Αλγόριθμοι:

Κεντρική θέση στο θέμα του μαθήματος (αλλά και στην επιστήμη της Πληροφορικής γενικότερα) κατέχουν έννοιες όπως *αλγόριθμοι*, *δεδομένα*, *τύποι* και *αναπαράσταση δεδομένων*, *πρόγραμμα*, *ανάλυση* και *σχεδιασμός*, *γλώσσες προγραμματισμού* κλπ. Αυτές τις έννοιες θα συναντήσουμε και εξετάσουμε λεπτομερειακά στην πορεία.

Ξεκινάμε με την έννοια του αλγόριθμου γιατί με αυτή είμαστε εξοικειωμένοι ακόμη και αν δεν έχουμε οποιαδήποτε ειδική σχέση με Πληροφορική και υπολογιστές. Λίγο ή πολύ, μπορούμε να πούμε ότι αλγόριθμο ονομάζουμε μια διαδοχή από προκαθορισμένα βήματα για την επίλυση ενός προβλήματος. Αυτό όμως δεν αρκεί για να μας δώσει να καταλάβουμε τη σημασία των αλγόριθμων ούτε και όλες τις πλευρές αυτής της έννοιας. Ας αρχίσουμε με μερικά απλά παραδείγματα:

- **Πρόσθεση**

Για να εκτελέσουμε την πρόσθεση δύο πολυψήφιων αριθμών με τον τρόπο που ξέρουμε από το σχολείο ακολουθούμε μία συγκεκριμένη διαδικασία με καθορισμένα βήματα. Όπως ξέρουμε, η πρόσθεση γίνεται διατάσσοντας τους προσθετέους με τις μονάδες του ενός κάτω από τις μονάδες του άλλου και παρόμοια για τις δεκάδες, εκατοντάδες και ανώτερες τάξεις μεγέθους. Ύστερα, με μια *επαναληπτική* διαδικασία προσθέτουμε τα ψηφία της ίδιας τάξης, καθώς και το κρατούμενο απο το προηγούμενο στάδιο, αν υπάρχει. Εφόσον το αποτέλεσμα έχει ψηφίο της αμέσως επόμενης τάξης μεγέθους, το μεταφέρουμε στο επόμενο στάδιο ως νέο κρατούμενο.

Για παράδειγμα ας θεωρήσουμε την πρόσθεση δύο τριψήφιων αριθμών, του Π = 567 με τον Ρ = 947. Για συντομία, θα συμβολίσουμε τα ψηφία με το γράμμα του αριθμού, Π ή Ρ ή Α για το άθροισμα, και ένα δείκτη για κάθε στάδιο της πρόσθεσης. Θεωρώντας ότι το κρατούμενο ψηφίο Κ είναι μηδέν, έχουμε τα παρακάτω στάδια:

Στάδιο 0: Κρατούμενο Κ = 0.

Κρατούμενο Κ				0
Προσθετέος Π	0	5	6	7
Προσθετέος Ρ	0	9	4	7
Αποτέλεσμα Α				

Στάδιο 1: Χ = Κ + Π1 + Ρ1 = 0 + 7 + 7 = 14
Α1 = 4 (τελευταίο ψηφίο του Χ)
Κ = 1 (πρώτο ψηφίο του Χ)

Κρατούμενο Κ			1	
Προσθετέος Π	0	5	6	7
Προσθετέος Ρ	0	9	4	7
Αποτέλεσμα Α				4

Στάδιο 2: Χ = Κ + Π2 + Ρ2 = 1 + 6 + 4 = 11
Α2 = 1 (τελευταίο ψηφίο του Χ)
Κ = 1 (πρώτο ψηφίο του Χ)

Κρατούμενο Κ		1		
Προσθετέος Π	0	5	6	7
Προσθετέος Ρ	0	9	4	7
Αποτέλεσμα Α			1	4

Στάδιο 3: Χ = Κ + Π3 + Ρ3 = 1 + 5 + 9 = 15
Α3 = 5 (τελευταίο ψηφίο του Χ)
Κ = 1 (πρώτο ψηφίο του Χ)

Κρατούμενο Κ	1			
Προσθετέος Π	0	5	6	7
Προσθετέος Ρ	0	9	4	7
Αποτέλεσμα Α		5	1	4

Στάδιο 4: Α4 = Κ

Κρατούμενο Κ	1			
Προσθετέος Π	0	5	6	7
Προσθετέος Ρ	0	9	4	7
Αποτέλεσμα Α	1	5	1	4

Η περιγραφή των παραπάνω σταδίων δεν είναι τόσο γενική όσο θα επέβαλλαν περιπτώσεις όπως η πρόσθεση περισσότερων από δύο αριθμούς, ωστόσο αυτό μας επιτρέπει να δούμε πιο καθαρά ορισμένα ενδιαφέροντα χαρακτηριστικά της διαδικασίας. Το ένα είναι η *επανάληψη.*Τα στάδια 1, 2 και 3 μπορούν να γραφούν με την εξής *γενική* μορφή:

Στάδιο n: Χ = Κ + $Π_n$ + $Ρ_n$
$Α_n$ = τελευταίο ψηφίο του Χ
Κ = πρώτο ψηφίο του Χ

Έχουμε δηλαδή μία βασική δομή που επαναλαμβάνεται αναλλοίωτη, ως προς τα βασικά της χαρακτηριστικά. Αυτό επιτρέπει επίσης να γενικεύσουμε την παραπάνω διαδικασία για οσοδήποτε μεγάλο αριθμό ψηφίων έχουν οι προσθετέοι.
Στο εσωτερικό αυτής της επαναληπτικής δομής, παρατηρούμε τη *διαδοχή* τριών στοιχειωδέστερων ενεργειών. Αυτές επίσης έχουν μία γενική μορφή από κοινού:

Ποσότητα στο αριστερό μέλος = τιμή παράστασης ή αποτέλεσμα ενεργειών στο δεξί μέλος.

