ΠΛΗΡΟΦΟΡΙΚΗ
Γ’ ΤΑΞΗΣ
ΓΕΝΙΚΟΥ ΛΥΚΕΙΟΥ

Σημειώσεις θεωρίας εξεταστέας ύλης

Δημόπουλος Ιωάννης
Καθηγητής Πληροφορικής

Περιεχόμενα

Ανάλυση Προβλήματος 4
Δεδομένο 4
Επεξεργασία Δεδομένων 4
Πληροφορία 4
Πρόβλημα 4
Κατανόηση Προβλήματος 5
Δομή Προβλήματος 5
Καθορισμός Απαιτήσεων 5
Βασικές Έννοιες Αλγορίθμων 7
Αλγόριθμος 7
Κριτήρια Πληρότητας Αλγορίθμου 7
Σπουδαιότητα Αλγορίθμων 8
Περιγραφή και Αναπαράσταση Αλγορίθμων 8
Διάγραμμα Ροής 9
Εισαγωγή στον Προγραμματισμό 10
Πρόγραμμα – Προγραμματισμός – Προγραμματιστής 10
Φυσικές και Τεχνητές Γλώσσες 10
Διαφορές φυσικών και τεχνητών γλωσσών 11
Τεχνικές Σχεδίασης Προγραμμάτων 11
Ιεραρχική σχεδίαση προγράμματος 11
Τμηματικός προγραμματισμός 11
Δομημένος προγραμματισμός 12
Αντικειμενοστραφής Προγραμματισμός 12
Προγραμματιστικά Περιβάλλοντα 13
Λειτουργία Μεταγλώττισης 13
Βασικές Έννοιες Προγραμματισμού 15
Το αλφάβητο της ΓΛΩΣΣΑΣ 15
Τύποι δεδομένων 15
Σταθερές 15
Μεταβλητές 16

Λειτουργία και Βασικά Χαρακτηριστικά Σταθερών και Μεταβλητών 16
Τελεστές και Εκφράσεις 16
Αριθμητικοί Τελεστές 16
Συγκριτικοί τελεστές 17
Λογικοί τελεστές 17
Συναρτήσεις 17
Αριθμητικές Εκφράσεις 18
Λογικές Εκφράσεις 19
Δομές Δεδομένων και Πίνακες 20
Πληροφορική και Δεδομένα 20
Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα 20
Στατικές και Δυναμικές Δομές Δεδομένων 20
Βασικές Λειτουργίες Δομών Δεδομένων 21
Πίνακας 22
Διαστάσεις Πινάκων 22
Τετραγωνικοί Πίνακες 22
Παράλληλοι Πίνακες 23
Πλεονεκτήματα – Μειονεκτήματα Πινάκων 23
Τυπικές Επεξεργασίες Πινάκων 23
Στοίβα 24
Ουρά 24
Λίστα 25
Διαφορές Λίστας σε σχέση με τον Πίνακα - Πλεονεκτήματα – Μειονεκτήματα 25
Βασικές πράξεις των συνδεδεμένων λιστών 26
Δένδρα 27
Υποδένδρα 28
Ισότητα Δένδρων και Διατεταγμένα Δένδρα 28
Σπουδαιότητα των Δένδρων 28
Δυαδικά Δένδρα 29
Δυαδικά Δένδρα Αναζήτησης 29
Ισορροπημένο Δυαδικό Δένδρο Αναζήτησης 30
Γράφοι 30
Τύποι Γράφων 31

Τμηματικός Προγραμματισμός 32
Χαρακτηριστικά των υποπρογραμμάτων 32
Πλεονεκτήματα του Τμηματικού Προγραμματισμού 33
Παράμετροι 33
Διαδικασίες 35
Συναρτήσεις 35
Διαφορές Διαδικασιών – Συναρτήσεων 36
Εκσφαλμάτωση 37
Συντακτικά λάθη 37
Λάθη κατά την εκτέλεση 37
Λογικά λάθη 38
Μέθοδος «Μαύρο κουτί» 38
Η μεθοδολογία για τη δημιουργία σεναρίων ελέγχου είναι η εξής: 39
Αντικειμενοστραφής Προγραμματισμός 40
Ορισμός 40
Μεθοδολογία 40
Διαγραμματική Αναπαράσταση 41
Κλάσεις: Αφαιρετικότητα και Ενθυλάκωση 41
Κληρονομικότητα 42
Πως γενικοποιούμε κλάσεις σε μια υπερκλάση και πως διαχειριζόμαστε ιδιότητες και μεθόδους.42
Πολυμορφισμός 43

Ανάλυση Προβλήματος

Δεδομένο

Με τον όρο **Δεδομένο** δηλώνεται οποιοδήποτε στοιχείο μπορεί να γίνει αντιληπτό από έναν τουλάχιστον παρατηρητή με μία από τις πέντε αισθήσεις του.

Δεδομένα είναι μια παράσταση γεγονότων, εννοιών ή εντολών, σε τυποποιημένη μορφή που είναι κατάλληλη για επικοινωνία, ερμηνεία ή επεξεργασία από άνθρωπο ή αυτόματα μέσα.

Επεξεργασία Δεδομένων

Επεξεργασία Δεδομένων είναι η διαδικασία κατά την οποία ένας "μηχανισμός" δέχεται δεδομένα, τα επεξεργάζεται σύμφωνα με έναν προκαθορισμένο τρόπο και αποδίδει πληροφορίες.

Επί χιλιετίες ο "μηχανισμός" επεξεργασίας των δεδομένων ήταν και εξακολουθεί να είναι ο ανθρώπινος εγκέφαλος. Στις μέρες μας, ένας άλλος "μηχανισμός" επεξεργασίας δεδομένων είναι ο υπολογιστής.

Επεξεργασία δεδομένων είναι η συστηματική εκτέλεση πράξεων σε δεδομένα. Παραδείγματα: χειρισμός, συγχώνευση, ταξινόμηση, μεταγλώττιση, κ.ά..

Πληροφορία

Με τον όρο **Πληροφορία** αναφέρεται οποιοδήποτε γνωσιακό στοιχείο προέρχεται από επεξεργασία δεδομένων.

Πληροφορία είναι γνώση που αφορά πράγματα όπως πράξεις, έννοιες, αντικείμενα, γεγονότα, ιδέες ή διεργασίες που μέσα σε συγκεκριμένο κείμενο έχουν μια ιδιαίτερη σημασία.

Πρόβλημα

Με τον όρο **Πρόβλημα** εννοείται μια κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, η δε λύση της δεν είναι γνωστή, ούτε προφανής.

Κατανόηση Προβλήματος

Η κατανόηση ενός προβλήματος αποτελεί συνάρτηση δύο παραγόντων, της **σωστής διατύπωσης** εκ μέρους του δημιουργού του και της αντίστοιχα **σωστής ερμηνείας** από τη μεριά εκείνου που καλείται να το αντιμετωπίσει.

Δομή Προβλήματος

Με τον όρο **δομή ενός προβλήματος** αναφερόμαστε στα συστατικά του μέρη, στα επιμέρους τμήματα που το αποτελούν καθώς επίσης και στον τρόπο που αυτά τα μέρη συνδέονται μεταξύ τους.

Η δυσκολία αντιμετώπισης των προβλημάτων ελαττώνεται όσο περισσότερο προχωράει η ανάλυσή τους σε απλούστερα προβλήματα.

Υπάρχουν δυο τρόποι ανάλυσης ενός προβλήματος:

- **Φραστική ανάλυση:** Το πρόβλημα χωρίζεται σε επιμέρους μικρότερα προβλήματα και χρησιμοποιείται απλός λόγος με βασικές έννοιες (συζήτηση) για την περιγραφή του όλου.
- **Διαγραμματική αναπαράσταση:** Το αρχικό πρόβλημα αναπαρίσταται από ένα ορθογώνιο παραλληλόγραμμο και κάθε ένα από τα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα αναπαρίσταται επίσης από ένα ορθογώνιο παραλληλόγραμμο. Τα παραλληλόγραμμα που αντιστοιχούν στα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα σχηματίζονται ένα επίπεδο χαμηλότερα. Έτσι σε κάθε κατώτερο επίπεδο, δημιουργείται η γραφική αναπαράσταση των προβλημάτων στα οποία αναλύονται τα προβλήματα του αμέσως υψηλότερου επιπέδου.

Καθορισμός Απαιτήσεων

Η σωστή επίλυση ενός προβλήματος προϋποθέτει τον επακριβή **προσδιορισμό των δεδομένων** που παρέχει το πρόβλημα. Απαιτεί επίσης τη λεπτομερειακή **καταγραφή των ζητούμενων** που αναμένονται σαν αποτελέσματα της επίλυσης του προβλήματος.

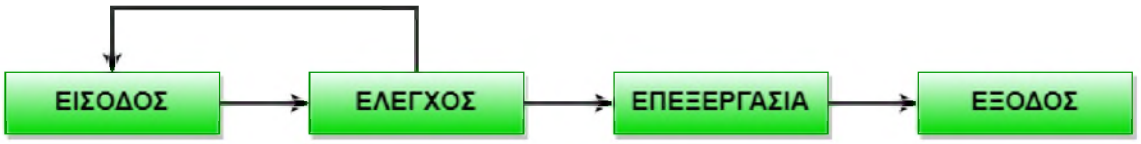
Δεν είναι πάντοτε εύκολο να διακρίνει κάποιος τα δεδομένα. Υπάρχουν πολλές περιπτώσεις προβλημάτων όπου τα δεδομένα θα πρέπει να "ανακαλυφθούν" μέσα στα λεγόμενα του προβλήματος. Η διαδικασία αυτή απαιτεί προσοχή, συγκέντρωση και σκέψη. Μεθοδολογία προσδιορισμού των δεδομένων ενός προβλήματος δεν υπάρχει, ούτε και μεθοδολογία εντοπισμού και αποσαφήνισης των ζητούμενων ενός προβλήματος.

Δεν είναι πάντοτε ιδιαίτερα κατανοητό τι ακριβώς ζητάει ένα πρόβλημα. Σε μια τέτοια περίπτωση θα πρέπει να θέτονται μια σειρά από ερωτήσεις με στόχο τη διευκρίνιση πιθανών απορριών σχετικά με τα ζητούμενα, τον τρόπο παρουσίασής τους, το εύρος τους κ.λπ. Οι ερωτήσεις αυτές μπορούν να απευθύνονται είτε στο δημιουργό του προβλήματος, είτε στον ίδιο μας τον εαυτό αν εμείς καλούμαστε να αντιμετωπίσουμε το πρόβλημα.

Τα στάδια αντιμετώπισης ενός προβλήματος είναι τρία:

- **Κατανόηση**, όπου απαιτείται η σωστή και πλήρης αποσαφήνιση των δεδομένων και των ζητούμενων του προβλήματος.
- **Ανάλυση**, όπου το αρχικό πρόβλημα διασπάται σε άλλα επιμέρους απλούστερα προβλήματα.
- **Επίλυση**, όπου υλοποιείται η λύση του προβλήματος, μέσω της λύσης των επιμέρους προβλημάτων.

Οι απαιτούμενες ενέργειες για την επίλυση ενός προβλήματος (αναφορικά με τα δεδομένα) είναι οι εξής:



- **Είσοδος**, κατά την οποία συλλέγονται και καταχωρούνται τα απαραίτητα δεδομένα.
- **Έλεγχος**, όπου τα δεδομένα ελέγχονται ως προς την ορθότητά τους και γίνονται οι απαραίτητες διορθώσεις, αν απαιτείται.
- **Επεξεργασία**, όπου γίνονται οι απαραίτητοι υπολογισμοί προκειμένου να βρεθούν τα ζητούμενα αποτελέσματα.
- **Έξοδος**, όπου εξάγεται η παραγόμενη πληροφορία, σύμφωνα με τις προδιαγραφές των απαιτήσεων επίλυσης του προβλήματος.

Βασικές Έννοιες Αλγορίθμων

Αλγόριθμος

Αλγόριθμος είναι μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος.

Προσοχή: Η έννοια του αλγόριθμου δεν συνδέεται αποκλειστικά και μόνο με προβλήματα της Πληροφορικής.

Κριτήρια Πληρότητας Αλγορίθμου

- **Είσοδος**

  Καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται, όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια άλλων απλών εντολών.

  Π.χ. Πρέπει να υπάρχει μια εντολή **Διάβασε** ή **Δεδομένα // … //** .

- **Έξοδος**

  Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν άλλο αλγόριθμο. Π.χ. Πρέπει να υπάρχει μια εντολή **Γράψε** ή **Εκτύπωσε** ή **Εμφάνισε** ή **Αποτελέσματα // … //** .

- **Καθοριστικότητα**

  Κάθε εντολή ενός αλγορίθμου πρέπει να εμπεριέχεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσης της. Π.χ. να μη γίνεται **διαίρεση με το 0**.

- **Περατότητα**

  Κάθε αλγόριθμος πρέπει να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Π.χ. να μην χρησιμοποιούμε δομή **ατέρμονης επανάληψης**.

- **Αποτελεσματικότητα**

  Κάθε μεμονωμένη εντολή του αλγορίθμου πρέπει να είναι απλή. Αυτό σημαίνει ότι μία εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη. Π.χ. η εντολη **x < y < z** είναι λάθος, η σωστή είναι x < y ΚΑΙ y < z.

