ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ

ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΜΑΤΟΣ

Κεφάλαιο 1ο Ανάλυση Προβλήματος

1.1. Η έννοια πρόβλημα

Σημαντικός είναι ο ορισμός (σελ. 3 σχολικού βιβλίου). Υπάρχουν προβλήματα σε οποιοδήποτε τομέα. Με την ανάπτυξη της πληροφορικής και τη μηχανογράφηση η αντιμετώπιση ορισμένων προβλημάτων γύρω μας γίνεται ευκολότερη και αποτελεσματικότερη. Βέβαια παρά τη βοήθεια και υποστήριξη που μας παρέχει η πληροφορική, στο χώρο των Η/Υ εμφανίζονται και νέα προβλήματα (όπως ο ιός του 2000).

1.2. Κατανόηση προβλήματος

Η κατανόηση ενός προβλήματος ή μιας άσκησης είναι το σημαντικότερο και πολλές φορές το δυσκολότερο σημείο. Π.χ. σε μια άσκηση είναι δυνατό να γνωρίζουμε καλά τις εντολές, ο αλγόριθμος που γράψαμε να είναι σωστός ή λογικός αλλά να μη δίνουμε λύση στο πρόβλημα αυτό καθαυτό έτσι ακριβώς όπως τίθεται (δηλ. να μην απαντάμε σ' αυτά που μας ζητούνται). Η κατανόηση ενός προβλήματος εξαρτάται σε μεγάλο βαθμό από τη διατύπωσή του. Τελικά η σωστή κατανόηση απαιτεί σωστή διατύπωση και σωστή ερμηνεία. Σε ένα πρόβλημα (ή άσκηση) πάντα να δίνετε μεγάλη προσπάθεια και τον απαραίτητο χρόνο στην κατανόησή του. Είναι σημαντικό να είστε βέβαιοι ότι έχετε κατανοήσει σωστά ποιο είναι το πρόβλημα και τι ζητείται τελικά.

Σημαντικοί είναι οι ορισμοί των όρων **δεδομένο**, **πληροφορία** και **επεξεργασία δεδομένων** (σελ. 8).

1.3. Δομή προβλήματος

Σημαντικός είναι ο ορισμός (σελ. 8). Η αντιμετώπιση ενός προβλήματος γίνεται ευκολότερη αν μπορέσουμε να το αναλύσουμε σε άλλα μικρότερα ή απλούστερα. Π.χ. σε μια άσκηση είναι δυνατό να `σπάσουμε` έναν πολύπλοκο αλγόριθμο σε επιμέρους τμήματα αλγορίθμων. Η ανάλυση αυτή αναδύει παράλληλα και τη δομή του προβλήματος. Ένα πρόβλημα μπορεί να περιγραφεί (και να αναλυθεί) **φραστικά** (δηλ. με λόγια π.χ. σελ. 9-10) ή **διαγραμματικά** (διαγραμματική αναπαράσταση π.χ. σχήμα 1.1 σελ. 11). Η δόμηση ενός προβλήματος συνδέεται και με άλλες έννοιες του κεφαλαίου 6 (βλ. παρ. 6.4.1, 6.4.2, 6.4.3 σελ. 132-133).

Σε γενικές γραμμές η αντιμετώπιση ενός προβλήματος αποτελείται από τις ενέργειες του παρακάτω σχήματος:

- Οι είσοδοι αφορούν καταχώρηση των δεδομένων του προβλήματος.
- Τα δεδομένα ελέγχονται και ίσως χρειαστεί επανακαταχώρηση.
- Γίνονται οι απαραίτητες επεξεργασίες ή υπολογισμοί προκειμένου να βρεθούν τα ζητούμενα αποτελέσματα (πληροφορίες).
- Οι έξοδοι αφορούν εξαγωγή των επιθυμητών αποτελεσμάτων.

1.4. Καθορισμός απαιτήσεων

Αφορά πρώτα τον ακριβή προσδιορισμό των **δεδομένων** ή εισόδων που υπάρχουν σ' ένα πρόβλημα

και ύστερα τον ακριβή προσδιορισμό των **ζητούμενων** ή εξόδων (δηλ. των πληροφοριών ή αποτελεσμάτων που επιλύουν το πρόβλημα). Π.χ. σε μια άσκηση ο καθορισμός των απαιτήσεων μπορεί να περιλαμβάνει ορισμό δεδομένων-μεταβλητών, πινάκων, κλπ. Πόσες και ποιες μεταβλητές θα χρησιμοποιήσω; Μήπως χρειάζομαι δομή δεδομένων (π.χ. πίνακα); Τι πρέπει να εκτυπώνεται τελικά; Αυτά είναι μερικά παραδείγματα ερωτημάτων που πρέπει να μας απασχολούν. Επομένως ο καθορισμός των απαιτήσεων πρέπει να γίνεται πάντα.

1.5. Κατηγορίες προβλημάτων

Είναι αρκετά σημαντικές όλες οι κατηγορίες. Προσέξτε τα κριτήρια με βάση τα οποία ταξινομούμε τα προβλήματα. Πρακτικά η πληροφορική δεν παρέχει ιδιαίτερη υποστήριξη για τα άλυτα ούτε για τα αδόμητα προβλήματα. Είναι όμως σημαντικός ο ρόλος της στα επιλύσιμα, στα δομημένα (αυτοματοποιημένα) και στα προβλήματα βελτιστοποίησης.

1.6. Πρόβλημα και υπολογιστής

Οι υπολογιστές δεν επιλύουν προβλήματα με `μαγικό` τρόπο. Προβλήματα λυνόντουσαν και πριν τους υπολογιστές. Οι υπολογιστές (και η επιστήμη της πληροφορικής) δρουν επικουρικά ή υποστηρικτικά στην επίλυση προβλημάτων. Π.χ. αν κλαπεί ένα αυτοκίνητο η πληροφορική δε μπορεί να δώσει άμεση λύση! Σε περίπτωση όμως που βρεθεί το αυτοκίνητο και μάλιστα σε περιοχή ευθύνης διαφορετικού αστυνομικού τμήματος από εκείνο που δηλώθηκε η κλοπή, τότε συνήθως ο κάτοχος του οχήματος ταλαιπωρείται. Πρέπει με δική του ευθύνη να `οργώσει` όσα περισσότερα αστυνομικά τμήματα μπορεί προκειμένου να ενημερώνεται. Αν όμως όλα τα αστυνομικά τμήματα ήταν συνδεδεμένα μεταξύ τους σ' ένα δίκτυο και υπήρχε κατάλληλο πρόγραμμα επικοινωνίας και ανταλλαγής πληροφοριών, τότε τα πράγματα θα ήταν πολύ καλύτερα.

Ο άνθρωπος υπερέχει ποιοτικά από τον υπολογιστή. Ο υπολογιστής μπορεί να εκτελεί μόνο αριθμητικές πράξεις (ουσιαστικά πρόσθεση), λογικές πράξεις (ουσιαστικά σύγκριση) και μεταφορά δεδομένων. Οι πράξεις βέβαια μπορούν να εκτελούνται ταχύτατα στον υπολογιστή. Επίσης ο υπολογιστής διαθέτει κύρια μνήμη (RAM) και περιφερειακές μνήμες (π.χ. δίσκοι). Άρα ο υπολογιστής διαθέτει τρομακτικά ισχυρούς μηχανισμούς για αποθήκευση και μεταφορά δεδομένων. Επομένως μπορούμε να αναθέσουμε ένα πρόβλημα σε έναν υπολογιστή όταν:

- Υπάρχουν πολύπλοκοι υπολογισμοί.
- Υπάρχουν διαδικασίες που επαναλαμβάνονται (αυτοματοποιούνται).
- Υπάρχουν πολλές πράξεις (αριθμητικές ή λογικές).
- Υπάρχει μεγάλο πλήθος (όγκος) δεδομένων.

Κεφάλαιο 2ο Βασικές Έννοιες Αλγορίθμων

2.1 Τι είναι αλγόριθμος

Σημαντικός είναι ο ορισμός (σελ. 25) και τα κριτήρια που πρέπει να ικανοποιεί ένας αλγόριθμος (σελ. 25-26):

- Είσοδος (**δεδομένα** που εισάγονται-κανένα ή συνήθως ένα/ πολλά.
- Έξοδος (**πληροφορίες**/ αποτελέσματα που εξάγονται-μία ή περισσότερες).
- Καθοριστικότητα (π.χ. να μην υπάρχουν εντολές που σε κάποια περίπτωση δεν ορίζονται, όπως διαίρεση με το μηδέν).

- Περατότητα (π.χ. να μην υπάρχουν δομές επανάληψης που δεν τελειώνουν ποτέ).
- Αποτελεσματικότητα (κάθε εντολή να είναι απλή, να μπορεί να εκτελείται και να δίνει σωστό αποτέλεσμα).

2.2 Σπουδαιότητα αλγορίθμων

Είναι αρκετά σημαντικές όλες οι σκοπιές (σελ. 27) με βάση τις οποίες η Πληροφορική μελετά τους αλγορίθμους.

2.3 Περιγραφή και αναπαράσταση αλγορίθμων

- Ελεύθερο κείμενο (δίχως κανόνες). Δε χρησιμοποιείται σχεδόν καθόλου αφού έτσι οι εντολές είναι ασαφείς και δεν είναι εκτελέσιμες (αναποτελεσματικότητα).
- Διαγραμματικές τεχνικές (π.χ. διαγράμματα ροής). Χρησιμοποιούνται αρκετά αν και κάπως έχουν ξεπεραστεί. Είναι δυνατόν να μετατραπούν σχετικά εύκολα σε σειρά εντολών οποιασδήποτε γλώσσας προγραμματισμού γενικής χρήσης (π.χ. Pascal, Basic, C, κλπ).
- Φυσική γλώσσα (π.χ. αγγλικά). Είναι περίπου το ίδιο με το ελεύθερο κείμενο (σε πολλά βιβλία δεν υπάρχει διάκριση). Δε χρησιμοποιείται σχεδόν καθόλου αφού έτσι οι εντολές είναι ασαφείς και δεν είναι καλά ορισμένες (δεν υπάρχει καθοριστικότητα). Αντί για φυσική γλώσσα χρησιμοποιείται ευρύτατα η **ψευδογλώσσα**.
- Κωδικοποίηση δηλ. απ' ευθείας δημιουργία κώδικα (εντολών) προγράμματος σε μία γλώσσα προγραμματισμού. Χρησιμοποιείται πάντα αν θέλουμε ο αλγόριθμος να μην υπάρχει μόνο σε θεωρητικό επίπεδο.
- Ψευδογλώσσα δηλ. μία θεωρητική (υποθετική) δομημένη γλώσσα προγραμματισμού. Η ψευδογλώσσα μπορεί να μετατραπεί πολύ εύκολα σε σειρά εντολών οποιασδήποτε υπάρχουσας γλώσσας προγραμματισμού γενικής χρήσης (π.χ. Pascal, Basic, C, κλπ).

2.4 Βασικές συνιστώσες/ εντολές ενός αλγορίθμου

Σημαντικά είναι τα <u>βασικά στοιχεία</u> μίας γλώσσας προγραμματισμού ή μίας ψευδογλώσσας δηλ. σταθερές, μεταβλητές, τελεστέοι, τελεστές κι εκφράσεις (σελ. 31). Τα στοιχεία αυτά περιγράφονται πιο αναλυτικά στο κεφάλαιο 7 (βλ. παρ. 7.1-7.7 σελ. 148-154):

- Σταθερές (π.χ. 100, 1821, 3,14, "ψηλός", "ανύπαρκτη τιμή", "English language", αληθής, ψευδής).
- Μεταβλητές (αριθμητικές, αλφαριθμητικές, λογικές).
- Τελεστέοι (δηλ. σταθερές ή μεταβλητές με μία λέξη).
- Τελεστές (π.χ. +, -, *, /, και, ή, όχι, =, <, <=, >, >=, <>).
- Εκφράσεις (π.χ. 500, a, a=0, x+y, (i<=50 ή done=ψευδής), κλπ).

<u>**Δομή ακολουθίας**</u>: Ονομάζεται και σειριακή ή ακολουθιακή δομή. Αποτελείται από ένα σύνολο εντολών που τοποθετούνται η μία κάτω από την άλλη. Χρησιμοποιείται (από μόνη της) για την επίλυση πολύ απλών προβλημάτων όπου η σειρά εκτέλεσης ενός συνόλου ενεργειών είναι δεδομένη. Χρησιμοποιείται ευρύτατα <u>σε συνδυασμό</u> με άλλες δομές (επιλογής, επανάληψης). Στη δομή αυτή ανήκουν οι εντολές **εισόδου**/ **εξόδου** (δηλ. **διάβασε**/ **εμφάνισε**) κι οι εντολές **ανάθεσης ή απόδοσης τιμής** (δηλ. Μεταβλητή ← Έκφραση). Οι εντολές αυτές περιγράφονται πιο αναλυτικά στο κεφάλαιο 7 (βλ. παρ. 7.8-7.9 σελ. 154-156).