Το δεύτερο ενδιαφέρον χαρακτηριστικό της διαδικασίας της πρόσθεσης επομένως, είναι η *διαδοχική* εκτέλεση ενεργειών (άθροιση κρατουμένου και ομολόγων ψηφίων, καταγραφή ψηφίου για το αποτέλεσμα και φύλαξη νέου κρατουμένου) που έχουν ως κοινό σημείο την *απόδοση* μιας τιμής σε μία παράμετρο του προβλήματος.
Εκτός από τα παραπάνω δύο στοιχεία (επανάληψη, διαδοχή ενεργειών απόδοσης τιμής) υπάρχουν και άλλες ενδιαφέρουσες πλευρές της διαδικασίας όπως η απόδοση *αρχικής τιμής* στο κρατούμενο ψηφίο Κ (στάδιο 0) και η ύπαρξη μιας ενδιάμεσης, βοηθητικής παραμέτρου *τοπικής εμβέλειας* (Χ) στην επαναληπτική δομή εύρεσης των ψηφίων. Τέλος, υπάρχει μία ακόμα ενδιαφέρουσα πλευρά η οποία δεν είναι άμεσα ορατή στη παραπάνω παρουσίαση της διαδικασίας της πρόσθεσης και αυτή είναι η λήψη απόφασης για τη συνέχιση ή λήξη της πράξης της πρόσθεσης, ανάλογα με το αν υπάρχουν ή όχι άλλα ψηφία σε τουλάχιστον έναν από τους προσθετέους. Αυτό όμως, θα μπορούσαμε να το διατυπώσουμε εναλλακτικά λέγοντας ότι όταν προσθέτουμε τους αριθμούς μετράμε τα ψηφία που μένουν και όταν ο μετρητής μηδενιστεί τότε η πρόσθεση τελειώνει. Παίρνοντας υπ' όψιν όλα τα παραπάνω θα μπορούσαμε να διατυπώσουμε τα βήματα για τη διαδικασία της πρόσθεσης δύο το πολύ *n*-ψήφιων αριθμών ως εξής:

Θέσε μετρητή υπόλοιπων ψηφίων Ψ = n
Θέσε Κ = 0
Για όσο ισχύει Ψ > 0 κάνε:
Θέσε Χ = Κ + $Π_n$ + Pn
Θέσε $Α_n$ = τελευταίο ψηφίο του Χ
Θέσε Κ = πρώτο ψηφίο του Χ

Μείωσε το Ψ κατά 1.

Αn+1 = K

Προτεινόμενη Ασκηση 1: Αναλύστε με τρόπο ανάλογο όπως πιο πάνω, τη διαδικασία για την αφαίρεση ενός αριθμού από έναν άλλο και διατυπώστε με συνοπτική και γενική μορφή τα βήματα αυτής. Μήπως χρειάζεται να εισάγετε άλλα στοιχεία εκτός από διαδοχή απόδοσης τιμών και επανάληψη;

– **Τριώνυμο**

Η απόφαση για συνέχιση ή μη μιας σειράς ενεργειών είναι ειδική περίπτωση της επιλογής ανάμεσα σε πολλές εναλλακτικές οδούς όπου η σχετική απόφαση παίρνεται με βάση κάποιο κριτήριο. 'Ενα απλό παράδειγμα όπου η επιλογή και λήψη απόφασης βρίσκεται στην καρδιά του προβλήματος είναι η επίλυση της δευτεροβάθμιας εξίσωσης. Γενικά, το πρόβλημα τίθεται ως εξής: με δεδομένο τριώνυμο με συντελεστές *a*, *b*, *c* και μεγιστοβάθμιο συντελεστή διάφορο του μηδενός να βρεθούν οι ρίζες του. Αυτό απαιτεί τον υπολογισμός της διακρίνουσας $\Delta = b^2 - 4\,a\,c$. Αν η διακρίνουσα είναι θετική τότε έχουμε δύο πραγματικές ρίζες, αν είναι μηδέν υπάρχει μία πραγματική ρίζα και τέλος, αν είναι αρνητική έχουμε δύο μιγαδικές συζυγείς ρίζες (ή καμία ρίζα, αν δε δεχόμαστε μιγαδικούς αριθμούς για το πρόβλημα που μας ενδιαφέρει).

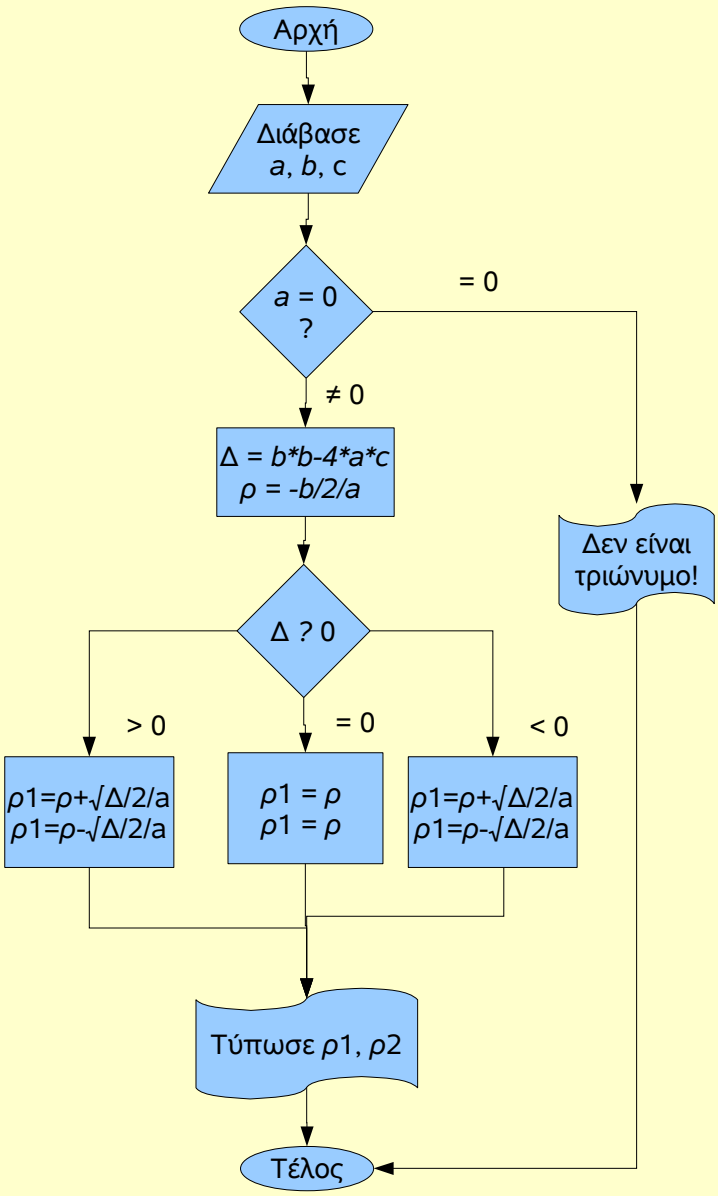
Το σημείο μιας μαθηματικής ή άλλης διαδικασίας όπου εμφανίζεται η ανάγκη της επιλογής μεταξύ πολλών δυνατών ενεργειών αναφέρεται συχνά ως σημείο διακλάδωσης. Αυτό γίνεται κατανοητό αν την "αλγεβρική" παρουσίαση των βημάτων επίλυσης που χρησιμοποιήσαμε στο παράδειγμα της πρόσθεσης, την αντικαταστήσουμε με μια οπτική αναπαράσταση της διαδικασίας λύσης του προβλήματος. Θα χρησιμοποιήσουμε λοιπόν, ένα *λογικό διάγραμμα* ή *διάγραμμα ροής* της λύσης, όπως φαίνεται στην Εικόνα 1.