Σπουδαιότητα Αλγορίθμων

Η Πληροφορική, λοιπόν, μπορεί να ορισθεί ως η επιστήμη που μελετά τους αλγορίθμους από τις ακόλουθες σκοπιές:

- **Υλικού**. Η ταχύτητα εκτέλεσης ενός αλγορίθμου επηρεάζεται από τις διάφορες τεχνολογίες υλικού, δηλαδή από τον τρόπο που είναι δομημένα σε μία ενιαία αρχιτεκτονική τα διάφορα συστατικά του υπολογιστή (δηλαδή ανάλογα με το αν ο υπολογιστής έχει κρυφή μνήμη και πόση, ανάλογα με την ταχύτητα της κύριας και δευτερεύουσας μνήμης κ.ο.κ.).
- **Γλωσσών Προγραμματισμού**. Το είδος της γλώσσας προγραμματισμού που χρησιμοποιείται (δηλαδή, χαμηλότερου ή υψηλότερου επιπέδου) αλλάζει τη δομή και τον αριθμό των εντολών ενός αλγορίθμου. Γενικά μία γλώσσα που είναι χαμηλότερου επιπέδου (όπως η assembly ή η γλώσσα C) είναι ταχύτερη από μία άλλη γλώσσα που είναι υψηλοτέρου επιπέδου (όπως η Basic ή Pascal). Ακόμη, σημειώνεται ότι διαφορές συναντώνται μεταξύ των γλωσσών σε σχέση με το πότε εμφανίσθηκαν.
- **Θεωρητική**. Το ερώτημα που συχνά τίθεται είναι αν πράγματι υπάρχει ή όχι κάποιος αποδοτικός αλγόριθμος για την επίλυση ενός προβλήματος. Η εξέταση αυτού του ερωτήματος είναι δύσκολο να σχολιασθεί στα πλαίσια του βιβλίου αυτού, επειδή απαιτεί μεγάλη θεωρητική κατάρτιση. Ωστόσο η προσέγγιση αυτή είναι ιδιαίτερα σημαντική, γιατί προσδιορίζει τα όρια της λύσης που θα βρεθεί σε σχέση με ένα συγκεκριμένο πρόβλημα.
- **Αναλυτική**. Μελετώνται οι υπολογιστικοί πόροι που απαιτούνται από έναν αλγόριθμο, όπως για παράδειγμα το μέγεθος της κύριας και της δευτερεύουσας μνήμης, ο χρόνος για λειτουργίες CPU και για λειτουργίες εισόδου/εξόδου κ.λπ.

Περιγραφή και Αναπαράσταση Αλγορίθμων

- **Ελεύθερο κείμενο**, που αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Έτσι εγκυμονεί τον κίνδυνο ότι μπορεί εύκολα να οδηγήσει σε μη εκτελέσιμη παρουσίαση παραβιάζοντας το τελευταίο χαρακτηριστικό των αλγορίθμων, δηλαδή την αποτελεσματικότητα.

- **Διαγραμματικές τεχνικές**, που συνιστούν ένα γραφικό τρόπο παρουσίασης του αλγορίθμου. Από τις διάφορες διαγραμματικές τεχνικές που έχουν επινοηθεί, η πιο παλιά και η πιο γνωστή ίσως, είναι το **διάγραμμα ροής**. Ωστόσο η χρήση διαγραμμάτων ροής για την παρουσίαση αλγορίθμων δεν αποτελεί την καλύτερη λύση, γι' αυτό και εμφανίζονται όλο και σπανιότερα στη βιβλιογραφία και στην πράξη.

- **Φυσική γλώσσα κατά βήματα**. Στην περίπτωση αυτή χρειάζεται προσοχή, γιατί μπορεί να παραβιασθεί το τρίτο βασικό χαρακτηριστικό ενός αλγορίθμου, όπως προσδιορίσθηκε προηγουμένως, δηλαδή το κριτήριο του καθορισμού.

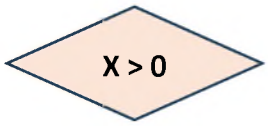
- **Κωδικοποίηση**, δηλαδή με ένα πρόγραμμα γραμμένο είτε σε μία ψευδογλώσσα είτε σε κάποια γλώσσα προγραμματισμού που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

Διάγραμμα Ροής

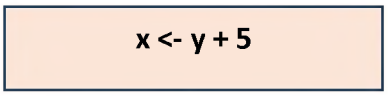
Ένα διάγραμμα ροής αποτελείται από ένα σύνολο γεωμετρικών σχημάτων, όπου το καθένα δηλώνει μία συγκεκριμένη ενέργεια ή λειτουργία. Τα γεωμετρικά σχήματα ενώνονται μεταξύ τους με βέλη, που δηλώνουν τη σειρά εκτέλεσης των ενεργειών αυτών. Τα κυριότερα χρησιμοποιούμενα γεωμετρικά σχήματα είναι τα εξής:

- **έλλειψη**, που δηλώνει την αρχή και το τέλος του κάθε αλγορίθμου,

- **ρόμβος**, που δηλώνει μία ερώτηση με δύο ή περισσότερες εξόδους για απάντηση,



- **ορθογώνιο**, που δηλώνει την εκτέλεση μίας ή περισσότερων πράξεων, και



- **πλάγιο παραλληλόγραμμο**, που δηλώνει είσοδο ή έξοδο στοιχείων.

Προσοχή: Το παραπάνω σχήματα ενώνονται με **βέλη** τα οποία καταδικνύουν και τη ροή του αλγορίθμου.

Εισαγωγή στον Προγραμματισμό

Πρόγραμμα – Προγραμματισμός – Προγραμματιστής

Πρόγραμμα (υπολογιστή) είναι μια ακολουθία εντολών κατάλληλων για επεξεργασία. Η επεξεργασία περιλαμβάνει τη χρήση μεταφραστικού προγράμματος για να προετοιμάσει το πρόγραμμα για εκτέλεση, καθώς και την ίδια την εκτέλεση του προγράμματος.

Προγραμματισμός ονομάζεται η διαδικασία δημιουργίας προγραμμάτων υπολογιστή.

Προγραμματιστής είναι το πρόσωπο υπεύθυνο για το σχεδιασμό, εγγραφή, έλεγχο, διόρθωση, συντήρηση και τεκμηρίωση ενός προγράμματος.

Φυσικές και Τεχνητές Γλώσσες

Οι **φυσικές γλώσσες** χρησιμοποιούνται για την επικοινωνία μεταξύ των ανθρώπων, ενώ οι **τεχνητές γλώσσες** χρησιμοποιούνται για την επικοινωνία του ανθρώπου με μια υπολογιστική μηχανή.

Μία γλώσσα προσδιορίζεται από το **αλφάβητό** της, το **λεξιλόγιό** της, τη **γραμματική** της και τη **σημασιολογία** της.

- **Αλφάβητο** μίας γλώσσας καλείται το σύνολο των βασικών δομικών στοιχείων που χρησιμοποιούνται από τη γλώσσα. Π.χ. Α-Ω, α-ω, 0-9, σύμβολα (, . ! ; :).
- Το **λεξιλόγιο** αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από τη γλώσσα. Π.χ. ΓΡΑΨΕ, ΔΙΑΒΑΣΕ, ΠΡΟΓΡΑΜΜΑ
- Η **γραμματική** αποτελείται από το τυπικό ή τυπολογικό και το συντακτικό.

  Τυπικό είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία λέξη είναι αποδεκτή (κλίση και ορθογραφία των λέξεων).

  Συντακτικό είναι το σύνολο των κανόνων που καθορίζει τη νομιμότητα της διάταξης και της σύνδεσης των λέξεων της γλώσσας για τη δημιουργία προτάσεων. Η γνώση του συντακτικού επιτρέπει τη δημιουργία σωστών προτάσεων στις φυσικές γλώσσες, ενώ στις γλώσσες προγραμματισμού τη δημιουργία σωστών εντολών.
- **Σημασιολογία** είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα. Στις γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας.

Διαφορές φυσικών και τεχνητών γλωσσών

Μία βασική διαφορά μεταξύ φυσικών και τεχνητών γλωσσών είναι **η δυνατότητα εξέλιξής τους**. Οι **φυσικές γλώσσες εξελίσσονται συνεχώς**, νέες λέξεις δημιουργούνται, κανόνες γραμματικής και σύνταξης αλλάζουν με την πάροδο του χρόνου και αυτό γιατί η γλώσσα χρησιμοποιείται για την επικοινωνία μεταξύ ανθρώπων, που εξελίσσονται και αλλάζουν ανάλογα με τις εποχές και τον κοινωνικό περίγυρο.

Αντίθετα οι **τεχνητές γλώσσες χαρακτηρίζονται από στασιμότητα**, αφού κατασκευάζονται συνειδητά για ένα συγκεκριμένο σκοπό. Οι γλώσσες προγραμματισμού αλλάζουν σε επίπεδο διαλέκτου ή σε επίπεδο επέκτασης.

Τεχνικές Σχεδίασης Προγραμμάτων

Ιεραρχική σχεδίαση προγράμματος

Η ιεραρχική σχεδίαση ή ιεραρχικός προγραμματισμός χρησιμοποιεί τη στρατηγική της συνεχούς διαίρεσης του προβλήματος σε υποπροβλήματα.

Η τεχνική της ιεραρχικής σχεδίασης και επίλυσης ή η διαδικασία σχεδίασης "από επάνω προς τα κάτω" περιλαμβάνει τον καθορισμό των βασικών λειτουργιών ενός προγράμματος, σε ανώτερο επίπεδο, και στη συνέχεια τη διάσπαση των λειτουργιών αυτών σε όλο και μικρότερες λειτουργίες, μέχρι το τελευταίο επίπεδο που οι λειτουργίες είναι πολύ άπλες, ώστε να επιλυθούν εύκολα.

Τμηματικός προγραμματισμός

Η Ιεραρχική Σχεδίαση Προγράμματος υλοποιείται με τον **Τμηματικό Προγραμματισμό**. Μετά την ανάλυση του προβλήματος σε αντίστοιχα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί ανεξάρτητη ενότητα (module), που γράφεται ξεχωριστά από τα υπόλοιπα τμήματα προγράμματος.

Τμηματικός Προγραμματισμός ονομάζεται η τεχνική σχεδίασης και ανάπτυξης των προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων. Π.χ. κλάσεις - συναρτήσεις - διαδικασίες.

Πλεονεκτήματα:

- διευκολύνει τη δημιουργία του προγράμματος,
- μειώνει τα λάθη και
- επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

Δομημένος προγραμματισμός

Ο **Δομημένος Προγραμματισμός** είναι η επικρατέστερη μεθοδολογία δημιουργίας προγραμμάτων, η οποία στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Πλεονεκτήματα:

- Δημιουργία απλούστερων προγραμμάτων
- Άμεση μεταφορά των αλγορίθμων σε προγράμματα
- Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους
- Ευκολότερη διόρθωση και συντήρηση

GOTO: Το μαύρο πρόβατο του προγραμματισμού

Η εντολή GOTO έχει ως αποτέλεσμα **την αλλαγή της ροής του προγράμματος**, της διακλάδωσης σε μία άλλη εντολή του προγράμματος **εκτός από την επόμενη**. Ο δομημένος προγραμματισμός προήλθε από την ανάγκη του περιορισμού της ανεξέλεγκτης χρήσης του GOTO.

Αντικειμενοστραφής Προγραμματισμός

Αντικειμενοστραφής προγραμματισμός ή **αντικειμενοστραφής σχεδίαση** είναι μια μεθοδολογία ανάπτυξης εφαρμογών η οποία στηρίζεται σε αυτόνομες προγραμματιστικές οντότητες με δική τους ταυτότητα και συμπεριφορά. Οι οντότητες αυτές καλούνται **αντικείμενα**, αντιστοιχούν σε φυσικές οντότητες ή έννοιες του φυσικού μας κόσμου, και δομούνται με βάση δεδομένα (ιδιότητες) που προσδιορίζουν την υπόστασή τους και ενέργειες (κανόνες συμπεριφοράς) που εφαρμόζονται πάνω στα δεδομένα. Σε μια εφαρμογή, ένα αντικείμενο είναι ο ομαδοποιημένος συνδυασμός δεδομένων και κώδικα, τα οποία έχουμε τη δυνατότητα να χειριστούμε ενιαία. Τα δεδομένα αποτελούν τα χαρακτηριστικά ενός αντικειμένου και αναφέρονται ως **ιδιότητες** ενώ οι ενέργειες καθορίζουν τη συμπεριφορά του. Οι ενέργειες στον αντικειμενοστραφή προγραμματισμό αναφέρονται και ως **μέθοδοι**.

Προγραμματιστικά Περιβάλλοντα

Κάθε πρόγραμμα που γράφτηκε σε οποιαδήποτε γλώσσα προγραμματισμού πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές γλώσσας μηχανής. Η μετατροπή αυτή επιτυγχάνεται με τη χρήση ειδικών μεταφραστικών προγραμμάτων. Υπάρχουν δύο μεγάλες κατηγορίες τέτοιων προγραμμάτων, οι **μεταγλωττιστές** και οι **διερμηνευτές**.

- Ο **μεταγλωττιστής** δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής. Το τελευταίο μπορεί να εκτελείται οποτεδήποτε από τον υπολογιστή και είναι τελείως ανεξάρτητο από το αρχικό πρόγραμμα.
- Ο **διερμηνευτής** διαβάζει μία προς μία τις εντολές του αρχικού προγράμματος και για καθεμία εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής.

Λειτουργία Μεταγλώττισης

Για τη δημιουργία, τη μετάφραση και την εκτέλεση ενός προγράμματος απαιτούνται τουλάχιστον τρία προγράμματα: ο **συντάκτης**, ο **μεταγλωττιστής** και ο **συνδέτης**.

Για την αρχική σύνταξη των προγραμμάτων και τη διόρθωσή τους στη συνέχεια χρησιμοποιείται ένα ειδικό πρόγραμμα που ονομάζεται **συντάκτης**. Ο συντάκτης είναι ουσιαστικά ένας μικρός επεξεργαστής κειμένου, με δυνατότητες όμως που διευκολύνουν τη γρήγορη γραφή των εντολών των προγραμμάτων.

Το αρχικό πρόγραμμα λέγεται **πηγαίο πρόγραμμα**, ενώ το πρόγραμμα που παράγεται από το μεταγλωττιστή λέγεται **αντικείμενο πρόγραμμα**.

Το αντικείμενο πρόγραμμα είναι μεν σε μορφή κατανοητή από τον υπολογιστή, αλλά συνήθως δεν είναι σε θέση να εκτελεστεί. Χρειάζεται να συμπληρωθεί και να συνδεθεί με άλλα τμήματα προγράμματος απαραίτητα για την εκτέλεσή του, τμήματα που είτε τα γράφει ο προγραμματιστής είτε βρίσκονται στις βιβλιοθήκες της γλώσσας. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση ονομάζεται **συνδέτης – φορτωτής**. Το αποτέλεσμα του συνδέτη είναι η παραγωγή του **εκτελέσιμου προγράμματος**, το οποίο είναι το τελικό πρόγραμμα που εκτελείται από τον υπολογιστή. Για το λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση.

Μεταγλωττιστής		Διερμηνευτής	
Θετικά	**Αρνητικά**	**Θετικά**	**Αρνητικά**
Το εκτελέσιμο πρόγραμμα (τελικό) λειτουργεί χωρίς καθυστερήσεις (εκτελείται όπως θα το δει ο τελικός αποδέκτης).	**Αργεί η διαδικασία της μεταγλώττισης** (πρέπει να παραχθεί το τελικό πρόγραμμα για να ελεγχθεί για την ορθότητα του).	**Άμεση εκτέλεση του αλγορίθμου** και άμεση διόρθωση των λαθών.	**Το εκτελέσιμο πρόγραμμα (τελικό) λειτουργεί με καθυστερήσεις** (εκτελείται εντολή - εντολή).

