλεονέκτημα είναι η άμεση μετάδοση της δομής ενός προγράμματος (εικόνα που είναι ακόμη καθαρότερη όταν δεν αναφέρονται οι παράμετροι που ανταλλάσσονται), η οποία μπορεί να ενισχυθεί με την προσθήκη περιγραφικού κειμένου. Ακόμη, με βάση το ΔΔ ενός προγράμματος, μπορούμε αμέσως να εκτιμήσουμε το βαθμό σύζευξης ανάμεσα στα τμήματά του.

5.4 Διαγράμματα HIPO

Τα **Διαγράμματα HIPO (Hierarchy + Input-Process-Output - δηλ. Ιεραρχία + Είσοδος-Επεξεργασία-Έξοδος)** αναπτύχθηκαν από την IBM (IBM, 1974, IBM, 1975) ως το εργαλείο γραφικής αναπαράστασης της σχεδίασης «από πάνω προς τα κάτω».

Ένα Διάγραμμα HIPO συγκεντρώνει ένα σύνολο τεχνικών αναπαράστασης. Έτσι, για την περιγραφή της δομής του λογισμικού, συνήθως, χρησιμοποιείται ένα ΔΔ, το οποίο αποτελεί και τον «οπτικοποιημένο πίνακα περιεχομένων» του συνόλου. Κάθε τμήμα ή διεργασία λογισμικού που εμφανίζεται στο ΔΔ περιγράφεται σε μια καρτέλα IPO, ενώ κάθε μεταβλητή που ανταλλάσσεται περιγράφεται σε μια καρτέλα στο Λεξικό Δεδομένων (ΛΔ). Τέλος, ένα ξεχωριστό διάγραμμα χρησιμεύει ως υπόμνημα των συμβόλων που χρησιμοποιούνται στο πακέτο.

Σε κάθε καρτέλα IPO εκτός από στοιχεία δημιουργίας και συντήρησης της αντίστοιχης διεργασίας, καταγράφονται τα δεδομένα που αυτή διαβάζει στην είσοδο ή παράγει στην έξοδο, οι τοπικές μεταβλητές που χρησιμοποιεί, και τα τμήματα λογισμικού που καλεί ή την καλούν. Ένα λεπτομερές διάγραμμα IPO περιλαμβάνει και περιγραφή της λειτουργίας του τμήματος λογισμικού με ψευδοκώδικα.

Μελέτη περίπτωσης (συνέχεια)

Στα Σχήματα 4.1 και 4.2 παρουσιάστηκαν τμήματα του ΔΔ του προγράμματος μισθοδοσίας. Στο Σχήμα 5.3 παρουσιάζεται μια από τις ΙΡΟ καρτέλες που συνέταξε ο Βύρων, ενώ στο Σχήμα 5.4 φαίνονται κάποιες καρτέλες από το ΛΔ.

ΔΙΑΓΡΑΜΜΑ ΙΡΟ

ΣΥΣΤΗΜΑ	ΜΙΣΘΟΔΟΣΙΑ	**ΣΧΕΔΙΑΣΤΗΣ**	
ΤΜΗΜΑ	ΣΤΟΙΧΕΙΑ-ΕΡΓΑΣΙΑΣ	**ΗΜΕΡΟΜΗΝΙΑ**	

ΚΑΛΕΙΤΑΙ ΑΠΟ	**ΚΑΛΕΙ**
ΜΙΣΘΟΔΟΣΙΑ-ΥΠΑΛΛΗΛΟΥ	ΗΜΕΡΕΣ-ΕΡΓΑΣΙΑΣ (ΚΑΡΤΑ,ΗΜΕ) ΗΜΕΡΕΣ ΑΠΟΥΣΙΑΣ(ΚΑΡΤΑ,ΗΜΑ) ΗΜΕΡΕΣ-ΤΑΞΙΔΙΟΥ(ΚΑΡΤΑ,ΗΜΤ) ΗΜΕΡΕΣ-ΥΠΕΡΩΡΙΩΝ(ΚΑΡΤΑ,ΗΜΥ) ΕΠΙΔΟΜΑ-ΠΑΡΑΓΩΓΙΚΟΤΗΤΑΣ(ΚΑΡΤΑ,ΕΠ)

ΕΙΣΟΔΟΙ	**ΕΞΟΔΟΙ**
ΚΥ	ΗΜΕ, ΗΜΑ, ΗΜΤ, ΗΜΥ, ΕΠ

ΕΠΕΞΕΡΓΑΣΙΑ

```
ΥΠΟΛΟΓΙΣΕ ΕΙΣΟΔΟΣ-ΚΑΡΤΑΣ (ΚΥ, ΚΑΡΤΑ);
ΓΙΑ όλες τις ημέρες του μήνα ΜΕ ΒΗΜΑ ημέρα ΕΠΑΝΕΛΑΒΕ
        ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΕΡΓΑΣΙΑΣ(ΚΑΡΤΑ,ΗΜΕ);
        ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΑΠΟΥΣΙΑΣ(ΚΑΡΤΑ,ΗΜΑ);
        ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΤΑΞΙΔΙΟΥ(ΚΑΡΤΑ,ΗΜΤ);
        ΥΠΟΛΟΓΙΣΕ ΗΜΕΡΕΣ-ΥΠΕΡΩΡΙΩΝ(ΚΑΡΤΑ,ΗΜΥ);
        ΥΠΟΛΟΓΙΣΕ ΕΠΙΔΟΜΑ-ΠΑΡΑΓΩΓΙΚΟΤΗΤΑΣ(ΚΑΡΤΑ,ΕΠ)
ΓΙΑ-ΤΕΛΟΣ
```

ΤΟΠΙΚΑ ΔΕΔΟΜΕΝΑ	**ΠΑΡΑΤΗΡΗΣΕΙΣ**
ΚΑΡΤΑ	

Σχήμα 5.3

ΙΡΟ καρτέλα για το τμήμα ΣΤΟΙΧΕΙΑ-ΕΡΓΑΣΙΑΣ

ΟΝΟΜΑ	ΚΑ
ΣΥΝΩΝΥΜΟ	Καθαρή αμοιβή
ΠΕΡΙΓΡΑΦΗ	ΕΙΝΑΙ «μεικτή αμοιβή» - «κρατήσεις»
ΤΥΠΟΣ	Δεκαδικός Αριθμός, 2 δεκαδικά ψηφία
ΠΕΔΙΟ ΤΙΜΩΝ	>= 0
ΠΑΡΑΓΕΤΑΙ	ΤΜΗΜΑ ΣΤΟΙΧΕΙΑ-ΜΙΣΘΟΔΟΣΙΑΣ
ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ	ΤΜΗΜΑ ΜΙΣΘΟΔΟΣΙΑ-ΥΠΑΛΛΗΛΟΥ ΤΜΗΜΑ ΕΚΤΥΠΩΣΗ-ΕΠΙΤΑΓΗΣ

ΟΝΟΜΑ	ΚΡ
ΣΥΝΩΝΥΜΟ	Κρατήσεις
ΠΕΡΙΓΡΑΦΗ	ΕΙΝΑΙ "ασφ. ταμείο" + "χαρτόσημο" + "ΟΓΑ χαρτοσήμου" + "παρακράτηση φόρου"
ΤΥΠΟΣ	Δεκαδικός Αριθμός, 2 δεκαδικά ψηφία
ΠΕΔΙΟ ΤΙΜΩΝ	>= 0
ΠΑΡΑΓΕΤΑΙ	ΤΜΗΜΑ ΣΤΟΙΧΕΙΑ-ΜΙΣΘΟΔΟΣΙΑΣ
ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ	ΤΜΗΜΑ ΜΙΣΘΟΔΟΣΙΑ-ΥΠΑΛΛΗΛΟΥ

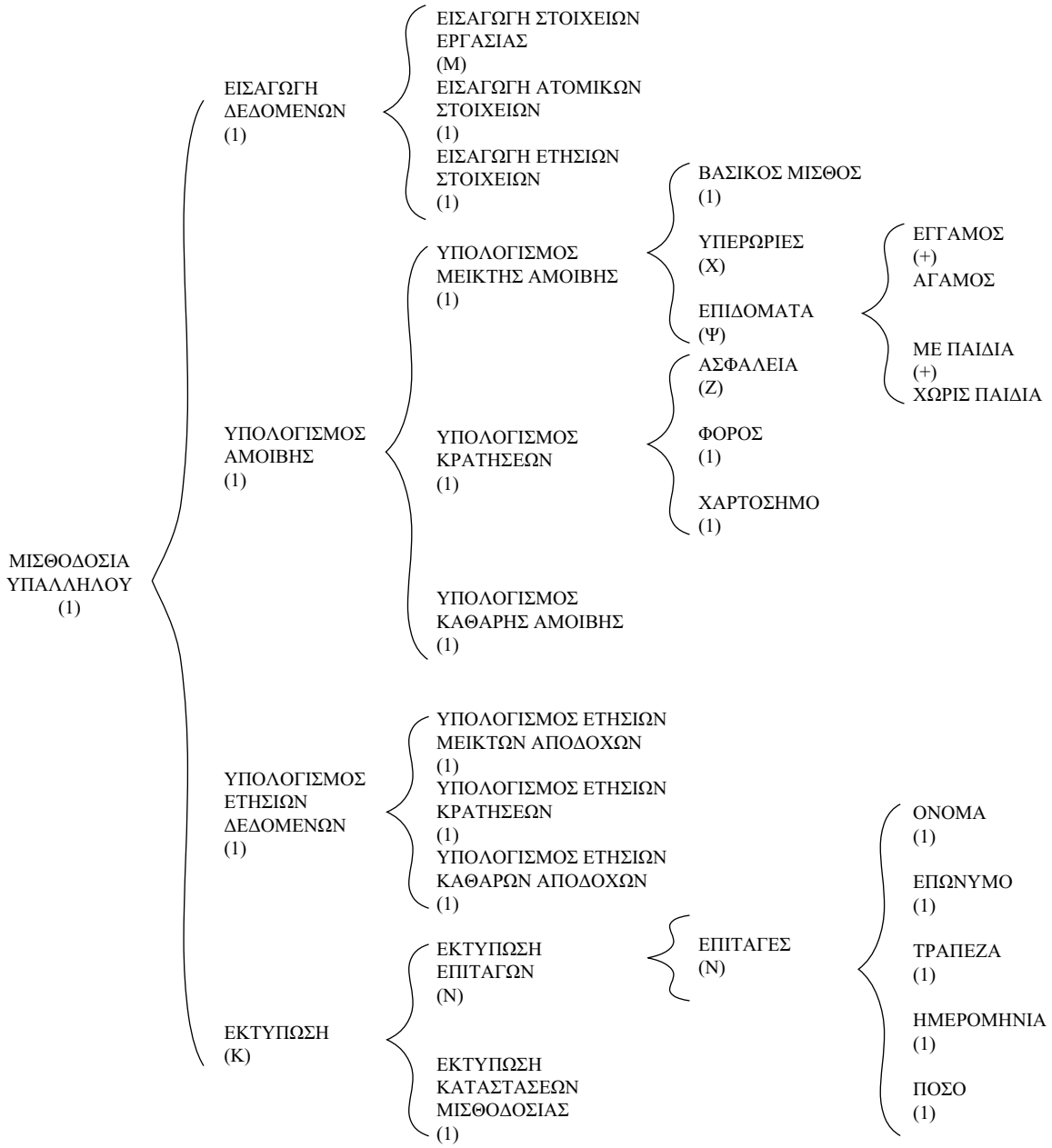
Σχήμα 5.4

Απόσπασμα από το ΛΔ του προγράμματος μισθοδοσίας

5.5 Διαγράμματα Warnier–Orr

Τα διαγράμματα αυτά είχαν αρχικά αναπτυχθεί για την ιεραρχική αναπαράσταση των δεδομένων (Warnier, 1974) και επεκτάθηκαν ώστε να είναι κατάλληλα για τη σχεδίαση συστημάτων (Orr, 1981). Έτσι, ένα **διάγραμμα Warnier–Orr (ΔΓΟ)** μπορεί να χρησιμοποιηθεί για την περιγραφή των δομών δεδομένων, της λογικής ροής του προγράμματος ή ακόμη και της συνολικής δομής του λογισμικού.

Η βασική ιδέα πίσω από τα ΔΓΟ είναι ότι ένα καλογραμμένο σύστημα λογισμικού είναι στενά συνδεδεμένο με τα δεδομένα που χρησιμοποιεί. Συνεπώς, η σχεδίαση του συστήματος μπορεί να αρχίσει σχεδιάζοντας τα δεδομένα.



Σχήμα 5.5

Περιγραφή του προγράμματος μισθοδοσίας με ΔΓΟ

Ο συμβολισμός που χρησιμοποιείται είναι σχετικά απλός: το σύμβολο { δηλώνει την ιεραρχία, η κατακόρυφη σειρά παράθεσης των τμημάτων δηλώνει τη σειρά εκτέλεσης ή χρήσης, και ένας αριθμός (1..N) σε παρένθεση κάτω από κάθε οντότητα δηλώνει τον αριθμό των επαναλήψεων χρήσης της οντότητας. Το σύμβολο ≈ δηλώνει την αποκλειστική διάζευξη ανάμεσα σε δύο οντότητες.

Μελέτη περίπτωσης (συνέχεια)

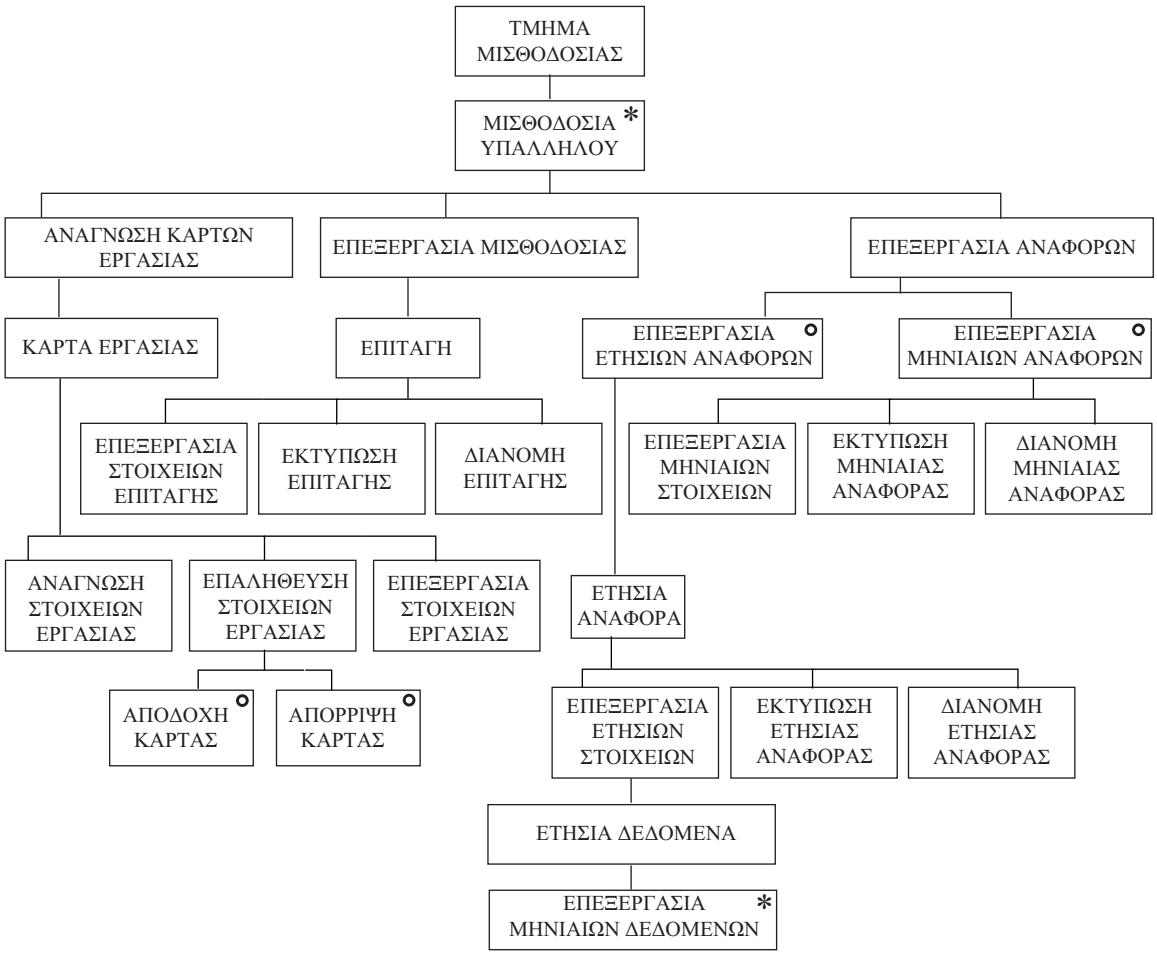
Στο Σχήμα 5.5, ο Βύρων περιγράφει πλήρως το λογισμικό υπολογισμού της μισθοδοσίας ενός υπαλλήλου χρησιμοποιώντας ΔΓΟ. Παρατηρήστε πως το συνολικό πρόγραμμα περιλαμβάνει τα τμήματα Εισαγωγής Δεδομένων, Υπολογισμού Αμοιβής, Υπολογισμού Ετήσιων Δεδομένων και Εκτύπωσης. Καθένα από αυτά αναλύεται με τη σειρά του σε πιο απλά τμήματα, ενώ στο επόμενο επίπεδο, περιγράφονται τα δεδομένα που κάθε τμήμα χρησιμοποιεί (π.χ., για τον υπολογισμό των κρατήσεων χρησιμοποιούνται τα ποσά της Ασφάλισης, του Φόρου και του Χαρτοσήμου).

5.6 Διάγραμμα Jackson

Το **διάγραμμα Jackson (ΔΤ)** επινοήθηκε ως το βασικό εργαλείο αναπαράστασης της μεθοδολογίας ανάπτυξης συστημάτων λογισμικού που προτείνει ο M.A. Jackson (Jackson, 1975). Το ΔΤ δεν περιγράφει μόνο τα τμήματα που συνθέτουν ένα σύστημα λογισμικού, αλλά και τη συσχέτιση μεταξύ τους (π.χ., κλήση ενός τμήματος από άλλο, ανταλλαγή μηνυμάτων, ενεργοποιήσεις κ.λπ.) με χρονικά ταξινομημένο τρόπο, ενώ μπορεί να συνοδεύεται και από περιγραφικό κείμενο.

Μελέτη περίπτωσης (συνέχεια)

Στο Σχήμα 5.6 φαίνεται το ΔΤ που κατασκεύασε ο Βύρων για το τμήμα μισθοδοσίας. Παρατηρήστε ότι το σύστημα λογισμικού του τμήματος μισθοδοσίας εκτελεί επαναληπτικά για κάθε υπάλληλο το πρόγραμμα υπολογισμού της μισθοδοσίας υπαλλήλου, το οποίο αποτελείται από τα τμήματα Ανάγνωσης Καρτών Εργασίας, Επεξεργασίας της Μισθοδοσίας και Επεξεργασίας των Αναφορών. Το τελευταίο εκτελείται είτε για την Επεξεργασία Μηνιαίων Αναφορών, είτε για την Επεξεργασία Ετήσιων Αναφορών. Κάθε ορθογώνιο αναπαριστά ένα τμήμα του προγράμματος και κάθε σύνδεση δηλώνει κλήση του ενός προγράμματος από κάποιο άλλο. Το σύμβολο «*» δηλώνει επαναλαμβανόμενη κλήση, ενώ το «ο» δηλώνει αμοιβαίο αποκλεισμό μεταξύ υποπρογραμμάτων.



Σχήμα 5.6

Περιγραφή του προγράμματος μισθοδοσίας με ΔΤ

5.7 Ποιό εργαλείο να επιλέξω;

Πολύ καλή ερώτηση. Όπως καταλαβαίνετε, βέβαια, δεν υπάρχει μια μόνο σωστή απάντηση. Ως μηχανικοί λογισμικού ή προγραμματιστές, πρέπει να φροντίζετε να δίνετε μια ολοκληρωμένη και σωστή περιγραφή του λογισμικού που αναπτύσσετε. Αυτό υπονοεί ότι μπορείτε να επιλέξετε την κατάλληλη κάθε φορά τεχνική και ότι γνωρίζετε να χειρίζεστε τις τεχνικές που χρησιμοποιείτε. Βέβαια, πάντα θα χρειάζεται να χρησιμοποιήσετε περισσότερες από μία τεχνικές!

Δραστηριότητα 5.2

Πριν προχωρήσετε, συνοψίστε σε ένα πίνακα τα χαρακτηριστικά κάθε εργαλείου που παρουσιάστηκε στο κεφάλαιο 5, μαζί με τον ψευδοκώδικα και τα ΔΡΠ. Έπειτα, συγκρίνετε την απάντησή σας με το ακόλουθο (Πίνακας 5.2) απόσπασμα από τον «Οδηγό Επιβίωσης του Σχεδιαστή Λογισμικού» που ο Βύρων έχει πάντα μαζί του.

Πίνακας 5.2

Σύγκριση των εργαλείων σχεδίασης προγραμμάτων

ΑΠΟΦΥΓΕΤΕ ΤΟ ΣΚΟΠΕΛΟ ΤΗΣ ΚΑΚΗΣ ΣΥΝΕΝΝΟΗΣΗΣ ΟΔΗΓΟΣ ΚΑΤΑΛΛΗΛΟΤΗΤΑΣ ΕΡΓΑΛΕΙΩΝ ΣΧΕΔΙΑΣΗΣ
Γλώσσα Σχεδίασης Προγράμματος Πρόκειται για συνδυασμό εντολών γλώσσας προγραμματισμού με περιγραφές σε φυσική δομημένη γλώσσα, οπότε είναι εύκολο να κατανοηθεί τόσο από προγραμματιστές, όσο και από τελικούς χρήστες. Αποτελεί επέκταση του ψευδοκώδικα στο ότι προσεγγίζει περισσότερο κάποια συγκεκριμένη γλώσσα προγραμματισμού, αλλά αυτό περιορίζει και την ανεξαρτησία της περιγραφής.
Δομοδιαγράμματα Παρέχουν μια συμπιεσμένη αλλά ξεκάθαρη αναπαράσταση των αλγορίθμων. Υποστηρίζουν τη σύνθεση δομημένων προγραμμάτων μόνο από τις επιτρεπόμενες δομές προγραμματισμού. Δεν παρέχουν (σε αναλογία με τον ψευδοκώδικα ή τη ΓΣΠ) επισκόπηση της δομής του προγράμματος.
HIPO και Διάγραμμα Δομής Επειδή περιλαμβάνει ένα σύνολο διαφορετικών αναπαραστάσεων, μπορεί να χρησιμοποιηθεί από όλους όσους εμπλέκονται στην ανάπτυξη του προγράμματος (χρήστες, σχεδιαστές, προγραμματιστές κ.α.). Συνδυάζει γραφική απεικόνιση (η οποία παρέχει μια επισκόπηση της δομής του προγράμματος) με αναλυτική περιγραφή τόσο των διεργασιών (της δομής, της επικοινωνίας και της λειτουργίας τους), αλλά και των δεδομένων. Επιτρέπει τη γρήγορη διαπίστωση του βαθμού συνοχής και σύζευξης του προγράμματος. Δεν αναπαριστά άμεσα τη ροή εκτέλεσης. Οδηγεί σε μεγάλου όγκου σχέδια, γι' αυτό και συνήθως δεν χρησιμοποιείται πέρα από το σημείο της χονδρικής σχεδίασης.
Διάγραμμα Jackson Σχεδόν μηχανική εφαρμογή για τη σχεδίαση προγράμματος που μετατρέπει εισόδους σε εξόδους. Ενδείκνυται για ανάπτυξη προγραμμάτων ξεκινώντας από τις δομές δεδομένων. Παρέχει άμεση επισκόπηση της δομής του προγράμματος, αλλά δεν αναπαριστά την επικοινωνία των τμημάτων. Απαιτεί πολλούς σχεδιαστικούς κύκλους.

Διάγραμμα Warnier–Orr

Είναι ευέλικτο εργαλείο, που μπορεί να περιγράψει λεπτομερώς τις δομές δεδομένων, τη λογική του προγράμματος, ακόμη και τη δομή του προγράμματος. Υιοθετεί τη «προσανατολισμένη-στα-δεδομένα» σχεδίαση. Έχει απλό συμβολισμό, οδηγεί σε «συμπιεσμένη» αναπαράσταση (περίπου κατά 25%) του προγράμματος και είναι κατανοητό και από «μη-ειδικούς». Δεν αναπαριστά τη ροή εκτέλεσης, την επικοινωνία των τμημάτων και το βαθμό τμηματοποίησης του προγράμματος.

Δραστηριότητα 5.3

Αφού έχετε μελετήσει προσεκτικά τη μελέτη περίπτωσης που παρατίθεται σε αυτό το κεφάλαιο, και τα εργαλεία σχεδίασης προγράμματος που περιγράφονται σε αυτό το κεφάλαιο και το κεφάλαιο 2, προσπαθήστε να διαμορφώσετε τη δική σας πρόταση σχετικά με τη χρήση των εργαλείων αυτών στις διάφορες δραστηριότητες σχεδίασης και ανάπτυξης ενός προγράμματος. Η δική μας πρόταση παρατίθεται στη συνέχεια.

Μια συνδυαστική, λοιπόν, πρόταση θα μπορούσε να περιλαμβάνει τα εξής:

- Τη χρήση HIPO διαγραμμάτων για την αναπαράσταση της δομής του προγράμματος,
- τη χρήση ΔΔ για την περιγραφή της επικοινωνίας μεταξύ των τμημάτων του προγράμματος, ανεξάρτητα από το εάν χρησιμοποιείται σχεδιάση από πάνω προς τα κάτω ή από κάτω προς τα πάνω,
- τη χρήση ΔΡΠ ή Δομοδιαγράμματος για την περιγραφή σε υψηλό επίπεδο της λειτουργίας κάθε προγράμματος,
- αφού εκτελεστούν μερικά βήματα της ΚΒΕ, τη χρήση ψευδοκώδικα ή ΓΣΠ για την λεπτομερή (ακριβή) περιγραφή του τρόπου λειτουργίας του προγράμματος, ανάλογα πάντα με τη γλώσσα προγραμματισμού που έχει επιλεγεί για την τελική υλοποίηση,
- ανάλογα με την περίπτωση, τη χρήση ΔΓΟ ή ΔΤ για την περιγραφή της δομής των δεδομένων και της εξάρτησης του προγράμματος από αυτά.

Βέβαια, η ταυτόχρονη χρήση πολλών τεχνικών παρέχει πλήρη περιγραφή και τεκμηρίωση του προγράμματος, αλλά κάνει δύσκολη την

όποια τροποποίηση της σχεδίασης, αφού θα πρέπει να ενημερωθούν όλες οι περιγραφές. Σε τελική ανάλυση λοιπόν, πρέπει να επιλέγετε τις τεχνικές που θα χρησιμοποιήσετε ανάλογα με τις απαιτήσεις του τελικού χρήστη, την πολυπλοκότητα του προγράμματος και την αυτοπεποίθηση που έχετε σχετικά με την κατανόηση του προβλήματος.

5.8 Αυματοποίηση των διαδικασιών προγραμματισμού

Η ιδέα είναι παλιά: ενίσχυση των ανθρώπινων δυνατοτήτων με τη χρήση εργαλείων. Η ελπίδα είναι ισχυρή: βελτίωση της παραγωγικότητας των προγραμματιστών και της ποιότητας των προγραμμάτων με τη χρήση αυτοματοποιημένων εργαλείων σχεδίασης και ανάπτυξης λογισμικού.

5.8.1 Case

Μεγάλη έμφαση στην ανάπτυξη τέτοιων εργαλείων δόθηκε στη δεκαετία του 1980, οπότε εμφανίστηκε και ο όρος **Τεχνολογία Λογισμικού Υποστηριζόμενη από Υπολογιστή (Computer–Aided Software Engineering – CASE).** Τα διάφορα εργαλεία CASE κατηγοριοποιούνται ανάλογα με το βαθμό υποστήριξης που παρέχουν σε:

- **εργαλεία (tools),** τα οποία είναι συνήθως μικρά, ανεξάρτητα προγράμματα που αυτοματοποιούν μια δραστηριότητα του κύκλου λογισμικού,
- **περιβάλλοντα εργασίας (workbenches),** τα οποία συνήθως περιλαμβάνουν ένα σύνολο εργαλείων για να υποστηρίξουν με εννιαίο τρόπο μία ή περισσότερες φάσεις του κύκλου λογισμικού,
- **ολοκληρωμένα περιβάλλοντα (integrated environments),** τα οποία υποστηρίζουν μέσα από ένα ενιαίο περιβάλλον πρόσβασης, ολόκληρο τον κύκλο ανάπτυξης λογισμικού.

Τα ολοκληρωμένα εργαλεία CASE επιτρέπουν π.χ., τη σχεδίαση του λογισμικού με ένα γραφικό ή λεκτικό μοντέλο (στο οποίο εφαρμόζουν συντακτικό έλεγχο), την αυτοματοποιημένη παραγωγή κώδικα σε κάποια γλώσσα προγραμματισμού, την παραγωγή συνοδευτικής τεκμηρίωσης, τον έλεγχο και τη δοκιμή του κώδικα, την ολοκλήρωση και εγκατάσταση του λογισμικού, ακόμη και τη διαχείριση της διαδικασίας. Στα μειονεκτήματά τους συγκαταλέγονται η παραγωγή όχι βέλ-

τιστου (αλλά όμως ορθού) κώδικα, το κόστος αγοράς και η ανάγκη εκπαίδευσης των στελεχών που θα το χρησιμοποιούν.

5.8.2 Εργαλεία προγραμματισμού

Το βασικότερο εργαλείο αυτόματης παραγωγής προγραμμάτων είναι, βεβαίως, η γλώσσα προγραμματισμού. Τελικά, όλα τα σχέδια και οι αλγόριθμοι του λογισμικού καταλήγουν να μεταφραστούν σε εντολές (source program code) σε κάποια γλώσσα προγραμματισμού. Έπειτα, ο μεταγλωττιστής (compiler) της γλώσσας αναλαμβάνει την παραγωγή του κώδικα του προγράμματος σε μορφή άμεσα κατανοητή (και συνεπώς εκτελέσιμη) από τον υπολογιστή (object program code).

Οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν όλα τα παραδείγματα και τις αρχές προγραμματισμού που έχουν ως τώρα παρουσιαστεί σε αυτόν τον τόμο. Τα εργαλεία με τα οποία γράφουμε τον κώδικα κάνουν συνήθως αυτόματο λεκτικό και συντακτικό έλεγχο, ενώ οι μεταγλωττιστές κάνουν σημασιολογικό έλεγχο. Επιπλέον, μας παρέχονται δυνατότητες δοκιμαστικής εκτέλεσης του προγράμματος (πρωτοτυποποίηση) με στόχο την ανεύρεση σφαλμάτων, εξομοίωσης της εκτέλεσης με στόχο τη βελτιστοποίηση της λειτουργίας, δημιουργίας επαναχρησιμοποιήσιμων τμημάτων λογισμικού κ.ά.

Οι τελευταίες εξελίξεις στο χώρο οδήγησαν στην ανάπτυξη εργαλείων που υποστηρίζουν οπτικοποιημένο (ή ενορατικό) προγραμματισμό (visual programming) και ταχεία ανάπτυξη εφαρμογών (Rapid Application Development – RAD). Η ανάπτυξη προγραμμάτων με τέτοια εργαλεία συνήθως ξεκινά από τη διεπαφή προγράμματος–χρήστη (user interface) και τις δυνατότητες που αυτή θα παρέχει στους χρήστες της εφαρμογής. Στη συνέχεια, αναπτύσσονται τμήματα προγράμματος τα οποία χειρίζονται τις διάφορες εντολές που δίνουν οι χρήστες ή τα γεγονότα που δημιουργούνται στο πρόγραμμα, καθώς και το «διάλογο» του προγράμματος με τους χρήστες (δηλαδή την παρουσίαση των δυνατοτήτων του, τον τρόπο ανάγνωσης και εκτύπωσης των δεδομένων κ.λπ.).

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν έξι ακόμη εργαλεία αναπαράστασης της σχεδίασης ενός προγράμματος. Έτσι, ολοκληρώνεται το τμήμα του τόμου που αναφέρεται σε τεχνικές σχεδίασης και εργαλεία αναπαράστασης της σχεδίασης ενός προγράμματος. Πριν προχωρήσετε στα επόμενα κεφάλαια, όπου εξετάζεται αναλυτικά το παράδειγμα διαδικασιακού προγραμματισμού, βεβαιωθείτε ότι μπορείτε να εφαρμόσετε καθεμία από τις τεχνικές και καθένα από τα εργαλεία αυτά και ότι έχετε κατανοήσει πλήρως τις έννοιες που παρουσιάσθηκαν.

Βιβλιογραφία Κεφαλαίου 5

[1] Caine, S. and Gordon, K. (1975), *PDL: a tool for Software design*, Proceedings of National Computer Conference, IFIP Press, σελ. 271–276.

[2] Chapin, N. (1974), *A new format for Flowchars,* Software Practice & Experience, 4(4), σελ. 341–357.

[3] IBM (1974), *Improved Programming Technologies-an Overview,* GC20-1850-0. IBM Co., White Plains, NY.

[4] IBM (1975), *HIPO–a Design Aid and Documentation Technique*, GC20–1851–1. IBM Co., White Plains, NY.

[5] Jackson, M. (1975), *Principles of Program Design,* Academic Press.

[6] Nassi and Shneiderman, B. (1973), *Flowchart Techniques for Structured Programming,* ACM SIGPLAN Notices.

[7] Orr, K. (1978), *Structured Design*, Yourdon Press, NY.

[8] Stevens, W., Myers, G. and Constantine, L. (1974), *Structured Design*, IBM Systems Journal, 13(2), σελ. 115–139.

[9] Warnier, D. J. (1974), *Logical Construction of Programs*, Van Nostrand Reinhold, NY

ΚΕΦΑΛΑΙΟ 6

Αρχές δομημένου διαδικασιακού προγραμματισμού

Σκοπός

Ο σκοπός του κεφαλαίου είναι να σας «μυήσει» στη φιλοσοφία του δομημένου προγραμματισμού και να σας μάθει να γράφετε αλγορίθμους για διαδικασιακά προγράμματα

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *περιγράψετε τις δύο αρχές, τις τρεις βασικές δομές (ακολουθία εντολών, απόφαση, επανάληψη) και τουλάχιστον άλλες τρεις προγραμματιστικές δομές του δομημένου προγραμματισμού*
- *αναφέρετε τουλάχιστον τρία χαρακτηριστικά του δομημένου προγραμματισμού*
- *ορίζετε τις έννοιες μεταβλητή, τύπος δεδομένων, δομή δεδομένων, τελεστής, παράμετρος, έκφραση*
- *σχεδιάσετε αλγορίθμους που χρησιμοποιούν πίνακες (αρχικοποίηση πίνακα, προσπέλαση στοιχείων πίνακα, πράξεις με στοιχεία ενός πίνακα και με πίνακες, κ.ά.)*
- *σχεδιάσετε αλγορίθμους που χρησιμοποιούν διασυνδεδεμένες λίστες (κατασκευή λίστας, πρόσθεση – παρεμβολή – διαγραφή κόμβου κ.ά.)*
- *αναφέρετε δύο γενικούς κανόνες εφαρμογής της εντολής GOTO*

Έννοιες κλειδιά

- *Δομημένος προγραμματισμός*
- *Ακολουθία εντολών*
- *Απόφαση – επιλογή*
- *Επανάληψη*
- *Φωλιασμένη απόφαση*
- *Επιλογή με πολλά ενδεχόμενα*
- *Μεταβλητή*
- *Σταθερά*

- *Τύπος δεδομένων*
- *Δομή δεδομένων*
- *Αρχικοποίηση μεταβλητής*
- *Τελεστής*
- *Παράμετρος*
- *Έκφραση*
- *Πίνακας*
- *Διασυνδεδεμένη λίστα*
- *Διπλά διασυνδεδεμένη λίστα*
- *Στοίβα*
- *Ουρά*
- *Διπλή ουρά*
- *Δένδρο*
- *GOTO*

Εισαγωγικές παρατηρήσεις

*Ο **δομημένος προγραμματισμός (structured programming)** ικανοποιεί τις αρχές και τις έννοιες που αναφέρθηκαν στα κεφάλαια 3 και 4, αποτελεί φυσική εξέλιξη της από πάνω προς τα κάτω σχεδίασης και ουσιαστικά περιλαμβάνει την ΚΒΕ και τον αρθρωτό προγραμματισμό. Η έννοια του δομημένου προγράμματος (και κατ'επέκταση, η τεχνική του δομημένου προγραμματισμού) πρωτοεμφανίστηκαν το 1966 (Bohm, 1966), αλλά διαδόθηκαν στις αρχές της δεκαετίας του 70. Η μεγάλη απήχηση που γνώρισε ο δομημένος προγραμματισμός οφείλεται στο ότι δεν χρησιμοποιεί εντολές αυθαίρετης αλλαγής της ροής ελέγχου του προγράμματος (η περίφημη εντολή GOTO). Στη μαζική χρήση τέτοιων εντολών είχε τότε αποδοθεί η μεγάλη δυσκολία κατανόησης και συντήρησης των προγραμμάτων που είχαν γραφεί σε προηγούμενες δεκαετίες (Dijkstra, 1968).*

Αντί της χρήσης εντολών GOTO, οι Bohm και Jacopini απέδειξαν ότι οποιοδήποτε πρόγραμμα υπολογιστή (ανεξάρτητα από τη γλώσσα που χρησιμοποιείται) μπορεί να γραφεί χρησιμοποιώντας τρεις βασικές

δομές προγραμματισμού (ακολουθία εντολών, επιλογή υπό συνθήκη και επανάληψη υπό συνθήκη), όταν ισχύουν οι εξής δύο προϋποθέσεις:

- *Σε κάθε δομή προγραμματισμού (και κατ' επέκταση στο πρόγραμμα) υπάρχει μόνο μια είσοδος (αρχή της ροής ελέγχου) και μια έξοδος (τέλος της ροής ελέγχου),*
- *η ροή μεταξύ της εισόδου και της εξόδου σε κάθε δομή προγραμματισμού είναι ομαλή και απρόσκοπτη (δηλαδή δεν συμβαίνει απότομος τερματισμός ή άλλου είδους διακοπή της ροής, δεν υπάρχουν εντολές που εκτελούνται άπειρες φορές κ.λπ.)*

Οι τρεις βασικές δομές δομημένου προγραμματισμού περιγράφονται στην ενότητα 6.1, ενώ στην ενότητα 6.2 παρουσιάζονται μερικές ακόμη δομές προγραμματισμού που χρησιμοποιούνται αρκετά συχνά.

Στη συνέχεια, παρουσιάζεται σε μορφή ψευδοκώδικα ένα πλήθος αλγορίθμων επίλυσης προβλημάτων, αφού γίνει στην ενότητα 6.3 μια σύντομη εισαγωγή σε βασικές έννοιες προγραμματισμού. Θα σας ήταν ιδιαίτερα χρήσιμο να προσπαθήσετε να εκφράσετε τους αλγορίθμους αυτούς χρησιμοποιώντας ΔΡΠ ή Δομοδιάγραμμα, αλλά και να μελετήσετε την υλοποίησή τους ως προγράμματα σε κάποια διαδικασιακή γλώσσα προγραμματισμού (π.χ., Pascal στο Cooper, 1985 ή C στο Kernighan, 1988). Ακόμη, κατά την εκπόνηση των δραστηριοτήτων, να θυμάστε ότι καθένας μας μπορεί να σχεδιάσει διαφορετικό αλγόριθμο επίλυσης του ίδιου προβλήματος, χωρίς απαραίτητα να έχει κάνει λάθος. Το ζητούμενο λοιπόν είναι να σχεδιάσετε έναν ορθό αλγόριθμο (σύμφωνα με αυτά που αναφέρθηκαν στο κεφάλαιο 2) και να τον συγκρίνετε με αυτόν που κάθε φορά προτείνουμε, με πρωταρχικό στόχο να εντοπίσετε διαφορές και όχι λάθη. Στη συνέχεια, αναλογιζόμενοι τις αιτίες που προκαλούν τις διαφορές, θα μπορέσετε να επιλέξετε τον καλύτερο κατά τη γνώμη σας αλγόριθμο.

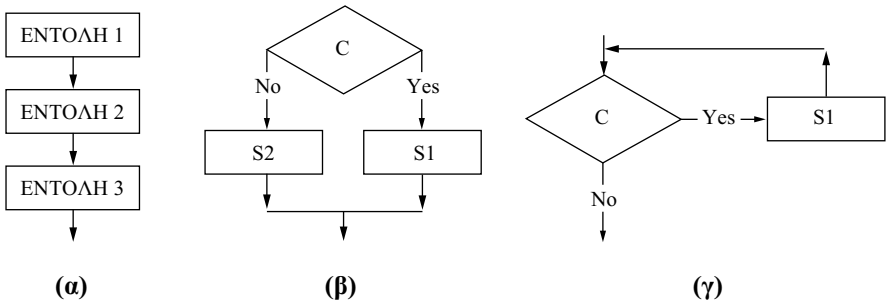
Οι ενότητες 6.4 και 6.5 περιγράφουν αντίστοιχα αλγορίθμους για τη διαχείριση των στατικών (πίνακες) και δυναμικών (λίστες) δομών αποθήκευσης δεδομένων που μπορούμε να χρησιμοποιήσουμε σε ένα πρόγραμμα. Παρόλο που οι αλγόριθμοι αυτοί περιγράφονται με σχετικά απλό τρόπο, καλό θα ήταν να είστε εξοικειωμένοι με την ύλη της Θ.Ε. «Δομές Δεδομένων» του ΕΑΠ.

Τέλος, στην ενότητα 6.6 ξεκαθαρίζουμε τις «επιτρεπόμενες» παραβιάσεις των κανόνων που ο δομημένος προγραμματισμός επιβάλλει.

6.1 Βασικές δομές προγραμματισμού

Οι τρεις βασικές προγραμματιστικές δομές που χρησιμοποιεί ο δομημένος προγραμματισμός είναι οι εξής (οι δομές αυτές έχουν συνοπτικά αναφερθεί στην ενότητα 2.3 και περιγράφονται χρησιμοποιώντας ΔΡΠ στο Σχήμα 6.1):

- Η **δομή της ακολουθίας εντολών**, η οποία απλά ορίζει ότι ο έλεγχος εκτέλεσης ενός προγράμματος προχωρά από μια εντολή στην επόμενη, χωρίς αυθαίρετη διακοπή [Σχήμα 6.1(α)].
- Η **δομή της απόφασης** (λέγεται και επιλογή), η οποία ορίζει πως όταν η συνθήκη C ισχύει (είναι αληθής), τότε εκτελείται το τμήμα S1, ενώ όταν η συνθήκη είναι ψευδής εκτελείται το S2 (άρα τα τμήματα κώδικα S1 και S2 είναι αμοιβαία αποκλειόμενα). Στο τέλος της εκτέλεσης κάθε τμήματος, ο έλεγχος μεταφέρεται στην οδηγία που δηλώνει το τέλος της δομής απόφασης και από εκεί στην επόμενη οδηγία [Σχήμα 6.1(β)].
- Η **δομή της επανάληψης** (πολλές φορές αναφέρεται και ως δομή ανακύκλωσης), η οποία ορίζει ότι η εκτέλεση ενός τμήματος μπορεί να επαναληφθεί πολλές φορές, ανάλογα με την τιμή της συνθήκης C. Έτσι, πριν εκτελεστεί το S1, ελέγχεται η τιμή της συνθήκης C. Εάν αυτή είναι αληθής, τότε εκτελείται το S1 και έπειτα ελέγχεται πάλι η συνθήκη. Με τον τρόπο αυτό, το S1 θα εκτελείται συνεχώς, έως ότου η συνθήκη πάψει να ισχύει (συνεπώς, για να αποφύγουμε την ατέρμονη επανάληψη της εκτέλεσης του S1, πρέπει να τροποποιούμε την τιμή της συνθήκης μέσα στο S1). Εάν η δομή επανάληψης περιγραφεί σωστά, τότε το S1 μπορεί να εκτε-



Σχήμα 6.1

Οι τρεις βασικές δομές του δομημένου προγραμματισμού

λεστεί από 0 έως N φορές. Η έξοδος συμβαίνει όταν η συνθήκη πάψει να ισχύει, και ο έλεγχος περνά από την οδηγία όπου ελέγχεται η τιμή της στην οδηγία που βρίσκεται αμέσως μετά το τέλος της δομής επανάληψης [Σχήμα 6.1(γ)].

- Η εκφραστική δύναμη του δομημένου προγραμματισμού απορρέει από τα εξής χαρακτηριστικά του:
  - Την «απλότητα» της δομής και την ευκολία κατανόησης των τριών βασικών προγραμματιστικών δομών
  - Τη δυνατότητα ακολουθιακού συνδυασμού οποιουδήποτε αριθμού από τέτοιες δομές στο «χτίσιμο» ενός προγράμματος (με την προϋπόθεση ότι ο συνδυασμός είναι λογικά ορθός)
  - Τη δυνατότητα συνδυασμού των τριών δομών σε οποιοδήποτε βαθμό, αφού μια δομή μπορεί να θεωρηθεί ως μία «σύνθετη» εντολή (π.χ., στο Σχήμα 6.1(γ), το τμήμα S1 μπορεί να είναι μια ακολουθία εντολών ή μια ολόκληρη δομή απόφασης)

Όπως θα διαπιστώσετε στη συνέχεια, το μεγαλύτερο πλεονέκτημα του δομημένου προγραμματισμού είναι η ευκρίνεια της περιγραφής του αλγορίθμου και του προγράμματος. Χρησιμοποιώντας δομημένο προγραμματισμό είναι πιο εύκολο να περιγράψουμε τις ιδέες μας σε άλλους, οπότε μειώνεται η πολυπλοκότητα και ο χρόνος που απαιτείται για την κατανόηση του κώδικα ενός προγράμματος, με αποτέλεσμα να αυξάνεται και η παραγωγικότητα της ομάδας προγραμματισμού. Αυτός είναι ο λόγος για τον οποίο οι δομημένες τεχνικές προγραμματισμού ταιριάζουν καλά τόσο με τις «από πάνω προς τα κάτω» όσο και με τις «από κάτω προς τα πάνω» σχεδιαστικές μεθοδολογίες.

6.2 Μερικές ακόμη προγραμματιστικές δομές

Οι τρεις προγραμματιστικές δομές που περιγράφηκαν είναι αρκετές για να γραφεί οποιοδήποτε δομημένο πρόγραμμα. Είναι όμως τόσο στοιχειώδεις, που οδηγούν σε τεράστιες περιγραφές λογικά πολύπλοκων προγραμμάτων. Για το λόγο αυτό, στις σύγχρονες γλώσσες προγραμματισμού έχουν ενσωματωθεί και κάποιες άλλες προγραμματιστικές δομές, οι οποίες μπορούν να περιγραφούν χρησιμοποιώντας τις τρεις βασικές και, έτσι, δεν παραβιάζουν τις αρχές του δομημένου προγραμματισμού.

6.2.1 Φωλιασμένες αποφάσεις

Πολλές φορές τα τμήματα S1 και S2 δεν περιλαμβάνουν μόνο απλές οδηγίες, αλλά έχουν πολύπλοκη δομή. Αυτό δεν απαγορεύεται από την τεχνική του δομημένου προγραμματισμού, αρκεί να τηρούνται οι δύο προϋποθέσεις που περιγράφηκαν στις εισαγωγικές παρατηρήσεις. Μια τέτοια περίπτωση έχουμε όταν η συνθήκη της εντολής επιλογής είναι σύνθετη (δηλαδή χρησιμοποιεί λογικούς τελεστές για να συνδυάσει απλούστερες συνθήκες), οπότε χρησιμοποιούμε **«φωλιασμένες» (nested) εντολές επιλογής**.

6.2.2 Επιλογή με πολλά ενδεχόμενα

Η ανάγκη για μια τέτοια προγραμματιστική δομή εμφανίζεται όταν η συνθήκη της δομής επιλογής μπορεί να πάρει περισσότερες από δύο τιμές, οπότε έχουμε και περισσότερα από δύο τμήματα κώδικα που, ενδεχομένως, θα εκτελεστούν. Προσέξτε, στην προηγούμενη ενότητα περιγράψαμε μια δομή επιλογής με πολλές συνθήκες, καθεμία από τις οποίες μπορούσε να πάρει δύο μόνο τιμές.

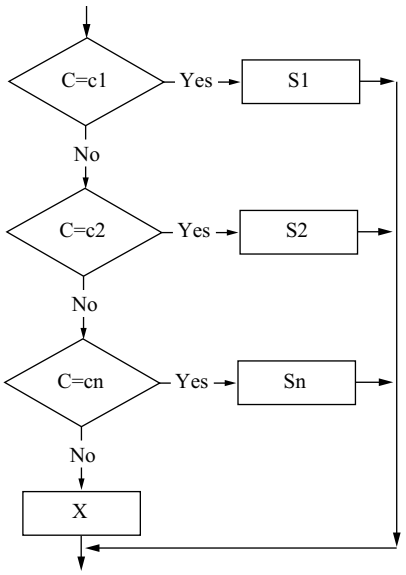
Η **δομή επιλογής με πολλά ενδεχόμενα** φαίνεται στο Σχήμα 6.2. Εάν η συνθήκη C πάρει την τιμή c1, εκτελείται το τμήμα κώδικα S1, εάν πάρει την τιμή c2, εκτελείται το S2, κ.ο.κ. Εάν η συνθήκη πάρει κάποια τιμή που δεν μας ενδιαφέρει (ή που δεν μπορούμε να προβλέψουμε), τότε εκτελείται το τμήμα X. Στο τέλος της εκτέλεσης κάθε τμήματος, ο έλεγχος μεταφέρεται στην εντολή που ακολουθεί τη δομή επιλογής. Σημειώστε ότι τα τμήματα S1, S2, ..., Sn, X είναι αμοιβαία αποκλειόμενα

Δραστηριότητα 6.1

Στην άσκηση αυτοαξιολόγησης 2 του κεφαλαίου 2 είχαμε περιγράψει έναν αλγόριθμο με τον οποίο μετράμε τη συχνότητα εμφάνισης των δέκα ψηφίων (0, 1, …, 9) σε μια ακολουθία 100 χαρακτήρων. Προσπαθήστε να ξαναγράψετε τον αλγόριθμο, χρησιμοποιώντας δομή επιλογής με πολλά ενδεχόμενα, πριν μελετήσετε τη δική μας πρόταση που ακολουθεί.

Παράδειγμα 6.1

Ανατρέξτε στην ενότητα 2.3 και το Σχήμα 2.7, όπου περιγράφεται ο αλγόριθμος MAX-XYZ, ο οποίος χρησιμοποιεί φωλιασμένες αποφάσεις.



Σχήμα 6.2
Δομή επιλογής με πολλά ενδεχόμενα

Σε σχέση με τον αλγόριθμο του κεφαλαίου 2, αντικαταστήσαμε τις 10 φωλιασμένες δομές απόφασης (ΕΑΝ-ΑΛΛΙΩΣ) με μία δομή πολλαπλής επιλογής, η οποία έχει 11 ενδεχόμενα (δείχνεται σε *πλάγια γραφή*). Τα δέκα πρώτα χρησιμοποιούνται για τη μέτρηση της συχνότητας εμφάνισης των ψηφίων, ενώ το τελευταίο περιγράφει την εξορισμού συμπεριφορά του προγράμματος, όταν δεν ισχύει καμία από τις προηγούμενες δέκα περιπτώσεις (εδώ, το πρόγραμμα μετρά και τη συχνότητα εμφάνισης χαρακτήρων που δεν είναι ψηφία, χρησιμοποιώντας τη μεταβλητή MX)

ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΡΗΣΗ-ΨΗΦΙΩΝ-ΕΚΔ2

```
ΔΕΔΟΜΕΝΑ
    N: CHAR ;
    I, MX: INTEGER ;
    M: ARRAY[0..9] OF INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 0 ΕΩΣ 9 ΕΠΑΝΕΛΑΒΕ
        M[I] := 0
    ΓΙΑ-ΤΕΛΟΣ;
```

```
ΓΙΑ Ι := 1 ΕΩΣ 100 ΕΠΑΝΕΛΑΒΕ
    ΔΙΑΒΑΣΕ(Ν);
        ΕΠΕΛΕΞΕ (Ν) ΑΠΟ
            ΠΕΡΙΠΤΩΣΗ (Ν = 1)
                Μ[1] := Μ[1] + 1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 2)
                Μ[2] := Μ[2] +1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 3)
                Μ[3] := Μ[3] + 1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 4)
                Μ[4] := Μ[4] +1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 5)
                Μ[5] := Μ[5] + 1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 6)
                Μ[6] := Μ[6] +1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 7)
                Μ[7] := Μ[7] + 1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 8)
                Μ[8] := Μ[8] +1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 9)
                Μ[9] := Μ[9] + 1 ;
            ΠΕΡΙΠΤΩΣΗ (Ν = 0)
                Μ[0] := Μ[0] +1 ;
            ΕΞ' ΟΡΙΣΜΟΥ
                ΜΧ := ΜΧ + 1
        ΕΠΙΛΟΓΗ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ ;
    ΥΠΟΛΟΓΙΣΕ ΜΟ-ΠΙΝΑΚΑ-1ΧΝ(Μ)
ΤΕΛΟΣ
```

6.2.3 Άλλα δύο είδη επαναλήψεων

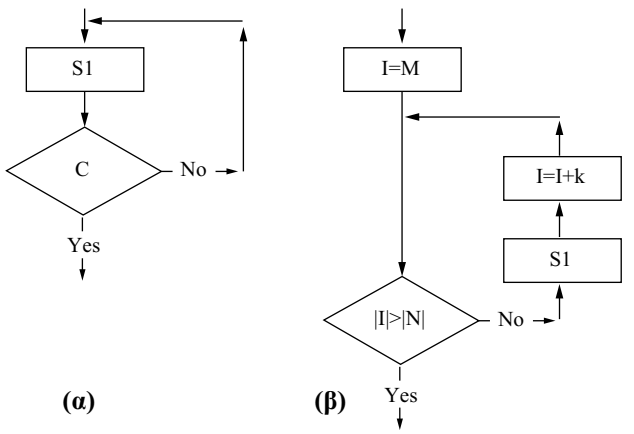
Η δομή επανάληψης που περιγράφηκε στην ενότητα 6.1 επαναλαμβάνει το τμήμα S1 από 0 έως N φορές, ανάλογα με την τιμή της συνθήκης C. Στην περίπτωση που θέλουμε το S1 να εκτελείται τουλάχιστον μία φορά ή να εκτελείται έναν καθορισμένο αριθμό φορών, η δομή αυτή οδηγεί στο γράψιμο επιπλέον κώδικα. Για να απλοποιηθεί

η υλοποίηση τέτοιων προγραμμάτων δημιουργήθηκαν δύο νέες δομές: Η δομή επανάληψης, στην οποία η συνθήκη ελέγχεται μετά την εκτέλεση του S1, και η δομή επανάληψης με μετρητή.