Σε αυτή την εικόνα διακρίνουμε τις ενέργειες απόδοσης τιμής στις παραμέτρους ή *μεταβλητές* του προβλήματος, Δ, *ρ*, *ρ*1 και *ρ*2 σε ορθογώνια πλαίσια ενώ η επιλογή εντοπίζεται στους ρόμβους όπου γράφεται και η συνθήκη ή το κριτήριο που επιβάλλει τη μία ή την άλλη κατεύθυνση προς την τελική λύση. Η ύπαρξη διαφορετικών "διαδρομών" προς τη λύση που ξεκινούν από το σημείο όπου εμφανίζεται η επιλογή και η ανάγκη λήψης απόφασης ανάλογα με τα δεδομένα και τις συνθήκες που αυτά ικανοποιούν δικαιολογεί τον όρο διακλάδωση. Παρατηρούμε επίσης, ότι εκτός από την απόδοση τιμών και την επιλογή υπάρχουν και κάποιες "βοηθητικές" ενέργειες, και συγκεκριμένα η ανάγνωση των δεδομένων (εδώ: συντελεστές του τριωνύμου που το χαρακτηρίζουν πλήρως) και η απόδοση (π.χ. εκτύπωση) των αποτελεσμάτων.

Οι δύο τρόποι αναπαράστασης που χρησιμοποιήσαμε ως εδώ για να αναπαραστήσουμε τα δύο παραδείγματα αλγόριθμων, δηλαδή ένα είδος συνοπτικής γλώσσας για την αναφορά στις στοιχειώδεις ενέργειες που λαμβάνουν χώρα και το διάγραμμα ροής χρησιμοποιούνται ευρέως για την περιγραφή αλγορίθμων. Η πρώτη αναπαράσταση έχει και αυτή όνομα και λέγεται αναπαράσταση σε *ψευδογλώσσα*.

Η επίλυση της δευτεροβάθμιας μπορεί να διατυπωθεί και αυτή σε ψευδογλώσσα:

Διάβασε a, b, c
Αν (a ≠ 0) τότε
    ρ = -b/2/a
    *Δ = b*b-4*a*c*
    Αν (Δ > 0) τότε
        ρ1 = ρ + √Δ/2/a
        ρ1 = ρ - √Δ/2/a
    αλλοιώς αν (Δ = 0) τότε
        ρ1 = ρ2 = ρ
    αλλοιώς
        ρ1 = ρ + i√|Δ|/2/a
        ρ1 = ρ – i√|Δ|/2/a
    Τύπωσε ρ1, ρ2
αλλοιώς
    Τύπωσε "Δεν είναι τριώνυμο!"



Εικόνα 1: λογικό διάγραμμα επίλυσης δευτεροβάθμιας εξίσωσης

– **Εύρεση μέγιστου και ελάχιστου από ένα πλήθος αριθμών**.

Έστω μία ακολουθία αριθμών που μπορεί να είναι οι καταγεγραμμένες θερμοκρασίες σε μια πόλη για ένα ορισμένο διάστημα, οι τιμές μιας μετοχής στο χρηματιστήριο ή το ύψος ή το βάρος καθενός από μια ομάδα ανθρώπων κλπ. Σε πολλές τέτοιες περιπτώσεις ενδιαφερόμαστε να εντοπίσουμε τον μέγιστο και τον ελάχιστο αριθμό, όπως επίσης και άλλα χαρακτηριστικά του υπό μελέτη δείγματος (μέση τιμή, διασπορά κλπ). Ένας απλός τρόπος για να βρούμε το μέγιστο και τον ελάχιστο είναι ο εξής: Διαβάζουμε τον πρώτο αριθμό και θεωρούμε ότι ταυτίζεται με το μέγιστο και τον ελάχιστο συγχρόνως. Επομένως, πρώτα δίνουμε αρχική τιμή στο μέγιστο και στον ελάχιστο. Διαβάζουμε τους υπόλοιπους αριθμούς έναν-έναν. Συγκρίνοντας τον εκάστοτε αριθμό με την τρέχουσα τιμή του μέγιστου και του ελάχιστου τις ενημερώνουμε ανάλογα με το αν ο νέος αριθμός είναι μεγαλύτερος από την πρώτη, μικρότερος από τη δεύτερη ή τίποτε από τα δύο. Θεωρώντας και το πρόβλημα του να καθοριστεί η θέση του ελάχιστου και μέγιστου στη δεδομένη ακολουθία αριθμών, ο αλγόριθμος θα μπορούσε να γραφεί σε ψευδογλώσσα κάπως έτσι:

Διάβασε πρώτο αριθμό X0
Θέσε μετρητή I = 1
Θέσε μετρητές θέσης μέγιστου και ελάχιστου IMIN = IMAX = 1
Θέσε MIN = MAX = X0
Όσο υπάρχει επόμενος αριθμός X κάνε:
    Αύξησε I κατά 1
    Αν X < MIN τότε
        θέσε MIN = X
        IMIN = I
    Αν X > MAXτότε
        θέσε MAX = X
        IMAX = I
Γράψε IMIN, MIN, IMAX, MAX

Είναι πολύ εύκολο να επαληθευτεί η εγκυρότητα της παραπάνω διαδικασίας με απλά παραδείγματα που δε χρειάζεται να δοθούν εδώ. Θα τονίσουμε μόνο ότι για άλλη μια φορά συναντούμε τις δομές της *επανάληψης* (έλεγχος για κάθε επόμενο αριθμό) και της *επιλογής* (αναλόγως αν ο αριθμός είναι μικρότερος από MIN ή μεγαλύτερος από MAX ενημερώνουμε τις *μεταβλητές* του προβλήματος).

Όλα τα παραπάνω και πολλά άλλα προβλήματα λύνονται μέσα από διαδικασίες κατά τις οποίες ακολουθούμε βήματα ορισμένα με σαφήνεια για την επεξεργασία των δεδομένων. Αυτού του είδους οι διαδικασίες λέγονται **αλγόριθμοι.**
Ορισμός: Αλγόριθμος είναι μία πεπερασμένη ακολουθία από σαφώς ορισμένες οδηγίες για την εκτέλεση ενός σκοπού, η οποία, για δεδομένη αρχική κατάσταση, οδηγεί σε σαφώς ορισμένες διαδοχικές καταστάσεις και πιθανά σε μία τελική κατάσταση.

Παρατηρούμε ότι σε όλες τις περιπτώσεις υπάρχουν οι εξής βασικές δομές:
- "συνένωση" ή παράθεση ή πιο απλά, διαδοχική εκτέλεση βημάτων,
- επιλογή
- επανάληψη ή "ανακύκλωση".