Βασικές Έννοιες Προγραμματισμού

Το αλφάβητο της ΓΛΩΣΣΑΣ

Το αλφάβητο της ΓΛΩΣΣΑΣ αποτελείται από τα γράμματα του ελληνικού και του λατινικού αλφαβήτου, τα ψηφία, καθώς και από ειδικά σύμβολα, που χρησιμοποιούνται για προκαθορισμένες ενέργειες.

- **Γράμματα**

  Κεφαλαία ελληνικού αλφαβήτου (Α-Ω)

  Πεζά ελληνικού αλφαβήτου (α-ω)

  Κεφαλαία λατινικού αλφαβήτου (A-Z)

  Πεζά λατινικού αλφαβήτου (a-z)

- **Ψηφία**

  0-9

- **Ειδικοί χαρακτήρες**

  + - * / = . ( ) . , ' ! & κενός χαρακτήρας ^

Τύποι δεδομένων

Ακέραιοι	Π.χ.	1, 0, 100, -100
Πραγματικοί	Π.χ.	0.1, -0.1, 3.14
Χαρακτήρες	Π.χ.	'Κ', 'Κώστας', 'σήμερα είναι Τετάρτη', 'Τα πολλαπλάσια του 15 είναι:'
Λογικοί	Π.χ.	ΑΛΗΘΗΣ, ΨΕΥΔΗΣ

Προσοχή! Λογικές τιμές **δεν μπορούμε να τις διαβάσουμε από το χρήστη**! Ο χρήστης μέσω της εντολής *Διάβασε*, μπορεί να δώσει μόνο Αριθμούς και Χαρακτήρες!

Σταθερές

Οι σταθερές είναι προκαθορισμένες τιμές που **δεν μεταβάλλονται** κατά τη διάρκεια εκτέλεσης του προγράμματος. Οι σταθερές είναι αντίστοιχου τύπου δεδομένων, δηλαδή ακέραιες, πραγματικές, αλφαριθμητικές ή λογικές.

Παραδείγματα:

ΠΙ = 3.14159
ΦΠΑ = 0.19
ΟΝΟΜΑ = 'Κώστας'

Μεταβλητές

Μια μεταβλητή παριστάνει μία ποσότητα που **η τιμή της μπορεί να μεταβάλλεται**. Μπορούμε να παρομοιάσουμε τη μεταβλητή και την αντίστοιχη θέση μνήμης σαν ένα γραμματοκιβώτιο ή ενός κουτιού, το οποίο εξωτερικά έχει ως όνομα το **όνομα** της μεταβλητής και ως **περιεχόμενο** εσωτερικά την τιμή που έχει εκείνη τη συγκεκριμένη στιγμή η μεταβλητή. Η ΓΛΩΣΣΑ επιτρέπει τη χρήση μεταβλητών των τεσσάρων τύπων που αναφέρθηκαν, δηλαδή ακεραίων, πραγματικών, χαρακτήρων και λογικών, ενώ η δήλωση του τύπου κάθε μεταβλητής γίνεται υποχρεωτικά στο τμήμα δήλωσης μεταβλητών.

Παραδείγματα:

ΠΡΑΓΜΑΤΙΚΕΣ: Εμβαδόν, Α
ΑΚΕΡΑΙΕΣ: ΤΙΜΗ, Ν, Ι, j, k
ΧΑΡΑΚΤΗΡΕΣ: Όνομα
ΛΟΓΙΚΕΣ: Έλεγχος, Βρέθηκε

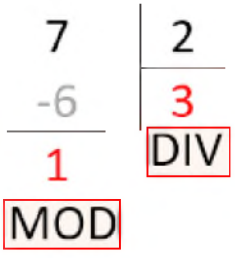
Λειτουργία και Βασικά Χαρακτηριστικά Σταθερών και Μεταβλητών

1. Το όνομα μιας σταθεράς ή μεταβλητής δεν αλλάζει.
2. Το όνομα μιας σταθεράς ή μεταβλητής είναι μοναδικό, δηλαδή δεν μπορεί να το έχει άλλη σταθερά ή μεταβλητή στο ίδιο πρόγραμμα.
3. Το είδος μιας σταθεράς ή μεταβλητής δεν αλλάζει.
4. Όταν αναθέτουμε νέα τιμή σε μια μεταβλητή, η παλιά τιμή παύει να υπάρχει.
5. Η δήλωση του τύπου κάθε μεταβλητής γίνεται υποχρεωτικά στο τμήμα δήλωσης μεταβλητών. Στο τμήμα δηλώσεων δεν δίνουμε αρχική τιμή στις μεταβλητές μας. Αυτό αν θέλουμε, το κάνουμε μετά την ΑΡΧΗ του προγράμματος.
6. Είναι καλή πρακτική να χρησιμοποιούνται ονόματα τα οποία να υπονοούν το περιεχόμενό τους.
7. **Όταν τελειώνουμε τη συγγραφή του προγράμματος μας**, ελέγχουμε το τμήμα δήλωσης των σταθερών - μεταβλητών για να μη ξεχάσουμε να δηλώσουμε κάποια από αυτές.

Τελεστές και Εκφράσεις

Αριθμητικοί Τελεστές

Τελεστής	Πράξη	Τελεστής	Πράξη
+	Πρόσθεση	DIV	Ακέραια διαίρεση
-	Αφαίρεση	MOD	Υπόλοιπο ακέραιας διαίρεσης
*	Πολλαπλασιασμός		
/	Διαίρεση		
^	Ύψωση σε δύναμη		



Σημειώσεις:

1. Οι τελεστές **MOD** και **DIV** δέχονται μόνο **ακέραιες** τιμές και παράγουν μόνο ακέραιες τιμές.
2. Το αποτέλεσμα μιας πράξης **διαίρεσης** είναι πάντα **πραγματικός αριθμός**, ακόμα και αν η διαίρεση είναι τέλεια και χωρίς δεκαδικά ψηφία.

Συγκριτικοί τελεστές

Τελεστής	Πράξη
=	Ίσο
<>	Διάφορο
>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο
<	Μικρότερο
<=	Μικρότερο ή ίσο

Σημείωση:

Εκτός από σύγκριση αριθμών **γίνεται σύγκριση και σε χαρακτήρες ή αλφαριθμητικά**. Στην περίπτωση αυτή το γράμμα που προηγείται στο αλφάβητο θεωρείται μικρότερο από το αυτό που έπεται.

π.χ. 'α' < 'β' είναι ΑΛΗΘΗΣ

Λογικοί τελεστές

Τελεστής	Πράξη
ΟΧΙ	Αντίθεση
ΚΑΙ	Σύζευξη
Ή	Διάζευξη

Ιεραρχία

Οι πράξεις σε μια λογική έκφραση εκτελούνται με την παρακάτω ιεραρχία:

1. ΟΧΙ
2. ΚΑΙ
3. Ή

Συναρτήσεις

Συμβολισμός	Υπολογισμός	Συμβολισμός	Υπολογισμός	Συμβολισμός	Υπολογισμός
ΗΜ(Χ)	Ημίτονο	**Τ_Ρ(Χ)**	Τετραγωνική Ρίζα	**Α_Μ(Χ)**	Ακέραιο Μέρος
ΣΥΝ(Χ)	Συνημίτονο	**ΛΟΓ(Χ)**	Φυσικός Λογάριθμος	**Α_Τ(Χ)**	Απόλυτη Τιμή
ΕΦ(Χ)	Εφαπτομένη	**Ε(Χ)**	e^x		

Σημειώσεις:

- Οι συναρτήσεις **ΗΜ()**, **ΣΥΝ()** και **ΕΦ()** δέχονται παράμετρο **σε μοίρες**.
- Το ακέραιο μέρος **Α_Μ()** ενός αριθμού **χ** ορίζεται, όπως στα μαθηματικά ο ακέραιος, με την ιδιότητα Α_Μ(χ) <= **χ** < Α_Μ(χ) + 1.
- Η απόλυτη τιμή **Α_Τ()** δέχεται αριθμητική παράμετρο, είτε ακέραιο αριθμό και επιστρέφει ακέραιο, είτε πραγματικό αριθμό και επιστρέφει πραγματικό.

Αριθμητικές Εκφράσεις

Οι αριθμητικές εκφράσεις υλοποιούν απλές ή σύνθετες μαθηματικές πράξεις. Κάθε έκφραση παριστάνει μια συγκεκριμένη αριθμητική τιμή, η οποία βρίσκεται **μετά την εκτέλεση των πράξεων**. Γι' αυτό είναι απαραίτητο **όλες οι μεταβλητές που εμφανίζονται** σε μια έκφραση **να έχουν οριστεί προηγούμενα**, δηλαδή να έχουν κάποια τιμή.

Ιεραρχία

Οι πράξεις σε μια αριθμητική έκφραση εκτελούνται με την παρακάτω ιεραρχία:

1. Ύψωση σε δύναμη (^)
2. Πολλαπλασιασμός (*) και διαίρεση (/, DIV, MOD)
3. Πρόσθεση (+) και αφαίρεση (-)

Σε περίπτωση που υπάρχει παρένθεση στην αριθμητική έκφραση, τότε εκτελούνται πρώτα οι πράξεις στη παρένθεση, σύμφωνα με την παραπάνω ιεραρχία. Πάντα πρέπει να χρησιμοποιούνται **ζεύγη** παρενθέσεων (όχι αγκύλες [ ]). Διαφορετικός αριθμός αριστερών από δεξιές παρενθέσεις στην ίδια έκφραση είναι ένα από τα πιο συνηθισμένα λάθη. Επίσης, δε χρησιμοποιούμε περιττές παρενθέσεις αν δε χρειάζονται!

Όταν έχουμε πράξεις της ίδιας ιεραρχίας, π.χ. πολλαπλασιασμό και διαίρεση, τότε οι πράξεις εκτελούνται διαδοχικά από τα αριστερά προς τα δεξιά (→).

Παραδείγματα:

Μαθηματικά	ΓΛΩΣΣΑ
a + 1	a + 1
$\frac{1}{2} \times \alpha^3$	1 / 2 * a ^ 3
$\frac{3x + 2y}{a - b}$	(3 * x + 2 * y) / (a - b)
2ημ(x)	2 * ΗΜ(x)

Λογικές Εκφράσεις

Για τη σύνταξη μιας **λογικής έκφρασης** ή **συνθήκης** χρησιμοποιούνται **σταθερές**, **μεταβλητές**, **αριθμητικές παραστάσεις**, **συγκριτικοί** και **λογικοί τελεστές**, καθώς και **παρενθέσεις**. Στις λογικές εκφράσεις γίνεται **σύγκριση της τιμής μίας έκφρασης**, που βρίσκεται αριστερά από το συγκριτικό τελεστή, **με την τιμή μιας άλλης έκφρασης**, που βρίσκεται δεξιά. Η σύγκριση λογικών μεταβλητών έχει έννοια μόνο στην περίπτωση του ίσου (=) και του διάφορου (<>), αφού οι τιμές που μπορούν να έχουν είναι ΑΛΗΘΗΣ και ΨΕΥΔΗΣ. Το αποτέλεσμα είναι μία λογική τιμή **ΑΛΗΘΗΣ** ή **ΨΕΥΔΗΣ**.

Ιεραρχία

Οι πράξεις σε μια λογική έκφραση εκτελούνται με την παρακάτω ιεραρχία:

1. Αριθμητικοί τελεστές
2. Συγκριτικοί τελεστές
3. Λογικοί τελεστές

Σε περίπτωση που υπάρχει παρένθεση στην αριθμητική έκφραση, τότε εκτελούνται πρώτα οι πράξεις στη παρένθεση, σύμφωνα με την παραπάνω ιεραρχία.

Όταν έχουμε πράξεις της ίδιας ιεραρχίας, π.χ. συγκριτικοί τελεστές, τότε οι πράξεις εκτελούνται διαδοχικά από τα αριστερά προς τα δεξιά (→).

Απλές και Σύνθετες Εκφράσεις

- Μια **απλή λογική έκφραση** περιέχει **έναν μόνο** συγκριτικό τελεστή.

  Π.χ. α > 0 ή Βρέθηκε = Ψευδής

- Οι **σύνθετες εκφράσεις** αποτελούνται από **δύο ή περισσότερες απλές**, οι οποίες συνδέονται μεταξύ τους με τη χρήση των τριών βασικών λογικών τελεστών **ΟΧΙ**, **ΚΑΙ**, **Ή**.

  Π.χ. α > 0 ΚΑΙ β < 4 ή Βρέθηκε = Ψευδής Ή i >= 10

Δομές Δεδομένων και Πίνακες

Πληροφορική και Δεδομένα

Πληροφορική θεωρείται η επιστήμη που μελετά τα δεδομένα από τις ακόλουθες σκοπιές:

- **Υλικού**. Το υλικό, δηλαδή η μηχανή, επιτρέπει στα δεδομένα ενός προγράμματος να αποθηκεύονται στην κύρια μνήμη και στις περιφερειακές συσκευές του υπολογιστή με διάφορες αναπαραστάσεις. Τέτοιες μορφές είναι η δυαδική, ο κώδικας ASCII, ο κώδικας EBCDIC, το συμπλήρωμα του 1 ή του 2 κ.λπ.
- **Γλωσσών προγραμματισμού**. Οι γλώσσες προγραμματισμού υψηλού επιτρέπουν τη χρήση διάφορων τύπων μεταβλητών για να περιγράψουν ένα δεδομένο. Ο μεταφραστής κάθε γλώσσας φροντίζει για την αποδοτικότερη μορφή αποθήκευσης, από πλευράς υλικού, κάθε μεταβλητής στον υπολογιστή.
- **Δομών Δεδομένων**. Δομή δεδομένων είναι ένα σύνολο δεδομένων μαζί με ένα σύνολο επιτρεπτών λειτουργιών επί αυτών. Για παράδειγμα, μία τέτοια δομή είναι η εγγραφή, που μπορεί να περιγράφει ένα είδος, ένα πρόσωπο κ.λπ. Η εγγραφή αποτελείται από τα πεδία που αποθηκεύουν χαρακτηριστικά διαφορετικού τύπου, όπως για παράδειγμα ο κωδικός, η περιγραφή κ.λπ. Άλλη μορφή δομής δεδομένων είναι το αρχείο που αποτελείται από ένα σύνολο εγγραφών. Μία επιτρεπτή λειτουργία σε ένα αρχείο είναι η σειριακή προσπέλαση όλων των εγγραφών του.
- **Ανάλυσης Δεδομένων**. Τρόποι καταγραφής και αλληλοσυσχέτισης των δεδομένων μελετώνται έτσι ώστε να αναπαρασταθεί η γνώση για πραγματικά γεγονότα. Οι τεχνολογίες των Βάσεων Δεδομένων, της Μοντελοποίησης Δεδομένων και της Αναπαράστασης Γνώσης ανήκουν σε αυτή τη σκοπιά μελέτης των δεδομένων.

Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

Δομή Δεδομένων είναι ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.

Στατικές και Δυναμικές Δομές Δεδομένων

Οι δομές δεδομένων διακρίνονται σε δύο μεγάλες κατηγορίες: τις στατικές και τις δυναμικές.

Στατικές δομές

Αποθηκεύονται σε συνεχόμενες θέσεις μνήμης και έχουν σταθερό μέγεθος, το οποίο καθορίζεται στην αρχή του προγράμματος. Οι στατικές δομές υλοποιούνται με **πίνακες**. Το προγραμματιστικό περιβάλλον ΓΛΩΣΣΑ, υποστηρίζει μόνο στατικές δομές. Ως εκ τούτου, η δομή του πίνακα αντιμετωπίζεται ως στατική και για να χρησιμοποιηθεί ένας πίνακας θα πρέπει να έχει πρώτα δηλωθεί, τόσο ο πίνακας, όσο και το μέγεθός του. Επιπλέον, οι δομές **ουρά** και **στοίβα** θεωρούνται επίσης στατικές δομές για τη ΓΛΩΣΣΑ, και στο πλαίσιο του μαθήματος υλοποιούνται με τη χρήση μονοδιάστατων πινάκων.

Δυναμικές δομές

Δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης αλλά στηρίζονται στην τεχνική της λεγόμενης δυναμικής παραχώρησης μνήμης. Με άλλα λόγια, οι δομές αυτές δεν έχουν σταθερό μέγεθος, αλλά ο αριθμός των κόμβων τους μεγαλώνει και μικραίνει καθώς στη δομή εισάγονται νέα δεδομένα ή διαγράφονται κάποια δεδομένα αντίστοιχα. Το μέγεθος της μνήμης καθορίζεται κατά την εκτέλεση του προγράμματος. Με δυναμικές δομές υλοποιούνται οι **λίστες**, τα **δένδρα** και οι **γράφοι**.

Πέρα από την διαφορά τους στον τρόπο αποθήκευσης στην κύρια μνήμη, θα πρέπει να γίνει κατανοητό ότι για τις στατικές δομές χρειάζεται να ορισθεί το μέγεθός τους, πριν από την έναρξη του προγράμματος, στο τμήμα δηλώσεων. Αντίθετα, για τις δυναμικές δομές το μέγεθός τους δύναται να ορισθεί και να τροποποιηθεί κατά την εκτέλεση του προγράμματος.

Μια δομή δεδομένων δεν είναι εγγενώς στατική ή δυναμική, αλλά εξαρτάται από τις δυνατότητες της γλώσσας προγραμματισμού που χρησιμοποιούμε και από τον τρόπο υλοποίησης της δομής στη γλώσσα αυτή. Δεν υποστηρίζουν όλες οι γλώσσες προγραμματισμού όλες τις δομές δεδομένων.

Βασικές Λειτουργίες Δομών Δεδομένων

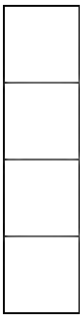
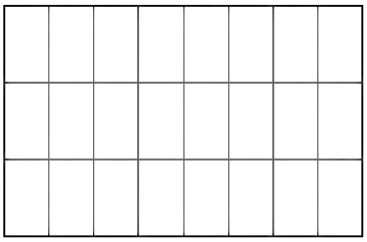
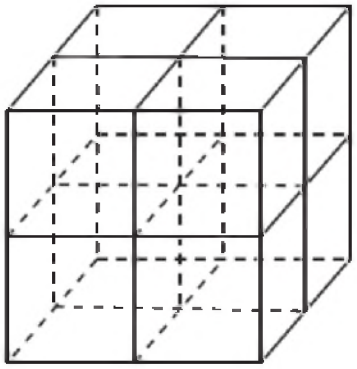
Οι βασικές **λειτουργίες** (ή αλλιώς **πράξεις**) επί των δομών δεδομένων είναι οι ακόλουθες:

- **Προσπέλαση**, πρόσβαση σε έναν κόμβο με σκοπό να εξετασθεί ή να τροποποιηθεί το περιεχόμενό του.
- **Εισαγωγή**, δηλαδή η προσθήκη νέων κόμβων σε μία υπάρχουσα δομή.
- **Διαγραφή**, που αποτελεί το αντίστροφο της εισαγωγής, δηλαδή ένας κόμβος αφαιρείται από μία δομή.
- **Αναζήτηση**, κατά την οποία προσπελαύνονται οι κόμβοι μιας δομής, προκειμένου να εντοπιστούν ένας ή περισσότεροι που έχουν μια δεδομένη ιδιότητα.
- **Ταξινόμηση**, όπου οι κόμβοι μιας δομής διατάσσονται κατά αύξουσα ή φθίνουσα σειρά.
- **Αντιγραφή**, κατά την οποία όλοι οι κόμβοι ή μερικοί από τους κόμβους μίας δομής αντιγράφονται σε μία άλλη δομή.
- **Συγχώνευση**, κατά την οποία δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή.
- **Διαχωρισμός**, που αποτελεί την αντίστροφη πράξη της συγχώνευσης.

Πίνακας

Πίνακας είναι ένα σύνολο αντικειμένων **ίδιου τύπου**, τα οποία αναφέρονται με ένα **κοινό όνομα**. Κάθε ένα από τα αντικείμενα που απαρτίζουν τον πίνακα λέγεται **στοιχείο του πίνακα**. Η αναφορά σε ατομικά στοιχεία του πίνακα γίνεται με το όνομα του πίνακα ακολουθούμενο από ένα **δείκτη**.

Διαστάσεις Πινάκων

Μονοδιάστατος	Δισδιάστατος	Ν-Διάστατος
A[4]	A[3,8]	A[x,y,z, ..., n]
		

Τετραγωνικοί Πίνακες

Κύρια Διαγώνιος	Δευτερεύουσα Διαγώνιος	Τριγωνικός	Συμμετρικός
Η Κύρια Διαγώνιος αποτελείται από τα στοιχεία A[i, j] του πίνακα για τα οποία ισχύει i = j.	Η Δευτερεύουσα Διαγώνιος ή Αντιδιαγώνιος, αποτελείται από τα στοιχεία A[i, j] του πίνακα διαστάσεων NxN για τα οποία ισχύει i + j = N + 1.	Τριγωνικός ονομάζεται ο πίνακας που περιέχει στοιχεία **μόνο** κάτω ή πάνω από τη κύρια διαγώνιό του. Η διαγώνιος είναι κενή ή μηδενική.	Συμμετρικός ονομάζεται ο πίνακας που περιέχει στοιχεία για τα οποία ισχύει A[i, j] = A[j, i].

A[4,4]

12	10	19	47
11	26	44	58
78	42	56	23
61	72	91	84

A[4,4]

12	10	19	47
11	26	44	58
78	42	56	23
61	72	91	84

A[4,4]

	10	19	47
11		44	58
78	42		23
61	72	91	

A[4,4]

12	10	19	47
10	26	44	72
19	44	56	23
47	72	23	84

Παράλληλοι Πίνακες

Δύο ή περισσότεροι πίνακες λέγονται **παράλληλοι**, αν σε αυτούς έχουμε αποθηκεύσει τα χαρακτηριστικά οντοτήτων με τέτοιο τρόπο ώστε τα δεδομένα κάθε οντότητας να βρίσκονται σε στοιχεία με την ίδια τιμή δείκτη.

Πλεονεκτήματα – Μειονεκτήματα Πινάκων

Πλεονεκτήματα

1. Οι πίνακες χρησιμοποιούνται στην περίπτωση που χρειάζεται να διαχειριστούμε **πολλά δεδομένα του ίδιου τύπου** και τα οποία απαιτείται να είναι αποθηκευμένα για μεταγενέστερη χρήση τους.
2. Ο πίνακας θεωρείται μια **δομή τυχαίας προσπέλασης**, σε αντίθεση με άλλες δομές δεδομένων που είναι δομές ακολουθιακής ή σειριακής προσπέλασης. Για να φθάσουμε για παράδειγμα σ' έναν κόμβο μιας λίστας, πρέπει να περάσουμε από όλους τους προηγούμενους, ξεκινώντας από τον πρώτο. Σε έναν πίνακα έχουμε άμεση πρόσβαση σε οποιοδήποτε στοιχείο του, αρκεί να γνωρίζουμε τη θέση του.

Μειονεκτήματα

1. Οι πίνακες, από την αρχή του προγράμματος, **δεσμεύουν συγκεκριμένο πλήθος θέσεων μνήμης**, ανάλογα με το μέγεθός τους.
2. Περιορίζουν τις δυνατότητες του προγράμματος, καθώς **το μέγεθός τους είναι προκαθορισμένο (δε μεταβάλλεται)**.

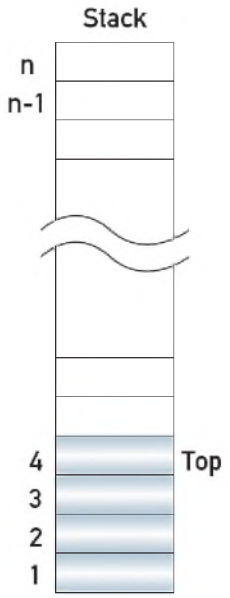
Τυπικές Επεξεργασίες Πινάκων

Οι τυπικές αυτές επεξεργασίες είναι:

- Υπολογισμός **αθροισμάτων** στοιχείων του πίνακα.
- Εύρεση του **μέγιστου** ή του **ελάχιστου** στοιχείου.
- **Ταξινόμηση** των στοιχείων του πίνακα.
- **Αναζήτηση** ενός στοιχείου του πίνακα.
  - **Σειριακή** ή **Γραμμική**
  - **Δυαδική** (Μόνο σε ταξινομημένο πίνακα)
- **Συγχώνευση** δύο πινάκων.
  - **Χωρίς Ταξινόμηση**
  - **Με ταξινόμηση** (Μόνο σε ταξινομημένους πίνακες)

Στοίβα

Στοίβα, ονομάζεται μια δομή δεδομένων το σύνολο των στοιχείων της οποίας είναι διατεταγμένο με τέτοιο τρόπο, ώστε τα στοιχεία που βρίσκονται στην κορυφή της στοίβας λαμβάνονται πρώτα, ενώ αυτά που βρίσκονται στο βάθος της στοίβας λαμβάνονται τελευταία.



Δύο είναι οι κύριες λειτουργίες σε μία στοίβα:

- η **ώθηση** στοιχείου στην κορυφή της στοίβας, και
- η **απώθηση** στοιχείου από τη στοίβα.

Η διαδικασία της ώθησης πρέπει οπωσδήποτε να ελέγχει, αν η στοίβα είναι γεμάτη, οπότε λέγεται ότι συμβαίνει **υπερχείλιση** της στοίβας. Αντίστοιχα, η διαδικασία απώθησης ελέγχει, αν υπάρχει ένα τουλάχιστον στοιχείο στη στοίβα, δηλαδή ελέγχει αν γίνεται **υποχείλιση** της στοίβας.

Μια στοίβα μπορεί να υλοποιηθεί πολύ εύκολα με τη βοήθεια ενός μονοδιάστατου πίνακα, χρησιμοποιώντας μια βοηθητική μεταβλητή (με όνομα συνήθως **top**) που χρησιμοποιείται για να δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας.

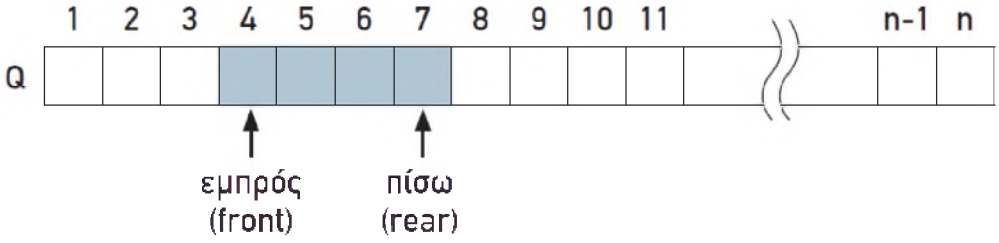
Ουρά

Ουρά, ονομάζεται μια δομή δεδομένων το σύνολο των στοιχείων της οποίας είναι διατεταγμένο με τέτοιο τρόπο, ώστε τα στοιχεία που τοποθετήθηκαν πρώτα στην ουρά να λαμβάνονται επίσης πρώτα.

Δύο είναι οι κύριες λειτουργίες σε μία ουρά:

- η **εισαγωγή** στοιχείου στο πίσω άκρο της ουράς, και
- η **εξαγωγή** στοιχείου από το εμπρός άκρο της ουράς.

Άρα, σε αντίθεση με τη δομή της στοίβας, στην περίπτωση της ουράς απαιτούνται δύο δείκτες: ο εμπρός (**front**) και ο πίσω (**rear**) δείκτης, που μας δίνουν τη θέση του στοιχείου που σε πρώτη ευκαιρία θα εξαχθεί και τη θέση του στοιχείου που μόλις εισήλθε.



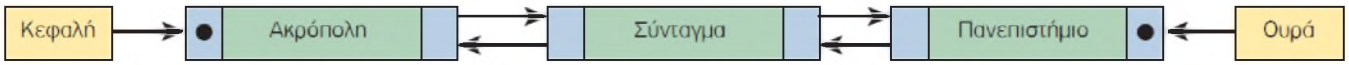
Λίστα

Η λίστα δεν είναι τίποτα άλλο παρά μία **συλλογή από αντικείμενα του ίδιου τύπου**. Μπορούμε να έχουμε δηλαδή λίστες από **λέξεις**, από **ονόματα** αλλά και από **αριθμούς**.