Στην πρώτη περίπτωση [Σχήμα 6.3(α)], η τοποθέτηση της συνθήκης επανάληψης στο τέλος της δομής οδηγεί στην υποχρεωτική εκτέλεση του S1 για μια φορά τουλάχιστον.

Στη δεύτερη περίπτωση [Σχήμα 6.3(β)], ο μετρητής I αρχικοποιείται σε κάποια τιμή M, η οποία αυξάνει κατά k σε κάθε επανάληψη (συνήθως k = 1). Η εκτέλεση του S1 επαναλαμβάνεται μέχρι ο μετρητής να ξεπεράσει την τελική του τιμή N. Σημειώστε πως το S1 εκτελείται:

- όταν M = N (συνεπώς το S1 εκτελείται ακριβώς μία φορά),
- όταν N < M, υποθέτοντας ότι το βήμα επανάληψης k είναι αρνητικός αριθμός,
- όταν η τιμή του μετρητή γίνει ακριβώς ίση με N (συνεπώς, το S1 εκτελείται τόσες φορές όσες το ακέραιο μέρος του αριθμού (N-M)/k).



Σχήμα 6.3
Δύο επί πλέον δομές επανάληψης

Παράδειγμα 6.2

Ανατρέξτε στην ενότητα 2.3 και το Σχήμα 2.5, όπου περιγράφεται ο αλγόριθμος ΜΟ-ΠΙΝΑΚΑ-MxN, ο οποίος χρησιμοποιεί δομή επανάληψης με μετρητή.

6.3 Εισαγωγή στον προγραμματισμό

Μέχρι τώρα, έχετε μελετήσει και έχετε συνθέσει πολλούς αλγορίθμους, για την περιγραφή των οποίων χρησιμοποιήσατε κυρίως ψευδοκώδικα. Παρόλο που έχουμε ρητά συμφωνήσει στη βασική δομή

ενός αλγορίθμου (ενότητα 2.2) και στο λεξιλόγιο που θα χρησιμοποιούμε για την περιγραφή τους (κεφάλαιο 3 και ενότητα 5.1), έχουμε μόνο σιωπηρά αποδεχθεί και χρησιμοποιούμε κάποιους κανόνες σύνταξης, τους οποίους θα συνοψίσουμε εδώ.

Ένας αλγόριθμος λοιπόν, αποτελείται από οδηγίες (μερικές φορές τις αναφέρουμε και ως «εντολές») γραμμένες στη σειρά. Συνήθως, κάθε οδηγία γράφεται σε ανεξάρτητη γραμμή, αν και μια οδηγία μπορεί να εκτείνεται σε πολλές γραμμές. Οι οδηγίες αυτές διαβάζουν τα δεδομένα εισόδου του αλγορίθμου, τα επεξεργάζονται, και παράγουν τα δεδομένα εξόδου (σύμφωνα με όσα αναφέρθηκαν στην ενότητα 1.2). Όμως, πού ακριβώς βρίσκονται τα δεδομένα για όσο χρόνο τα χρειάζεται ο αλγόριθμος;

Όλοι οι προγραμματιστές έχουμε συμφωνήσει ότι για να αναφερθούμε στα δεδομένα ενός αλγορίθμου χρησιμοποιούμε μεταβλητές. Μια **μεταβλητή (variable)** αντιπροσωπεύει ένα χώρο αποθήκευσης ενός αντικειμένου δεδομένων (π.χ., η ημέρα του μήνα, το ύψος ενός ανθρώπου, το όνομα ενός προϊόντος, ένας ακέραιος αριθμός κ.ά.), και έχει τα εξής στοιχεία:

- Ένα όνομα, το οποίο φροντίζουμε να είναι μοναδικό και όσο το δυνατό αυτο-επεξηγηματικό του ρόλου της μεταβλητής μέσα στον αλγόριθμο.
- Έναν τύπο, ο οποίος καθορίζει το είδος των δεδομένων που μπορεί να αποθηκεύσει η μεταβλητή. Οι τύποι δεδομένων μπορεί να είναι απλοί (π.χ., ακέραιος, χαρακτήρας κ.ά.) αλλά μπορεί να είναι και σύνθετες «λογικές κατασκευές» που λέγονται **δομές δεδομένων (data structures).** Οι μεταβλητές της δεύτερης κατηγορίας λέγονται «συναθροιστού τύπου» (compound variables) και έχουν εσωτερική δομή.
- Ένα σύνολο δυνατών τιμών (πεδίο τιμών), το οποίο εν μέρει καθορίζεται από τον τύπο της μεταβλητής.
- Την τρέχουσα τιμή, η οποία πρέπει να ανήκει οπωσδήποτε στο σύνολο τιμών της μεταβλητής, αλλιώς θα δημιουργηθεί αστοχία του προγράμματος που υλοποιεί τον αλγόριθμο

Κάθε μεταβλητή που χρησιμοποιείται στον αλγόριθμο πρέπει να δηλωθεί στην αρχή του, στο τμήμα ΔΕΔΟΜΕΝΑ. Εκεί, δηλώνονται κυρίως

το όνομα και ο **τύπος δεδομένων** της μεταβλητής (και ίσως το σύνολο των δυνατών τιμών, εάν αυτό δεν προκύπτει από τον τύπο της). Σε κάθε μεταβλητή καταχωρείται, πριν αυτή χρησιμοποιηθεί, μια αρχική τιμή, γίνεται όπως λέμε **αρχικοποίηση μεταβλητής**. Η τιμή αυτή μπορεί να αλλάξει, εάν η μεταβλητή συμμετέχει σε οδηγίες του αλγορίθμου (πιο συγκεκριμένα, εάν η μεταβλητή βρίσκεται στο αριστερό μέρος κάποιας οδηγίας καταχώρησης) ή απλά να χρησιμοποιηθεί σε διάφορους υπολογισμούς (εάν η μεταβλητή βρίσκεται στο δεξί μέρος κάποιας οδηγίας καταχώρησης).

Στο τμήμα ΔΕΔΟΜΕΝΑ θα δηλώνουμε μόνο τις μεταβλητές που ανήκουν σε κάποιον από τους εξής τύπους:

- Αριθμός (INTEGER, εάν είναι ακέραιος, ή REAL, εάν είναι πραγματικός): πρόκειται για κάποιον από τους γενικά αποδεκτούς αρνητικούς ή θετικούς αριθμούς
- Χαρακτήρας (CHAR): αναπαριστά μια ακολουθία χαρακτήρων οποιουδήποτε μήκους και, συνήθως, αναγράφεται μέσα σε εισαγωγικά
- Δυαδικός αριθμός (BOOLEAN): μπορεί να πάρει μόνο δύο τιμές, 0 ή 1 (TRUE ή FALSE)
- Πίνακας (ARRAY): περιγράφεται στην ενότητα 6.5
- Λίστα (LIST): περιγράφεται στην ενότητα 6.6
- Δείκτης (POINTER) : περιγράφεται στην ενότητα 6.6

Εάν θέλουμε να χρησιμοποιήσουμε κάποιον άλλο τύπο, πρέπει πρώτα να τον ορίσουμε στο τμήμα ΤΥΠΟΣ ως σύνθεση των βασικών τύπων και, έπειτα, στο τμήμα ΔΕΔΟΜΕΝΑ, μπορούμε να ορίζουμε μεταβλητές αυτού του τύπου (στην ενότητα 6.6 αναφέρονται τέτοια παραδείγματα).

Υπάρχει όμως και η περίπτωση να θέλουμε κάποια μεταβλητή του προγράμματος να έχει σταθερή τιμή, η οποία θα παραμένει αναλλοίωτη καθ' όλη τη διάρκεια εκτέλεσης του προγράμματος. Αυτές οι «μεταβλητές» καλούνται **σταθερές (constants)** και δηλώνονται, συνήθως, σε ένα ξεχωριστό τμήμα δηλώσεων στην αρχή του προγράμματος. Ουσιαστικά, πρόκειται για ονομασία σταθερών ποσοτήτων με ένα όνομα που μπορούμε εύκολα να θυμόμαστε. Η δήλωση μιας σταθεράς περιλαμβάνει το όνομα και την τιμή της, π.χ.:

ΣΤΑΘΕΡΕΣ

PI = 3.141592654 ;

Προσοχή: Σύμφωνα με όσα έχουν αναφερθεί έως τώρα, μια σταθερά δεν μπορεί να βρίσκεται στο αριστερό μέρος κάποιας οδηγίας καταχώρησης ή υπολογισμού!

Στις οδηγίες ενός αλγορίθμου, οι μεταβλητές χρησιμοποιούνται για να γίνουν διάφορες (αριθμητικές ή λογικές) πράξεις. Κάθε πράξη περιγράφεται από τον αντίστοιχο **τελεστή**, όπως **φαίνεται** στον Πίνακα 6.1.

Πίνακας 6.1

Οι τελεστές των διαφόρων πράξεων

ΑΡΙΘΜΗΤΙΚΟΙ		**ΣΧΕΣΙΑΚΟΙ - ΛΟΓΙΚΟΙ**	
:=	ΚΑΤΑΧΩΡΗΣΗ	=	ΙΣΟ ΜΕ
+	ΠΡΟΣΘΕΣΗ	>	ΜΕΓΑΛΥΤΕΡΟ ΑΠΟ
-	ΑΦΑΙΡΕΣΗ	>=	ΜΕΓΑΛΥΤΕΡΟ Ή ΙΣΟ
*	ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ	<	ΜΙΚΡΟΤΕΡΟ ΑΠΟ
/	ΔΙΑΙΡΕΣΗ	<=	ΜΙΚΡΟΤΕΡΟ Ή ΙΣΟ
Div	ΑΚΕΡΑΙΑ ΔΙΑΙΡΕΣΗ	< >	ΔΙΑΦΟΡΟ (ΟΧΙ ΙΣΟ)
Mod	ΥΠΟΛΟΙΠΟ ΔΙΑΙΡΕΣΗΣ	AND	ΛΟΓΙΚΟ «ΚΑΙ» (ΣΥΖΕΥΞΗ)
^	ΕΜΜΕΣΗ ΠΡΟΣΠΕΛΑΣΗ	OR	ΛΟΓΙΚΟ «Ή» (ΔΙΑΖΕΥΞΗ)
		NOT	ΛΟΓΙΚΟ «ΟΧΙ» (ΑΝΑΙΡΕΣΗ)

Οι πράξεις με τελεστές που περιέχουν οι οδηγίες (λέγονται και **εκφράσεις**) υπολογίζονται από αριστερά προς τα δεξιά (οι λογικές πράξεις προηγούνται των αριθμητικών). Αρχικά υπολογίζονται οι διαιρέσεις και τα γινόμενα και, ακολούθως, οι προσθέσεις και οι αφαιρέσεις. Εάν επιθυμούμε να αλλάξουμε αυτή τη σειρά, πρέπει να χρησιμοποιήσουμε παρενθέσεις.

Τέλος, είδαμε ότι πολλές φορές ένας αλγόριθμος χρησιμοποιεί (καλεί) έναν άλλο αλγόριθμο. Σε τέτοια περίπτωση, οι δύο αλγόριθμοι μπορεί να ανταλλάξουν κάποιες μεταβλητές, οι οποίες στη συγκεκριμένη

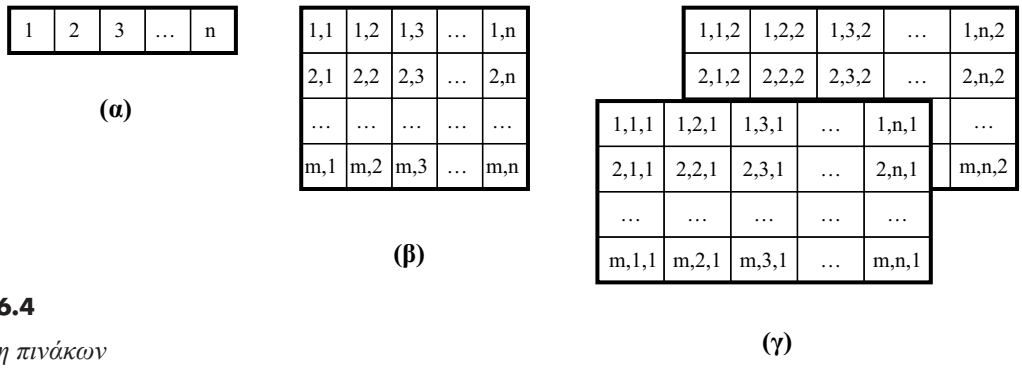
περίπτωση λέγονται **παράμετροι (parameters).** Το καλούμενο πρόγραμμα μπορεί να χρησιμοποιήσει μια μεταβλητή που δηλώνεται στο καλών πρόγραμμα (λέγεται «σφαιρική μεταβλητή» – global variable) ή μια παράμετρο που «περνιέται» κατά την κλήση, χωρίς αυτό να δηλωθεί ξανά. Περισσότερα για τη σχέση προγραμμάτων – υποπρογραμμάτων θα συζητήσουμε στο κεφάλαιο 7. Προς το παρόν, να θυμάστε ότι:

Η απρόσεκτη χρήση σφαιρικών μεταβλητών μπορεί να προκαλέσει «παρενέργειες» (side–effects) κατά την εκτέλεση του προγράμματος, οπότε καλό είναι να τις αποφεύγουμε κατά το δυνατόν.

Όσο μεγαλύτερος ο αριθμός των παραμέτρων που ανταλλάσσονται ανάμεσα σε δύο προγράμματα, τόσο μεγαλύτερος και ο βαθμός της σύζευξης των προγραμμάτων (θυμηθείτε όσα συζητήσαμε στην ενότητα 4.3.2).

6.4 Πίνακες

Ο **πίνακας (array)** είναι η πιο κοινή δομή δεδομένων στον προγραμματισμό και υποστηρίζεται από όλες τις γλώσσες προγραμματισμού υψηλού επιπέδου. Σχεδόν οποιοδήποτε σύνολο στοιχείων μπορεί να αποθηκευθεί σε κάποιο πίνακα, γιατί κάθε αναφορά στα στοιχεία ενός πίνακα, συνήθως, γίνεται με βάση τη θέση τους στον πίνακα και όχι με βάση την τιμή τους. Στο Σχήμα 6.4 δείχνονται διάφοροι πίνακες. Παρατηρήστε ότι ένας πίνακας θεωρείται ως ένα σύνολο θέσεων (κελιών), καθεμία από τις οποίες περιγράφεται με ένα σύνολο «συντεταγμένων». Ο πίνακας του Σχήματος 6.4(α) είναι μονοδιάστατος 1xN, ενώ αυτός του Σχήματος 6.4(β) είναι δύο διαστάσεων MxN. Στο Σχήμα 6.4(γ) φαίνεται ένας πίνακας τριών διαστάσεων MxNx2, ο οποίος αναπαρίσταται ως ένα σύνολο (δύο στην περίπτωση αυτή) διδιάστατων πινάκων.



Σχήμα 6.4
Τρία είδη πινάκων

Στον τόμο αυτό έχουμε σιωπηρά υιοθετήσει ένα συγκεκριμένο τρόπο ορισμού ενός πίνακα ο οποίος, εκτός από το όνομα του πίνακα παρέχει τις εξής απαραίτητες πληροφορίες:

- Ποιες είναι οι διαστάσεις του πίνακα και πώς γίνεται αναφορά σε ένα στοιχείο του.
- Ποιο είναι το μέγεθος του πίνακα, δηλαδή πόσα στοιχεία μπορεί να αποθηκεύσει.
- Ποιος είναι ο τύπος δεδομένων των στοιχείων του πίνακα (όλα τα στοιχεία ενός πίνακα είναι υποχρεωτικά του ίδιου τύπου δεδομένων).

Άσκηση αυτοαξιολόγησης 6.1

Πριν συνεχίσετε, σκεφτείτε ποιες από τις παραπάνω πληροφορίες σάς δίνει η δήλωση P: ARRAY[1..M,1..N] OF INTEGER.

Η δομή των πινάκων είναι τέτοια, ώστε η αρχικοποίηση και η προσπέλασή τους, τις περισσότερες φορές, απαιτεί τη χρήση δομών επανάληψης. Έτσι, σε σύγκριση με τις δυναμικές δομές δεδομένων (περιγράφονται στην ενότητα 6.5), οι πίνακες όχι μόνο καταλαμβάνουν πολύ χώρο στη μνήμη, αλλά απαιτούν και περισσότερο χρόνο για την επεξεργασία τους.

Για παράδειγμα, σκεφτείτε ένα πίνακα 10.000 θέσεων από τις οποίες στο 90% των περιπτώσεων χρησιμοποιούνται μόνο οι 1.000: κάθε προσπέλαση του πίνακα απαιτεί πάντα 10.000 επαναλήψεις, ενώ

δεσμεύονται 10.000 «θέσεις» στη μνήμη (οι περισσότερες από τις οποίες στο 90% των περιπτώσεων είναι κενές). Αντίθετα, εάν είχε χρησιμοποιηθεί μια διασυνδεδεμένη λίστα, αυτή στο 90% των περιπτώσεων θα είχε 1.000 κόμβους (οπότε έχουμε πολύ αποδοτικότερη χρήση των πόρων του συστήματος), ενώ μόνο στο 10% των περιπτώσεων θα είχε 10.000 κόμβους (οπότε «πληρώνουμε» ένα αρκετά βαρύ τίμημα χώρου και επεξεργασίας).

- Οι πίνακες έχουν το πλεονέκτημα ότι επιτρέπουν την τυχαία προσπέλαση οποιουδήποτε στοιχείου τους, αλλά και το μειονέκτημα ότι έχουν σταθερές διαστάσεις που δεν αλλάζουν κατά την εκτέλεση του προγράμματος.

Παράδειγμα 6.3

Ας περιγράψουμε έναν αλγόριθμο με τον οποίο θα καταχωρήσουμε το αποτέλεσμα του πολλαπλασιασμού ενός ακέραιου αριθμού Z με όλους τους αριθμούς από 1 έως 10 (λέγεται και «προπαίδεια» του αριθμού Z) στον πίνακα PROD, ο οποίος έχει 10 θέσεις:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΡΟΠΑΙΔΕΙΑ(Z)
ΔΕΔΟΜΕΝΑ
    I: INTEGER ;
    PROD: ARRAY[1..10] OF INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 1 ΕΩΣ 10 ΕΠΑΝΕΛΑΒΕ
        PROD[I] := Z*I
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Ο ακέραιος I που χρησιμοποιείται για να διατρέξει τα στοιχεία του πίνακα καλείται **«απαριθμητής» (index).** Τα όρια (bounds) τιμών του απαριθμητή ενός μονοδιάστατου πίνακα N στοιχείων είναι από 1 έως N.

Εάν θέλουμε να τυπώσουμε τα περιεχόμενα του πίνακα PROD, τότε μπορούμε:

- είτε να προσθέσουμε μετά τη δομή επανάληψης που υπολογίζει την προπαίδεια, μια ακόμη δομή επανάληψης. Σε τέτοια περίπτω-

ση, πρώτα υπολογίζονται όλα τα στοιχεία του πίνακα και έπειτα τυπώνονται όλα μαζί:

```
ΓΙΑ I := 1 ΕΩΣ 10 ΕΠΑΝΕΛΑΒΕ
        ΤΥΠΩΣΕ(PROD[I])
ΓΙΑ-ΤΕΛΟΣ
```

- είτε, μέσα στην υπάρχουσα δομή επανάληψης, να προσθέσουμε την εντολή **ΤΥΠΩΣΕ**(PROD[I]), οπότε κάθε στοιχείο θα εκτυπώνεται μόλις υπολογιστεί.

Δραστηριότητα 6.2

Εάν θεωρήσουμε μια σκακιέρα σαν ένα πίνακα 8×8, τότε προσπαθήστε να σχεδιάσετε έναν αλγόριθμο που θα σημειώνει ποια τετράγωνα απειλεί μια βασίλισσα που έχει τοποθετηθεί σε οποιαδήποτε από τις τέσσερις γωνίες.

Χρησιμοποιώντας την τεχνική της ΚΒΕ που γνωρίσαμε στην ενότητα 3.1, σχεδιάζουμε την πρώτη μορφή του αλγορίθμου ως εξής:

```
ΑΛΓΟΡΙΘΜΟΣ ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ-ΕΚΔ0
ΔΕΔΟΜΕΝΑ
    CHESS: ARRAY[1..8,1..8] OF INTEGER ;
ΑΡΧΗ
    “ΑΡΧΙΚΟΠΟΙΗΣΗ ΣΚΑΚΙΕΡΑΣ”
    “ΣΗΜΕΙΩΣΗ ΤΕΤΡΑΓΩΝΩΝ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)”
    “ΣΗΜΕΙΩΣΗ ΤΕΤΡΑΓΩΝΩΝ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)”
    “ΣΗΜΕΙΩΣΗ ΤΕΤΡΑΓΩΝΩΝ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,1)”
    “ΣΗΜΕΙΩΣΗ ΤΕΤΡΑΓΩΝΩΝ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,8)”
ΤΕΛΟΣ
```

Κατ’ αρχήν, ας θυμηθούμε ότι μια βασίλισσα απειλεί όλα τα τετράγωνα που βρίσκονται στην ίδια γραμμή, στήλη, αριστερή και δεξιά διαγώνιο με αυτή. Στην περίπτωση του προβλήματος, επειδή κάθε

βασίλισσα βρίσκεται σε μια γωνία, θα εξετάσουμε μόνο γραμμή, στήλη και αριστερή ή δεξιά διαγώνιο, κατά περίπτωση. Ο αλγόριθμος λοιπόν γίνεται:

ΑΛΓΟΡΙΘΜΟΣ ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ-ΕΚΔ1

ΔΕΔΟΜΕΝΑ

CHESS: ARRAY[1..8,1..8] OF INTEGER ;

ΑΡΧΗ

“ΑΡΧΙΚΟΠΟΙΗΣΗ ΣΚΑΚΙΕΡΑΣ”
“ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)”
“ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)”
“ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)”
“ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)”
“ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)”
“ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)”
“ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,1)”
“ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,1)”
“ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,1)”
“ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,8)”
“ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,8)”
“ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,8)”

ΤΕΛΟΣ

Στη συνέχεια, παρατηρούμε ότι η βασίλισσα στο (1,1) απειλεί την ίδια γραμμή με τη βασίλισσα στο (1,8), την ίδια στήλη με τη βασίλισσα στο (8,1) και την ίδια διαγώνιο με τη βασίλισσα στο (8,8), η βασίλισσα στο (1,8) απειλεί την ίδια στήλη με τη βασίλισσα στο (8,8) και την ίδια διαγώνιο με τη βασίλισσα στο (8,1), και η βασίλισσα στο (8,1) απειλεί την ίδια γραμμή με τη βασίλισσα στο (8,8). Αφαιρούμε τους περιττούς υπολογισμούς και ο αλγόριθμος γίνεται:

ΑΛΓΟΡΙΘΜΟΣ ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ-ΕΚΔ2

ΔΕΔΟΜΕΝΑ

CHESS: ARRAY[1..8,1..8] OF INTEGER ;

ΑΡΧΗ

```
"ΑΡΧΙΚΟΠΟΙΗΣΗ ΣΚΑΚΙΕΡΑΣ"
"ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)"
"ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)"
"ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,1)"
"ΣΗΜΕΙΩΣΗ ΣΤΗΛΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)"
"ΣΗΜΕΙΩΣΗ ΔΙΑΓΩΝΙΟΥ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (1,8)"
"ΣΗΜΕΙΩΣΗ ΓΡΑΜΜΗΣ ΠΟΥ ΑΠΕΙΛΕΙ Η ΒΑΣΙΛΙΣΣΑ (8,1)"
```

ΤΕΛΟΣ

Τώρα μπορούμε να υπολογίσουμε ποιες συγκεκριμένες γραμμές και στήλες πρέπει να σημειωθούν, καθώς και τον τρόπο που θα αρχικοποιηθεί ο πίνακας που αντιπροσωπεύει τη σκακιέρα:

ΑΛΓΟΡΙΘΜΟΣ ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ-ΕΚΔ3

```
ΔΕΔΟΜΕΝΑ
    CHESS: ARRAY[1..8,1..8] OF INTEGER ;
    I, K: INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
        ΓΙΑ K:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
            CHESS [I,K] := 0
        ΓΙΑ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ
    ΥΠΟΛΟΓΙΣΕ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ(1, CHESS) ;
    ΥΠΟΛΟΓΙΣΕ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ(8, CHESS) ;
    ΥΠΟΛΟΓΙΣΕ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ(1, CHESS) ;
    ΥΠΟΛΟΓΙΣΕ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ(8, CHESS) ;
    ΥΠΟΛΟΓΙΣΕ ΔΕΞΙΑ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(CHESS) ;
    ΥΠΟΛΟΓΙΣΕ ΑΡΙΣΤΕΡΗ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(CHESS) ;
ΤΕΛΟΣ
```

Όπως γνωρίζετε, για να αρχικοποιήσουμε μονοδιάστατο πίνακα χρειαζόμαστε μια δομή επανάληψης. Συνεπώς, για ένα πίνακα δύο διαστάσεων θα χρειαστούμε δύο τέτοιες δομές (καθεμία με το δικό της απαριθμητή), οι οποίες πρέπει να είναι φωλιασμένες. Έτσι, η εντολή

CHESS [I, K] := 0 θα εκτελεστεί συνολικά 8*8 = 64 φορές. Τα όρια τιμών του απαριθμητή I είναι από 1 έως 8 και του K από 1 έως 8 επίσης. Επειδή ο απαριθμητής K των στηλών μεταβάλλεται πιο γρήγορα από τον απαριθμητή I των γραμμών (αλλιώς, η δομή επανάληψης που προσπελαύνει τις στήλες είναι φωλιασμένη μέσα σε αυτή των γραμμών), τα στοιχεία του πίνακα θα αρχικοποιηθούν κατά γραμμές.

Εκτός από την αρχικοποίηση, οι υπόλοιπες εργασίες του προγράμματος εκτελούνται από ανεξάρτητα τμήματα λογισμικού, σύμφωνα με την τεχνική του αρθρωτού προγραμματισμού (ενότητα 3.2). Οι αλγόριθμοι των επί μέρους τμημάτων περιγράφονται στη συνέχεια. Προσέξτε τη χρήση των απαριθμητών για την προσπέλαση γραμμών, στηλών και διαγωνίων του πίνακα.

ΑΛΓΟΡΙΘΜΟΣ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ (N, CHESS)

```
ΔΕΔΟΜΕΝΑ
    I : INTEGER ;
ΑΡΧΗ
    ΓΙΑ I:=1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
        CHESS [N,I] := 1
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

ΑΛΓΟΡΙΘΜΟΣ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ (N, CHESS)

```
ΔΕΔΟΜΕΝΑ
    I : INTEGER ;
ΑΡΧΗ
ΓΙΑ I:=1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
        CHESS [I,N] := 1
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

ΑΛΓΟΡΙΘΜΟΣ ΔΕΞΙΑ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ (CHESS)

```
ΔΕΔΟΜΕΝΑ
    I : INTEGER ;
ΑΡΧΗ
    ΓΙΑ I:=1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
        CHESS [I,I] := 1
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

ΑΛΓΟΡΙΘΜΟΣ ΑΡΙΣΤΕΡΗ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(CHESS)

```
ΔΕΔΟΜΕΝΑ
    I : INTEGER ;
ΑΡΧΗ
    ΓΙΑ I:=1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
        CHESS [I,(8-I)+1] := 1
    ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Δραστηριότητα 6.3

Ο ανάστροφος πίνακας Β του διδιάστατου πίνακα Α έχει ως στήλες τις γραμμές και ως γραμμές τις στήλες του Α. Περιγράψτε τον αλγόριθμο που επιλύει το πρόβλημα για ένα πίνακα χαρακτήρων Μ × Μ χρησιμοποιώντας, όπως έχουμε ήδη συναντήσει πιο πριν, δύο φωλιασμένες δομές επανάληψης. Έπειτα, συγκρίνετε την απάντησή σας με τον αλγόριθμο που ακολουθεί

ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΣΤΡΟΦΟΣ-ΠΙΝΑΚΑΣ

```
ΔΕΔΟΜΕΝΑ
    A, B: ARRAY[1..M, 1..M] OF CHAR ;
    I, K: INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 1 ΕΩΣ M ΕΠΑΝΕΛΑΒΕ
        ΓΙΑ K := 1 ΕΩΣ M ΕΠΑΝΕΛΑΒΕ
            B[I,K] := A[K,I]
        ΓΙΑ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ

ΤΕΛΟΣ
```

Δραστηριότητα 6.4

Το γινόμενο δύο πινάκων Α Μ × Ν και Β Ν × Τ, είναι ένας πίνακας G Μ × Τ, του οποίου το στοιχείο G[n,m] προκύπτει αθροίζοντας τα γινόμενα κάθε στοιχείου της γραμμής n του Α επί το αντίστοιχο στοιχείο της στήλης m του Β. Περιγράψτε τον αλγόριθμο που επιλύει το πρόβλημα. Έπειτα, συγκρίνετε την απάντησή σας με τον αλγόριθμο που ακολουθεί.

ΑΛΓΟΡΙΘΜΟΣ ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ

```
ΔΕΔΟΜΕΝΑ
    A: ARRAY[1..M, 1..N] OF INTEGER ;
    B: ARRAY[1..N, 1..T] OF INTEGER ;
    G: ARRAY[1..M, 1..T] OF INTEGER ;
    I, K, Z, EL: INTEGER ;
ΑΡΧΗ
```

```
ΓΙΑ I := 1 ΕΩΣ M ΕΠΑΝΕΛΑΒΕ
    ΓΙΑ Z := 1 ΕΩΣ T ΕΠΑΝΕΛΑΒΕ
        EL := 0 ;
        ΓΙΑ K := 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
            EL := EL + A[I,K]*B[K,Z]
        ΓΙΑ-ΤΕΛΟΣ
        G[I,Z] := EL
    ΓΙΑ-ΤΕΛΟΣ
ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Ο αλγόριθμος αυτός περιλαμβάνει τρεις φωλιασμένες δομές επανάληψης. Παρατηρήστε το «παιχνίδι» που γίνεται με τους απαριθμητές, καθώς η λύση του προβλήματος έγκειται στη σωστή χρήση τους: Ο απαριθμητής I προσπελαύνει τις γραμμές του πίνακα A, αλλά και του G, ενώ ο Z διατρέχει τις στήλες του B και του G. Ο απαριθμητής K ουσιαστικά διατρέχει την «κοινή» διάσταση των πινάκων A και B, η οποία καθορίζει και το πλήθος των όρων του αθροίσματος των γινομένων: σε κάθε επανάληψη του K, το νέο γινόμενο προστίθεται στο προηγούμενο (τρέχον) άθροισμα, το οποίο φυλάσσεται στη μεταβλητή EL (η οποία αρχικοποιείται στην τιμή 0 πριν από κάθε νέα είσοδο στη δομή επανάληψης του K). Αφού προστεθούν όλοι οι όροι, το προκύπτον άθροισμα καταχωρείται στο αντίστοιχο στοιχείο του πίνακα G. Σημειώστε ότι στοιχεία του πίνακα G υπολογίζονται κατά γραμμές, αφού ο Z μεταβάλλεται εσωτερικά του I.

Μελέτη περίπτωσης (συνέχεια)

(Η μελέτη περίπτωσης που ακολουθεί βασίζεται σε μια ιδέα του Καθ. Π. Πιντέλα, Ακαδημαϊκού Υπεύθυνου της Θ.Ε., τον οποίο ο συγγραφέας ευχαριστεί).

Η Ελένη είναι ικανοποιημένη από την εργασία του Βύρωνα και του ζήτησε να σχεδιάσει τα τμήματα ενός προγράμματος διαχείρισης της αποθήκης τού κάθε υποκαταστήματος της CHILDWARE. Στη συνέχεια, θα παρουσιάσουμε τους αλγόριθμους για τα εξής τρία από τα προβλήματα που πρέπει να λύνει το πρόγραμμα:

1. Υπολογισμός της συνολικής ποσότητας τεμαχίων ενός είδους που έχουν όλα τα υποκαταστήματα.
2. Υπολογισμός της συνολικής ποσότητας όλων των ειδών που έχει στην αποθήκη του ένα υποκατάστημα.

3. Εύρεση του υποκαταστήματος που έχει τη μεγαλύτερη και τη μικρότερη ποσότητα κάποιου είδους.

Δραστηριότητα 6.5

Πριν προχωρήσετε στην ανάγνωση των αλγορίθμων που έχει ετοιμάσει ο Βύρων, δοκιμάστε να σχεδιάσετε τους δικούς σας και να τους συγκρίνετε με αυτούς που ακολουθούν. Υποθέστε ότι, προς το παρόν, το πρόγραμμα πρέπει να λειτουργεί για 10 υποκαταστήματα και 50 είδη.

Στους αλγορίθμους που ακολουθούν χρησιμοποιήσαμε το διδιάστατο πίνακα STOCK για να αποθηκεύσουμε τις ποσότητες των ειδών που διαθέτουν τα υποκαταστήματα (οι στήλες αναπαριστούν είδη και οι γραμμές υποκαταστήματα). Επιπλέον, χρησιμοποιήσαμε δύο σταθερές, τις STORES και ITEMS για να αποθηκεύσουμε το πλήθος των υποκαταστημάτων και των ειδών που διαθέτει προς το παρόν η CHILDWARE. Έτσι, εάν κάποια από αυτές τις ποσότητες αλλάξει, η μόνη τροποποίηση που χρειάζεται ο αλγόριθμος είναι η αλλαγή της τιμής της αντίστοιχης σταθεράς (για παράδειγμα, εάν η επιχείρηση ανοίξει ένα καινούριο κατάστημα, τότε η τιμή της STORES θα γίνει 11 σε όλα τα προγράμματα όπου αυτή χρησιμοποιείται).

ΑΛΓΟΡΙΘΜΟΣ ΠΟΣΟΤΗΤΑ-ΕΙΔΟΥΣ

```
ΣΤΑΘΕΡΕΣ
    STORES = 10 ;
    ITEMS = 50 ;
ΔΕΔΟΜΕΝΑ
    TOTAL-AMT, K, ITEM-INDEX: INTEGER ;
    STOCK: ARRAY[1..STORES, 1..ITEMS] OF INTEGER ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(ITEM-INDEX) ;
    ΕΑΝ ((ITEM-INDEX > 0) AND (ITEM-INDEX < ITEMS+1)) ΤΟΤΕ
        TOTAL-AMT := 0 ;
        ΓΙΑ K := 1 ΕΩΣ STORES ΕΠΑΝΕΛΑΒΕ
            TOTAL-AMT := TOTAL-AMT + STOCK[K,ITEM-INDEX]
        ΓΙΑ-ΤΕΛΟΣ ;
        ΤΥΠΩΣΕ("ΤΑ ΥΠΟΚΑΤΑΣΤΗΜΑΤΑ ΔΙΑΘΕΤΟΥΝ ΣΥΝΟΛΙΚΑ", TOTAL-AMT,
```

```
        "ΤΕΜΑΧΙΑ ΤΟΥ ΕΙΔΟΥΣ", ITEM-INDEX)
    ΑΛΛΙΩΣ
            ΤΥΠΩΣΕ («Ο ΚΩΔΙΚΟΣ ΤΟΥ ΕΙΔΟΥΣ ΠΡΕΠΕΙ ΝΑ ΕΙΝΑΙ ΑΚΕ-
                    ΡΑΙΟΣ ΑΡΙΘΜΟΣ ΑΝΑΜΕΣΑ ΣΤΟ 1 ΚΑΙ ΣΤΟ », ITEMS)
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Στον αλγόριθμο αυτό, δίνουμε τον κωδικό του είδους για το οποίο αναζητούμε τη συνολική ποσότητα (ITEM-INDEX). Εάν ο κωδικός είναι σωστός (δηλαδή ακέραιος αριθμός ανάμεσα στο 1 και 50), ο αλγόριθμος υπολογίζει και τυπώνει τη συνολική ποσότητα που διαθέτουν όλα τα υποκαταστήματα, αλλιώς τυπώνει ένα μήνυμα λάθους.

```
ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΟΛΟ-ΥΠΟΚΑΤΑΣΤΗΜΑΤΟΣ
ΣΤΑΘΕΡΕΣ
    STORES = 10 ;
    ITEMS = 50 ;
ΔΕΔΟΜΕΝΑ
    TOTAL-AMT, K, STORE-INDEX: INTEGER ;
    STOCK: ARRAY[1..STORES, 1..ITEMS] OF INTEGER ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(STORE-INDEX) ;
    ΕΑΝ ((STORE-INDEX > 0) AND (STORE-INDEX < STORES+1)) ΤΟΤΕ
        TOTAL-AMT := 0 ;
        ΓΙΑ K := 1 ΕΩΣ ITEMS ΕΠΑΝΕΛΑΒΕ
            TOTAL-AMT := TOTAL-AMT + STOCK[STORE-INDEX,K]
        ΓΙΑ-ΤΕΛΟΣ ;
        ΤΥΠΩΣΕ("ΤΟ ΥΠΟΚΑΤΑΣΤΗΜΑ", STORE-INDEX, "ΔΙΑΘΕΤΕΙ ΣΥΝΟΛΙΚΑ",
        TOTAL-AMT, "ΤΕΜΑΧΙΑ")
    ΑΛΛΙΩΣ
        ΤΥΠΩΣΕ («Ο ΚΩΔΙΚΟΣ ΤΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ ΠΡΕΠΕΙ ΝΑ ΕΙΝΑΙ ΑΚΕΡΑΙΟΣ
        ΑΡΙΘΜΟΣ ΑΝΑΜΕΣΑ ΣΤΟ 1 ΚΑΙ ΣΤΟ », STORES)
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Αντίστοιχα, στον αλγόριθμο αυτό χρησιμοποιούμε τον κωδικό καταστήματος STORE-INDEX για να βρούμε το σύνολο όλων των ειδών ενός υποκαταστήματος. Πάλι, ο αλγόριθμος εκτελείται μόνο εάν ο κωδικός είναι σωστός. Αλλιώς, τυπώνει ένα μήνυμα λάθους.

ΑΛΓΟΡΙΘΜΟΣ MIN-MAX-ΠΟΣΟΤΗΤΑ-ΕΙΔΟΥΣ

```
ΣΤΑΘΕΡΕΣ
    STORES = 10 ;
    ITEMS = 50 ;
ΔΕΔΟΜΕΝΑ
    STOCK: ARRAY[1..STORES, 1..ITEMS] OF INTEGER ;
    MIN, MAX, MIN-S, MAX-S, K, ITEM-INDEX: INTEGER ;
ΑΡΧΗ
    ΔΙΑΒΑΣΕ(ITEM-INDEX) ;
    ΕΑΝ ((ITEM-INDEX > 0) AND (ITEM-INDEX < ITEMS+1)) ΤΟΤΕ
        MIN-S := MAX-S := 1
        ΓΙΑ K := 2 ΕΩΣ STORES ΕΠΑΝΕΛΑΒΕ
            ΕΑΝ (STOCK[K,ITEM-INDEX] > STOCK[MAX-S,ITEM-INDEX]) ΤΟΤΕ
                MAX-S := K
            ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ (STOCK[K,ITEM-INDEX] < STOCK[MIN-S,ITEM-INDEX]) ΤΟΤΕ
                MIN-S := K
            ΕΑΝ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ
        ΤΥΠΩΣΕ("ΓΙΑ ΤΟ ΕΙΔΟΣ", ITEM-INDEX, "ΤΟ ΥΠΟΚΑΤΑΣΤΗΜΑ", MAX-S, "ΔΙΑ-
        ΘΕΤΕΙ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΠΟΣΟΤΗΤΑ ΑΠΟ", STOCK[MAX-S,ITEM-INDEX],
        "ΤΕΜΑΧΙΑ", ΚΑΙ ΤΟ ΥΠΟΚΑΤΑΣΤΗΜΑ", MIN-S, "ΔΙΑΘΕΤΕΙ ΤΗ ΜΙΚΡΟΤΕΡΗ
        ΠΟΣΟΤΗΤΑ ΑΠΟ", STOCK[MIN-S,ITEM-INDEX], "ΤΕΜΑΧΙΑ")
    ΑΛΛΙΩΣ
        ΤΥΠΩΣΕ(«Ο ΚΩΔΙΚΟΣ ΤΟΥ ΕΙΔΟΥΣ ΠΡΕΠΕΙ ΝΑ ΕΙΝΑΙ ΑΚΕΡΑΙΟΣ ΑΡΙΘΜΟΣ
        ΑΝΑΜΕΣΑ ΣΤΟ 1 ΚΑΙ ΣΤΟ », ITEMS)
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Μελέτη περίπτωσης (συνέχεια)

Αφού έλυσε αυτά τα σχετικά απλά προβλήματα, ο Βύρων αποφάσισε να ασχοληθεί με κάτι περισσότερο σύνθετο. Γνωρίζοντας ότι ταυτόχρονα με τη διαθέσιμη στην αποθήκη ενός υποκαταστήματος ποσότητα κάθε είδους, η κεντρική αποθήκη διατηρεί για κάθε είδος και υποκατάστημα έναν πίνακα με το κρίσιμο όριο κάτω από το οποίο χρειάζεται να γίνει αναπλήρωση της κάθε αποθήκης, ο Βύρων θα σχεδιάσει έναν αλγόριθμο, ο οποίος βρίσκει για κάθε υποκατάστημα τα είδη που χρειάζονται αναπλήρωση και την ποσότητα που λείπει.

Δραστηριότητα 6.6

Πριν προχωρήσετε, προσπαθήστε να βοηθήσετε τον Βύρωνα να σχεδιάσει έναν αλγόριθμο για το πρόβλημα αυτό. Έπειτα μελετήστε τον αλγόριθμο που ακολουθεί.

Για το πρόβλημα αυτό χρειαζόμαστε ένα πίνακα τριών διαστάσεων. Όπως έχουμε ήδη αναφέρει, ένας τέτοιος πίνακας μπορεί να θεωρηθεί ως ένα σύνολο διδιάστατων υποπινάκων. Στην περίπτωσή μας, αυτοί είναι δύο (όπως καθορίζεται από την «τρίτη» διάσταση): Ο υποπίνακας STOCK[1..STORES, 1..ITEMS, **1**], ο οποίος θα περιέχει (όπως πιο πριν) τις ποσότητες της αποθήκης κάθε υποκαταστήματος, ενώ ο υποπίνακας STOCK[1.. STORES, 1..ITEMS, **2**] θα περιέχει τα όρια αναπλήρωσης. Ακόμη, χρησιμοποιούμε τη σταθερά SURPLUS για να αποθηκεύσουμε την επιπλέον ποσότητα που θα παραγγελθεί πάνω από το όριο αναπλήρωσης κάθε είδους.

ΑΛΓΟΡΙΘΜΟΣ ΥΠΟΛΟΓΙΣΜΟΣ-ΑΝΑΠΛΗΡΩΣΗΣ

```
ΣΤΑΘΕΡΕΣ
        STORES = 10 ;
        ITEMS = 50 ;
        SURPLUS = 5 ;
ΔΕΔΟΜΕΝΑ
        REPL, K: INTEGER ;
        STOCK: ARRAY[1..STORES, 1..ITEMS, 1..2] OF INTEGER ;
ΑΡΧΗ
        ΓΙΑ K := 1 ΕΩΣ STORES ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ I := 1 ΕΩΣ ITEMS ΕΠΑΝΕΛΑΒΕ
```

```
            ΕΑΝ (STOCK[K, I, 1] <= STOCK[K, I, 2]) ΤΟΤΕ
                REPL := STOCK[K, I, 2] - STOCK[K, I, 1] + SURPLUS;
                ΤΥΠΩΣΕ("ΤΟ ΥΠΟΚΑΤΑΣΤΗΜΑ", K, "ΧΡΕΙΑΖΕΤΑΙ", REPL,
                "ΤΕΜΑΧΙΑ ΤΟΥ ΕΙΔΟΥΣ", I)
            ΕΑΝ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ

ΤΕΛΟΣ
```

6.4.1 Πρόληψη σφαλμάτων

Οι πίνακες χρησιμοποιούνται τόσο συχνά, ώστε αποτελούν μια από τις κυριότερες πηγές σφαλμάτων σε ένα πρόγραμμα. Τα σφάλματα αυτά οφείλονται κυρίως στην υπέρβαση των ορίων του πίνακα (που είναι στατικά), η οποία μπορεί να συμβεί:

- καθώς τον διατρέχει ο απαριθμητής μιας δομής επανάληψης, π.χ., ψάχνοντας για ένα στοιχείο που δεν περιέχεται τελικά στον πίνακα. Εάν δεν χρησιμοποιούμε το πλήθος των στοιχείων του πίνακα ως επιπλέον συνθήκη τερματισμού, τότε κάποια στιγμή, αναπόφευκτα, ο απαριθμητής θα ξεπεράσει τα όρια του πίνακα και θα προκαλέσει σφάλμα εκτέλεσης. Εάν χρησιμοποιούμε τέτοια συνθήκη, θα πρέπει μετά το τέλος της δομής επανάληψης να εξετάσουμε εάν πραγματικά βρέθηκε το στοιχείο, ή απλά τελείωσαν τα στοιχεία του πίνακα
- καθώς προσπαθούμε να χρησιμοποιήσουμε π.χ. τις τιμές των γειτονικών προς τη θέση (χ,ψ) θέσεων ενός πίνακα $M \times N$. Εάν $\chi = 1$, $\psi = 1$, $\chi = M$ ή $\psi = N$ κάποιες από τις γειτονικές θέσεις, απλά, δεν υπάρχουν!

Ακόμη, όταν ο πίνακας είναι πολλών διαστάσεων, χρειάζεται προσοχή στη χρήση των φωλιασμένων δομών επανάληψης, καθώς μια απροσεξία μπορεί να προκαλέσει σφάλμα.

Για να προλάβουμε τέτοια σφάλματα, καλό είναι κατά την ανάπτυξη του προγράμματος να συμπεριλάβουμε εντολές αμυντικού προγραμματισμού, με τις οποίες σε διάφορα σημεία του προγράμματος θα τυπώνουμε τα στοιχεία του πίνακα και τις τιμές των απαριθμητών που τον διατρέχουν.

6.5 Δυναμικές δομές δεδομένων

Για να αποθηκεύσουμε ένα σύνολο δεδομένων έχουμε έως τώρα χρησιμοποιήσει πίνακες. Σύμφωνα με όσα γνωρίζουμε, λοιπόν, εάν η Ελένη ζητήσει ένα πρόγραμμα για να διατηρεί έναν ταξινομημένο κατάλογο με τα ονόματα των συνεργατών της εταιρείας της (π.χ., των προμηθευτών, των αντιπροσώπων κ.ά.), ο Βύρων θα χρησιμοποιήσει ένα μονοδιάστατο πίνακα χαρακτήρων. Εάν αποθηκεύεται και το τηλέφωνο του κάθε συνεργάτη, τότε θα χρησιμοποιήσει πίνακα δύο διαστάσεων.

Λοιπόν, εάν το σκεφτούμε λίγο περισσότερο, θα δούμε ότι κάτι δεν πάει καλά εδώ. Τι θα γίνει εάν η Ελένη συνάψει εμπορική συμφωνία με ένα νέο συνεργάτη; Το όνομά του θα πρέπει να εισαχθεί στην κατάλληλη θέση του πίνακα των συνεργατών, οπότε μερικά από τα υπάρχοντα ονόματα ίσως χρειαστεί να μετακινηθούν. Και εάν ο πίνακας γεμίσει; (όπως θυμάστε, ο αριθμός των θέσεων ενός πίνακα είναι πάντα σταθερός). Ε, τότε η Ελένη μάλλον θα πρέπει να διακόψει τη συνεργασία της με κάποιον από τους υπάρχοντες συνεργάτες! Από την άλλη, για κάθε συνεργάτη, η Ελένη μπορεί να καταγράψει το πολύ ένα τηλέφωνο!

Δε χρειάζεται να σκεφτούμε πολύ για να συμπεράνουμε ότι είναι παράλογο να προσαρμόζουμε τις απαιτήσεις των χρηστών στις δυνατότητες ενός προγράμματος, ενώ το αντίθετο θα έπρεπε κανονικά να συμβαίνει. Το πρόβλημα, στην περίπτωση αυτή, οφείλεται στην ακαταλληλότητα της δομής αποθήκευσης των δεδομένων που χρησιμοποιούμε. Ο πίνακας είναι μια στατική δομή δεδομένων: όχι μόνο δεν μπορούμε να αλλάξουμε μέσα στο πρόγραμμα το μέγιστο πλήθος στοιχείων που μπορεί να αποθηκεύσει, αλλά είναι αρκετά δύσκολο ακόμη και να παρεμβάλλουμε ένα νέο στοιχείο σε μια «κατειλημμένη» θέση του.

Για να ξεπεραστούν τέτοια προβλήματα, ορίζονται δυναμικές δομές δεδομένων, οι οποίες αποτελούνται από κόμβους συνδεδεμένους με δείκτες. Πώς; Δεν το καταλάβατε; Ήταν αναμενόμενο, μην ανησυχείτε. Θα αρχίσουμε πρώτα εξηγώντας τους δείκτες και θα επανέλθουμε στις δυναμικές δομές δεδομένων.

6.5.1 Δείκτες

Έχουμε ήδη αναφέρει στην ενότητα 6.3 ότι στον προγραμματισμό χρησιμοποιούμε τις μεταβλητές για να αναφερθούμε σε κάποια δυνητικά μεταβαλλόμενη ποσότητα και τις σταθερές για να αναφερθούμε σε κάποια αμετάβλητη. Τι συμβαίνει όμως κάθε φορά που εμείς δηλώνουμε την πρόθεσή μας να χρησιμοποιήσουμε μια μεταβλητή ή μια σταθερά;

Κάθε δήλωση στο τμήμα ΣΤΑΘΕΡΕΣ ή ΔΕΔΟΜΕΝΑ αναγκάζει τον υπολογιστή, κατά την εκτέλεση του αντίστοιχου προγράμματος, να δεσμεύσει ένα μικρό τμήμα της μνήμης του (συνήθως, λέμε ότι δεσμεύει μια «θέση μνήμης»), στο οποίο θα καταχωρεί ή από το οποίο θα διαβάζει την τιμή της μεταβλητής. Κατά τη μεταγλώττιση του προγράμματος, συσχετίζεται η θέση αυτή με το όνομα και τον τύπο της μεταβλητής.

Έτσι, κάθε φορά που εμείς θέλουμε να χρησιμοποιήσουμε την τιμή της μεταβλητής (δηλαδή τα περιεχόμενα της θέσης μνήμης) αρκεί να χρησιμοποιήσουμε το όνομα της μεταβλητής. Αυτός ο τρόπος χρήσης λέγεται άμεση προσπέλαση (direct access) της θέσης μνήμης. Για παράδειγμα, στο Σχήμα 6.5, η μεταβλητή MAX συσχετίζεται άμεσα με τη θέση μνήμης 12364 όπου φυλάσσεται η τιμή 145. Εμείς δεν χρειάζεται να χρησιμοποιούμε τον αριθμό 12364 για να αναφερθούμε στην τιμή 145 – αρκεί η χρήση του «μνημονικού» ονόματος MAX.

Υπάρχει όμως και ένα άλλο είδος μεταβλητής, η τιμή της οποίας είναι το όνομα (δηλαδή, η διεύθυνση) μιας θέσης μνήμης. Η μεταβλητή αυτή καλείται **δείκτης (pointer)** γιατί δείχνει (αναφέρεται) σε μια θέση μνήμης και μας επιτρέπει με τον τρόπο αυτό να έχουμε έμμεση προσπέλαση (indirect access) στα περιεχόμενά της. Για παράδειγμα, στο Σχήμα 6.5, η τιμή της μεταβλητής POINT1 (η οποία φυλάσσεται στη θέση μνήμης 12365) είναι μια άλλη διεύθυνση μνήμης, η 12366. Εκεί φυλάσσεται μια άλλη τιμή, η 42, στην οποία εμείς μπορούμε να έχουμε έμμεση προσπέλαση μέσω της POINT1

ΟΝΟΜΑ ΜΕΤΑΒΛΗΤΗΣ		ΔΙΕΥΘΥΝΣΗ ΜΝΗΜΗΣ	ΤΙΜΗ
MAX	Συσχετίζεται άμεσα με τη θέση	12364	145
POINT1	Συσχετίζεται άμεσα με τη θέση	12365	12366
	Συσχετίζεται έμμεσα με τη θέση	12366	42
POINT2	Συσχετίζεται άμεσα με τη θέση	12367	NIL
	Συσχετίζεται έμμεσα με τη θέση	??	

Σχήμα 6.5

Αναπαράσταση της συσχέτισης μεταβλητών με θέσεις μνήμης

Συνεπώς, οι δείκτες μπορούν να χρησιμοποιηθούν με δύο τρόπους:

- Ως μηχανισμοί έμμεσης προσπέλασης (dereferencing) στην τιμή της θέσης μνήμης στην οποία δείχνουν, μετά από ειδική επεξεργασία της τιμής του δείκτη. Για να δηλωθεί αυτή η επεξεργασία χρησιμοποιείται ο τελεστής ^. Για παράδειγμα, η έκφραση B^ := 10 σημαίνει ότι στη θέση μνήμης όπου δείχνει ο B καταχωρείται έμμεσα η τιμή 10.
- Ως συνηθισμένες μεταβλητές των οποίων η τιμή είναι κάποια θέση μνήμης. Για παράδειγμα, εάν A και B είναι δείκτες, τότε η έκφραση A := B σημαίνει ότι ο δείκτης A δείχνει στην ίδια θέση μνήμης με τον δείκτη B (βέβαια, η δυνατότητα προσπέλασης της θέσης μνήμης στην οποία έδειχνε ο A πριν την καταχώρηση – εάν υπήρχε τέτοια – έχει για πάντα χαθεί).

Τι συμβαίνει όμως με τη μεταβλητή δείκτη POINT2 στο Σχήμα 6.5; Είναι σωστός ή λάθος ο τρόπος ορισμού της; Για εσάς που αναρωτιέστε, ορίστε η απάντηση: Οι περισσότερες γλώσσες προγραμματισμού μας επιτρέπουν να δηλώσουμε μεταβλητές δείκτη χωρίς να καταχωρήσουμε αμέσως κάποια τιμή σε αυτές. Τότε, λέμε κατά σύμβαση ότι η τιμή του δείκτη είναι NIL, που ουσιαστικά σημαίνει ότι ο δείκτης δε δείχνει πουθενά! Προσοχή όμως: οποιαδήποτε εντολή της μορφής X := POINT2^ είναι λανθασμένη, αφού, σύμφωνα με τα προηγούμενα, δεν υπάρχει κάποια θέση μνήμης την οποία να προσπελαύνει έμμεσα ο POINT2.

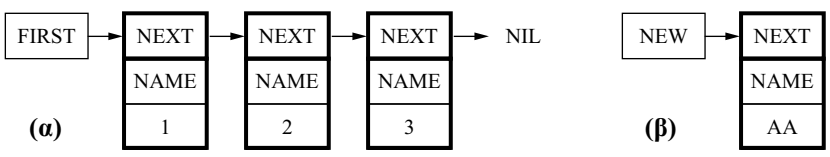
Άσκηση αυτοαξιολόγησης 6.2

Αναφερόμενοι στο Σχήμα 6.5 προσπαθήστε να συσχετίσετε τις οδηγίες ενός προγράμματος που χρησιμοποιεί τις μεταβλητές MAX και POINT1 (δείχνονται στην αριστερή στήλη) με τα εξαγόμενα στη δεξιά στήλη.