Τα βήματα που "συνενώνονται" με τη διαδοχική τους παράθεση μπορεί να είναι σύνθετα και να αποτελούνται με τη σειρά τους από άλλες δομές συνένωσης διαδοχικών βημάτων, επιλογής και επανάληψης (ή και συνδυασμούς τους) που και αυτές επίσης αναλύονται σε άλλες απλούστερες. Στο κατώτερο δυνατό επίπεδο θα συναντήσουμε απλές "λειτουργίες" ανάθεσης μιας τιμής σε μια από τις μεταβλητές του προβλήματος. Αυτή η τιμή μπορεί να προέρχεται από τον υπολογισμό μιας *παράστασης* ή να είναι η τρέχουσα τιμή μιας άλλης μεταβλητής ή ακόμη και κάποια σταθερά (π.χ. ένας αριθμός).
Οι παραπάνω τρεις βασικές δομές αποδεικνύεται ότι μπορούν να αποτελέσουν τους "δομικούς λίθους" για όλων των ειδών τους αλγόριθμους.
Επίσης, είναι εύλογο ότι:
Τα σύνθετα προβλήματα με άγνωστη προς το παρόν λύση αντιμετωπίζονται ευκολώτερα αν τα "σπάσουμε" σε μικρότερα για τα οποία ξέρουμε τη λύση.

> ***Προτεινόμενη Άσκηση 2***: Αναλύστε και διατυπώστε με τρόπο ανάλογο όπως στην Άσκηση 1 τη διαδικασία για τον πολλαπλασιασμό και τη διαίρεση δύο αριθμών. Πώς μπορείτε να αναλύσετε το πρόβλημα σε υποπροβλήματα; Υποθέστε π.χ. ότι αφού το πρόβλημα της πρόσθεσης πολυψήφιων αριθμών έχει λυθεί, έχετε στη διάθεσή σας μια διαδικασία *Πρόσθεση*(Π, *m*, Ρ, *n*) που δίνει το άθροισμα δύο αριθμών Π και Ρ με *m* και *n* ψηφία αντιστοίχως το οποίο μπορείτε να αποδώσετε σε μια παράμετρο Χ ως εξής: Χ = *Πρόσθεση*(Π, *m*, Ρ, *n*).

Για σύνθετα προβλήματα που απαιτούν πολύ χρόνο (επίλυση πολυσύνθετων συστημάτων εξισώσεων) ή διαχείριση μεγάλου όγκου δεδομένων (π.χ. τραπεζικές συναλλαγές κλπ) θα ήταν ευχής έργο να μπορέσουμε να τυποποιήσουμε και να αυτοματοποιήσουμε τις παραπάνω διαδικασίες.
- Με την τυποποίηση μπορούμε να τις διατυπώσουμε με γενική μορφή και να εφαρμόσουμε έναν

αλγόριθμο για πολλά και διαφορετικά προβλήματα που η ανάλυσή τους δείχνει ότι έχουν την ίδια τυπική μορφή, ανεξάρτητα από το συγκεκριμένο τους περιεχόμενο.

- Η αυτοματοποίηση της διαδικασίας με κάποια μηχανικά μέσα επιταχύνει την επεξεργασία των δεδομένων και βέβαια μας απαλλάσσει από την υποχρέωση να ασχοληθούμε με το κοπιαστικό και πληκτικό μέρος της διαδικασίας επιτρέποντάς μας να ασχολούμαστε με πιο ουσιώδη πράγματα και να ζούμε μια πιο εύκολη και άνετη ζωή.
  Προτεινόμενη Άσκηση 3: Διερωτηθείτε γιατί, παρά τις τεράστιες δυνατότητες των σύγχρονων υπολογιστών, η ζωή μας παραμένει δύσκολη και πολύπλοκη. Προτείνετε τουλάχιστον τρεις πιθανές αιτίες. Συμπεραίνετε ότι οι υπολογιστές και η πληροφορική είναι σε τελική ανάλυση πράγματα άχρηστα και ίσως και επιβλαβή ή όχι και γιατί;

Ένα πρώτο βήμα προς την κατεύθυνση της τυποποίησης και αυτοματοποίησης είναι η διατύπωση των αλγορίθμων σε μια ενιαία κωδικοποιημένη "Ψευδογλώσσα" για την αναπαράστασή τους. Θα μπορούσαμε να χρησιμοποιήσουμε μια τέτοια συμβολική ή "συνθηματική" γλώσσα για να ορίσουμε τα απαραίτητα βήματα εκτέλεσης ενός αλγορίθμου, όπως άλλωστε είδαμε πιο πάνω και με συγκεκριμένα παραδείγματα.

Μετά, με κάποιον τρόπο, θα έπρεπε να μπορούμε να εκτελέσουμε το κάθε βήμα αυτόματα. Αυτό, προφανώς είναι η δουλειά των σύγχρονων υπολογιστών στους οποίους πρέπει να περάσουμε με κάποιο τρόπο τις εντολές αυτής της γλώσσας. Αυτό στην πράξη επιτυγχάνεται με τη βοήθεια των γλωσσών προγραμματισμού που μας επιτρέπουν να γράψουμε *προγράμματα,* δηλαδή να διατυπώσουμε με συγκεκριμένες λέξεις-κλειδιά τα διαδοχικά βήματα εκτέλεσης των αλγορίθμων. Το πώς ακριβώς γίνεται αυτό, το πώς ο υπολογιστής παραλαμβάνει τα προγράμματα που γράφουμε και εκτελεί τις εντολές που του δίνουμε μέσω αυτών θα το κατανοήσουμε αν εμβαθύνουμε για λίγο στην εσωτερική δομή του υπολογιστή (αρχιτεκτονική) που σχετίζεται με την *αποθήκευση*, *επεξεργασία* και *ανάκτηση* της πληροφορίας.

Ηλεκτρονική Επεξεργασία Πληροφορίας

Αρχιτεκτονική Ψηφιακού Ηλεκτρονικού Υπολογιστή:

Έναν υπολογιστή μπορούμε να τον εξετάσουμε από την άποψη της υλικής δομής του, όπως και από την άποψη της πληροφορίας που αποθηκεύει και χειρίζεται με ποικίλους τρόπους.

Υλικό (hardware): Ως τέτοιο ορίζεται το σύνολο των μηχανικών, ηλεκτρικών και ηλεκτρονικών μερών του υπολογιστή, καθώς και των συσκευών που συνδέονται με αυτόν.

Λογισμικό (software): Με τον όρο αυτόν αναφερόμαστε στα προγράμματα που χρησιμεύουν στη συντονισμένη λειτουργία των μερών του υπολογιστή και στην επικοινωνία με το χρήστη (*λειτουργικό σύστημα*), καθώς και τα προγράμματα είτε του κατασκευαστή είτε δικά μας, που μας εξυπηρετούν σε διάφορους σκοπούς και στην επίλυση ποικίλων προβλημάτων (*εφαρμογές*).