Δομή επιλογής: Αποτελείται από ένα σύνολο εντολών που εκτελούνται κατά περίπτωση. Η διαδικασία της επιλογής περιλαμβάνει τον έλεγχο κάποιας συνθήκης με δύο δυνατές τιμές (αληθής, ψευδής) και στη συνέχεια την απόφαση εκτέλεσης κάποιας εντολής ανάλογα με τη συνθήκη. Στη δομή αυτή ανήκουν οι εντολή **Αν** κι η εντολή **Επίλεξε**. Οι εντολές αυτές περιγράφονται πιο αναλυτικά στο κεφάλαιο 8 (βλ. παρ. 8.1 σελ. 166-173).

Η εντολή **Αν** συντάσσεται (στη γενική της μορφή) ως εξής:

Αν Συνθήκη_1 **τότε**
    Εντολή_1_1
    [Εντολή_1_2] Σύνολο Εντολών 1
    ……
[**αλλιώς_αν** Συνθήκη_2 **τότε**
    Εντολή_2_1
    [Εντολή_2_2] Σύνολο Εντολών 2
    ……
]
……
[**αλλιώς**
    Εντολή_n_1
    [Εντολή_n_2] Σύνολο Εντολών n
    ……
]
Τέλος_αν

Οι λέξεις που είναι έντονες λέγονται **δεσμευμένες** λέξεις. Αυτό σημαίνει ότι είναι δεσμευτικές και δε μπορούν να αντικατασταθούν με άλλη λέξη. Δηλ. δεν επιτρέπεται να χρησιμοποιήσουμε δικές μας λέξεις αντί γι' αυτές ούτε μπορούμε να τις παραλείψουμε. Π.χ. δε μπορούμε να αντικαταστήσουμε το **Αν** με το Εφόσον ούτε μπορούμε να παραλείψουμε το **τότε**. Οι τετράγωνες αγκύλες [] σημαίνουν πως οτιδήποτε περιέχεται σ' αυτές είναι προαιρετικό. Π.χ. τα τμήματα **αλλιώς_αν** και **αλλιώς** δεν υπάρχουν οπωσδήποτε (υποχρεωτικά) σε μια εντολή **Αν**. Το τμήμα **αλλιώς** ερμηνεύεται ως `σε κάθε άλλη περίπτωση`. Οι συνθήκες δεν είναι τίποτα άλλο παρά λογικές εκφράσεις (π.χ. a=0, x+y>10, (j>=5 και found=αληθής), κλπ).

Σε περίπτωση που η συνθήκη ισχύει (δηλ. είναι αληθής) εκτελείται το αντίστοιχο σύνολο εντολών (που περιέχει τουλάχιστο μία εντολή). Σε μία εντολή **Αν** κάθε φορά εκτελείται ένα από τα σύνολα εντολών κατά περίπτωση. Η μοναδική περίπτωση στην οποία δεν εκτελείται κανένα σύνολο εντολών είναι να μην ισχύει καμία συνθήκη και να μην υπάρχει καθόλου το τμήμα **αλλιώς**. Εντολή μπορεί να είναι μία οποιαδήποτε αποδεκτή εντολή (π.χ. εισόδου/ εξόδου, ανάθεσης τιμής, επιλογής, επανάληψης, κλπ). Σε περίπτωση που μία από τις εντολές είναι άλλη εντολή **Αν** τότε έχουμε **Εμφωλευμένο Αν**. Προφανώς μπορούμε να εμφωλεύσουμε πολλές εντολές **Αν** μέσα σε μία εντολή **Αν** (κάθε μία πρέπει να έχει δικό της **Τέλος_αν**). Το εμφωλευμένο **Αν** για να μην οδηγήσει σε πιθανά λάθη πρέπει να χρησιμοποιείται για να καλύψει μία ξεκάθαρη υποπερίπτωση μιας περίπτωσης (π.χ. **Αν** city="Άγρα" **τότε** …… και μετά εμφώλευση του **Αν** class = "Γ Λυκείου" **τότε** …).

Συνηθισμένα τεχνικά λάθη στη χρήση της εντολής **Αν**:

α) **Αν** a>0 **τότε**
        εμφάνισε "α θετικός"
    αλλιώς_αν b<=0 **τότε**

εμφάνισε “β αρνητικός”

Τέλος_αν

Εδώ δεν υπάρχει κανένα λάθος στη σύνταξη της εντολής η οποία είναι αποδεκτή. Σε περίπτωση όμως που a<=0 και b>0 δεν εκτυπώνεται τίποτα! Αυτό συμβαίνει διότι υπάρχουν δύο συνθήκες στην ίδια εντολή που ελέγχουν όμως τελείως διαφορετικά πράγματα (το a και το b). Απαιτούνται λοιπόν δύο απλές και ξεχωριστές εντολές **Αν**. Επομένως για την αποφυγή λαθών καλό είναι οι συνθήκες σε μία εντολή να μην ελέγχουν διαφορετικά (σε νόημα) πράγματα.

β) **Αν** a>0 **τότε**

Αν a<=0 **τότε**

εμφάνισε “α θετικός”

Τέλος_αν

Τέλος_αν

Ούτε εδώ δεν υπάρχει κανένα λάθος στη σύνταξη αλλά υπάρχει αλόγιστη χρήση εμφωλευμένου **Αν**. Το δεύτερο **Αν** δεν πρόκειται να εκτελεστεί ποτέ! Το παραπάνω τμήμα αλγορίθμου λοιπόν δεν έχει κανένα νόημα. Επομένως για την αποφυγή λαθών καλό είναι τα εμφωλευμένα **Αν** να ελέγχουν διαφορετικά (σε νόημα) πράγματα. Πιο συγκεκριμένα πρέπει να ελέγχουν συγκεκριμένες (διακριτές) υποσυνθήκες της συνθήκης στην οποία περιέχονται (π.χ. για συνταξιοδότηση γυναικών χρησιμοποιώ τις εντολές **Αν** gender=“female” **τότε** … και μετά εμφώλευση του **Αν** age >= 63 **τότε** …).

Η εντολή **Επίλεξε** ή **Επέλεξε** συντάσσεται (στη γενική της μορφή) ως εξής:

Επίλεξε έκφραση
Περίπτωση Λίστα_Τιμών_1
Εντολή_1_1
[Εντολή_1_2] Σύνολο Εντολών 1
……
[**Περίπτωση** Λίστα_Τιμών_2
Εντολή_2_1
[Εντολή_2_2] Σύνολο Εντολών 2
……
]
……
[**Περίπτωση Αλλιώς**
Εντολή_n_1
[Εντολή_n_2] Σύνολο Εντολών n
……
]
Τέλος_επιλογών

Οι λέξεις που είναι έντονες είναι πάλι **δεσμευμένες** λέξεις. Η έκφραση συνήθως είναι μία μεταβλητή ή γενικότερα μία λογική έκφραση. Η λίστα τιμών μπορεί να είναι μία απλή σταθερά ή μεταβλητή (π.χ. 5, 100, a), ένα

εύρος τιμών (>=30, <200, {1,2,3}, κλπ). Σε περίπτωση που η έκφραση επιβεβαιώνει τη λίστα τιμών εκτελείται το αντίστοιχο σύνολο εντολών (που περιέχει τουλάχιστο μία εντολή).

Σε μία εντολή **Επέλεξε** κάθε φορά εκτελείται ένα από τα σύνολα εντολών κατά περίπτωση. Η μοναδική περίπτωση στην οποία δεν εκτελείται κανένα σύνολο εντολών είναι να μην επιβεβαιώνεται καμία λίστα τιμών και να μην υπάρχει καθόλου το τμήμα **Περίπτωση Αλλιώς**. Το τμήμα **Περίπτωση Αλλιώς** ερμηνεύεται ως `σε κάθε άλλη περίπτωση`. Είναι προφανές πως η εντολή **Επίλεξε** είναι παρόμοια με την εντολή **Αν**. Επομένως είναι δυνατόν να μετατρέψουμε μία εντολή **Αν** σε εντολή **Επίλεξε** κι αντίστροφα. Στις περισσότερες όμως γλώσσες προγραμματισμού η εντολή **Αν** εκφράζει τη γενική (βασική) δομή επιλογής κι η εντολή **Επίλεξε** είναι βοηθητική ή δεν είναι καν διαθέσιμη. Η εντολή **Επίλεξε** είναι πολύ χρήσιμη όταν έχουμε να διαλέξουμε τιμές μιας μεταβλητής από μία (ή περισσότερες) λίστες τιμών, διαφορετικά καλό είναι να αποφεύγεται.

Δομή επανάληψης: Αποτελείται από ένα σύνολο εντολών που εκτελούνται πολλές φορές (αυτοματοποιημένα). Εφαρμόζεται όταν μια σειρά εντολών πρέπει να εκτελεστεί σε ένα σύνολο περιπτώσεων, που έχουν κάτι κοινό. Η διαδικασία της επανάληψης είναι πολύ συχνή. Στη δομή αυτή ανήκουν η εντολή **Όσο**, η εντολή **Αρχή_επανάληψης** κι η εντολή **Για**. Οι εντολές αυτές περιγράφονται πιο αναλυτικά στο κεφάλαιο 8 (βλ. παρ. 8.2 σελ. 173-181).

Η εντολή **Όσο** συντάσσεται (στη γενική της μορφή) ως εξής:

Όσο Συνθήκη **επανέλαβε**

Εντολή_1

[Εντολή_2] Σύνολο Εντολών

……

Τέλος_επανάληψης

Οι λέξεις που είναι έντονες είναι πάλι **δεσμευμένες** λέξεις. Οι τετράγωνες αγκύλες [] σημαίνουν πάλι πως οτιδήποτε περιέχεται σ' αυτές είναι προαιρετικό. Η συνθήκη είναι μία λογική έκφραση (που είναι είτε αληθής είτε ψευδής). Πρέπει να προσέχουμε οι μεταβλητές που υπάρχουν στη συνθήκη να έχουν πάρει τιμές (π.χ. με χρήση εντολών εισόδου ή ανάθεσης τιμής) πριν την εκτέλεση της εντολής **Όσο**. Το σύνολο εντολών είναι δυνατό να εκτελεστεί από 0 έως άπειρες φορές. Βέβαια πρέπει να εξασφαλίσουμε ότι η συνθήκη είναι τέτοια που δεν θα υπάρχει πρόβλημα περατότητας (βλ. παρ. 2.1).

Η εντολή **Όσο** λειτουργεί ως εξής: Αρχικά ελέγχεται η συνθήκη. Αν είναι αληθής τότε εκτελείται το σύνολο εντολών. Στη συνέχεια ελέγχεται εκ νέου η συνθήκη κι αν είναι αληθής τότε εκτελείται πάλι το σύνολο εντολών. Όταν η συνθήκη γίνει ψευδής τότε τελειώνει η επανάληψη κι ο αλγόριθμος συνεχίζεται με την εντολή που ακολουθεί το **Τέλος_επανάληψης**.

Η εντολή **Αρχή_επανάληψης** συντάσσεται (στη γενική της μορφή) ως εξής:

Αρχή_επανάληψης

Εντολή_1

[Εντολή_2] Σύνολο Εντολών

……

Μέχρις_ότου Συνθήκη

Η συνθήκη είναι προφανώς πάλι μία λογική έκφραση (που είναι είτε αληθής είτε ψευδής). Το σύνολο εντολών είναι δυνατό να εκτελεστεί από 1 έως άπειρες φορές. Βέβαια πρέπει να εξασφαλίσουμε ότι η συνθήκη είναι τέτοια που δεν θα υπάρχει πρόβλημα περατότητας (βλ. παρ. 2.1).

Η εντολή **Αρχή_επανάληψης** λειτουργεί ως εξής: Αρχικά εκτελείται το σύνολο εντολών. Στη συνέχεια

ελέγχεται η συνθήκη κι αν είναι ψευδής τότε εκτελείται πάλι το σύνολο εντολών (υπάρχει αντίθεση με την εντολή **Όσο**!). Όταν η συνθήκη γίνει αληθής τότε τελειώνει η επανάληψη κι ο αλγόριθμος συνεχίζεται με την εντολή που ακολουθεί το **Μέχρις_ότου**. Η βασική λοιπόν διαφορά με την εντολή **Όσο** είναι η θέση της συνθήκης στη ροή εκτέλεσης των εντολών.

Η εντολή **Για** συντάσσεται (στη γενική της μορφή) ως εξής:

Για μετρητής **από** κάτω_όριο **μέχρι** άνω_όριο [**με_βήμα** βήμα]

Εντολή_1

[Εντολή_2] Σύνολο Εντολών

……

Τέλος_επανάληψης

Ο μετρητής είναι μία μεταβλητή (συνήθως ακέραιος). Το κάτω_όριο είναι μία σταθερά ή μεταβλητή (συνήθως ακέραιος) που καθορίζει την αρχική τιμή του μετρητή. Το άνω_όριο είναι μία σταθερά ή μεταβλητή (συνήθως ακέραιος) που καθορίζει την τελική τιμή του μετρητή. Το βήμα είναι μία σταθερά ή μεταβλητή (συνήθως ακέραιος) που καθορίζει πως θα μεταβάλλεται ο μετρητής. Το βήμα είναι προαιρετικό κι αν δε σημειώνεται είναι ίσο με 1 (δηλ. κάθε φορά ο μετρητής αυξάνεται κατά 1). Το σύνολο εντολών είναι δυνατό να εκτελεστεί τόσες φορές όσες καθορίζει ο μετρητής (0 ή περισσότερες φορές).