Μία (απλά) **συνδεδεμένη λίστα** είναι ένα **σύνολο κόμβων** διατεταγμένων γραμμικά (ο ένας μετά τον άλλο). Κάθε κόμβος περιέχει εκτός από τα **δεδομένα** του και έναν **δείκτη** που δείχνει προς τον επόμενο κόμβο. Ο δείκτης του τελευταίου κόμβου δε δείχνει σε κάποιον κόμβο (δείκτης στο κενό). Για να το δηλώσουμε αυτό λέμε ότι το πεδίο δείκτη του τελευταίου κόμβου έχει την τιμή **NULL**. Για να προσπελάσουμε τους κόμβους της λίστας χρειάζεται να γνωρίζουμε τη διεύθυνση (θέση στη μνήμη) του πρώτου κόμβου της λίστας. Η διεύθυνση αυτή αποθηκεύεται σε μία ειδική μεταβλητή που την ονομάζουμε συνήθως **Κεφαλή (Head)**.

- Απλά συνδεδεμένη λίστα

- Διπλά συνδεδεμένη λίστα



Διαφορές Λίστας σε σχέση με τον Πίνακα - Πλεονεκτήματα – Μειονεκτήματα

1. Ο πίνακας θεωρείται μια δομή **τυχαίας προσπέλασης**, σε αντίθεση με μια λίστα που είναι στην ουσία μια δομή ακολουθιακής ή **σειριακής προσπέλασης**. Για να φθάσουμε, δηλαδή, σ' έναν κόμβο μιας λίστας πρέπει να περάσουμε από όλους τους προηγούμενους ξεκινώντας από τον πρώτο.
2. Ο πίνακας έχει **σταθερό μέγεθος**, το οποίο δηλώνεται εξαρχής κατά την υλοποίηση. Αυτό γίνεται, διότι ο πίνακας είναι στατική δομή δεδομένων σε αντίθεση με τη λίστα που είναι **δυναμική δομή** και το μέγεθός της μπορεί να μεταβάλλεται καθώς εισέρχονται νέοι κόμβοι στη λίστα ή διαγράφονται κάποιοι άλλοι.
3. Οι κόμβοι της λίστας αποθηκεύονται σε **μη συνεχόμενες θέσεις μνήμης** σε αντιδιαστολή με τους πίνακες, όπου τα στοιχεία αποθηκεύονται σε **συνεχόμενες θέσεις μνήμης**.

Στα **πλεονεκτήματα των λιστών** (έναντι των πινάκων) συγκαταλέγονται τα εξής:

1. Το δυναμικό τους **μέγεθος**.
2. Η ευκολία **εισαγωγής** και **διαγραφής** από οποιοδήποτε μέρος της λίστας.
3. Η **μη αναγκαιότητα δήλωσης** του μεγέθους τους.

Στα **μειονεκτήματα των λιστών** (έναντι των πινάκων) περιλαμβάνονται τα εξής:

1. Η τυχαία πρόσβαση στη λίστα δεν επιτρέπεται, διότι η **προσπέλαση γίνεται σειριακά**. Είναι αδύνατο να φτάσετε στον n-οστό κόμβο μιας απλά συνδεδεμένης λίστας χωρίς πρώτα να περάσετε από όλους τους κόμβους διαδοχικά μέχρι τον συγκεκριμένο κόμβο ξεκινώντας από τον πρώτο κόμβο. Εναλλακτικά, στην περίπτωση της διπλά συνδεμένης λίστας μπορείτε να ξεκινήσετε και από τον τελευταίο κόμβο. Επομένως, δεν μπορούμε να πραγματοποιήσουμε με αποτελεσματικό τρόπο δυαδική αναζήτηση σε συνδεδεμένες λίστες.
2. Οι συνδεδεμένες λίστες έχουν πολύ μεγαλύτερη επιβάρυνση (όσον αφορά το **μέγεθος κάθε κόμβου**) από τους πίνακες, αφού οι συνδεδεμένοι κόμβοι της λίστας είναι δυναμικά κατανεμημένοι (οι οποίοι είναι λιγότερο αποτελεσματικοί στη χρήση της μνήμης) και κάθε κόμβος στη λίστα πρέπει, επιπλέον, να αποθηκεύσει έναν πρόσθετο δείκτη που θα δείχνει στον επόμενο κόμβο. Στην περίπτωση των διπλά συνδεδεμένων λιστών χρειαζόμαστε επιπλέον έναν δεύτερο δείκτη που θα δείχνει στον προηγούμενο κόμβο.

Βασικές πράξεις των συνδεδεμένων λιστών

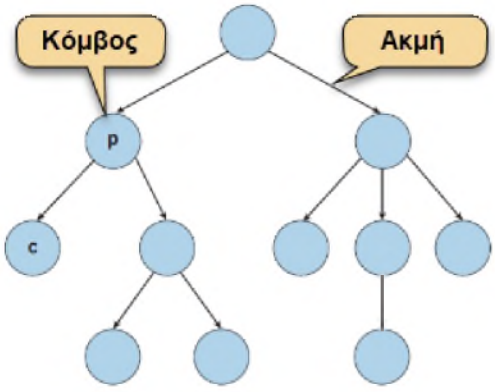
Οι βασικές πράξεις των συνδεδεμένων λιστών είναι οι παρακάτω:

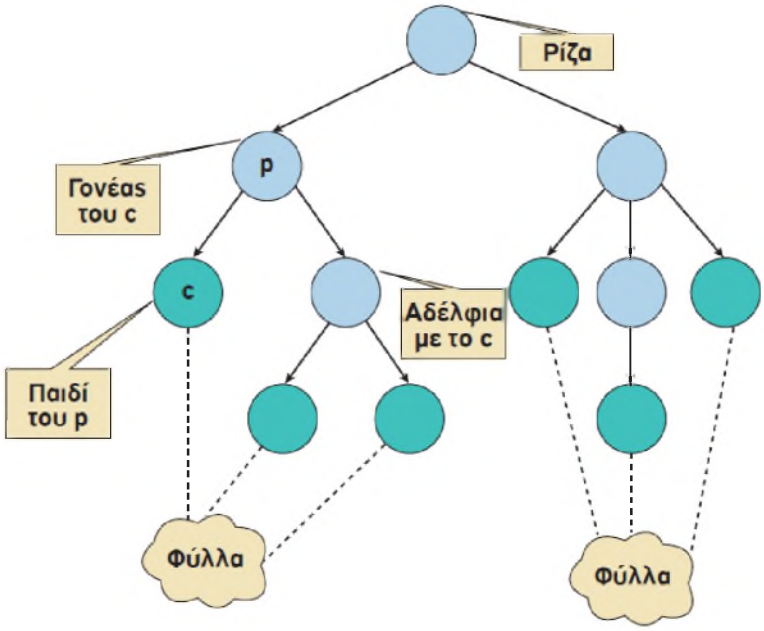
1. **Εισαγωγή** κόμβου στη λίστα (εισαγωγή κόμβου στην αρχή, στο τέλος της λίστας ή ενδιάμεσα).
2. **Διαγραφή** κόμβου από τη λίστα (διαγραφή από την αρχή, το τέλος της λίστας ή ενδιάμεσα).
3. **Έλεγχος** για το αν η λίστα είναι **κενή**.
4. **Αναζήτηση** κόμβου για την εύρεση συγκεκριμένου στοιχείου.
5. **Διάσχιση της λίστας** και **προσπέλαση** των στοιχείων της (π.χ. εκτύπωση των δεδομένων που περιέχονται σε όλους τους κόμβους της λίστας).

Δένδρα

Ένα **δένδρο** είναι μία δομή που αποτελείται από ένα σύνολο **κόμβων** και ένα σύνολο **ακμών μεταξύ των κόμβων** με βάση τους εξής κανόνες:

- Υπάρχει ένας ξεχωριστός κόμβος που ονομάζεται **ρίζα**. Αυτός είναι ένας κόμβος χωρίς γονέα.
- Για κάθε κόμβο c, εκτός από τη ρίζα, υπάρχει **μόνο μια ακμή** που καταλήγει στον κόμβο αυτόν ξεκινώντας από κάποιον άλλον κόμβο p. Ο κόμβος p ονομάζεται **γονέας** του c και ο κόμβος c **παιδί** του p.
- Για κάθε κόμβο υπάρχει μία **μοναδική διαδρομή**, δηλαδή, μια ακολουθία διαδοχικών ακμών, που ξεκινάει από τη ρίζα και τερματίζει σε αυτόν τον κόμβο.





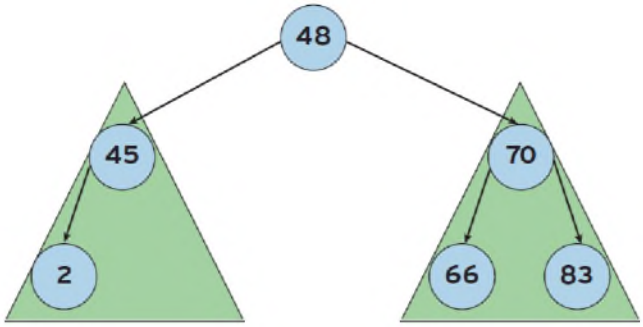
Δένδρο θεωρούμε και το **κενό δένδρο**, δηλαδή το δένδρο που δεν έχει ούτε κόμβους, ούτε ακμές. Το κενό δένδρο είναι το μόνο δένδρο χωρίς ρίζα.

Μπορούμε να έχουμε ένα **απλό δένδρο**, το οποίο να απαρτίζεται από **έναν μόνο κόμβο**. Αυτός ο κόμβος είναι και ρίζα του απλού αυτού δένδρου, διότι δεν έχει γονέα και φύλλο, και διότι δεν έχει παιδιά.

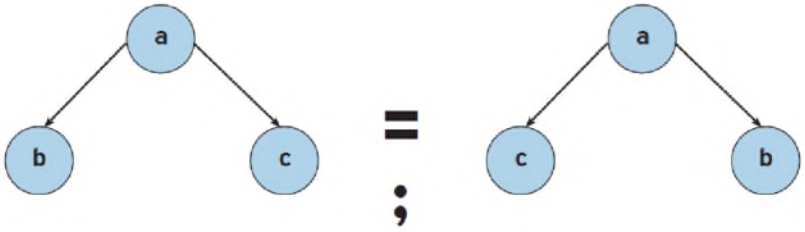
Απλό Δένδρο	Κενό Δένδρο
	

Υποδένδρα

Μέσα σε ένα δένδρο μπορούμε να εντοπίσουμε και άλλα μικρότερα δένδρα, που ονομάζονται υποδένδρα. Πιο συγκεκριμένα, κάθε κόμβος ενός δένδρου μπορεί να θεωρηθεί ως ρίζα ενός υποδένδρου, δηλαδή ενός άλλου μικρότερου δένδρου, που ξεκινάει από τον κόμβο αυτόν. Σύμφωνα με τα προηγούμενα, ένα δένδρο **n** κόμβων περιέχει **n-1** υποδένδρα.



Ισότητα Δένδρων και Διατεταγμένα Δένδρα



Έστω ότι θέλουμε να εξετάσουμε αν τα δύο παραπάνω δένδρα είναι **ίδια (ίσα)** ή **όχι**. Αν για κάθε κόμβο ενός δένδρου υπάρχει μία **γραμμική σχέση μεταξύ των παιδιών του κόμβου** αυτού (για παράδειγμα το παιδί που εισέρχεται πρώτο μπαίνει στα αριστερά του γονέα και δεξιά αυτού του παιδιού εισέρχονται τα υπόλοιπα αδέρφια, ανάλογα με το χρόνο εισαγωγής τους), τότε αναφερόμαστε σε ένα **διατεταγμένο δένδρο**. Σε αυτή την περίπτωση, τα παραπάνω δένδρα **δεν είναι ίσα**!

Όμως, επειδή κατά βάση τα δένδρα ως δομή δεδομένων θεωρούνται μη-γραμμικά και ευέλικτα, και εφόσον δεν τίθεται θέμα διάταξης, τα παραπάνω δένδρα **είναι ίσα**!

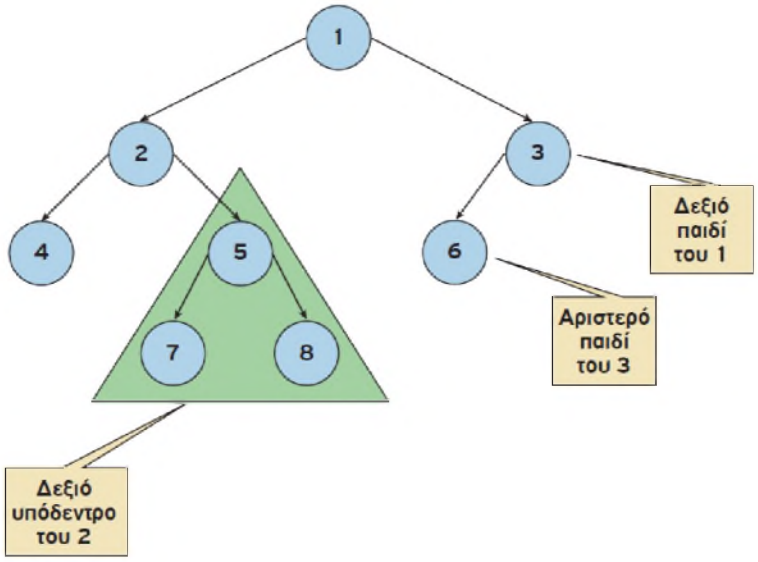
Σπουδαιότητα των Δένδρων

Υπάρχουν δύο λόγοι για τους οποίους τα δένδρα είναι τόσο ισχυρά.

- Ο πρώτος λόγος αναφέρεται στη **δυναμικότητα των δένδρων**. Είναι πολύ εύκολο να **προστεθεί**, να **αφαιρεθεί** ή να **αναζητηθεί** ένα στοιχείο σε ένα δένδρο, όπως θα δούμε στη συνέχεια.
- Ο δεύτερος βασικός λόγος είναι ότι **η δομή των δένδρων μεταφέρει πληροφορίες**. Δηλαδή, δεν είναι μόνο το περιεχόμενο των κόμβων που έχει σημασία, αλλά και ο τρόπος με τον οποίο αυτοί οι κόμβοι είναι διατεταγμένοι μεταξύ τους. Η σπουδαιότητα της δομής και οι πληροφορίες που μεταφέρει, θα γίνει περισσότερο κατανοητή με τη παρουσίαση των παρακάτω αναπαραστάσεων.