ΤΥΠΩΣΕ(MAX)	12366
ΤΥΠΩΣΕ(MAX^)	
ΤΥΠΩΣΕ(POINT1)	42
ΤΥΠΩΣΕ(POINT1^)	
POINT1 := MAX ΤΥΠΩΣΕ(POINT1)	
	ΛΑΘΟΣ ΟΔΗΓΙΑ
POINT1^ := MAX ΤΥΠΩΣΕ(POINT1)	
POINT1^ := MAX ΤΥΠΩΣΕ(POINT1^)	145

6.5.2 Διασυνδεδεμένες λίστες

Η πιο γενική δυναμική δομή δεδομένων είναι η **διασυνδεδεμένη λίστα (linked list).** Μια λίστα, όπως φαίνεται στο Σχήμα 6.6(α), είναι ένα σύνολο στοιχείων (λέγονται και κόμβοι). Κάθε στοιχείο αποτελείται από ένα σύνολο μεταβλητών. Σε κάποιες από αυτές αποθηκεύονται τα δεδομένα της θέσης, ενώ μία μεταβλητή είναι τύπου δείκτη και λειτουργεί ως σύνδεσμος που «δείχνει» στο επόμενο στοιχείο της λίστας (ο δείκτης του τελευταίου κόμβου έχει κατά σύμβαση την τιμή NIL).



Σχήμα 6.6
Μια διασυνδεδεμένη λίστα

Η προσπέλαση των δεδομένων μιας λίστας δεν γίνεται πια με βάση τη θέση τους, αφού η λίστα είναι δυναμική. Αντίθετα, υπάρχει ένας ακόμη δείκτης, ο οποίος δείχνει πάντα το πρώτο στοιχείο της λίστας. Αρχίζοντας από αυτόν και ακολουθώντας τους συνδέσμους, μπορούμε να φτάσουμε σε όποιο κόμβο θέλουμε.

- Οι διασυνδεδεμένες λίστες επιτρέπουν μεν τη δυναμική πρόσθεση ή διαγραφή κόμβων, οπότε τα όρια τους μπορεί να αλλάζουν κατά τη διάρκεια της εκτέλεσης του προγράμματος, έχουν όμως το μειονέκτημα ότι η προσπέλαση στα στοιχεία τους είναι ακολουθιακή.

Στα παραδείγματα που ακολουθούν θα περιγράψουμε τρόπους δημιουργίας μιας λίστας, πρόσθεσης και διαγραφής ενός κόμβου από μια λίστα.

Παράδειγμα 6.4

Δημιουργία μιας διασυνδεδεμένης λίστας

Έστω, λοιπόν, ότι ο Βύρων ανέλαβε να φτιάξει το πρόγραμμα που θα χρησιμοποιεί η Ελένη για να διατηρεί τα ονόματα των συνεργατών της. Αφού μελέτησε τα πιθανά προβλήματα με τους πίνακες, ο Βύρων αποφάσισε να χρησιμοποιήσει τη διασυνδεδεμένη λίστα PARTNER ως δομή αποθήκευσης των δεδομένων. Αρχικά, ο Βύρων δοκιμάζει να δημιουργήσει μια λίστα που περιέχει 10 κόμβους, όπου θα καταχωρηθούν τα στοιχεία δέκα συνεργατών. Κάθε κόμβος περιλαμβάνει τον κωδικό (μεταβλητή ΑΑ) και το όνομα του συνεργάτη (μεταβλητή NAME), καθώς και τον σύνδεσμο με τον επόμενο κόμβο (μεταβλητή δείκτη NEXT). Επειδή ο κόμβος ουσιαστικά αποτελεί μια νέα σύνθετη δομή δεδομένων, την ορίζει πρώτα στο τμήμα ΤΥΠΟΙ.

ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑ-ΛΙΣΤΑΣ

```
ΤΥΠΟΙ
        ΚΟΜΒΟΣ:     ΑΑ: INTEGER ;
                    NAME: CHAR ;
                    NEXT: POINTER[ΚΟΜΒΟΣ] ;
ΔΕΔΟΜΕΝΑ
        PARTNER: LIST OF ΚΟΜΒΟΣ ;
        CURRENT, FIRST, NEW: POINTER[ΚΟΜΒΟΣ] ;
ΑΡΧΗ
```

```
ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
NEW^.ΑΑ := 1 ;
NEW^.NEXT := NIL ;
FIRST := NEW ;
CURRENT := FIRST ;
ΓΙΑ Ι := 2 ΕΩΣ 10 ΕΠΑΝΕΛΑΒΕ
        ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
        NEW^.ΑΑ := Ι ;
        NEW^.NEXT := NIL
        CURRENT^.NEXT := NEW ;
        CURRENT := NEW;
ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Στην αρχή, δημιουργούμε το πρώτο στοιχείο της λίστας, στο οποίο δείχνει ο δείκτης FIRST. Για να δημιουργήσουμε ένα νέο στοιχείο, χρησιμοποιούμε ένα τμήμα κώδικα (ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW)) το οποίο δημιουργεί ένα νέο, ανεξάρτητο κόμβο και μας επιστρέφει ένα δείκτη σε αυτόν, τον NEW. Σημειώστε, ότι ο νέος κόμβος πρέπει να είναι του ίδιου τύπου με τους άλλους που ήδη περιέχει η λίστα PARTNER, γι' αυτό και ο δείκτης NEW ορίζεται ως τύπου POINTER[ΚΟΜΒΟΣ].

Έπειτα, μέσα από μια δομή επανάληψης εννέα βημάτων, δημιουργούμε διαδοχικά τους υπόλοιπους κόμβους της λίστας. Κάθε νέος κόμβος προστίθεται στο τέλος της υπάρχουσας λίστας. Στο Σχήμα 6.6 φαίνεται ένα στιγμιότυπο της διαδικασίας.

Για το πέρασμα της λίστας χρησιμοποιείται ο δείκτης CURRENT, ο οποίος αρχικά δείχνει στο πρώτο στοιχείο της λίστας (δηλαδή, εκεί όπου δείχνει και ο FIRST), ενώ στη συνέχεια, δείχνει στο τελευταίο κάθε φορά στοιχείο. Σε κάθε βήμα της δομής επανάληψης (ο συμβολισμός ΔΕΙΚΤΗΣ^.ΜΕΤΑΒΛΗΤΗ αναφέρεται στην τιμή της ΜΕΤΑΒΛΗΤΗΣ του κόμβου όπου δείχνει ο ΔΕΙΚΤΗΣ):

- Δίνουμε την τιμή του μετρητή στη μεταβλητή ΑΑ του νέου κόμβου (NEW^.ΑΑ := Ι).
- Δίνουμε την τιμή NIL στο σύνδεσμο του νέου κόμβου

(NEW^.NEXT := NIL).

- Κάνουμε το σύνδεσμο του τελευταίου στοιχείου της υπάρχουσας λίστας να δείχνει στο νέο στοιχείο (CURRENT^.NEXT := NEW)
- Κάνουμε το δείκτη CURRENT να δείχνει στο νέο (και μέχρι τώρα τελευταίο) στοιχείο της λίστας (CURRENT := NEW). Εδώ, δε χρειάζεται ο τελεστής έμμεσης προσπέλασης, αφού πρόκειται για καταχώρηση της τιμής ενός δείκτη σε έναν άλλο δείκτη.

Σημειώστε πως χρησιμοποιούμε ένα νέο δείκτη (CURRENT) για να διαπεράσουμε τη λίστα, γιατί εάν μετακινήσουμε τον δείκτη FIRST, τότε θα «χάσουμε» τον αρχικό κόμβο της λίστας.

Για να καταχωρήσει ο Βύρων το όνομα «Απόστολος Καμέας» στον τρίτο κόμβο της λίστας, πρέπει τώρα να χρησιμοποιήσει την εντολή:

FIRST^.NEXT^.NEXT^.NAME := «Απόστολος Καμέας»

Ο δείκτης FIRST είπαμε ότι δείχνει πάντα στο πρώτο στοιχείο της λίστας. Με τον τρόπο αυτό προσπελαύνουμε διαδοχικά τους κόμβους, μέχρι να φτάσουμε στη μεταβλητή NAME του τρίτου κόμβου (στην οποία ουσιαστικά φτάνουμε μετά από έμμεση προσπέλαση του δείκτη του δεύτερου κόμβου). Εάν έχετε πρόβλημα στην κατανόηση, δοκιμάστε να «οπτικοποιήσετε» τη διαδικασία ζωγραφίζοντας τη λίστα και τους δείκτες σε ένα χαρτί!

Παράδειγμα 6.5

Πρόσθεση ενός νέου κόμβου στο τέλος μιας λίστας

Έχουμε λοιπόν, δημιουργήσει τη λίστα PARTNER, η οποία περιλαμβάνει ένα κόμβο για κάθε συνεργάτη. Ας δούμε τώρα τον αλγόριθμο που προσθέτει ένα νέο κόμβο στο τέλος της λίστας:

ΑΛΓΟΡΙΘΜΟΣ ΠΡΟΣΘΕΣΗ-ΚΟΜΒΟΥ-ΣΤΟ-ΤΕΛΟΣ

ΤΥΠΟΙ

```
ΚΟΜΒΟΣ:    ΑΑ: INTEGER ;
           NAME: CHAR ;
           NEXT: POINTER[ΚΟΜΒΟΣ] ;
```

ΔΕΔΟΜΕΝΑ

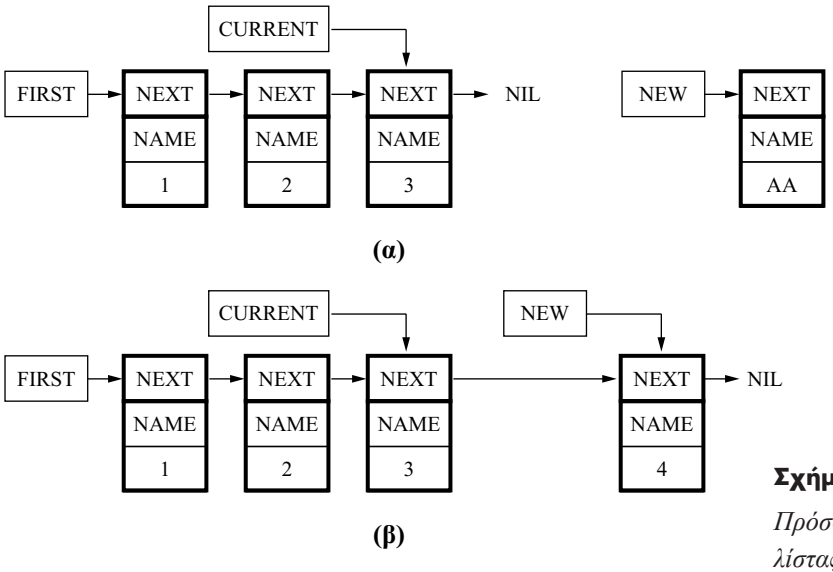
```
PARTNER: LIST OF ΚΟΜΒΟΣ ;
```

```
        CURRENT, FIRST, NEW: POINTER[ΚΟΜΒΟΣ] ;
ΑΡΧΗ
        ΕΑΝ (FIRST < > NIL) ΤΟΤΕ
                CURRENT := FIRST ;
                ΕΝΟΣΩ (CURRENT^.NEXT <> NIL) ΕΠΑΝΕΛΑΒΕ
                        CURRENT := CURRENT^.NEXT
                ΕΝΟΣΩ-ΤΕΛΟΣ ;
                ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
                NEW^.NEXT := NIL ;
                CURRENT^.NEXT := NEW
        ΑΛΛΙΩΣ
                ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
                NEW^.NEXT := NIL ;
                FIRST := NEW ;
                CURRENT := FIRST ;
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Ο δείκτης FIRST δείχνει στην αρχή της λίστας PARTNER. Ο αλγόριθμος διατρέχει τη λίστα μέχρι να φτάσει στον τελευταίο κόμβο της (του οποίου η τιμή της μεταβλητής δείκτη θα είναι NIL), όπου και προσθέτει το νέο κόμβο. Σημειώστε ότι εάν η λίστα είναι κενή (οπότε FIRST = NIL) ο αλγόριθμος αναγκάζεται ουσιαστικά να τη δημιουργήσει, προσθέτοντας τον πρώτο της κόμβο.

Σε κάθε βήμα της δομής επανάληψης, ο CURRENT μετακινείται ένα κόμβο προς το τέλος της λίστας και δείχνει στο στοιχείο όπου δείχνει και ο σύνδεσμος του τρέχοντος στοιχείου. Αυτό γίνεται με την οδηγία CURRENT := CURRENT^.NEXT, η οποία ερμηνεύεται ως εξής: η νέα τιμή του δείκτη CURRENT είναι η τιμή του δείκτη NEXT του κόμβου όπου δείχνει τώρα ο CURRENT (πρόκειται για έμμεση προσπέλαση του CURRENT), δηλαδή ο επόμενος του τρέχοντος κόμβου!

Η προώθηση του δείκτη CURRENT συνεχίζεται μέχρι το στοιχείο του οποίου ο σύνδεσμος δείχνει στο τίποτα (CURRENT^.NEXT = NIL), δηλαδή μέχρι το τελευταίο στοιχείο της λίστας.



Σχήμα 6.7.
Πρόσθεση κόμβου στο τέλος μιας λίστας

Στο σημείο αυτό [Σχήμα 6.7(α)] έχουμε μία λίστα, στην αρχή της οποίας δείχνει ο FIRST και στο τελευταίο στοιχείο της ο CURRENT και ένα ανεξάρτητο κόμβο στον οποίο δείχνει ο NEW. Για να προσθέσουμε τον κόμβο αυτόν στο τέλος της λίστας, αρκεί να «παίξουμε» λίγο με τους δείκτες. Έτσι, κάνουμε το σύνδεσμο του νέου στοιχείου της λίστας να δείχνει στο πουθενά (NEW^.NEXT := NIL) και το σύνδεσμο του τελευταίου στοιχείου της λίστας να δείχνει (από το τίποτα) στο νέο κόμβο (CURRENT^.NEXT := NEW).

Σημειώστε ότι, μόλις συνδεθεί το νέο στοιχείο, ο CURRENT θα δείχνει πια στο προ-τελευταίο στοιχείο της λίστας [Σχήμα 6.7(β)], οπότε CURRENT^.NEXT είναι ο κόμβος στον οποίο δείχνει ο σύνδεσμος του προ-τελευταίου κόμβου (δηλαδή ο τελευταίος κόμβος της λίστας), και CURRENT^.NEXT^.NEXT είναι ο κόμβος στον οποίο δείχνει ο σύνδεσμος του κόμβου στον οποίο δείχνει ο σύνδεσμος του προ–τελευταίου κόμβου (δεν υπάρχει τέτοιος κόμβος, γι' αυτό και η τιμή αυτού του συνδέσμου είναι NIL).

Παράδειγμα 6.6

Παρεμβολή ενός νέου κόμβου στη λίστα

Ας δούμε τώρα έναν αλγόριθμο που παρεμβάλλει στη λίστα ένα νέο κόμβο για τον αντιπρόσωπο με το όνομα "Νικόλαος Καρκαντός", μετά τον κόμβο του αντιπροσώπου "Απόστολος Καμέας" (υποθέτουμε ότι αυτός ο κόμβος υπάρχει στη λίστα, αλλιώς ο νέος κόμβος θα προστεθεί στο τέλος της λίστας):

ΑΛΓΟΡΙΘΜΟΣ ΠΑΡΕΜΒΟΛΗ-ΚΟΜΒΟΥ

```
ΤΥΠΟΙ
        ΚΟΜΒΟΣ:     ΑΑ: INTEGER ;
                    NAME: CHAR ;
                    NEXT: POINTER[ΚΟΜΒΟΣ] ;
ΔΕΔΟΜΕΝΑ
        PARTNER: LIST OF ΚΟΜΒΟΣ ;
        CURRENT, FIRST, NEW: POINTER[ΚΟΜΒΟΣ] ;
ΑΡΧΗ
        ΕΑΝ (FIRST < > NIL) ΤΟΤΕ
                CURRENT := FIRST ;
                ΕΝΟΣΩ ((CURRENT^.NAME <> "Απόστολος Καμέας") AND
                (CURRENT^.NEXT <> NIL)) ΕΠΑΝΕΛΑΒΕ
                        CURRENT := CURRENT^.NEXT
                ΕΝΟΣΩ-ΤΕΛΟΣ ;
                ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
                NEW^.NAME := "Νικόλαος Καρκαντός" ;
                NEW^.NEXT := CURRENT^.NEXT ;
                CURRENT^.NEXT := NEW ;
        ΑΛΛΙΩΣ
                ΥΠΟΛΟΓΙΣΕ ΔΗΜΙΟΥΡΓΗΣΕ-ΚΟΜΒΟ(NEW) ;
                NEW^.NAME := "Νικόλαος Καρκαντός" ;
                NEW^.NEXT := NIL ;
                FIRST := NEW ;
                CURRENT := FIRST ;
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Όπως παρατηρείτε, ο αλγόριθμος αυτός μοιάζει πολύ με τον προηγούμενο, καθώς χειρίζεται τους δείκτες και τους συνδέσμους με τον ίδιο τρόπο όπως πριν: Κάνουμε το σύνδεσμο του νέου στοιχείου να δείχνει στον επόμενο από το σημείο παρεμβολής κόμβο, και το σύνδεσμο του προηγούμενου από το σημείο παρεμβολής κόμβου να δείχνει στο νέο στοιχείο. Όπως και πριν, εάν η λίστα είναι κενή, ο αλγόριθμος ουσιαστικά προσθέτει τον πρώτο της κόμβο.

Παράδειγμα 6.7

Διαγραφή ενός κόμβου από τη λίστα

Η διαγραφή ενός κόμβου από τη λίστα των συνεργατών είναι κάπως πιο περίπλοκη. Όπως θα έχετε ήδη υποθέσει, αυτό σημαίνει ότι πρέπει να κάνουμε το σύνδεσμο του προηγούμενου από τον προς διαγραφή κόμβο να δείχνει στον επόμενο από τον προς διαγραφή κόμβο.

Έστω ότι θέλουμε τώρα να διαγράψουμε τον κόμβο του συνεργάτη "Απόστολος Καμέας". Εάν χρησιμοποιήσουμε την ίδια δομή επανάληψης για να εντοπίσουμε τον προς διαγραφή κόμβο, τότε, πραγματικά, στην έξοδό της ο CURRENT θα δείχνει στον προς διαγραφή κόμβο, ή, σε περίπτωση που αυτός δε βρεθεί, στο τελευταίο στοιχείο της λίστας (οπότε δεν υπάρχει κάτι να διαγράψουμε). Το πρόβλημα τώρα είναι ότι πρέπει να «γυρίσουμε» το δείκτη μια θέση πίσω για να αλλάξουμε το σύνδεσμο του προηγούμενου στοιχείου, κάτι που δεν είναι δυνατό στις (απλά) διασυνδεδεμένες λίστες.

Ένας τρόπος να ξεπεράσουμε το πρόβλημα είναι να κάνουμε το δείκτη να ψάχνει έναν κόμβο πιο κάτω από αυτόν στον οποίο δείχνει (CURRENT^.NEXT^.NAME). Σε τέτοια περίπτωση, πρέπει να μεταχειριστούμε ειδικά τον πρώτο κόμβο. Ακόμη, παρατηρήστε ότι εάν η λίστα είναι κενή, ο αλγόριθμος τυπώνει ένα μήνυμα και τερματίζει, χωρίς περαιτέρω επεξεργασία. Τέλος, πρέπει να αναφερθεί ότι οι γλώσσες προγραμματισμού διαθέτουν ειδικές εντολές με τις οποίες «αποδίδεται» πίσω στο σύστημα ο χώρος μνήμης που καταλαμβάνει ένας κόμβος, όταν αυτός διαγραφεί.

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗ-ΚΟΜΒΟΥ

```
ΤΥΠΟΙ
        ΚΟΜΒΟΣ:     ΑΑ: INTEGER ;
                    NAME: CHAR ;
                    NEXT: POINTER[ΚΟΜΒΟΣ] ;
ΔΕΔΟΜΕΝΑ
        PARTNER: LIST OF ΚΟΜΒΟΣ ;
        CURRENT, FIRST, NEW: POINTER[ΚΟΜΒΟΣ] ;
ΑΡΧΗ
        ΕΑΝ (FIRST = NIL) ΤΟΤΕ
                ΤΥΠΩΣΕ(«ΚΕΝΗ ΛΙΣΤΑ»)
        ΑΛΛΙΩΣ ΕΑΝ (FIRST^.NAME = "Απόστολος Καμέας") ΤΟΤΕ
```

```
                FIRST := FIRST^.NEXT
            ΑΛΛΙΩΣ
                CURRENT := FIRST ;
                ΕΝΟΣΩ ((CURRENT^.NEXT^.NAME <> "Απόστολος Καμέας") AND
                (CURRENT^.NEXT^.NEXT <> NIL)) ΕΠΑΝΕΛΑΒΕ
                    CURRENT := CURRENT^.NEXT
                ΕΝΟΣΩ-ΤΕΛΟΣ ;
                ΕΑΝ (CURRENT^.NEXT^.NAME = "Απόστολος Καμέας")
                ΤΟΤΕ
                    CURRENT^.NEXT := CURRENT^.NEXT^.NEXT
                ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Άλλες εργασίες που μπορούμε να εκτελέσουμε σε μια λίστα είναι η συνένωση δύο λιστών σε μία, η διάσπαση μιας λίστας σε δύο, η δημιουργία αντιγράφου της λίστας, η ταξινόμηση των κόμβων της λίστας με βάση το τμήμα δεδομένων, κ.ά.

Όπως είδαμε, οι απλά διασυνδεδεμένες λίστες μπορεί να γίνουν δύσχρηστες επειδή δεν επιτρέπουν τη μετακίνηση του δείκτη στο προηγούμενο από το τρέχον στοιχείο. Το πρόβλημα αυτό λύνεται εάν προσθέσουμε και τη δυνατότητα των «προς-τα-πίσω» συνδέσμων, οπότε έχουμε τις διπλά διασυνδεδεμένες λίστες (doubly linked lists), στις οποίες κάθε κόμβος περιλαμβάνει δύο τμήματα συνδέσμων, τα PREV και NEXT (το πρώτο δείχνει στον προηγούμενο και το δεύτερο στον επόμενο κόμβο).

Δραστηριότητα 6.7

Πώς τροποποιείται ο αλγόριθμος διαγραφής ενός στοιχείου όταν έχουμε διπλά διασυνδεδεμένη λίστα;

Στην περίπτωση της διπλά διασυνδεδεμένης λίστας, δε χρειάζεται να κάνουμε «πρόβλεψη» του μεθεπόμενου στοιχείου, ούτε να μεταχειριστούμε με ειδικό τρόπο το πρώτο στοιχείο, αφού μπορούμε ανά πάσα

στιγμή να «βρούμε» το προηγούμενο στοιχείο από αυτό που δείχνει ο CURRENT διά μέσου του δείκτη PREV.

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗ-ΚΟΜΒΟΥ-2

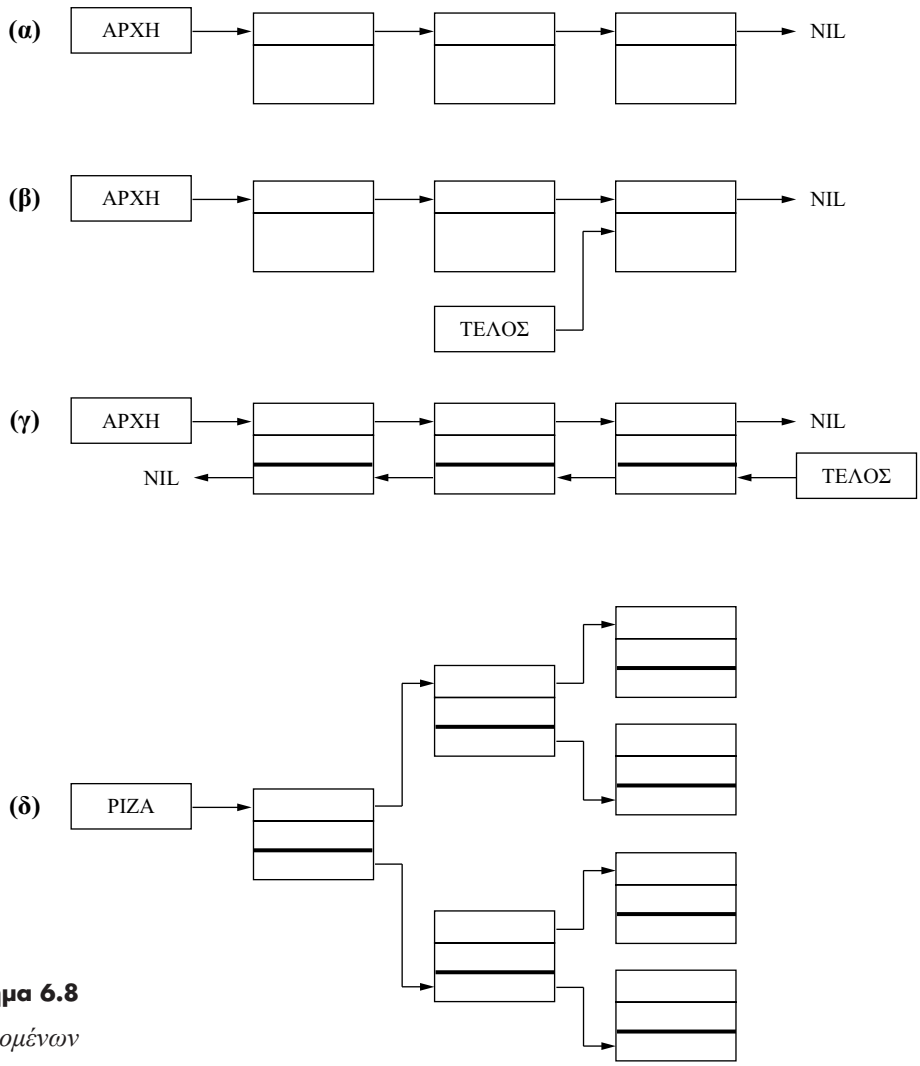
```
ΤΥΠΟΙ
        ΚΟΜΒΟΣ1:    ΑΑ: INTEGER ;
                    NAME: CHAR ;
                    NEXT: POINTER[ΚΟΜΒΟΣ1] ;
                    PREV: POINTER[ΚΟΜΒΟΣ1] ;
ΔΕΔΟΜΕΝΑ
        PARTNER: LIST OF ΚΟΜΒΟΣ1 ;
        CURRENT, FIRST: POINTER[ΚΟΜΒΟΣ1] ;
ΑΡΧΗ
        CURRENT := FIRST ;
        ΕΝΟΣΩ ((CURRENT^.NAME <> "Απόστολος Καμέας") AND (CURRENT^.NEXT <> NIL))
        ΕΠΑΝΕΛΑΒΕ
                CURRENT := CURRENT^.NEXT
        ΕΝΟΣΩ-ΤΕΛΟΣ ;
        ΕΑΝ (CURRENT^.NAME = " Απόστολος Καμέας") ΤΟΤΕ
                CURRENT^.NEXT^.PREV := CURRENT^.PREV ;
                CURRENT^.PREV^.NEXT := CURRENT^.NEXT
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

6.5.3 Άλλες δυναμικές δομές δεδομένων

Οι διασυνδεδεμένες λίστες είναι μια πολύ γενική δομή δεδομένων, όλες οι δυνατότητες της οποίας πολύ σπάνια χρειάζονται για την αλγοριθμική λύση ενός προβλήματος. Έτσι, ανάλογα με τις λειτουργίες που χρησιμοποιούνται και τον τρόπο δημιουργίας, ορίζονται κάποιες πιο απλές δυναμικές δομές δεδομένων, όπως:

- **Η στοίβα (stack):** Πρόκειται για μια λίστα στην οποία όλες οι προσθέσεις ή οι διαγραφές κόμβων γίνονται πάντα στο ένα άκρο της μόνο [δείκτης ΑΡΧΗ στο Σχήμα 6.8(α)]. Δομή στοίβας χρησιμοποιούμε όταν στοιβάζουμε τα βιβλία μας ή μερικά κιβώτια το ένα πάνω στο άλλο.

- **Η ουρά (queue):** Πρόκειται για μια λίστα στην οποία όλες οι προσθέσεις κόμβων γίνονται πάντα στο ένα άκρο της [δείκτης ΤΕΛΟΣ στο Σχήμα 6.8(β)] και όλες οι διαγραφές κόμβων γίνονται πάντα στο άλλο άκρο της [δείκτης ΑΡΧΗ στο Σχήμα 6.8(β)]. Όλοι γνωρίζουμε τις ουρές που σχηματίζονται στις Τράπεζες, στις Εφορίες κ.ά.
- **Η διπλή ουρά (deque):** Πρόκειται για μια λίστα στην οποία όλες οι προσθέσεις ή οι διαγραφές κόμβων γίνονται πάντα στα άκρα της μόνο [δείκτες ΑΡΧΗ και ΤΕΛΟΣ στο Σχήμα 6.8(γ)]. Τέτοια δομή χρησιμοποιούμε π.χ., για να κατασκευάσουμε ένα κολιέ από μαρ-



Σχήμα 6.8

Δυναμικές δομές δεδομένων

γαριτάρια, αφού μπορούμε να προσθέσουμε ή να αφαιρέσουμε ένα μαργαριτάρι από οποιαδήποτε από τις δύο άκρες του σχοινιού.

- **Το δένδρο (tree):** Πρόκειται για μια λίστα στην οποία ο σύνδεσμος κάθε κόμβου δείχνει σε μηδέν ή περισσότερους κόμβους (παιδιά), με τον περιορισμό ότι σε κάθε κόμβο δείχνει μόνο ένας άλλος κόμβος (δηλαδή κάθε κόμβος έχει μόνο έναν γονέα). Αυτή η δομή είναι σχετικά πολύπλοκη και χρησιμοποιείται π.χ., στην αναπαράσταση της δομής καταλόγων και αρχείων του σκληρού δίσκου του υπολογιστή μας. Στο Σχήμα 6.8(δ) φαίνεται ένα δυαδικό δένδρο, στο οποίο κάθε κόμβος έχει υποχρεωτικά το πολύ δύο παιδιά.

6.5.4 Πρόληψη σφαλμάτων

Τα πιο συνηθισμένα λάθη, όταν προγραμματίζουμε με δείκτες (δηλαδή όταν χρησιμοποιούμε δυναμικές δομές δεδομένων) είναι δύο ειδών: Η χρήση δεικτών που δεν έχουν αρχικοποιηθεί και ο λάθος χειρισμός των ορίων της λίστας.

Στην πρώτη περίπτωση, πρέπει να θυμάστε ότι ένας δείκτης πρέπει πάντα να δείχνει σε μια θέση ή ένα κόμβο, αλλιώς λέμε ότι ο δείκτης είναι αόριστος (undefined). Θυμηθείτε την περίπτωση δημιουργίας ενός νέου κόμβου, όπου χρησιμοποιείται ο δείκτης NEW για να δείχνει σε αυτόν.

Ο λάθος χειρισμός των ορίων της λίστας μπορεί να οδηγήσει σε λίστες με «άπειρο» πλήθος κόμβων ή σε ατέρμονες αναζητήσεις στοιχείων. Η πρώτη περίπτωση μπορεί να εμφανιστεί όταν ξεχάσουμε να θέσουμε κάποια συνθήκη τερματισμού της δομής επανάληψης που δημιουργεί μια λίστα. Σε τέτοια περίπτωση, επειδή στην πραγματικότητα κάθε νέος κόμβος είναι ένα τμήμα της μνήμης του υπολογιστή, μόλις η διαθέσιμη μνήμη τελειώσει, θα προκληθεί σφάλμα εκτέλεσης του προγράμματος.

Σφάλματα της δεύτερης κατηγορίας συμβαίνουν όταν δεν υπάρχει κόμβος που να ικανοποιεί τα κριτήρια αναζήτησης και έχουμε παραλείψει να θέσουμε κάποια άλλη συνθήκη τερματισμού της αναζήτησης. Τέτοια συνθήκη είναι η CURRENT^.NEXT <> NIL που χρησιμοποιείται μαζί με τη συνθήκη αναζήτησης στα παραδείγματα 6 και 7.

Άλλα σημεία που πρέπει να προσέχουμε όταν προγραμματίζουμε δυναμικές δομές δεδομένων είναι τα εξής:

- Να μη βγάζουμε στοιχεία από μια άδεια λίστα ή στοίβα.
- Να μη «χάνουμε» κόμβους (π.χ., μετακινώντας το δείκτη FIRST).
- Να μη δημιουργούμε κυκλικές λίστες.
- Να μην αναζητούμε την τιμή ενός NIL pointer.
- Να μη διαγράφουμε τον κόμβο όπου δείχνει ένας δείκτης χωρίς να διαγράψουμε και το δείκτη (σε τέτοια περίπτωση, ο δείκτης λέμε ότι «βρίσκεται σε εκκρεμότητα» - dangling pointer).
- Να εξετάζουμε καλά τις οριακές καταστάσεις, όπως
  - εάν ο αλγόριθμος εφαρμόζεται σωστά στον πρώτο και τον τελευταίο κόμβο,
  - εάν ο αλγόριθμος ισχύει για μια λίστα που είναι ή γίνεται κενή,
  - εάν αναφερόμαστε στον κόμβο που δείχνει ο σύνδεσμος του τελευταίου κόμβου (κάτι που δεν επιτρέπεται).

Στις δυναμικές δομές δεδομένων δεν είναι δυνατή η εκτύπωση ενός στιγμιότυπου των τιμών των μεταβλητών του προγράμματος (η οποία αποτελεί την αποτελεσματικότερη τεχνική εκσφαλμάτωσης), αφού η τιμή μιας μεταβλητής δείκτη είτε θα είναι NIL είτε θα είναι κάποια διεύθυνση μνήμης.

Συνεπώς, η καλύτερη τεχνική αμυντικού προγραμματισμού που εφαρμόζεται στις λίστες είναι η εκτύπωση ολόκληρης της δομής δεδομένων, χρησιμοποιώντας ένα τμήμα κώδικα σαν το ακόλουθο:

```
CURRENT := FIRST ;
I := 0 ;
ΕΝΟΣΩ (CURRENT < > NIL) ΕΠΑΝΕΛΑΒΕ
        I := I+1 ;
        ΤΥΠΩΣΕ("Ο ΚΟΜΒΟΣ", I, "ΠΕΡΙΛΑΜΒΑΝΕΙ",
        CURRENT^.ΜΕΤΑΒΛΗΤΕΣ-ΔΕΔΟΜΕΝΩΝ) ;
        CURRENT := CURRENT^.NEXT
ΕΝΟΣΩ-ΤΕΛΟΣ
```

6.6 Μπορώ να παραβώ τους κανόνες;

Αν και η χρήση προγραμματιστικών δομών που έχουν μια είσοδο και μια έξοδο γενικά βελτιώνει την ευκρίνεια των προγραμμάτων, συχνά εμφανίζονται περιπτώσεις όπου η ευκρίνεια του προγράμματος βελτιώνεται εάν παραβιαστεί αυτός ο κανόνας!

Δύο κοινές τέτοιες περιπτώσεις είναι η έξοδος από φωλιασμένες δομές επανάληψης και η διαχείριση σφαλμάτων (όταν π.χ., υιοθετείται αμυντικό στυλ προγραμματισμού). Στις περιπτώσεις αυτές χρησιμοποιούνται εντολές που μεταφέρουν αυθαίρετα τον έλεγχο της εκτέλεσης του προγράμματος σε μια καθορισμένη εντολή.

Η περισσότερο διαδεδομένη τέτοια εντολή είναι η **GOTO**. Είναι τόσοι οι αφορισμοί και άλλα τόσα τα υπέρ της επιχειρήματα, που αυτή η εντολή αποτελεί «μύθο» του προγραμματισμού (γι' αυτό προτιμώ στο βιβλίο να χρησιμοποιηθεί χωρίς μετάφραση!). Αν και θεωρείται υπεύθυνη για όλα τα ελαττώματα των προγραμμάτων (αλλά όχι των προγραμματιστών), εν τούτοις συμπεριλαμβάνεται στο λεξιλόγιο όλων των διαδεδομένων γλωσσών προγραμματισμού (και βρίσκεται στην άκρη του μυαλού όλων των προγραμματιστών).

- Σαν γενικό κανόνα εφαρμογής της GOTO, να θυμάστε ότι:
  - επιτρέπεται η χρήση GOTO που διακλαδώνουν τον έλεγχο προς-τα-εμπρός, ώστε να τερματιστεί κάποια άλλη τοπική δομή, ή που χρησιμοποιούνται για τοπική διαχείριση λαθών,
  - απαγορεύεται η χρήση GOTO που μεταφέρουν τον έλεγχο σε απομακρυσμένα τμήματα κώδικα, ή μέσα και έξω από δομές προγραμματισμού, καταστρέφοντας τη γραμμικότητα και την τοπικότητα της ροής ελέγχου και εκτέλεσης

Παράδειγμα 6.8

Έστω ότι μετράμε το ύψος της βροχής στην πόλη μας ακριβώς τα μεσάνυχτα κάθε ημέρα του έτους και διατηρούμε τα στοιχεία αυτά σε ένα πίνακα 52×7 (δηλαδή σε κάθε γραμμή του πίνακα γράφουμε τα στοιχεία που συλλέγουμε για όλες τις ημέρες μιας εβδομάδας). Στο τέλος κάθε έτους χρησιμοποιούμε τον αλγόριθμο που ακολουθεί για

να βρούμε το ελάχιστο ύψος βροχής του έτους και την ημέρα κατά την οποία αυτό κατεγράφη.

ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟ-ΥΨΟΣ-ΒΡΟΧΗΣ

```
ΔΕΔΟΜΕΝΑ
        WEEK, DAY, I, K: INTEGER;
        RAIN: ARRAY [1..52, 1..7] OF INTEGER;
ΑΡΧΗ
        WEEK := 1 ;
        DAY := 1 ;
        ΓΙΑ I := 1 ΕΩΣ 52 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := 1 ΕΩΣ 7 ΕΠΑΝΕΛΑΒΕ
                        ΕΑΝ (RAIN[I,K] < RAIN[WEEK,DAY]) ΤΟΤΕ
                                WEEK := I ;
                                DAY := K ;
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
        ΤΥΠΩΣΕ("ΤΟ ΕΛΑΧΙΣΤΟ ΥΨΟΣ ΒΡΟΧΗΣ ΤΟΥ ΕΤΟΥΣ ΗΤΑΝ", RAIN[WEEK,DAY], "ΕΚΑΤΟΣΤΑ ΚΑΙ ΣΗΜΕΙΩΘΗΚΕ ΓΙΑ ΠΡΩΤΗ ΦΟΡΑ ΤΗΝ ΗΜΕΡΑ", DAY, "ΤΗΣ ΕΒΔΟΜΑΔΑΣ", WEEK)
ΤΕΛΟΣ
```

Ο αλγόριθμος αυτός θα εκτελέσει 52*7 = 364 επαναλήψεις πριν βρει το ελάχιστο ύψος βροχής. Υπάρχει όμως μεγάλη πιθανότητα να είναι άσκοπες όλες αυτές οι επαναλήψεις. Γιατί; Γιατί, είναι πολύ πιθανό για αρκετές ημέρες το ύψος της βροχής να είναι 0, δηλαδή να μην έχει βρέξει καθόλου! Αφού είναι αδύνατο να έχουμε αρνητικό ύψος βροχής, αυτό θα είναι και το ελάχιστο ύψος βροχής του έτους. Έτσι, σύμφωνα με την αρχική περιγραφή, αρκεί ο αλγόριθμος να καταγράψει την πρώτη ημέρα που σημειώθηκε μηδενικό ύψος βροχής και να τερματίσει.

Επειδή δεν μπορούμε να τροποποιήσουμε τον αλγόριθμο ώστε να αναζητά το πρώτο στοιχείο του πίνακα RAIN που είναι 0 (υπάρχει πάντα

η - πάρα πολύ μικρή βέβαια - πιθανότητα κάποια χρονιά να βρέχει - έστω και λίγο - όλες τις ημέρες), θα προσθέσουμε την οδηγία GOTO για να διακόψουμε την εκτέλεση του αλγορίθμου μόλις βρεθεί το πρώτο 0:

```
ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟ-ΥΨΟΣ-ΒΡΟΧΗΣ-ΜΕ-ΔΙΑΚΟΠΗ
ΔΕΔΟΜΕΝΑ
        WEEK, DAY, I, K: INTEGER;
        RAIN: ARRAY [1..52, 1..7] OF INTEGER;
ΑΡΧΗ
        WEEK := 1 ;
        DAY := 1 ;
        ΓΙΑ I := 1 ΕΩΣ 52 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := 1 ΕΩΣ 7 ΕΠΑΝΕΛΑΒΕ
                        ΕΑΝ (RAIN[I,K] < RAIN[WEEK,DAY]) ΤΟΤΕ
                                WEEK := I ;
                                DAY := K ;
                                ΕΑΝ (RAIN[WEEK,DAY] = 0) ΤΟΤΕ
                                        GOTO ΕΥΡΗΚΑ
                                ΕΑΝ-ΤΕΛΟΣ
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;

ΕΥΡΗΚΑ:
        ΤΥΠΩΣΕ("ΤΟ ΕΛΑΧΙΣΤΟ ΥΨΟΣ ΒΡΟΧΗΣ ΤΟΥ ΕΤΟΥΣ ΗΤΑΝ", RAIN[WEEK,DAY], "ΕΚΑΤΟΣΤΑ ΚΑΙ ΣΗΜΕΙΩΘΗΚΕ ΓΙΑ ΠΡΩΤΗ ΦΟΡΑ ΤΗΝ ΗΜΕΡΑ", DAY, "ΤΗΣ ΕΒΔΟΜΑΔΑΣ", WEEK)
ΤΕΛΟΣ
```

Η οδηγία GOTO παραπέμπει στο σημείο ΕΥΡΗΚΑ μόλις βρεθεί το πρώτο μηδενικό στοιχείο του πίνακα. Το σημείο αυτό καλείται «ετικέτα» (label) και αποτελεί ένα είδος «διεύθυνσης» μέσα στον αλγόριθμο, η οποία δεν μεταφράζεται σε εκτελέσιμη εντολή. Μετά την εκτέλεση της οδηγίας GOTO, το πρόγραμμα συνεχίζει από την οδηγία που ακολουθεί την ετικέτα ΕΥΡΗΚΑ.

Τι γίνεται τώρα στην περίπτωση που (εξαιτίας κάποιου λάθους κατά την εισαγωγή των στοιχείων) ένα στοιχείο του πίνακα έχει τιμή μικρότερη από μηδέν;

Εάν ο αλγόριθμός μας συναντήσει το στοιχείο αυτό πριν συναντήσει το πρώτο 0, τότε αυτό θα τυπώσει ως ελάχιστο ύψος βροχής! Αυτό οφείλεται στο ότι η σύγκριση με 0 γίνεται αφού βρεθεί μια νέα ελάχιστη τιμή και όχι για κάθε στοιχείο του RAIN (κάτι αναμενόμενο, αφού συμφωνήσαμε ότι δεν ψάχνουμε για το πρώτο 0 του πίνακα).

Για να αντιμετωπίσουμε αυτή την (διόλου απίθανη) περίπτωση, χρειάζεται να προσθέσουμε οδηγίες που θα διαχειρίζονται το λάθος που ανακύπτει με την εισαγωγή αρνητικών τιμών.

ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟ ΥΨΟΣ ΒΡΟΧΗΣ
ΜΕ ΔΙΑΚΟΠΗ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ ΣΦΑΛΜΑΤΩΝ

```
ΔΕΔΟΜΕΝΑ
        WEEK, DAY, I, K: INTEGER;
        RAIN: ARRAY [1..52, 1..7] OF INTEGER;
ΑΡΧΗ
        WEEK := 1 ;
        DAY := 1 ;
        ΓΙΑ I := 1 ΕΩΣ 52 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := 1 ΕΩΣ 7 ΕΠΑΝΕΛΑΒΕ
ΔΙΟΡΘΩΣΗ:
                        ΕΑΝ (RAIN[I,K] < RAIN[WEEK,DAY]) ΤΟΤΕ
                                ΕΑΝ (RAIN[I,K] < 0) ΤΟΤΕ
                                        GOTO ΛΑΘΟΣ
                                ΕΑΝ-ΤΕΛΟΣ ;
                                WEEK := I ;
                                DAY := K ;
                                ΕΑΝ (RAIN[WEEK,DAY] = 0) ΤΟΤΕ
                                        GOTO ΕΥΡΗΚΑ
                                ΕΑΝ-ΤΕΛΟΣ
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
```

GOTO ΕΥΡΗΚΑ ;

ΛΑΘΟΣ:
ΑΡΧΗ
ΤΥΠΩΣΕ*(“ΒΡΕΘΗΚΕ ΑΡΝΗΤΙΚΗ ΤΙΜΗ ΚΑΤΑΧΩΡΗΜΕΝΗ ΣΤΗΝ ΗΜΕΡΑ”, Κ, “ΤΗΣ ΕΒΔΟΜΑΔΑΣ”, Ι) ;*
RAIN[I,K] := - RAIN[I,K] ;
ΤΥΠΩΣΕ*(“ΓΙΑ ΝΑ ΣΥΝΕΧΙΣΤΕΙ Η ΕΚΤΕΛΕΣΗ, Η ΑΡΝΗΤΙΚΗ ΤΙΜΗ ΜΕΤΑΤΡΑΠΗΚΕ ΣΕ ΘΕΤΙΚΗ”) ;*
GOTO *ΔΙΟΡΘΩΣΗ*
ΤΕΛΟΣ

ΕΥΡΗΚΑ:
ΤΥΠΩΣΕ(“ΤΟ ΕΛΑΧΙΣΤΟ ΥΨΟΣ ΒΡΟΧΗΣ ΤΟΥ ΕΤΟΥΣ ΗΤΑΝ”, RAIN[WEEK,DAY], “ΕΚΑΤΟΣΤΑ ΚΑΙ ΣΗΜΕΙΩΘΗΚΕ ΓΙΑ ΠΡΩΤΗ ΦΟΡΑ ΤΗΝ ΗΜΕΡΑ”, DAY, “ΤΗΣ ΕΒΔΟΜΑΔΑΣ”, WEEK)

ΤΕΛΟΣ

Μόλις λοιπόν ο αλγόριθμος ανακαλύψει αρνητική τιμή, μεταφέρει απότομα τον έλεγχο ροής σε συγκεκριμένο σημείο του κώδικα, όπου γίνεται κάποιου είδους αντιμετώπιση του λάθους. Έπειτα, ο έλεγχος επιστρέφει στο «κυρίως τμήμα» του αλγορίθμου και επαναλαμβάνεται η σύγκριση με τη διορθωμένη τώρα πια τιμή. Παρατηρήστε ότι η γραμμικότητα του αλγορίθμου έχει καταστραφεί: το τμήμα διαχείρισης του λάθους και το τμήμα εκτύπωσης του αποτελέσματος, αν και βρίσκονται στη σειρά, ποτέ δεν εκτελούνται ακολουθιακά. Ακόμη, μετά το τέλος των επαναλήψεων, απαιτείται διακλάδωση στην οδηγία εκτύπωσης του αποτελέσματος, για να αποφευχθεί η άκαιρη εκτέλεση των οδηγιών διαχείρισης του λάθους.

Σύνοψη

Στο κεφάλαιο αυτό γνωρίσατε τις βασικές δομές του δομημένου προγραμματισμού και τις αρχές του διαδικασιακού παραδείγματος προγραμματισμού. Είδατε ότι κάθε τέτοιο πρόγραμμα μπορεί να συντεθεί από τρεις βασικές δομές (ακολουθία, επιλογή, επανάληψη), όμως σε πολλές σύγχρονες γλώσσες προγραμματισμού υποστηρίζονται και άλλες πιο σύνθετες προγραμματιστικές δομές (π.χ., πολλαπλή επιλογή, άλλες δομές επανάληψης κ.ά.).

Μετά από μια εισαγωγή στις βασικές έννοιες του προγραμματισμού, αρχίσατε να ασχολείστε με διαδικασιακό προγραμματισμό. Μέχρι το τέλος του κεφαλαίου, σας παρουσιάστηκαν διάφοροι αλγόριθμοι επεξεργασίας δεδομένων για τη λύση προβλημάτων μικρής έως μεσαίας πολυπλοκότητας. Όπως είδατε, κάθε αλγόριθμος είναι στενά συνδεδεμένος με τη δομή αναπαράστασης των δεδομένων που χρησιμοποιεί (θυμηθείτε το σχήμα δεδομένα εισόδου – επεξεργασία – δεδομένα εξόδου), γι' αυτό και οι αλγόριθμοι παρουσιάστηκαν κατανεμημένοι σε αυτούς που χειρίζονται στατικές (π.χ., πίνακες) και δυναμικές (π.χ., λίστες) δομές δεδομένων. Στο τέλος του κεφαλαίου, μέσα από τη συζήτηση για την αμφισβητούμενη χρησιμότητα της εντολής GOTO, σας δόθηκε η ευκαιρία να δείτε την πρακτική εφαρμογή τεχνικών αμυντικού προγραμματισμού.

Το κεφάλαιο αυτό είναι ίσως το πιο σημαντικό κεφάλαιο του τόμου, γι' αυτό και βεβαιωθείτε ότι έχετε κατανοήσει καλά το περιεχόμενό του πριν προχωρήσετε στο επόμενο κεφάλαιο, όπου παρουσιάζονται μερικοί σύνθετοι αλγόριθμοι.

Βιβλιογραφία Κεφαλαίου 6

[1] Bohm, C. & Jacopini, G. (1966), *Flow Diagrams, Turing Machines, and Languages with only two formation rules*. Communications of the ACM, 9(5), σελ. 366–371.

[2] Cooper, D. & Clancy, M. (1985), *Oh! Pascal!*. Norton & Company, NY.

[3] Dijkstra, W. E. (1968), *GOTO Statement Considered Harmful*. Communications of the ACM, 11(3), σελ. 147–148.

[4] Kernighan, B. & Ritchie, D. (1988), *Η γλώσσα προγραμματισμού C*. Prentice–Hall (Ελληνική έκδοση: Κλειδάριθμος).

ΚΕΦΑΛΑΙΟ 7

Προχωρημένα θέματα διαδικασιακού προγραμματισμού

Σκοπός

Το κεφάλαιο αυτό συμπληρώνει το θέμα του διαδικασιακού προγραμματισμού παρουσιάζοντας μερικές προηγμένες τεχνικές προγραμματισμού. Για την καλύτερη κατανόησή τους, αυτές παρουσιάζονται μέσα από ένα σύνολο (σχετικά) πολύπλοκων αλγορίθμων.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *Εξηγήσετε τη διαφορά μεταξύ διαδικασιών και συναρτήσεων*
- *Υπολογίσετε την εμβέλεια μιας μεταβλητής και να διακρίνετε τις τοπικές από τις σφαιρικές μεταβλητές*
- *Διαχωρίσετε ανάμεσα στο πέρασμα παραμέτρων με τιμή και μέσω διευθύνσεως, μεταξύ κυρίως προγράμματος και υπο-προγραμμάτων*
- *Γράψετε αναδρομικούς αλγορίθμους και να τους διακρίνετε από τους επαναληπτικούς αλγορίθμους*
- *Επιλύετε προβλήματα με την τεχνική της οπισθοδρόμησης*
- *Περιγράφετε τα βήματα τριών τουλάχιστον αλγορίθμων ταξινόμησης και ενός αλγορίθμου αναζήτησης*

Έννοιες κλειδιά

- *Κυρίως πρόγραμμα*
- *Υπο-πρόγραμμα*
- *Διαδικασίες*
- *Συναρτήσεις*
- *Εμβέλεια*
- *Καθολικές (σφαιρικές) δηλώσεις*
- *Τοπικές δηλώσεις*
- *Παράμετρος*
- *Πέρασμα παραμέτρων με τιμή*

- *Πέρασμα παραμέτρων μέσω διευθύνσεως (ή με αναφορά)*
- *Αναδρομή*
- *Οπισθοδρόμηση*
- *Ταξινόμηση*
- *Αναζήτηση*

Εισαγωγικές παρατηρήσεις

Έχοντας αποκτήσει εμπειρία στην ανάπτυξη διαδικασιακών προγραμμάτων και το χειρισμό των βασικών δομών δεδομένων, ήρθε η ώρα να μεταβούμε από το επίπεδο του κωδικοποιητή (με την έννοια του ανθρώπου που απλά γράφει κώδικα) στο επίπεδο του προγραμματιστή (με την έννοια του ανθρώπου που σχεδιάζει και προγραμματίζει τη λύση ενός προβλήματος βάσει αρχών προγραμματισμού).

Στο κεφάλαιο αυτό, λοιπόν, θα ασχοληθούμε αρχικά (ενότητα 7.1) με την εσωτερική δομή των προγραμμάτων που αναπτύσσουμε. Γνωρίζουμε ήδη ότι τα διαδικασιακά προγράμματα συντίθενται από ένα σύνολο δομών προγραμματισμού. Προχωρώντας ένα βήμα ακόμη, θα δούμε ότι ένα διαδικασιακό πρόγραμμα μπορεί να συντίθεται και από άλλα προγράμματα (τα λεγόμενα «υπο–προγράμματα»), τα οποία μπορεί επίσης να αποτελούνται από άλλα υπο-προγράμματα κ.ο.κ., δημιουργώντας μία ιεραρχία τμημάτων κώδικα. Εκτός από τον τρόπο ορισμού και χρήσης των υπο-προγραμμάτων, θα ασχοληθούμε και με θέματα επικοινωνίας ανάμεσα στα διάφορα υπο–προγράμματα, κλήσης και ανταλλαγής δεδομένων κ.λπ.

Στη συνέχεια, θα παρουσιαστεί ένα σύνολο προγραμματιστικών τεχνικών για την επίλυση πολύπλοκων προβλημάτων[1]*, δηλαδή:*

- *Η αναδρομή (ενότητα 7.2), κατά την οποία ένα πρόγραμμα χρησιμοποιεί τον εαυτό του ως υπο–πρόγραμμα!*
- *Η οπισθοδρόμηση (ενότητα 7.3), με την οποία μπορούμε να επιλέγουμε τις σωστές λύσεις μέσα από ένα τεράστιο σύνολο πιθανών λύσεων*

[1] *Ο ενδιαφερόμενος αναγνώστης μπορεί στο (Cooper, 1985) να μελετήσει την υλοποίηση σε γλώσσα Pascal της πλειοψηφίας των αλγορίθμων που περιλαμβάνονται στο κεφάλαιο αυτό.*

- *Η ταξινόμηση (ενότητα 7.4), με την οποία τοποθετούμε σε σειρά (διατάσσουμε) ένα σύνολο στοιχείων*
- *Η αναζήτηση (ενότητα 7.4), με την οποία βρίσκουμε στα γρήγορα ένα στοιχείο μέσα σε ένα σύνολο ομοειδών στοιχείων*

Οι τεχνικές αυτές χρησιμοποιούν τις δομές δεδομένων που περιγράψαμε στο κεφάλαιο 6, αλλά και την έννοια των υπο–προγραμμάτων που περιγράφεται στην ενότητα 7.1, την οποία θα πρέπει να κατανοήσετε πριν προχωρήσετε στις επόμενες ενότητες.

7.1 Υπο–προγράμματα

Ήδη από το κεφάλαιο 3 έχουμε αναφερθεί στη διάσπαση ενός προβλήματος σε μικρότερα (και ευκολότερο να λυθούν) προβλήματα, ως μία εξαιρετικά αποτελεσματική μέθοδο ανάπτυξης προγραμμάτων. Είχαμε μάλιστα συναντήσει και διάφορα στυλ προγραμματισμού, τα οποία υιοθετούν αυτή την προσέγγιση. Στο κεφάλαιο 4 μελετήσαμε τεχνικές σχεδίασης προγραμμάτων, οι οποίες βασίζονται κατεξοχήν στη διάσπαση ενός πολύπλοκου προγράμματος σε μικρότερα προγράμματα.

Στο εξής, λοιπόν, θα θεωρούμε ως **κυρίως πρόγραμμα (main program)** το τμήμα εκείνο του προγράμματος που χρησιμοποιείται για να υλοποιεί τις βασικές λειτουργίες του αλγορίθμου και (ή μερικές φορές, μόνο) για να «συντονίζει» την εκτέλεση των υπόλοιπων λειτουργιών. Αυτές υλοποιούνται ως **υπο–προγράμματα (sub-programs)** που καλούνται από το κυρίως πρόγραμμα.

Σημειώστε, βέβαια, πως τίποτε δεν μας εμποδίζει να χρησιμοποιούμε κάθε υπο-πρόγραμμα σαν ένα «κυρίως» πρόγραμμα που έχει τα δικά του υπο-προγράμματα, χτίζοντας με τον τρόπο αυτό ένα πρόγραμμα ως μία ιεραρχία τμημάτων λογισμικού.

Στον ψευδοκώδικα που έχουμε υιοθετήσει στον τόμο αυτό, ένα υπο-πρόγραμμα:

- δηλώνεται με το είδος του, το όνομα του και τον κατάλογο των παραμέτρων, εάν υπάρχει,
- αμέσως μετά τη δήλωση ακολουθεί το τμήμα ΔΙΕΠΑΦΗ, όπου περιγράφεται ο τύπος των παραμέτρων εισόδου/εξόδου,
- καλείται από ένα άλλο πρόγραμμα με το όνομα του και ένα σύνολο τιμών ή μεταβλητών που αντιστοιχούν μία-προς-μία με τις παραμέτρους, οι οποίες έχουν δηλωθεί αμέσως μετά το όνομα του υπο-προγράμματος.

Σχόλιο Μελέτης

Ανατρέξτε στις ενότητες 4.2.2 και 4.2.3, όπου περιγράφεται το πρόγραμμα ΜΙΣΘΟΔΟΣΙΑ-ΥΠΑΛΛΗΛΟΥ και τα υπο–προγράμματά του. Μελετήστε τον τρόπο με τον οποίο αυτά καλούνται από το κυρίως πρόγραμμα.

7.1.1 Διαδικασίες και συναρτήσεις

Ένα υπο–πρόγραμμα χρησιμοποιείται για να εκτελέσει μία ή περισσότερες συγκεκριμένες λειτουργίες. Η σύζευξη των προγραμμάτων μας μειώνεται εάν χρησιμοποιούμε αυτάρκη υπο-προγράμματα που έχουν υψηλό βαθμό συνοχής (δηλαδή, εάν δηλώνουμε μέσα σε ένα υπο-πρόγραμμα όλα τα δεδομένα που μόνο αυτό χρειάζεται).

Τα είδη των υπο-προγραμμάτων είναι δύο:

- οι **διαδικασίες (procedures)**, οι οποίες ομαδοποιούν μια ακολουθία ενεργειών και επιστρέφουν τα αποτελέσματα σε κάποιες μεταβλητές (σημειώστε ότι το αποτέλεσμα της εκτέλεσης μιας διαδικασίας μπορεί να μην επιστρέφεται σε μεταβλητές, αλλά απλά να είναι αντιληπτό π.χ. στην οθόνη)
- οι **συναρτήσεις (functions)**, οι οποίες αντιπροσωπεύουν ένα σύνολο ενεργειών που οδηγούν στον υπολογισμό μιας τιμής. Αυτή καταχωρείται στο όνομα της συνάρτησης, η οποία χρησιμοποιείται ως μεταβλητή.

Παράδειγμα 7.1

(*Το παράδειγμα βασίζεται σε μια πρόταση της Καθ. Ε. Κεραυνού, Κριτικής Αναγνώστριας της ΘΕ, την οποία ο συγγραφέας ευχαριστεί).*

Στη συνέχεια, παρουσιάζεται μία έκδοση του αλγορίθμου ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ, τον οποίο είχαμε συναντήσει στη δραστηριότητα 4 του κεφαλαίου 6. Έχει ήδη αναφερθεί ότι για τον υπολογισμό του γινομένου δύο πινάκων χρειάζεται να υπολογίσουμε τα γινόμενα ενός αριθμού γραμμών του ενός με έναν αριθμό στηλών του άλλου. Εάν παρατηρήσουμε ότι κάθε γραμμή ή στήλη ενός πίνακα είναι ουσιαστικά ένας μονοδιάστατος πίνακας, τότε μπορούμε να αυξήσουμε την ευκρίνεια αλλά και την απόδοση του αλγορίθμου γράφοντας μια συνάρτηση SCALAR-PROD υπολογισμού του γινομένου δύο μονοδιάστατων πινάκων, την οποία θα καλούμε στο βασικό μας πρόγραμμα. Ακόμη, χρησιμοποιούμε τη διαδικασία ROWCOL, η οποία επιστρέφει στον μονοδιάστατο πίνακα SCAL, ανάλογα με την τιμή της λογικής μεταβλητής ROW, τη γραμμή (ROW = TRUE) ή τη στήλη (ROW = FALSE) που αντιστοιχεί στο δείκτη IND ενός διδιάστατου πίνακα AR.

ΑΛΓΟΡΙΘΜΟΣ ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ-ΕΚΔ1