Η εντολή **Για** λειτουργεί ως εξής: Αρχικά ο μετρητής παίρνει αρχική τιμή ίση με το κάτω_όριο. Αν ο μετρητής είναι μικρότερος ή ίσος από το άνω όριο εκτελείται το σύνολο εντολών. Στη συνέχεια αυξάνεται ο μετρητής κατά 1 (ή κατά βήμα αν αυτό υπάρχει). Αν ο μετρητής είναι μικρότερος ή ίσος από το άνω όριο εκτελείται πάλι το σύνολο εντολών. Όταν ο μετρητής γίνει μεγαλύτερος από το άνω_όριο τότε τελειώνει η επανάληψη κι ο αλγόριθμος συνεχίζεται με την εντολή που ακολουθεί το **Τέλος_επανάληψης**.

Η γενικότερη και ισχυρότερη εντολή επανάληψης είναι η εντολή **Όσο**. Με αυτήν μπορούμε να πραγματοποιήσουμε οποιαδήποτε επαναληπτική διαδικασία. Αυτό βέβαια δε σημαίνει ότι η εντολή Όσο είναι πάντα η ευκολότερη στη χρήση. Η εντολή **Αρχή_επανάληψης** εξυπηρετεί καλύτερα αν θέλουμε ένα σύνολο εντολών να εκτελεστεί τουλάχιστο μία φορά και είναι προτιμότερο η συνθήκη να ελέγχεται μετά τις εντολές αυτές. Η εντολή **Για** εξυπηρετεί πάρα πολύ αν δε θέλουμε να ελέγχουμε εμείς τη ροή του προγράμματος με συνθήκες αλλά γνωρίζουμε ότι το σύνολο εντολών πρέπει να εκτελεστεί συγκεκριμένες φορές (π.χ. 5, 10 κλπ). Είναι δυνατό να μετατρέψουμε μία από τις εντολές επανάληψης στις άλλες δύο μορφές (π.χ. μία εντολή **Για** στις εντολές **Όσο** ή **Αρχή_επανάληψης**).

Επιπλέον στοιχεία ψευδογλώσσας:

Στην πρώτη γραμμή ενός αλγορίθμου βάζουμε πάντα **Αλγόριθμος** όνομα_αλγορίθμου όπου όνομα_αλγορίθμου είναι ένα λεκτικό που περιγράφει (εν συντομία) τον αλγόριθμο. Στην τελευταία γραμμή ενός αλγορίθμου βάζουμε πάντα **Τέλος** όνομα_αλγορίθμου.

Στην δεύτερη γραμμή ενός αλγορίθμου μπορούμε να βάλουμε (αν χρειάζεται) τα δεδομένα εισόδου εντός των συμβόλων //……// (π.χ. Δ**εδομένα** // x, y, table //). Αυτό το κάνουμε αν π.χ. έχουμε δεδομένη την τιμή μιας μεταβλητής ή δεδομένο έναν πίνακα (γεμάτο με τιμές) και δε χρειάζεται να χρησιμοποιήσουμε εντολές εισόδου (δηλ. **διάβασε**). Στην προτελευταία γραμμή ενός αλγορίθμου μπορούμε να βάλουμε (αν χρειάζεται) τα αποτελέσματα εξόδου εντός των συμβόλων //……// (π.χ. **Αποτελέσματα** // min, max, table //). Αυτό το κάνουμε αν π.χ. σ' ένα αλγόριθμο έχουμε στην έξοδο ως πληροφορίες την τιμή μιας μεταβλητής ή ενός πίνακα και δε θέλουμε να χρησιμοποιήσουμε εντολές εξόδου (δηλ. **εμφάνισε**).

Κεφάλαιο 3ο Δομές Δεδομένων και Αλγόριθμοι

3.1. Δεδομένα

Δεδομένα ή data (είσοδοι) είναι ακατέργαστα γεγονότα και η επιλογή τους εξαρτάται από τον τύπο του προβλήματος. Τα δεδομένα είναι στοιχεία που είναι γνωστά, ή γίνονται συνήθως εύκολα αντιληπτά από τις αισθήσεις μας και περιγράφουν την πραγματικότητα. Πληροφορίες (information) ή αποτελέσματα (εξόδους) παίρνουμε μετά την επεξεργασία των δεδομένων. Συνήθως σε κάθε δεδομένο αντιστοιχούμε μία μεταβλητή.

Είναι αρκετά σημαντικές όλες οι σκοπιές (σελ. 53-54) με βάση τις οποίες η Πληροφορική μελετά τα δεδομένα (αντιστοιχούν περίπου με αυτές των αλγορίθμων).

3.2. Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

Σημαντικός είναι ο ορισμός της δομής δεδομένων (σελ. 54) και οι βασικές λειτουργίες (πράξεις) επί των δομών (σελ. 54-55). Κάθε δομή είναι ένα **σύνολο δεδομένων** που αποτελείται από **κόμβους** (μέλη του συνόλου).

- Προσπέλαση (access) είναι η πρόσβαση σε κάποιο κόμβο (μέλος της δομής) με σκοπό να εξετασθεί, να εκτυπωθεί ή να μεταβληθεί το περιεχόμενό του. Π.χ. αν σ' ένα πίνακα Α αναφερθώ στον κόμβο Α[3] τότε έχω προσπέλαση στο τρίτο στοιχείο του πίνακα.
- Εισαγωγή (insertion) είναι η προσθήκη νέων κόμβων (μελών) σε μια δομή δεδομένων. Π.χ. αν σ' ένα πίνακα Α δώσω την εντολή Α[10] ← 100 τότε εισάγω ένα νέο κόμβο με τιμή εκατό στο δέκατο στοιχείο του πίνακα (υποθέτω ότι είναι κενό). Αν ο κόμβος υπάρχει ήδη (δεν είναι κενός) τότε μιλάμε για τροποποίηση (update).
- Διαγραφή (deletion) είναι η αφαίρεση (εξαγωγή) ενός κόμβου από μια δομή δεδομένων. Π.χ. αν σε μία στοίβα εξάγω ένα στοιχείο αυτό θα αφαιρεθεί από την κορυφή της.
- Αναζήτηση (search) είναι η προσπέλαση των κόμβων μίας δομής δεδομένων (προφανώς με κάποιο αλγόριθμο) με σκοπό να βρεθούν ένας ή περισσότεροι κόμβοι με συγκεκριμένη ιδιότητα. Μετά την αναζήτηση μπορεί να εφαρμοστούν άλλες πράξεις (δηλ. εισαγωγή, τροποποίηση, διαγραφή, κλπ). Π.χ. είναι δυνατό να αναζητηθεί μία συγκεκριμένη τιμή με σκοπό αν υπάρχει να διαγραφεί.
- Ταξινόμηση (sort) δηλαδή αναδιάταξη των κόμβων μίας δομής κατά αύξουσα ή φθίνουσα σειρά.

Οι πιο συνηθισμένες δομές είναι ο πίνακας (table ή array), η εγγραφή (record), η ουρά (queue), η στοίβα (stack), η λίστα (list), το δένδρο (tree) κλπ. Πρακτικά για να υλοποιήσουμε ένα πρόγραμμα σε μία γλώσσα προγραμματισμού (ή έστω ψευδογλώσσα) χρειαζόμαστε τόσο αλγορίθμους όσο και δομές δεδομένων (δηλ. ισχύει το Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα). Μόνο σε πολύ απλά προγράμματα δε χρειαζόμαστε δομές δεδομένων. Κάποιες δομές δεδομένων (π.χ. πίνακας, εγγραφή) υποστηρίζονται άμεσα (προσφέρονται έτοιμες προς δήλωση) από τις περισσότερες γλώσσες προγραμματισμού γενικής χρήσης (Pascal, Basic, C, κλπ). Άλλες (π.χ. στοίβα, ουρά, λίστα, δένδρο) υλοποιούνται έμμεσα μέσω δηλώσεων των δομών και μεταβλητών που υποστηρίζονται άμεσα. Φυσικά απαιτείται κι η ανάπτυξη κατάλληλων αλγορίθμων για κάθε λειτουργία (πράξη) επί των δομών αυτών.

3.3. Πίνακες

Ο πίνακας είναι μία στατική δομή δεδομένων. Σε μια γλώσσα προγραμματισμού αν π.χ. δηλώσω έναν πίνακα ακεραίων Α με 50 θέσεις τότε αμέσως δεσμεύεται στατικά όλη η απαιτούμενη μνήμη (και για τις 50 θέσεις

ακόμα κι αν αρχικά είναι κενές) και απελευθερώνεται μόνο στο τέλος του προγράμματος (δηλ. δεν δεσμεύεται ούτε απελευθερώνεται μνήμη κατά τη διάρκεια εκτέλεσης του προγράμματος). Ένας πίνακας είναι ένα σύνολο ομοειδών στοιχείων (π.χ. αριθμών, ονομάτων κλπ). Η δομή δεδομένων του πίνακα περιγράφεται πιο αναλυτικά στο κεφάλαιο 9 (βλ. παρ. 9.1-9.2 σελ. 184-191). Υπάρχουν μονοδιάστατοι και πολυδιάστατοι (π.χ. δισδιάστατοι) πίνακες.

3.4. Στοίβα

Μία στοίβα είναι μία LIFO (τελευταίο μέσα, πρώτο έξω) δομή δεδομένων. Αυτό σημαίνει ότι το τελευταίο στοιχείο που έχει εισαχθεί θα είναι το πρώτο που θα εξαχθεί κατόπιν από τη στοίβα. Δύο είναι οι βασικές λειτουργίες σε μία στοίβα η ώθηση (ή εισαγωγή) κι η απώθηση (ή εξαγωγή). Η ώθηση ενός νέου κόμβου στη στοίβα γίνεται στην κορυφή της αν φυσικά η στοίβα δεν είναι γεμάτη. Η απώθηση ενός κόμβου από τη στοίβα γίνεται από την κορυφή της αν φυσικά η στοίβα δεν είναι κενή. Π.χ. αν σε μία κενή στοίβα εισάγω με τη σειρά τέσσερις κόμβους με αντίστοιχες τιμές 5, 7, 9 και 3 και μετά εξάγω δύο κόμβους τότε πρώτα θα εξαχθεί ο κόμβος με τιμή 3 (αφού μπήκε τελευταίος), μετά η τιμή 9 κι η στοίβα θα περιέχει τελικά τους κόμβους με τις τιμές 5 (στη βάση) κι 7 (στην κορυφή). Μία εύκολη (στατική) υλοποίηση μιας στοίβας γίνεται με χρήση ενός μονοδιάστατου πίνακα. Μας χρειάζεται βέβαια και μια βοηθητική μεταβλητή που θα δείχνει πάντα την κορυφή της στοίβας.

3.5. Ουρά

Μία ουρά είναι μία FIFO (πρώτο μέσα, πρώτο έξω) δομή δεδομένων. Αυτό σημαίνει ότι το πρώτο στη σειρά στοιχείο που έχει εισαχθεί θα είναι το πρώτο που θα εξαχθεί κατόπιν από την ουρά. Δύο είναι οι βασικές λειτουργίες σε μία ουρά η εισαγωγή κι η εξαγωγή. Η εισαγωγή ενός νέου κόμβου στην ουρά γίνεται στο πίσω μέρος της αν φυσικά η ουρά δεν είναι γεμάτη. Η εξαγωγή ενός κόμβου από την ουρά γίνεται από το εμπρός μέρος της αν φυσικά δεν είναι κενή. Π.χ. αν σε μία κενή ουρά εισάγω με τη σειρά τέσσερις κόμβους με αντίστοιχες τιμές 5, 7, 9 και 3 και μετά εξάγω δύο κόμβους τότε πρώτα θα εξαχθεί ο κόμβος με τιμή 5 (αφού μπήκε πρώτος), μετά η τιμή 7 κι η ουρά θα περιέχει τελικά τους κόμβους με τις τιμές 9 (μπροστά) και 3 (πίσω). Μία εύκολη (στατική) υλοποίηση μιας ουράς γίνεται με χρήση ενός μονοδιάστατου πίνακα. Μας χρειάζονται βέβαια και δύο βοηθητικές μεταβλητές που θα δείχνουν πάντα το εμπρός και το πίσω άκρο της ουράς αντίστοιχα.

3.6. Αναζήτηση

Υπάρχουν δύο βασικοί αλγόριθμοι αναζήτησης ενός στοιχείου ή κόμβου (με συγκεκριμένο περιεχόμενο) σ' ένα πίνακα. Η **σειριακή** (sequential) ή γραμμική αναζήτηση κι η **δυαδική** (binary) αναζήτηση. Απλούστερη αλλά και πιο χρονοβόρα είναι η σειριακή αναζήτηση (παρ. 3.6 σελ. 64) αφού απαιτεί με τη σειρά προσπέλαση των κόμβων του πίνακα μέχρι να εντοπιστεί ο κόμβος που θέλουμε. Αν το στοιχείο που αναζητούμε δεν υπάρχει τότε πρέπει να προσπελαστούν όλοι οι κόμβοι του πίνακα.