Υλικό

Από την άποψη του υλικού μπορούμε να διακρίνουμε τα εξής κύρια συστατικά μέρη ή κατηγορίες:

- Κύρια Μνήμη
- Κεντρική Μονάδα Επεξεργασίας (CPU). Αυτή με τη σειρά της υποδιαιρείται στα ακόλουθα δύο μέρη:
  - Αριθμητική και Λογική Μονάδα
  - Μονάδα Ελέγχου
- Ελεγκτές και Περιφερειακές συσκευές

Μνήμη.

Η πληροφορία προς χρήση αποθηκεύεται στα ηλεκτρονικά κυκλώματα που αποτελούν τη *μνήμη* του υπολογιστή. Η πληροφορία αυτή και συνεπώς και η χωρητικότητα της μνήμης, μπορεί να μετρηθεί. Η βασική μονάδα πληροφορίας και, αντιστοίχως, χωρητικότητας της μνήμης είναι το bit (=binary digit). Αυτό έχει καθιερωθεί ως μονάδα αποθηκευμένης πληροφορίας λόγω της ευκολίας υλοποίησής του σε

ηλεκτρονικά κυκλώματα, και συγκεκριμένα ως στοιχειώδες κύκλωμα που μπορεί να βρεθεί σε δυο διακριτές καταστάσεις: φορτισμένο-αφόρτιστο, χαμηλή τάση-υψηλή τάση, κλπ. (παρεμπιπτόντως, οπτικά κυκλώματα που λειτουργούν με laser μπορούν να αποθηκεύσουν πληροφορία και σε άλλα συστήματα πέραν του δυαδικού. Η σχετική έρευνα φιλοδοξεί να οδηγήσει σε νέου τύπου αρχιτεκτονικές).
Λόγω της χρήσης δυαδικού συστήματος για την αποθήκευση της πληροφορίας, οι δυνάμεις του 2 παίζουν σημαντικό ρόλο στη μέτρηση της μνήμης ενός υπολογιστικού συστήματος. Για να επικοινωνήσει ο άνθρωπος με τη μηχανή χρειάζεται να μεταδώσει πληροφορία εκφρασμένη σε σύμβολα όπως γράμματα και αριθμοί τα οποία πρέπει να μετατραπούν σε αντίστοιχες δυαδικές αναπαραστάσεις. Ανάλογα με τον αριθμό των συμβόλων που πρέπει να αποθηκευτούν, τα στοιχειώδη τμήματα της μνήμης, τα bits, πρέπει να ομαδοποιηθούν κατά τέτοιο τρόπο ώστε μία ομάδα να μπορεί να αναπαραστήσει όλα τα σύμβολα ανάλογα με το συνδυασμό των καταστάσεων, 0 ή 1, στις οποίες βρίσκονται τα επί μέρους bits. Αυτές οι ομαδοποιήσεις λέγονται *κελλιά* (cells) ή "*λέξεις*" (words) και συνήθως αποτελούνται από 8 bits επειδή οι δυνατοί συνδυασμοί των 0 και 1 είναι 2^8=256 που είναι επαρκείς για την αναπαράσταση των δεκαδικών ψηφίων, των γραμμάτων του λατινικού αλφαβήτου και άλλων συμβόλων αν χρειαστεί. Η λέξη των 8 bits λέγεται και byte. Αν χρειαστεί αποθήκευση πληροφορίας περισσότερης από όση μπορούν να παράσχουν οι 256 συνδυασμοί θα χρησιμοποιηθούν τα αμέσως γειτονικά bytes. Η χωρητικότητα της μνήμης εκφράζεται σε μονάδες byte και σε δυνάμεις του 2. Έτσι, 1 kiloByte (kB), 1 MegaByte (MB), 1 GigaByte (GB) σημαίνουν αντίστοιχα 2^{10} = 1024, 2^{20} = 1048576 και 2^{30} = 1073741824 bytes, αντίστοιχα. Παρατηρείστε τη διαφορά ανάμεσα στους παραπάνω αριθμούς και τη συνήθη χρήση των προθεμάτων kilo, Mega και Giga που συνήθως υποδηλώνουν ακριβώς 1000, 1000000 και 1000000000 αντίστοιχα.
Τέλος, πρέπει να πούμε ότι κάθε byte της μνήμης χαρακτηρίζεται από μία **διεύθυνση**, δηλαδή έναν ακέραιο αριθμό που καθορίζει επακριβώς τη θέση της μέσα στη μνήμη. Αυτό είναι ένα στοιχείο που εκμεταλλεύονται πολλές γλώσσες προγραμματισμού για να επιτύχουν ταχύτερη επεξεργασία των δεδομένων, κάνοντας χρήση των διευθύνσεων μνήμης όπου είναι αποθηκευμένες οι χρήσιμες πληροφορίες αντί για τα ίδια τα δεδομένα τα οποία και χειρίζονται μόνο όταν αυτό καταστεί απαραίτητο. Αυτό ισχύει γενικότερα και για τα διάφορα *λειτουργικά συστήματα* και τον τρόπο με τον οποίο αποθηκεύουν, ανακαλούν και διαγράφουν δεδομένα. Έτσι, αν ένα αρχείο με χρήσιμες πληροφορίες έχει αποθηκευτεί σε μια περιοχή της μνήμης που αρχίζει στη διεύθυνση Α και τελειώνει στη διεύθυνση Β, τότε αυτό το αρχείο χαρακτηρίζεται μοναδικά από τη διεύθυνση της αρχής του, Α που είναι και η διεύθυνση του αρχείου. Όταν θελήσουμε να ανοίξουμε ή να επεξεργαστούμε το αρχείο, τότε ο υπολογιστής θα ανατρέξει στη διεύθυνση Α για να αρχίσει την ανάγνωση των bytes επειδή την έχει σημειώσει σε άλλη περιοχή της μνήμης που είναι αφιερωμένη σε αυτό το σκοπό (δηλαδή παίζει το ρόλο καταλόγου ή ευρετηρίου), οπότε "ξέρει" ότι εκεί θα βρει το συγκεκριμένο αρχείο. Παρόμοια, αν θελήσουμε να σβήσουμε το αρχείο, ο υπολογιστής θα διαγράψει απλώς τη διεύθυνση από το "ευρετήριο" δηλώνοντας έτσι ότι η περιοχή που αρχίζει από το σημείο Α και εκτείνεται μέχρι μια ορισμένη "απόσταση" είναι ελεύθερη για χρήση, χωρίς να χρειαστεί να σβήσει όλα τα δεδομένα μεταξύ Α και Β.
Τα κυκλώματα της μνήμης διακρίνονται σε RAM (Random Access Memory – μνήμη τυχαίας προσπέλασης) και σε ROM (Read Only Memory – μνήμη μόνο για ανάγνωση). Η πρώτη είναι αυτή με την οποία κάνουμε "τη δουλειά μας" όταν χρησιμοποιούμε τον υπολογιστή: εκεί "φορτώνονται" τα προγράμματα για να εκτελεστούν και τα δεδομένα που θα χειριστούν. Αλλά τα δεδομένα αυτά σβήνονται και χάνονται όταν ο υπολογιστής σβήνει. Το αντίθετο συμβαίνει με τη ROM. Αυτή περιέχει δεδομένα χρήσιμα για τη λειτουργία του υπολογιστή τα οποία διατηρούνται ακόμη και με σβηστό υπολογιστή.