```
ΔΕΔΟΜΕΝΑ
        A: ARRAY[1..M, 1..N] OF INTEGER ;
        B: ARRAY[1..N, 1..T] OF INTEGER ;
        G: ARRAY[1..M, 1..T] OF INTEGER ;
        FACT1, FACT2: ARRAY[1..N] OF INTEGER ;
        I, Z: INTEGER ;
```

ΔΙΑΔΙΚΑΣΙΑ ROWCOL(ROW, IND, AR, SCAL)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                ROW: BOOLEAN ;
                IND: INTEGER ;
                AR: ARRAY[1..M, 1..N] OF INTEGER ;
        ΕΞΟΔΟΣ
                SCAL: ARRAY[1..N] OF INTEGER ;
ΔΕΔΟΜΕΝΑ
        I: INTEGER ;
ΑΡΧΗ
        ΕΑΝ (ROW) ΤΟΤΕ
                ΓΙΑ I := 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
                        SCAL[I] := AR[IND, I]
                ΓΙΑ-ΤΕΛΟΣ
        ΑΛΛΙΩΣ
                ΓΙΑ I := 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
                        SCAL[I] := AR[I, IND]
                ΓΙΑ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ
```

ΣΥΝΑΡΤΗΣΗ SCALAR-PROD(R,C): INTEGER ;

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                R, C: ARRAY[1..K] OF INTEGER ;
        ΕΞΟΔΟΣ
```

```
            SCALAR-PROD: INTEGER ;
ΔΕΔΟΜΕΝΑ
        I, TEMP: INTEGER ;
ΑΡΧΗ
        TEMP := 0 ;
        ΓΙΑ I := 1 ΕΩΣ Κ ΕΠΑΝΕΛΑΒΕ
                TEMP := TEMP + R[I] * C[I]
        ΓΙΑ-ΤΕΛΟΣ ;
        SCALAR-PROD := TEMP
ΤΕΛΟΣ-ΣΥΝΑΡΤΗΣΗΣ

ΑΡΧΗ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
        ΓΙΑ I := 1 ΕΩΣ Μ ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ Z := 1 ΕΩΣ Τ ΕΠΑΝΕΛΑΒΕ
                        ΥΠΟΛΟΓΙΣΕ ROWCOL (TRUE, I, A, FACT1) ;
                        ΥΠΟΛΟΓΙΣΕ ROWCOL (FALSE, Z, B, FACT2) ;
                        G[I,Z] := SCALAR-PROD(FACT1, FACT2)
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

Παρατηρήστε ότι ο βασικός αλγόριθμος είναι σχετικά μικρός σε μέγεθος, αφού αρκετά μεγάλο τμήμα των υπολογισμών διεξάγεται στα επί μέρους τμήματα (μία διαδικασία και μία συνάρτηση). Η κλήση των τμημάτων μέσα στο κυρίως πρόγραμμα είναι διαφορετική για το καθένα: η κλήση διαδικασιών μοιάζει με αυτή που χρησιμοποιούσαμε έως τώρα, ενώ η κλήση συναρτήσεων συνήθως γίνεται μέσα σε μια έκφραση, αφού οι συναρτήσεις επιστρέφουν κάποια τιμή στη μεταβλητή που ορίζει το όνομά τους. Ακόμη, προτιμήσαμε να δηλώσουμε τα επί μέρους τμήματα μέσα στον βασικό αλγόριθμο, μια δυνατότητα που παρέχουν οι σύγχρονες γλώσσες προγραμματισμού.

Δραστηριότητα 7.1

Δοκιμάστε να ξαναγράψετε τον αλγόριθμο ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ που παρουσιάζεται στη δραστηριότητα 2 του κεφαλαίου 6 χρησιμοποιώντας υπο-προγράμματα. Στη συνέχεια, μελετήστε τη δική μας πρόταση.

Υπόδειξη: Ξεκινήστε μετατρέποντας τα υπο–προγράμματα σε διαδικασίες και προσέξτε το πέρασμα των παραμέτρων.

Καθώς μελετάτε τον προτεινόμεενο αλγόριθμο που ακολουθεί, παρατηρήστε ότι χρησιμοποιούμε δύο μεταβλητές πίνακα για να αναφερθούμε στην σκακιέρα, την CH στο κυρίως πρόγραμμα και την CHESS μέσα στα υποπρογράμματα. Ο λόγος θα εξηγηθεί στην ενότητα 7.1.3 μαζί με την σημασία του συμβόλου «%».

```
ΑΛΓΟΡΙΘΜΟΣ ΒΑΣΙΛΙΣΣΕΣ-ΣΤΙΣ-ΓΩΝΙΕΣ-ΕΚΔ4

ΔΕΔΟΜΕΝΑ
        CH : ARRAY[1..8,1..8] OF INTEGER ;
        I, K: INTEGER ;


ΔΙΑΔΙΚΑΣΙΑ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ(N, %CHESS)

ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                N: INTEGER ;
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
        ΕΞΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
ΔΕΔΟΜΕΝΑ
        I : INTEGER ;
ΑΡΧΗ
        ΓΙΑ I:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                CHESS [I,N] := 1
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;
```

ΔΙΑΔΙΚΑΣΙΑ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ(N, %CHESS)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                N: INTEGER ;
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
        ΕΞΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
ΔΕΔΟΜΕΝΑ
        I : INTEGER ;
ΑΡΧΗ
        ΓΙΑ I:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                CHESS[N,I] := 1
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;
```

ΔΙΑΔΙΚΑΣΙΑ ΔΕΞΙΑ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(%CHESS)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
        ΕΞΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
ΔΕΔΟΜΕΝΑ
        I : INTEGER ;
ΑΡΧΗ
        ΓΙΑ I:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                CHESS[I,I] := 1
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;
```

ΔΙΑΔΙΚΑΣΙΑ ΑΡΙΣΤΕΡΗ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(%CHESS)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
        ΕΞΟΔΟΣ
                CHESS : ARRAY[1..8,1..8] OF INTEGER ;
```

```
ΔΕΔΟΜΕΝΑ
        I : INTEGER ;
ΑΡΧΗ
        ΓΙΑ I:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                CHESS[I,(8-I)+1] := 1
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
        ΓΙΑ I := 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K:= 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                        CH[I,K] := 0
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
        ΥΠΟΛΟΓΙΣΕ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ(1, CH) ;
        ΥΠΟΛΟΓΙΣΕ ΓΡΑΜΜΗ-ΣΚΑΚΙΕΡΑΣ(8, CH) ;
        ΥΠΟΛΟΓΙΣΕ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ(1, CH) ;
        ΥΠΟΛΟΓΙΣΕ ΣΤΗΛΗ-ΣΚΑΚΙΕΡΑΣ(8, CH) ;
        ΥΠΟΛΟΓΙΣΕ ΔΕΞΙΑ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(CH) ;
        ΥΠΟΛΟΓΙΣΕ ΑΡΙΣΤΕΡΗ-ΔΙΑΓ-ΣΚΑΚΙΕΡΑΣ(CH)

ΤΕΛΟΣ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

7.1.2 Εμβέλεια

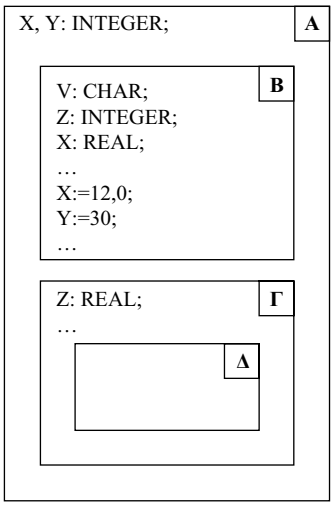
Με τον όρο **εμβέλεια (scope)** περιγράφουμε την «έκταση» μέσα σε ένα πρόγραμμα στην οποία ισχύει η δήλωση μιας μεταβλητής, σταθεράς ή υπο-προγράμματος. Έχουμε ήδη αναφέρει τί σημαίνει η δήλωση μιας μεταβλητής ή σταθεράς σε ένα πρόγραμμα: Κατά την εκτέλεση του, δεσμεύεται μια θέση μνήμης, η οποία συσχετίζεται με το όνομα της μεταβλητής ή σταθεράς, και στην οποία αποθηκεύεται κάθε φορά η τιμή της.

- Ο γενικός κανόνας της εμβέλειας είναι:

  Η δήλωση μιας μεταβλητής, ή σταθεράς, ή ενός υπο-προγράμματος ισχύει μέσα στο (υπο) πρόγραμμα στο οποίο βρίσκεται, και σε όλα τα υπο-προγράμματα που δηλώνονται μέσα σε αυτό.

Παράδειγμα 7.2

Στο Σχήμα 7.1 παρουσιάζεται μία ιεραρχία τεσσάρων υπο-προγραμμάτων, σε καθένα από τα οποία ορίζονται ή χρησιμοποιούνται κάποιες μεταβλητές.



Σχήμα 7.1.
Ιεραρχία υπο-προγραμμάτων για το παράδειγμα 2

Από τον γενικό κανόνα συμπεραίνουμε ότι μέσα σε ένα υπο–πρόγραμμα (έστω το Β) ισχύουν δύο είδη δηλώσεων:

- Αυτές που γίνονται στο τμήμα δηλώσεων του υπο-προγράμματος Β, δηλαδή **τοπικές (local) δηλώσεις**. Αυτό σημαίνει ότι, μόλις ο έλεγχος εκτέλεσης μεταφερθεί μέσα στο υπο-πρόγραμμα Β, δεσμεύονται θέσεις μνήμης για όλες τις τοπικές μεταβλητές. Οι θέσεις αυτές απελευθερώνονται (και κατά συνέπεια, οι μεταβλητές αυτές παύουν να «υπάρχουν») μόλις η εκτέλεση του υπο-προγράμματος τελειώσει και ο έλεγχος μεταφερθεί ξανά στο πρόγραμμα Α.
- Αυτές που γίνονται στο (υπό)πρόγραμμα Α μέσα στο οποίο έχει δηλωθεί το υπο-πρόγραμμα Β, δηλαδή **σφαιρικές ή καθολικές (global) δηλώσεις**. Οι θέσεις μνήμης για τις μεταβλητές αυτές εξακολουθούν να είναι δεσμευμένες ακόμη και όταν ο έλεγχος μεταφερθεί μέσα στο υπο-πρόγραμμα Β, ενώ αποδεσμεύονται μόνο όταν τελειώσει η εκτέλεση του Α.

Δύο σημεία πρέπει να προσέξουμε εδώ. Το πρώτο είναι η περίπτωση μια μεταβλητή, έστω η Χ, η οποία έχει δηλωθεί στο Α (θα αναφερόμαστε σε αυτή ως «σφαιρική Χ»), να δηλώνεται ξανά με το ίδιο όνομα στο Β (θα αναφερόμαστε σε αυτή ως «τοπική Χ»).

Τι σημαίνει αυτό; Ότι ενώ αρχικά είχε δεσμευθεί μια θέση μνήμης για την «σφαιρική Χ», στην πραγματικότητα, δεσμεύεται μια νέα θέση μνήμης για την «τοπική Χ». Άρα, υπάρχουν δύο θέσεις μνήμης δεσμευμένες στο ίδιο όνομα μεταβλητής! Εάν τώρα μέσα στο Β υπάρχει μια εντολή που χρησιμοποιεί τη Χ, π.χ. X := 12.0, σε ποια θέση μνήμης θα καταχωρηθεί η νέα τιμή;

- Στην περίπτωση αυτή, η τοπική δήλωση «επικαλύπτει» την σφαιρική, για όσο διάστημα ο έλεγχος εκτέλεσης βρίσκεται μέσα στο Β.

Συνεπώς, η τιμή 12.0 θα καταχωρηθεί στη θέση μνήμης που έχει

συσχετισθεί με την «τοπική Χ», και βέβαια, θα χαθεί μόλις τελειώσει η εκτέλεση του Β.

Το δεύτερο σημείο που χρειάζεται προσοχή είναι η χρήση σφαιρικών μεταβλητών μέσα σε ένα υπο-πρόγραμμα. Έστω, λοιπόν η μεταβλητή Υ, η οποία δηλώνεται στο Α, και έστω ότι στο Β υπάρχει η εντολή Υ := 30. Είναι σωστή η εντολή; Εάν ναι, τι θα συμβεί στο πρόγραμμα;

- Η προσπέλαση (ιδιαίτερα η καταχώρηση τιμής) μιας σφαιρικής μεταβλητής μέσα από ένα υπο-πρόγραμμα είναι σωστή, αλλά δεν είναι προγραμματιστικά ορθή, και πρέπει να αποφεύγεται. Τέτοιου είδους προγραμματισμός δημιουργεί ισχυρή σύζευξη ανάμεσα σε υπο-προγράμματα και είναι δυνατό να προκαλέσει παρενέργειες (side effects) κατά την εκτέλεση του προγράμματος, οι οποίες ανιχνεύονται πολύ δύσκολα!

Το πρόγραμμα λοιπόν θα καταχωρήσει την τιμή 30 στη θέση μνήμης που έχει συσχετιστεί με την Υ, αλλά εσείς θα πρέπει στο εξής να θυμάστε ότι η τιμή της Υ αλλάζει μέσα στο Β.

Δραστηριότητα 7.2

Παρενέργειες δεν εμφανίζονται όταν χρησιμοποιείται από ένα υπο–πρόγραμμα μια σφαιρική σταθερά. Μπορείτε να εξηγήσετε γιατί;

Όπως καταλαβαίνετε, όταν μέσα σε ένα υπο–πρόγραμμα χρησιμοποιείται η τιμή μίας σφαιρικής σταθεράς, δεν δημιουργούνται παρενέργειες, αφού η τιμή μιας σταθεράς δεν μπορεί να αλλάξει. Εάν όμως μέσα στο υπο–πρόγραμμα δηλωθεί μία μεταβλητή που έχει το ίδιο όνομα με τη σφαιρική σταθερά, τότε κατά τη διάρκεια εκτέλεσης του υπο-προγράμματος, μπορεί να γίνει καταχώρηση στην τοπική μεταβλητή (γενικότερα, οποιαδήποτε χρήση του ονόματος αναφέρεται στην τοπική μεταβλητή και όχι στην σφαιρική σταθερά).

Άσκηση αυτοαξιολόγησης 7.1

Ανατρέχοντας στο Σχήμα 7.1, προσπαθήστε να επιλέξετε τη σωστή απάντηση σε καθεμία από τις παρακάτω ερωτήσεις. Μόνο μία απάντηση είναι κάθε φορά σωστή.

1. Για το υπο–πρόγραμμα Β, η Χ είναι:
   (α) τοπική μεταβλητή (β) σφαιρική μεταβλητή (γ) τοπική σταθερά (δ) σφαιρική σταθερά

2. Για το υπο–πρόγραμμα Β, η Υ είναι:
   (α) τοπική μεταβλητή (β) σφαιρική μεταβλητή (γ) τοπική σταθερά (δ) σφαιρική σταθερά

3. Για το υπο–πρόγραμμα Γ, η Ζ είναι:
   (α) τοπική μεταβλητή (β) σφαιρική μεταβλητή (γ) τοπική σταθερά (δ) σφαιρική σταθερά

4. Μέσα στο υπο–πρόγραμμα Α, ο τύπος της Ζ είναι:
   (α) integer (β) real (γ) char (δ) δεν ορίζεται

5. Μέσα στο υπο-πρόγραμμα Δ, ο τύπος της Ζ είναι:
   (α) integer (β) real (γ) char (δ) δεν ορίζεται

7.1.3 Παράμετροι

Αρκετές φορές θα γράψουμε υπο-προγράμματα που χρησιμοποιούν μόνο τοπικά δεδομένα για να εκτελέσουν ένα σύνολο υπολογισμών και να εμφανίσουν το αποτέλεσμα στην οθόνη. Τις περισσότερες φορές όμως, τα υπο–προγράμματα που θα αναπτύξουμε θα χρειάζεται να χρησιμοποιούν δεδομένα από το κυρίως πρόγραμμα για να εκτελέσουν τους υπολογισμούς τους. Το αποτέλεσμα των υπολογισμών μπορεί να το τυπώνουν απευθείας στην οθόνη, αλλά είναι προτιμότερο να το επιστρέφουν στο κυρίως πρόγραμμα.

- Τα δεδομένα που ανταλλάσσονται μεταξύ του κυρίως προγράμματος και ενός υπο–προγράμματος λέγονται **παράμετροι (parameters)**. Για να γίνει σωστά η ανταλλαγή (λέγεται «πέρασμα παραμέτρων») πρέπει οι τιμές ή μεταβλητές που χρησιμοποιούνται στην κλήση του υπο–προγράμματος να αντιστοιχούν μία–προς–μία με τις παραμέτρους που περιλαμβάνει η δήλωση του υπο-προγράμματος.

Οι παράμετροι δηλώνονται πάντα στο τμήμα ΔΙΕΠΑΦΗ του υπο–προ-

γράμματος και δεν χρειάζεται να ξαναδηλωθούν στο τμήμα ΔΕΔΟΜΕΝΑ ή κάπου αλλού. Κατά την κλήση ενός υπο–προγράμματος, οι παράμετροι «κινούνται» προς δύο κατευθύνσεις:

- Εισέρχονται από το πρόγραμμα στο υπο–πρόγραμμα: Στην περίπτωση αυτή μόνο η τιμή της παραμέτρου (**πέρασμα με τιμή - pass by value**) περνιέται στο υπο–πρόγραμμα, το οποίο μπορεί να τη χρησιμοποιήσει. Στην πραγματικότητα, πρόκειται για μία τοπική μεταβλητή, η οποία δηλώνεται στο τμήμα ΔΙΕΠΑΦΗ του υποπρογράμματος και αρχικοποιείται σε κάποια τιμή κατά την κλήση του από το κυρίως πρόγραμμα.
- Επιστρέφουν από το υπο–πρόγραμμα στο πρόγραμμα: Εάν το υποπρόγραμμα χρειάζεται να επιστρέψει στο κυρίως πρόγραμμα κάποια απάντηση ως τιμή σε μία παράμετρο, τότε η παράμετρος αυτή συσχετίζεται με μία σφαιρική μεταβλητή του κυρίως προγράμματος, για την οποία λειτουργεί ως ψευδώνυμο (**πέρασμα με αναφορά ή πέρασμα μέσω διευθύνσεως – pass by reference**). Έτσι, το υπο–πρόγραμμα μπορεί να διαβάσει απευθείας την τιμή μιας τέτοιας μεταβλητής (χωρίς να χρειάζεται η δημιουργία τοπικού αντιγράφου της), ενώ η τιμή που επιστρέφει (μέσω της παραμέτρου) καταχωρείται αυτόματα στην αντίστοιχη σφαιρική μεταβλητή. Στο εξής, θα χρησιμοποιούμε το σύμβολο «%» κατά τη δήλωση παραμέτρων που περνιούνται μέσω διευθύνσεως (προσοχή: κατά την κλήση ενός υποπρογράμματος, δεν μπορούμε να χρησιμοποιήσουμε μια απόλυτη τιμή στη θέση μιας παραμέτρου που περνιέται μέσω διευθύνσεως).

Παράδειγμα 7.3 Έστω ο ακόλουθος αλγόριθμος, ο οποίος διαβάζει δύο αριθμούς, αντιστρέφει τη σειρά τους και τους τυπώνει:

ΑΛΓΟΡΙΘΜΟΣ ΑΝΤΣΤΡΟΦΗ-ΔΥΟ-ΑΡΙΘΜΩΝ

ΔΕΔΟΜΕΝΑ

FIRST, SECOND: INTEGER;

ΔΙΑΔΙΚΑΣΙΑ ΕΙΣΟΔΟΣ-ΑΡΙΘΜΩΝ(Α,Β)

ΔΙΕΠΑΦΗ

```
        ΕΙΣΟΔΟΣ
        ΕΞΟΔΟΣ
                A, B: INTEGER;
ΑΡΧΗ
        ΤΥΠΩΣΕ("ΠΟΙΟΣ ΕΙΝΑΙ Ο ΠΡΩΤΟΣ ΑΡΙΘΜΟΣ ;");
        ΔΙΑΒΑΣΕ (A);
        ΤΥΠΩΣΕ("ΠΟΙΟΣ ΕΙΝΑΙ Ο ΔΕΥΤΕΡΟΣ ΑΡΙΘΜΟΣ ;");
        ΔΙΑΒΑΣΕ (B)
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;
```

ΔΙΑΔΙΚΑΣΙΑ ΑΝΤΙΣΤΡΟΦΗ-ΣΕΙΡΑΣ(A,B)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                A, B: INTEGER;
        ΕΞΟΔΟΣ
                A, B: INTEGER;
ΔΕΔΟΜΕΝΑ
        TEMP: INTEGER;
ΑΡΧΗ
        TEMP := A;
        A := B;
        B := TEMP
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;
```

ΔΙΑΔΙΚΑΣΙΑ ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ(X,Y)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                X, Y: INTEGER;
        ΕΞΟΔΟΣ
ΑΡΧΗ
        ΤΥΠΩΣΕ("ΟΙ ΑΡΙΘΜΟΙ ΣΕ ΑΝΤΙΣΤΡΟΦΗ ΣΕΙΡΑ ΕΙΝΑΙ", X, "ΚΑΙ", Y)
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ;

ΑΡΧΗ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

```
    ΥΠΟΛΟΓΙΣΕ ΕΙΣΟΔΟΣ-ΑΡΙΘΜΩΝ(FIRST, SECOND) ;
    ΥΠΟΛΟΓΙΣΕ ΑΝΤΙΣΤΡΟΦΗ-ΣΕΙΡΑΣ(FIRST, SECOND) ;
    ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ(FIRST, SECOND)
ΤΕΛΟΣ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

Παρ' όλο που το πρόβλημα θα μπορούσε να είχε λυθεί χωρίς τη χρήση υπο-προγραμμάτων, παρατηρήστε πόσο κομψή είναι αυτή η λύση. Το κυρίως πρόγραμμα περιλαμβάνει μόνο τρεις οδηγίες με τις οποίες καλούνται τα υπο–προγράμματα. Μέσα στα τελευταία γίνονται, με δομημένο τρόπο, όλα τα βήματα επεξεργασίας που συνθέτουν τον αλγόριθμο.

Το κυρίως πρόγραμμα χρησιμοποιεί δύο σφαιρικές μεταβλητές, τις FIRST και SECOND, οι οποίες αρχικοποιούνται με το τέλος του υπο-προγράμματος ΕΙΣΟΔΟΣ-ΑΡΙΘΜΩΝ. Το πέρασμα παραμέτρων γίνεται μέσω διευθύνσεως και η αντιστοιχία είναι μία-προς-μία: η Α αντιστοιχεί στην FIRST και η Β στην SECOND.

Η αντιστροφή της σειράς γίνεται στο υπο-πρόγραμμα ΑΝΤΙΣΤΡΟΦΗ-ΣΕΙΡΑΣ, το οποίο διαβάζει στην είσοδο δύο μεταβλητές Α και Β και τις επιστρέφει έχοντας καταχωρήσει την τιμή της μίας στην άλλη (πέρασμα μέσω διευθύνσεως). Ενδιάμεσα χρησιμοποιεί την τοπική μεταβλητή TEMP. Οι Α και Β είναι τοπικές μεταβλητές επίσης, αλλά κατά την κλήση του υπο-προγράμματος συσχετίζονται με τις σφαιρικές μεταβλητές FIRST και SECOND. Έτσι, με το τέλος της εκτέλεσης του υπο-προγράμματος, είναι οι τιμές των σφαιρικών μεταβλητών που έχουν τελικά εναλλαχθεί.

Τέλος, η εκτύπωση γίνεται στο τμήμα ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ, το οποίο διαβάζει στην είσοδο δύο μεταβλητές Χ και Υ και τις τυπώνει. Κατά την κλήση του, αυτές οι τοπικές μεταβλητές αρχικοποιούνται στις τιμές των FIRST και SECOND αντίστοιχα (πέρασμα με τιμή).

Άσκηση αυτοαξιολόγησης 7.2

Ανατρέξτε στο παράδειγμα 3 του κεφαλαίου και σημειώστε στον πίνακα που ακολουθεί με ποιόν, κατά τη γνώμη σας, τρόπο γίνεται το πέρασμα παραμέτρων ανάμεσα στο κυρίως πρόγραμμα και τα υπο-προγράμματα που αναφέρονται στην αριστερή στήλη:

	ΜΕΣΩ ΔΙΕΥΘΥΝΣΕΩΣ	ΜΕ ΤΙΜΗ
ΥΠΟΛΟΓΙΣΕ ΕΙΣΟΔΟΣ ΑΡΙΘΜΩΝ		
Α		
Β		
ΥΠΟΛΟΓΙΣΕ ΑΝΤΙΣΤΡΟΦΗ–ΣΕΙΡΑΣ		
Α		
Β		
ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ		
Χ		
Ψ		

Άσκηση αυτοαξιολόγησης 7.3

Ανατρέξτε στο παράδειγμα 3 του κεφαλαίου και σημειώστε στον πίνακα που ακολουθεί τι είναι, κατά τη γνώμη σας, η κάθε μεταβλητή που αναφέρεται στην αριστερή στήλη:

	ΤΟΠΙΚΗ	ΣΦΑΙΡΙΚΗ
FIRST		
A		
TEMP		
Y		

7.1.4 Πώς υλοποιείται η κλήση των υπο–προγραμμάτων

Η εκτέλεση καθεμιάς εντολής ενός προγράμματος μεταβάλλει την κατάστασή του (έχουμε παρουσιάσει στην ενότητα 1.2.2 τον τρόπο με τον οποίο ο υπολογιστής εκτελεί τις εντολές ενός προγράμματος). Η κατάσταση ενός προγράμματος ορίζεται γενικά από τις τιμές που έχουν οι μεταβλητές του και από την επόμενη προς εκτέλεση εντολή και ουσιαστικά περιλαμβάνει όλες τις πληροφορίες που χρειάζεται ο υπολογιστής για να συνεχίσει να εκτελεί το πρόγραμμα (στην κατά-

σταση ενός προγράμματος περιλαμβάνεται και ένα σύνολο μεταβλητών που χρησιμοποιεί ο ίδιος ο υπολογιστής για να γνωρίζει εάν έχει συμβεί κάποια ειδική κατάσταση ή κάποιο λάθος κατά την εκτέλεση).

Ας γυρίσουμε ξανά στο Σχήμα 7.1. Όταν, λοιπόν, το πρόγραμμα Α καλέσει το υπο-πρόγραμμα Β, ο έλεγχος εκτέλεσης μεταφέρεται μέσα στο υπο-πρόγραμμα Β. Ο υπολογιστής τότε, αποθηκεύει την κατάσταση του Α κατά τη στιγμή της κλήσης (δηλαδή, τις τιμές των Χ και Υ και την επόμενη εντολή), ώστε να μπορεί να συνεχίσει ακριβώς από το σημείο που σταμάτησε μόλις τελειώσει η εκτέλεση του Β.

Προφανώς εάν υπάρχει μία ολόκληρη ιεραρχία προγραμμάτων, όπως στο Σχήμα 7.1, πρέπει κάθε φορά να αποθηκεύεται η κατάσταση του καλούντος υπο-προγράμματος. Επιπρόσθετα, οι πληροφορίες αυτές πρέπει να τοποθετηθούν με τέτοιο τρόπο ώστε κάθε φορά ο υπολογιστής να συνεχίζει με το σωστό πρόγραμμα. Για παράδειγμα, όταν τελειώσει η εκτέλεση του Δ, πρέπει να συνεχιστεί η εκτέλεση του Γ και όχι του Β!

ΕΝΑΡΞΗ Α	ΠΡΟΣΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Α ΣΤΗ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Α
ΚΛΗΣΗ Β	ΠΡΟΣΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Β ΣΤΗ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Β ΚΑΤΑΣΤΑΣΗ Α
ΤΕΡΜΑΤΙΣΜΟΣ Β	ΑΦΑΙΡΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Β ΑΠΟ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Α
ΚΛΗΣΗ Γ	ΠΡΟΣΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Γ ΣΤΗ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Γ ΚΑΤΑΣΤΑΣΗ Α
ΚΛΗΣΗ Δ	ΠΡΟΣΘΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Δ ΣΤΗ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Δ ΚΑΤΑΣΤΑΣΗ Γ ΚΑΤΑΣΤΑΣΗ Α
ΤΕΡΜΑΤΙΣΜΟΣ Δ	ΑΦΑΙΡΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Δ ΑΠΟ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Γ ΚΑΤΑΣΤΑΣΗ Α
ΤΕΡΜΑΤΙΣΜΟΣ Γ	ΑΦΑΙΡΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Γ ΑΠΟ ΣΤΟΙΒΑ	ΚΑΤΑΣΤΑΣΗ Α
ΤΕΡΜΑΤΙΣΜΟΣ Α	ΑΦΑΙΡΕΣΗ ΚΑΤΑΣΤΑΣΗΣ Α ΑΠΟ ΣΤΟΙΒΑ	(ΑΔΕΙΑ)

Σχήμα 7.2

Η μεταβολή της στοίβας του προγράμματος Α

Για το σκοπό αυτό χρησιμοποιείται μία δομή στοίβας. Μόλις συμβεί μία κλήση υπο–προγράμματος, η κατάσταση του προγράμματος που εκτελείται τοποθετείται στην κορυφή της στοίβας. Με τον τρόπο αυτό, στην κορυφή της στοίβας βρίσκεται πάντα η κατάσταση του προγράμματος που έχει καλέσει τελευταίο κάποιο υπο-πρόγραμμα. Μόλις τελειώσει η εκτέλεση του υπο-προγράμματος, διαγράφεται η κατάστασή του από τη στοίβα, ανακαλείται η πρώτη κατάσταση και συνεχίζει η εκτέλεση του αντίστοιχου προγράμματος.

Στο Σχήμα 7.2 φαίνεται ο τρόπος με τον οποίο μεταβάλλονται τα περιεχόμενα της στοίβας του προγράμματος Α του Σχήματος 7.1.

Δραστηριότητα 7.3

Δοκιμάστε να συνοψίσετε σε 250 περίπου λέξεις τα κυριότερα σημεία που αναφέρθηκαν στην ενότητα 7.1 για τη χρήση των υποπρογραμμάτων. Στη συνέχεια συγκρίνετε την απάντησή σας με το κείμενο που ακολουθεί.

■ Ας συνοψίσουμε τα κυριότερα σημεία που χρειάζεται να θυμάστε όταν χρησιμοποιείτε υπο–προγράμματα:

- Μπορούμε (και πρέπει) να ορίζουμε υπο–προγράμματα τόσο στο κυρίως πρόγραμμα, όσο και μέσα σε οποιοδήποτε υποπρόγραμμα.
- Μια διαδικασία αναπαριστά μια ακολουθία ενεργειών και αντικαθιστά μια εντολή προγράμματος. Μία συνάρτηση αναπαριστά μία τιμή και αντικαθιστά μία έκφραση.
- Ένα υπο–πρόγραμμα δομείται όπως ένα οποιοδήποτε πρόγραμμα.
- Οι μεταβλητές, οι σταθερές και τα υπο-προγράμματα που δηλώνονται μέσα σε ένα πρόγραμμα έχουν καθολική (σφαιρική) εμβέλεια και μπορούν να χρησιμοποιηθούν από όλα τα υπο–προγράμματα που δηλώνονται μέσα στο πρόγραμμα.
- Οι μεταβλητές, οι σταθερές και τα υπο-προγράμματα που δηλώνονται μέσα σε ένα υπο–πρόγραμμα έχουν περιορισμένη (τοπική) εμβέλεια και μπορούν να χρησιμοποιηθούν μόνο μέσα στο υπο–πρόγραμμα (και όχι σε κάποιο εξωτερικό πρόγραμμα).

- Μια τοπική δήλωση επικαλύπτει μια σφαιρική δήλωση με το ίδιο όνομα για όσο διάστημα ο έλεγχος εκτέλεσης βρίσκεται στο υπο–πρόγραμμα
- Εάν υπάρχουν τοπικές μεταβλητές που αρχικοποιούνται σε τιμές που περνιούνται από το κυρίως πρόγραμμα προς το υπο–πρόγραμμα, τότε έχουμε πέρασμα παραμέτρων με τιμή
- Εάν χρειάζεται να δώσει τιμή το υπο-πρόγραμμα σε κάποια μεταβλητή του κυρίως προγράμματος, τότε η σφαιρική μεταβλητή συσχετίζεται αυτόματα με μία τοπική μεταβλητή (πέρασμα παραμέτρων μέσω διευθύνσεως)
- Εάν το υπο-πρόγραμμα δίνει τιμή σε μία σφαιρική μεταβλητή κάνοντας απ' ευθείας καταχώρηση (και όχι μέσω περάσματος παραμέτρων μέσω διευθύνσεως), τότε μπορεί να εμφανιστούν επικίνδυνες παρενέργειες κατά την εκτέλεση του προγράμματος.

7.2 Αναδρομή

Ένας αλγόριθμος καλείται **αναδρομικός (recursive)** όταν μέσα στο σώμα του περιλαμβάνεται οδηγία με την οποία, άμεσα ή έμμεσα καλεί τον εαυτό του. Η **αναδρομή (recursion)** είναι μια κομψή τεχνική προγραμματισμού, η οποία όταν χρησιμοποιείται σωστά, οδηγεί στην ανάπτυξη ευκρινών και αποτελεσματικών προγραμμάτων. Από την άλλη πλευρά, όλοι οι αναδρομικοί αλγόριθμοι μπορούν να γραφούν με τη χρήση των συνηθισμένων δομών επανάληψης, αλλά στις περισσότερες περιπτώσεις, ο αλγόριθμος που προκύπτει είναι δυσνόητος.

Η αναδρομή χρησιμοποιείται πολύ όταν το πρόγραμμα προσπελαύνει δομές δεδομένων που είναι από τη φύση τους αναδρομικές (π.χ. δένδρα, λίστες κ.λπ.). Το βασικό μειονέκτημα της τεχνικής αυτής, εκτός από τη δυσκολία εξοικείωσης στη χρήση της, είναι ότι το πρόγραμμα καταναλώνει πολλούς υπολογιστικούς πόρους (κυρίως μνήμη). Αυτό οφείλεται στον τρόπο με τον οποίο οι σύγχρονες γλώσσες προγραμματισμού υλοποιούν την κλήση υπο–προγραμμάτων, δηλαδή, στη χρήση του σωρού, όπως περιγράφηκε στην ενότητα 7.1.4.

Παράδειγμα 7.4

Ας περιγράψουμε έναν αλγόριθμο, ο οποίος διαβάζει έναν αριθμό και αντιστρέφει τη σειρά των ψηφίων του. Στη συνέχεια, στο αριστερό μέρος παρατίθεται ένας επαναληπτικός αλγόριθμος και στο δεξί ένας αναδρομικός αλγόριθμος για το συγκεκριμένο πρόβλημα.

ΑΛΓΟΡΙΘΜΟΣ ΑΝΤΙΣΤΡΟΦΗ-ΑΡΙΘΜΟΥ-ΕΠΑΝ

```
ΔΕΔΟΜΕΝΑ
        DIG, N: INTEGER ;
ΑΡΧΗ
        ΔΙΑΒΑΣΕ(N) ;
        ΕΠΑΝΕΛΑΒΕ
                DIG := N mod 10 ;
                ΤΥΠΩΣΕ(DIG) ;
                N := N div 10
        ΜΕΧΡΙ (N=0)
ΤΕΛΟΣ
```

ΑΛΓΟΡΙΘΜΟΣ ΑΝΤΙΣΤΡΟΦΗ-ΑΡΙΘΜΟΥ-ΑΝΑΔΡ(N)

```
ΔΕΔΟΜΕΝΑ
        DIG, N: INTEGER ;
ΑΡΧΗ
        DIG := N mod 10 ;
        ΤΥΠΩΣΕ(DIG) ;
        N := N div 10 ;
        ΕΑΝ (N <> 0) ΤΟΤΕ
            ΥΠΟΛΟΓΙΣΕ ΑΝΤΙΣΤΡΟΦΗ-
            ΑΡΙΘΜΟΥ-ΑΝΑΔΡ(N)
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Ο αλγόριθμος στα αριστερά, αφού διαβάσει τον αριθμό N, τον διαιρεί με 10 και τυπώνει το υπόλοιπο της διαίρεσης. Εάν το πηλίκο της διαίρεσης είναι διαφορετικό από 0, τότε τα βήματα επαναλαμβάνονται για το πηλίκο, μέχρι αυτό να γίνει 0. Παρατηρήστε ότι χρησιμοποιούμε δομή επανάληψης με τη συνθήκη στο τέλος, αφού ένας αριθμός θα έχει τουλάχιστον ένα ψηφίο.

Ο αλγόριθμος στα δεξιά, αφού διαβάσει ως είσοδο τον αριθμό N, τον διαιρέσει με 10 και τυπώσει το υπόλοιπο της διαίρεσης, εξετάζει και αυτός το πηλίκο της διαίρεσης. Εάν αυτό είναι διαφορετικό από 0, καλεί τον εαυτό του περνώντας στην είσοδο το πηλίκο της διαίρεσης.

Ο αλγόριθμος με επανάληψη χρησιμοποιεί δύο μεταβλητές, DIG και N, στις οποίες απλά καταχωρεί το υπόλοιπο και το πηλίκο της διαίρεσης σε κάθε επανάληψη. Ο αλγόριθμος με αναδρομή, χρησιμοποιεί μεν τις ίδιες μεταβλητές, αλλά κάθε φορά που καλεί τον εαυτό του χρειάζεται να αποθηκεύει στη στοίβα την τρέχουσα κατάσταση, δηλαδή τις τιμές των μεταβλητών και τις εντολές που απομένουν για εκτέ-

λεση. Η στοίβα λοιπόν μεγαλώνει μέχρι το πηλίκο να γίνει 0. Τότε, αφού δεν καλείται ο αλγόριθμος ξανά, εκτελούνται οι εντολές που απέμειναν από την τελευταία κλήση, έπειτα αυτές που απέμειναν από την προτελευταία, κ.ο.κ, μέχρι να αδειάσει η στοίβα.

Δραστηριότητα 7.4

Διατυπώστε έναν αναδρομικό αλγόριθμο για τον υπολογισμό των αριθμών Fibonacci, όταν οι δύο πρώτοι αριθμοί της σειράς είναι 0 και 1. Η δική μας πρόταση ακολουθεί.

```
ΑΛΓΟΡΙΘΜΟΣ FIB-REC

ΔΕΔΟΜΕΝΑ
        N: INTEGER ;


ΣΥΝΑΡΤΗΣΗ FIBONACCI(B)
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                B: INTEGER ;
        ΕΞΟΔΟΣ
                FIBONACCI: INTEGER ;
ΑΡΧΗ
        ΕΑΝ (B = 1 OR B = 2) ΤΟΤΕ
                FIBONACCI := 1
        ΑΛΛΙΩΣ
                FIBONACCI := FIBONACCI(B-1) + FIBONACCI(B-2)
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΣΥΝΑΡΤΗΣΗΣ ;

ΑΡΧΗ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
        ΔΙΑΒΑΣΕ(N) ;
        ΤΥΠΩΣΕ(FIBONACCI(N))
ΤΕΛΟΣ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

Η εντολή **ΤΥΠΩΣΕ**(FIBONACCI(N)) ουσιαστικά χρησιμοποιεί τη συνάρτηση FIBONACCI(N) ως μεταβλητή. Η ίδια εντολή θα μπορούσε να γραφεί αναλυτικότερα ως:

```
TEMP := FIBONACCI(N)
ΤΥΠΩΣΕ(TEMP)
```

Παρατηρήστε ότι χρήσιμοποιώντας τη συνάρτηση FIBONACCI(N), αποφεύγουμε τις μεταβλητές υπολογισμού που είχαμε χρησιμοποιήσει στην έκδοση χωρίς αναδρομή του αλγορίθμου (Δραστηριότητα 1 / Κεφάλαιο 3).

Δραστηριότητα 7.5 (δύσκολη)

Διατυπώστε έναν αναδρομικό αλγόριθμο για το παιχνίδι «Πύργοι του Ανόι» και ιχνηλατήστε την εκτέλεση του για 4 δίσκους.

Το παιχνίδι αυτό παίζεται με τρεις στύλους και Ν δίσκους διαφορετικού μεγέθους, οι οποίοι είναι τοποθετημένοι στον πρώτο στύλο έτσι, ώστε ο μεγαλύτερος να βρίσκεται στο τέλος του σωρού και ο μικρότερος στην κορυφή. Ο στόχος είναι να μετακινηθεί ο σωρός όπως είναι στον τρίτο στύλο, με τις εξής προϋποθέσεις: μόνο ένας δίσκος μπορεί να μετακινηθεί κάθε φορά, και δεν επιτρέπεται ένας δίσκος να τοποθετηθεί πάνω από ένα μικρότερο δίσκο.

Υπόδειξη: πριν περιγράψετε τον αλγόριθμο, δοκιμάστε να παίξετε το παιχνίδι με φυσικά αντικείμενα, για να ανακαλύψετε την κρυμμένη αναδρομή. Ξεκινήστε με 1 και 2 δίσκους, και έπειτα δοκιμάστε με 3, πριν επιχειρήσετε να παίξετε με 4 δίσκους.

Ακολουθώντας την υπόδειξη, βλέπουμε ότι το πρόβλημα του ενός δίσκου είναι πολύ απλό: μετακινούμε το δίσκο από το στύλο Α στο στύλο Γ. Όταν οι δίσκοι είναι δύο, χρειάζεται να χρησιμοποιήσουμε και το μεσαίο στύλο: Μετακινούμε το μικρό δίσκο στο στύλο Β, έπειτα μετακινούμε το μεγάλο δίσκο στο στύλο Γ, και τέλος μετακινούμε το μικρό δίσκο από το στύλο Β στο στύλο Γ.

Όταν έχουμε τρεις δίσκους, το πρόβλημα αποκτά ενδιαφέρον. Μετακινούμε το δίσκο 1 στο στύλο Γ, έπειτα το δίσκο 2 στο στύλο Β, έπειτα το δίσκο 1 στο στύλο Β (σημείο α), έπειτα το δίσκο 3 στο στύλο Γ, έπειτα το δίσκο 1 στο στύλο Α, έπειτα το δίσκο 2 στο στύλο Γ, και τέλος το δίσκο 1 στο στύλο Γ.

Εάν όμως διαβάσετε λίγο περισσότερο προσεκτικά την παραπάνω περιγραφή, θα δείτε ότι ουσιαστικά μετακινούμε τους δύο πρώτους δίσκους από το στύλο Α στο στύλο Β (σημείο α), έπειτα μετακινούμε το μεγαλύτερο δίσκο στο Γ, και τέλος μετακινούμε τους δύο πρώτους δίσκους από το στύλο Β στο Γ. Πώς όμως μετακινούμε δύο δίσκους από ένα στύλο σε κάποιον άλλο; Μα αυτό το έχουμε ήδη περιγράψει.

Ερχόμαστε τώρα στην περίπτωση των τεσσάρων δίσκων. Όπως έχετε ήδη καταλάβει, αρκεί να μετακινήσουμε τους τρεις πρώτους δίσκους στο στύλο Β, έπειτα τον δίσκο 4 στο στύλο Γ, και τέλος τους τρεις πρώτους δίσκους από το στύλο Β στο στύλο Γ. Πώς όμως μετακινούμε τρεις δίσκους από ένα στύλο σε κάποιον άλλο; Μα αυτό το έχουμε ήδη περιγράψει.

Η αναδρομικότητα του αλγορίθμου είναι τώρα εμφανής, αλλά για να βοηθηθούμε λίγο περισσότερο, ας ονομάσουμε το στύλο Α όπου βρίσκονται οι 4 δίσκοι ΑΡΧικό, το στύλο Γ όπου θα τοποθετηθούν τελικά οι δίσκοι ΤΕΛικό και τον άλλο στύλο (Β) ενδιάΜΕΣΟ. Τότε, για να λύσουμε το πρόβλημα των Ν δίσκων, χρειάζεται να μετακινήσουμε τους Ν-1 πρώτους δίσκους στον ενδιάΜΕΣΟ στύλο, έπειτα τον δίσκο Ν στον ΤΕΛικό στύλο, και τέλος τους Ν-1 πρώτους δίσκους από τον ενδιάΜΕΣΟ στύλο στον ΤΕΛικό στύλο. Η μετακίνηση των Ν-1 δίσκων από το στύλο Α στο στύλο Β είναι το ίδιο ουσιαστικά πρόβλημα, εάν θεωρήσουμε το στύλο Β όπου βρίσκονται οι Ν-1 δίσκοι ως ΑΡΧικό, το στύλο Α όπου θα τοποθετηθούν οι δίσκοι ως ΤΕΛικό και τον άλλο στύλο (Γ) ως ενδιάΜΕΣΟ!

Ακολουθεί ο αναδρομικός αλγόριθμος.

ΑΛΓΟΡΙΘΜΟΣ ΠΥΡΓΟΙ-ΤΟΥ-ΑΝΟΪ

ΔΕΔΟΜΕΝΑ

K: INTEGER ;

ΔΙΑΔΙΚΑΣΙΑ ΤΟΠΟΘΕΤΗΣΕ(Ν, ΑΡΧ, ΤΕΛ, ΜΕΣΟ)

ΔΙΕΠΑΦΗ

ΕΙΣΟΔΟΣ

N: INTEGER ;

ΑΡΧ, ΤΕΛ, ΜΕΣΟ: CHAR ;

```
        ΕΞΟΔΟΣ
ΑΡΧΗ
        ΕΑΝ (N=1) ΤΟΤΕ
                ΤΥΠΩΣΕ("ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ", Ν, "ΑΠΟ", ΑΡΧ, "ΣΕ", ΤΕΛ)
        ΑΛΛΙΩΣ
                ΥΠΟΛΟΓΙΣΕ ΤΟΠΟΘΕΤΗΣΕ(Ν-1, ΑΡΧ, ΜΕΣΟ, ΤΕΛ) ;
                ΤΥΠΩΣΕ("ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ", Ν, "ΑΠΟ", ΑΡΧ, "ΣΕ", ΤΕΛ);
                ΥΠΟΛΟΓΙΣΕ ΤΟΠΟΘΕΤΗΣΕ(Ν-1, ΜΕΣΟ, ΤΕΛ, ΑΡΧ)
        ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ
        ΔΙΑΒΑΣΕ(Ν) ;
        ΥΠΟΛΟΓΙΣΕ ΤΟΠΟΘΕΤΗΣΕ(Ν, «Α», «Γ», «Β»)
ΤΕΛΟΣ
```

Για Ν=4, ο αλγόριθμός μας τυπώνει τα ακόλουθα μηνύματα:

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Α ΣΕ Β

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 2 ΑΠΟ Α ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Β ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 3 ΑΠΟ Α ΣΕ Β

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Γ ΣΕ Α

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 2 ΑΠΟ Γ ΣΕ Β

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Α ΣΕ Β

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 4 ΑΠΟ Α ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Β ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 2 ΑΠΟ Β ΣΕ Α

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Γ ΣΕ Α

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 3 ΑΠΟ Β ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Α ΣΕ Β

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 3 ΑΠΟ Α ΣΕ Γ

ΜΕΤΑΚΙΝΗΣΗ ΔΙΣΚΟΥ 1 ΑΠΟ Β ΣΕ Γ

Σωστό δεν είναι;

7.3 Οπισθοδρόμηση

Στην ενότητα αυτή περιγράφουμε την εφαρμογή των πινάκων σε μια κατηγορία αρκετά σύνθετων προβλημάτων, τα οποία απαιτούν τη χρήση της τεχνικής της **οπισθοδρόμησης (backtracking)**. Η τεχνική αυτή είναι παρόμοια με αυτή που θα χρησιμοποιούσαμε για να βρούμε την έξοδο ενός λαβύρινθου: Δοκιμάζουμε ένα μονοπάτι, μέχρις ότου βρούμε ότι αυτό οδηγεί σε αδιέξοδο, οπότε επιστρέφουμε στην τελευταία στροφή (που ουσιαστικά αποτελεί το τελευταίο σημείο αποδεδειγμένα λανθασμένης απόφασης) και διαλέγουμε έναν από τους υπόλοιπους δρόμους, κ.ο.κ.

Γενικεύοντας, τα προβλήματα αυτής της κατηγορίας έχουν ένα τεράστιο σύνολο πιθανών απαντήσεων, το οποίο εμείς διατρέχουμε χρησιμοποιώντας έναν αλγόριθμο. Γνωρίζοντας τις συνθήκες που πρέπει να ικανοποιεί κάποια σωστή λύση (το πρόβλημα μπορεί να έχει πολλές σωστές λύσεις!), εξετάζουμε καθεμία πιθανή απάντηση, μέχρι να συναντήσουμε μια λύση. Ένας πίνακας θα μπορούσε να χρησιμοποιηθεί για να διατηρούμε συνεχώς την κατάσταση της προσπάθειάς μας.

Παράδειγμα 7.5 Ένα διάσημο πρόβλημα αυτής της κατηγορίας είναι οι «8 βασίλισσες», στο οποίο προσπαθούμε να τοποθετήσουμε οκτώ βασίλισσες σε μια 8×8 σκακιέρα με τρόπο ώστε καμία να μην απειλείται από κάποια άλλη.