3.7. Ταξινόμηση

Υπάρχουν αρκετοί αλγόριθμοι ταξινόμησης (συνήθως κατά αύξουσα σειρά) των κόμβων ενός πίνακα. Απλούστερη αλλά και πιο χρονοβόρα είναι η **ταξινόμηση φυσαλίδας** (παρ. 3.7 σελ. 68). Η λογική αυτού του αλγορίθμου είναι η εξής: Αρχικά το μικρότερο στοιχείο όλων τοποθετείται στην πρώτη θέση του πίνακα. Στη συνέχεια, το δεύτερο μικρότερο στοιχείο τοποθετείται στην δεύτερη θέση του πίνακα, κοκ μέχρις ότου ταξινομηθεί όλος ο πίνακας.

Για καλύτερη κατανόηση του αλγορίθμου έστω μη ταξινομημένος πίνακας 9 θέσεων (σελ. 65 σχολικού

βιβλίου). Για κάθε κόμβο του πίνακα εκτελείται η παρακάτω επαναληπτική διαδικασία. Ξεκινάμε την επαναληπτική διαδικασία από το **δεύτερο** κόμβο του πίνακα (θα δούμε παρακάτω γιατί). Αρχικά συγκρίνουμε μεταξύ τους τα περιεχόμενα του 9ου και του 8ου κόμβου. Αν το περιεχόμενο του 9ου κόμβου είναι μικρότερο τότε αντιμεταθέτουμε τα περιεχόμενα των δύο κόμβων. Κατόπιν συγκρίνουμε μεταξύ τους τα περιεχόμενα του 8ου και του 7ου κόμβου. Αν το περιεχόμενο του 8ου κόμβου είναι μικρότερο τότε αντιμεταθέτουμε τα περιεχόμενα των δύο κόμβων, κοκ. Είναι προφανές ότι κάθε φορά προωθούνται μπροστά τα μικρότερα στοιχεία. Τελικά συγκρίνουμε μεταξύ τους τα περιεχόμενα του 2ου και του 1ου κόμβου. Αν το περιεχόμενο του 2ου κόμβου είναι μικρότερο τότε αντιμεταθέτουμε τα περιεχόμενα των δύο κόμβων. Με τον τρόπο αυτό το μικρότερο στοιχείο όλων τοποθετείται στην πρώτη θέση του πίνακα. Να γιατί η επαναληπτική διαδικασία ξεκίνησε από το δεύτερο κόμβο! Συνεχίζουμε την επαναληπτική αυτή διαδικασία από τον τρίτο κόμβο του πίνακα. Εξυπακούεται ότι τα περιεχόμενα των κόμβων έχουν αλλάξει με σκοπό την ταξινόμηση όλων των κόμβων. Συγκρίνουμε λοιπόν πάλι μεταξύ τους τα περιεχόμενα του 9ου και του 8ου κόμβου, κοκ. Τελικά συγκρίνουμε μεταξύ τους τα περιεχόμενα του 3ου και του 2ου κόμβου. Προφανώς με τον τρόπο αυτό το δεύτερο μικρότερο στοιχείο όλων τοποθετείται στη δεύτερη θέση του πίνακα, κοκ.

Η αντιμετάθεση δύο μεταβλητών είναι μία λεπτή λειτουργία. Π.χ. έστω δύο μεταβλητές x, y όπου x = 10 και y = 20. Η αντιμετάθεσή τους θα πρέπει να έχει ως αποτέλεσμα x = 20 και y = 10. Θα υπέθετε κανείς ότι με τις απλές εντολές x ← y και μετά y ← x θα μπορούσε να γίνει η αντιμετάθεση. Όμως με την πρώτη εντολή το x θα πάρει την τιμή 10 και με τη δεύτερη εντολή το y θα πάρει κι αυτό την τιμή 10 αφού άλλαξε η τιμή του x! Γι' αυτό μας χρειάζεται να χρησιμοποιήσουμε μια βοηθητική μεταβλητή έστω temp. Με τη σειριακή εκτέλεση των εντολών temp ← x, x ← y, y ← temp η αντιμετάθεση γίνεται σωστά αφού η temp `κρατάει` την αρχική τιμή του x.

Κεφάλαιο 4ο Τεχνικές Σχεδίασης Αλγορίθμων

4.1. Ανάλυση προβλημάτων

Μετά την κατανόηση ενός προβλήματος, στη φάση της ανάλυσης γίνεται ο καθορισμός των απαιτήσεων (βλ. και §1.4 σελ. 11). Πρέπει βέβαια να ληφθεί υπόψη ότι ένα πρόβλημα μπορεί να είναι αρκετά σύνθετο και να επιδέχεται περισσότερες από μία λύσεις ή προσεγγίσεις. Πιο συγκεκριμένα κατά το στάδιο της ανάλυσης:

- Καταγράφονται τα δεδομένα και τα ζητούμενα (αποτελέσματα) του προβλήματος
- Αποτυπώνονται οι ιδιαιτερότητες του προβλήματος (αν υπάρχουν)
- Καταγράφονται οι συνθήκες και προϋποθέσεις που πρέπει να πληρούνται (π.χ. σε ένα εμπορικό πρόγραμμα πρέπει να λαμβάνεται υπόψη ότι στις παραμεθόριες περιοχές ο ΦΠΑ είναι 13% ενώ στην υπόλοιπη χώρα είναι 18%)
- Επιλέγεται κάποια κατάλληλη μέθοδος σχεδίασης (μεθοδολογία) της λύσης (π.χ. 'διαίρει και βασίλευε' - top/down)
- Αποτυπώνεται ο σχεδιασμός (π.χ. με χρήση μίας **δομημένης** ψευδογλώσσας ή διαγραμμάτων ροής)
- Επιλέγεται κάποια κατάλληλη γλώσσα προγραμματισμού **υλοποίησης** της λύσης (π.χ. η γλώσσα C)

Αφού γίνει η ανάλυση του προβλήματος, ακολουθεί η επίλυσή του, παράγεται δηλαδή ο **κώδικάς** του (π.χ. εντολές γλώσσας C) ο οποίος όταν εκτελεστεί πρέπει να δίνει τα επιθυμητά αποτελέσματα. Σημειώνεται ότι τα στάδια της κατανόησης, ανάλυσης και επίλυσης δεν είναι εντελώς ανεξάρτητα ούτε αποκομμένα το ένα από το άλλο. Δηλαδή, αν προχωρώντας στην ανάλυση (σχεδιασμό) υπάρξει ανάγκη, είναι δυνατόν να συμπληρώσουμε στοιχεία του σταδίου της κατανόησης. Επίσης, αν προχωρώντας στην επίλυση (κωδικοποίηση) υπάρξει ανάγκη, είναι δυνατόν να επιστρέψουμε πίσω στη φάση της ανάλυσης, κ.ο.κ.

4.2. Μέθοδοι σχεδίασης αλγορίθμων

Πρόκειται για θεωρητικές προσεγγίσεις που χρησιμοποιούνται κατά τη διάρκεια της φάσης της ανάλυσης. Υπάρχουν πολλές τεχνικές σχεδίασης αλγορίθμων. Οι πιο βασικές είναι:

- 'Διαίρει και βασίλευε' ή προσέγγιση από επάνω προς τα κάτω - top/ down. Χρησιμοποιείται ευρύτατα και βασίζεται στη σταδιακή επίλυση ή διάσπαση ενός προβλήματος σε υποπροβλήματα. Ξεκινάμε δηλαδή από το αρχικό πρόβλημα, το διαιρούμε σε μια σειρά από γενικά υποπροβλήματα τα οποία στη συνέχεια διαιρούμε εκ νέου σε ειδικότερα (απλούστερα) υποπροβλήματα, κ.ο.κ. Η φιλοσοφία αυτή αποτυπώνεται μέσω της **ιεραρχικής σχεδίασης** (βλ. και §6.4.1 σελ. 132).
- Δυναμικός προγραμματισμός ή προσέγγιση από κάτω προς τα επάνω - bottom/ up. Χρησιμοποιείται κάπως πιο σπάνια, κυρίως για την επίλυση προβλημάτων βελτιστοποίησης. Βασίζεται στην αντίστροφη φιλοσοφία της τεχνικής 'διαίρει και βασίλευε'. Δηλαδή από την αρχή επιλύονται τα μικρότερα (επιμέρους) προβλήματα και σιγά-σιγά οι επιμέρους λύσεις καταλήγουν στη σύνθεση της λύσης του αρχικού προβλήματος.
- Άπληστη μέθοδος. Χρησιμοποιείται ακόμα πιο σπάνια, επίσης για την επίλυση προβλημάτων βελτιστοποίησης, όταν δε μας ενδιαφέρει η απολύτως καλύτερη (βέλτιστη) λύση αλλά αρκεί μία προσέγγισή της (παράδειγμα ταχυδρομικού διανομέα - Α λύση - σελ. 82).

Κεφάλαιο 6ο Εισαγωγή στον Προγραμματισμό

6.1 Η έννοια του προγράμματος

Ο προγραμματισμός είναι το τελευταίο στάδιο της επίλυσης. Τα προγράμματα γράφονται τελικά σε κάποια γλώσσα προγραμματισμού.

- Πρόγραμμα = Διατύπωση αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή (δηλ. εντέλει σε ***δυαδικό σύστημα***)
- Ο άνθρωπος επικοινωνεί με τον Η/Υ μόνο μέσω προγραμμάτων
- **Πρόγραμμα** = Δ***εδομένα*** + ***Αλγόριθμοι***
- Το στάδιο του προγραμματισμού ακολουθεί τα στάδια της κατανόησης (προσδιορισμού του προβλήματος) και της ανάλυσης
- Ο προγραμματισμός δίνει την εσφαλμένη εντύπωση ότι οι υπολογιστές είναι 'έξυπνες' μηχανές. Στην πραγματικότητα απλά εκτελούν στοιχειώδεις ενέργειες (πράξεις, συγκρίσεις, μεταφορές δεδομένων).

6.2 Ιστορική αναδρομή (γλώσσες προγραμματισμού)

Μία πρώτη ταξινόμηση των γλωσσών προγραμματισμού είναι ανάλογα με το κριτήριο της γενιάς στην οποία ανήκουν (επίπεδο γλώσσας). Έτσι έχουμε:

- Γλώσσες 1ης γενιάς ή πολύ χαμηλού επιπέδου που είναι οι γλώσσες μηχανής (machine languages).
- Γλώσσες 2ης γενιάς ή χαμηλού επιπέδου που είναι οι συμβολικές γλώσσες (assembly).
- Γλώσσες 3ης γενιάς ή υψηλού επιπέδου που είναι οι γλώσσες Fortran, Cobol, Basic, Algol, Pascal, PL/1, C, C++, Java, κλπ.
- Γλώσσες 4ης γενιάς ή πολύ υψηλού επιπέδου που είναι ορισμένες γλώσσες ερωταπαντήσεων, γλώσσες εφαρμογών βάσεων δεδομένων (π.χ. SQL) κλπ. Στις γλώσσες 4ης γενιάς ανήκει και η PROLOG όπως κι άλλες γλώσσες.

Η γλώσσα μηχανής είναι άμεσα αντιληπτή από τον υπολογιστή και απ' ευθείας εκτελέσιμη. Αποτελείται από μία ακολουθία δυαδικών ψηφίων (δηλ. 0 και 1). Προφανώς είναι πολύ δύσχρηστη και επίπονη στη χρήση γι' αυτό χρησιμοποιούνταν στους πρώτους υπολογιστές πριν ακόμα αναπτυχθούν γλώσσες υψηλότερου επιπέδου. Η γλώσσα μηχανής είναι άμεσα **εξαρτημένη** από την αρχιτεκτονική (υλικό ή μηχανή) του υπολογιστή στον οποίο εκτελείται. Γράφοντας προγράμματα σε γλώσσα μηχανής 'μιλάμε' απ' ευθείας με το υλικό του συγκεκριμένου υπολογιστή (π.χ. με τη μνήμη). Δύο διαφορετικοί υπολογιστές (δηλ. με διαφορετικό επεξεργαστή) δεν έχουν την ίδια γλώσσα μηχανής. Επομένως τα προγράμματα που έχουν γραφεί σε γλώσσα μηχανής δεν είναι καθόλου εύκολα μεταφέρσιμα σε άλλους υπολογιστές. Στο παρακάτω παράδειγμα θεωρούμε ότι στο συγκεκριμένο Η/Υ ο κωδικός **00110010** σημαίνει φόρτωση (αποθήκευση) στη μνήμη, ενώ ο κωδικός **01011100** σημαίνει πρόσθεση (εικονικό παράδειγμα):

00110010 00000011 10000001 → Βάλε τον αριθμό 3, σε μία συγκεκριμένη θέση μνήμης (π.χ. θέση μνήμης 1 έστω με κωδικό 10000001)

00110010 00000100 10000010 → Βάλε τον αριθμό 4, σε μία συγκεκριμένη θέση μνήμης (π.χ. θέση μνήμης 2 έστω με κωδικό 10000010)

01011100 00000011 10000001 → Πρόσθεσε τις δύο θέσεις μνήμης (ουσιαστικά τα περιεχόμενά τους)

Οι συμβολικές γλώσσες είναι μία εξέλιξη των γλωσσών μηχανής. Χρησιμοποιούν συντομογραφίες και σύμβολα (της αγγλικής γλώσσας) που είναι πιο κατανοητές στον άνθρωπο. Έχουν τη δυνατότητα να μετατρέπουν εσωτερικά τις εντολές στην ισοδύναμη ακολουθία δυαδικών ψηφίων (δηλ. 0 και 1) αφού διαθέτουν **συμβολομεταφραστή** (assembler). Μία εντολή συμβολικής γλώσσας αντιστοιχεί σε μία αρκετά μεγάλη ακολουθία δυαδικών αριθμών. Επομένως οι συμβολικές γλώσσες είναι λιγότερο δύσχρηστες και επίπονες στη χρήση, αφού ο κώδικας είναι πιο 'συμμαζεμένος'. Η συμβολική γλώσσα παραμένει άμεσα **εξαρτημένη** από την αρχιτεκτονική (υλικό ή μηχανή) του υπολογιστή στον οποίο εκτελείται. Γράφοντας προγράμματα σε συμβολική γλώσσα πάλι 'μιλάμε' απ' ευθείας με το υλικό του συγκεκριμένου υπολογιστή (π.χ. με τη μνήμη). Δύο διαφορετικοί υπολογιστές (δηλ. με διαφορετικό επεξεργαστή) δεν έχουν κατ' ανάγκη την ίδια συμβολική γλώσσα. Επομένως τα προγράμματα που έχουν γραφεί σε συμβολική γλώσσα δεν είναι καθόλου εύκολα μεταφέρσιμα σε άλλους υπολογιστές. Π.χ.