Η Κύρια Μνήμη συνδέεται μέσω διαύλου (bus) με την Κεντρική Μονάδα Επεξεργασίας, την οποία και εξετάζουμε στη συνέχεια.

Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ)
Η ΚΜΕ διαθέτει καταχωρητές (registers) για την αποθήκευση συγκεκριμένης πληροφορίας που είναι

“κελλιά” εντελώς ανάλογα με τις λέξεις της μνήμης.

Μονάδα Ελέγχου (Control Unit):
Αυτή διαβάζει τη μνήμη, δεδομένα και εντολές προγραμμάτων, τα μεταφέρει στους καταχωρητές, πληροφορεί την Αριθμητική – Λογική Μονάδα (ΑΛΜ) για τις διευθύνσεις των καταχωρητών όπου βρίσκονται τα δεδομένα καθώς και αυτές όπου θα αποθηκευτούν τυχόν αποτελέσματα, όπως επίσης και για τις πράξεις που πρέπει να επιτελεστούν ανάλογα με τις αποκωδικοποιημένες εντολές που διάβασε από τη μνήμη και μεταβάλλει τα δεδομένα ανάλογα με τις “οδηγίες” που θα πάρει από την ΑΛΜ.

Αριθμητική – Λογική Μονάδα, ΑΛΜ (Arithmetic Logic Unit, ALU).
Εκτελεί τις στοιχειώδεις πράξεις της αριθμητικής και της λογικής (COMPARE, AND, OR, SHIFT, ROTATE).

Περιφερειακά:
Μπορούμε να τα διακρίνουμε σε μαζικής αποθήκευσης (ταινίες, CD, σκληροί δίσκοι κλπ) και εισόδου/εξόδου (i/o – οθόνες, πληκτρολόγια, ποντίκια κλπ).
Στις συσκευές μαζικής αποθήκευσης η πληροφορία αποθηκεύεται με βάση τις ίδιες αρχές όπως και στην ΚΜ του ΗΥ (δυαδικό σύστημα κλπ) αν και ο συγκεκριμένος τρόπος υλοποίησης διαφέρει (μαγνήτιση/απομαγνήτιση, σημάδια ή καθαρές περιοχές στην επιφάνεια του CD κλπ). Σε σχέση με την ΚΜ είναι: πιο αργά, μεταφέρονται, αποθηκεύουν περισσότερη πληροφορία και τη διατηρούν μετά τη διακοπή της ηλεκτρικής τροφοδοσίας.

Ελεγκτές
συντονίζουν τις ενέργειες των περιφερειακών με αυτές του καθαυτό υπολογιστή.

Αναπαράσταση συμβόλων και αριθμών.
Πώς αναπαρίσταται η πληροφορία στους ΗΥ; Αυτό ποικίλλει ανάλογα με το είδος της πληροφορίας:
Χαρακτήρες και σύμβολα: κώδικας ASCII (ο πιο συνηθισμένος – προσωπικοί υπολογιστές), EBCDIC (mainframes). Κάθε byte έχει διάφορους συνδυασμούς ψηφίων 0 ή 1. Αν αυτά τα δούμε σαν αναπαράσταση ενός αριθμού στο δυαδικό σύστημα τότε κάθε byte μπορεί να περιέχει έναν αριθμό από 0 ως 255. Αν αντιστοιχίσουμε κάθε τέτοιο αριθμό σε ένα σύμβολο ή χαρακτήρα (όπου περιλαμβάνουμε και τα αριθμητικά ψηφία θεωρούμενα ως χαρακτήρες) έχουμε ένα σύστημα κωδικοποίησης. Αν ο υπολογιστής “ξέρει” με κάποιο τρόπο ότι σε ένα συγκεκριμένο byte θέλουμε να αποθηκεύονται χαρακτήρες και αυτός ο υπολογιστής χρησιμοποιεί π.χ. κωδικοποίηση ASCII, τότε κάθε τιμή που θα βρει σε αυτό το byte θα την ερμηνεύσει ως τον αντίστοιχο χαρακτήρα.
Ακέραιοι: δυαδικό σύστημα. Δυνάμεις του 2 από 0 και άνω. Π.χ. $10010110|_2 = 150|_{10}$
Αντίθετα από τους χαρακτήρες, οι ακέραιοι είναι “ακατέργαστα” δεδομένα, δηλαδή αν ο υπολογιστής “ξέρει” ότι σε κάποιο συγκεκριμένο byte αποθηκεύονται ακέραιοι, θα εκλάβει την αριθμητική τιμή που θα βρει εκεί ως τέτοια και όχι ως κωδικοποίηση κάποιου συμβόλου.
Οι αριθμοί που μπορούν να αποθηκευτούν σε ένα byte κυμαίνονται από 0 μέχρι 255, αλλά αυτό δεν επιτρέπει την αναπαράσταση των αρνητικών. Διέξοδος: το πρώτο bit να είναι το bit προσήμου: 1 = αρνητικός και 0 = θετικός. Τότε το εύρος γίνεται από -127 μέχρι 127. Εναλλακτική λύση που χρησιμοποιείται ευρέως είναι το συμπλήρωμα ως προς 2.
Ο αριθμός των διαθέσιμων bits επηρεάζει το μέγεθος των ακέραιων αριθμών που μπορούν να αποθηκευτούν. Η γλώσσα C επιτρέπει τη χρήση διαφορετικών τύπων ακεραίων που μπορεί να έχουν ή να μην έχουν πρόσημο καθώς και να καταλαμβάνουν 8 ή περισσότερα bits.
Πραγματικοί: Δυαδικό. Δυνάμεις του 2 από 0 και άνω αριστερά της υποδιαστολής και από -1 και κάτω δεξιά της υποδιαστολής. Χρήση επιστημονικής γραφής: mantissa X βάση ^ εκθέτης. Απαιτείται διαθέσιμη μνήμη για: mantissa, εκθέτη, πρόσημο αυτών των δυο.
Ο αριθμός των διαθέσιμων bits επηρεάζει τόσο το μέγεθος όσο και την ακρίβεια των πραγματικών αριθμών που μπορούν να αποθηκευτούν.
Διευθύνσεις μνήμης: Τίποτε το παράξενο. Μπορούν να αναπαρασταθούν και αποθηκευτούν ως ακέραιοι