Πόσες πιθανές λύσεις έχει το πρόβλημα αυτό; Ένας απλός τρόπος να το βρούμε είναι να σκεφτούμε ότι ενώ πρέπει υποχρεωτικά να τοποθετήσουμε μια βασίλισσα σε κάθε στήλη (εάν δύο βασίλισσες τοποθετηθούν στην ίδια στήλη, τότε αλληλο-απειλούνται), κάθε βασίλισσα μπορεί να τοποθετηθεί σε οποιαδήποτε από τις οκτώ γραμμές της σκακιέρας. Συνεπώς οι πιθανές διατάξεις είναι 8^8=16.777.216!

Βέβαια, ένας αλγόριθμος που απλά διατρέχει όλες αυτές τις πιθανές απαντήσεις μια-προς-μια δεν παρουσιάζει μεγάλη πρακτική χρησιμότητα. Εμείς χρειαζόμαστε έναν αλγόριθμο, ο οποίος κάθε φορά που ανακαλύπτει μια λανθασμένη απόφαση, να αποκλείει όσο το δυνατό μεγαλύτερο αριθμό πιθανών (αλλά λανθασμένων) διατάξεων χωρίς να τις εξετάσει.

Αυτό μπορεί να γίνει εύκολα εάν θυμηθούμε ότι μια βασίλισσα απει-

λεί ένα πιόνι στο σκάκι, όταν αυτό βρίσκεται στην ίδια γραμμή ή στήλη με τη βασίλισσα, ή αριστερή ή δεξιά διαγώνιο σχετικά με τη βασίλισσα.

Χρησιμοποιούμε, λοιπόν, εάν διδιάστατο πίνακα C[8 × 8] για να διατηρούμε την τρέχουσα κατάσταση της σκακιέρας. Ξεκινάμε τοποθετώντας την πρώτη βασίλισσα στην πρώτη στήλη και σημειώνουμε όλα τα τετράγωνα που είναι πλέον «απαγορευμένα». Έπειτα, προσπαθούμε να τοποθετήσουμε τη δεύτερη βασίλισσα σε κάποιο μη–απαγορευμένο τετράγωνο της δεύτερης στήλης. Εάν το καταφέρουμε, σημειώνουμε τα νέα απαγορευμένα τετράγωνα και συνεχίζουμε με την τρίτη βασίλισσα, κ.ο.κ.

Για κάθε νέα βασίλισσα (έστω τη Ν-ιοστή) που δοκιμάζουμε να τοποθετήσουμε, χρησιμοποιούμε το τμήμα ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(Ν). Ο αλγόριθμος αυτός δοκιμάζει με τη σειρά καθεμία από τις 8 γραμμές. Εάν καταφέρει να τοποθετήσει τη βασίλισσα σε κάποιο τετράγωνο, τότε σημειώνει με 0 όλα τα απαγορευμένα τετράγωνα, και καλεί αναδρομικά τον εαυτό του για την επόμενη βασίλισσα (πρώτα βέβαια εξετάζει εάν υπάρχει επόμενη βασίλισσα, ή έχει τελειώσει επιτυχώς η αναζήτηση). Εάν η κλήση του ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(Μ) αποτύχει, τότε ο έλεγχος επιστρέφει στην προηγούμενη κλήση ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(Μ-1), ακυρώνει τις απαγορεύσεις και δοκιμάζει την επόμενη γραμμή. Εάν καμία γραμμή δεν είναι μη-απαγορευμένη, τότε ο αλγόριθμος επιστρέφει στην κλήση ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(Μ-2) κ.ο.κ.

Για παράδειγμα, εάν η πρώτη βασίλισσα τοποθετηθεί στη γραμμή 1, και κάθε φορά τοποθετούμε την επόμενη βασίλισσα στο πρώτο μη–απαγορευμένο τετράγωνο που συναντούμε, τότε η έκτη βασίλισσα δεν μπορεί να τοποθετηθεί πουθενά (μπορείτε να το ελέγξετε και μόνοι σας)! Ο αλγόριθμος, αφού δοκιμάσει ένα-προς-ένα τα τετράγωνα της έκτης στήλης, θα ακυρώσει την τοποθέτηση της πέμπτης βασίλισσας, την οποία θα επανατοποθετήσει στην πρώτη επόμενη μη–απαγορευμένη γραμμή, πριν δοκιμάσει ξανά για την έκτη βασίλισσα.

Μελετήστε προσεκτικά τον αλγόριθμο που ακολουθεί, καθώς είναι ένας από τους πιο σύνθετους αλγόριθμους που περιλαμβάνονται στον τόμο αυτό. Θα ήταν ωφέλιμο να δοκιμάζατε να εφαρμόσετε τον αλγόριθμο μέχρι να βρείτε π.χ. τις τρεις πρώτες λύσεις (το πλήθοςτων σωστών λύσεων είναι 92).

ΑΛΓΟΡΙΘΜΟΣ ΟΚΤΩ-ΒΑΣΙΛΙΣΣΕΣ

ΔΕΔΟΜΕΝΑ

```
I, K: INTEGER ;
C: ARRAY[1..8,1..8] OF BOOLEAN ;
SECLINE: ARRAY[1..8] OF BOOLEAN ;
SECLDIAG: ARRAY[2..16] OF BOOLEAN ;
SECRDIAG: ARRAY [-7..7] OF BOOLEAN ;
```

ΔΙΑΔΙΚΑΣΙΑ ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(COL)

ΔΙΕΠΑΦΗ

ΕΙΣΟΔΟΣ

COL: INTEGER ;

ΕΞΟΔΟΣ

ΔΕΔΟΜΕΝΑ

ROW: INTEGER ;

ΑΡΧΗ

```
ROW:= 1 ;
ΕΠΑΝΕΛΑΒΕ
    ΕΑΝ (SECLINE[ROW] = 1 AND SECLDIAG[ROW + COL] = 1 AND
    SECRDIAG[ROW - COL] = 1) ΤΟΤΕ
        SECLINE[ROW] := 0 ;
        SECLDIAG[ROW + COL] := 0 ;
        SECRDIAG[ROW - COL] := 0 ;
        C[ROW, COL] := 1 ;
        ΕΑΝ (COL < 8) ΤΟΤΕ
            ΥΠΟΛΟΓΙΣΕ ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(COL + 1)
        ΑΛΛΙΩΣ
            ΤΥΠΩΣΕ(C)
        ΕΑΝ-ΤΕΛΟΣ ;
        SECLINE[ROW] := 1 ;
        SECLDIAG[ROW + COL] := 1 ;
        SECRDIAG[ROW - COL] := 1 ;
        C[ROW, COL] := 0 ;
    ΕΑΝ-ΤΕΛΟΣ ;
    ROW:= ROW + 1
```

```
        ΜΕΧΡΙ (ROW > 8)
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ
        ΓΙΑ I := 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                SECLINE[I] := 1
        ΓΙΑ-ΤΕΛΟΣ ;
        ΓΙΑ I := 2 ΕΩΣ 16 ΕΠΑΝΕΛΑΒΕ
                SECLDIAG := 1
        ΓΙΑ-ΤΕΛΟΣ ;
        ΓΙΑ I := -7 ΕΩΣ 7 ΕΠΑΝΕΛΑΒΕ
                SECRDIAG := 1
        ΓΙΑ-ΤΕΛΟΣ ;
        ΓΙΑ I := 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := 1 ΕΩΣ 8 ΕΠΑΝΕΛΑΒΕ
                        C[I,K] := 0
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
        ΥΠΟΛΟΓΙΣΕ ΔΟΚΙΜΗ-ΣΤΗΛΗΣ(1)
ΤΕΛΟΣ
```

Αυτός ο αλγόριθμος πρώτα αρχικοποιεί τους πίνακες που θα χρησιμοποιήσουμε. Εκτός από τον πίνακα C (τη σκακιέρα), θα χρησιμοποιήσουμε:

- τον πίνακα SECLINE, ο οποίος επιτρέπει ή απαγορεύει τη χρήση κάποιας γραμμής του C (περιέχει την τιμή 1 για τις γραμμές που μπορούν να χρησιμοποιηθούν σε κάθε βήμα),
- τους πίνακες SECRDIAG και SECRDIAG, οι οποίοι με τον ίδιο τρόπο επιτρέπουν ή απαγορεύουν τη χρήση κάποιας διαγωνίου του C. Σημειώστε πώς τα στοιχεία C[α,β] που βρίσκονται στην ίδια αριστερή διαγώνιο με το στοιχείο C[μ,ν] έχουν όλα α+β = μ+ν, ενώ αυτά που βρίσκονται στην ίδια δεξιά διαγώνιο έχουν α-β = μ-ν.

Χρησιμοποιούμε αυτούς τους πίνακες για να καταγράφουμε προσωρινά τις γραμμές και διαγωνίους που μπορούμε να χρησιμοποιήσουμε σε κάθε βήμα, αποφεύγοντας να επέμβουμε στον C (παρά μόνο για

να σημειώσουμε τη θέση μιας βασίλισσας), ώστε να είναι εύκολη η αναίρεση των καταγραφών όταν συμβεί οπισθοδρόμηση.

Μετά το τέλος της αρχικοποίησης, ο αλγόριθμος καλεί τη διαδικασία ΔΟΚΙΜΗ-ΣΤΗΛΗΣ, με παράμετρο την πρώτη στήλη του C. Η διαδικασία αυτή προχωρά σημειώνοντας κάθε φορά τις απαγορευμένες θέσεις της σκακιέρας, και μόλις καταφέρει να τοποθετήσει μία νέα βασίλισσα καλεί τον εαυτό της αναδρομικά για να υπολογίσει τη θέση της επόμενης βασίλισσας. Εάν καταφέρει να τοποθετήσει επιτυχώς μία βασίλισσα στη στήλη 8, αυτό σημαίνει ότι έχει βρεθεί μία λύση, οπότε τυπώνεται ο C. Εάν δεν καταφέρει να τοποθετήσει μία βασίλισσα σε κάποια στήλη πριν την όγδοη, αναιρεί όλες τις αλλαγές και τερματίζει, μεταφέροντας τον έλεγχο στην προηγούμενη κλήση του εαυτού της, κατά την οποία ψάχνει επαναληπτικά για μία άλλη διαθέσιμη θέση στη σκακιέρα.

7.4 Ταξινόμηση και αναζήτηση

Ταξινόμηση (sorting) είναι η τοποθέτηση των στοιχείων ενός συνόλου κατά μια συγκεκριμένη σειρά. Απλά παραδείγματα είναι η ταξινόμηση των στοιχείων ενός πίνακα ακεραίων κατά φθίνουσα σειρά, η ταξινόμηση ενός αρχείου ονομάτων κατά αύξουσα αλφαβητική σειρά κ.ά.

Αναζήτηση (searching) είναι η προσπάθεια ανεύρεσης ενός στοιχείου μέσα σε ένα σύνολο. Σημειώστε ότι το στοιχείο μπορεί να βρεθεί, μπορεί και όχι. Πολύ συνηθισμένη περίπτωση είναι η αναζήτηση στοιχείων σε μια βάση δεδομένων (π.χ. η αναζήτηση ενός βιβλίου με βάση τον τίτλο του στη ηλεκτρονική βάση δεδομένων της βιβλιοθήκης του ΕΑΠ).

Όταν αναπτύσσουμε συστήματα λογισμικού που επεξεργάζονται μεγάλο όγκο δεδομένων, σχεδόν πάντα παρουσιάζεται η ανάγκη ταξινόμησης (όλων ή μέρους) των δεδομένων αυτών. Αντίστοιχα, παρουσιάζεται και η ανάγκη αναζήτησης ενός στοιχείου που έχει κάποια χαρακτηριστικά μέσα σε ένα σύνολο στοιχείων. Στην πραγματικότητα, οι αλγόριθμοι ταξινόμησης και αναζήτησης είναι τόσο σημαντικοί, που ο D. Knuth τους έχει αφιερώσει ένα ολόκληρο τόμο (τόμος 3) του επτάτομου έργου του «The Art of Computer Programming» (Knuth, 1973).

Στην ενότητα αυτή θα παρουσιαστούν, με λίγο σχολιασμό, οι πιο σημαντικοί αλγόριθμοι ταξινόμησης και αναζήτησης. Εάν ενδιαφέρεστε για περισσότερα, ρίξτε μια ματιά στην προτεινόμενη βιβλιογραφία, στο τέλος του τόμου αυτού.

7.4.1 Ταξινόμηση με επιλογή

Ο αλγόριθμος **ταξινόμησης με επιλογή (selection sort)** έχει ήδη περιγραφεί και αναλυθεί στη δραστηριότητα 4 της ενότητας 2.3 (καλύτερα να ρίξετε μια ματιά στον ψευδοκώδικα). Στα γρήγορα σας υπενθυμίζω εδώ τον τρόπο λειτουργίας του.

Εάν, λοιπόν, έχουμε έναν πίνακα P[1..N] με N στοιχεία, ο αλγόριθμος αυτός σε κάθε επανάληψη, βρίσκει το μεγαλύτερο (ή μικρότερο) από τα N στοιχεία, έστω P[K], και το αντιμεταθέτει με το στοιχείο P[1]. Έπειτα, εφαρμόζει την ίδια πρακτική στα N-1 στοιχεία του υπο–πίνακα P[2..N]. Όταν τελειώσει, ο πίνακας P είναι ταξινομημένος κατά φθίνουσα (ή αύξουσα) σειρά.

7.4.2 Ταξινόμηση με παρεμβολή

Ο αλγόριθμος **ταξινόμησης με παρεμβολή (insertion sort)** βασίζεται σε παρόμοια, απλή φιλοσοφία με τον προηγούμενο. Σε κάθε επανάληψη, το τμήμα του πίνακα P που βρίσκεται αριστερά του εκάστοτε πρώτου στοιχείου είναι ήδη ταξινομημένο. Μόνο που, αντί να βρίσκει το μικρότερο από τα εναπομένοντα στοιχεία του πίνακα και να το αντιμεταθέτει με το εκάστοτε πρώτο στοιχείο, αυτός ο αλγόριθμος διαβάζει το επόμενο κάθε φορά στοιχείο P[K] και το τοποθετεί στην κατάλληλη θέση ανάμεσα στα ήδη ταξινομημένα κατά αύξουσα σειρά P[1..K-1] στοιχεία στο αριστερό τμήμα του πίνακα.

Για να το πετύχει αυτό, διατρέχει τα στοιχεία αυτά αντίστροφα (χρησιμοποιεί το μετρητή CUR, ο οποίος παίρνει τιμές από K-1 προς 1), και ταυτόχρονα τα μετακινεί μια θέση προς τα δεξιά, έως ότου συναντήσει κάποιο στοιχείο P[CUR] που είναι μικρότερο από το P[K] (το οποίο έχει ήδη καταχωρηθεί στη μεταβλητή EL για να μη χαθεί κατά τη μετακίνηση των στοιχείων του πίνακα προς τα δεξιά). Αφού το P[CUR] έχει ήδη μετακινηθεί στη θέση P[CUR+1], το P[K] (δηλαδή η EL) παρεμβάλλεται στην ουσιαστικά κενή θέση P[CUR].

Ο αλγόριθμος χρησιμοποιεί έναν πίνακα με N+1 στοιχεία, γιατί στη

θέση P[0] καταχωρείται ένας πολύ μικρός αριθμός (-MAXINT), ώστε κατά την διαπέραση του πίνακα προς τα αριστερά να μη βρεθεί ο μετρητής CUR εκτός των ορίων του πίνακα (πάντα θα ισχύει EL > P[0], οπότε θα σταματήσει η εκτέλεση της ΕΝΟΣΩ δομής επανάληψης και το EL θα καταχωρηθεί στην πρώτη θέση του πίνακα).

ΑΛΓΟΡΙΘΜΟΣ ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΠΑΡΕΜΒΟΛΗ

```
ΔΕΔΟΜΕΝΑ
        EL, CUR, K, N: INTEGER ;
        P: ARRAY [0..N] OF INTEGER ;
ΑΡΧΗ
        P[0] := -MAXINT ;
        ΓΙΑ K := 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
                EL := P[K] ;
                CUR := K ;
                ΕΝΟΣΩ (P[CUR-1] > EL) ΕΠΑΝΕΛΑΒΕ
                        P[CUR] := P[CUR-1] ;
                        CUR := CUR-1
                ΕΝΟΣΩ-ΤΕΛΟΣ ;
                P[CUR] := EL
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

Μια περισσότερο αποδοτική έκδοση του αλγορίθμου ελέγχει στο τέλος κάθε περάσματος εάν έγιναν ανταλλαγές στοιχείων. Έτσι, ως συνθήκη τερματισμού της εξωτερικής επανάληψης μπορεί να χρησιμοποιηθεί το γεγονός ότι στο τελευταίο πέρασμα δεν έγινε καμιά ανταλλαγή στοιχείων (άρα ο πίνακας είναι ταξινομημένος).

7.4.3 Ταξινόμηση Φυσαλίδας

Ο αλγόριθμος αυτός **(bubble sort)** διαπερνά τον πίνακα P[1..N] αρχίζοντας από το στοιχείο P[N] και συγκρίνοντας τα γειτονικά στοιχεία ανά δύο (δηλαδή συγκρίνει πρώτα τα P[N] και P[N-1], έπειτα τα P[N-1] και P[N-2] κ.ο.κ.). Στο τέλος της διαπέρασης, το μικρότερο στοιχείο του πίνακα θα βρίσκεται στη θέση P[1]. Στην επόμενη διαπέρα-

ση, η διαδικασία επαναλαμβάνεται για τον υπο–πίνακα P[2..N] και το μικρότερο από τα N-1 στοιχεία «επιπλέει» μέχρι τη θέση P[2], κ.ο.κ.

ΑΛΓΟΡΙΘΜΟΣ ΤΑΞΙΝΟΜΗΣΗ-ΦΥΣΑΛΙΔΑΣ

```
ΔΕΔΟΜΕΝΑ
        I, K, N, TEMP: INTEGER ;
        P: ARRAY [1..N] OF INTEGER ;
ΑΡΧΗ
        ΓΙΑ I := 1 ΕΩΣ N-1 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := N ΕΩΣ I+1 ΕΠΑΝΕΛΑΒΕ
                        ΕΑΝ (P[K] < P[K-1]) ΤΟΤΕ
                                TEMP := P[K-1] ;
                                P[K-1] := P[K] ;
                                P[K] := P[K-1]
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

7.4.4 Γρήγορη ταξινόμηση

Ο αλγόριθμος της **γρήγορης ταξινόμησης (quicksort)** προτάθηκε από τον C.A.R. Hoare και βασίζεται στην αρχή «διαίρει και βασίλευε». Η βασική ιδέα είναι να τοποθετούμε σε κάθε πέρασμα του πίνακα τα μεγάλα στοιχεία στο δεξί μισό και τα μικρά στο αριστερό.

Ο κάθε πίνακας χωρίζεται στα δύο διαλέγοντας κάθε φορά ένα μεσαίο στοιχείο του (σχεδόν) στην τύχη, το οποίο καταχωρείται στη MID. Έπειτα, εξετάζουμε το αριστερό μισό του πίνακα για κάποιο στοιχείο μεγαλύτερο από το MID και το δεξί μισό για κάποιο μικρότερο. Αυτά τα δύο στοιχεία βρίσκονται εμφανώς σε λάθος πλευρά του πίνακα, οπότε τα ανταλλάσσουμε. Εάν αρχίσουμε το ψάξιμο από τις άκρες του πίνακα προχωρώντας προς τη μέση, κάποια στιγμή οι δύο δείκτες αναζήτησης θα συναντηθούν, οπότε σταματά η ταξινόμηση. Στο σημείο αυτό, το στοιχείο που είχε επιλεγεί ως μεσαίο βρίσκεται πια στη σωστή (αναφορικά με την τελική ταξινόμηση του πίνακα) θέση, οπότε το

αγνοούμε στη συνέχεια.

Συνεπώς, μετά από αυτό το πέρασμα του πίνακα, το αριστερό μισό περιλαμβάνει «μικρούς» αριθμούς και το δεξί «μεγάλους», ενώ το μεσαίο στοιχείο βρίσκεται στην τελική του θέση. Τώρα, εφαρμόζουμε τον αλγόριθμο στο αριστερό μισό του πίνακα, το οποίο χωρίζουμε στα δύο κ.λπ. Έπειτα, ο αλγόριθμος εφαρμόζεται στο δεξί μισό, και η διαδικασία επαναλαμβάνεται μέχρι οι προκύπτοντες υπο–πίνακες να περιέχουν μόνο ένα ή δύο στοιχεία ταξινομημένα.

Ακολουθεί ο ψευδοκώδικας του αλγορίθμου γρήγορης ταξινόμησης. Από την ανάλυση που προηγήθηκε, καταλαβαίνετε ότι πρόκειται για αναδρομικό αλγόριθμο, σε αντίθεση με τους υπόλοιπους αλγορίθμους ταξινόμησης που παρουσιάστηκαν, οι οποίοι χρησιμοποιούν δομές επανάληψης.

ΑΛΓΟΡΙΘΜΟΣ ΓΡΗΓΟΡΗ-ΤΑΞΙΝΟΜΗΣΗ

```
ΔΕΔΟΜΕΝΑ
        P: ARRAY[1..N] OF INTEGER ;
        N: INTEGER ;
```

ΔΙΑΔΙΚΑΣΙΑ QUICKSORT(START, END, %AR)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                START, END: INTEGER ;
                AR: ARRAY[1..N] OF INTEGER ;
        ΕΞΟΔΟΣ
                AR: ARRAY[1..N] OF INTEGER ;
ΔΕΔΟΜΕΝΑ
        LEFT, RIGHT, MID, TEMP: INTEGER ;
ΑΡΧΗ
        LEFT := START ;
        RIGHT := END ;
        MID := AR[START+END div 2] ;
        ΕΠΑΝΕΛΑΒΕ
                ΕΝΟΣΩ (AR[LEFT] < MID) ΕΠΑΝΕΛΑΒΕ
                        LEFT := LEFT + 1
```

```
            ΕΝΟΣΩ-ΤΕΛΟΣ ;
            ΕΝΟΣΩ (AR[RIGHT] > MID) ΕΠΑΝΕΛΑΒΕ
                  RIGHT:= RIGHT - 1
            ΕΝΟΣΩ-ΤΕΛΟΣ ;
            ΕΑΝ (LEFT < RIGHT) ΤΟΤΕ
                  TEMP := AR[LEFT] ;
                  AR[LEFT] := AR[RIGHT] ;
                  AR[RIGHT] := TEMP ;
                  LEFT := LEFT + 1 ;
                  RIGHT := RIGHT -1
            ΕΑΝ-ΤΕΛΟΣ
      ΜΕΧΡΙ (RIGHT <= LEFT) ;
      ΕΑΝ (START < RIGHT) ΤΟΤΕ
            ΥΠΟΛΟΓΙΣΕ QUICKSORT(START, RIGHT, AR)
      ΕΑΝ-ΤΕΛΟΣ ;
      ΕΑΝ (END > LEFT) ΤΟΤΕ
            ΥΠΟΛΟΓΙΣΕ QUICKSORT(LEFT, END, AR)
      ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ
      ΔΙΑΒΑΣΕ(P) ;
      ΥΠΟΛΟΓΙΣΕ QUICKSORT(1, N, P)
ΤΕΛΟΣ
```

7.4.5 Δυαδική αναζήτηση

Αφού είδαμε όλους αυτούς τους αλγορίθμους ταξινόμησης, ας δούμε τώρα πώς θα μπορούσαν να χρησιμοποιηθούν. Σκεφτείτε τον τηλεφωνικό κατάλογο της πόλης σας. Πότε είναι πιο εύκολο να βρείτε το τηλέφωνο ενός φίλου σας (ή να καταλάβετε ότι ο φίλος σας δεν έχει τηλέφωνο;) όταν οι συνδρομητές του ΟΤΕ έχουν καταχωρηθεί με αλφαβητική σειρά, ή όταν η σειρά παράθεσης των ονομάτων είναι τυχαία;

Στις περισσότερες λοιπόν περιπτώσεις είναι πιο εύκολο να βρούμε ένα συγκεκριμένο στοιχείο ενός πίνακα (ή να διαπιστώσουμε την απουσία του), όταν ο πίνακας είναι ταξινομημένος. Σε τέτοια περίπτωση, έχει

αποδειχθεί ότι ο καλύτερος αλγόριθμος που μπορούμε να χρησιμοποιήσουμε είναι η **δυαδική αναζήτηση (binary seacrh)**.

Πρόκειται για άλλον έναν αλγόριθμο που υλοποιεί την αρχή «διαίρει και βασίλευε» (και συνεπώς περιγράφεται αναδρομικά). Η ιδέα είναι να χωρίσουμε τον πίνακα P σε δύο μέρη, να δούμε σε ποιο μισό είναι πιθανό να βρίσκεται το στοιχείο EL που αναζητούμε, έπειτα να χωρίσουμε το τμήμα αυτό του πίνακα στα δύο κ.ο.κ. Ο αλγόριθμος που ακολουθεί επιστρέφει τη θέση POS του στοιχείου EL μέσα στον πίνακα P, ή 0 σε περίπτωση που αυτό δεν βρεθεί.

ΑΛΓΟΡΙΘΜΟΣ ΔΥΑΔΙΚΗ-ΑΝΑΖΗΤΗΣΗ

```
ΔΕΔΟΜΕΝΑ
    P: ARRAY[1..N] OF INTEGER ;
    EL, N, POS: INTEGER ;
```

ΔΙΑΔΙΚΑΣΙΑ BINSEARCH(KEY, AR, LEFT, RIGHT, %INDEX)

```
ΔΙΕΠΑΦΗ
    ΕΙΣΟΔΟΣ
        KEY, LEFT, RIGHT: INTEGER ;
        AR: ARRAY[1..N] OF INTEGER ;
    ΕΞΟΔΟΣ
        INDEX: INTEGER ;
ΔΕΔΟΜΕΝΑ
    MID: INTEGER ;
ΑΡΧΗ
    MID := (LEFT + RIGHT) div 2 ;
    ΕΑΝ (LEFT > RIGHT) ΤΟΤΕ
        INDEX := 0
    ΑΛΛΙΩΣ
        ΕΑΝ (KEY = AR[MID]) ΤΟΤΕ
            INDEX := MID
        ΑΛΛΙΩΣ
            ΕΑΝ (KEY < AR[MID]) ΤΟΤΕ
                ΥΠΟΛΟΓΙΣΕ BINSEARCH(KEY, AR, LEFT,
                MID-1, INDEX)
```

```
            ΑΛΛΙΩΣ
                ΕΑΝ (KEY > AR[MID]) ΤΟΤΕ
                    ΥΠΟΛΟΓΙΣΕ BINSEARCH (KEY, AR,
                    MID+1, RIGHT, INDEX)
                ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
    ΕΑΝ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ

ΑΡΧΗ
    ΔΙΑΒΑΣΕ(P) ;
    ΥΠΟΛΟΓΙΣΕ BINSEARCH(EL, P, 1, N, POS)
ΤΕΛΟΣ
```

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε ένα σύνολο προχωρημένων τεχνικών διαδικασιακού προγραμματισμού. Δόθηκε έμφαση στην ανάπτυξη εσωτερικής δομής σε ένα πρόγραμμα με τη χρήση υπο-προγραμμάτων (διαδικασιών και συναρτήσεων) και αναπτύχθηκαν ζητήματα σχετικά με την εμβέλεια της δήλωσης των μεταβλητών και τους τρόπους περάσματος παραμέτρων.

Στη συνέχεια παρουσιάστηκαν οι εξής τεχνικές:

- *Αναδρομικού προγραμματισμού: Πρόκειται για μία κομψή προγραμματιστική τεχνική, στην οποία ένα πρόγραμμα «καλεί» τον εαυτό του για να επιλύσει ένα πρόβλημα επαναληπτικής φύσης.*
- *Οπισθοδρόμησης: Πρόκειται για μία αναδρομική τεχνική, με την οποία μπορούμε σχετικά γρήγορα να εξετάσουμε ένα μεγάλο σύνολο πιθανών λύσεων και να επιλέξουμε τις ορθές.*
- *Ταξινόμησης και αναζήτησης: Παρουσιάστηκε ένα σύνολο αλγορίθμων με τους οποίους μπορούμε να τοποθετήσουμε σε σειρά ένα σύνολο στοιχείων, ή να αναζητήσουμε ένα συγκεκριμένο στοιχείο μέσα στο σύνολο.*

Με το κεφάλαιο αυτό, ουσιαστικά συμπληρώνεται το θέμα «διαδικασιακός προγραμματισμός». Στο επόμενο και τελευταίο κεφάλαιο του τόμου θα ευαισθητοποιηθείτε (ελπίζω) σε θέματα γενικότερης αντιμετώπισης του προγραμματισμού ως δημιουργικό έργο και όχι ως απλή εργασία ρουτίνας!

Βιβλιογραφία Κεφαλαίου 7

[1] D. Cooper & M. Clancy (1985), Oh! Pascal!. Norton & Company, NY.

[2] D. Knuth (1973), The Art of Computer Programming: Sorting & Searching. Addison–Wesley, Reading:MA.

ΚΕΦΑΛΑΙΟ 8

Ειδικά ζητήματα

Σκοπός

Στο τελευταίο κεφάλαιο του τόμου αναπτύσσονται κάποια εξειδικευμένα ζητήματα προγραμματισμού, η κατανόηση της σημασίας των οποίων απαιτεί καλή γνώση όχι μόνο των προγραμματιστικών τεχνικών, αλλά και της γενικότερης φιλοσοφίας που πρέπει να αναπτύξει ένας προγραμματιστής

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει αυτό το κεφάλαιο, θα μπορείτε να:

- *Αναφέρετε τα δύο είδη τεκμηρίωσης λογισμικού*
- *Περιγράψετε το περιεχόμενο επτά τουλάχιστον εγγράφων που απευθύνονται στους χρήστες του λογισμικού*
- *Αναφέρετε δέκα τουλάχιστον έγγραφα που περιέχει η τεκμηρίωση συστήματος*
- *Περιγράψετε τα τρία είδη της εσωτερικής τεκμηρίωσης*
- *Εξηγήσετε τη διαφορά μεταξύ των εννοιών «έλεγχος λογισμικού» και «εκσφαλμάτωση»*
- *Περιγράψετε τρεις τρόπους διαχείρισης σφαλμάτων*
- *Μετρήσετε την αποδοτικότητα ενός αλγορίθμου*
- *Εξηγήσετε τις επτά περισσότερο συνηθισμένες τιμές αποδοτικότητας*

Έννοιες κλειδιά

- *Τεκμηρίωση χρήστη*
- *Τεκμηρίωση συστήματος*
- *Εσωτερική / εξωτερική τεκμηρίωση*
- *Σχόλια, ελάττωμα, έλεγχος*
- *Εκσφαλμάτωση*
- *Διαχείριση σφαλμάτων*
- *Αποδοτικότητα*
- *Συμβολισμός κεφαλαίου Ο*

Εισαγωγικές παρατηρήσεις

Τα περιεχόμενα του κεφαλαίου αυτού μπορεί αρχικά να φαίνονται ετερόκλητα. Τα συνδέει όμως η πραγματικότητα: Πρόκειται για πολύ σημαντικά ζητήματα, τα οποία έχουν άμεση σχέση με τον προγραμματισμό, αλλά τα οποία οι προγραμματιστές (εκούσια ή ακούσια) τείνουν να αγνοούν.

Το κεφάλαιο αυτό, λοιπόν, δε στοχεύει τόσο να σας μεταδώσει γνώσεις, όσο να σας πείσει να υιοθετήσετε μία ορθή στάση κατά την ανάπτυξη προγραμμάτων. Η στάση αυτή, ως μέρος μίας γενικότερης φιλοσοφίας, αρνείται να θυσιάσει την ποιότητα υπέρ της ποσότητας, αρνείται να δεχθεί τη γρήγορη και πρόχειρη λύση με το επιχείρημα της εξοικονόμησης χρόνου και πόρων, αρνείται να δεχθεί ότι οι προγραμματιστές προτιμούν να «βάλουν το κεφάλι κάτω και να γράψουν τον κώδικα» αντί να αναζητήσουν και να αξιολογήσουν την καλύτερη λύση.

Η εμπειρία μου από την διαχείριση έργων ανάπτυξης λογισμικού με οδήγησε στο συμπέρασμα ότι καλύτερα να «πληρώνει» κανείς ό,τι χρειάζεται να «πληρωθεί» νωρίς στον κύκλο ανάπτυξης, γιατί αργότερα το κόστος θα είναι πολλαπλάσιο. Είναι προτιμότερο, λοιπόν:

- *Να παραχθούν όλα τα έγγραφα της τεκμηρίωσης συστήματος (ενότητα 8.1) κατά τις διαδοχικές φάσεις ανάπτυξης, αντί να χρησιμοποιήσουμε κάποιους ανθρώπους στο τέλος του έργου για να τα γράψουν, γιατί (α) διακινδυνεύουμε την εξέλιξη του έργου, αφού οι ομάδες ανάπτυξης δεν θα μπορούν να συνεννοηθούν χωρίς ενδιάμεσα παραδοτέα, και (β) όταν ένα έργο τελειώσει, κανείς δεν θέλει να ασχοληθεί ξανά μαζί του.*
- *Να σχολιαστεί ο κώδικας του λογισμικού καθώς αυτός αναπτύσσεται, ώστε να είναι δυνατή η μετέπειτα συντήρηση του συστήματος.*
- *Να δοκιμαστεί συστηματικά το λογισμικό και να γραφεί κώδικας διαχείρισης σφαλμάτων (ενότητα 8.2), γιατί κανείς δεν θα χρησιμοποιήσει ένα λογισμικό που δεν δουλεύει σωστά (αφήστε τις άλλες συνέπειες: οικονομικές, εργασιακές, νομικές κ.λπ.).*
- *Να χρησιμοποιήσουμε αποδοτικούς αλγορίθμους (ενότητα 8.3), ώστε να αποφύγουμε σπατάλη χρόνου και πόρων.*

8.1 Τεκμηρίωση

Ένα σύστημα λογισμικού περιλαμβάνει, εκτός από τον κώδικα των προγραμμάτων, και ένα σύνολο από έγγραφα τα οποία παράγονται κατά τις διάφορες φάσεις ανάπτυξης (Sommerville, 1996). Τα έγγραφα αυτά είναι γνωστά ως **«τεκμηρίωση του συστήματος» (system documentation)** και περιγράφουν διάφορα χαρακτηριστικά του συστήματος, απαιτήσεις των χρηστών, αποφάσεις της ομάδας ανάπτυξης, τα αποτελέσματα των ελέγχων κ.λπ.

Η τεκμηρίωση ενός συστήματος είναι δύο ειδών:

- **Εξωτερική (external)**, στην οποία περιλαμβάνονται όλα τα έγγραφα που δημιουργούνται ανεξάρτητα από τον κώδικα, και
- **Εσωτερική (internal)**, η οποία περιλαμβάνει τον σχολιασμό της λειτουργίας των προγραμμάτων που ενσωματώνεται μέσα στον ίδιο τον κώδικα.

Δραστηριότητα 8.1

Αποτελεί λυπηρή διαπίστωση ότι ο μεγαλύτερος όγκος της τεκμηρίωσης ενός συστήματος είναι κακογραμμένος, δυσνόητος και πολλές φορές ελλιπής ή πεπαλαιωμένος. Προσπαθήστε σε 100–150 λέξεις να εξηγήσετε τη σημασία της καλής τεκμηρίωσης, αλλά και τους λόγους στους οποίους πιστεύετε ότι οφείλεται η σημερινή κατάσταση.

Η υιοθέτηση κάποιας μεθοδολογίας για την ανάπτυξη του λογισμικού εγγυάται ότι τα έγγραφα τεκμηρίωσης θα παράγονται κατά τα προβλεφθέντα χρονικά διαστήματα, ώστε να είναι διαθέσιμα όταν τα χρειαστεί κάποια επόμενη δραστηριότητα του κύκλου ανάπτυξης. Αντίθετα, όταν η ανάπτυξη του λογισμικού δεν είναι συστηματική, τα έγγραφα αυτά συνήθως παράγονται όλα μαζί στο τέλος του κύκλου ανάπτυξης, με αποτέλεσμα η ποιότητά τους να είναι χαμηλή και η χρησιμότητά τους περιορισμένη.

Η ποιότητα της τεκμηρίωσης ενός συστήματος είναι το ίδιο σημαντική με την ποιότητα του κώδικά του. Χωρίς πληροφορίες για τον τρόπο χρήσης και κατανόησης των λειτουργιών του συστήματος, η χρησιμότητά του υποβαθμίζεται σημαντικά. Από την άλλη πλευρά, η τεκ-

μηρίωση χαμηλής ποιότητας συνήθως προκαλεί σύγχυση στους χρήστες, με αποτέλεσμα να μη χρησιμοποιείται καθόλου. Είναι γνωστό ότι οι έμπειροι χρήστες ξεκινούν τη χρήση ενός νέου λογισμικού αποφεύγοντας συνειδητά να χρησιμοποιήσουν την προσφερόμενη τεκμηρίωση, αφού προτιμούν να ανακαλύψουν οι ίδιοι τις δυνατότητες του συστήματος!

Η παραγωγή ποιοτικής τεκμηρίωσης δεν είναι ούτε εύκολη, ούτε φθηνή. Για το λόγο αυτό, καλό είναι να υιοθετείται μια τυποποιημένη διαδικασία παραγωγής εγγράφων και να ελέγχεται η ποιότητα της τεκμηρίωσης. Μία καλή λύση είναι η χρήση αυτοματοποιημένων εργαλείων παραγωγής τεκμηρίωσης (μία άλλη είναι να ανατεθεί η τεκμηρίωση σε ξεχωριστή ομάδα μέσα στο έργο). Εσείς μπορείτε να ξεκινήσετε από τις οδηγίες για το ύφος γραφής που περιέχει ο Πίνακας 8.1.

Πίνακας 8.1

Οδηγίες συγγραφής εγγράφων τεκμηρίωσης

• Να χρησιμοποιείτε ενεργητική αντί παθητική φωνή.	• Να μην χρησιμοποιείτε μεγάλες και σύνθετες προτάσεις.
• Να αναφέρετε καταλόγους αντί να γράφετε προτάσεις χαρακτηριστικών.	• Να μην κάνετε αναφορές μόνο με τη χρήση αριθμών.
• Να περιγράφετε με δύο τρόπους τα πολύπλοκα σημεία.	• Να μην φλυαρείτε.
• Να είστε ακριβείς και να ορίζετε τις έννοιες που χρησιμοποιείτε.	• Να χρησιμοποιείτε σύντομες παραγράφους.
• Να χρησιμοποιείτε επικεφαλίδες διαφόρων επιπέδων.	• Να ελέγχετε την συντακτική και λεκτική ορθότητα του κειμένου.

8.1.1 Εξωτερική τεκμηρίωση

Η εξωτερική τεκμηρίωση διακρίνεται σε:

- **τεκμηρίωση για το χρήστη (user documentation)**, στην οποία περιλαμβάνονται όλα τα έγγραφα που περιγράφουν τις λειτουργίες του συστήματος (χωρίς να περιγράφουν τον τρόπο με τον οποίο αυτές υλοποιούνται), τα οποία αναπτύσσονται για να δοθούν στους χρήστες του συστήματος, και

- **τεκμηρίωση για το σύστημα (system documentation)**, η οποία ομαδοποιεί όλα τα έγγραφα που περιγράφουν τεχνικές όψεις του συστήματος, τα οποία παράγονται κατά τις διάφορες φάσεις ανάπτυξης.

Τεκμηρίωση χρήστη

Στην τεκμηρίωση που αναπτύσσεται για το χρήστη πρέπει να περιλαμβάνονται τουλάχιστον τα εξής πέντε έγγραφα:

- **Λειτουργική περιγραφή:** απευθύνεται σε κάποιον που θέλει να εκτιμήσει τις δυνατότητες του συστήματος. Συνοψίζει τις απαιτήσεις από το σύστημα και τους στόχους της ομάδας ανάπτυξης. Αναφέρει τί κάνει και τί δεν κάνει το σύστημα, χρησιμοποιώντας απλά παραδείγματα όπου είναι απαραίτητο. Καλό είναι να περιλαμβάνει πολλά διαγράμματα και σχήματα, χωρίς όμως να μπαίνει σε λεπτομέρειες ή να περιγράφει όλες τις λειτουργίες του συστήματος.
- **Οδηγός εγκατάστασης:** απευθύνεται στους προγραμματιστές συστήματος. Περιέχει οδηγίες για την εγκατάσταση του συστήματος σε κάποιο συγκεκριμένο περιβάλλον λειτουργίας. Πρέπει να περιγράφει τις απαιτήσεις του συστήματος σε υλικό και εξοπλισμό, τα αρχεία που θα εγκατασταθούν στο σκληρό δίσκο, τις παρεμβάσεις και τις ρυθμίσεις που θα κάνει το σύστημα (ή ο προγραμματιστής συστήματος που θα το εγκαταστήσει) στα υπάρχοντα αρχεία διαμόρφωσης, καθώς και τον τρόπο εκκίνησης του λογισμικού
- **Εισαγωγικό εγχειρίδιο:** απευθύνεται στους νέους και άπειρους χρήστες. Περιγράφει με απλά λόγια πώς μπορεί ο χρήστης να ξεκινήσει να χρησιμοποιεί το σύστημα. Καθοδηγεί το χρήστη μέσα από παραδείγματα στις απλές και «κοινές» λειτουργίες που υποστηρίζει το λογισμικό. Παρέχει επίσης τρόπους διαφυγής από απλές προβληματικές καταστάσεις με τις λιγότερες δυνατές απώλειες για τους χρήστες.
- **Εγχειρίδιο αναφοράς:** απευθύνεται στους έμπειρους χρήστες. Πρόκειται για το πλέον καθοριστικό έγγραφο όπου περιγράφονται αναλυτικά όλες οι λειτουργίες που υποστηρίζει το σύστημα. Η περιγραφή μπορεί, όπου είναι δυνατό, να γίνει χρησιμοποιώντας

τυπικά ή ημι–τυπικά μοντέλα, αφού είναι αποδεκτή η «θυσία» της ευαναγνωσιμότητας υπέρ της πληρότητας. Περιέχει ακόμη ένα κατάλογο όλων των μηνυμάτων που εμφανίζει το σύστημα και τις καταστάσεις κατά τις οποίες αυτά παράγονται, καθώς και τον τρόπο αντιμετώπισης από την πλευρά του χρήστη και του συστήματος. Αναλυτικά περιεχόμενα και ευρετήριο είναι απολύτως απαραίτητα.

- **Εγχειρίδιο διαχείρισης:** είναι χρήσιμο μόνο για συστήματα που απαιτούν την παρέμβαση ενός διαχειριστή, στον οποίο και απευθύνεται. Περιγράφει τα μηνύματα διαχείρισης που παράγει το λογισμικό, τον τρόπο αντιμετώπισης για καθένα από αυτά και την απαιτούμενη συντήρηση (υλικού και λογισμικού).

Επί πλέον αυτών των πέντε εγγράφων, τα σύγχρονα συστήματα λογισμικού συνήθως συνοδεύονται από:

- κάρτες συνοπτικής αναφοράς λειτουργιών, με τις οποίες οι έμπειροι χρήστες μπορούν «στα γρήγορα» να βρουν τον τρόπο εκτέλεσης κάποιας συγκεκριμένης λειτουργίας,
- άρθρα και εργασίες που έχουν συγγράψει έμπειροι χρήστες και περιγράφουν απόψεις περί την εφαρμογή του συστήματος και την εκτέλεση ασυνήθιστων εργασιών με το λογισμικό,
- βοήθεια σε ηλεκτρονική μορφή (online help), στην οποία ο χρήστης μπορεί να ανατρέχει καθώς χρησιμοποιεί το σύστημα,
- ηλεκτρονικές διευθύνσεις δικτύου, στις οποίες ο εξουσιοδοτημένος χρήστης μπορεί να βρει πληροφορίες για την εξέλιξη του συστήματος ή να ζητήσει βοήθεια ή να ανταλλάξει απόψεις με άλλους χρήστες.

Τεκμηρίωση συστήματος

Η τεκμηρίωση συστήματος περιλαμβάνει όλα τα έγγραφα που περιγράφουν τη μεθοδολογία ανάπτυξης του συστήματος, τα οποία έχουν παραχθεί κατά τις διάφορες φάσεις του κύκλου ανάπτυξης. Σε αυτά περιλαμβάνονται οπωσδήποτε:

- ο ορισμός και οι προδιαγραφές των απαιτήσεων, μαζί με σύντομη επεξήγηση,

- οι συνολικές προδιαγραφές και η αρχιτεκτονική του συστήματος, όπου περιγράφονται και επεξηγούνται τα τμήματα που αποτελούν το λογισμικό,
- για κάθε τμήμα, ένα έγγραφο όπου περιγράφονται οι προδιαγραφές του τμήματος, οι μονάδες που το αποτελούν και οι προδιαγραφές εισόδου/εξόδου και επικοινωνίας αυτών,
- για κάθε μονάδα, ένα έγγραφο όπου περιγράφονται οι είσοδοι, οι έξοδοι, τα τοπικά και σφαιρικά δεδομένα, η επικοινωνία με άλλες μονάδες και ο κώδικας που υλοποιεί τις λειτουργίες της,
- ένα σχέδιο ελέγχου, όπου περιγράφονται όλες οι δοκιμές που σχεδιάστηκαν και εφαρμόστηκαν στις μονάδες του συστήματος,
- ένα σχέδιο ελέγχου συγκρότησης, όπου περιγράφονται όλες οι δοκιμές που σχεδιάστηκαν και εφαρμόστηκαν κατά τη συγκρότηση των τμημάτων από μονάδες,
- ένα σχέδιο ελέγχου αποδοχής, όπου, σε συνεργασία με τους χρήστες, καταγράφονται όλες οι δοκιμές που θα πρέπει να ξεπεράσει το σύστημα πριν γίνει αποδεκτό προς χρήση,
- ένα συνολικό ευρετήριο και ένας κατάλογος των επικαλύψεων ανάμεσα στα έγγραφα αυτά, ώστε να είναι δυνατή η ανεύρεση και η ενσωμάτωση των αλλαγών που θα γίνουν πριν το σύστημα παραδοθεί.

8.1.2 Εσωτερική τεκμηρίωση

Ενώ η εξωτερική τεκμηρίωση απευθύνεται στους χρήστες του λογισμικού και τους διαχειριστές του έργου ανάπτυξης, η εσωτερική τεκμηρίωση απευθύνεται κυρίως στους προγραμματιστές που θα διορθώσουν ή θα βελτιώσουν τον κώδικα. Συνήθως συσχετίζεται με κάθε μονάδα προγράμματος και περιλαμβάνει:

- έναν τυποποιημένο πρόλογο, στον οποίο δίνονται γενικές πληροφορίες για τη μονάδα προγράμματος (ο Πίνακας 8.2 αποτελεί μια ανάλογη πρόταση),
- σχόλια που ενσωματώνονται σε διάφορα σημεία του εκτελέσιμου κώδικα (ο Πίνακας 8.3 συνοψίζει κάποιες οδηγίες για τη συγγραφή των σχολίων),

- τη χρήση αυτο–επεξηγηματικών ονομάτων για μεταβλητές, δομές δεδομένων, διαδικασίες κ.λπ. του προγράμματος.

Στην πραγματικότητα, η μορφή με την οποία θα γράψουμε ένα πρόγραμμα μπορεί να διευκολύνει την κατανόησή του σε μεταγενέστερο στάδιο. Αυτός είναι ο λόγος για τον οποίο στον τόμο αυτό, αλλά και σε όλα τα βιβλία προγραμματισμού, οι εντολές των παρατιθέμενων προγραμμάτων δεν στοιχίζονται όλες στο αριστερό περιθώριο της γραμμής. Όπως θα έχετε προσέξει, φροντίζουμε να στοιχίζουμε στην ίδια νοητή κατακόρυφο όλες τις εντολές που ανήκουν στην ίδια ομάδα ή προγραμματιστική δομή. Άλλες τεχνικές που διευκολύνουν την ανάγνωση ενός προγράμματος είναι η χρήση έντονης γραφής για τις δεσμευμένες λέξεις της γλώσσας, η έμφαση στις επικεφαλίδες (υπο)προγραμμάτων, η χρήση παρενθέσεων στις πολύπλοκες αριθμητικές παραστάσεις και τις συνθήκες, η παρεμβολή κενών γραμμών στον κώδικα κ.ά.

Πίνακας 8.2

Τυπικός πρόλογος τεκμηρίωσης μονάδας προγράμματος

Ονοματεπώνυμο προγραμματιστή	Λειτουργίες
Ημερομηνία μεταγλώττισης	Αλγόριθμοι που χρησιμοποιούνται
Ονοματεπώνυμο διορθωτή	Παράμετροι και τύποι
Ημερομηνία τροποποίησης	Ισχυρισμοί εισόδου
Σκοπός τροποποίησης	Ισχυρισμοί εξόδου
Περιορισμοί χρονισμού	Σφαιρικές μεταβλητές
Παρενέργειες	Κύριες δομές δεδομένων
Διαχείριση σφαλμάτων	Καλούμενα υπο–προγράμματα
Παραδοχές	Καλούντα υπο–προγράμματα

Πίνακας 8.3
Οδηγίες σχολιασμού του κώδικα

- Ελαχιστοποιήστε την ανάγκη σχολίων ενσωματωμένων στον κώδικα χρησιμοποιώντας προλόγους τεκμηρίωσης, δομημένο προγραμματισμό, αυτο-επεξηγηματικά ονόματα των μερών του προγράμματος, κ.λπ.
- Συσχετίστε σχόλια με τα πιο σημαντικά τμήματα του κώδικα, όπως αυτά που εκτελούν σημαντικές λειτουργίες στα δεδομένα, περιέχουν την εντολή GOTO, διαχειρίζονται σφάλματα, κ.λπ.
- Κατά το σχολιασμό, χρησιμοποιήστε ορολογία που σχετίζεται με το αντικείμενο του προβλήματος
- Χρησιμοποιήστε κενές γραμμές και εσοχές
- Σχολιάστε κάθε μεταγενέστερη προσθήκη ή τροποποίηση του αρχικού κώδικα
- Αντί να χρησιμοποιείτε μακροσκελή σχόλια για να περιγράψετε ένα πολύπλοκο τμήμα κώδικα, καλύτερα να ξαναγράψετε τον κώδικα
- Διασφαλίστε ότι ο κώδικας και τα αντίστοιχα σχόλια συμφωνούν τόσο μεταξύ τους, όσο και με τις απαιτήσεις και τις προδιαγραφές του λογισμικού

Άσκηση αυτοαξολόγησης 8.1

Συμπληρώστε τα κενά στο κείμενο που ακολουθεί χρησιμοποιώντας κάποιες από τις ακόλουθες φράσεις (προσοχή: δεν χρειάζεται να χρησιμοποιήσετε όλες τις φράσεις):

Εσωτερική, εξωτερική, ενδιάμεση, χρήστη, σύστημα, διαχειριστή, πελάτη, λειτουργική περιγραφή, εγχειρίδιο αναφοράς, οδηγός συντήρησης, εγγύηση, οδηγός εγκατάστασης, εισαγωγικό εγχειρίδιο, συνοπτική κάρτα λειτουργιών, εγχειρίδιο διαχείρισης, προδιαγραφές, συμφωνίες, αρχιτεκτονική, τμήματα, δεδομένα, μεταβλητές, ολοκλήρωσης, ελέγχου, πρόλογος, πίνακας, σχόλια, σημεία ελέγχου, σύντομων, αυτο-επεξηγηματικών.

Η …… τεκμηρίωση περιλαμβάνει τα έγγραφα που δεν σχετίζονται με τον κώδικα, δηλαδή αυτά που περιγράφουν τις λειτουργίες του συστήματος και απευθύνονται στους χρήστες (τα

έγγραφα αυτά καλούνται «τεκμηρίωση για το ……») και αυτά που περιγράφουν τεχνικές όψεις του συστήματος και απευθύνονται στην ομάδα διαχείρισης ή συντήρησης του συστήματος (τα έγγραφα αυτά καλούνται «τεκμηρίωση για το ……»).

Στα έγγραφα που απευθύνονται στους χρήστες πρέπει να περιλαμβάνεται μία συνοπτική περιγραφή των δυνατοτήτων και των λειτουργιών του συστήματος (……) και οδηγίες για την εγκατάσταση του συστήματος (……). Ακόμη, χρειάζεται ένα εγχειρίδιο για τους νέους χρήστες (……), ένα αναλυτικότερο εγχειρίδιο για έμπειρους χρήστες (……) και ένα εγχειρίδιο για την αντιμετώπιση διάφορων προβλημάτων (……). Στα έγγραφα περιγραφής του συστήματος περιλαμβάνονται όλα τα έγγραφα που παρήχθησαν κατά την ανάπτυξη του συστήματος και περιέχουν τις ……, την ……, τα …… και τις διαδικασίες ……

Η …… τεκμηρίωση σχετίζεται με τμήματα του κώδικα. Για καθένα από αυτά περιλαμβάνει ένα ……, ο οποίος περιγράφει γενικά το τμήμα, και …… σε διάφορα σημεία του πίνακα. Βοηθά αρκετά και η χρήση …… ονομάτων στις μεταβλητές του τμήματος.

8.2 Διαχείριση λαθών

Στην ιδανική περίπτωση, ένα πρόγραμμα λογισμικού που παραδίδεται προς χρήση δεν θα πρέπει να παρουσιάζει σφάλματα εκτέλεσης. Στην πράξη, αυτό είναι σχεδόν αδύνατο, ιδιαίτερα όταν πρόκειται για ένα μεγάλο σύστημα λογισμικού. Οι προγραμματιστές λένε ότι «ο αριθμός των λαθών που απομένουν σε ένα πρόγραμμα είναι ανάλογος με τον αριθμό των λαθών που έχουν ήδη ανακαλυφθεί και διορθωθεί».

Για την ανακάλυψη των λαθών χρησιμοποιούνται διάφορες τεχνικές (με το θέμα ασχολείται διεξοδικά η Θ.Ε. «Εγκυροποίηση Λογισμικού»), από τις οποίες η πιο διαδεδομένη είναι ο έλεγχος του προγράμματος (program testing). Κατά τον έλεγχο του προγράμματος, αυτό εκτελείται χρησιμοποιώντας δεδομένα που προσομοιάζουν όσο το δυνατό καλύτερα τα αληθινά δεδομένα που θα χρησιμοποιεί το

λογισμικό όταν παραδοθεί προς χρήση. Από τη μελέτη των πραγματικών εξόδων του προγράμματος (και τη σύγκριση με τις αναμενόμενες εξόδους) είναι δυνατή η ανακάλυψη λαθών ή παραλείψεων.

Τα σφάλματα που θα βρεθούν σε ένα πρόγραμμα λογισμικού πρέπει να απαλειφθούν. Η διαδικασία αυτή καλείται εκσφαλμάτωση (debugging – αναφέρεται και ως «αποσφαλμάτωση»). Μετά τη διόρθωση των λαθών, το λογισμικό πρέπει να δοκιμαστεί ξανά.

- Πριν προχωρήσουμε, θεωρώ σκόπιμο να εξηγήσουμε τη διαφορά ανάμεσα στον έλεγχο και τη εκσφαλμάτωση ενός προγράμματος:

  Εκσφαλμάτωση (debugging) είναι το σύνολο των διορθώσεων που κάνουμε σ' ένα πρόγραμμα πριν θεωρήσουμε ότι είναι εντελώς έτοιμο προς χρήση. Πρόκειται για μια απόπειρα να απαλλαγούμε από ελαττώματα του προγράμματος που μας είναι γνωστά.

  Ελεγχος (testing) είναι το σύνολο των ενεργειών που εκτελούμε με ένα πρόγραμμα με στόχο να το αναγκάσουμε να τερματίσει τη λειτουργία του απότομα ή με μη προβλεπόμενο τρόπο (δηλαδή, να «κολλήσει», όπως συνηθίζουν να λένε οι προγραμματιστές). Στην πραγματικότητα, προσπαθούμε να «αποδείξουμε» ότι το πρόγραμμα έχει και άλλα λάθη.

  Να ξεκαθαρίσουμε κάτι πρώτα: ο έλεγχος ενός προγράμματος μπορεί μόνο να δείξει ότι υπάρχουν λάθη. Δεν μπορεί να αποδείξει ότι δεν υπάρχουν!

  Να ξεκαθαρίσουμε και κάτι άλλο: σε ένα πρόγραμμα που έχουμε γράψει εμείς, συνήθως άλλος κάνει τον έλεγχο και εμείς την εκσφαλμάτωση!

Όσο εξαντλητικοί και συστηματικοί και αν είναι οι έλεγχοι που θα κάνουμε σε ένα πρόγραμμα, δυστυχώς δεν μπορούμε να αποδείξουμε ότι ο κώδικας δεν περιέχει **ελαττώματα (bugs)**. Όταν εκτελεστεί μία ελαττωματική εντολή, τότε θα προκληθεί **σφάλμα εκτέλεσης (error)**, ή ακόμη και **αστοχία (failure)** του λογισμικού. Γενικά, χρησιμοποιούμε τους όρους «λάθος», «σφάλμα» ή «εξαίρεση» (exception) για να

αναφερθούμε σε ένα μη αναμενόμενο γεγονός που συνέβη κατά την εκτέλεση ενός προγράμματος.

Ο σωστός χειρισμός των εξαιρέσεων αυξάνει την αξιοπιστία του προγράμματος. Ετσι, εάν η ομάδα ανάπτυξης μπορέσει να προβλέψει την πιθανότητα εμφάνισης ενός λάθους, τότε πρέπει να ενσωματώσει στο πρόγραμμα ειδικό κώδικα για να το διαχειριστεί (exception handling). Αφού το πρόγραμμα διαγνώσει την εμφάνιση του λάθους, καλεί τον κώδικα διαχείρισης του λάθους, ο οποίος:

- εάν το λάθος είναι σοβαρό, φροντίζει για τον ελεγχόμενο τερματισμό της εκτέλεσης του προγράμματος, ώστε να περιοριστούν οι απώλειες για το χρήστη, ενώ
- εάν το λάθος μπορεί να αντιμετωπιστεί, προσπαθεί να διορθώσει την κατάσταση ώστε να συνεχιστεί κανονικά η εκτέλεση του προγράμματος.

Όμως, εάν δεν υπάρχει κώδικας χειρισμού για κάποιο λάθος, ο έλεγχος της εκτέλεσης μεταφέρεται στο λειτουργικό σύστημα, το οποίο προσπαθεί να αντιμετωπίσει το σφάλμα, τερματίζοντας τις περισσότερες φορές την εκτέλεση του προγράμματος.

Σχόλιο μελέτης

Στην ενότητα 3.3.3 αναφερθήκαμε στον αμυντικό προγραμματισμό ως ένα στυλ προγραμματισμού που προσπαθεί να αντιμετωπίσει τις αναμενόμενες αστοχίες ενός προγράμματος. Στην ενότητα 6.6 αναφέρεται και ένα παράδειγμα χειρισμού λάθους με τη χρήση της εντολής GOTO. Θα ήταν χρήσιμο στο σημείο αυτό να θυμηθείτε ξανά τις δύο αυτές ενότητες.

■ Η διαχείριση λαθών μπορεί να γίνει με πολλούς τρόπους:

- Χρησιμοποιώντας την εντολή GOTO για να μεταφέρουμε τον έλεγχο σε κάποιο ειδικό τμήμα κώδικα, όπως κάναμε στην ενότητα 6.6.
- Χρησιμοποιώντας ειδικές μεταβλητές σε κάθε υπο-πρόγραμμα, οι οποίες λειτουργούν ως «δείκτες σφαλμάτων» (error indicators). Ανάλογα με την τιμή της μεταβλητής, το καλόν πρόγραμμα μπορεί να συμπεράνει εάν η εκτέλεση του υπο–προ-

γράμματος έγινε κανονικά ή παρουσιάσθηκε κάποιο σφάλμα.

- Χρησιμοποιώντας εγγενείς δομές χειρισμού λαθών που παρέχει η γλώσσα προγραμματισμού.

Θα σας συμβούλευα να αποφύγετε την πρώτη λύση για τους λόγους που αναφέρθηκαν στην ενότητα 6.6. Η δεύτερη τεχνική είναι πολύ διαδεδομένη σε γλώσσες προγραμματισμού που δεν παρέχουν εγγενείς μηχανισμούς διαχείρισης λαθών. Παρουσιάζει όμως προβλήματα:

- Εάν ο δείκτης σφάλματος είναι σφαιρική μεταβλητή, υπάρχει ο κίνδυνος να εμφανιστούν παρενέργειες,
- Μπορεί το υπο–πρόγραμμα όπου παρουσιάσθηκε το λάθος να βρίσκεται χαμηλά σε μία ιεραρχία υπο–προγραμμάτων (φωλιασμένα υπο–προγράμματα). Εάν το σφάλμα είναι σοβαρό, ίσως έχει επιπτώσεις και στην εκτέλεση των εξωτερικών υπο–προγραμμάτων. Σε τέτοια περίπτωση δεν είναι ξεκάθαρο ποιο υπο–πρόγραμμα πρέπει να χειριστεί το λάθος,
- Σε κάθε περίπτωση, χρειάζεται να γραφεί ένας σημαντικός όγκος κώδικα, ο οποίος θα χειρίζεται τα αναμενόμενα σφάλματα. Εκτός της επιβράδυνσης που προκαλεί στην εκτέλεση του προγράμματος, κανείς δεν μπορεί να εγγυηθεί ότι ο κώδικας αυτός δεν έχει ελαττώματα!

Η τρίτη λύση είναι η προτιμότερη, γι' αυτό και οι περισσότερες σύγχρονες γλώσσες προγραμματισμού παρέχουν τέτοιες δομές. Έτσι κι εμείς, θα επεκτείνουμε τον ψευδοκώδικα που χρησιμοποιούμε, υιοθετώντας μια προσέγγιση παρόμοια με αυτή της Ada, ώστε μέσα σε κάθε υπο-πρόγραμμα να επιτρέπεται:

- Ο ορισμός μεταβλητών τύπου EXCEPTION
- Η καταχώρηση τιμής TRUE σε μια τέτοια μεταβλητή η οποία θα προκαλεί την ενεργοποίηση του διαχειριστή λαθών με την εντολή:

  ΜΕ-ΕΞΑΙΡΕΣΗ (μεταβλητή exception) **ΟΤΑΝ** (συνθήκη λάθους)

- Η παράθεση τμήματος κώδικα χειρισμού του λάθους αμέσως μετά την εντολή αυτή, ή σε άλλο σημείο του προγράμματος, στη μορφή:

ΟΤΑΝ (μεταβλητή exception) **ΚΑΝΕ**

Οδηγίες χειρισμού του λάθους

ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ

Σημειώστε ότι εάν δεν υπάρχει κώδικας χειρισμού του σφάλματος μέσα στο υπο-πρόγραμμα όπου αυτό συνέβη, ο χειρισμός θα μεταβιβάζεται στο καλών πρόγραμμα

- Μετά την εκτέλεση του κώδικα διαχείρισης του λάθους, ο έλεγχος επιστρέφει στην οδηγία μετά από αυτή που προκάλεσε το λάθος, εκτός και αν το λάθος είναι σοβαρό, οπότε μέσα στον κώδικα χειρισμού προβλέπεται ο απότομος τερματισμός της εκτέλεσης του προγράμματος με την οδηγία **ΔΙΑΚΟΠΗ**

Παράδειγμα 8.1 Ας ξαναγράψουμε τον αλγόριθμο του παραδείγματος 8 του κεφαλαίου 6 χρησιμοποιώντας την παραπάνω δομή χειρισμού λαθών αντί της GOTO. Ο νέος αλγόριθμος είναι ο ακόλουθος:

ΑΛΓΟΡΙΘΜΟΣ ΕΛΑΧΙΣΤΟ-ΥΨΟΣ-ΒΡΟΧΗΣ-ΜΕ-ΕΞΑΙΡΕΣΗ