LDA 3 $01 → Βάλε τον αριθμό 3, σε μία συγκεκριμένη θέση μνήμης (π.χ. θέση μνήμης 1)

LDA 4 $02 → Βάλε τον αριθμό 4, σε μία συγκεκριμένη θέση μνήμης (π.χ. θέση μνήμης 2)

ADD $01 $02 → Πρόσθεσε τις δύο θέσεις μνήμης (ουσιαστικά τα περιεχόμενά τους)

Οι γλώσσες υψηλού επιπέδου είναι πολύ πιο απλές και κατανοητές στον άνθρωπο. Έχουν επίσης τη δυνατότητα να μετατρέπουν εσωτερικά τις εντολές στην ισοδύναμη ακολουθία δυαδικών ψηφίων (δηλ. 0 και 1) αφού διαθέτουν είτε **μεταγλωττιστή** (compiler) είτε **διερμηνευτή** (interpreter). Μία εντολή γλώσσας υψηλού επιπέδου αντιστοιχεί σε μία ή περισσότερες εντολές συμβολικής γλώσσας και εντέλει σε μία πολύ μεγάλη ακολουθία δυαδικών αριθμών. Επομένως οι γλώσσες υψηλού επιπέδου είναι πολύ εύχρηστες και καθόλου επίπονες στη χρήση, αφού ο κώδικας είναι ακόμα πιο 'συμμαζεμένος'. Οι γλώσσες υψηλού επιπέδου είναι **ανεξάρτητες** από την

αρχιτεκτονική (υλικό) του υπολογιστή στον οποίο εκτελούνται. Για να γράψουμε προγράμματα σε μία γλώσσα υψηλού επιπέδου δε χρειάζεται πια να 'μιλάμε' απ' ευθείας με το υλικό του συγκεκριμένου υπολογιστή. Δύο διαφορετικοί υπολογιστές (δηλ. με διαφορετικό επεξεργαστή) έχουν ακριβώς την ίδια γλώσσα υψηλού επιπέδου (π.χ. τις ίδιες ακριβώς εντολές Pascal). Αρκεί βέβαια να διαθέτουν τον ίδιο μεταγλωττιστή της Pascal. Επομένως τα προγράμματα που έχουν γραφεί σε μία γλώσσα υψηλού επιπέδου είναι εύκολα μεταφέρσιμα σε άλλους υπολογιστές. Π.χ. (σε Pascal):

X := 3 → Βάλε τον αριθμό 3 σε μία μεταβλητή X

Y := 4 → Βάλε τον αριθμό 4 σε μία μεταβλητή Y

Y := X + Y → Πρόσθεσε τις δύο μεταβλητές

Οι γλώσσες πολύ υψηλού επιπέδου ή 4ης γενιάς είναι ακόμα περισσότερο απλές και κατανοητές στον άνθρωπο, γι' αυτό δεν απευθύνονται αποκλειστικά και μόνο σε προγραμματιστές αλλά ακόμα και σε απλούς χειριστές (χρήστες) υπολογιστών. Μία εντολή γλώσσας 4ης γενιάς γενικά αντιστοιχεί σε πολλές εντολές γλώσσας 3ης γενιάς (υψηλού επιπέδου) και εντέλει σε μία πάρα πολύ μεγάλη ακολουθία δυαδικών αριθμών. Επομένως οι γλώσσες 4ης γενιάς είναι πάρα πολύ εύχρηστες και καθόλου επίπονες στη χρήση, αφού ο κώδικας είναι ακόμα πιο 'συμμαζεμένος'. Οι γλώσσες 4ης γενιάς είναι ακόμα πιο **ανεξάρτητες** από την αρχιτεκτονική (υλικό ή μηχανή) του υπολογιστή στον οποίο εκτελούνται.

Μία δεύτερη ταξινόμηση των γλωσσών προγραμματισμού είναι ανάλογα με το κριτήριο του τρόπου προγραμματισμού (ιδέα-φιλοσοφία στην οποία βασίζονται). Έτσι έχουμε:

- Γλώσσες **αλγοριθμικές** ή **διαδικασιακές** (π.χ. Fortran, Cobol, Pascal, Basic, C, κλπ). Πρόκειται για τις πιο παλαιές και 'κλασσικές' γλώσσες. Ξεκινάμε τον προγραμματισμό από τις λειτουργίες (ή διαδικασίες) που έχουμε να υλοποιήσουμε. Αφού τις αναλύσουμε, για κάθε διαδικασία φτιάχνεται κι ένας αλγόριθμος. Τα δεδομένα είναι κατά κάποιο τρόπο πιο δευτερεύοντα αφού ορίζονται και χρησιμοποιούνται μέσα στις διαδικασίες.
- Γλώσσες **αντικειμενοστραφείς** (π.χ. C++, Smalltalk, Java, κλπ). Υπάρχουν γλώσσες καθαρά αντικειμενοστραφείς (π.χ. Smalltalk, Java, κλπ) κι άλλες (π.χ. C++) που προέκυψαν από μία αλγοριθμική γλώσσα ως επέκτασή της ώστε να υποστηρίζεται η αντικειμενοστραφής φιλοσοφία. Εδώ ξεκινάμε τον προγραμματισμό από τα δεδομένα που έχουμε στο συγκεκριμένο πρόβλημα. Αφού αναλυθούν τα δεδομένα, μετά την κατάλληλη μορφοποίηση δομούνται ως αντικείμενα. Οι διαδικασίες είναι κατά κάποιο τρόπο πιο δευτερεύουσες αφού ορίζονται και χρησιμοποιούνται μέσα στα αντικείμενα (λειτουργίες μεταξύ αντικειμένων).
- Γλώσσες **συναρτησιακές** (π.χ. Lisp). Η Lisp, που ανήκει στο χώρο της τεχνητής νοημοσύνης, βασίζεται στη δομή δεδομένων της λίστας. Όλες οι λειτουργίες στη γλώσσα αυτή γίνονται μέσω συναρτήσεων μεταξύ λιστών.
- Γλώσσες **μη διαδικασιακές** ή **λογικές** (π.χ. Prolog). Συνήθως αυτές είναι γλώσσες πάρα πολύ υψηλού επιπέδου. Πολλές από αυτές ανήκουν στο χώρο της τεχνητής νοημοσύνης. Οι φιλοσοφία τους είναι αντίθετη με τον διαδικασιακό (αλγοριθμικό) προγραμματισμό. Προσπαθούν να προσομοιώσουν την ανθρώπινη λογική στους υπολογιστές. Για παράδειγμα ένα έμπειρο σύστημα (όπως ένα σύστημα πρόγνωσης καιρού) μπορεί να 'μαθαίνει' με την πάροδο του χρόνου όπως ο άνθρωπος.
- Γλώσσες **ερωταπαντήσεων** (π.χ. SQL). Πολλές από αυτές ανήκουν στο χώρο των βάσεων δεδομένων. Ο χρήστης μπορεί να υποβάλλει ερωτήματα στον υπολογιστή και να παίρνει τις αναγκαίες πληροφορίες με την επιθυμητή μορφή.
- Γλώσσες **οπτικές** (visual) ή **οδηγούμενες από το γεγονός** (event-driven) Π.χ. Visual Basic, Visual

C, Visual C++, κλπ. Οι κλασσικές διαδικασιακές γλώσσες (λόγω και της παλαιότητάς τους) δεν εκμεταλλεύονται τα νέα γραφικά περιβάλλοντα επικοινωνίας μεταξύ ανθρώπου-υπολογιστή (π.χ. Windows). Οι οπτικές γλώσσες διαθέτουν ένα μεγάλο πλούτο από λειτουργίες εισόδου/ εξόδου (δεδομένων/ πληροφοριών) και μας επιτρέπουν να δημιουργούμε όλο το περιβάλλον ενός προγράμματος με εύχρηστο και γραφικό τρόπο (πλαίσια διαλόγου, μενού, κουμπάκια, παράθυρα, κλπ). Σε ένα τέτοιο περιβάλλον γράφουμε μικρά κομμάτια εντολών για κάθε γεγονός. Π.χ. ένα γεγονός μπορεί να είναι το πάτημα ενός κουμπιού.

- Γ λώσσες **παράλληλες** (π.χ. Occam, Parallel C)**.** Εκμεταλλεύονται αρχιτεκτονικές με περισσότερους από έναν επεξεργαστές που λειτουργούν παράλληλα. Η απόδοση (ταχύτητα) στα συστήματα αυτά έτσι αυξάνει κατακόρυφα.

Μία τρίτη ταξινόμηση των γλωσσών προγραμματισμού είναι ανάλογα με το κριτήριο της περιοχής χρήσης (σκοποί, χρήσεις στις οποίες εφαρμόζονται). Έτσι έχουμε:

- Γλώσσες **γενικής χρήσης** (π.χ. Basic, Pascal). Είναι αρκετά καλές κι αποτελεσματικές για μία ευρεία γκάμα εφαρμογών. Βέβαια εδώ μπορούμε να εντάξουμε (θεωρητικά) τόσο τις γλώσσες **επιστημονικής κατεύθυνσης** (π.χ. Fortran) όσο και τις γλώσσες **εμπορικής κατεύθυνσης** (π.χ. Cobol).
- Γλώσσες **προγραμματισμού συστημάτων** (π.χ. C). Πολλές συσκευές (π.χ. video) ή τεχνολογικά συστήματα (καρτοτηλέφωνα, μηχανήματα αυτόματων τραπεζικών συναλλαγών, μηχανήματα ΠΡΟΠΟ-ΛΟΤΤΟ, κλπ) εκτελούν προγράμματα όπως περίπου ο υπολογιστής. Επομένως χρειάζονται κάποια γλώσσα προγραμματισμού που να διαθέτει και αρετές χαμηλού επιπέδου για να εκμεταλλεύεται το συγκεκριμένο υλικό (μηχάνημα).
- Γλώσσες **τεχνητής νοημοσύνης** (π.χ. Prolog, Lisp). Μία αδυναμία των παραδοσιακών γλωσσών είναι ότι διαφέρουν κατά πολύ από την ανθρώπινη λογική. Οι γλώσσες τεχνητής νοημοσύνης προσπαθούν κατά κάποιο τεχνητό τρόπο να προσομοιώσουν την ανθρώπινη νοημοσύνη. Χρησιμοποιούνται σε παιχνίδια (π.χ. σκάκι), έμπειρα συστήματα, κλπ.
- Γλώσσες **ειδικής χρήσης**. Είναι γλώσσες που χρησιμοποιούνται για πολύ εξειδικευμένες εφαρμογές (π.χ. σχεδίαση-γραφικά, εκπαίδευση, κλπ).

Στον παρακάτω πίνακα φαίνονται οι πιο γνωστές γλώσσες προγραμματισμού με τα πιο βασικά χαρακτηριστικά τους. Η παράθεσή τους είναι περίπου χρονολογική.