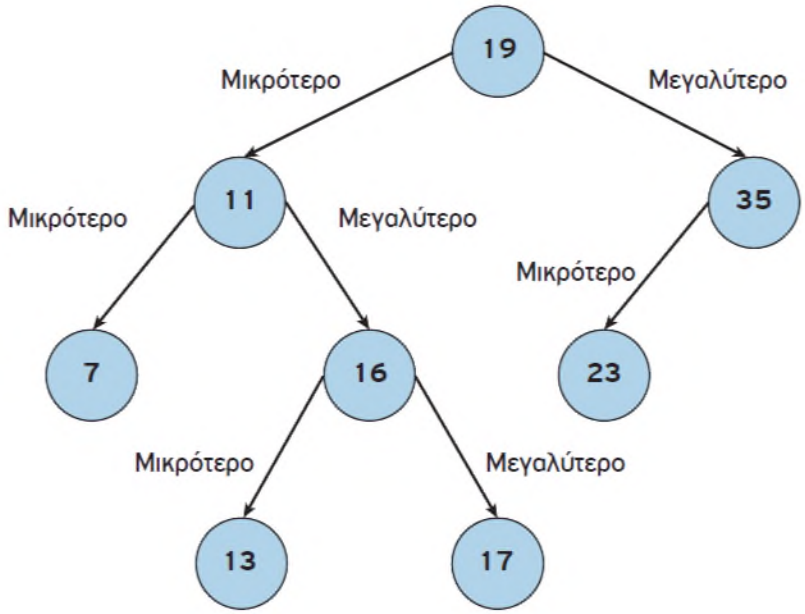
Δυαδικά Δένδρα

Ένα **δυαδικό δένδρο (binary tree)** είναι ένα **διατεταγμένο δένδρο**, στο οποίο κάθε κόμβος έχει **το πολύ δύο παιδιά**, το **αριστερό** και το **δεξί** παιδί. Μπορούμε, συνεπώς, να μιλάμε για αριστερό και δεξιό υποδένδρο ενός κόμβου.



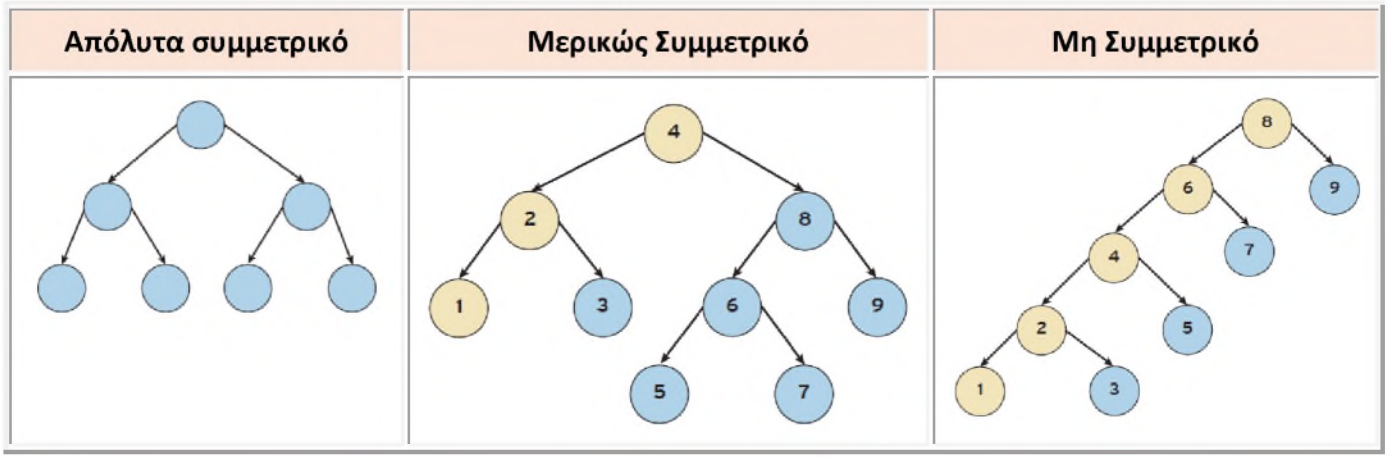
Δυαδικά Δένδρα Αναζήτησης

Ένα **δυαδικό δένδρο αναζήτησης** είναι ένα δυαδικό δένδρο, όπου για κάθε κόμβο **u**, όλοι οι κόμβοι του **αριστερού** υποδένδρου έχουν τιμές **μικρότερες** της τιμής του κόμβου **u** και όλοι οι κόμβοι του **δεξιού** υποδένδρου έχουν τιμές **μεγαλύτερες (ή ίσες)** της τιμής του κόμβου **u**. Για λόγους απλούστευσης θεωρούμε ότι δεν υπάρχουν τιμές ίσες με την τιμή του κόμβου u.



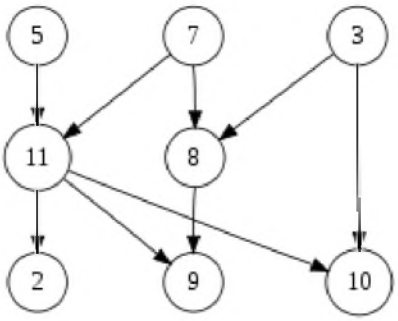
Ισορροπημένο Δυαδικό Δένδρο Αναζήτησης

Αν θέλουμε να έχουμε **γρήγορους αλγόριθμους αναζήτησης** πρέπει να αποθηκεύουμε τις τιμές στα δυαδικά δένδρα αναζήτησης με έναν συγκεκριμένο τρόπο, ώστε τα δένδρα που δημιουργούνται να έχουν συμμετρία. Έτσι δημιουργούνται τα **ισορροπημένα δένδρα αναζήτησης**.



Γράφοι

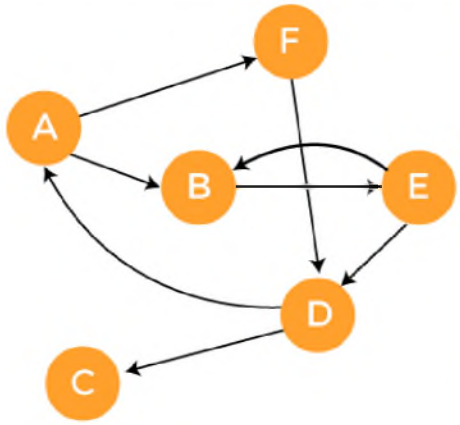
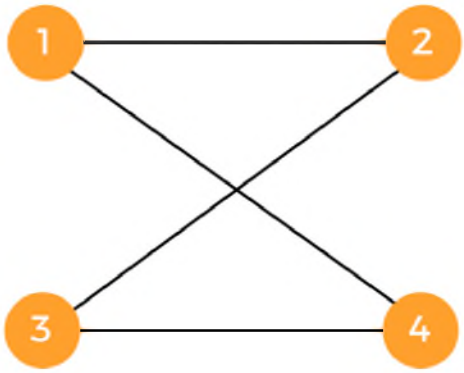
Ένας **γράφος (graph)** είναι μία δομή που αποτελείται από ένα σύνολο **κόμβων** (ή σημείων ή κορυφών) και ένα σύνολο **γραμμών** (ή ακμών ή τόξων) που ενώνουν μερικούς ή όλους τους κόμβους. Ο γράφος αποτελεί την πιο **γενική δομή δεδομένων**, με την έννοια ότι όλες οι προηγούμενες δομές που παρουσιάστηκαν μπορούν να θεωρηθούν περιπτώσεις γράφων.



Τύποι Γράφων

Ανάλογα με τον τρόπο λειτουργίας των ακμών ενός γράφου, διαμορφώνονται δυο διαφορετικοί τύποι γράφων, ο **κατευθυνόμενος** γράφος και η **μη κατευθυνόμενος**. Αυτοί οι τύποι ορίζονται ως εξής:

- Εάν **όλες οι ακμές** σε έναν γράφο **έχουν κατεύθυνση**, ο γράφος ονομάζεται **κατευθυνόμενος γράφος**.
- Εάν **όλες οι ακμές** σε έναν γράφο **δεν έχουν κατεύθυνση**, ο γράφος ονομάζεται **μη κατευθυνόμενος γράφος**.

Κατευθυνόμενος Γράφος	Μη Κατευθυνόμενος Γράφος
	

Τμηματικός Προγραμματισμός

Τμηματικός προγραμματισμός ονομάζεται η **τεχνική** σχεδίασης και ανάπτυξης των προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων.

Τα υποπρογράμματα χρησιμοποιούνται στην περίπτωση όπου έχουμε να επιλύσουμε ένα **σύνθετο** και **πολύπλοκο** πρόβλημα με τη χρήση υπολογιστή. Ο καλύτερος τρόπος για την επίλυση του σύνθετου προβλήματος είναι να διαιρεθεί σε μικρότερα προβλήματα και αυτά σε άλλα μικρότερα κ.ο.κ. (**ιεραρχική σχεδίαση**) και να σχεδιάσουμε υποπρογράμματα (απλά μικρά προγράμματα) για τα μικρότερα προβλήματα. Με τον τρόπο αυτό είναι δυνατό να αντιμετωπισθεί πιο εύκολα η επίλυση του αρχικού σύνθετου προβλήματος.

Αρχικό πρόγραμμα ονομάζεται το ολικό αδόμητο πρόβλημα πριν αυτό διασπαστεί σε υποπρογράμματα.

Υποπρόγραμμα ονομάζεται ένα τμήμα προγράμματος που επιτελεί ένα αυτόνομο έργο και έχει γραφτεί χωριστά από το υπόλοιπο πρόγραμμα.

Κύριο πρόγραμμα ονομάζεται το πρόγραμμα από το οποίο ξεκινά (ή καλείται) η εκτέλεση των υποπρογραμμάτων.

Χαρακτηριστικά των υποπρογραμμάτων

Υπάρχουν τρεις βασικές ιδιότητες που πρέπει να διακρίνουν τα υποπρογράμματα:

- **Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο**. Στην πραγματικότητα κάθε υποπρόγραμμα ενεργοποιείται με την είσοδο σε αυτό που γίνεται πάντοτε από την αρχή του, εκτελεί ορισμένες ενέργειες, και απενεργοποιείται με την έξοδο από αυτό που γίνεται πάντοτε από το τέλος του.
- **Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα**. Αυτό σημαίνει ότι κάθε υποπρόγραμμα μπορεί να σχεδιαστεί, να αναπτυχθεί και να συντηρηθεί αυτόνομα, χωρίς να επηρεαστούν άλλα υποπρογράμματα. Στην πράξη βέβαια η απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί.
- **Κάθε υποπρόγραμμα πρέπει να μην είναι πολύ μεγάλο**. Η έννοια του μεγάλου προγράμματος είναι υποκειμενική, αλλά πρέπει κάθε υποπρόγραμμα να είναι τόσο, ώστε να είναι εύκολα κατανοητό για να μπορεί να ελέγχεται. Γενικά κάθε υποπρόγραμμα πρέπει να εκτελεί μόνο μία λειτουργία. Αν εκτελεί περισσότερες λειτουργίες, τότε συνήθως μπορεί και πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

Πλεονεκτήματα του Τμηματικού Προγραμματισμού

Η σωστή χρήση του τμηματικού προγραμματισμού εξασφαλίζει τέσσερα βασικά χαρακτηριστικά ενός αποδοτικού και αποτελεσματικού προγράμματος:

- **Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος**. Επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι στην αντιμετώπιση του συνολικού προβλήματος. Με τη σταδιακή επίλυση των υποπροβλημάτων και τη δημιουργία των αντιστοίχων υποπρογραμμάτων τελικά επιλύεται το συνολικό πρόβλημα.
- **Διευκολύνει την κατανόηση και διόρθωση του προγράμματος**. Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση ενός συγκεκριμένου τμήματός του, χωρίς οι αλλαγές αυτές να επηρεάσουν όλο το υπόλοιπο πρόγραμμα. Επίσης διευκολύνει οποιονδήποτε χρειαστεί να διαβάσει και να κατανοήσει τον τρόπο που λειτουργεί το πρόγραμμα. Όπως έχει πολλές φορές τονιστεί αυτό είναι πολύ σημαντικό χαρακτηριστικό του σωστού προγραμματισμού, αφού ένα μεγάλο πρόγραμμα στον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.
- **Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος**. Πολύ συχνά χρειάζεται η ίδια λειτουργία σε διαφορετικά σημεία ενός προγράμματος. Από τη στιγμή που ένα υποπρόγραμμα έχει γραφεί, μπορεί το ίδιο να καλείται από πολλά σημεία του προγράμματος. Έτσι μειώνονται το μέγεθος του προγράμματος, ο χρόνος που απαιτείται για τη συγγραφή του και οι πιθανότητες λάθους, ενώ ταυτόχρονα το πρόγραμμα γίνεται πιο εύληπτο και κατανοητό.
- **Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού**. Ένα υποπρόγραμμα που έχει γραφεί μπορεί να χρησιμοποιηθεί πολύ εύκολα και σε άλλα προγράμματα. Από τη στιγμή που έχει δημιουργηθεί, η χρήση του δεν διαφέρει από τη χρήση των ενσωματωμένων συναρτήσεων που παρέχει η γλώσσα προγραμματισμού, όπως για τον υπολογισμό του ημίτονου ή του συνημίτονου ή την εντολή με την οποία εκτελεί μία συγκεκριμένη διαδικασία, για παράδειγμα γράφει στην οθόνη (εντολή ΓΡΑΨΕ). Αν λοιπόν χρειάζεται συχνά κάποια λειτουργία που δεν υποστηρίζεται απευθείας από τη γλώσσα, όπως για παράδειγμα η εύρεση του μικρότερου δύο αριθμών, τότε μπορεί να γραφεί το αντίστοιχο υποπρόγραμμα. Η συγγραφή πολλών υποπρογραμμάτων και η δημιουργία βιβλιοθηκών με αυτά, ουσιαστικά επεκτείνουν την ίδια τη γλώσσα προγραμματισμού.

Παράμετροι

Μία **παράμετρος** είναι μία **μεταβλητή** που επιτρέπει το πέρασμα της τιμής της από ένα τμήμα προγράμματος σε ένα άλλο.

Κάθε υποπρόγραμμα για να ενεργοποιηθεί **καλείται**, όπως λέγεται, από ένα **άλλο υποπρόγραμμα** ή **το αρχικό πρόγραμμα** (κύριο πρόγραμμα). Το υποπρόγραμμα είναι αυτόνομο και ανεξάρτητο τμήμα προγράμματος, αλλά συχνά πρέπει να επικοινωνεί με το υπόλοιπο πρόγραμμα. Συνήθως δέχεται τιμές από το τμήμα προγράμματος που το καλεί και μετά την εκτέλεση επιστρέφει σε αυτό νέες τιμές, αποτελέσματα. Οι τιμές αυτές που περνούν από το ένα υποπρόγραμμα στο άλλο λέγονται **παράμετροι**.