```
ΔΕΔΟΜΕΝΑ
        WEEK, DAY, I, K: INTEGER;
        RAIN: ARRAY [1..52, 1..7] OF INTEGER;
        NEGRAIN, ZERORAIN: EXCEPTION ;
ΑΡΧΗ
        WEEK := 1 ;
        DAY := 1 ;
        ΓΙΑ I := 1 ΕΩΣ 52 ΕΠΑΝΕΛΑΒΕ
                ΓΙΑ K := 1 ΕΩΣ 7 ΕΠΑΝΕΛΑΒΕ
                        ΜΕ-ΕΞΑΙΡΕΣΗ (ZERORAIN) ΟΤΑΝ (RAIN[I,K] = 0) ;
                        ΜΕ-ΕΞΑΙΡΕΣΗ (NEGRAIN) ΟΤΑΝ (RAIN[I,K] < 0) ;
                        ΕΑΝ (RAIN[I,K] < RAIN[WEEK,DAY]) ΤΟΤΕ
                                WEEK := I ;
                                DAY := K
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ
        ΓΙΑ-ΤΕΛΟΣ ;
```

```
ΟΤΑΝ (NEGRAIN) ΚΑΝΕ
        ΤΥΠΩΣΕ("ΒΡΕΘΗΚΕ ΑΡΝΗΤΙΚΗ ΤΙΜΗ ΚΑΤΑΧΩΡΗΜΕΝΗ ΣΤΗΝ
        ΗΜΕΡΑ", Κ, "ΤΗΣ ΕΒΔΟΜΑΔΑΣ", Ι) ;
        RAIN[I,K] := - RAIN[I,K] ;
        ΤΥΠΩΣΕ("ΓΙΑ ΝΑ ΣΥΝΕΧΙΣΤΕΙ Η ΕΚΤΕΛΕΣΗ, Η ΑΡΝΗΤΙΚΗ
        ΤΙΜΗ ΜΕΤΑΤΡΑΠΗΚΕ ΣΕ ΘΕΤΙΚΗ") ;
ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ ;

ΟΤΑΝ (ZERORAIN) ΚΑΝΕ
        ΤΥΠΩΣΕ("ΤΟ ΕΛΑΧΙΣΤΟ ΥΨΟΣ ΒΡΟΧΗΣ ΤΟΥ ΕΤΟΥΣ ΗΤΑΝ",
        RAIN[WEEK,DAY], "ΕΚΑΤΟΣΤΑ ΚΑΙ ΣΗΜΕΙΩΘΗΚΕ ΓΙΑ ΠΡΩΤΗ
        ΦΟΡΑ ΤΗΝ ΗΜΕΡΑ", DAY, "ΤΗΣ ΕΒΔΟΜΑΔΑΣ", WEEK) ;
        ΔΙΑΚΟΠΗ
ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ
```

ΤΕΛΟΣ

Όταν ανακαλυφθεί αρνητική τιμή στο ύψος βροχής, ενεργοποιείται η EXCEPTION μεταβλητή NEGRAIN, η οποία προκαλεί την ενεργοποίηση του αντίστοιχου κώδικα χειρισμού. Αυτός μετατρέπει την αρνητική τιμή σε θετική και επιτρέπει την συνέχιση της εκτέλεσης του προγράμματος, θεωρώντας ότι το σφάλμα δεν είναι σοβαρό.

Εάν όμως εντοπιστεί μηδενικό ύψος βροχής, ενεργοποιείται η EXCEPTION μεταβλητή ZERORAIN, η οποία τελικά προκαλεί διακοπή της εκτέλεσης του προγράμματος. Μπορεί αυτό να σας φαίνεται περίεργο (εξάλλου δεν είναι λάθος να μη βρέξει καθόλου κάποια μέρα του έτους), αλλά στην πραγματικότητα πρόκειται για έναν έμμεσο τρόπο να διακόψουμε την εκτέλεση του προγράμματος την πρώτη φορά που θα συναντήσουμε μηδενικό ύψος βροχής, αφού αποκλείεται να συναντήσουμε στη συνέχεια χαμηλότερη τιμή.

Δραστηριότητα 8.2

Ανατρέξτε στη μελέτη περίπτωσης του κεφαλαίου 6 και βοηθήστε το Βύρωνα να γράψει έναν αλγόριθμο για να υπολογίζει την ποσότητα ενός συγκεκριμένου είδους που διαθέτει το υποκατάστημα της CHILDWARE το οποίο βρίσκεται στην πόλη σας (ο αλγόριθμος θα διαβάζει από το πληκτρολόγιο το είδος και την πόλη). Ο αλγόριθμος θα υπολογίζει ακόμη την ποσότητα του είδους που διαθέτουν το προηγούμενο και το επόμενο υποκατάστημα, όπως αυτά έχουν τοποθετηθεί στον πίνακα των υποκαταστημάτων. Φροντίστε να ενσωματώσετε κώδικα διαχείρισης των πιθανών σφαλμάτων. Έπειτα, συγκρίνετε την απάντησή σας με τη δική μας πρόταση.

Υπόδειξη: Λάβετε υπόψη σας την περίπτωση να εισαχθούν λανθασμένα στοιχεία από το πληκτρολόγιο, αλλά και τις περιπτώσεις πιθανών σφαλμάτων τα οποία μπορεί να εμφανιστούν όταν χρησιμοποιούμε πίνακες (ενότητα 6.4.1).

Σίγουρα υπάρχουν πολλοί διαφορετικοί αλγόριθμοι για το συγκεκριμένο πρόβλημα, οπότε μην ανησυχείτε εάν ο αλγόριθμός σας διαφέρει από αυτόν που προτείνεται εδώ. Η προτεινόμενη απάντηση επιλέχθηκε κυρίως για να δείξει τον τρόπο χειρισμού των αναμενόμενων λαθών και ικανοποιεί τα χαρακτηριστικά του τμηματοποιημένου προγραμματισμού.

ΑΛΓΟΡΙΘΜΟΣ ΠΟΣΟΤΗΤΑ-ΕΙΔΟΥΣ-ΑΝΑ-ΥΠΟΚΑΤΑΣΤΗΜΑ

```
ΣΤΑΘΕΡΕΣ
        STORES = 10 ;
        ITEMS = 50 ;
ΔΕΔΟΜΕΝΑ
        STOCK: ARRAY [1..STORES, 1..ITEMS] OF INTEGER ;
```

ΔΙΑΔΙΚΑΣΙΑ ΕΙΣΑΓΩΓΗ(%SEEKSTORE, %SEEKITEM)

```
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
        ΕΞΟΔΟΣ
                SEEKSTORE, SEEKITEM: INTEGER ;
```

```
ΔΕΔΟΜΕΝΑ
        COR: BOOLEAN ;
        WRONGSTORE, WRONGITEM: EXCEPTION ;
ΑΡΧΗ
        ΕΠΑΝΕΛΑΒΕ
                ΤΥΠΩΣΕ("ΠΛΗΚΤΡΟΛΟΓΕΙΣΤΕ ΤΟΝ ΚΩΔΙΚΟ ΤΟΥ ΥΠΟΚΑΤΑΣΤΗΜΑΤΟΣ") ;
                ΔΙΑΒΑΣΕ(SEEKSTORE) ;
                ΤΥΠΩΣΕ("ΠΛΗΚΤΡΟΛΟΓΕΙΣΤΕ ΤΟΝ ΚΩΔΙΚΟ ΤΟΥ ΕΙΔΟΥΣ") ;
                ΔΙΑΒΑΣΕ(SEEKITEM) ;
                COR := FALSE ;
                ΜΕ-ΕΞΑΙΡΕΣΗ (WRONGSTORE) ΟΤΑΝ ((SEEKSTORE < 1) OR (SEEKSTORE
                > STORES)) ;
                ΜΕ-ΕΞΑΙΡΕΣΗ (WRONGITEM) ΟΤΑΝ ((SEEKITEM < 1) OR (SEEKITEM >
                ITEMS) ;
        ΜΕΧΡΙ (COR = FALSE) ;

        ΟΤΑΝ (WRONGSTORE) ΚΑΝΕ
                ΤΥΠΩΣΕ("Ο ΚΩΔΙΚΟΣ ΤΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ ΕΊΝΑΙ ΛΑΝΘΑΣΜΕΝΟΣ - ΠΡΟ-
                ΣΠΑΘΗΣΤΕ ΞΑΝΑ") ;
                COR := TRUE
        ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ

        ΟΤΑΝ (WRONGITEM) ΚΑΝΕ
                ΤΥΠΩΣΕ("Ο ΚΩΔΙΚΟΣ ΤΟΥ ΕΙΔΟΥΣ ΕΊΝΑΙ ΛΑΝΘΑΣΜΕΝΟΣ - ΠΡΟΣΠΑΘΗ-
                ΣΤΕ ΞΑΝΑ") ;
                COR := TRUE
        ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ
ΤΕΛΟΣ- ΔΙΑΔΙΚΑΣΙΑΣ ;
```

```
ΔΙΑΔΙΚΑΣΙΑ ΕΚΤΥΠΩΣΗ(SEEKSTORE, SEEKITEM, STOCK) ;
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                SEEKSTORE, SEEKITEM: INTEGER ;
                STOCK: ARRAY [1..STORES, 1..ITEMS] OF INTEGER ;
        ΕΞΟΔΟΣ
ΔΕΔΟΜΕΝΑ
```

```
        WRONGSTORE: EXCEPTION ;
ΑΡΧΗ
        ΜΕ-ΕΞΑΙΡΕΣΗ (WRONGSTORE) ΟΤΑΝ ((SEEKSTORE < 1) OR (SEEKSTORE >
        STORES)) ;
        ΤΥΠΩΣΕ("ΤΟ ΚΑΤΑΣΤΗΜΑ", SEEKSTORE, "ΔΙΑΘΕΤΕΙ", STOCK[SEEKSTORE,
        SEEKITEM], "ΚΟΜΜΑΤΙΑ ΤΟΥ ΕΙΔΟΥΣ", SEEKITEM) ;

        ΟΤΑΝ (WRONGSTORE) ΚΑΝΕ
                ΤΥΠΩΣΕ("Ο ΚΩΔΙΚΟΣ ΤΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ ΕΙΝΑΙ ΛΑΝΘΑΣΜΕΝΟΣ") ;
                ΔΙΑΚΟΠΗ
        ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
        ΥΠΟΛΟΓΙΣΕ ΕΙΣΑΓΩΓΗ(SEEKSTORE, SEEKITEM) ;
        ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ(SEEKSTORE, SEEKITEM) ;
        ΤΥΠΩΣΕ("ΠΡΟΗΓΟΥΜΕΝΟ ΚΑΤΑΣΤΗΜΑ") ;
        ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ(SEEKSTORE-1, SEEKITEM) ;
        ΤΥΠΩΣΕ("ΕΠΟΜΕΝΟ ΚΑΤΑΣΤΗΜΑ") ;
        ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ(SEEKSTORE+1, SEEKITEM)
ΤΕΛΟΣ {ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ}
```

Δύο είναι οι αναμενόμενες κατηγορίες λαθών: λανθασμένα στοιχεία στην είσοδο και πρόβλημα με τα όρια του πίνακα STOCK.

Τα αναμενόμενα λάθη της πρώτης περίπτωσης, τα οποία αναφέρονται στην εισαγωγή από το χρήστη λανθασμένου κωδικού υποκαταστήματος ή είδους, αντιμετωπίζονται τοπικά στη διαδικασία ΕΙΣΑΓΩΓΗ, όπου διαβάζονται οι δύο κωδικοί. Ανάλογα με την περίπτωση, δημιουργείται η εξαίρεση WRONGSTORE ή η εξαίρεση WRONGITEM, οι οποίες ειδοποιούν το χρήστη για το λάθος και του ζητούν να πληκτρολογήσει ξανά τους κωδικούς.

Στη δεύτερη περίπτωση, εμφανίζεται λάθος όταν προσπαθούμε να τυπώσουμε τα στοιχεία:

- του προηγούμενου καταστήματος, όταν ο χρήστης έχει πληκτρολογήσει τον κωδικό του πρώτου καταστήματος,

- του επόμενου καταστήματος, όταν ο χρήστης έχει πληκτρολογήσει τον κωδικό του τελευταίου καταστήματος.

Και στις δύο περιπτώσεις, δημιουργείται η εξαίρεση WRONGSTORE, την οποία διαχειρίζεται τοπικά η διαδικασία ΕΚΤΥΠΩΣΗ. Σημειώστε ότι η εξαίρεση αυτή:

- Είναι διαφορετική από την εξαίρεση με το ίδιο όνομα που εμφανίζεται στη διαδικασία ΕΙΣΑΓΩΓΗ. Όπως έχουμε αναφέρει, για να διαχειριστούμε ένα σφάλμα, δηλώνουμε μια αντίστοιχη μεταβλητή τύπου EXCEPTION, οπότε ισχύουν οι κανόνες της εμβέλειας,
- Προκαλεί τελικά τη διακοπή της εκτέλεσης της διαδικασίας, πριν προσπαθήσει να υπολογίσει το περιεχόμενο της (ανύπαρκτης) θέσης του πίνακα STOCK. Πρόκειται για μια κλασική περίπτωση όπου το πρόγραμμά μας διαχειρίζεται ένα αναμενόμενο σφάλμα πριν αυτό γίνει αντιληπτό από το λειτουργικό σύστημα (και προκληθεί σφάλμα εκτέλεσης).

Δραστηριότητα 8.3

Προσπαθήστε να γράψετε έναν αλγόριθμο, ο οποίος θα αφαιρεί ένα κόμβο από μία δυναμική λίστα που υλοποιεί μια δομή στοίβας. Φροντίστε να ενσωματώσετε κώδικα διαχείρισης των πιθανών σφαλμάτων. Αφού δώσετε την απάντησή σας, μελετήστε και τη δική μας απάντηση, η οποία παρουσιάζεται στη συνέχεια.

Υπόδειξη: να λάβετε υπόψη σας τις περιπτώσεις πιθανών σφαλμάτων τα οποία μπορεί να εμφανιστούν όταν χρησιμοποιούμε λίστες (ενότητα 6.5.4).

Όπως γνωρίζουμε, σε μία δομή στοίβας που υλοποιείται με διασυνδεδεμένη λίστα, αφαιρείται πρώτος ο τελευταίος κόμβος που έχει προστεθεί. Η πρόσθεση ή η αφαίρεση ενός κόμβου γίνεται πάντα από το ίδιο άκρο της λίστας, το οποίο στην περίπτωση της στοίβας καλείται «κορυφή».

Για να διευκολυνθείτε στη μελέτη σας, περιγράφουμε και το εξωτερικό πρόγραμμα, όπου ορίζεται η δομή της στοίβας και ο τύπος των στοιχείων της: κάθε κόμβος περιέχει ένα πεδίο δεδομένων και ένα σύν-

δεσμο τύπου δείκτη. Ακόμη, εκεί ορίζεται και ο δείκτης TOP, ο οποίος δείχνει στην αρχή της στοίβας. Η κλήση της διαδικασίας από το εξωτερικό πρόγραμμα για να αφαιρέσει ένα στοιχείο της στοίβας είναι: POP(TOP, STACK, POPEL, FLAG).

ΑΛΓΟΡΙΘΜΟΣ ΣΤΟΙΒΑ

```
ΤΥΠΟΙ
        EL:     DATA: CHAR ;
                NEXT: POINTER[EL] ;
ΔΕΔΟΜΕΝΑ
        STACK: LIST OF EL ;
        TOP: POINTER[EL];

ΔΙΑΔΙΚΑΣΙΑ POP(%TOP, %STACK, %POPEL, %FLAG)
ΔΙΕΠΑΦΗ
        ΕΙΣΟΔΟΣ
                STACK: LIST OF EL ;
                TOP: POINTER[EL];
        ΕΞΟΔΟΣ
                FLAG: BOOLEAN
                STACK: LIST OF EL ;
                TOP: POINTER[EL];
                POPEL: POINTER[EL] ;
ΔΕΔΟΜΕΝΑ
        EMPTYSTACK: EXCEPTION ;
ΑΡΧΗ
        POPEL := TOP ;
        ΜΕ-ΕΞΑΙΡΕΣΗ (EMPTYSTACK) ΟΤΑΝ (TOP = NIL) ;
        TOP := TOP^.NEXT ;
        POPEL^.NEXT := NIL ;
        FLAG := FALSE ;

        ΟΤΑΝ (EMPTYSTACK) ΚΑΝΕ
                ΤΥΠΩΣΕ("ΔΕΝ ΜΠΟΡΕΙΤΕ ΝΑ ΑΦΑΙΡΕΣΕΤΕ ΣΤΟΙΧΕΙΟ ΑΠΟ
                ΚΕΝΗ ΣΤΟΙΒΑ") ;
                FLAG := TRUE ;
```

ΔΙΑΚΟΠΗ
ΕΞΑΙΡΕΣΗ-ΤΕΛΟΣ
ΤΕΛΟΣ-ΔΙΑΔΙΚΑΣΙΑΣ ;

ΑΡΧΗ {ΕΞΩΤΕΡΙΚΟ ΠΡΟΓΡΑΜΜΑ}
. . .
ΤΕΛΟΣ {ΕΞΩΤΕΡΙΚΟ ΠΡΟΓΡΑΜΜΑ}

Παρατηρήστε ότι όλες οι παράμετροι περνιούνται μέσω διευθύνσεως. Αυτό είναι αναμενόμενο, αφού η διαδικασία αυτή τροποποιεί άμεσα τα περιεχόμενα της δομής στοίβας, αφαιρώντας ένα στοιχείο από αυτή. Μετά την αφαίρεση, ο δείκτης TOP δείχνει στο επόμενο στοιχείο (αν υπάρχει) και η στοίβα περιέχει ένα στοιχείο λιγότερο (εάν δεν ήταν αρχικά κενή). Η διαδικασία επιστρέφει και το στοιχείο που αφαιρέθηκε μέσω του δείκτη POPEL.

Απαραίτητη προϋπόθεση για όσα ακολουθούν είναι η στοίβα να κατασκευάζεται σωστά, δηλαδή:

- ο σύνδεσμος του τελευταίου στοιχείου έχει τιμή NIL,
- όταν η στοίβα είναι κενή, ο δείκτης TOP να έχει τιμή NIL (αυτό διασφαλίζεται εν μέρει από την εντολή TOP := TOP^.NEXT, εάν ισχύει και η προηγούμενη παρατήρηση).

Ας εξετάσουμε τι μπορεί να πάει στραβά σε μια τέτοια διαδικασία. Κατ' αρχήν, εξετάζουμε το πλήθος των στοιχείων της στοίβας. Εάν η στοίβα είναι κενή (οπότε θα ισχύει TOP = NIL), συμβαίνει η εξαίρεση EMPTYSTACK. Ο έλεγχος μεταφέρεται στο κώδικα διαχείρισης του σφάλματος, ο οποίος τυπώνει ένα μήνυμα λάθους και θέτει τη BOOLEAN μεταβλητή FLAG σε τιμή TRUE, ώστε να αντιληφθεί το εξωτερικό πρόγραμμα ότι δεν έγινε η αφαίρεση του στοιχείου. Έπειτα, διακόπτεται η εκτέλεση της διαδικασίας.

Σημειώστε πως θα μπορούσαμε να διαχειριστούμε το σφάλμα και με μια δομή απόφασης του είδους:

ΕΑΝ (TOP < > **NIL**) **ΤΟΤΕ**

<εντολές μη κενής στοίβας>

ΑΛΛΙΩΣ

<εντολές κενής στοίβας>

ΕΑΝ-ΤΕΛΟΣ

Εάν η στοίβα έχει κατασκευαστεί σωστά, ο δείκτης στο στοιχείο που (υποτίθεται ότι) αφαιρούμε παίρνει την τιμή NIL, οπότε είμαστε σίγουροι ότι δεν θα χρησιμοποιηθεί κάποια τυχαία τιμή. Σημειώνω εδώ ότι:

- Το εξωτερικό πρόγραμμα μπορεί να κάνει περαιτέρω διαχείριση του σφάλματος της κενής στοίβας (ανάλογα με το που χρησιμοποιείται η στοίβα), με βάση την τιμή της FLAG. Κάτι τέτοιο προϋποθέτει ότι κατά την κλήση θα ισχύει FLAG = FALSE,
- Εάν η διαχείριση του σφάλματος είναι στην ευθύνη του εξωτερικού προγράμματος, ίσως δεν είναι σωστό να τυπώνει η διαδικασία POP το μήνυμα λάθους, αλλά το εξωτερικό πρόγραμμα.

Εάν η στοίβα δεν είναι κενή, τότε αφαιρείται το πρώτο (κορυφαίο) στοιχείο της, με την εντολή POPEL := TOP, ενώ ο δείκτης TOP δείχνει στο επόμενο στοιχείο της στοίβας, με την εντολή TOP := TOP^.NEXT. Εάν η στοίβα περιέχει μόνο ένα στοιχείο (και έχει κατασκευαστεί σωστά), τότε θα ισχύει TOP^.NEXT := NIL, και η λίστα θα θεωρείται στο εξής κενή. Σε κάθε περίπτωση, η τιμή της FLAG γίνεται FALSE, σημαίνοντας ότι έγινε η αφαίρεση του στοιχείου.

Έπειτα, φροντίζουμε να θέσουμε την τιμή NIL στο σύνδεσμο του κόμβου που αφαιρέθηκε (με την εντολή POPEL^.NEXT := NIL), αφού το στοιχείο αυτό δεν ανήκει πια στη στοίβα. Με τον τρόπο αυτό αποφεύγουμε το πρόβλημα των δεικτών που βρίσκονται σε εκκρεμότητα.

8.3 Αποδοτικότητα

Στην αρχή αυτού του τόμου (ενότητα 2.1) είχα ισχυριστεί ότι οι αλγόριθμοι περιγράφουν τα βήματα που λύνουν ένα πρόβλημα και ότι η έκφραση των αλγορίθμων σε κάποια γλώσσα προγραμματισμού αποτελεί ένα πρόγραμμα. Αμέσως μετά (ενότητα 2.2) ανέφερα ότι, στις περισσότερες περιπτώσεις, δεν αρκεί μόνο να βρούμε κάποιον σωστό αλγόριθμο που λύνει ένα πρόβλημα, αλλά έχει σημασία και η "ποιότητα" του αλγορίθμου.

Δεν πρόκειται, βέβαια, στο τέλος του τόμου να αλλάξω άποψη! Ήρθε όμως η ώρα να συζητήσουμε περισσότερο το θέμα αυτό, δηλαδή την **αποδοτικότητα (efficiency)** των αλγορίθμων.

- Κατ' αρχήν, χρειάζεται να αναγνωρίσουμε ότι πρέπει να ασχοληθούμε με την αποδοτικότητα των αλγορίθμων και όχι των προγραμμάτων: Ενώ είναι βέβαιο ότι ένας βελτιωμένος αλγόριθμος θα οδηγήσει στην ανάπτυξη πιο αποδοτικών προγραμμάτων, το αντίστροφο δεν ισχύει.

Ένας σημαντικός παράγοντας που επηρεάζει την αποδοτικότητα είναι και η στρατηγική που χρησιμοποιούμε για τη σχεδίαση του προγράμματος. Στον τόμο αυτό έχουμε γνωρίσει διάφορες στρατηγικές ανάλυσης (κατά βήμα εκλέπτυνση, σχηματική ανάλυση κ.ά.) και σχεδίασης (σχεδίαση και ανάπτυξη από πάνω προς τα κάτω, από κάτω προς τα πάνω κ.ά.) ενός προγράμματος.

Εκτός από αυτές, υπάρχει και η μέθοδος της **«κατά μέτωπο επίθεσης» (brute force)** στο πρόβλημα, κατά την οποία επαναλαμβάνουμε απλά βήματα πολλές φορές (αντί να σκεφτούμε έναν πιο κομψό αλλά περισσότερο πολύπλοκο τρόπο). Είναι αλήθεια ότι μία τέτοια προσέγγιση βοηθά στη γρήγορη κατανόηση του προβλήματος και οδηγεί γρήγορα σε μία λύση, αλλά έχει το μειονέκτημα ότι συνήθως δεν οδηγεί στην καλύτερη λύση.

Βέβαια, ο πρώτος έλεγχος που κάνουμε σε έναν αλγόριθμο, ανεξάρτητα από τη στρατηγική σχεδίασης που χρησιμοποιήθηκε, είναι και ο πιο κρίσιμος: Δουλεύει; Έπειτα, μπορούμε να ασχοληθούμε με την αποδοτικότητα του αλγορίθμου, δηλαδή με την «επιβάρυνση» που αυτός προκαλεί στον υπολογιστή.

- Ένας αλγόριθμος ή ένα πρόγραμμα είναι αποδοτικά, εάν πετυχαίνουν το στόχο τους κάνοντας την ελάχιστη δυνατή χρήση των υπολογιστικών πόρων, δηλαδή του χώρου και του χρόνου. Ως υπολογιστικός χώρος νοείται η μνήμη (κεντρική ή περιφερειακή) του υπολογιστή. Ως υπολογιστικός χρόνος δεν νοείται ο χρόνος που χρειάζεται ένα πρόγραμμα για να εκτελεστεί σε έναν υπολογιστή (καθώς αυτός εξαρτάται από τον υπολογιστή), αλλά το πλήθος των βημάτων που απαιτούνται για να τερματίσει ο αλγόριθμος.

Μετρούμε την αποδοτικότητα ενός αλγορίθμου με βάση κάποιον παράγοντα N, ο οποίος επηρεάζει σημαντικά τον αριθμό των βημάτων που απαιτούνται μέχρι τον τερματισμό του αλγορίθμου. Για παράδειγμα, τέτοιοι παράγοντες είναι το πλήθος των ψηφίων που πρέπει να εξεταστεί κάθε φορά, το πλήθος των στοιχείων του πίνακα που συγκρίνεται σε κάθε επανάληψη του αλγόριθμου ταξινόμησης κ.ά.

Χρησιμοποιούμε το συμβολισμό του «κεφαλαίου Ο» για να δηλώσουμε την τάξη μεγέθους της αποδοτικότητας ενός αλγορίθμου σε σχέση με τον παράγοντα N. Σημειώστε ότι μας ενδιαφέρει η τάξη μεγέθους και όχι η ακριβής μέτρηση της αποδοτικότητας, γι' αυτό και αγνοούμε σταθερούς παράγοντες με τους οποίους πολλαπλασιάζεται ή αυξάνεται προσθετικά ο N (Cooper, 1985, Knuth, 1973).

Μερικές τυπικές τιμές αποδοτικότητας παρατίθενται στον Πίνακα 8.4.

Παράδειγμα 8.2

Εστω οι ακόλουθοι δύο αλγόριθμοι που υπολογίζουν το άθροισμα των πρώτων N αριθμών αρχίζοντας από το 1:

ΑΛΓΟΡΙΘΜΟΣ ΑΘΡ-ΣΕΙΡΑΣ-ΕΚΔ0

```
ΔΕΔΟΜΕΝΑ
 I, N, SUM: INTEGER ;
ΑΡΧΗ
 ΔΙΑΒΑΣΕ(N) ;
 SUM := 0 ;
 ΓΙΑ I := 1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
  SUM := SUM + I
 ΓΙΑ-ΤΕΛΟΣ ;
 ΤΥΠΩΣΕ(SUM)
ΤΕΛΟΣ
```

ΑΛΓΟΡΙΘΜΟΣ ΑΘΡ-ΣΕΙΡΑΣ-ΕΚΔ1

```
ΔΕΔΟΜΕΝΑ
 N, SUM: INTEGER ;
ΑΡΧΗ
 ΔΙΑΒΑΣΕ(N) ;
 SUM := (1 + N) * (N / 2) ;
 ΤΥΠΩΣΕ(SUM)
ΤΕΛΟΣ
```

Κάθε εκτέλεση του αλγορίθμου ΑΘΡ-ΣΕΙΡΑΣ-ΕΚΔ0 περιλαμβάνει τρεις εντολές που εκτελούνται μία φορά και μία πράξη που εκτελείται N φορές, δηλαδή συνολικά N+3 βήματα. Μπορούμε λοιπόν να ισχυριστούμε ότι η αποδοτικότητα του αλγορίθμου είναι της τάξης μεγέθους N, δηλαδή γραμμική O(N).

Από την άλλη πλευρά, κάθε εκτέλεση του αλγορίθμου ΑΘΡ-ΣΕΙΡΑΣ-ΕΚΔ1 περιλαμβάνει τρεις εντολές που εκτελούνται πάντα μία φορά. Η αποδοτικότητα λοιπόν του αλγορίθμου είναι σταθερή O(3) και ανεξάρτητη του N.

Πίνακας 8.4

Οι περισσότερο συνηθισμένες τιμές της αποδοτικότητας

- **O(1)** Σταθερός χρόνος, ο οποίος δεν εξαρτάται καθόλου από τα δεδομένα
- **O(N)** Γραμμικός χρόνος: εάν διπλασιαστεί ο N, διπλασιάζεται και ο χρόνος εκτέλεσης
- **O($\log_2$N)** Λογαριθμικός χρόνος, ο οποίος αυξάνεται αργά καθώς αυξάνεται ο N: όταν ο N τετραπλασιάζεται, ο χρόνος διπλασιάζεται
- **O(N*$\log_2$N)** Σχεδόν γραμμικός χρόνος: υπονοεί ότι ένας γραμμικός αλγόριθμος καλεί ένα αλγόριθμο χρόνου O($\log_2$N)
- **O(N^2)** Αλγόριθμος τετραγωνικού χρόνου: όταν ο N διπλασιάζεται, ο χρόνος τετραπλασιάζεται
- **O(N^3)** Αλγόριθμος κυβικού χρόνου: όταν ο N διπλασιάζεται, ο χρόνος οκταπλασιάζεται
- **O(2^N)** Εκθετικός χρόνος: όταν ο N είναι 10, ο χρόνος είναι περίπου 1.000, ενώ όταν ο N διπλασιαστεί (γίνει δηλαδή 20), ο χρόνος γίνεται 1.000.000! Οι αλγόριθμοι με τέτοια απόδοση θεωρείται ότι δεν έχουν πρακτική εφαρμογή

Άσκηση αυτοαξιολόγησης 8.2

Δοκιμάστε να υπολογίσετε την αποδοτικότητα των εξής αλγορίθμων του τόμου (σε παρένθεση οι ενότητες του τόμου όπου περιγράφονται οι αλγόριθμοι):

ΑΛΓΟΡΙΘΜΟΣ	ΑΠΟΔΟΤΙΚΟΤΗΤΑ
FIBONACCI (3.1)	
MAX-XYZ (3.2)	
ΑΝΑΣΤΡΟΦΟΣ-ΠΙΝΑΚΑΣ (6.4)	
ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ (6.4)	
ΠΡΟΣΘΕΣΗ-ΚΟΜΒΟΥ-ΣΕ-ΛΙΣΤΑ (6.5)	
ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ (7.4)	
ΤΑΞΙΝΟΜΗΣΗ-ΦΥΣΑΛΙΔΑΣ (7.4)	

Σύνοψη

Στο τελευταίο αυτό κεφάλαιο του τόμου προσπάθησα να σας πείσω για την ανάγκη να είναι κανείς «συνειδητοποιημένος προγραμματιστής» αντί να είναι απλά «συγγραφέας κώδικα». Ο Πραγματικός Προγραμματιστής (ΠΠ) – ένα όν αρκετά σπάνιο στις ημέρες μας – θεωρεί ότι η ανάπτυξη λογισμικού ξεφεύγει από την απλή συγγραφή κώδικα και περιλαμβάνει, μέσα στα άλλα, δραστηριότητες όπως τεκμηρίωση, πρόληψη και διαχείριση σφαλμάτων και μέτρηση της αποδοτικότητας των αλγορίθμων. Τέτοιες δραστηριότητες μπορεί στον αδαή ή τον άπειρο να φαντάζουν σπατάλη χρόνου και πόρων. Ο ΠΠ γνωρίζει όμως ότι είναι ακριβώς αυτά τα χαρακτηριστικά που θα κάνουν τη διαφορά ανάμεσα σε ένα λογισμικό που χρησιμοποιείται και σε ένα που απορρίπτεται, ανάμεσα στην επένδυση και τη σπατάλη, ανάμεσα στην ικανοποίηση και τη δυσαρέσκεια, εν τέλει, ανάμεσα στην επιτυχημένη επαγγελματική σταδιοδρομία και την αποτυχία. Να θυμάστε ότι ο μηχανικός λογισμικού κρίνεται συνεχώς από το αποτέλεσμα και όχι από τις προθέσεις!

Βιβλιογραφία Κεφαλαίου 8

[1] D. Cooper & M. Clancy (1985), Oh! Pascal!, Norton & Company, NY.

[2] D. Knuth (1973), The Art of Computer Programming: Fundamental Algorithms, Addison–Wesley, Reading:MA.

[3] I. Sommerville (1996), Software Engineering, Addison–Wesley.

Επίλογος

Ο τόμος που μόλις τελείωσε αποτελεί μία σύντομη εισαγωγή σε έναν όμορφο κόσμο, ο οποίος κατοικείται από ανθρώπους δημιουργικούς, τους «προγραμματιστές». Επειδή ο προγραμματιστής τις πιο πολλές φορές είναι αντιμέτωπος με τον εαυτό του (και όχι με τον υπολογιστή, όπως λανθασμένα πολλοί πιστεύουν), τείνει να θεωρηθεί αντικοινωνικός. Δεν είναι έτσι!

Ο προγραμματιστής, μπορεί ως «καλλιτέχνης» να μιλά τη δική του προσωπική γλώσσα, αλλά γνωρίζει ως «μηχανικός» να μιλά τη γλώσσα που καταλαβαίνουν όλοι οι συνεργάτες του. Ξέρει να ενσωματώνει στη δουλειά του τα καλύτερα στοιχεία των συναδέλφων και καταθέτει σε κάθε τμήμα λογισμικού που αναπτύσσει μέρος του εαυτού του. Και το κυριότερο: ο προγραμματιστής φροντίζει κάθε φορά να σχεδιάζει και να αναπτύσσει το καλύτερο λειτουργικό λογισμικό (αρχή του ικανού προγραμματιστή).

Είναι μεγάλη η ευθύνη της συγγραφής ενός βιβλίου, το οποίο θα διδάσκει προγραμματισμό σε άπειρους αλλά φιλόδοξους προγραμματιστές. Ήταν αδύνατο να χωρέσει σε ένα μόνο τόμο όλη η γνώση που έχει συσσωρευθεί σχετικά με την ανάπτυξη προγραμμάτων. Η μεγαλύτερη δυσκολία που αντιμετώπισα κατά τη συγγραφή του τόμου δεν ήταν η επιλογή του υλικού, αλλά η απόρριψη όλου του υπόλοιπου υλικού που τελικά δεν συμπεριέλαβα (για παράδειγμα, σε πρόχειρη μορφή έχω ετοιμάσει υλικό που καλύπτει άλλα τρία περίπου κεφάλαια).

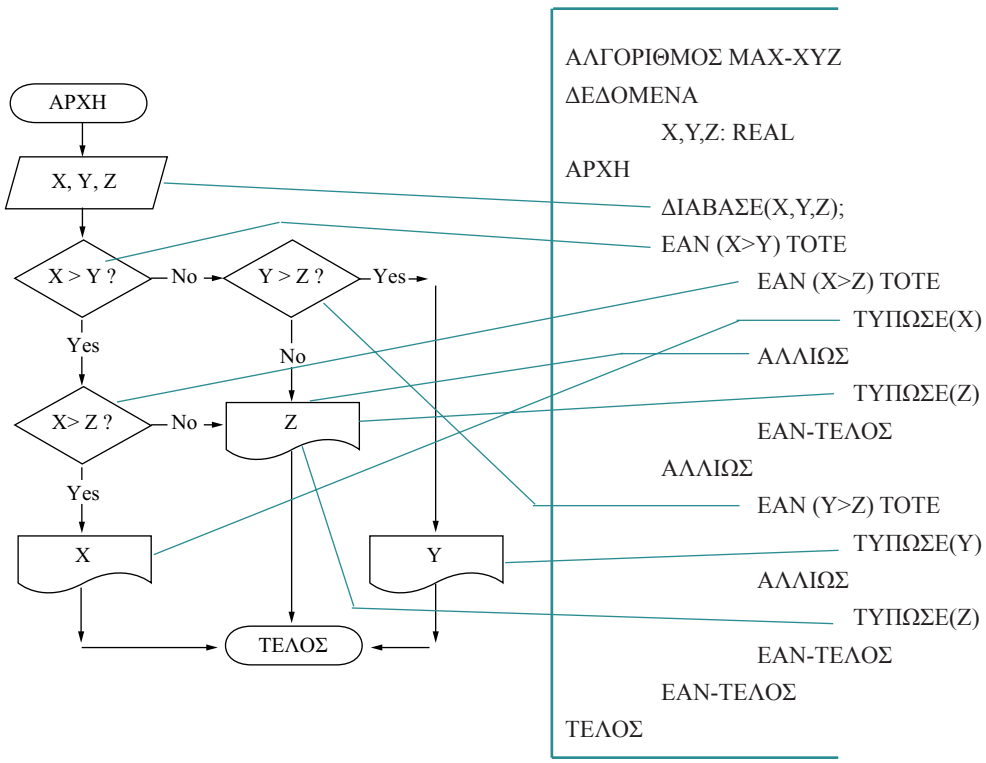
Πολύτιμη ήταν η βοήθεια και η καθοδήγηση του Ακαδημαϊκού Υπεύθυνου της Θεματικής Ενότητα, καθ. Παναγιώτη Πιντέλα, της Κριτικής Αναγνώστριας, καθ. Ελπίδας Κεραυνού–Παπαηλιού και του μέλους της ΜΕΑ του ΕΑΠ. Το ελάχιστο που μπορώ να κάνω είναι να τους ευχαριστήσω θερμά.

Όπως και να έχουν τα πράγματα, ο τόμος αυτός σας άνοιξε την πόρτα ενός καινούριου κόσμου. Στην επόμενη ενότητα περιγράφονται κάποια από τα αξιοθέατα του κόσμου αυτού, τα οποία περιμένουν να τα ανακαλύψετε. Μην αργείτε λοιπόν...

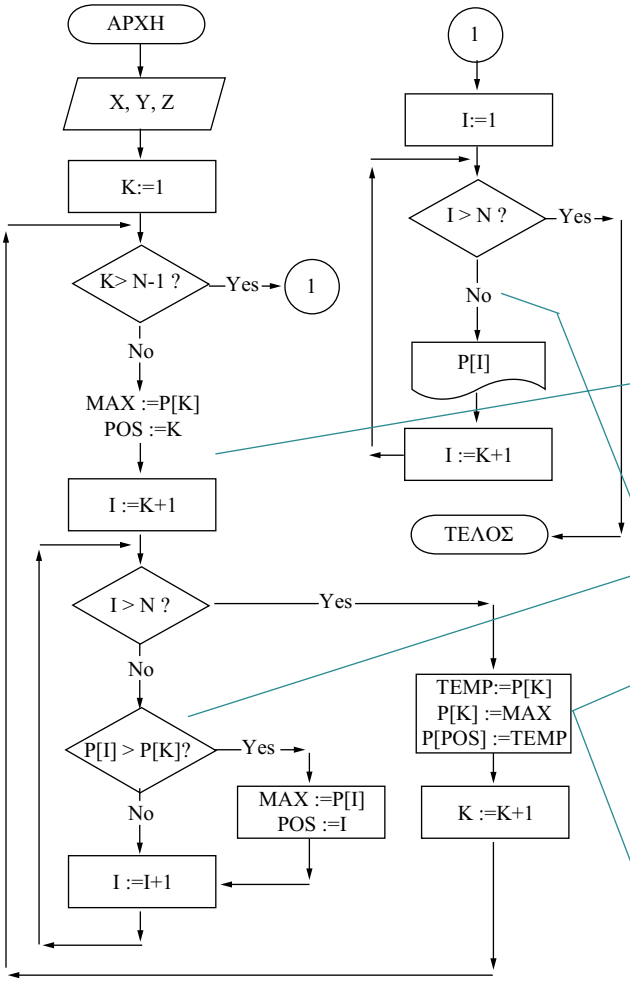
Μπορείτε να επικοινωνείτε μαζί μου στη διεύθυνση kameas@math.upatras.gr. Τα σχόλια και οι παρατηρήσεις επί του έργου είναι περισσότερο από ευπρόσδεκτες

Απαντήσεις ασκήσεων αυτοαξιολόγησης 2.1

Στη συνέχεια, παρουσιάζονται τα σωστά ΔΡΠ και οι αντίστοιχοι αλγόριθμοι με τη μορφή σχολίων. Διακεκομμένες γραμμές έχουν χρησιμοποιηθεί για να συνδέσουν οπτικά κάποια τμήματα των αλγορίθμων με αντίστοιχα τμήματα των ΔΡΠ.



Στο προηγούμενο ΔΡΠ παρατηρήστε την αναπαράσταση των φωλιασμένων δομών απόφασης. Στα δύο επόμενα ΔΡΠ προσέξτε τη χρήση των συνδέσεων εντός σελίδας και των σχολίων, η οποία βοηθά την ευκρίνεια των ΔΡΠ. Βέβαια, τα ΔΡΠ θα μπορούσαν να είχαν σχεδιαστεί σε ένα τμήμα. Ακόμη, παρατηρήστε την αναπαράσταση των δομών επανάληψης.

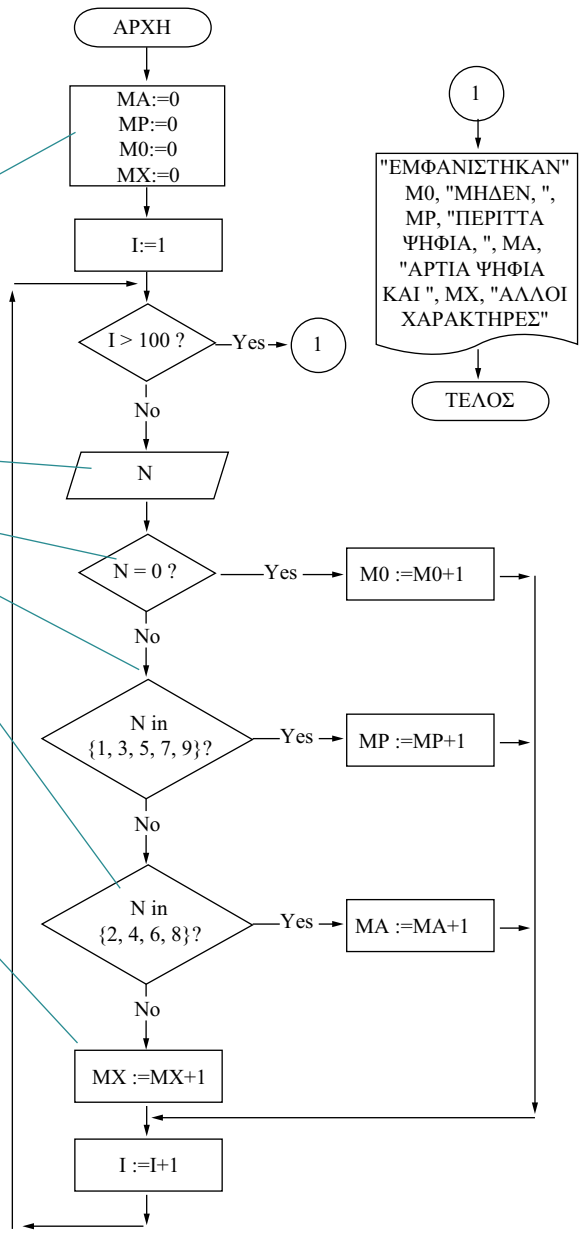



```
ΑΛΓΟΡΙΘΜΟΣ ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ
ΔΕΔΟΜΕΝΑ
        I,K,N,MAX,POS,TEMP: INTEGER;
        P: ARRAY [1..N] OF INTEGER;
ΑΡΧΗ
        ΓΙΑ K:=1 ΕΩΣ N-1 ΕΠΑΝΕΛΑΒΕ
                MAX:=P[K];
                POS:=K;
                ΓΙΑ I:=K+1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
                        ΕΑΝ (P[I]>P[K]) ΤΟΤΕ
                                MAX:=P[I];
                                POS:=I
                        ΕΑΝ-ΤΕΛΟΣ
                ΓΙΑ-ΤΕΛΟΣ;
                TEMP:P[K];
                P[K]:=MAX;
                P[POS]:TEMP
        ΓΙΑ-ΤΕΛΟΣ;
        ΓΙΑ I:=1 ΕΩΣ N ΕΠΑΝΕΛΑΒΕ
                ΤΥΠΩΣΕ(P[I])
        ΓΙΑ-ΤΕΛΟΣ
ΤΕΛΟΣ
```

ΕΝΑΛΛΑΓΗ ΑΡΙΘΜΩΝ
ΚΑΙ ΑΠΟΘΗΚΕΥΣΗ
ΜΕΓΑΛΥΤΕΡΟΥ

```
ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΡΗΣΗ-ΨΗΦΙΩΝ
ΔΕΔΟΜΕΝΑ
    N: CHAR;
    I, MA, MP, M0, MX: INTEGER;
    P: ARRAY [1..N] OF INTEGER;
ΑΡΧΗ
    MA=MP=MX=M0:=0;
    ΓΙΑ I:=1 ΕΩΣ 100 ΕΠΑΝΕΛΑΒΕ
        ΔΙΑΒΑΣΕ(N);
        ΕΑΝ (N=0) ΤΟΤΕ;
            M0:=M0+1
        ΑΛΛΙΩΣ ΕΑΝ (N in{1,3,5,7,9}) ΤΟΤΕ
                MP:=MP+1
            ΑΛΛΙΩΣ ΕΑΝ (N in{2,4,6,8}) ΤΟΤΕ
                    MA:=MA+1
                ΑΛΛΙΩΣ
                    MX:=MX+1
                ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ;
ΤΥΠΩΣΕ(«ΕΜΦΑΝΙΣΤΗΚΑΝ», M0, «ΜΗΔΕΝ», MA,
«ΑΡΤΙΑ ΨΗΦΙΑ», MP, «ΠΕΡΙΤΤΑ ΨΗΦΙΑ ΚΑΙ», MX,
«ΑΛΛΟΙ ΧΑΡΑΚΤΗΡΕΣ»)