ΟΝΟΜΑ ΓΛΩΣΣΑΣ	**ΓΕΝΙΑ Ή ΕΠΙΠΕΔΟ**	**ΤΡΟΠΟΣ ΠΡΟΓΡ/ΣΜΟΥ**	**ΧΡΗΣΗ**	**ΠΑΡΑΤΗΡΗΣΕΙΣ**
Μηχανής	1η, Πολύ χαμηλό			Χρησιμοποιήθηκε στους πρώτους Η/Υ
Συμβολική	2η, Χαμηλό		Μη δομημένη (επιβάλλεται η χρήση GOTO)	
Fortran	3η,Υψηλό	Αλγοριθμικός	Επιστημονική	Πρώτη γλώσσα υψηλού επιπέδου
Cobol	3η,Υψηλό	Αλγοριθμικός	Εμπορική	Χρησιμοποιείται σχεδόν σε κάθε κατηγορία υπολογιστή
Algol	3η,Υψηλό	Αλγοριθμικός	Γενική	Απέτυχε εμπορικά αλλά επηρέασε πολύ άλλες γλώσσες
Lisp	3η - 4η, Πολύ Υψηλό	Συναρτησιακός	Τεχνητή νοημοσύνη	Χρησιμοποιεί τη δομή της λίστας
Prolog	4η, Πολύ Υψηλό	Λογικός (Μη διαδικαφιακός)	Τεχνητή νοημοσύνη (έμπειρα	

Logo	3η,Υψηλό	Μη διαδικασιακός ή οπτικός		Ειδικής χρήσης Εκπαιδευτική Υπάρχει και οπτική υλοποίησή της
PL/1	3η,Υψηλό	Αλγοριθμικός	Γενική	Απέτυχε εμπορικά
Basic	3η,Υψηλό	Αλγοριθμικός	Γενική	Απλή και για αρχαρίους
Pascal	3η,Υψηλό	Αλγοριθμικός	Γενική	Καθιέρωσε το δομημένο προγραμματισμό

ΟΝΟΜΑ ΓΛΩΣΣΑΣ	ΓΕΝΙΑ Ή ΕΠΙΠΕΔΟ	ΤΡΟΠΟΣ ΠΡΟΓΡ/ΣΜΟΥ	ΧΡΗΣΗ	ΠΑΡΑΤΗΡΗΣΕΙΣ
C	3η,Υψηλό	Αλγοριθμικός	Γενική, Επιστημονική, Προγραμματισμός συστημάτων	Πολύ ισχυρή, με δυνατότητες χαμηλού επιπέδου
C++	3η,Υψηλό	Αντικειμενο-στραφής	Γενική, Επιστημονική, Προγραμματισμός συστημάτων	Πολύ ισχυρή, με δυνατότητες χαμηλού επιπέδου Επέκταση της C
SQL	4η, Πολύ Υψηλό		Ερωταπαντήσεων	Εμπορική Βάσεις Δεδομένων
dBASE, Clipper, Access	4η, Πολύ Υψηλό		Ερωταπαντήσεων	Εμπορική Βάσεις Δεδομένων
Occam	3η,Υψηλό	Παράλληλος	Ειδική	
Parallel C	3η,Υψηλό	Παράλληλος	Ειδική	Επέκταση της C
Visual C	3η,Υψηλό	Οπτικός	Γενική, Επιστημονική, Προγραμματισμός συστημάτων	Επέκταση της C
Visual C++	3η,Υψηλό	Οπτικός και Αντικειμενο-στραφής	Γενική, Επιστημονική, Προγραμματισμός συστημάτων	Επέκταση της C
Visual Basic	3η,Υψηλό	Οπτικός	Γενική	Επέκταση της Basic
Java	3η,Υψηλό	Αντικειμενο-στραφής	Διαδίκτυο	

Από τα παραπάνω, είναι προφανές ότι δεν υπάρχει μία τέλεια γλώσσα προγραμματισμού για όλες τις εφαρμογές. Επομένως πρέπει πάντα να επιλέγεται η πιο κατάλληλη γλώσσα κατά περίπτωση.

6.3 Φυσικές και τεχνητές γλώσσες

Οι γλώσσες προγραμματισμού είναι τεχνητές γλώσσες. Ωστόσο **όλες** οι γλώσσες προσδιορίζονται από:

- Το αλφάβητο. Π.χ. η γλώσσα προγραμματισμού Pascal περιέχει μόνο τους λατινικούς χαρακτήρες (κεφαλαία και μικρά), τα 10 ψηφία και ορισμένα μόνο σημεία στίξης (π.χ. το ;)
- Το λεξιλόγιο δηλ. το σύνολο των αποδεκτών λέξεων της γλώσσας. Συνήθως σε μία γλώσσα προγραμματισμού, το λεξιλόγιο αποτελείται από 20-100 λέξεις μόνο (αποδεκτές λέξεις-εντολές). Προφανώς οι φυσικές γλώσσες (π.χ. αγγλικά, ελληνικά) είναι πολύ πιο πλούσιες σε λεξιλόγιο.
- Τη γραμματική που αποτελείται από το τυπικό μέρος και το συντακτικό. Π.χ. στην Pascal η λέξη **four** δεν είναι τυπικά αποδεκτή ενώ η λέξη **for** είναι τυπικά αποδεκτή (εντολή επανάληψης για). Επίσης στην Pascal η εντολή **for i:=1 until 10 do** δεν είναι συντακτικά σωστή παρόλο που οι λέξεις for, until, do είναι τυπικά σωστές. Η σωστή σύνταξη της πρότασης είναι **for i:=1 to 10 do**.
- Τη σημασιολογία δηλ. το νόημα. Π.χ. στην Pascal η εντολή y:=x+1 δεν έχει νόημα αν δεν έχει οριστεί το x (απροσδιοριστία).

6.4 Τεχνικές σχεδίασης προγραμμάτων

Πρόκειται για προσεγγίσεις που χρησιμοποιούνται στο τέλος της φάσης της ανάλυσης για να αποτυπωθεί ο σχεδιασμός της επίλυσης. Οι πιο βασικές τεχνικές είναι:

- Ιεραρχική σχεδίαση προγράμματος. Προκύπτει συνήθως μετά τη φάση της 'Διαίρει και βασίλευε' ανάλυσης ή από επάνω προς τα κάτω - top/ down.
- Τμηματικός προγραμματισμός. Υλοποιεί την ιεραρχική σχεδίαση σε μία γλώσσα προγραμματισμού

(π.χ. Pascal). Κάθε υποπρόβλημα αποτελεί ανεξάρτητη ενότητα (υποαλγόριθμο ή υποδιαδικασία). Εκτενέστερη αναφορά στον τμηματικό προγραμματισμό γίνεται και στα επόμενα. (βλ. §10.1)

- Δομημένος προγραμματισμός. Βασίζεται στη χρήση των δομών ακολουθίας, επιλογής, επανάληψης και μόνο. Αποθαρρύνει την ανεξέλεγκτη χρήση των εντολών GOTO.

Πολύ σημαντικά είναι και τα πλεονεκτήματα του δομημένου προγραμματισμού (βλ. §6.4 σελ. 136):

6.5 Αντικειμενοστραφής προγραμματισμός

Η αντικειμενοστραφής φιλοσοφία έχει ήδη περιγραφεί στα προηγούμενα (βλ. §6.2).

6.6 Παράλληλος προγραμματισμός

Η παράλληλη φιλοσοφία έχει ήδη περιγραφεί στα προηγούμενα (βλ. §6.2).

6.7 Προγραμματιστικά περιβάλλοντα

Σε κάθε προγραμματιστικό περιβάλλον υπάρχουν τα κατάλληλα **εργαλεία** που βοηθούν τον προγραμματιστή στην εργασία του. Η εργασία αυτή αρχίζει με τη σύνταξη (γράψιμο) του προγράμματος (κώδικα) και ολοκληρώνεται με την εκτέλεσή του. Πιο συγκεκριμένα σε ένα παραδοσιακό προγραμματιστικό περιβάλλον ισχύουν τα ακόλουθα:

Σχεδόν όλες οι γλώσσες προγραμματισμού διαθέτουν ένα ειδικό εργαλείο (πρόγραμμα) που ονομάζεται **συντάκτης-διορθωτής**. Ο συντάκτης είναι απλά ένας κειμενογράφος που επιτρέπει την αρχική σύνταξη (γράψιμο) του προγράμματος και τη διόρθωσή του στη συνέχεια (π.χ. σβήσιμο χαρακτήρων-εντολών). Το σύνολο των εντολών της γλώσσας προγραμματισμού (δηλ. ο αρχικός κώδικας) που γράφουμε ονομάζεται **πηγαίο πρόγραμμα** (ή πηγαίος κώδικας).

Είναι προφανές ότι το πηγαίο πρόγραμμα δε μπορεί να εκτελεστεί, αφού είναι απαραίτητη η ‘μετάφρασή’ του σε δυαδικό σύστημα. Μόνο στη περίπτωση που το πηγαίο πρόγραμμα έχει γραφεί σε γλώσσα μηχανής, το πηγαίο πρόγραμμα είναι κι εκτελέσιμο. Για να γίνει αυτή η ‘μετάφραση’, στις γλώσσες υψηλού επιπέδου, ακολουθούνται δύο διαφορετικές τεχνικές. Του **μεταγλωττιστή** (compiler) και του **διερμηνέα** (interpreter).

Πολλές παραδοσιακές γλώσσες χρησιμοποιούν μεταγλωττιστή (π.χ. Pascal). Αρχικά από όλο το πηγαίο (source) πρόγραμμα παράγεται μετά τη μεταγλώττιση ένα ενδιάμεσο πρόγραμμα, που ονομάζεται **αντικείμενο** ή αντικειμενικό (object) πρόγραμμα. Τονίζεται ότι το αντικείμενο πρόγραμμα δε σχετίζεται με την έννοια του αντικειμενοστραφούς προγραμματισμού. Εξυπακούεται ότι για να δημιουργηθεί το αντικείμενο πρόγραμμα η μεταγλώττιση πρέπει να είναι επιτυχής (δηλ. δεν πρέπει να υπάρχουν λάθη τυπικά ή συντακτικά στον πηγαίο κώδικα). Αν υπάρχουν λάθη, κατάλληλα μηνύματα μας βοηθούν να τα διορθώσουμε. Το αντικείμενο πρόγραμμα είναι κατανοητό από τον υπολογιστή (σε μορφή δυαδικού συστήματος) αλλά δεν είναι δυνατό να εκτελεστεί. Είναι απαραίτητο να συνδεθεί με άλλα τμήματα (π.χ. βιβλιοθήκες γλώσσας ή προγραμματιστή). Αυτή τη λειτουργία την αναλαμβάνει ο **συνδέτης-φορτωτής** (linker-loader) που παράγει και το τελικό **εκτελέσιμο** (executable) πρόγραμμα (κώδικα). Επομένως σε ένα σύστημα με μεταγλωττιστή, η διαδικασία είναι σύνταξη → μεταγλώττιση → σύνδεση → εκτέλεση. Τα απαραίτητα εργαλεία είναι ο συντάκτης, ο μεταγλωττιστής κι ο συνδέτης και τα αρχεία που δημιουργούνται είναι το πηγαίο, το αντικείμενο και το εκτελέσιμο. Εξυπακούεται ότι αν γίνουν αλλαγές ή διορθώσεις στον πηγαίο κώδικα, η διαδικασίες της μεταγλώττισης και σύνδεσης πρέπει να επαναληφθούν μέχρι να παραχθεί το τελικό εκτελέσιμο πρόγραμμα. Από τη στιγμή που θα παραχθεί το τελικό εκτελέσιμο πρόγραμμα δε χρειαζόμαστε

κανένα εργαλείο (ούτε συντάκτη, ούτε μεταγλωττιστή ούτε συνδέτη) για να το χρησιμοποιήσουμε (ή να το 'τρέξουμε'). Μπορούμε ακόμα και να μεταφέρουμε το τελικό αυτό εκτελέσιμο αρχείο σε άλλον υπολογιστή (συμβατό) και να το εκτελέσουμε δίχως ο δεύτερος αυτός υπολογιστής να διαθέτει καν τη γλώσσα προγραμματισμού και τα εργαλεία της.

Ορισμένες άλλες γλώσσες χρησιμοποιούν διερμηνευτή ή διερμηνέα (π.χ. Basic). Αρχικά μετά από κάθε εντολή (γραμμή) του πηγαίου (source) κώδικα γίνεται η 'μετάφραση' (διερμηνεία ή διερμήνευση). Αν υπάρχουν λάθη στην εντολή, κατάλληλα μηνύματα μας βοηθούν να τα διορθώσουμε. Η βασική διαφορά του διερμηνευτή (interpreter) με το μεταγλωττιστή (compiler) είναι ότι η διερμηνεία γίνεται εντολή-εντολή (γραμμή-γραμμή), ενώ η μεταγλώττιση γίνεται για όλο το πρόγραμμα. Σε συστήματα με διερμηνευτή δε δημιουργείται αντικείμενο πρόγραμμα. Ακόμη δεν υπάρχει συνδέτης-φορτωτής και μετά από τη διερμηνεία το πρόγραμμα μπορεί να εκτελεστεί άμεσα. Επομένως σε ένα σύστημα με διερμηνευτή, η διαδικασία είναι σύνταξη → διερμηνεία → εκτέλεση. Τα απαραίτητα εργαλεία είναι ο συντάκτης κι ο διερμηνευτής και το μόνο αρχείο που δημιουργείται είναι το πηγαίο. Δηλαδή δεν παράγεται εκτελέσιμο αρχείο που σημαίνει ότι ακόμα κι αν δεν έχουν γίνει αλλαγές ή διορθώσεις στον πηγαίο κώδικα, η διαδικασία της διερμηνείας επιβάλλεται πάντα πριν να εκτελεστεί το πρόγραμμα.