Οι παράμετροι λοιπόν είναι σαν τις κοινές μεταβλητές ενός προγράμματος με μία ουσιώδη διαφορά, χρησιμοποιούνται για να περνούν τιμές στα υποπρογράμματα.

Πραγματικές παράμετροι

Οι πραγματικές παράμετροι είναι οι μεταβλητές του Κύριου Προγράμματος που περνούν τις τιμές τους στις αντίστοιχες τυπικές παραμέτρους της διαδικασίας.

Η λίστα των πραγματικών παραμέτρων καθορίζει τις παραμέτρους στην κλήση του υποπρογράμματος. Οι πραγματικές παράμετροι ονομάζονται και ως απλά **παράμετροι**.

Τυπικές παράμετροι

Οι τυπικές παράμετροι είναι οι μεταβλητές της διαδικασίας που λαμβάνουν τιμή από τις πραγματικές παραμέτρους του προγράμματος, κατά την κλήση της διαδικασίας. Η διαδικασία ορίζει αυτές τις μεταβλητές, χρησιμοποιεί τις τιμές τους κατά τη διάρκεια εκτέλεσης των εντολών της και στον τερματισμό της, επιστρέφει τις τιμές αυτών των παραμέτρων στις αντίστοιχες πραγματικές του κυρίως προγράμματος.

Η λίστα των τυπικών παραμέτρων καθορίζει τις παραμέτρους στη δήλωση του υποπρογράμματος. Οι τυπικές παράμετροι ονομάζονται και ως **ορίσματα**.

Εμβέλεια Σταθερών - Μεταβλητών

Το τμήμα του προγράμματος που ισχύουν οι μεταβλητές λέγεται **εμβέλεια μεταβλητών**.

Απεριόριστη εμβέλεια

Σύμφωνα με αυτή την αρχή όλες οι μεταβλητές και όλες οι σταθερές είναι γνωστές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, άσχετα που δηλώθηκαν. Όλες οι μεταβλητές είναι **καθολικές**.

Η απεριόριστη εμβέλεια καταστρατηγεί την αρχή της αυτονομίας των υποπρογραμμάτων, δημιουργεί πολλά προβλήματα και τελικά είναι αδύνατη για μεγάλα προγράμματα με πολλά υποπρογράμματα, αφού ο καθένας που γράφει κάποιο υποπρόγραμμα πρέπει να γνωρίζει τα ονόματα όλων των μεταβλητών που χρησιμοποιούνται στα υπόλοιπα υποπρογράμματα.

Περιορισμένη εμβέλεια

Η περιορισμένη εμβέλεια υποχρεώνει όλες τις μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, να δηλώνονται σε αυτό το τμήμα. Όλες οι μεταβλητές είναι **τοπικές**, δηλαδή ισχύουν για το υποπρόγραμμα στο οποίο δηλώθηκαν. Στη ΓΛΩΣΣΑ έχουμε περιορισμένη εμβέλεια!

Τα πλεονεκτήματα της περιορισμένης εμβέλειας είναι η **απόλυτη αυτονομία όλων των υποπρογραμμάτων** και η **δυνατότητα να χρησιμοποιείται οποιοδήποτε όνομα**, χωρίς να ενδιαφέρει αν το ίδιο χρησιμοποιείται σε άλλο υποπρόγραμμα.

- **Μερικώς Περιορισμένη εμβέλεια**

Σύμφωνα με αυτή την αρχή **άλλες μεταβλητές είναι τοπικές** και **άλλες καθολικές**. Κάθε γλώσσα προγραμματισμού έχει τους δικούς της κανόνες και μηχανισμούς για τον τρόπο και τις προϋποθέσεις που ορίζονται οι μεταβλητές ως τοπικές ή καθολικές.

Η μερικώς περιορισμένη εμβέλεια προσφέρει μερικά πλεονεκτήματα στον πεπειραμένο προγραμματιστή, αλλά για τον αρχάριο περιπλέκει το πρόγραμμα δυσκολεύοντας την ανάπτυξή του.

Διαδικασίες

Η διαδικασία είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος.

- Εισάγει δεδομένα
- Εκτελεί υπολογισμούς
- Μεταβάλλει τις τιμές των μεταβλητών
- Τυπώνει αποτελέσματα

Με τη χρήση των παραμέτρων αυτές τις τιμές μπορούν να τις μεταφέρουν και στα άλλα υποπρογράμματα.

Συναρτήσεις

Η συνάρτηση είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει **μόνο μία τιμή** με το όνομά της (όπως οι μαθηματικές συναρτήσεις).

Θα πρέπει να επισημανθεί ότι οι συναρτήσεις **δεν μπορούν να έχουν εντολές εισόδου-εξόδου** και ως εκ τούτου δε δύναται να πραγματοποιηθεί **κλήση διαδικασίας από συνάρτηση**! Γίνεται όμως μέσα σε μια συνάρτηση να χρησιμοποιηθεί (καλεστεί) κάποια άλλη συνάρτηση.

Διαφορές Διαδικασιών – Συναρτήσεων

	Διαδικασίες	Συναρτήσεις
Λειτουργία	Οι διαδικασίες μπορούν να εκτελέσουν μια **οποιαδήποτε λειτουργία**, π.χ. να εισάγουν δεδομένα, να εκτελούν υπολογισμούς, να τυπώνουν αποτελέσματα και να αλλάζουν τις τιμές των μεταβλητών.	Οι συναρτήσεις **υπολογίζουν μόνο μία τιμή** και μόνο αυτή επιστρέφουν στο κύριο πρόγραμμα ή στο υποπρόγραμμα που τις κάλεσε. Η τιμή αυτή μπορεί να είναι ΑΡΙΘΜΗΤΙΚΗ (ΑΚΕΡΑΙΑ ή ΠΡΑΓΜΑΤΙΚΗ), ΧΑΡΑΚΤΗΡΑΣ ή ΛΟΓΙΚΗ. Επίσης η συνάρτηση **δεν μπορεί να επιστρέψει πίνακα**!
Επιστροφή Τιμών	Οι διαδικασίες μεταφέρουν τα αποτελέσματα στα άλλα υποπρογράμματα **με τη χρήση παραμέτρων**.	Οι συναρτήσεις μεταφέρουν το αποτέλεσμα στο κύριο πρόγραμμα ή στο υποπρόγραμμα που τις κάλεσε **με το όνομά τους** και όχι με τη χρήση παραμέτρων. Μοιάζουν με τις μαθηματικές συναρτήσεις.
Εκτέλεση	Οι διαδικασίες εκτελούνται αν γράψουμε την εντολή **ΚΑΛΕΣΕ** και μετά το όνομα της διαδικασίας.	Οι συναρτήσεις εκτελούνται **με τη χρήση του ονόματος τους** μέσα σε οποιαδήποτε εντολή.
Παράμετροι	Οι διαδικασίες μπορούν να δεχθούν **καμία**, **μια** ή **περισσότερες παραμέτρους**.	Οι συναρτήσεις πρέπει **οπωσδήποτε να δεχθούν μια** η περισσότερες παραμέτρους.
Είσοδος - Έξοδος	Οι διαδικασίες μπορούν να έχουν εντολές εισόδου - εξόδου (ΓΡΑΨΕ, ΔΙΑΒΑΣΕ, ΚΑΛΕΣΕ, κλπ.)	Οι συναρτήσεις **δεν μπορούν να περιέχουν εντολές εισόδου ή εξόδου**.

Εκσφαλμάτωση

Η διαδικασία **ελέγχου**, **εντοπισμού** και **διόρθωσης** των σφαλμάτων ενός προγράμματος καλείται **εκσφαλμάτωση (debugging)**. Στόχος της διαδικασίας εκσφαλμάτωσης είναι ο εντοπισμός των σημείων του προγράμματος που προκαλούν προβλήματα στη λειτουργία του.

Συντακτικά λάθη

Τα **συντακτικά λάθη** ή αλλιώς τα **λάθη κατά την υλοποίηση** προκαλούνται κυρίως από λανθασμένη σύνταξη εντολών προγράμματος. Τέτοια λάθη μπορεί να είναι η λανθασμένη συγγραφή μιας δεσμευμένης λέξης της γλώσσας προγραμματισμού ή η χρήση μιας δομής ελέγχου χωρίς την εντολή τερματισμού της.

Ένα λάθος που προκαλείται κατά τη συγγραφή του προγράμματος **ανιχνεύεται από το μεταγλωττιστή**, ο οποίος εμφανίζει προς τον προγραμματιστή κάποιο προειδοποιητικό μήνυμα. Αν το πρόγραμμα περιέχει ένα λάθος αυτής της μορφής, δεν επιτρέπεται η εκτέλεσή του, μέχρι να το διορθώσει ο προγραμματιστής.

Τα σύγχρονα προγραμματιστικά περιβάλλοντα μας προφυλάσσουν αυτόματα από τα λάθη κατά την υλοποίηση, αφού παρέχουν εργαλεία αυτόματου ελέγχου σύνταξης των εντολών και παρακολουθούν τον προγραμματιστή κατά τη συγγραφή του προγράμματος. Μόλις διαπιστώσουν κάποιο συντακτικό λάθος, σταματούν και απαιτούν τη διόρθωσή του. Συνήθως αντιλαμβάνονται ακριβώς το λάθος που δημιουργήθηκε και προτείνουν αναλυτικά τον τρόπο διόρθωσής του, εμφανίζοντας σε ενημερωτικό πλαίσιο την ορθή σύνταξη της εντολής που προκλήθηκε το λάθος.

Λάθη κατά την εκτέλεση

Τα **λάθη που προκαλούνται κατά το χρόνο εκτέλεσης** του προγράμματος είναι πιο επώδυνα γιατί συνήθως εμφανίζονται σε πραγματικό περιβάλλον εκτέλεσης και τις περισσότερες φορές προκαλούν τον **αντικανονικό τερματισμό της εφαρμογής** και το κρέμασμα του συστήματος.

Όταν ένα λάθος προκληθεί κατά την εκτέλεση της εφαρμογής, είναι δυνατό να αντιμετωπισθεί μόνο με τη χρήση εντολών προγράμματος που το παγιδεύουν και εκτελούν τις κατάλληλες διαδικασίες χειρισμού του.

Η πρόληψη τέτοιων λαθών είναι αρκετά δύσκολη, αφού συνήθως οφείλονται σε καταστάσεις που δεν είναι εύκολο να ελεγχθούν από τον προγραμματιστή, ενώ πολλές φορές εμφανίζονται μετά από μεγάλο χρονικό διάστημα. Τέτοια λάθη είναι δυνατό να προκληθούν από την κλήση μιας διαδικασίας με δεδομένα που δεν μπορεί να χειριστεί, όπως η αναζήτηση διαγραμμένων αρχείων, η προσπάθεια διαίρεσης ενός αριθμού με το μηδέν, η υπερχείλιση μιας αριθμητικής μεταβλητής ή από δυσλειτουργία του υλικού μέρους του υπολογιστή, όπως η καταστροφή του σκληρού δίσκου του συστήματος, ο τερματισμός μιας σύνδεσης δικτύου και η αποσύνδεση του εκτυπωτή.

Λογικά λάθη

Τα λογικά λάθη είναι συνήθως **λάθη σχεδιασμού** και δεν προκαλούν τη διακοπή της εκτέλεσης του προγράμματος. Ενώ ο μεταγλωττιστής της γλώσσας προγραμματισμού δεν ανιχνεύει κανένα συντακτικό λάθος και κατά την εκτέλεση του προγράμματος δεν παρουσιάζονται ανεπιθύμητες καταστάσεις σφαλμάτων, τελικά **δεν παράγονται τα επιθυμητά αποτελέσματα**.

Η ανίχνευση τέτοιων λαθών δεν είναι δυνατό να πραγματοποιηθεί από κάποιο εργαλείο του υπολογιστή και διαπιστώνονται μόνο με τη διαδικασία ελέγχου (testing) και την ανάλυση των αποτελεσμάτων των προγραμμάτων.

Μέθοδος «Μαύρο κουτί»

Ένα **σενάριο ελέγχου** περιγράφει τα δεδομένα εισόδου ολόκληρου του προγράμματος ή τμήματος του προγράμματος (διαδικασία, συνάρτηση) και τα αναμενόμενα αποτελέσματα. Τα σενάρια ελέγχου εκτελούνται, είτε σε πραγματικό περιβάλλον προγραμματισμού είτε εικονικά με δημιουργία πίνακα τιμών των μεταβλητών. Σε περίπτωση αποκλίσεων μεταξύ των αναμενόμενων και των πραγματικών αποτελεσμάτων, υπάρχει λάθος το οποίο πρέπει να εντοπιστεί και να διορθωθεί.

Μια δημοφιλής τεχνική ελέγχου είναι ο **έλεγχος μαύρου κουτιού (black-box testing)**. Ονομάζεται έτσι επειδή τα δεδομένα εισόδου στα σενάρια ελέγχου προκύπτουν από τις προδιαγραφές του προγράμματος, αγνοώντας εντελώς τον κώδικα. Δηλαδή το πρόγραμμα μοιάζει σαν να βρίσκεται μέσα σε ένα μαύρο κουτί που κρύβει το περιεχόμενό του. Είναι σημαντικό να δημιουργούνται διαστήματα και για τις μη έγκυρες τιμές εισόδου, καθώς δεν μπορούμε να είμαστε σίγουροι ότι ένα πρόγραμμα θα τροφοδοτείται μόνο με έγκυρες τιμές.

Ιδανικά θα θέλαμε να ελέγξουμε όλες τις τιμές εισόδου και όλα τα πιθανά αποτελέσματα. Αυτό όμως είναι αδύνατο. Γι' αυτό προσπαθούμε να βρούμε αντιπροσωπευτικές τιμές για τα δεδομένα εισόδου που θα παράγουν αντιπροσωπευτικά αποτελέσματα. Το πρώτο βήμα είναι η **δημιουργία ισοδύναμων διαστημάτων τιμών** για τα δεδομένα εισόδου. Τα διαστήματα θεωρούνται ισοδύναμα, καθώς αν δεν υπάρχουν λάθη, τότε όλες οι τιμές ενός διαστήματος εισόδου θα παράγουν τιμές που θα ανήκουν στο ίδιο διάστημα αποτελεσμάτων.