ΤΕΛΟΣ
```



Εάν τα καταφέρατε και στα τρία διαγράμματα, συγχαρητήρια! Η άσκηση αυτή έχει στόχο να σας βοηθήσει να καταλάβετε πώς προκύπτει ένα ΔΡΠ από τον αλγόριθμο ενός προγράμματος. Φαίνεται πώς έχετε κατανοήσει τον τρόπο χρήσης των δύο τεχνικών.

Εάν δεν τα καταφέρατε και τόσο καλά, προσέξτε. Για να απαντήσετε σωστά χρειάζεται να κατανοήσετε το συμβολισμό και των δύο τεχνι-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

κών, όπως παρουσιάζεται στις ενότητες 2.3 και 2.4. Ίσως να συναντήσετε δυσκολίες στην αναπαράσταση ολόκληρων προγραμματιστικών δομών (π.χ. δομή απόφασης, δομή επανάληψης). Μην ανησυχείτε όμως! Κάτι τέτοιο είναι αναμενόμενο σε αυτό το στάδιο. Τα πράγματα θα ξεκαθαρίσουν περισσότερο στο κεφάλαιο 6, όπου παρουσιάζεται ο δομημένος προγραμματισμός.

2.2

Ο αλγόριθμος αυτός μπορεί να θεωρηθεί ως επέκταση του αλγορίθμου ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ-ΨΗΦΙΩΝ, αφού διαβάζει στην είσοδο μια ακολουθία 100 χαρακτήρων και μετρά τη συχνότητα εμφάνισης καθενός από τα ψηφία 0 έως 9. Αντί για ξεχωριστούς μετρητές, αυτή τη φορά χρησιμοποιήσαμε ένα πίνακα ακεραίων, Μ, 10 θέσεων (ουσιαστικά, κάθε θέση του πίνακα χρησιμοποιείται ως μετρητής για το αντίστοιχο με το δείκτη της ψηφίο). Η πρώτη δομή επανάληψης χρησιμοποιείται για την αρχικοποίηση του πίνακα Μ. Αφού τελειώσει με την επεξεργασία της εισόδου, ο αλγόριθμος καλεί τον αλγόριθμο ΜΟΠΙΝΑΝΑ-1ΧΝ για να υπολογίσει το μέσο όρων των συχνοτήτων εμφάνισης των ψηφίων.

Η σωστή περιγραφή του αλγόριθμου με ψευδοκώδικα, όπως προκύπτει από το ΔΡΠ της άσκησης, έχει ως εξής:

ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΡΗΣΗ-ΨΗΦΙΩΝ

```
ΔΕΔΟΜΕΝΑ
    N: CHAR ;
    I: INTEGER ;
    M: ARRAY[0..9] OF INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 0 ΕΩΣ 9 ΕΠΑΝΕΛΑΒΕ
        M[I] := 0 ;
    ΓΙΑ-ΤΕΛΟΣ
    ΓΙΑ I := 1 ΕΩΣ 100 ΕΠΑΝΕΛΑΒΕ
    ΔΙΑΒΑΣΕ(N);
        ΕΑΝ (N = 0) ΤΟΤΕ
            M[0] := M[0]+1
        ΑΛΛΙΩΣ ΕΑΝ (N = 1) ΤΟΤΕ
```

```
                    M[1] := M[1]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 2) ΤΟΤΕ
                    M[2] := M[2]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 3) ΤΟΤΕ
                    M[3] := M[3]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 4) ΤΟΤΕ
                    M[4] := M[4]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 5) ΤΟΤΕ
                    M[5] := M[5]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 6) ΤΟΤΕ
                    M[6] := M[6]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 7) ΤΟΤΕ
                    M[7] := M[7]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 8) ΤΟΤΕ
                    M[8] := M[8]+1
                ΑΛΛΙΩΣ ΕΑΝ (N = 9) ΤΟΤΕ
                    M[9] := M[9]+1
                ΕΑΝ-ΤΕΛΟΣ
            ΕΑΝ-ΤΕΛΟΣ
        ΕΑΝ-ΤΕΛΟΣ
    ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΕΑΝ-ΤΕΛΟΣ
ΓΙΑ-ΤΕΛΟΣ ;
ΥΠΟΛΟΓΙΣΕ ΜΟ-ΠΙΝΑΚΑ-1ΧΝ(Μ)
ΤΕΛΟΣ
```

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Για τη μέτρηση των συχνοτήτων χρησιμοποιήσαμε 10 φωλιασμένες δομές απόφασης (ΕΑΝ-ΑΛΛΙΩΣ), μία για κάθε διαφορετικό ψηφίο. Αυτός δεν είναι ο κομψότερος προγραμματιστικά τρόπος. Στην ενότητα 6.2.2 θα περιγραφεί μια καλύτερη έκδοση του αλγορίθμου, η οποία χρησιμοποιεί δομή πολλαπλής επιλογής, ενώ μια ακόμη καλύτερη έκδοση «εκμεταλλεύεται» τη δομή πίνακα, ως εξής:

ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΡΗΣΗ-ΨΗΦΙΩΝ-ΕΚΔ1

```
ΔΕΔΟΜΕΝΑ
    N: CHAR ;
    I: INTEGER ;
    M: ARRAY[0..9] OF INTEGER ;
ΑΡΧΗ
    ΓΙΑ I := 0 ΕΩΣ 9 ΕΠΑΝΕΛΑΒΕ
        M[I] := 0 ;
    ΓΙΑ-ΤΕΛΟΣ
    ΓΙΑ I := 1 ΕΩΣ 100 ΕΠΑΝΕΛΑΒΕ
        ΔΙΑΒΑΣΕ(N);
        ΕΑΝ (N >= 0 AND N <= 9) ΤΟΤΕ
            M[N] := M[N] + 1
        ΕΑΝ-ΤΕΛΟΣ
    ΓΙΑ-ΤΕΛΟΣ ;
    ΥΠΟΛΟΓΙΣΕ ΜΟ-ΠΙΝΑΚΑ-1ΧΝ(M)
ΤΕΛΟΣ
```

Ο αλγόριθμος αυτός χρησιμοποιεί το χαρακτήρα N που διαβάζει κάθε φορά στην είσοδο ως δείκτη στον πίνακα M (ο έλεγχος στη δομή ΕΑΝ εξασφαλίζει ότι μόνο οι τιμές του N που αντιστοιχούν σε ψηφία θα χρησιμοποιηθούν ως δείκτες) και αυξάνει κάθε φορά κατά 1 την τιμή που περιέχεται στην αντίστοιχη θέση.

Εάν τα καταφέρατε στη «μετάφραση» του ΔΡΠ, συγχαρητήρια! Η άσκηση αυτή λειτουργεί συμπληρωματικά προς την προηγούμενη και έχει στόχο να σας βοηθήσει να καταλάβετε πώς προκύπτει ο ψευδοκώδικας από το ΔΡΠ ενός προγράμματος. Φαίνεται, λοιπόν, πως έχετε κατανοήσει τον τρόπο χρήσης των δύο τεχνικών.

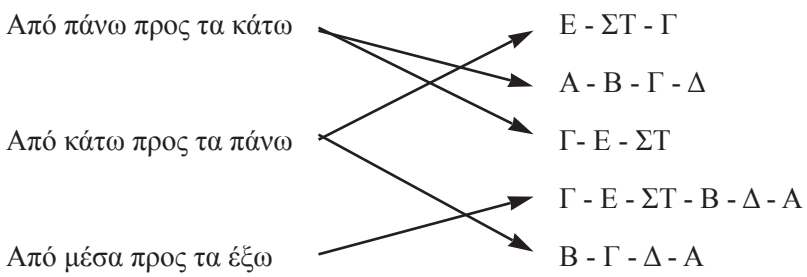
Εάν συναντήσατε δυσκολίες, μην απογοητεύεστε! Είναι γεγονός ότι

Απαντήσεις ασκήσεων αυτοαξιολόγησης

το ΔΡΠ είναι κάπως «κακογραμμένο», αφού είναι σπασμένο σε τρία τμήματα. Διαβάστε πάλι την ενότητα 2.4 για να δείτε τη χρήση των συνδέσεων εντός σελίδας και προσπαθήστε να «ανακατασκευάσετε» το ΔΡΠ σε ένα τμήμα. Ίσως τώρα, με μία ακόμη επανάληψη της ενότητας 2.3, η άσκηση σας φανεί πιο εύκολη! Μην εγκαταλείψετε την προσπάθεια, αφού θα συναντήσετε παρόμοια προβλήματα στο κεφάλαιο 6.

4.1

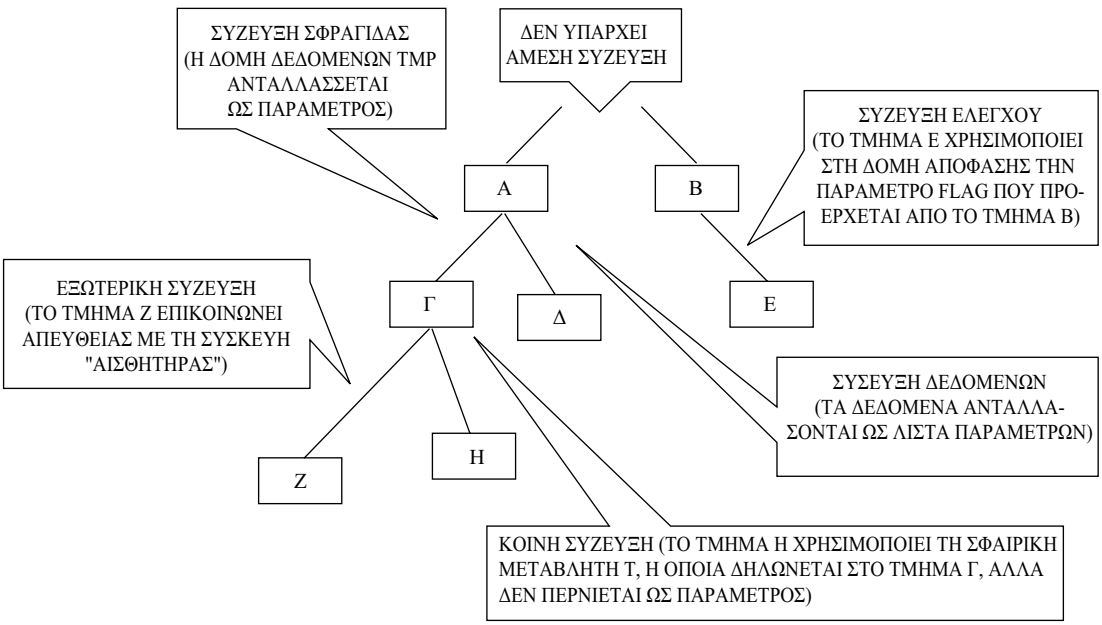
Η σωστή απάντηση ακολουθεί:



Εάν έχετε απαντήσει σωστά, συγχαρητήρια. Έχετε κατανοήσει τις διαφορές των μεθοδολογιών σχεδίασης. Εάν απαντήσατε λάθος, δεν πειράζει. Μια επανάληψη της ενότητας 4.2 θα σας βοηθήσει να αναγνωρίσετε και να διορθώσετε τα λάθη σας. Ένας πρακτικός τρόπος να τα καταφέρετε καλύτερα είναι να προσέξετε τη θέση του σημείου εκκίνησης της σχεδίασης σε σχέση με το διάγραμμα του συστήματος.

4.2 Απαντήσεις ασκήσεων αυτοαξιολόγησης

Οι σωστές απαντήσεις και η αιτιολόγηση φαίνονται στο ακόλουθο σχήμα:



Εάν απαντήσατε σωστά και στα έξι κουτάκια, συγχαρητήρια! Έχετε κατανοήσει τις περισσότερο συχνά εμφανιζόμενες περιπτώσεις σύζευξης, οπότε είναι σχεδόν βέβαιο ότι θα αποφύγετε τέτοια λάθη όταν θα προγραμματίζετε. Εάν όμως δεν τα καταφέρατε, προσέξτε! Η σύζευξη οδηγεί σε προγράμματα κακής ποιότητας, τα οποία είναι δύσκολο να ελεγχθούν και να συντηρηθούν. Επειδή όμως κάποιοι βαθμοί σύζευξης είναι επιθυμητοί, καλύτερα να ξαναδιαβάσετε την ενότητα 4.4.2 και να προσπαθήσετε πάλι.

6.1

Η δήλωση