Η τεχνική του μεταγλωττιστή έχει το μειονέκτημα των διαδικασιών μεταγλώττισης και σύνδεσης (χρονική καθυστέρηση). Όμως το παραγόμενο εκτελέσιμο πρόγραμμα είναι ταχύτερο και μάλιστα μεταφέρσιμο. Τα συστήματα με διερμηνευτή είναι πολύ αποτελεσματικά για σύντομα προγράμματα ειδικότερα όταν ο ολικός φόρτος του συστήματος είναι μικρός. Διαφορετικά, τα συστήματα με μεταγλωττιστή πλεονεκτούν. Σήμερα σε πολλά προγραμματιστικά περιβάλλοντα υπάρχουν και μεικτές υλοποιήσεις (π.χ. διερμηνευτής που παράγει και εκτελέσιμο αρχείο). Επίσης, σε ένα περιβάλλον οπτικού προγραμματισμού μπορεί να έχουμε μικροδιαφορές (π.χ. ειδικούς συντάκτες).

Κεφάλαιο 7ο Βασικές Έννοιες Προγραμματισμού

7.1 Το αλβάβητο της ΓΛΩΣΣΑΣ

Στις επόμενες παραγράφους θα χρησιμοποιηθεί ως γλώσσα προγραμματισμού η **ΓΛΩΣΣΑ**. Φυσικά δε πρόκειται για μία υπαρκτή γλώσσα προγραμματισμού. Η ΓΛΩΣΣΑ είναι καθαρά εκπαιδευτική και ελάχιστα διαφέρει από την ψευδογλώσσα που χρησιμοποιήθηκε στις προηγούμενες παραγράφους. Έχοντας μελετήσει την έννοια των φυσικών και τεχνητών γλωσσών (βλ. §6.3) και τα προγραμματιστικά περιβάλλοντα (βλ. §6.7), δεν είναι δύσκολο να κατανοηθεί η ΓΛΩΣΣΑ που είναι πιο 'αυστηρή' από την ψευδογλώσσα ειδικά στη δήλωση των μεταβλητών.

Στο αλφάβητο της ΓΛΩΣΣΑΣ περιλαμβάνονται όλα τα γράμματα (πεζά, κεφαλαία, ελληνικά, λατινικά), τα ψηφία (0-9) κι ορισμένοι ειδικοί χαρακτήρες (+ - * / = κλπ).

7.2 Τύποι δεδομένων

Η ΓΛΩΣΣΑ υποστηρίζει τους βασικούς τύπους δεδομένων δηλαδή ακεραίους, πραγματικούς, χαρακτήρες (κείμενο-αλφαριθμητικά) και λογικούς (Αληθής, Ψευδής).

7.3 Σταθερές

Η ΓΛΩΣΣΑ υποστηρίζει την αντιστοίχιση σταθερών τιμών με ονόματα. Βέβαια, τέτοιες σταθερές πρέπει

να δηλωθούν στην αρχή του προγράμματος. Π.χ.

ΣΤΑΘΕΡΕΣ

ΠΙ = 3.14

7.4 Μεταβλητές

Η ΓΛΩΣΣΑ υποστηρίζει τη χρήση μεταβλητών των τεσσάρων τύπων που αναφέρθηκαν (βλ. §7.2). Βέβαια, επισημαίνεται ότι στη ΓΛΩΣΣΑ **όλες οι μεταβλητές πρέπει να δηλώνονται υποχρεωτικά** στην αρχή του προγράμματος στο τμήμα δηλώσεων. Αυτή είναι μία βασική διαφορά από ην ψευδογλώσσα. Ασφαλώς στη ΓΛΩΣΣΑ δε χρησιμοποιούνται τα επιπλέον (πρόσθετα) στοιχεία της ψευδογλώσσας δηλαδή οι λέξεις αλγόριθμος, δεδομένα, αποτελέματα. (βλ. §2.4). Π.χ.οι δηλώσεις μεταβλητών στη ΓΛΩΣΣΑ γίνονται ως εξής:

ΜΕΤΑΒΛΗΤΕΣ

ΠΡΑΓΜΑΤΙΚΕΣ: ποσό

ΑΚΕΡΑΙΕΣ: τιμή

ΧΑΡΑΚΤΗΡΕΣ: όνομα, επίθετο

ΛΟΓΙΚΕΣ: έγγαμος

Σημειώνεται ότι στη ΓΛΩΣΣΑ οι λέξεις ΣΤΑΘΕΡΕΣ, ΜΕΤΑΒΛΗΤΕΣ, ΠΡΑΓΜΑΤΙΚΕΣ, ΑΚΕΡΑΙΕΣ ΧΑΡΑΚΤΗΡΕΣ, ΛΟΓΙΚΕΣ είναι δεσμευμένες. Η έννοια των δεσμευμένων λέξεων έχει ήδη εξηγηθεί αφού τέτοιες λέξεις υπάρχουν και στην ψευδογλώσσα (βλ. §2.4).

7.5 Αριθμητικοί τελεστές

Η ΓΛΩΣΣΑ υποστηρίζει τους αριθμητικούς τελεστές +, -, *, /, ^ (ύψωση σε δύναμη), DIV, (ακέραια διαίρεση), MOD (υπόλοιπο ακέραιας διαίρεσης). Π.χ. 3^2 είναι τρία εις το τετράγωνο δηλ. 9. Ακόμα 7 DIV 2 = 3 (προσοχή όχι 3,5!) και 7 MOD 2 = 1 αφού το πηλίκο της ακέραιας διαίρεσης είναι 3 και το υπόλοιπο 1.

7.6 Συναρτήσεις

Η ΓΛΩΣΣΑ υποστηρίζει πολλές μαθηματικές συναρτήσεις (ημίτονο, συνημίτονο, κλπ).

7.7 Αριθμητικές εκφράσεις

Στη ΓΛΩΣΣΑ οι αριθμητικές εκφράσεις δε διαφέρουν από αυτές της ψευδογλώσσας. Σε μία τέτοια έκφραση μπορεί να υπάρχουν αριθμητικές σταθερές, μεταβλητές, συναρτήσεις, τελεστές και παρενθέσεις. Π.χ μία αριθμητική έκφραση μπορεί να είναι:

(((7 * x) + (9 * y)) ^ 2) / T_P(a)

Μεγάλη προσοχή πρέπει να δοθεί στην **προτεραιότητα** των πράξεων.

7.8 Εντολή εκχώρησης

Στη ΓΛΩΣΣΑ η εντολή εκχώρησης ή απόδοσης τιμής δε διαφέρει από αυτή της ψευδογλώσσας (δηλ. χρησιμοποιείται πάλι το σύμβολο ←). Βέβαια σε καμία περίπτωση η εντολή αυτή δε πρέπει να εκλαμβάνεται ως εξίσωση. Π.χ.

x ← y ^ 2 + 7

όνομα ← 'Ιωάννου'

παντρεμένος ← αληθής

7.9 Εντολές εισόδου-εξόδου

Στη ΓΛΩΣΣΑ οι εντολές εισόδου εξόδου ελάχιστα διαφέρουν από αυτές της ψευδογλώσσας. Για την εισαγωγή δεδομένων χρησιμοποιείται η εντολή Δ**ΙΑΒΑΣΕ**, ενώ για την εμφάνιση των αποτελεσμάτων η εντολή **ΓΡΑΨΕ** (αντί ΕΜΦΑΝΙΣΕ της ψευδογλώσσας). Π.χ.

Δ**ΙΑΒΑΣΕ** όνομα, ηλικία

ΓΡΑΨΕ όνομα, ' είσαι ', ηλικία, 'ετών'

7.10 Δομή προγράμματος

Στη ΓΛΩΣΣΑ κάθε πρόγραμμα έχει την παρακάτω δομή:

ΠΡΟΓΡΑΜΜΑ όνομα_προγράμματος

[**ΣΤΑΘΕΡΕΣ**

.....

]

[**ΜΕΤΑΒΛΗΤΕΣ**

.....

]

ΑΡΧΗ

.....

.....

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

Οι λέξεις που είναι έντονες και με κεφαλαία είναι **δεσμευμένες** λέξεις. Οι τετράγωνες αγκύλες [] σημαίνουν πως οτιδήποτε περιέχεται σ' αυτές είναι προαιρετικό. Στο τμήμα των σταθερών μπορούμε να δηλώσουμε αν θέλουμε σταθερές τιμές με ονόματα. Στο τμήμα των μεταβλητών μπορούμε να δηλώσουμε τις μεταβλητές του προγράμματος μαζί με τους αντίστοιχους τύπους. Στο τμήμα μεταξύ των λέξεων **ΑΡΧΗ** και **ΤΕΛΟΣ** γράφουμε τις εντολές του προγράμματος.

Κεφάλαιο 8ο Επιλογή και Επανάληψη

8.1 Εντολές επιλογής

Στη ΓΛΩΣΣΑ υποστηρίζονται οι εντολές επιλογής **ΑΝ** και **ΕΠΙΛΕΞΕ** όπως ακριβώς και στην ψευδογλώσσα. Οι εντολές αυτές έχουν αναλυθεί διεξοδικά (βλ. §2.4).

8.2 Εντολές επανάληψης

Στη ΓΛΩΣΣΑ υποστηρίζονται οι εντολές επανάληψης **ΟΣΟ...ΕΠΑΝΕΛΑΒΕ**, **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ...ΜΕΧΡΙΣ_ΟΤΟΥ** και **ΓΙΑ...ΑΠΟ...ΜΕΧΡΙ** όπως ακριβώς και στην ψευδογλώσσα. Οι εντολές αυτές έχουν αναλυθεί διεξοδικά (βλ. §2.4).

Κεφάλαιο 9ο Πίνακες

9.1 Μονοδιάστατοι πίνακες

Στη ΓΛΩΣΣΑ υποστηρίζεται η δομή δεδομένων του πίνακα. Βέβαια οι πίνακες πρέπει να δηλώνονται στο τμήμα δηλώσεων, όπως άλλωστε κι όλες οι μεταβλητές. Π.χ. ένας μονοδιάστατος πίνακας ακεραίων 20 θέσεων, δηλώνεται ως εξής:

ΜΕΤΑΒΛΗΤΕΣ

ΑΚΕΡΑΙΕΣ: βαθμοί [20]

Ο χειρισμός των πινάκων γίνεται όπως ακριβώς στην ψευδογλώσσα (βλ. §3.3). Για τη διαχείριση πινάκων συνήθως χρησιμοποιείται η εντολή ΓΙΑ...ΑΠΟ...ΜΕΧΡΙ, αφού σε έναν πίνακα είναι γνωστό το μέγεθός του. Οι βασικότερες λειτουργίες επί των πινάκων είναι οι παρακάτω:

- Προσπέλαση. Αν έχω ένα πίνακα με όνομα βαθμοί και θέλω να προσπελάσω (π.χ. να εμφανίσω) το τέταρτο στοιχείο (κόμβο), τότε χρησιμοποιώ την εντολή **ΓΡΑΨΕ** βαθμοί[4]. Μπορούμε να φανταστούμε το βαθμοί[4] σα μία μεταβλητή. Για να προσπελάσουμε ολόκληρο τον πίνακα, πρέπει βέβαια να χρησιμοποιήσουμε εντολή επανάληψης.

  ΓΙΑ i **ΑΠΟ** 1 **ΜΕΧΡΙ** 20 **ΕΠΑΝΕΛΑΒΕ**

  ΓΡΑΨΕ βαθμοί[i]

  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

- Εισαγωγή. Π.χ. αν σ' ένα πίνακα βαθμοί δώσω την εντολή βαθμοί[3] ← 17 τότε εισάγω ένα νέο κόμβο με τιμή δεκαεπτά στο τρίτο στοιχείο του πίνακα (υποθέτω ότι είναι κενό). Αν ο κόμβος υπάρχει ήδη (δεν είναι κενός) τότε μιλάμε για τροποποίηση (update). Για να εισάγουμε τιμές σ' έναν πίνακα από το πληκτρολόγιο, πρέπει βέβαια να χρησιμοποιήσουμε εντολή επανάληψης.

  ΓΙΑ i **ΑΠΟ** 1 **ΜΕΧΡΙ** 20 **ΕΠΑΝΕΛΑΒΕ**

  Δ**ΙΑΒΑΣΕ** βαθμοί[i]

  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  Ή ισοδύναμα:

  ΓΙΑ i **ΑΠΟ** 1 **ΜΕΧΡΙ** 20 **ΕΠΑΝΕΛΑΒΕ**

  Δ**ΙΑΒΑΣΕ** x

  βαθμοί[i] ← x

  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

- Διαγραφή. Εφόσον ο πίνακας είναι μία στατική δομή δεδομένων (βλ. §3.3) η διαγραφή μπορεί να γίνει μόνο εικονικά. Π.χ. για να διαγραφούν όλα τα στοιχεία ενός πίνακα μπορούμε να θέσουμε την τιμή 0 σε κάθε στοιχείο του πίνακα (αρχικοποίηση).

  ΓΙΑ i **ΑΠΟ** 1 **ΜΕΧΡΙ** 20 **ΕΠΑΝΕΛΑΒΕ**

  βαθμοί[i] ← 0

  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

- Αναζήτηση. Στα προηγούμενα έχουν δοθεί αλγόριθμοι αναζήτησης (βλ. §3.6) .
- Ταξινόμηση. Στα προηγούμενα έχουν δοθεί αλγόριθμοι ταξινόμησης (βλ. §3.7).

9.2 Πότε πρέπει να χρησιμοποιούνται πίνακες

Η χρήση πινάκων εξυπηρετεί όταν θέλουμε να χειριστούμε ένα σύνολο δεδομένων του ίδιου τύπου (π.χ.