Είναι σημαντικό να δημιουργούνται διαστήματα και για τις **μη έγκυρες τιμές εισόδου**, καθώς δεν μπορούμε να είμαστε σίγουροι ότι ένα πρόγραμμα θα τροφοδοτείται μόνο με έγκυρες τιμές.

Μετά τον καθορισμό των διαστημάτων πρέπει να επιλεγούν τιμές για τα σενάρια ελέγχου που να καλύπτουν όλα τα διαστήματα. Αφού τα διαστήματα είναι ισοδύναμα, μπορεί να επιλεγεί οποιαδήποτε τιμή από κάθε διάστημα. Μια καλύτερη στρατηγική είναι να γίνει **έλεγχος των ακραίων τιμών κάθε διαστήματος**, καθώς η εμπειρία έχει δείξει ότι τα περισσότερα λάθη γίνονται σε αυτά τα σημεία. Αυτό είναι λογικό, αν σκεφτούμε ότι τα διαστήματα τιμών θα υλοποιηθούν με κάποια μορφή δομής επιλογής, οπότε μπορεί να υπάρχουν λάθη στις λογικές συνθήκες, π.χ. συμπερίληψη ακραίας τιμής (<= αντί για <, >= αντί για >), παράλειψη ακραίας τιμής (< αντί για <=, > αντί για >=).

Η μεθοδολογία για τη δημιουργία σεναρίων ελέγχου είναι η εξής:

1. Δημιουργία ισοδύναμων διαστημάτων τιμών

2. Καθορισμός ακραίων τιμών διαστημάτων

3. Δημιουργία σεναρίων ελέγχου, όπως είναι αυτό που φαίνεται παρακάτω.

Πίνακας σεναρίου ελέγχου			
Α/Α	**Είσοδος**	**Αναμενόμενο Αποτέλεσμα**	**Περίπτωση που ελέγχεται**
1	-1	Μη έγκυρη βαθμολογία	Άνω άκρο διαστήματος **βαθμός < 0**
2	0	Ανεπιτυχής εξέταση	Κάτω άκρο διαστήματος **0 <= βαθμός < 10**
3	9	Ανεπιτυχής εξέταση	Άνω άκρο διαστήματος **0 <= βαθμός < 10**
4	10	Επιτυχής εξέταση	Κάτω άκρο διαστήματος **10 <= βαθμός <= 20**
5	20	Επιτυχής εξέταση	Άνω άκρο διαστήματος **10 <= βαθμός <= 20**
6	21	Μη έγκυρη βαθμολογία	Κάτω άκρο διαστήματος **βαθμός > 20**

Η τεχνική που περιγράφηκε ανωτέρω, μπορεί να εφαρμοστεί και σε **υποπρογράμματα**. Αυτό είναι συνηθισμένο σε μεγάλα προγράμματα που αποτελούνται από χιλιάδες γραμμές κώδικα τα οποία είναι οργανωμένα σε υποπρογράμματα. Σε αυτές τις περιπτώσεις **πρώτα ελέγχεται κάθε υποπρόγραμμα μεμονωμένα**. Αφού διαπιστωθεί η ορθή λειτουργία του καθενός, μόνο τότε πραγματοποιείται έλεγχος ολόκληρου του προγράμματος.

Αντικειμενοστραφής Προγραμματισμός

Ορισμός

Αντικειμενοστραφής προγραμματισμός ή **αντικειμενοστραφής σχεδίαση** είναι μια μεθοδολογία ανάπτυξης εφαρμογών η οποία στηρίζεται σε αυτόνομες προγραμματιστικές οντότητες με δική τους ταυτότητα και συμπεριφορά. Οι οντότητες αυτές καλούνται **αντικείμενα**, αντιστοιχούν σε φυσικές οντότητες ή έννοιες του φυσικού μας κόσμου, και δομούνται με βάση δεδομένα (ιδιότητες) που προσδιορίζουν την υπόστασή τους και ενέργειες (κανόνες συμπεριφοράς) που εφαρμόζονται πάνω στα δεδομένα. Σε μια εφαρμογή, ένα αντικείμενο είναι ο ομαδοποιημένος συνδυασμός δεδομένων και κώδικα, τα οποία έχουμε τη δυνατότητα να χειριστούμε ενιαία. Τα δεδομένα αποτελούν τα χαρακτηριστικά ενός αντικειμένου και αναφέρονται ως **ιδιότητες** ενώ οι ενέργειες καθορίζουν τη συμπεριφορά του. Οι ενέργειες στον αντικειμενοστραφή προγραμματισμό αναφέρονται και ως **μέθοδοι**.

Μεθοδολογία

Το **χτίσιμο** μιας αντικειμενοστραφούς εφαρμογής επιτυγχάνεται με τη δημιουργία και τον χειρισμό **αντικειμένων** τα οποία πρέπει να συνεργαστούν για την επίτευξη του κοινού στόχου που είναι η επίλυση του προβλήματος. Με ποιον όμως τρόπο εργαζόμαστε, ώστε να εντοπίσουμε τα απαραίτητα δομικά στοιχεία της εφαρμογής;

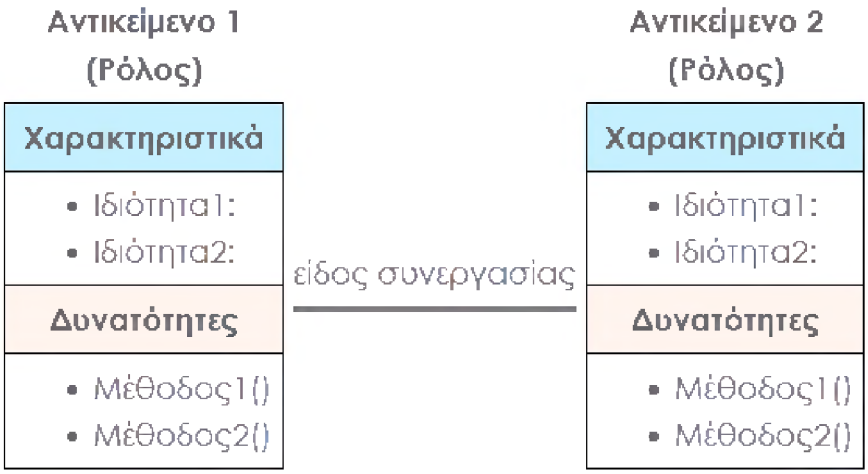
Το μόνο που έχουμε να κάνουμε είναι να **αναλύσουμε το πρόβλημα** το οποίο θέλουμε να επιλύσουμε, δηλαδή να αναγνωρίσουμε και να καταγράψουμε τα βασικά συστατικά στοιχεία της διαδικασίας επίλυσής του που είναι:

1. τα **αντικείμενα** που συμμετέχουν με βάση τον ρόλο τους στο συγκεκριμένο σενάριο,
2. οι **ιδιότητες** κάθε αντικειμένου, δηλ. τα σχετικά με το συγκεκριμένο πρόβλημα χαρακτηριστικά του, και
3. οι **υπηρεσίες** που προσφέρει ή οι **ενέργειες** που υλοποιεί κάθε αντικείμενο (μέθοδοι) προς αξιοποίηση από άλλες, ώστε να αναπτυχθούν οι απαραίτητες συνεργασίες μεταξύ των αντικειμένων για την επίλυση του προβλήματος.

Διαγραμματική Αναπαράσταση

Ένα **αντικειμενοστραφές πρόγραμμα** δομείται ως ένα **δίκτυο συνεργαζόμενων οντοτήτων** που είναι τα αντικείμενα. Κάθε αντικείμενο έχει ένα συγκεκριμένο ρόλο στην εφαρμογή και παρέχει μια υπηρεσία ή εκτελεί μια ενέργεια (μέθοδο) που χρησιμοποιείται από άλλα μέλη του δικτύου, δηλαδή από άλλα αντικείμενα, για την υλοποίηση της συνεργασίας που θα επιλύσει το πρόβλημα.

Αφού εντοπίσουμε τα συστατικά επίλυσης του προβλήματος, μπορούμε να τα οργανώσουμε σε μια απλή διαγραμματική αναπαράσταση χρησιμοποιώντας **παραλληλόγραμμα** για την αποτύπωση των αντικειμένων, των ιδιοτήτων και των μεθόδων τους και **γραμμές σύνδεσης** για την περιγραφή του είδους της μεταξύ τους συνεργασίας, όπως φαίνεται παρακάτω:



Η αναπαράσταση αυτή είναι ιδιαίτερα σημαντική, διότι **μας δίνει την εποπτική εικόνα** των συνεργαζόμενων οντοτήτων του προβλήματός μας και ουσιαστικά αποτελεί το σχέδιο επίλυσής του με βάση την αντικειμενοστραφή προσέγγιση.

Κλάσεις: Αφαιρετικότητα και Ενθυλάκωση

Στον Αντικειμενοστραφή Προγραμματισμό το αντικείμενο αποτελεί έναν ***θύλακα***, δηλαδή ένα σακούλι στο οποίο αποθηκεύει και συνδυάζει τα δεδομένα (**ιδιότητες**) και τις λειτουργίες (**μεθόδους**) του. Λέμε λοιπόν ότι τα αντικείμενα παρέχουν έναν τρόπο ενθυλάκωσης δεδομένων και λειτουργιών σε αυτά.

Σε μια αντικειμενοστραφή εφαρμογή κάθε αντικείμενο αποτελεί ξεχωριστή οντότητα και περιέχει ενσωματωμένες τις ιδιότητες (δεδομένα) και τους κανόνες συμπεριφοράς του (μεθόδους). Η δυνατότητα ενός αντικειμένου να συνδυάζει εσωτερικά τα δεδομένα και τις μεθόδους χειρισμού του καλείται **ενθυλάκωση**. Την ενθυλάκωση μπορούμε να την παρομοιάσουμε σαν ένα **κέλυφος** που υπάρχει γύρω από κάθε αντικείμενο και **διαχωρίζει τον εσωτερικό από τον εξωτερικό του κόσμο**.

Ο γενικός τύπος ενός αντικειμένου καλείται **κλάση** και καθορίζει τις αρχικές ιδιότητες και τη συμπεριφορά κάθε αντικειμένου που προέρχεται από αυτή. Μια κλάση αποτελεί ένα **αφαιρετικό στοιχείο** και μπορεί να παράγει ένα απεριόριστο πλήθος δομικά ίδιων αντικειμένων.

Κληρονομικότητα

Η δυνατότητα δημιουργίας ιεραρχιών αντικειμένων καλείται **κληρονομικότητα**. Με βάση την κληρονομικότητα, μια κλάση μπορεί να περιγραφεί γενικά, και στη συνέχεια μέσω αυτής της κλάσης να οριστούν **υποκλάσεις αντικειμένων**. Η κλάση απόγονος (**υποκλάση**) κληρονομεί και μπορεί να χρησιμοποιήσει όλα τα δεδομένα (ιδιότητες) και τις μεθόδους που περιέχει η κλάση πρόγονος (**υπερκλάση**).

Σε μια σχέση κληρονομικότητας, η κλάση-**πρόγονος** περιλαμβάνει τις **κοινές ιδιότητες και μεθόδους όλων των κλάσεων-απογόνων της**, ενώ οι κλάσεις-**απόγονοι** εμφανίζουν **μόνο τις διαφορετικές τους ιδιότητες και μεθόδους** αφού τις κοινές τις κληρονομούν από τη γονική τους κλάση.

Πως γενικοποιούμε κλάσεις σε μια υπερκλάση και πως διαχειριζόμαστε ιδιότητες και μεθόδους

Αν θέλουμε να δημιουργήσουμε μια υπερκλάση που να γενικεύει κάποιες κλάσεις του σεναρίου μας, τότε θα πρέπει να προσέξουμε τα εξής:

1. Να βρούμε τις **κοινές ιδιότητες** που έχουν όλες οι **υποκλάσεις**. Αυτές θα γίνουν οι ιδιότητες της υπερκλάσης.

   ΠΡΟΣΟΧΗ! Αν δεν υπάρχουν κοινές ιδιότητες, αφήνουμε τις ιδιότητες της υπερκλάσης **κενές**!

2. Να βρούμε τις **κοινές μεθόδους** που έχουν όλες οι **υποκλάσεις**. Αυτές θα γίνουν οι μέθοδοι της υπερκλάσης.

   ΠΡΟΣΟΧΗ! Αν δεν υπάρχουν κοινές μέθοδοι, αφήνουμε τις μεθόδους της υπερκλάσης **κενές**!

Αν έχουμε την υπερκλάση και θέλουμε να δημιουργήσουμε υποκλάσεις, τότε θα πρέπει να προσέξουμε τα εξής:

1. Η υποκλάση κληρονομεί **ΥΠΟΧΡΕΩΤΙΚΑ όλες** τις **ιδιότητες** και τις **μεθόδους** της υπερκλάσης της.
2. Στη **νέα υποκλάση** δε γράφουμε ξανά τις **ιδιότητες** που έχουν κληρονομηθεί. Γράφουμε μόνο **τις ξεχωριστές δικές της**.
3. Στη **νέα υποκλάση** δε γράφουμε ξανά τις **μεθόδους** που έχουν κληρονομηθεί, **ΕΚΤΟΣ** και αν αυτές πραγματοποιούνται με διαφορετικό τρόπο. Τότε θα πρέπει να τις γράψουμε ως μεθόδους της υποκλάσης γιατί γίνονται με δικό της τρόπο. Επίσης, γράφουμε επιπρόσθετες δικές της ξεχωριστές μεθόδους.

Πολυμορφισμός

Πολυμορφισμός είναι μια ιδιότητα του αντικειμενοστραφούς προγραμματισμού με την οποία **μια λειτουργία μπορεί να υλοποιείται με πολλούς διαφορετικούς τρόπους**.

Πολυμορφισμός είναι η ικανότητα να συμπεριφερόμαστε διαφορετικά, ανάλογα με το αντίστοιχο πλαίσιο μέσα στο οποίο λειτουργούμε.

Ένας **άντρας** για παράδειγμα, ανάλογα με το που βρίσκεται, αναλαμβάνει και διαφορετικούς ρόλους.

- Στη **δουλειά** του είναι *Επαγγελματίας* **και** *Συνεργάτης* **και** *Σύμβουλος*
- Στην **οικογένεια** του είναι *Πατέρας* **και** *Σύζυγος*
- Σε ένα **κατάστημα** είναι *Πελάτης*
- κ.ο.κ.