αριθμοί σε δυαδική μορφή. Έτσι, αν για κάποιο λόγο, πρέπει να αποθηκεύσουμε τη διεύθυνση Α ενός κελλιού μνήμης, θα την καταγράψουμε σε δυαδική μορφή σε κάποιο άλλο κελλί με τη δική του, βεβαίως, διεύθυνση Β (που δεν έχει καμία ιδιαίτερη σχέση με την Α).
Εντολές: Ο επεξεργαστής (ΚΜΕ) έχει ηλεκτρονικά κυκλώματα που υλοποιούν στοιχειώδεις πράξεις μεταξύ των bits που της μεταφέρονται ή γενικότερα εντολές που αφορούν αυτά τα bits (οι γνωστές αριθμητικές πράξεις και στοιχειώδεις λογικές πράξεις). Αυτές είναι επίσης αποθηκευμένες με δυαδική μορφή (σαν πίνακες αλήθειας). Οι στοιχειώδεις αυτές πράξεις ή εντολές, αποτελούν αυτό που ονομάζουμε **γλώσσα μηχανής** και μπορούν να χρησιμεύσουν στην υλοποίηση πιο σύνθετων διαδικασιών οι οποίες με τη σειρά τους συντίθενται σε ένα ανώτερο επίπεδο για να δώσουν σύνθετες εντολές κάποιας γλώσσας προγραμματισμού ή κάποιας εντολής που μεταβιβάζεται στον ΗΥ μέσω κάποιου περιφερειακού. Αντίστροφα, οι εντολές που με οποιονδήποτε τρόπο διαβιβάζονται στην ΚΜΕ, μετατρέπονται σε δυαδικό κώδικα που παριστάνει μια ακολουθία από τέτοιες στοιχειώδεις εντολές του επεξεργαστή.
Συμπέρασμα: η πληροφορία που αποθηκεύεται στη μνήμη είναι κωδικοποιημένη σε δυαδικό σύστημα. Η *σύμβαση* όμως με την οποία μεταφράζεται ο δυαδικός κώδικας για να μας δώσει πίσω την πληροφορία, διαφέρει ανάλογα με το είδος των δεδομένων μας: σύμβολα, ακέραιοι αριθμοί, πραγματικοί αριθμοί, διευθύνσεις μνήμης, εντολές. Έτσι, όταν δίνουμε μια εντολή για τον χειρισμό του περιεχομένου μνήμης μιας συγκεκριμένης διεύθυνσης, πρέπει να προσδιορίσουμε τι είδους πληροφορία υπάρχει σε αυτή τη μνήμη: ακέραιος, πραγματικός, σύμβολο κλπ. Επίσης, πρέπει να προσδιορίσουμε σε τι έκταση μνήμης εκτείνεται η προς επεξεργασία πληροφορία (πόσα bits καταλαμβάνει). Διαφορετικά, ο υπολογιστής δεν θα ξέρει τι ακριβώς να κάνει με αυτή. Οι διάφορες γλώσσες προγραμματισμού, όπως η C, περιλαμβάνουν ειδικές **εντολές δήλωσης μεταβλητών** που αποσκοπούν στο να δεσμεύσουν περιοχές της μνήμης για αποθήκευση συγκεκριμένου είδους πληροφορίας όπως το έχει ορίσει ο χρήστης-προγραμματιστής.
Σημείωση: όταν για κάποιο λόγο θέλουμε να καταγράψουμε δυαδικές ακολουθίες σε μορφή πιο ευανάγνωστη, συνηθίζεται να χρησιμοποιούμε δεκαεξαδικό (ή και οκταδικό) σύστημα.

Γλώσσες Προγραμματισμού
Περί επιπέδων:
Μπορούμε να τις κατατάξουμε σε διάφορα **επίπεδα**. Στο πιο χαμηλό επίπεδο βρίσκεται η γλώσσα μηχανής που προαναφέρθηκε και η οποία διαφέρει από ΗΥ σε ΗΥ λόγω της διαφορετικής **αρχιτεκτονικής** που χρησιμοποιείται σε κάθε είδους επεξεργαστή. Περιλαμβάνει στοιχειώδεις αριθμητικές και λογικές πράξεις καθώς και άλλου είδους χειρισμό δεδομένων, όπως μεταφορά από μια θέση μνήμης σε άλλη, π.χ. για τη μεταφορά δεδομένων από τα περιφερειακά στην Κύρια Μνήμη και από εκεί στους καταχωρητές του επεξεργαστή.
Στο αμέσως ανώτερο επίπεδο, βρίσκεται η συμβολική γλώσσα (assembly) η οποία δεν είναι παρά η ένα-προς-ένα αντιστοιχία μεταξύ του δυαδικού κώδικα κάθε εντολής της γλώσσας μηχανής και ειδικών συντετμημένων λέξεων της αγγλικής γλώσσας. Αυτό επέτρεπε στους πρώτους προγραμματιστές πριν αρκετές δεκαετίες να γράψουν τα πρώτα τους προγράμματα με μια σχετική ευκολία. Πώς όμως αντιλαμβάνεται ο υπολογιστής τις εντολές της συμβολικής γλώσσας; Είναι εξοπλισμένος με ένα πρόγραμμα, τον συμβολομεταφραστή (assembler) που μετατρέπει τις συμβολικές εντολές σε κατάλληλο δυαδικό κώδικα. Όπως και η γλώσσα μηχανής, έτσι και η συμβολική γλώσσα είναι εξειδικευμένη στο συγκεκριμένο τύπο υπολογιστή.
Σε ακόμη ανώτερα επίπεδα βρίσκονται οι διάφορες σύγχρονες γλώσσες προγραμματισμού που λέγονται και γλώσσες υψηλού επιπέδου. Αυτές δικαιολογημένα αποκαλούνται "γλώσσες" γιατί έχουν το δικό τους συντακτικό και τη δική τους γραμματική δηλαδή ένα σύνολο από κανόνες που πρέπει να ακολουθούνται με συνέπεια κατά τη συγγραφή προγραμμάτων. Έχουν επίσης το δικό τους λεξιλόγιο, ένα σύνολο από "δεσμευμένες" λέξεις, ενώ ο χρήστης μπορεί να προσθέσει δικά του ονόματα για τις κάθε είδους ποσότητες και πληροφορίες που υπεισέρχονται στο πρόβλημα και αποτελούν τις *σταθερές* και *μεταβλητές* του προγράμματος. Οι "προτάσεις" που διατυπώνονται σε αυτές τις γλώσσες είναι εντολές προς τον ΗΥ που αναλύονται σε πολλές στοιχειώδεις εντολές σε γλώσσα μηχανής για την ύπαρξη και την εκτέλεση των οποίων δε χρειάζεται να απασχολείται ο προγραμματιστής. Έτσι, για