ακεραίους). Ασφαλώς η χρήση πίνακα είναι προτιμότερη από τη χρήση πάρα πολλών μεταβλητών του ίδιου τύπου. Π.χ. αν έχουμε να διαχειριστούμε τα ονόματα 10 μαθητών, είναι προτιμότερο να ορίσουμε έναν πίνακα 10 θέσεων παρά 10 απλές μεταβλητές.

Τα βασικότερα μειονεκτήματα των πινάκων είναι τα εξής:

- Δέσμευση μνήμης. Με τη δήλωσή του, κάθε πίνακας δεσμεύει στατικά πολλές θέσεις μνήμης. Ακόμα κι αν ο πίνακας δε χρησιμοποιείται στο πρόγραμμα η μνήμη δεσμεύεται(!). Επομένως, η χρήση πολλών και μεγάλων πινάκων, ιδιαίτερα αν γίνεται δίχως λόγο, επιβαρύνει το πρόγραμμα.
- Περιορισμός στο μέγεθος του πίνακα. Οι πίνακες ως στατικές δομές έχουν σταθερό μέγεθος που δηλώνεται στην αρχή του προγράμματος. Αυτό μπορεί να δημιουργήσει πρόβλημα σε ορισμένες περιπτώσεις. Π.χ. έστω πίνακας Α που έχει δηλωθεί ως πίνακας ακεραίων 100 θέσεων. Αν στο πρόγραμμα υπάρχει εντολή του τύπου Α[i] ← x, θα προκληθεί σφάλμα εκτέλεσης σε περίπτωση που η τιμή του i υπερβεί το 100.

9.3 Πολυδιάστατοι πίνακες

Στη ΓΛΩΣΣΑ υποστηρίζεται η δομή του πίνακα πολλών διαστάσεων. Π.χ. ένας δισδιάστατος πίνακας ακεραίων 5 γραμμών και 7 στηλών, δηλώνεται ως εξής:

ΜΕΤΑΒΛΗΤΕΣ

ΧΑΡΑΚΤΗΡΕΣ: πρόγραμμα [5, 7]

Ο χειρισμός των πινάκων γίνεται όπως περίπου στους μονοδιάστατους πίνακες. Οι βασικότερες λειτουργίες επί των δισδιάστατων πινάκων είναι οι παρακάτω:

- Προσπέλαση. Αν έχω ένα δισδιάστατο πίνακα 5 γραμμών και 7 στηλών με όνομα πρόγραμμα και θέλω να προσπελάσω (π.χ. να εμφανίσω) το στοιχείο (κόμβο) της τρίτης γραμμής και της πέμπτης στήλης, τότε χρησιμοποιώ την εντολή **ΓΡΑΨΕ** πρόγραμμα[3,5]. Μπορούμε να φανταστούμε το πρόγραμμα[3,5] σα μία μεταβλητή που περιέχει το μάθημα της τρίτης εργάσιμης μέρας (π.χ. Τετάρτη) και της πέμπτης ώρας του σχολικού προγράμματος. Για να προσπελάσουμε ολόκληρο τον πίνακα, πρέπει βέβαια να χρησιμοποιήσουμε εντολή επανάληψης.

```
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5 ΕΠΑΝΕΛΑΒΕ
        ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 7 ΕΠΑΝΕΛΑΒΕ
                ΓΡΑΨΕ πρόγραμμα[i,j]
        ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

- Εισαγωγή. Π.χ. αν σ' ένα πίνακα πρόγραμμα δώσω την εντολή πρόγραμμα[1,3] ← 'Αγγλικά' τότε εισάγω ένα νέο κόμβο με τιμή 'Αγγλικά' στο στοιχείο της πρώτης γραμμής και τρίτης στήλης του πίνακα (υποθέτω ότι είναι κενό). Αν ο κόμβος υπάρχει ήδη (δεν είναι κενός) τότε μιλάμε για τροποποίηση (update). Για να εισάγουμε τιμές σ' έναν πίνακα από το πληκτρολόγιο, πρέπει να χρησιμοποιήσουμε **εμφωλευμένες** εντολές επανάληψης:

```
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5 ΕΠΑΝΕΛΑΒΕ
        ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 7 ΕΠΑΝΕΛΑΒΕ
                ΔΙΑΒΑΣΕ πρόγραμμα[i,j]
```

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

9.4 Τυπικές επεξεργασίες πινάκων

Οι βασικές επεξεργασίες σε πίνακες τόσο μονοδιάστατους όσο και δισδιάστατους έχουν ήδη αναφερθεί στα προηγούμενα.

Κεφάλαιο 10ο Υποπρογράμματα

10.1 Τμηματικός προγραμματισμός

Οι βασικές αρχές του τμηματικού προγραμματισμού έχουν ήδη αναφερθεί στα προηγούμενα (βλ. §6.4).

Σημαντικός είναι ο ορισμός του τμηματικού προγραμματισμού (σελ. 205 σχολικού βιβλίου).

Πρακτικά ο τμηματικός προγραμματισμός σημαίνει το ‘σπάσιμο’ ενός προγράμματος (ουσιαστικά ενός αλγορίθμου που μπορεί να είναι μεγάλος και περίπλοκος), σε επιμέρους απλούστερους υποαλγορίθμους ή υποπρογράμματα. Ήδη από το κεφάλαιο 1 (βλ. σχήμα §3.1) έχει αναφερθεί ότι πολλές φορές (σε γενικές γραμμές) η αντιμετώπιση ενός προβλήματος αποτελείται από τυποποιημένες διαδικασίες ή ενέργειες (π.χ. είσοδο δεδομένων, επεξεργασίες, έξοδο αποτελεσμάτων). Κάθε μία από αυτές τις διαδικασίες μπορεί να αποτελεί ένα υποπρόγραμμα (υποαλγόριθμο) ή να ‘σπάει’ πάλι σε επιμέρους υποπρογράμματα. Δηλαδή ένα **υποπρόγραμμα** είναι ένα αυτόνομο τμήμα προγράμματος και συνήθως γράφεται ξεχωριστά από το υπόλοιπο πρόγραμμα. Π.χ. αν έχουμε ένα μεγάλο και περίπλοκο πρόγραμμα, μπορούμε να το αναλύσουμε σε πολλά υποπρογράμματα και να αναθέσουμε κάθε υποπρόγραμμα σε άλλο προγραμματιστή. Έτσι μία ογκώδης εργασία μπορεί να διαμοιραστεί σε πολλούς προγραμματιστές. Το αρχικό πρόγραμμα (κύριο πρόγραμμα) μέσω ειδικής εντολής (βλ. §10.5) καλεί κάθε υποπρόγραμμα για να το ενεργοποιήσει. Ασφαλώς είναι δυνατό ένα υποπρόγραμμα να καλέσει άλλο υποπρόγραμμα, κ.ο.κ.

10.2 Χαρακτηριστικά των υποπρογραμμάτων

Το ‘σπάσιμο’ ενός προγράμματος σε επιμέρους υποπρογράμματα προϋποθέτει την αντίστοιχη διάκριση του αρχικού προβλήματος σε μικρότερα υποπροβλήματα. Όπως έχει ήδη ανεφερθεί αυτό γίνεται στη φάση της ανάλυσης (βλ. §4.1)

Σημαντικές είναι οι ιδιότητες των υποπρογραμμάτων (σελ. 208 σχολικού βιβλίου).

10.3 Πλεονεκτήματα του τμηματικού προγραμματισμού

Σημαντικά είναι όλα τα χαρακτηριστικά του τμηματικού προγραμματισμού (σελ. 208-209 σχολικού βιβλίου).

10.4 Παράμετροι

Σημαντικός είναι ο ορισμός της παραμέτρου (σελ. 210 σχολικού βιβλίου).

Πρακτικά οι παράμετροι είναι απλές μεταβλητές που χρησιμοποιούνται για την επικοινωνία μεταξύ υποπρογραμμάτων ή μεταξύ του κυρίου προγράμματος και υποπρογραμμάτων. Δηλαδή η παράμετρος περνάει τιμές από ένα τμήμα προγράμματος σε άλλο (βλ. §10.5).

10.5 Διαδικασίες και συναρτήσεις

Υπάρχουν δύο κατηγορίες υποπρογραμμάτων οι **διαδικασίες** (procedures) κι οι **συναρτήσεις** (functions).

Η διαδικασία είναι η γενικότερη και ισχυρότερη μορφή υποπρογραμμάτων. Μπορούμε να τη θεωρήσουμε ως μικρό πρόγραμμα διότι είναι δυνατό να εκτελέσει οποιαδήποτε λειτουργία.

Η συνάρτηση είναι η πιο περιορισμένη μορφή υποπρογραμμάτων. Μπορούμε να τη θεωρήσουμε ως σύντομο πρόγραμμα που υπολογίζει μόνο μία τιμή που φέρει το όνομά της. Η τιμή αυτή επιστρέφετε στο κυρίως πρόγραμμα ή στο υποπρόγραμμα που την κάλεσε. Τυπικό παράδειγμα είναι οι μαθηματικές συναρτήσεις. Έχει ήδη αναφερθεί (βλ. §7.6) ότι οι ΓΛΩΣΣΑ διαθέτει ορισμένες ενσωματωμένες βασικές συναρτήσεις (ημίτονο, συνημίτονο, κλπ). Πέρα από αυτές, ο κάθε προγραμματιστής μπορεί να ορίσει κι άλλες δικές του συναρτήσεις

Είναι προφανές ότι κάθε συνάρτηση μπορεί να γραφεί ως διαδικασία, ενώ το αντίστροφο δεν ισχύει.

Στη ΓΛΩΣΣΑ υποστηρίζονται τόσο οι διαδικασίες όσο κι οι συναρτήσεις. Έστω ένα απλό πρόβλημα υπολογισμού του τετραγώνου ενός θετικού αριθμού. Δηλαδή δίνεται ένας ακέραιος αριθμός x και θέλουμε να υπολογίσουμε και να εμφανίσουμε το τετράγωνό του. Ασφαλώς ο αλγόριθμος μπορεί να γραφεί δίχως χρήση υποπρογραμμάτων. Μπορούμε όμως να ‘σπάσουμε’ τον αλγόριθμο σε τρία επιμέρους μέρη. Την εισαγωγή των δεδομένων (διαδικασία), τον υπολογισμό του τετραγώνου (συνάρτηση) και την εμφάνιση των αποτελεσμάτων (διαδικασία).

```
ΔΙΑΔΙΚΑΣΙΑ Είσοδος_δεδομένων(Αριθμός)
ΜΕΤΑΒΛΗΤΕΣ
   ΑΚΕΡΑΙΕΣ: Αριθμός
ΑΡΧΗ
   ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
         ΓΡΑΨΕ ‘Δώστε ένα θετικό ακέραιο: ’
         ΔΙΑΒΑΣΕ Αριθμός
   ΜΕΧΡΙΣ_ΟΤΟΥ Αριθμός > 0
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Προφανώς οι λέξεις που είναι έντονες και με κεφαλαία είναι **δεσμευμένες** λέξεις. Η παραπάνω διαδικασία επιστρέφει έναν ακέραιο αριθμό (τη μεταβλητή Αριθμός) που αποτελεί παράμετρό της.

```
ΣΥΝΑΡΤΗΣΗ Τετράγωνο(Αριθμός): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
   ΠΡΑΓΜΑΤΙΚΕΣ: Αριθμός
ΑΡΧΗ
   Τετράγωνο ← Αριθμός ^ 2
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Πάλι οι λέξεις που είναι έντονες και με κεφαλαία είναι **δεσμευμένες** λέξεις. Η παραπάνω συνάρτηση δέχεται ως παράμετρο έναν ακέραιο αριθμό (τη μεταβλητή Αριθμός) και επιστρέφει το τετράγωνό του.

```
ΔΙΑΔΙΚΑΣΙΑ Εμφάνιση_αποτελεσμάτων(Αριθμός)
ΜΕΤΑΒΛΗΤΕΣ
   ΑΚΕΡΑΙΕΣ: Αριθμός
ΑΡΧΗ
```

ΓΡΑΨΕ 'Το αποτέλεσμα είναι: ', Αριθμός

ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ

Προφανώς και πάλι οι λέξεις που είναι έντονες και με κεφαλαία είναι **δεσμευμένες** λέξεις. Η παραπάνω διαδικασία δέχεται ως παράμετρο έναν ακέραιο αριθμό (τη μεταβλητή Αριθμός) και απλά τον εμφανίζει.

Το κυρίως πρόγραμμα έχει ως εξής;

ΠΡΟΓΡΑΜΜΑ Υπολογισμός_τετραγώνου

ΜΕΤΑΒΛΗΤΕΣ

ΑΚΕΡΑΙΕΣ: x, y

ΑΡΧΗ

ΚΑΛΕΣΕ Είσοδος_δεδομένων(x)

y ← Τετράγωνο(x)

ΚΑΛΕΣΕ Εμφάνιση_αποτελεσμάτων(y)

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

Οι λέξεις που είναι έντονες και με κεφαλαία είναι **δεσμευμένες** λέξεις. Οι διαδικασίες καλούνται με χρήση της εντολής **ΚΑΛΕΣΕ**, ενώ οι συναρτήσεις απλά με το όνομά τους που ισούται με την τιμή που επιστρέφουν.