```
P: ARRAY[1..M,1..N] OF INTEGER
```

Υποδηλώνει ότι ο πίνακας P:

- είναι δύο διαστάσεων,

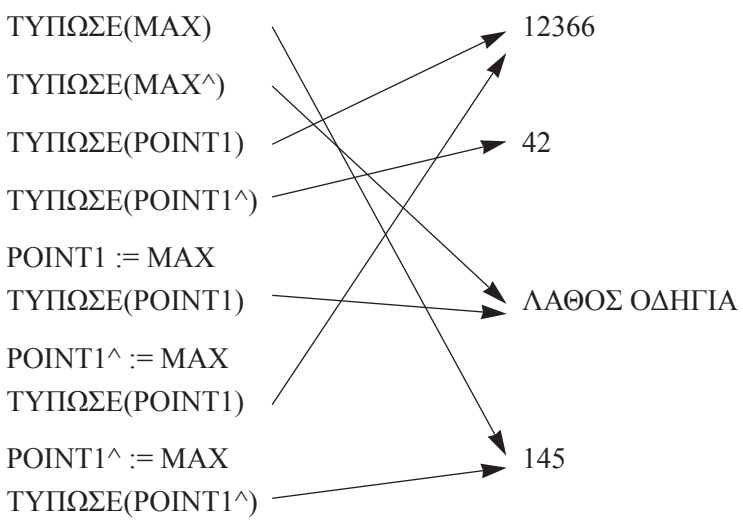
Απαντήσεις ασκήσεων αυτοαξιολόγησης

- έχει M γραμμές και N στήλες,
- μπορεί συνολικά να αποθηκεύσει M*N στοιχεία,
- όλα τα στοιχεία αυτά είναι ακέραιοι αριθμοί,
- η αναφορά σε κάποιο από αυτά γίνεται δίνοντας τη γραμμή και τη στήλη στην οποία βρίσκεται (π.χ., P[2,3]).

Εάν απαντήσατε σωστά, μπράβο σας! Έχετε κατανοήσει τον τρόπο που δηλώνονται οι πίνακες και το είδος των πληροφοριών που μας δίνει μια τέτοια δήλωση. Εάν δεν τα καταφέρατε, προσοχή! Ήταν μια σχετικά εύκολη ερώτηση, γι' αυτό καλύτερα να ξαναμελετήσετε την αρχή της ενότητας 6.4.

6.2

Οι σωστές αντιστοιχίσεις δείχνονται στη συνέχεια:



Προσέξτε τη χρήση του τελεστή έμμεσης προσπέλασης (dereferencing operator): Η τιμή της μεταβλητής POINT1 είναι η θέση μνήμης 12366, ενώ η τιμή της μεταβλητής POINT1^ είναι το περιεχόμενο της θέσης μνήμης 12366, δηλαδή 42. Η μεταβλητή MAX δεν είναι δείκτης, άρα οι οδηγίες ΤΥΠΩΣΕ(MAX^) και POINT1 := MAX είναι λάθος (υπάρχει ασυμβατότητα στους τύπους δεδομένων των μεταβλητών). Αντί-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

θετα, είναι ορθή η εντολή POINT1^ := MAX, η οποία τοποθετεί ως τιμή της θέσης μνήμης όπου δείχνει ο POINT1 (δηλαδή της 12366) την τιμή της μεταβλητής MAX. Τη νέα τιμή μάς την τυπώνει η οδηγία ΤΥΠΩΣΕ(POINT1^) και όχι η ΤΥΠΩΣΕ(POINT1). Να θυμάστε, όμως, ότι συνήθως, δεν μπορούμε να τυπώσουμε την τιμή ενός δείκτη, αλλά μόνο την τιμή της θέσης στην οποία αυτός δείχνει.

Η άσκηση αυτή είναι πραγματικά δύσκολη, γιατί οι έννοιες του δείκτη και της έμμεσης προσπέλασης είναι δυσνόητες ακόμη και για έμπειρους προγραμματιστές. Εάν απαντήσατε σωστά, σας αξίζουν πολλά «μπράβο». Εάν απαντήσατε λάθος, πάλι σας αξίζει ένα «μπράβο» γιατί προσπαθήσατε. Σε κάθε περίπτωση, όμως, ρίξτε άλλη μια ματιά στην ενότητα 6.5.1, αλλά και στο σχετικό κεφάλαιο της Θ.Ε. «Δομές Δεδομένων».

7.1

Οι σωστές απαντήσεις είναι:

1. Για το υπο-πρόγραμμα Β, η Χ είναι: (α) τοπική μεταβλητή
2. Για το υπο-πρόγραμμα Β, η Υ είναι: (β) σφαιρική μεταβλητή
3. Για το υπο-πρόγραμμα Γ, η Ζ είναι: (α) τοπική μεταβλητή
4. Μέσα στο υπο-πρόγραμμα Α, ο τύπος της Ζ είναι: (δ) δεν ορίζεται
5. Μέσα στο υπο-πρόγραμμα Δ, ο τύπος της Ζ είναι: (β) real

Εάν απαντήσατε σωστά, συγχαρητήρια! Έχετε κατανοήσει την έννοια της τοπικής και της σφαιρικής μεταβλητής. Εάν δεν τα καταφέρατε, μελετήστε με μεγαλύτερη προσοχή την ενότητα 7.1.2.

7.2

Η σωστή απάντηση είναι η εξής:

	ΜΕΣΩ ΔΙΕΥΘΥΝΣΕΩΣ	**ΜΕ ΤΙΜΗ**
ΥΠΟΛΟΓΙΣΕ ΕΙΣΟΔΟΣ ΑΡΙΘΜΩΝ		
Α	Χ	
Β	Χ	
ΥΠΟΛΟΓΙΣΕ ΑΝΤΙΣΤΡΟΦΗ–ΣΕΙΡΑΣ		
Α	Χ	
Β	Χ	
ΥΠΟΛΟΓΙΣΕ ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ		
Χ		Χ
Ψ		Χ

Οδηγούμαστε στις απαντήσεις αυτές από το γεγονός ότι οι δύο πρώτες διαδικασίες επιστρέφουν τιμή σε σφαιρικές μεταβλητές, ενώ η τρίτη απλά τυπώνει τους αριθμούς. Συνεπώς, ο σωστός τρόπος γραφής της επικεφαλίδας κάθε διαδικασίας είναι ο εξής:

ΔΙΑΔΙΚΑΣΙΑ ΕΙΣΟΔΟΣ-ΑΡΙΘΜΩΝ(%Α, %Β)

ΔΙΑΔΙΚΑΣΙΑ ΑΝΤΙΣΤΡΟΦΗ-ΣΕΙΡΑΣ(%Α, %Β)

ΔΙΑΔΙΚΑΣΙΑ ΕΚΤΥΠΩΣΗ-ΑΡΙΘΜΩΝ(Χ, Υ)

Εάν συμπληρώσατε τον πίνακα σωστά, τότε μπράβο! Έχετε κατανοήσει τους δύο μηχανισμούς περάσματος παραμέτρων. Εάν δεν τα καταφέρατε, μην ανησυχείτε. Σίγουρα με πιο προσεκτική μελέτη της ενότητας 7.1.3 θα τα καταφέρετε την επόμενη φορά.

7.3

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Η σωστή απάντηση είναι η εξής:

	ΤΟΠΙΚΗ	ΣΦΑΙΡΙΚΗ
FIRST		X
A	X	
TEMP	X	
Y	X	

Αρκεί να παρατηρήσετε το σημείο όπου δηλώνεται η κάθε μεταβλητή. Θα δείτε ότι μόνο η FIRST δηλώνεται στο κυρίως πρόγραμμα, ενώ όλες οι υπόλοιπες δηλώνονται μέσα σε κάποια διαδικασία. Εάν τα καταφέρατε, συγχαρητήρια! Έχετε κατανοήσει το μηχανισμό λειτουργίας των υπο-προγραμμάτων. Εάν αποτύχατε, καλύτερα να επαναλάβετε την ενότητα 7.1, η οποία περιέχει ένα πολύ βασικό κομμάτι προγραμματιστικής γνώσης.

8.1

Το κείμενο, όταν τα κενά του συμπληρωθούν σωστά, έχει ως εξής:

Η *εξωτερική* τεκμηρίωση περιλαμβάνει τα έγγραφα που δεν σχετίζονται με τον κώδικα, δηλαδή αυτά που περιγράφουν τις λειτουργίες του συστήματος και απευθύνονται στους χρήστες (τα έγγραφα αυτά καλούνται «τεκμηρίωση για το *χρήστη*») και αυτά που περιγράφουν τεχνικές όψεις του συστήματος και απευθύνονται στην ομάδα διαχείρισης ή συντήρησης του συστήματος (τα έγγραφα αυτά καλούνται «τεκμηρίωση για το *σύστημα*»).

Στα έγγραφα που απευθύνονται στους χρήστες πρέπει να περιλαμβάνεται μία συνοπτική περιγραφή των δυνατοτήτων και των λειτουργιών του συστήματος (*λειτουργική περιγραφή*), και οδηγίες για την εγκατάσταση του συστήματος (*οδηγός εγκατάστασης*). Ακόμη, χρειάζεται ένα εγχειρίδιο για τους νέους χρήστες (*εισαγωγικό εγχειρίδιο*), ένα αναλυ-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

τικότερο εγχειρίδιο για έμπειρους χρήστες (*εγχειρίδιο αναφοράς*) και ένα εγχειρίδιο για την αντιμετώπιση διάφορων προβλημάτων (*εγχειρίδιο διαχείρισης*). Στα έγγραφα περιγραφής του συστήματος περιλαμβάνονται όλα τα έγγραφα που παρήχθησαν κατά την ανάπτυξη του συστήματος και περιέχουν τις *προδιαγραφές*, την *αρχιτεκτονική*, τα *δεδομένα* και τις διαδικασίες *ελέγχου*.

Η *εσωτερική* τεκμηρίωση σχετίζεται με τμήματα του κώδικα. Για καθένα από αυτά περιλαμβάνει ένα *πρόλογο*, ο οποίος περιγράφει γενικά το τμήμα, και *σχόλια* σε διάφορα σημεία του πίνακα. Βοηθά αρκετά και η χρήση *αυτο-επεξηγηματικών* ονομάτων στις μεταβλητές του τμήματος.

Οι λέξεις *ενδιάμεση, διαχειριστή, πελάτη, οδηγός συντήρησης, εγγύηση, συνοπτική κάρτα λειτουργιών, συμφωνίες, τμήματα, μεταβλητές, ολοκλήρωσης, πίνακας, σημεία ελέγχου, σύντομων* δεν ταιριάζουν σε κανένα κενό.

Εάν απαντήσατε σωστά σε όλα τα κενά, σας αξίζουν συγχαρητήρια! Έχετε καταλάβει τα είδη τεκμηρίωσης και το περιεχόμενο κάθε εγγράφου. Εάν αστοχήσατε σε κάποια κενά, καλύτερα να επαναλάβετε την ενότητα 8.1.

8.2

Ο πίνακας πρέπει να συμπληρωθεί ως εξής:

ΑΛΓΟΡΙΘΜΟΣ	ΑΠΟΔΟΤΙΚΟΤΗΤΑ
FIBONACCI (3.1)	O(1)
MAX-XYZ (3.2)	O(N)
ΑΝΑΣΤΡΟΦΟΣ-ΠΙΝΑΚΑΣ (6.4)	$O(N^2)$
ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ (6.4)	$O(N^3)$
ΠΡΟΣΘΕΣΗ-ΚΟΜΒΟΥ-ΣΕ-ΛΙΣΤΑ (6.5)	O(N)
ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ (7.4)	$O(N^2)$
ΤΑΞΙΝΟΜΗΣΗ-ΦΥΣΑΛΙΔΑΣ (7.4)	$O(N^2)$

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Ο αλγόριθμος MAX-XYZ εκτελεί κάθε φορά 15 εντολές, ανεξάρτητα από τα δεδομένα εισόδου, άρα η αποδοτικότητά του είναι σταθερή.

Ο αλγόριθμος FIBONACCI περιέχει μία δομή επανάληψης που εκτελείται N-2 φορές, άρα η αποδοτικότητά του είναι περίπου N.

Ο αλγόριθμος ΑΝΑΣΤΡΟΦΟΣ-ΠΙΝΑΚΑΣ περιέχει δύο φωλιασμένες επαναλήψεις από 1 έως Μ, άρα η αποδοτικότητά του είναι της τάξης N^2. Αντίστοιχα, ο αλγόριθμος ΓΙΝΟΜΕΝΟ-ΠΙΝΑΚΩΝ περιέχει τρεις φωλιασμένες δομές επανάληψης, άρα η αποδοτικότητά του είναι της τάξης N^3.

Ο αλγόριθμος ΠΡΟΣΘΕΣΗ-ΚΟΜΒΟΥ-ΣΕ-ΛΙΣΤΑ διατρέχει όλους τους κόμβους της λίστας μέχρι να προσθέσει ένα νέο κόμβο στο τέλος της, άρα η αποδοτικότητά του είναι ανάλογη με το πλήθος των κόμβων.

Ο αλγόριθμος ΤΑΞΙΝΟΜΗΣΗ-ΜΕ-ΕΠΙΛΟΓΗ περιέχει μία δομή επανάληψης, η οποία κάθε φορά εκτελείται για ένα βήμα λιγότερο, καθώς σε κάθε επανάληψη ταξινομείται και ένα στοιχείο του πίνακα. Συνεπώς, για N στοιχεία, η επανάληψη θα εκτελεστεί συνολικά N+(N-1)+(N-2)+...+1 φορές, οπότε η αποδοτικότητα του αλγορίθμου είναι (N(N+1))/2, δηλαδή περίπου N^2. Με το ίδιο σκεπτικό, συμπεραίνουμε ότι και ο αλγόριθμος ΤΑΞΙΝΟΜΗΣΗ-ΦΥΣΑΛΙΔΑΣ είναι της τάξης N^2.

Η άσκηση αυτή είναι δύσκολη και απαιτεί ιδιαίτερη προσοχή. Εάν τα καταφέρατε, μπράβο σας. Έχετε κατανοήσει τι σημαίνει «αποδοτικότητα» και το σημαντικότερο, μπορείτε να την εφαρμόσετε στην πράξη. Εάν αποτύχατε, προσέξτε: Οι πέντε πρώτοι αλγόριθμοι είναι σχετικά απλοί, οπότε η αποτυχία οφείλεται μάλλον σε απροσεξία. Μελετήστε την ενότητα 8.3 και προσπαθήστε πάλι. Ενδεχόμενη αποτυχία στους τελευταίους δυο αλγορίθμους μάλλον οφείλεται σε ανεπαρκή κατανόηση του μηχανισμού των αλγορίθμων. Μαζί με την ενότητα 8.3, καλύτερα να μελετήσετε και τις ενότητες όπου περιγράφονται οι αλγόριθμοι αυτοί, πριν προσπαθήσετε ξανά να απαντήσετε στην άσκηση.

ΠΡΟΤΕΙΝΟΜΕΝΗ ΒΙΒΛΙΟΓΡΑΦΙΑ ΓΙΑ ΠΑΡΑΠΕΡΑ ΜΕΛΕΤΗ

Στον τόμο αυτό, ο προγραμματισμός των υπολογιστών προσεγγίστηκε ως τμήμα της ευρύτερης προσπάθειας για την ανάπτυξη ορθού και ποιοτικού λογισμικού, η οποία είναι γνωστή ως Τεχνολογία Λογισμικού (Software Engineering). Τα περισσότερα εργαλεία σχεδίασης και αναπαράστασης λογισμικού που περιγράφονται στα κεφάλαια 2 και 5, οι έννοιες που σχετίζονται με τη σημασία της σχεδίασης προγραμμάτων και το ρόλο της στο συνολικό κύκλο ανάπτυξης λογισμικού (κεφάλαιο 4) καθώς και οι διάφορες πρακτικές προγραμματισμού (κεφάλαιο 3) περιέχονται κυρίως σε βιβλία Τεχνολογίας Λογισμικού. Δύο από τα καλύτερα βιβλία στο αντικείμενο αυτό είναι τα:

Software Engineering (European edition) του Ian Sommerville (εκδ. Addison–Wesley, 1996)

Software Engineering: a practitioner's approach του Roger Pressman (εκδ. McGraw–Hill, 1994)

Αν και σε καθένα από αυτά αφιερώνεται μόνο ένα ή δύο κεφάλαια για τον προγραμματισμό και την ανάπτυξη λογισμικού, ο ενδιαφερόμενος αναγνώστης θα βρει πολλά στοιχεία σχετικά με τον κύκλο ζωής του λογισμικού, τα οποία θα τον βοηθήσουν να αποκτήσει σφαιρικότερη αντίληψη για την Τεχνολογία Λογισμικού. Σας τα συστήνω ανεπιφύλακτα!

Εάν δυσκολεύεστε στην ανάγνωση ξενόγλωσσων κειμένων, μπορείτε αντί αυτών να μελετήσετε το εξίσου καλό βιβλίο του **Μανώλη Σκορδαλάκη (Εισαγωγή στην Τεχνολογία Λογισμικού, εκδόσεις Συμμετρία)**.

Υπάρχει και το βιβλίο **«Σχεδίαση Προγράμματος (δομημένη και αλγοριθμική) των Ν. Ιωαννίδη, Κ. Μαρινάκη και Σ. Μπακογιάννη (εκδόσεις Νέων Τεχνολογιών – Ελιξ)**, το οποίο επικεντρώνεται στη σχεδίαση αλγορίθμων με τη χρήση Διαγραμμάτων Προγράμματος και ψευδοκώδικα. Αν το στυλ γραφής του είναι κάπως παρωχημένο, ο αναγνώστης που θα μελετήσει το βιβλίο αυτό προσεκτικά, θα βρει κάτω από την επιφάνεια ένα σύνολο χρήσιμων αλγορίθμων, για καθένα από τους οποίους οι συγγραφείς παρουσιάζουν τα βήματα σχεδίασης και υλοποίησης με διδακτικό τρόπο.

Όσον αφορά στον ίδιο τον προγραμματισμό, συνηθίζεται να παρουσιάζονται οι αρχές του μαζί με κάποια γλώσσα προγραμματισμού. Περισσότερο διαδεδομένες «εκπαιδευτικές» γλώσσες προγραμματισμού είναι η Pascal και η Ada. Οποιοδήποτε βιβλίο για τις γλώσσες αυτές είναι σχεδόν σίγουρο ότι θα αποτελεί και μία καλή εισαγωγή στον προγραμματισμό. Στον τόμο αυτό, επειδή μελετήσαμε τον διαδικασιακό προγραμματισμό, χρησιμοποίησα αρκετά στοιχεία από το βιβλίο **«Oh! Pascal!» του Doug Cooper (εκδ. Norton & Company, 1993)**. Πρόκειται για ένα εξαιρετικά καλογραμμένο βιβλίο, το οποίο παρουσιάζει με απλό (και πολλές φορές απολαυστικό) τρόπο όλα τα θέματα του προγραμματισμού. Σε κάθε του κεφάλαιο περιέχει πολλά παραδείγματα και ακόμη περισσότερες ασκήσεις, ενώ η βασική ύλη για τις δομές προγραμματισμού συνοδεύεται από ειδικά ζητήματα πρόληψης σφαλμάτων και γενικότερα ζητήματα ορθής προγραμματιστικής τακτικής.

Ένα αντίστοιχο και πολύ καλό βιβλίο, το οποίο όμως παρουσιάζει τη γλώσσα προγραμματισμού C, είναι το **«The C programming language» των Brian Kernighan και Dennis Ritchie (Prentice–Hall)**, το οποίο έχει κυκλοφορήσει και στα Ελληνικά από τις εκδόσεις Κλειδάριθμος με τον τίτλο **«Η γλώσσα προγραμματισμού C»**. Το βιβλίο αυτό απευθύνεται σε κάποιον που γνωρίζει τις βασικές αρχές προγραμματισμού και θέλει να βελτιώσει τις επιδόσεις του στη διαδεδομένη γλώσσα C (η γλώσσα C, αντίθετα με την Pascal, είναι πολύ διαδεδομένη ανάμεσα στους επαγγελματίες προγραμματιστές).

Ο ενδιαφερόμενος αναγνώστης μπορεί να βρει και κάποια βιβλία που δίνουν έμφαση στους αλγορίθμους περισσότερο από τη γλώσσα προγραμματισμού. Ένα πολύ καλό από αυτά τα βιβλία είναι και το **«Fundamentals of Data Structures in Pascal» των Ellis Horowitz και Sartaj Sahni (εκδ. Freeman & Co, 1999)**, στο οποίο περιγράφεται ένα πλήθος αλγορίθμων για τη δημιουργία και διαχείριση δομών δεδομένων. Το ιδιαίτερο χαρακτηριστικό του βιβλίου είναι ότι οι συγγραφείς χρησιμοποιούν τη γλώσσα Pascal για την περιγραφή των αλγορίθμων.

Ένα παλαιότερο, αλλά εξίσου καλό εισαγωγικό βιβλίο είναι αυτό των **Aho, Hopcroft & Ullmann «The Design and Analysis of Computer Algorithms» (εκδ. Addison–Wesley, 1974).**

Άφησα τελευταία την αναφορά στο έργο που θεωρείται ορόσημο στη διδασκαλία του προγραμματισμού και αποτελεί τη «βίβλο» των καταρτισμένων προγραμματιστών. Πρόκειται για το τρίτομο έργο του **Donald Knuth «The Art of Computer Programming» (εκδ. Addison–Wesley)**, το οποίο περιλαμβάνει τους τόμους:

- Fundamental Algorithms (3rd edition, 1997)
- Seminumerical Algorithm (3rd edition, 1997)
- Sorting and Searching (2rd edition, 1998)

Πρόκειται για ένα ολοκληρωμένο έργο, το οποίο προσεγγίζει τους αλγορίθμους που χρειάζεται κανείς σε κάθε πεδίο εφαρμογής του προγραμματισμού. Η παρουσίαση είναι αρκετά θεωρητική και το ύφος γραφής είναι πυκνό, με αποτέλεσμα να χρειάζεται προσπάθεια από πλευράς του αναγνώστη για την πλήρη κατανόηση. Δεν γνωρίζω όμως άλλο έργο που να συμπυκνώνει τόσο πολλή, πολύτιμη και διαχρονική γνώση. Το συστήνω σε όλους τους αναγνώστες ενδιαφέρονται να αποκτήσουν μία γενικότερη παιδεία γύρω από τους αλγορίθμους και τον προγραμματισμό.

Άλλα καλά προχωρημένα βιβλία είναι αυτό των **Aho, Hopcroft και Ullman «Data Structure & Algorithms» (εκδ. Addison–Wesley, 1983)** και το πιό σύγχρονο των **Horowitz, Sahni και Rajasekavan «Computer Algorithms: pseudocode» (εκδ. Computer Science Press, 1997).**

Πριν σας αφήσω να βυθιστείτε στην ομορφιά της «τέχνης» του προγραμματισμού και της σχεδίασης λογισμικού, όπως αυτή παρουσιάζεται μέσα από τα διάφορα βιβλία, ήθελα να επισημάνω το εξής: Μπορεί τα περισσότερα από τα βιβλία αυτά να έχουν εκδοθεί πριν από χρόνια (για παράδειγμα, τα βιβλία του Knuth εκδόθηκαν για πρώτη φορά το 1968!), όμως είναι ακόμη (αποδεδειγμένα) χρήσιμα. Σχεδόν καμία από τις βασικές έννοιες του προγραμματισμού (ιδιαίτερα του δομημένου διαδικασιακού προγραμματισμού) δεν έχει αλλάξει τα τελευταία χρόνια.

Program Execution Terminated … Press any key to Continue

Adams, Douglas: συγγραφέας και παραγωγός ραδιοφωνικών εκπομπών. Ανάμεσα στα γνωστότερα βιβλία του είναι η τριλογία: The Hitch–hiker's Guide to the Galaxy, The Restaurant at the End of the Universe, Life – the Universe – and Everything, So long and Thanks for all the Fish, Mostly harmless. Όλα του τα βιβλία έχουν κυκλοφορήσει σε Ελληνική μετάφραση – σας τα συνιστώ ανεπιφύλακτα

bit (binary digit): πρόκειται για τη μικρότερη ποσότητα πληροφορίας, η οποία μας επιτρέπει να διαχωρίσουμε ανάμεσα σε δύο δυνατές καταστάσεις (π.χ. λάμπα αναμμένη ή σβηστή). Αναπαρίσταται με ένα δυαδικό ψηφίο, το οποίο αποτελεί π.χ. την τιμή μιας μεταβλητής η οποία μπορεί να πάρει τις τιμές 0 ή 1

Boolean: βασικός τύπος δεδομένων (ονομάστηκε έτσι προς τιμή του G. Boole, θεμελιωτή της Μαθηματικής Λογικής). Μια μεταβλητή τύπου Boolean (αλλιώς, λέγεται και λογική μεταβλητή) μπορεί να πάρει δύο τιμές: 1 (TRUE) ή 0 (FALSE)

byte: σύνολο 8 bits. Επειδή το bit είναι πολύ μικρή μονάδα πληροφορίας, συνήθως χρησιμοποιείται το byte, το οποίο μας δίνει τη δυνατότητα να αναπαραστήσουμε $2^8 = 256$ διαφορετικές καταστάσεις. Όλες οι μεγαλύτερες μονάδες πληροφορίας ορίζονται ως πολλαπλάσια του byte, δηλ 1 KB = 1024 (2^{10}) bytes, 1 MB = 1024 KB, 1 GB = 1024 MB, κ.λπ.

CASE (computer–aided software engineering): μεθοδολογία ανάπτυξης λογισμικού, η οποία χρησιμοποιεί υπολογιστές για την υποστήριξη όλων των φάσεων του κύκλου ζωής. Συνήθως αναφέρεται στο ειδικό λογισμικό που μπορεί να χρησιμοποιηθεί για το σκοπό αυτό

GOTO: εντολή με την οποία ο έλεγχος της ροής ενός προγράμματος μεταφέρεται αυθαίρετα σε μια εντολή που βρίσκεται σε κάποιο σημείο του προγράμματος. Η χρήση της εντολής GOTO δεν συστήνεται κατά κανόνα η χρήση της στο δομημένο προγραμματισμό απαγορεύεται)

HIPO (hierarchy + input – process – output) diagram: σύνολο εργαλείων αναπαράστασης ενός προγράμματος. Περιλαμβάνει ένα διάγραμμα δομής του λογισμικού, το οποίο επιτρέπει την επισκόπηση της αρχιτεκτονικής, και ένα σύνολο από καρτέλες IPO. Κάθε καρτέ-

Γλωσσάρι όρων

λα IPO περιγράφει ένα τμήμα ή υποσύστημα του λογισμικού, καταγράφοντας εισόδους, εξόδους, επεξεργασία, τοπικά δεδομένα, συγγραφέα κ.λπ. Η περιγραφή συμπληρώνεται με τις καρτέλες του λεξικού δεδομένων, οι οποίες περιγράφουν τα δεδομένα που χειρίζεται το λογισμικό. Τέλος, περιλαμβάνεται και ένα υπόμνημα, το οποίο επεξηγεί τα σύμβολα που χρησιμοποιούνται στα άλλα εργαλεία

Jackson diagram: διαγραμματικό εργαλείο περιγραφής της αρχιτεκτονικής ενός συστήματος λογισμικού, της συσχέτισης μεταξύ των τμημάτων που το αποτελούν και της ακολουθίας εκτέλεσης των διαφόρων λειτουργιών

O notation: ο συμβολισμός του «κεφαλαίου Ο» δηλώνει την τάξη μεγέθους της αποδοτικότητας ενός αλγορίθμου σε σχέση με ένα παράγοντα πολυπλοκότητας N. Μας ενδιαφέρει η τάξη μεγέθους και όχι η ακριβής μέτρηση της αποδοτικότητας, γι' αυτό και αγνοούμε σταθερούς παράγοντες με τους οποίους πολλαπλασιάζεται ή αυξάνεται προσθετικά ο N. Μερικές τυπικές τιμές αποδοτικότητας είναι:.

- O(1) Σταθερός χρόνος, ο οποίος δεν εξαρτάται καθόλου από τα δεδομένα
- O(N) Γραμμικός χρόνος: εάν διπλασιαστεί ο N, διπλασιάζεται και ο χρόνος εκτέλεσης.
- $O(\log_2 N)$ Λογαριθμικός χρόνος, ο οποίος αυξάνεται αργά καθώς αυξάνεται ο N: όταν ο N τετραπλασιάζεται, ο χρόνος διπλασιάζεται.
- $O(N*\log_2 N)$ Σχεδόν γραμμικός χρόνος: υπονοεί ότι ένας γραμμικός αλγόριθμος καλεί ένα αλγόριθμο χρόνου $O(\log_2 N)$
- $O(N^2)$ Αλγόριθμος τετραγωνικού χρόνου: όταν ο N διπλασιάζεται, ο χρόνος τετραπλασιάζεται.
- $O(N^3)$ Αλγόριθμος κυβικού χρόνου: όταν ο N διπλασιάζεται, ο χρόνος οκταπλασιάζεται.
- $O(2^n)$ Εκθετικός χρόνος: όταν ο N είναι 10, ο χρόνος είναι περίπου 1.000, ενώ όταν ο N διπλασιαστεί (γίνει δηλαδή 20), ο χρόνος γίνεται 1.000.000! Οι αλγόριθμοι με τέτοια απόδοση θεωρείται ότι δεν έχουν πρακτική εφαρμογή

Γλωσσάρι όρων

Warnier – Orr diagram: εργαλείο αναπαράστασης της ιεραρχικής δομής των δεδομένων, και μέσω αυτής, της λογικής ροής του προγράμματος και της συνολικής δομής του λογισμικού.

ακέραιος (integer): βασικός τύπος δεδομένων. Μια μεταβλητή τύπου ακεραίου μπορεί να πάρει ως τιμή έναν ακέραιο αριθμό. Σε κάθε υπολογιστή, υπάρχει πάντα ο μικρότερος αποδεκτός ακέραιος (-MAXINT) και ο μεγαλύτερος αποδεκτός ακέραιος (MAXINT).

ακολουθία εντολών (command sequence): η βασική δομή του δομημένου προγραμματισμού, η οποία μας επιτρέπει να συνδυάσουμε δύο εντολές, τοποθετώντας τη μία μετά την άλλη, καθορίζοντας με τον τρόπο αυτό και τη σειρά εκτέλεσης των εντολών.

αλγόριθμος (algorithm): η δομημένη περιγραφή της διαδικασίας επίλυσης ενός προβλήματος, συνήθως σε μορφή κατανοητή από ανθρώπους. Ένας αλγόριθμος αποτελείται από ένα πεπερασμένο αριθμό διακριτών βημάτων (λέγονται οδηγίες), τα οποία πρέπει να εκτελεστούν ακολουθιακά με μια συγκεκριμένη σειρά. Κάθε βήμα ενός ορθού αλγορίθμου αρχίζει, εκτελείται για συγκεκριμένο χρόνο και τελειώνει. Το ίδιο και ο αλγόριθμος: δεν υπάρχουν ορθοί αλγόριθμοι που εκτελούνται ατελείωτα.

αλγόριθμος του Ευκλείδη: ο πρώτος αλγόριθμος που έχει καταγραφεί είναι αυτός της εύρεσης του μέγιστου κοινού διαιρέτη δύο αριθμών. Τον περιγράφει ο Ευκλείδης στο βιβλίο του «Στοιχεία».

αναδρομή (recursion): τεχνική προγραμματισμού, κατά την οποία ένα τμήμα κώδικα (διαδικασία, συνήθως) καλεί τον εαυτό του. Η ορθή χρήση της αναδρομής οδηγεί σε συνεκτικούς και κομψούς αλγορίθμους. Η λανθασμένη χρήση καταναλώνει γρήγορα τους πόρους του συστήματος (κυρίως τη μνήμη) και μπορεί να οδηγήσει σε απότομο τερματισμό της εκτέλεσης του προγράμματος.

αναζήτηση (searching): η προσπάθεια ανεύρεσης ενός στοιχείου δεδομένων μέσα σε ένα σύνολο ή μια δομή δεδομένων. Υπάρχουν διάφοροι αλγόριθμοι αναζήτησης, με γνωστότερη τη δυαδική αναζήτηση.

απαριθμητής (index): μεταβλητή που χρησιμοποιείται για να διατρέξει μία διάσταση ενός πίνακα. Οι τιμές που επιτρέπεται να πάρει ο απαριθμητής βρίσκονται ανάμεσα στο κάτω και το πάνω όριο του

Γλωσσάρι όρων

πλήθους των στοιχείων της διάστασης.

αποδοτικότητα αλγορίθμου (algorithm efficiency): μέτρο της ποιότητας ενός αλγορίθμου που βασίζεται στη χρήση των υπολογιστικών πόρων. Ένας αποδοτικός αλγόριθμος κάνει την ελάχιστη δυνατή χρήση μνήμης και χρόνου – δηλαδή βημάτων – επεξεργασίας. Η αποδοτικότητα μετρείται με βάση κάποιον παράγοντα N, ο οποίος επηρεάζει σημαντικά τον αριθμό των βημάτων που απαιτούνται μέχρι τον τερματισμό του αλγορίθμου. Για παράδειγμα, τέτοιοι παράγοντες είναι το πλήθος των ψηφίων που πρέπει να εξεταστεί κάθε φορά, το πλήθος των στοιχείων του πίνακα που συγκρίνεται σε κάθε επανάληψη του αλγόριθμου ταξινόμησης κ.ά. Χρησιμοποιούμε το συμβολισμό του «κεφαλαίου Ο» για να δηλώσουμε την τάξη μεγέθους της αποδοτικότητας ενός αλγορίθμου σε σχέση με τον παράγοντα N.

απόφαση (decision): βασική δομή προγραμματισμού, με την οποία επιλέγεται η εκτέλεση μιας ανάμεσα από δύο εναλλακτικές εντολές (ή ομάδες εντολών) με βάση την τιμή κάποιας λογικής συνθήκης.

αρθρωτός προγραμματισμός (modular programmaning): πρακτική προγραμματισμού, σύμφωνα με την οποία ένα πρόγραμμα συντίθεται από αυτόνομα τμήματα κώδικα, με τέτοιο τρόπο ώστε να είναι δυνατή η τροποποίηση των επιμέρους τμημάτων χωρίς να επηρεάζεται η λειτουργία του συνολικού προγράμματος.

αστοχία (failure): η εκτός προδιαγραφών λειτουργία του λογισμικού, η οποία οφείλεται σε συντακτικό ή λογικό σφάλμα (error) ή άλλο ελάττωμα (fault) του λογισμικού. Η αστοχία μπορεί να εκδηλωθεί με διάφορους τρόπους, όπως η απότομη (μη–αναμενόμενη) διακοπή της λειτουργίας του λογισμικού, η λανθασμένη λειτουργία (που οδηγεί στην παραγωγή λανθασμένων δεδομένων εξόδου), κ.ά.

Βύρων: μηχανικός λογισμικού της εταιρείας THUNDERSOFT, ο οποίος έχει αναλάβει το έργο της ανάπτυξης ενός πληροφοριακού συστήματος για την επιχείρηση CHILDWARE. Μήπως κάτι συμβαίνει ανάμεσα σε αυτόν και την Ελένη;

γλώσσα σχεδίασης προγράμματος (program design language): γλώσσα περιγραφής διαδικασιακών προγραμμάτων, η οποία μοιάζει πολύ με γλώσσα προγραμματισμού υψηλού επιπέδου. Υποστη-

Γλωσσάρι όρων

ρίζει την ενσωμάτωση περιγραφικού κειμένου μέσα στις εντολές της. Δεν υπάρχουν μεταγλωττιστές για τη γλώσσα αυτή, υπάρχουν προ–επεξεργαστές, οι οποίοι μετατρέπουν τις περιγραφές σε γραφικά μοντέλα.

δεδομένα (data): τμήμα «ωμής» πληροφορίας (δηλαδή, πληροφορία η οποία δεν έχει ακόμη υποστεί επεξεργασία). Τα δεδομένα συνήθως εισέρχονται σε κάποιο σύστημα επεξεργασίας δεδομένων (π.χ. ένα πρόγραμμα) και περιγράφουν τις συνθήκες για το υπό επίλυση πρόβλημα. Καταχρηστικά χρησιμοποιείται και ο όρος «δεδομένα εξόδου», αντί του ορθότερου «πληροφορία εξόδου»

δείκτης (pointer): βασικός τύπος δεδομένων. Η τιμή μιας μεταβλητής αυτού του τύπου είναι η διεύθυνση μιας θέσης μνήμης όπου βρίσκεται ένα στοιχείο δεδομένων. Για την προσπέλαση της διεύθυνσης της θέσης μνήμης χρησιμοποιείται το όνομα της μεταβλητής (π.χ. point), ενώ για την προσπέλαση της τιμής του στοιχείου δεδομένων, χρησιμοποιείται ο τελεστής έμμεσης προσπέλασης (π.χ. point^). Όταν δεν έχει ακόμη καταχωρηθεί τιμή σε μια μεταβλητή δείκτη, αυτή καλείται «αόριστη» (στην περίπτωση αυτή, η μεταβλητή δείχνει σε μια τυχαία θέση μνήμης), ενώ όταν θέλουμε η μεταβλητή να μη δείχνει σε κάποια θέση μνήμης, καταχωρούμε σε αυτή την τιμή NIL. Ένας δείκτης μπορεί να βρεθεί σε εκκρεμότητα (dangling pointer), όταν διαγραφεί η θέση μνήμης (ή ο κόμβος) στην οποία δείχνει.

δένδρο (tree): δομή δεδομένων (είδος διασυνδεδεμένης λίστας), η οποία αποτελείται από κόμβους διασυνδεδεμένους με ιεραρχικό τρόπο. Ο πρώτος κόμβος αποτελεί τη ρίζα (root) του δένδρου. Κάθε κόμβος έχει ένα προηγούμενο κόμβο (γονικός κόμβος – parent node) και ένα ή περισσότερους κόμβους στους οποίους δείχνουν οι σύνδεσμοί του (κόμβοι παιδιά – child nodes). Οι κόμβοι που δεν έχουν παιδιά καλούνται φύλλα (leaves), και η διαδρομή από ένα κόμβο σε άλλο καλείται μονοπάτι (path). Εάν σε κάθε κόμβο επιτρέπεται να έχει το πολύ δύο παιδιά, τότε το δένδρο καλείται «δυαδικό» (binary tree).

δέσμευση (binding): η δέσμευση μιας θέσης μνήμης για να καταχωρηθεί η τιμή μιας μεταβλητής.

Γλωσσάρι όρων

δήλωση (declaration): οι γλώσσες προγραμματισμού απαιτούν στην αρχή κάθε προγράμματος ή υπο–προγράμματος να δηλώνονται οι τύποι, οι σταθερές, οι μεταβλητές και τα υπο–προγράμματα που θα χρησιμοποιηθούν σε αυτό το τμήμα κώδικα. Η δήλωση γίνεται συνήθως με το όνομα και τον τύπο κάθε σταθεράς ή μεταβλητής, το όνομα και τις παραμέτρους κλήσης κάθε υπο–προγράμματος, και το όνομα και τους τύπους των επιμέρους πεδίων κάθε σύνθετου τύπου.

διάγραμμα δομής (structure chart): γραφικό εργαλείο περιγραφής του αρχιτεκτονικού σχεδίου ενός προγράμματος. Ευνοεί την περιγραφή της συγκρότησης του προγράμματος από υπο–προγράμματα και του τρόπου επικοινωνίας μεταξύ αυτών.

διάγραμμα ροής προγράμματος (program flowchart): γραφική τεχνική αναπαράστασης της ροής εκτέλεσης και των βημάτων ενός αλγορίθμου. Χρησιμοποιεί ένα σύνολο τυποποιημένων συμβόλων για την αναπαράσταση του είδους της κάθε οδηγίας ενός αλγορίθμου, καθένα από τα οποία επιγράφεται με το περιεχόμενο της οδηγίας. Η σύνδεση μεταξύ των συμβόλων αναπαριστά τη ροή εκτέλεσης, η οποία ξεκινά πάντα από το σύμβολο ΑΡΧΗ και τελειώνει στο σύμβολο ΤΕΛΟΣ. Κάθε σύμβολο ενός διαγράμματος ροής μπορεί να αναλύεται σε ένα επιμέρους διάγραμμα.

διαδικασία (procedure): ένα υπο–πρόγραμμα, το οποίο περιλαμβάνει τοπικές δηλώσεις μεταβλητών και δομών δεδομένων και επικοινωνεί με το κυρίως πρόγραμμα ή με άλλα υπο–προγράμματα μέσω παραμέτρων.

διασυνδεδεμένη λίστα (linked list): βασική δομή δεδομένων, η οποία αποτελείται από ένα σύνολο κόμβων που είναι διασυνδεδεμένοι με συνδέσμους. Όλοι οι κόμβοι της λίστας έχουν την ίδια εσωτερική δομή, η οποία περιλαμβάνει τουλάχιστον ένα πεδίο δεδομένων και ένα σύνδεσμο τύπου δείκτη. Ο σύνδεσμος κάθε κόμβου δείχνει στον επόμενο κόμβο, εκτός από το σύνδεσμο του τελευταίου κόμβου, ο οποίος (κατά σύμβαση) έχει την τιμή NIL. Υπάρχει και ο δείκτης TOP, ο οποίος δείχνει στον πρώτο κόμβο της λίστας (πρόκειται για εξωτερική μεταβλητή, όχι για κόμβο της λίστας). Η προσπέλαση ενός κόμβου γίνεται ακολουθιακά, ξεκινώντας από τον κόμβο όπου δείχνει ο TOP και ακολουθώντας τους συνδέσμους των κόμβων, μέχρι

Γλωσσάρι όρων

να φτάσουμε στον κόμβο που αναζητούμε ή στο τέλος της λίστας.

διαχείριση σφάλματος (error handling): η δραστηριότητα με την οποία αντιμετωπίζεται η εμφάνιση σφάλματος κατά την εκτέλεση ενός προγράμματος. Οι σύγχρονες γλώσσες προγραμματισμού συμπεριλαμβάνουν συντακτικές δομές με τις οποίες μπορεί να δηλωθεί ο τρόπος διαχείρισης των αναμενόμενων σφαλμάτων για κάποιο υπο–πρόγραμμα (συνήθως μέσω του μηχανισμού των εξαιρέσεων).

διεπαφή (διαπροσωπείο) προγράματος – χρήστη (user interface): το τμήμα του λογισμικού μέσω του οποίου ο χρήστης έχει πρόσβαση στις λειτουργίες του λογισμικού. Μεταφορικά, πρόκειται για το προσωπείο που παρουσιάζει το λογισμικό στο χρήστη.

διεργασία (process): η μορφή ενός προγράμματος την ώρα που εκτελείται, ή αναμένει να εκτελεστεί. Η διαφορά ενός προγράμματος από μια διεργασία είναι ότι η διεργασία βρίσκεται υποχρεωτικά στην κύρια μνήμη του υπολογιστή (ενώ το πρόγραμμα μπορεί να βρίσκεται στη βοηθητική) και δεσμεύει πόρους του συστήματος (οπωσδήποτε μνήμη και ίσως επεξεργαστική ισχύ, εάν εκτελείται).

διπλά διασυνδεδεμένη λίστα (doubly linked list): διασυνδεδεμένη λίστα, στην οποία ο κάθε κόμβος έχει δύο συνδέσμους: έναν που δείχνει στον επόμενο κόμβο, και έναν που δείχνει στον προηγούμενο. Αντίστοιχα, εκτός από τον δείκτη TOP, ο οποίος δείχνει στο ένα άκρο της λίστας, υπάρχει και ο δείκτης END, ο οποίος δείχνει στο άλλο άκρο της. Έτσι, η διαπέραση της λίστας για την ανεύρεση ενός στοιχείου μπορεί να γίνει προς δύο κατευθύνσεις.

διπλή ουρά (dequeue): ουρά στην οποία επιτρέπεται η πρόσθεση ή η διαγραφή κόμβου σε οποιοδήποτε από τα δύο άκρα της.

δομή δεδομένων (data structure): σχήμα οργάνωσης δεδομένων, με το οποίο ομαδοποιούνται ομοειδή δεδομένα. Διαδεδομένες δομές δεδομένων είναι ο πίνακας, η διασυνδεδεμένη λίστα κ.ά.

δομημένη γλώσσα (structured language): υποσύνολο των λέξεων μιας φυσικής γλώσσας που χρησιμοποιείται για την περιγραφή αλγορίθμων. Πλεονεκτεί της φυσικής γλώσσας στην έλλειψη αμφισημίας, αλλά μειονεκτεί έναντι του ψευδοκώδικα στη λεπτομέρεια

Γλωσσάρι όρων

και ακρίβεια της περιγραφής.

δομημένος προγραμματισμός (structured programming): τεχνική προγραμματισμού, σύμφωνα με την οποία κάθε πρόγραμμα μπορεί να συντεθεί από τρεις βασικές προγραμματιστικές δομές (ακολουθία, απόφαση / επιλογή, επανάληψη) ή άλλες δομές, οι οποίες όμως προκύπτουν από σύνθεση των βασικών. Επιπλέον, πρέπει να ισχύουν δύο αρχές:

- Κάθε δομή προγραμματισμού (και κατ' επέκταση ολόκληρο το πρόγραμμα) έχει μόνο ένα σημείο εισόδου και ένα σημείο εξόδου
- Η ροή μεταξύ της εισόδου και της εξόδου μιας δομής προγραμματισμού ξεικνά από την είσοδο, τελειώνει στην έξοδο, είναι απρόσκοπτη και δεν τερματίζεται απότομα.

δομοδιάγραμμα (box / Nassi–Shneiderman diagram): γραφικό εργαλείο σχεδίασης προγραμμάτων, το οποίο περιέχει ξεχωριστή αναπαράσταση για κάθε προγραμματιστική δομή, με αποτέλεσμα να μην είναι δυνατή η παράβαση των κανόνων του δομημένου προγραμματισμού.

είσοδος (input): δεδομένα που εισέρχονται σε ένα σύστημα (βλ. σύστημα).

εκσφαλμάτωση (debugging): η διόρθωση του κώδικα ώστε να αφαιρεθούν τα σφάλματα που ανακαλύφθηκαν κατά τον έλεγχο του λογισμικού.

έκφραση (expression): ένας συνδυασμός με τη χρήση τελεστών από μεταβλητές, αριθμητικές τιμές και σταθερές. Όταν καταχωρηθούν τιμές στις μεταβλητές, η έκφραση αποδίδει και αυτή μια τιμή, η οποία μπορεί στη συνέχεια να καταχωρηθεί σε μεταβλητή, να περαστεί ως παράμετρος, να χρησιμοποιηθεί ως συνθήκη κ.λπ.

ελάττωμα (fault): σφάλμα προγραμματισμού, το οποίο μπορεί να προκαλέσει αστοχία του προγράμματος.

έλεγχος (testing): η δοκιμαστική εκτέλεση του λογισμικού χρησιμοποιώντας δεδομένα τα οποία προσομοιάζουν τα πραγματικά δεδομένα που θα χρησιμοποιεί το λογισμικό όταν τεθεί σε λειτουργία. Κάθε δοκιμαστική εκτέλεση καλείται «περίπτωση ελέγχου» (test

Γλωσσάρι όρων

case), ενώ τα δεδομένα εισόδου που χρησιμοποιούνται σε αυτή καλούνται «δεδομένα ελέγχου» (test data). Τα δεδομένα ελέγχου σχεδιάζονται με στόχο να δοκιμαστεί η συμπεριφορά του λογισμικού σε σχέση με τις προδιαγραφές εισόδου / εξόδου και να ανακαλυφθούν σφάλματα που ίσως προκαλέσουν αστοχία του λογισμικού. Διακρίνουμε τις ακόλουθες τρεις μεθόδους σχεδίασης περιπτώσεων ελέγχου:

- Τον λειτουργικό έλεγχο (functional testing) ή έλεγχο αδιαφανούς κουτιού (black box testing), όπου περιλαμβάνονται τεχνικές σχεδίασης περιπτώσεων ελέγχου με τις οποίες δοκιμάζεται η εξωτερικά παρατηρήσιμη συμπεριφορά του λογισμικού,
- Τον δομικό έλεγχο (structural testing) ή έλεγχο διαφανούς κουτιού (white box testing), όπου περιλαμβάνονται τεχνικές σχεδίασης περιπτώσεων ελέγχου με τις οποίες δοκιμάζεται η ορθότητα του κώδικα του λογισμικού, και
- Τον έλεγχο διεπαφών (interface testing), όπου περιλαμβάνονται τεχνικές σχεδίασης περιπτώσεων ελέγχου με τις οποίες δοκιμάζεται η επικοινωνία και η συγκρότηση των τμημάτων του λογισμικού

Ελένη: διευθύντρια της επιχείρησης CHILDWARE, η οποία (ορθώς) αποφάσισε τη μηχανοργάνωση της επιχείρησης. Μήπως σκέφτεται να «οργανώσει» και την υπόλοιπη ζωή της;

εμβέλεια (scope): η «έκταση» (δηλαδή ο αριθμός των υπο–προγραμμάτων) στην οποία ισχύει η δήλωση μιας μεταβλητής, σταθεράς ή υπο–προγράμματος.

εντολή (statement, command): το μικρότερο τμήμα ενός προγράμματος. Οι εντολές είναι γραμμένες σε κάποια γλώσσα προγραμματισμού. Η εκτέλεση μιας εντολής από τον υπολογιστή προκαλεί κάποιες ενέργειες από αυτόν. Οι εντολές μπορεί να περιέχονται σε πρόγραμμα, αλλά μπορεί να δίνονται και από τον χρήστη.

εξαίρεση (exception): μια συνθήκη ή αλλαγή κατάστασης, η οποία προκαλεί τη διακοπή της ομαλής εκτέλεσης του προγράμματος και την εκτέλεση ενός ειδικού υπο–προγράμματος χειρισμού της.

έξοδος (output): δεδομένα (πληροφορία) που παράγονται από ένα

Γλωσσάρι όρων

σύστημα (βλ. σύστημα).

επανάληψη (loop): προγραμματιστική δομή με την οποία επαναλαμβάνεται η εκτέλεση μιας εντολής ή μιας ομάδας εντολών. Ο αριθμός των επαναλήψεων καθορίζεται είτε με μετρητή στην αρχή της δομής, είτε με συνθήκη στην αρχή ή το τέλος της δομής.

επαναχρησιμοποίηση (reuse): η χρήση του ίδιου τμήματος κώδικα σε περισσότερα του ενός προγράμματα (αν και αρκετές φορές, είναι περισσότερο συμφέρουσα η επαναχρησιμοποίηση προδιαγραφών και σχεδίων λογισμικού). Η σχεδίαση του κώδικα ώστε να είναι δυνατή η επαναχρησιμοποίησή του επιβαρύνει την αρχική ανάπτυξή του, όμως τα μακροπρόθεσμα οφέλη που προκύπτουν είναι μεγαλύτερα, καθώς η επαναχρησιμοποίηση εξοικονομεί αρκετούς πόρους.

επεξεργασία (processing): η χρήση των δεδομένων εισόδου από τον υπολογιστή, ώστε να παραχθεί πληροφορία στην έξοδο. Τα βήματα της επεξεργασίας περιγράφονται από αλγορίθμους και υλοποιούνται με προγράμματα. Οι περισσότερο διαδεδομένες μορφές επεξεργασίας είναι η επιλογή, η σύγκριση, η επανάληψη και η εφαρμογή πράξεων επί των δεδομένων. Ας σημειωθεί ότι η επεξεργασία είναι ανεξάρτητη από τα δεδομένα.

επιλογή με πολλά ενδεχόμενα (case command): προγραμματιστική δομή απόφασης, η συνθήκη της οποίας μπορεί να λάβει περισσότερες από δύο τιμές, οπότε και τα πιθανά μονοπάτια εκτέλεσης είναι περισσότερα από δύο.

θέση μνήμης (memory location): η κύρια μνήμη του υπολογιστή διαιρείται λογικά σε έναν αριθμό θέσεων μνήμης. Όλες οι θέσεις μνήμης έχουν την ίδια χωρητικότητα, η οποία μετρείται σε bits. Σε κάθε θέση μνήμης αντιστοιχεί μια μοναδική διεύθυνση μνήμης και φυλάσσεται ένα στοιχείο δεδομένων. Η προσπέλαση του στοιχείου δεδομένων γίνεται χρησιμοποιώντας τη διεύθυνση μνήμης. Ο άμεσος χειρισμός διευθύνσεων μνήμης μέσα από ένα πρόγραμμα προϋποθέτει γνώσεις του τρόπου διαχείρισης της μνήμης από το λειτουργικό σύστημα. Συνεπώς, το πρόγραμμα εξαρτάται από τον υπολογιστή στον οποίο εκτελείται, και, σε τελική ανάλυση, γίνεται δυσανάγνωστο. Προτιμούμε, λοιπόν, να χρησιμοποιούμε το όνομα μεταβλητής

Γλωσσάρι όρων

για να αναφερθούμε σε ένα στοιχείο δεδομένων, αναφερόμενοι έμμεσα με τον τρόπο αυτό στη θέση μνήμης όπου φυλάσσεται η τιμή του.

καθολική (σφαιρική) δήλωση / μεταβλητή (global declaration / variable): η δήλωση μιας σταθεράς ή μεταβλητής ή ενός υπο–προγράμματος μέσα στο κυρίως πρόγραμμα. Με τον τρόπο αυτό, η δήλωση ισχύει και σε όλα τα υπο–προγράμματα (εκτός από εκείνα όπου υπάρχει τοπική δήλωση στο ίδιο όνομα), οπότε η προσπέλαση και η χρήση της σταθεράς, της μεταβλητής ή του υπο–προγράμματος μπορεί να γίνει από κάθε υπο–πρόγραμμα. Στην πράξη, όμως, αυτή η πρακτική αποθαρρύνεται γιατί δημιουργεί παρενέργειες.

Καμέας, Απόστολος: συνεργάτης της Ελένης.

κανάλια (δίαυλοι) διακίνησης πληροφοριών (buses): σύνολο ηλεκτρικά αγώγιμων δρόμων, οι οποίοι συνδέουν εξαρτήματα του υπολογιστή και χρησιμοποιούνται για την ανταλλαγή δεδομένων (με τη μορφή ηλεκτρικών σημάτων) μεταξύ τους. Ένας υπολογιστής περιλαμβάνει τρία κανάλια: το κανάλι δεδομένων (data bus), στο οποίο διακινούνται δεδομένα, το κανάλι διευθύνσεων (address bus), στο οποίο διακινούνται διευθύνσεις μνήμης, και το κανάλι ελέγχου (control bus), στο οποίο διακινούνται σήματα ελέγχου.

Καρκαντός, Νικόλαος: άλλος ένας συνεργάτης και οικονομικός σύμβουλος της Ελένης.

κατά βήμα εκλέπτυνση (stepwise refinement): πρακτική προγραμματισμού η οποία οδηγεί στην ανάπτυξη ενός προγράμματος ξεκινώντας από μια αφαιρετική περιγραφή του τύπου «είσοδος δεδομένων – επεξεργασία – έξοδος» και προσθέτοντας διαδοχικά περισσότερη λεπτομέρεια σε κάθε βήμα. Κάθε ενδιάμεση περιγραφή είναι ορθή και προκύπτει συνήθως αντικαθιστώντας μια οδηγία της προηγούμενης περιγραφής με μια ομάδα ή δομή οδηγιών.

καταχώρηση (assignment): η απόδοση τιμής σε μια μεταβλητή κατά την εκτέλεση του προγράμματος (λέγεται και δέσμευση – binding). Συνήθως αναφέρεται σε εντολές της μορφής a := x, όπου a είναι η μεταβλητή, := είναι ο τελεστής καταχώρησης, και x είναι η τιμή, η οποία μπορεί να είναι:

Γλωσσάρι όρων

- αριθμητική τιμή, λογική τιμή ή χαρακτήρας,
- μια έκφραση που δίνει τιμή του ίδιου τύπου με τη μεταβλητή,
- μια άλλη μεταβλητή του ίδιου τύπου που ήδη έχει κάποια τιμή,
- μια συνάρτηση που επιστρέφει μια τιμή, κ.ά.

κεντρική μονάδα επεξεργασίας (central processing unit – CPU): η μονάδα του υπολογιστή όπου υφίστανται επεξεργασία (εκτελούνται) οι εντολές των προγραμμάτων. Η ΚΜΕ μπορεί να ανακαλέσει την επόμενη εντολή προς εκτέλεση από τη μνήμη, να τη διερμηνεύσει, να ανακαλέσει τα δεδομένα που περιλαμβάνει η εντολή, να την εκτελέσει και να τοποθετήσει τα αποτελέσματα της εκτέλεσης πίσω στη μνήμη. Στην πραγματικότητα, η ΚΜΕ είναι ένα πολύπλοκο ολοκληρωμένο ψηφιακό κύκλωμα (chip). Οι περισσότερο διαδεδομένες εταιρείες κατασκευής ΚΜΕ για προσωπικούς υπολογιστές είναι η Intel (80386, 80486, Pentium, Pentium II κ.ά.) και η Motorola (68020, 68030 κ.ά.). Το ζητούμενο από την ΚΜΕ είναι η ταχύτητα, γι' αυτό και τα διάφορα «μοντέλα» συγκρίνονται κυρίως με βάση τη συχνότητα λειτουργίας τους (όσο μεγαλύτερη, τόσο πιο γρήγορη η ΚΜΕ) και τον αριθμό MIPS (δηλαδή το πλήθος των εντολών που εκτελούν ανά δευτερόλεπτο).

κυκλική λίστα (circular list): διασυνδεδεμένη λίστα, στην οποία ο σύνδεσμος του τελευταίου κόμβου δείχνει πάλι στον πρώτο κόμβο της λίστας.

λειτουργική ανεξαρτησία (functional independence): επιθυμητή ιδιότητα του λογισμικού, η οποία επιτυγχάνεται όταν σχεδιάζουμε κάθε τμήμα του έτσι ώστε να υλοποιεί όσο το δυνατό λιγότερες λειτουργίες (μία στην καλύτερη περίπτωση) και να χρησιμοποιείται από τα υπόλοιπα τμήματα με απλό τρόπο. Η λειτουργική ανεξαρτησία μετρείται με δύο ποιοτικά κριτήρια, τη συνοχή (υψηλός βαθμός αυξάνει τη λειτουργική ανεξαρτησία) και τη σύζευξη (υψηλός βαθμός μειώνει τη λειτουργική ανεξαρτησία).

λειτουργικό σύστημα (operating system): το σημαντικότερο λογισμικό συστήματος, το οποίο είναι υπεύθυνο για τη διαχείριση των πόρων (επεξεργαστής, μνήμη, περιφερειακές συσκευές) του υπολογιστή.

Γλωσσάρι όρων

λεξικό δεδομένων (data dictionary): σύνολο από καρτέλες περιγραφής των δεδομένων που χειρίζεται ένα σύστημα. Περιλαμβάνεται μια καρτέλα για κάθε αντικείμενο δεδομένων, απλό ή σύνθετο, στο οποίο περιγράφεται το όνομα και ο τύπος του αντικειμένου, η διεργασία που το παράγει, ο τρόπος σύνθεσής του από απλούστερα αντικείμενα (εάν είναι σύνθετο), οι διεργασίες που το χρησιμοποιούν ή το καταναλώνουν, τα σημεία όπου αυτό αποθηκεύεται κ.ά.

λογισμικό (software): ένα ολοκληρωμένο (αναφορικά με ένα αντικείμενο) σύνολο προγραμμάτων ηλεκτρονικού υπολογιστή. Ο συνώνυμος όρος «πρόγραμμα» συνήθως αναφέρεται σε ένα αυτόνομο πρόγραμμα ή τμήμα προγράμματος, το οποίο μπορεί να ανήκει σε κάποιο πακέτο λογισμικού. Οι δύο κύριες κατηγορίες λογισμικού είναι το λογισμικό συστήματος (που περιλαμβάνει προγράμματα διαχείρισης του υπολογιστή και υποστήριξης της λειτουργίας του) και το λογισμικό εφαρμογών (το οποίο περιλαμβάνει εφαρμογές τελικού χρήστη, όπως επεξεργαστές κειμένου, πακέτα γραφικών κ.ά.).

μεθοδολογίες σχεδίασης (design methodologies): σύνολα κανόνων που ελαχιστοποιούν τον κίνδυνο αποτυχίας των δραστηριοτήτων σχεδίασης ενός προγράμματος. Οι μεθοδολογίες σχεδίασης αποτελούνται από φάσεις που εκτελούνται διαδοχικά και διακρίνονται:

- με βάση το κριτήριο της σχεδιαστικής διαδικασίας, σε ακολουθιακές (κάθε φάση μπορεί να εκτελεστεί μια φορά) και επαναληπτικές (επιτρέπεται η οπισθοδρόμηση σε προηγούμενη φάση και η επανάληψή της)
- με βάση το κριτήριο της σχεδιαστικής κατεύθυνσης, σε μεθοδολογίες σχεδίασης από πάνω προς τα κάτω (πρώτα σχεδιάζεται το πλήρες σύστημα, το οποίο διασπάται σε υπο–συστήματα κ.ο.κ.), από κάτω προς τα πάνω (πρώτα σχεδιάζονται τα μικρότερα κρίσιμα τμήματα, τα οποία συγκροτούνται διαδοχικά σε μεγαλύτερα τμήματα μέχρι να σχεδιαστεί το πλήρες σύστημα) και από τη μέση προς την άκρη (πρώτα σχεδιάζονται τα γνωστά τμήματα και στη συνέχεια ακολουθείται σχεδίαση από πάνω προς τα κάτω, ή από κάτω προς τα πάνω, ανάλογα με την περίπτωση)

Γλωσσάρι όρων

- με βάση το κριτήριο της μονάδας διάσπασης, σε μεθοδολογίες βασισμένες στις διεργασίες (το σύστημα σχεδιάζεται με βάση τη διάσπαση των λειτουργιών του συστήματος), μεθοδολογίες βασισμένες στα δεδομένα (βασίζονται στη διάσπαση των δεδομένων) και συνδυαστικές μεθοδολογίες (εξετάζουν μαζί δεδομένα και διεργασίες – οι περισσότερο γνωστές είναι οι αντικειμενοστραφείς μεθοδολογίες)

μεταβλητή (variable): μια θέση μνήμης (ή ένα σύνολο θέσεων μνήμης), στην οποία αναφερόμαστε (για να διευκολυνθούμε προγραμματιστικά) χρησιμοποιώντας ένα όνομα. Οι μεταβλητές χρησιμοποιούνται ως αποθήκες δεδομένων κατά την εκτέλεση ενός προγράμματος. Το περιεχόμενο της αντίστοιχης θέσης μνήμης διατηρεί δεδομένα του τύπου της μεταβλητής για όσο χρόνο το πρόγραμμα στο οποίο η μεταβλητή έχει δηλωθεί βρίσκεται στη μνήμη. Το όνομα και ο τύπος της μεταβλητής δηλώνεται στο τμήμα αρχικοποίησης του προγράμματος, ενώ η πρώτη καταχώρηση τιμής καλείται «αρχικοποίηση μεταβλητής». Το πρόγραμμα αυτό μπορεί να γράψει ή να διαβάσει τα περιεχόμενα της θέσης μνήμης.

μεταγλωττιστής (compiler): ειδικό λογισμικό, το οποίο δέχεται στην είσοδο τον κώδικα ενός προγράμματος (λέγεται πηγαίος κώδικας – source code) γραμμένο σε κάποια γλώσσα προγραμματισμού. Για κάθε γλώσσα προγραμματισμού υπάρχει ένας μεταγλωττιστής που αναγνωρίζει τα σύμβολα και τις συντακτικές δομές της (κάθε μεταγλωττιστής είναι αφιερωμένος σε μία μόνο γλώσσα προγραμματισμού). Ένας μεταγλωττιστής μεταφράζει τον πηγαίο κώδικα σε κώδικα αναγνωρίσιμο από τον υπολογιστή (object code). Ο κώδικας αυτός μπορεί να εκτελεστεί από τον υπολογιστή ανεξάρτητα από τη γλώσσα στην οποία γράφτηκε το αρχικό πρόγραμμα.

μνήμη (memory): μία συσκευή όπου μπορεί να αποθηκευθούν και να ανακληθούν δεδομένα ή πληροφορία. Διακρίνεται σε κύρια (εσωτερική) και δευτερεύουσα (εξωτερική). Η κύρια μνήνη συνήθως αναφέρεται στη μνήμη RAM (μνήμη τυχαίας προσπέλασης) του υπολογιστή, όπου βρίσκονται τα προγράμματα και τα δεδομένα που αυτός μεταχειρίζεται κάθε φορά. Ο υπο-

Γλωσσάρι όρων

λογιστής μπορεί να γράφει και να διαβάζει τις θέσεις της μνήμης RAM, τα περιεχόμενα της οποίας χάνονται μόλις διακοπεί η παροχή ηλεκτρικού ρεύματος στον υπολογιστή. Ένα τμήμα της, η μνήμη ανάγνωσης–μόνο (ROM) περιέχει τα προγράμματα που χρησιμοποιούνται για την εκκίνηση του υπολογιστή, γι' αυτό και τα περιεχόμενά της δεν είναι δυνατό να τροποποιηθούν με ενέργειες του χρήστη, ούτε χάνονται όταν κλείσουμε τον υπολογιστή. Η δευτερεύουσα μνήμη περιλαμβάνει συσκευές όπως ο σκληρός δίσκος, ο εύκαμπτος δίσκος, η μαγνητική ταινία, το CD κ.ά. Λογικά, η μνήμη διαιρείται σε θέσεις μνήμης ίσης χωρητικότητας.

μοντέλο καταρράκτη (waterfall model): πρόκειται για το περισσότερο διαδεδομένο μοντέλο κύκλου ζωής λογισμικού. Αποτελείται από τις εξής φάσεις: Ανάλυση απαιτήσεων, Σχεδίαση συστήματος και λογισμικού, Υλοποίηση και έλεγχος τμημάτων, Ολοκλήρωση και έλεγχος συστήματος, Χρήση και συντήρηση. Οι φάσεις εκτελούνται ακολουθιακά με σχετική επικάλυψη, ενώ νεώτερες εκδόσεις του επιτρέπουν και επιστροφή σε προηγούμενες φάσεις.

μοντέλο κύκλου ζωής (life cycle model): μια ολοκληρωμένη διαδικασία, η οποία αποτελείται από φάσεις, βήματα και δραστηριότητες και οδηγεί, ξεκινώντας από μια αρχική περιγραφή των απαιτήσεων, στην ανάπτυξη και τελική παράδοση του λογισμικού. Υπάρχουν διάφορα είδη μοντέλων κύκλου ζωής, ανάλογα με το αν οι φάσεις είναι διαδοχικές ή επικαλυπτόμενες, με το αν οι φάσεις εκτελούνται ακολουθιακά ή κυκλικά, με το αν επιτρέπεται ή όχι η επιστροφή σε προηγούμενη φάση, κ.λπ. Συνήθως, στο τέλος κάθε φάσης προκύπτει ένας αριθμός παραδοτέων, τα οποία χρησιμοποιούνται στις επόμενες φάσεις (αλλά και για τον έλεγχο της ομαλής εξέλιξης του έργου). Τα περισσότερο διαδεδομένα μοντέλα κύκλου ζωής είναι το μοντέλο καταρράκτη, το σπειροειδές μοντέλο, το μοντέλο πρωτοτυποποίησης κ.ά.

οδηγία (command): το μικρότερο βήμα ενός αλγορίθμου. Αντίθετα με τις εντολές προγράμματος, οι οδηγίες είναι γραμμένες σε ψευδογλώσσα (και όχι σε κάποια γλώσσα προγραμματισμού) και δεν αναγνωρίζονται από τον υπολογιστή. Μια οδηγία μπορεί να είναι απλή (ατομική) ή σύνθετη (δηλαδή να αποτελείται από πολλές οδη-

Γλωσσάρι όρων

γίες συσχετισμένες με δομές προγραμματισμού). Στη δεύτερη περίπτωση πρόκειται για ομάδα οδηγιών (block).

ομάδα οδηγιών ή εντολών (block): σύνθετη οδηγία ή εντολή, η οποία αποτελείται από πολλές οδηγίες ή εντολές συσχετισμένες με δομές προγραμματισμού (ακολουθία, απόφαση, επανάληψη κ.ά.). Μια ομάδα οδηγιών ή εντολών συνήθως περικλείεται ανάμεσα στις δεσμευμένες λέξεις ΑΡΧΗ και ΤΕΛΟΣ.

οπισθοδρόμηση (backtracking): τεχνική προγραμματισμού για την ανεύρεση της ορθής λύσης μέσα σε ένα πολύ μεγάλο χώρο λύσεων. Σύμφωνα με την τεχνική αυτή (η οποία έχει βρει μεγάλη εφαρμογή στην Τεχνητή Νοημοσύνη), το πρόγραμμα δημιουργεί με διαδοχικές αποφάσεις ένα μονοπάτι επίλυσης. Μόλις φτάσει σε αδιέξοδο, οπισθοδρομεί στην τελευταία απόφαση και ακολουθεί την εναλλακτική λύση (αν υπάρχει).

οπτικοποιημένος (ενορατικός) προγραμματισμός (visual programming): μεθοδολογία ανάπτυξης προγραμμάτων ξεκινώντας από το διαπροσωπείο (διεπαφή) του προγράμματος με το χρήστη και υλοποιώντας τις λειτουργίες που χειρίζονται τα γεγονότα που προκαλεί ο χρήστης στο πρόγραμμα. Υποστηρίζεται από πολύ εύχρηστα εργαλεία, τα οποία αναπαριστούν με γραφικό τρόπο τόσο το διαπροσωπείο του συστήματος, όσο και διάφορες εσωτερικές λειτουργίες του. Η μεθοδολογία αυτή εφαρμόζεται κυρίως κατά την ταχεία ανάπτυξη εφαρμογών (rapid application development).

ουρά (queue): δομή δεδομένων (υλοποιείται με διασυνδεδεμένη λίστα), στην οποία η πρόσθεση κόμβων μπορεί να γίνει μόνο από το ένα άκρο της και η διαγραφή μόνο από το άλλο. Έτσι, οι κόμβοι της λίστας διαγράφονται με τη σειρά με την οποία είχαν προστεθεί (δομή First In First Out – FIFO).

παράδειγμα προγραμματισμού (programming paradigm): ένα σύνολο κανόνων, το οποίο περιγράφει το θεμιτό τρόπο σύνθεσης ενός προγράμματος από εντολές, τμήματα, ή υπο–προγράμματα. Διαδεδομένα παραδείγματα προγραμματισμού είναι ο προστακτικός προγραμματισμός, στον οποίο απαιτείται να περιγράψουμε ακριβώς τα βήματα της διαδικασίας επίλυσης, σε αντίθεση με το δηλωτικό προγραμματισμό, όπου χρειάζεται να περιγράψουμε τη

Γλωσσάρι όρων

ζητούμενη λύση, καθώς και ο διαδικασιακός προγραμματισμός, ο οποίος δίνει έμφαση στην επεξεργασία των δεδομένων, σε αντίθεση με τον αντικειμενοστραφή προγραμματισμό, ο οποίος επικεντρώνεται στα δεδομένα που χειρίζεται ένα πρόγραμμα.

παράμετρος (parameter): αριθμητική τιμή, μεταβλητή ή έκφραση η οποία ανταλλάσσεται ανάμεσα σε δύο προγράμματα, κατά την κλήση του ενός από το άλλο (στο διαδικασιακό προγραμματισμό, το ένα αποτελεί υπο–πρόγραμμα του άλλου). Η διαδικασία αυτή καλείται πέρασμα παραμέτρων και μπορεί να είναι δύο ειδών: πέρασμα με τιμή και πέρασμα μέσω διευθύνσεως (με αναφορά).

πέρασμα παραμέτρων με τιμή (parameter pass by value): όταν μία παράμετρος Υ, η οποία έχει δηλωθεί σε ένα πρόγραμμα Α, περνιέται σε ένα υπο–πρόγραμμα Β του Α με τιμή, τότε το Β δεν έχει δικαίωμα να τροποποιήσει την τιμή της Υ. Έτσι, εάν π.χ. στο Β υπάρχει εντολή στην οποία η Υ βρίσκεται στο αριστερό μέρος, η καταχώρηση της τιμής γίνεται σε μια τοπική μεταβλητή Υ, η οποία δε σχετίζεται με τη σφαιρική μεταβλητή Υ που είχε δηλωθεί στο Α.

πέρασμα παραμέτρων μέσω διευθύνσεως (parameter pass by reference): όταν μία παράμετρος Υ, η οποία έχει δηλωθεί σε ένα πρόγραμμα Α, περνιέται σε ένα υπο–πρόγραμμα Β του Α μέσω διεθύνσεως, τότε οποιαδήποτε καταχώρηση στην Υ αναφέρεται στη μεταβλητή Υ που είχε δηλωθεί στο Α (δηλαδή, δεν δημιουργείται τοπικό αντίγραφο της Υ). Με τον τρόπο αυτό, μπορεί το Β να επιστρέψει στο Α τα αποτελέσματα των υπολογισμών που περιλαμβάνει.

πίνακας (array): βασική δομή δεδομένων, η οποία αποτελείται από ένα σύνολο στοιχείων δεδομένων του ίδιου τύπου. Ο πίνακας μπορεί να έχει μία (οπότε τα στοιχεία είναι ακολουθιακά διατεταγμένα) ή περισσότερες διαστάσεις. Η αναφορά σε κάθε στοιχείο μπορεί να γίνει με τυχαία σειρά, χρησιμοποιώντας το όνομα του πίνακα και έναν απαριθμητή για κάθε διάσταση του πίνακα, ο οποίος περιγράφει τη (μοναδική) θέση του στοιχείου στον πίνακα.

πληροφορία (information): το νόημα που αποκτούν τα δεδομένα όταν υποστούν επεξεργασία (π.χ. όταν τοποθετηθούν μέσα σε κάποιο πλαίσιο, ή ερμηνευθούν από ανθρώπους). Οι υπολογιστές επεξεργάζονται τα δεδομένα και παράγουν πληροφορία χωρίς να «γνωρίζουν» ή να

Γλωσσάρι όρων

«αντιλαμβάνονται» το νόημά τους. Στα Μαθηματικά, η πληροφορία ορίζεται ως μέγε.θος αντίστροφο της εντροπίας και μετρείται σε bits

πραγματικός (real): βασικός τύπος δεδομένων. Μια μεταβλητή τύπου πραγματικού αριθμού μπορεί να πάρει ως τιμή οποιοδήποτε πραγματικό αριθμό. Το πλήθος των επιτρεπόμενων δεκαδικών ψηφίων εξαρτάται από την δηλωθείσα ακρίβεια αναπαράστασης του πραγματικού αριθμού, η οποία μπορεί να είναι απλή (single precision) ή διπλή (double precision).

πρόγραμμα (program): ακολουθία εντολών, οι οποίες είναι γραμμένες σύμφωνα με τους κανόνες μιας γλώσσας προγραμματισμού και μπορούν να αναγνωρισθούν και να εκτελεσθούν από ηλεκτρονικό υπολογιστή. Το πρόγραμμα είναι το αποτέλεσμα του προγραμματισμού.

προγραμματισμός (programming): η επιστήμη (και τέχνη) της δημιουργίας προγραμμάτων για ηλεκτρονικούς υπολογιστές. Πρόκειται για το μετασχηματισμό ενός αλγορίθμου σε μορφή κατανοητή από τον υπολογιστή, με τη χρήση μιας γλώσσας προγραμματισμού. Με τον τρόπο αυτό, ο υπολογιστής μπορεί να ακολουθήσει τα βήματα του αλγορίθμου για να επιλύσει ένα πρόβλημα.

προσπέλαση (access): η πρόσβαση σε μια ή περισσότερες θέσεις μνήμης, με στόχο την ανάγνωση ή την εγγραφή ενός ή περισσότερων στοιχείων δεδομένων. Η πρόσβαση που γίνεται χρησιμοποιώντας τη διεύθυνση της θέσης μνήμης ή το όνομα της μεταβλητής που έχει συσχετισθεί με το στοιχείο δεδομένων καλείται «άμεση». Η προσπέλαση που γίνεται με τη χρήση μεταβλητής τύπου δείκτη καλείται «έμμεση».

πρωτότυπο (prototype): λογισμικό το οποίο υλοποιεί τις κυριότερες (περισσότερο χρήσιμες ή κρίσιμες) λειτουργίες ενός μεγαλύτερου συστήματος. Το πρωτότυπο αναπτύσσεται συνήθως δοκιμαστικά, ώστε η ομάδα ανάπτυξης να καταλήξει σε συγκεκριμένη συμφωνία με τους τελικούς χρήστες για το σύνολο των λειτουργιών του λογισμικού.

πρωτοτυποποίηση (prototyping): μεθοδολογία ανάπτυξης λογισμικού, κατά την οποία αναπτύσσεται πολύ νωρίς ένα πρωτότυπο του συστήματος, χωρίς να έχει προηγηθεί διεξοδική ανάλυση απαιτήσεων. Το πρωτότυπο παραδίδεται στους χρήστες, οι οποίοι το χρη-

Γλωσσάρι όρων

σιμοποιούν και το σχολιάζουν. Με βάση τα σχόλια των χρηστών παράγονται οι τελικές απαιτήσεις και προδιαγραφές του λογισμικού. Η ανάπτυξη του συνολικού συστήματος μπορεί να επεκτείνει το πρωτότυπο (incremental development), ή να ξεκινά από την αρχή (τότε το πρωτότυπο καλείται throw away prototype).

σταθερά (constatnt): το όνομα μιας θέσης μνήμης, της οποίας τα περιεχόμενα δεν επιτρέπεται να αλλάξουν κατά την εκτέλεση του προγράμματος. Το όνομα, ο τύπος δεδομένων και η τιμή της θέσης μνήμης δηλώνονται στο τμήμα αρχικοποίησης του προγράμματος.

στοίβα (stack): δομή δεδομένων (υλοποιείται με διασυνδεδεμένη λίστα), στην οποία η πρόσθεση και η διαγραφή κόμβων γίνεται μόνο από το ένα άκρο της. Έτσι, οι κόμβοι διαγράφονται με σειρά αντίστροφη από τη σειρά με την οποία είχαν προστεθεί (δομή Last In First Out – LIFO).

στυλ προγραμματισμού (programming style): το σύνολο των προσωπικών προγραμματιστικών επιλογών και προτιμήσεων ενός προγραμματιστή, το οποίο περιγράφεται από ένα σύνολο κανόνων της μορφής «κάνε ...» ή «μην κάνεις ...». Τρία διαδεδομένα στυλ προγραμματισμού είναι ο προγραμματισμός για επαναχρησιμοποίηση, ο προγραμματισμός με πλεονασμό και ο αμυντικός προγραμματισμός

σύζευξη (coupling): μέτρο του βαθμού διασύνδεσης και αλληλεξάρτησης των τμημάτων ενός προγράμματος (χαμηλός βαθμός σύζευξης αυξάνει τη λειτουργική ανεξαρτησία). Εξαρτάται από την πολυπλοκότητα του τρόπου επικοινωνίας των τμημάτων και από τις παραμέτρους που περνιούνται κατά την κλήση ενός τμήματος από ένα άλλο. Επιθυμητή είναι η έλλειψη σύζευξης, αλλά μπορούμε να ανεχθούμε και σύζευξη δεδομένων ή σφραγίδας.

συνάρτηση (function): ένα υπο–πρόγραμμα, το οποίο επιστρέφει μια τιμή καταχωρημένη στο όνομά του.

σύνδεσμος (link): το πεδίο ενός κόμβου μιας διασυνδεδεμένης λίστας, το οποίο δείχνει στον επόμενο ή τον προηγούμενο κόμβο. Πρόκειται για μεταβλητή τύπου δείκτη. Κάθε κόμβος που ανήκει σε μια δομή λίστας έχει τουλάχιστον ένα σύνδεσμο.

συνθήκη (condition): απλή ή σύνθετη λογική έκφραση που μπορεί

Γλωσσάρι όρων

να πάρει τιμή 1 (TRUE) ή 0 (FALSE). Μία απλή λογική έκφραση χτίζεται χρησιμοποιώντας μεταβλητές ή σταθερές που συσχετίζονται με σχεσιακούς τελεστές (=, <, <=, >, >= < >), π.χ. a >= 5. Η σύνθετη λογική έκφραση χτίζεται συσχετίζοντας απλές εκφράσεις με λογικούς τελεστές (NOT, AND, OR και συνδυασμούς αυτών), π.χ. a >= 5 AND b < 8.

συνοχή (cohesion): μέτρο της λειτουργικής ανεξαρτησίας ενός προγράμματος (υψηλός βαθμός συνοχής αυξάνει τη λειτουργική ανεξαρτησία). Ένα συνεκτικό τμήμα επιτελεί όσο το δυνατό λιγότερες λειτουργίες μέσα στο πρόγραμμα (μία στην ιδανική περίπτωση) και χρειάζεται πολύ λίγη αλληλεπίδραση με άλλα τμήματα. Περισσότερο επιθυμητές μορφές συνοχής είναι η λειτουργική, η ακολουθιακή και η επικοινωνιακή.

σύστημα (system): ένα σύνολο συνεργαζόμενων τμημάτων, τα οποία δέχονται έναν αριθμό εισόδων από το περιβάλλον του συστήματος, τις μετασχηματίζουν εφαρμόζοντας μεθόδους επεξεργασίας και παράγουν έναν αριθμό εξόδων προς το περιβάλλον. Η συμπεριφορά ενός συστήματος μπορεί να εξαχθεί μελετώντας τις εξόδους σε σχέση με τις εισόδους του, ή αναλύοντας τις μεθόδους επεξεργασίας που εφαρμόζει. Στο παρόν ενδιαφερόμαστε για τα υπολογιστικά ή πληροφοριακά συστήματα, τα οποία δέχονται στην είσοδό τους δεδομένα, τα επεξεργάζονται εφαρμόζοντας αλγορίθμους και παράγουν στην έξοδό τους πληροφορία. Ένα πληροφοριακό σύστημα αποτελείται από συνεργαζόμενα τμήματα λογισμικού, υπολογιστές ή άλλες συσκευές στις οποίες εκτελείται το λογισμικό και ανθρώπους που το χρησιμοποιούν.

σφάλμα (error): μια τιμή ή συνθήκη που διαφέρει από την αναμενόμενη τιμή ή συνθήκη.

σχεδίαση προγράμματος (program design): φάση του κύκλου ζωής λογισμικού, κατά την οποία περιγράφεται, σε διάφορα επίπεδα λεπτομέρειας, η αρχιτεκτονική του λογισμικού και η συμπεριφορά κάθε τμήματος που το απαρτίζει. Η σχεδίαση χρησιμοποιεί τις απαιτήσεις των χρηστών και τις προδιαγραφές του λογισμικού. Πρόκειται για καθοριστική φάση, κατά την οποία λαμβάνονται αποφάσεις για τη διάσπαση του συνολικού συστήματος, τον τρόπο συγκρότησης του συστήματος από τα επιμέρους τμήματα, τις δια-

Γλωσσάρι όρων

δικασίες ελέγχου και αποδοχής, το υλικό και τον εξοπλισμό που απαιτεί το λογισμικό, κ.ά. Οι δραστηριότητες της σχεδίασης είναι απαραίτητες και πολύπλοκες, αλλά, είναι και πολύ δημιουργικές.

σχεδίαση / προγραμματισμός από κάτω προς τα πάνω (bottom – up design / programming): μεθοδολογία σχεδίασης κατά την οποία περιγράφονται πρώτα τα τμήματα που υλοποιούν τις κρίσιμες λειτουργίες του χαμηλότερου επιπέδου. Στη συνέχεια, τα τμήματα αυτά συγκροτούνται σε μεγαλύτερα τμήματα, τα οποία συγκροτούνται σε υπο–συστήματα κ.ο.κ. Αντίστοιχα, κατά τον προγραμματισμό από κάτω προς τα πάνω, υλοποιούνται πρώτα τα περισσότερο «εσωτερικά» τμήματα του προγράμματος, τα οποία στη συνέχεια συνενώνονται σε μεγαλύτερα τμήματα κ.ο.κ. Σε κάθε βήμα, το συνολικό πρόγραμμα αναπαρίσταται ως ένας οδηγός, ο οποίος χρησιμοποιεί τα διάφορα τμήματα.

σχεδίαση / προγραμματισμός από πάνω προς τα κάτω (top – down design / programming): μεθοδολογία σχεδίασης κατά την οποία ορίζεται πρώτα η λειτουργικότητα του συστήματος σε υψηλό επίπεδο. Στη συνέχεια, κάθε λειτουργία αναλύεται σε απλούστερες λειτουργίες, καθεμία από τις οποίες αναλύεται σε περισσότερο απλές λειτουργίες κ.ο.κ. Αντίστοιχα, κατά τον προγραμματισμό από πάνω προς τα κάτω, πρώτα αναπτύσσεται ο σκελετός του προγράμματος και στη θέση του κάθε υπο–προγράμματος τοποθετείται ένα στέλεχος, το οποίο εξομοιώνει τη συμπεριφορά του υπο–προγράμματος. Στη συνέχεια, κάθε στέλεχος αντικαθίσταται από τον κώδικα που υλοποιεί πραγματικά τις λειτουργίες του υπο–προγράμματος, το οποίο μπορεί με τη σειρά του να περιέχει άλλα στελέχη.

σχεδίαση / προγραμματισμός από τη μέση προς την άκρη (middle–out design / programming): μεθοδολογία σχεδίασης κατά την οποία σχεδιάζονται πρώτα τα γνωστά τμήματα ενός συστήματος, τα οποία μπορεί στη συνέχεια να διασπαστούν σε απλούστερα. Το συνολικό σύστημα χτίζεται από τη συγκρότηση αυτών των τμημάτων. Αντίστοιχα, κατά τον προγραμματισμό από τη μέση προς την άκρη, υλοποιούνται πρώτα τα γνωστά υποσυστήματα και έπειτα τα υπόλοιπα τμήματα, υψηλότερου ή χαμηλότερου επιπέδου.

σχόλια (comments): κείμενο που έχει ενσωματωθεί στον κώδικα ενός

Γλωσσάρι όρων

προγράμματος για να τεκμηριώσει τη λειτουργία του. Το κείμενο αυτό είναι συνήθως κατανοητό από προγραμματιστές, ενώ αγνοείται από τον μεταγλωττιστή της γλώσσας προγραμματισμού.

σωρός (heap): ένα πλήρες δυαδικό δένδρο, στο οποίο η τιμή ενός κόμβου είναι μεγαλύτερη από την τιμή κάθε παιδιού του. Η δομή αυτή χρησιμοποιείται κατά την εκχώρηση τμημάτων της μνήμης του υπολογιστή στις μεταβλητές των προγραμμάτων.

ταξινόμηση (sorting): η διάταξη, σύμφωνα με συγκεκριμένη σειρά, των στοιχείων ενός συνόλου ή μιας δομής δεδομένων. Υπάρχουν πολλοί αλγόριθμοι ταξινόμησης με διαφορετική απόδοση, όπως ταξινόμηση με επιλογή, ταξινόμηση φυσαλλίδας, γρήγορη ταξινόμηση κ.ά.

τεκμηρίωση (documentation): ένα σύνολο από έγγραφα, τα οποία παράγονται κατά τις διάφορες φάσεις του κύκλου ζωής ενός λογισμικού. Η τεκμηρίωση διακρίνεται σε:

- εξωτερική (external), στην οποία περιλαμβάνονται όλα τα έγγραφα που δημιουργούνται ανεξάρτητα από τον κώδικα, και
- εσωτερική (internal), η οποία περιλαμβάνει τον σχολιασμό της λειτουργίας των προγραμμάτων που ενσωματώνεται μέσα στον ίδιο τον κώδικα.

Η εξωτερική τεκμηρίωση διακρίνεται ακόμη σε:

- τεκμηρίωση για το χρήστη (user documentation), στην οποία περιλαμβάνονται όλα τα έγγραφα που περιγράφουν τις λειτουργίες του συστήματος (χωρίς να περιγράφουν τον τρόπο με τον οποίο αυτές υλοποιούνται), τα οποία αναπτύσσονται για να δοθούν στους χρήστες του συστήματος, και
- τεκμηρίωση για το σύστημα (system documentation), η οποία ομαδοποιεί όλα τα έγγραφα που περιγράφουν τεχνικές όψεις του συστήματος.

τεκμηρίωση χρήστη (user documentation): στην τεκμηρίωση που αναπτύσσεται για το χρήστη πρέπει να περιλαμβάνονται τουλάχιστον τα εξής πέντε έγγραφα: λειτουργική περιγραφή, οδηγός εγκατάστασης, εισαγωγικό εγχειρίδιο, εγχειρίδιο αναφοράς, και εγχειρίδιο διαχείρισης, ενώ τα σύγχρονα συστήματα λογισμικού συνή-

Γλωσσάρι όρων

θως συνοδεύονται και από κάρτες συνοπτικής αναφοράς λειτουργιών, άρθρα και εργασίες που έχουν συγγράψει έμπειροι χρήστες και περιγράφουν απόψεις περί την εφαρμογή του συστήματος και την εκτέλεση ασυνήθιστων εργασιών με το λογισμικό, βοήθεια σε ηλεκτρονική μορφή, ηλεκτρονικές διευθύνσεις δικτύου, στις οποίες ο εξουσιοδοτημένος χρήστης μπορεί να βρει πληροφορίες για την εξέλιξη του συστήματος ή να ζητήσει βοήθεια ή να ανταλλάξει απόψεις με άλλους χρήστες, κ.ά.

τελεστής (operator): σύμβολο το οποίο δηλώνει μια πράξη σε ένα ή δύο ορίσματα, τα οποία μπορεί να είναι αριθμητικές ή λογικές τιμές, μεταβλητές, σταθερές ή εκφράσεις. Παραδείγματα τελεστών είναι: +, -, *, /, NOT, OR, AND, =, >, >=, <, <=, < >, :=, κ.ά.

τελεστής έμμεσης προσπέλασης (dereferencing operator): τελεστής που εφαρμόζεται σε μια μεταβλητή τύπου δείκτη και επιστρέφει την τιμή του στοιχείου δεδομένων (της θέσης μνήμης) όπου δείχνει η τιμή της μεταβλητής.

τμήμα λογισμικού (software module): ένα σύνολο κώδικα και δεδομένων, το οποίο υλοποιεί μία λειτουργία. Το τμήμα λειτουργεί ως αυτόνομο μέρος ενός προγράμματος, η συμπεριφορά του οποίου δηλώνεται στη διεπαφή του, ενώ η υλοποίηση της συμπεριφοράς (δηλαδή ο κώδικας) είναι «κρυμμένος» από το υπόλοιπο πρόγραμμα.

τοπική δήλωση / μεταβλητή (local declaration / variable): η δήλωση μιας σταθεράς, μια μεταβλητής ή ενός υπο–προγράμματος μέσα σε ένα υπο–πρόγραμμα. Τότε, η εμβέλεια των δηλωθέντων καλύπτει μόνο το υπο–πρόγραμμα και όλα τα υπο–προγράμματα αυτού, ενώ η δήλωση δεν ισχύει στο κυρίως πρόγραμμα. Μια τοπική δήλωση μπορεί να έχει το ίδιο όνομα με μια σφαιρική δήλωση. Στην περίπτωση αυτή, για όσο χρόνο είναι ενεργό (εκτελείται) το υπο–πρόγραμμα ή κάποιο υπο–πρόγραμμα αυτού, ισχύει η τοπική δήλωση, ενώ μόλις ο έλεγχος επιστρέψει στο καλών πρόγραμμα, ισχύει η σφαιρική δήλωση.

τύπος δεδομένων (data type): ορίζει το επιτρεπόμενο πεδίο τιμών μιας μεταβλητής. Σιωπηρά, ορίζει και τις λειτουργίες που επιτρέπονται στα δεδομένα που διατηρεί η μεταβλητή αυτή. Οι περισσότερο συνηθισμένοι τύποι δεδομένων είναι Integer, Real, Character,

Γλωσσάρι όρων

Pointer, Boolean κ.ά.

υλικό ή υλισμικό (hardware): τα ορατά ή απτά συστατικά τμήματα ενός υπολογιστή που είναι ενσωματωμένα σε αυτόν ή αποτελούν τις περιφερειακές του συσκευές.

υπο–πρόγραμμα (subroutine): γενικός όρος για τμήμα κώδικα που αποτελείται από ένα σύνολο εντολών και ομαδοποιεί μια ή περισσότερες λειτουργίες. Ένα υπο–πρόγραμμα συνήθως χρησιμοποιείται (καλείται) από το κυρίως πρόγραμμα ή άλλα υπο–προγράμματα, ενώ και το ίδιο μπορεί να καλεί άλλα υπο–προγράμματα. Εάν απαιτείται επικοινωνία κατά την κλήση, αυτή γίνεται με το πέρασμα παραμέτρων. Τα υπο–προγράμματα συνήθως εμφανίζονται με τη μορφή διαδικασιών ή συναρτήσεων.

φωλιασμένη απόφαση (nested if): προγραμματιστική δομή απόφασης, το ένα τουλάχιστο από τα εξαγόμενα της οποίας περιέχει άλλη μία δομή απόφασης.

χαρακτήρας (character): βασικός τύπος δεδομένων. Μια μεταβλητή τύπου χαρακτήρα μπορεί να πάρει ως τιμή οποιοδήποτε ψηφίο, γράμμα, ή άλλο ειδικό σημείο υποστηρίζει ο υπολογιστής. Εάν η τιμή είναι ένα σύνολο (μια ακολουθία) χαρακτήρων, τότε η μεταβλητή ορίζεται ως τύπου string.

ψευδοκώδικας (pseudocode): τεχνική λεκτικής περιγραφής αλγορίθμων, η οποία χρησιμοποιεί συντακτικές δομές γλωσσών προγραμματισμού, συμβολισμό από τη Μαθηματική Λογική και ένα αυστηρά καθορισμένο υποσύνολο των λέξεων μιας φυσικής γλώσσας. Συνδυάζει, λοιπόν, την ακρίβεια και αυστηρότητα των γλωσσών προγραμματισμού με την εκφραστική δύναμη της φυσικής γλώσσας. Ο ψευδοκώδικας δεν είναι εκτελέσιμος από τον υπολογιστή.

ψηφίδα λογισμικού (software component): τμήμα λογισμικού που έχει κωδικοποιηθεί με τρόπο ώστε να είναι απευθείας επαναχρησιμοποιήσιμο.

Βιβλιογραφία

ΕΛΛΗΝΟΓΛΩΣΣΗ

[1] Adams D. (1979), *The Hitch-hiker's guide to the Galaxy*. Ελλ. Μετ. εκδ. Ars-Longa.

[2] Ιωαννίδης Ν, Μαρινάκης Κ. και Μπακογιάννης Σ. (1980), *Σχεδίαση Προγράμματος (δομημένη και αλγοριθμική)*. Εκδόσεις Νέων Τεχνολογιών - Ελιξ

[3] Kernighan B. and Ritchie D. (1988), *Η γλώσσα προγραμματισμού* C. Prentice-Hall (Ελληνική έκδοση: Κλειδάριθμος).

[4] Μπεμ Α. και Καραμπατζός Γ. (1991), *Εισαγωγή στην πληροφορική,* Συμμετρία:Αθήνα.

[5] Σκορδαλάκης Ε. (1991), *Εισαγωγή στην Τεχνολογία Λογισμικού,*. Εκδόσεις Συμμετρία, Αθήνα.

ΞΕΝΟΓΛΩΣΣΗ

[6] Bohm C. and Jacopini G. (1966), *Flow Diagrams, Turing Machines, and Languages with only two formation rules*. Communications of the ACM, 9(5), σελ. 366-371.

[7] Caine S. and Gordon K. (1975), PDL: *a tool for software design*. Proceedings of National Computer Conference, IFIP Press, σελ. 271-276.

[8] Chapin (1974) N. , *A new format for Flowcharts*. Software Practice & Experience, 4(4), σελ. 341-357.

[9] Constantine L. and Yourdon E. (1979), *Structured Design*. Prentice-Hall: Englewood Cliffs.

[10] Cooper D. and Clancy M. (1985), *Oh ! Pascal !*. Norton & Company, NY.

[11] Davis W.S. (1983), *Systems Analysis and Design: a structured approach*. Addison-Wesley

[12] Dijkstra E.W. (1968), *GOTO Statement Considered Harmful.* Communications of the ACM, 11(3), σελ. 147-148.

[13] Fairley R. (1985), *Software Engineering Concepts*. McGraw-Hill: Singapore.

[14] IBM (1974), *Improved Programming Technologies-an overview.* GC20-1850-0. IBM Co., White Plains, NY.

Βιβλιογραφία

[15] IBM (1975), *HIPO-a Design Aid and Documentation Technique.* GC20-1851-1. IBM Co., White Plains, NY.

[16] Jackson M. (1975), *Principles of Program Design*. Academic Press.

[17] Knuth D. (1973), *The Art of Computer Programming: Fundamental Algorithms.* Addison-Wesley, Reading:MA.

[18] Knuth D. (1973), *The Art of Computer Programming: Sorting & Searching*. Addison-Wesley, Reading:MA.

[19] Myers G. (1978), *Composite / Structured Design. V*an Nostrand Reinhold: New York.

[20] Nassi K. and Shneiderman B. (1973), *Flowchart Techniques for Structured Programming.* ACM SIGPLAN Notices.

[21] Orr K. (1978), *Structured Design*. Yourdon Press, NY.

[22] Pintelas P.E. (1978), *Lecture notes on program schemas*. Computer Education journal, June 78.

[23] Pressman R. (1994), *Software Engineering: a practitioner's approach*. McGraw-Hill, England.

[24] Shooman M.L. (1983), *Software Engineering*, McGraw-Hill:Tokyo, Japan.

[25] Stevens W, Myers G. and Constantine L. (1974), *Structured Design*. IBM Systems Journal, 13(2), σελ. 115-139.

[26] Sommerville I. (1996), *Software Engineering*. Addison-Wesley, USA

[27] Tanenbaum A. (1984), *Structured Computer Organization*. Prentice Hall: New Jersey.

[28] Vessey L. and Glass R. (1998), *Strong vs. Weak approaches to systems development*. Communications of the ACM, 41(4), σελ. 99-102.

[29] Warnier J.D. (1974), *Logical Construction of Programs*.Van Nostrand Reinhold, NY.

[30] Wirth N. (1971), *Program development with Stepwise Refinement.* Communications of the ACM, 14(4).