παράδειγμα, όταν έχουμε να λύσουμε μία εξίσωση δεύτερου βαθμού δε θα γράψουμε από την αρχή τον αλγόριθμο για τις αριθμητικές πράξεις (πρόσθεση, αφαίρεση κλπ) αλλά ο υπολογιστής θα "ξέρει" ότι τα αντίστοιχα σύμβολα (+, - κλπ) ερμηνεύονται ως μια ακολουθία στοιχειωδών εντολών στη γλώσσα μηχανής, που είναι αποθηκευμένες και έτοιμες για χρήση, στην ΚΜΕ. Συνεπώς, εμείς θα γράψουμε μόνο τις βασικές εντολές που αναφέρονται στο χειρισμό των αριθμών που μας δίνονται και τα υπόλοιπα γίνονται από μόνα τους. Οι εντολές των γλωσσών προγραμματισμού υψηλού επιπέδου αντιστοιχούν στις τρεις βασικές δομές εκτέλεσης των αλγορίθμων όσο και σε άλλες παρεμφερείς, συμπληρωματικές και βοηθητικές λειτουργίες (π.χ. δήλωση μεταβλητών = δέσμευση περιοχών της μνήμης).

Και πώς μεταφράζονται τα προγράμματα σε γλώσσα μηχανής; Εδώ υπεισέρχεται ο μεταγλωττιστής (compiler), που είναι το αντίστοιχο του συμβολομεταφραστή, μόνο που είναι ένα πρόγραμμα πιο πολύπλοκο. Ο μεταγλωττιστής διαβάζει το πρόγραμμα που έχουμε συντάξει σε μια γλώσσα προγραμματισμού και παράγει ένα άλλο πρόγραμμα γραμμένο σε δυαδικό κώδικα, το **εκτελέσιμο**, το οποίο και εκτελεί ο υπολογιστής. Ο μεταγλωττιστής μπορεί να διαφέρει από σύστημα σε σύστημα, όμως ένα πρόγραμμα που έχει γραφεί σε μία γλώσσα υψηλού επιπέδου, κατά κανόνα μπορεί να εκτελεστεί απαράλλαχτο ή το πολύ-πολύ με μικρές τροποποιήσεις σε άλλα συστήματα, αρκεί να υπάρχει σε αυτά εγκατεστημένος ο αντίστοιχος μεταγλωττιστής. Δηλαδή τα προγράμματα σε γλώσσες υψηλού επιπέδου χαρακτηρίζονται από *μεταφερσιμότητα* ή *φορητότητα* (portability).

Ο μεταγλωττιστής μπορεί να εντοπίσει ορισμένα λάθη στο πρόγραμμα και να ενημερώσει το χρήστη ώστε να τα διορθώσει. Δε μπορεί να ανακαλύψει βέβαια όλα τα λάθη, δηλαδή τα λάθη λογικής στην κατάστρωση του αλγορίθμου, τα οποία είναι ευθύνη του προγραμματιστή να τα ανακαλύψει.
Ερώτηση: ο μεταγλωττιστής πώς και σε ποια γλώσσα γράφεται; ***Απάντηση***: Σε πολλές γλώσσες είναι δυνατό να γραφεί ένας μεταγλωττιστής, συχνά όμως, χρησιμοποιείται η διαδικασία που είναι γνωστή ως bootstrapping: αρχικά γράφεται ένα μικρό μέρος της γλώσσας που θέλουμε να αναπτύξουμε σε κάποια προϋπάρχουσα, ενδεχομένως και assembly, και στη συνέχεια, αυτός ο μεταγλωττιστής χρησιμοποιείται για να γράψει έναν άλλο πιο προηγμένο στην ίδια του τη γλώσσα όπου προσθέτει περισσότερα στοιχεία.

Αξίζει να πούμε ότι εκτός από τους μεταγλωττιστές υπάρχουν και οι μεταφραστές ή διερμηνευτές (interpreters) που δεν παράγουν εκτελέσιμο πρόγραμμα αλλά διαβάζουν μία-μία τις εντολές του αρχικού προγράμματος, τις μεταφράζουν σε δυαδικό κώδικα που εκτελείται και μετά πάνε στην επόμενη εντολή. Έτσι λειτουργούσε π.χ. η γλώσσα BASIC. Αυτός ο τρόπος δεν είναι τόσο γρήγορος όσο η μεταγλώττιση.

Οι γλώσσες υψηλού επιπέδου χωρίζονται και αυτές σε διάφορες κατηγορίες. Η κατάταξη των γλωσσών με βάση όλα τα δυνατά κριτήρια θα ήταν αρκετά πολύπλοκη καθώς υπάρχουν και πολλές αλληλεπικαλύψεις μεταξύ των διαφόρων κατηγοριών. Μια απλουστευμένη εικόνα, με μερικά γνωστά παραδείγματα, θα μπορούσε να είναι αυτή που παρουσιάζεται στον επόμενο πίνακα. Αξίζει να πούμε ότι οι κατηγορίες αυτές δεν είναι απολύτως ξένες μεταξύ τους. Π.χ. η γλώσσα C++ μπορεί να χρησιμοποιηθεί και για διαδικαστικό προγραμματισμό, αντίθετα από τη Java που είναι καθαρά αντικειμενοστρεφής. Το παρόν μάθημα θα βασιστεί στην υλοποίηση αλγορίθμων με τη βοήθεια της διαδικαστικής γλώσσας C.

Κατηγορία	***Αγγλικός όρος***	***Παραδείγματα γλωσσών***
Διαδικαστικές ή προστακτικές *Βασίζονται σε διαδοχική εκτέλεση εντολών. Οι σύγχρονες διαδικαστικές γλώσσες έχουν εντολές που επιτρέπουν τη σαφή και ευκρινή διατύπωση των βασικών αλγοριθμικών δομών κ«ι ευνοούν το δομημένο προγραμματισμό.*	Procedural, imperative	Fortran77, Quick Basic, C, Pascal, Fortran 90/95 (?)
Αντικειμενοστρεφείς *Βασίζονται σε ορισμό γενικευμένων τύπων ή κατηγοριών (κλάσεων) που περιγράφουν τα δεδομένα ως "αντικείμενα" ικανά να "ανταλλάσουν" πληροφορία μέσω συναρτήσεων-μελών.*	Object oriented	Simula, Smalltalk, C++, Java, Fortran 90/95 (?), Fortran 2003
Συναρτησιακές	Functional	Lisp
Δηλωτικές	Declarative	Prolog

Είχαμε πει στην ενότητα για τους αλγόριθμους, ότι πολύ συχνά συμφέρει να υποδιαιρούμε το πρόβλημα προς επίλυση σε άλλα επιμέρους που είναι πιο εύκολα ή έχουν γνωστή λύση. Οι γλώσσες υψηλού επιπέδου επιτρέπουν τη συγγραφή ξεχωριστών ενοτήτων (**συναρτήσεις**, **ρουτίνες, procedures, υποπρογράμματα**) που να υλοποιούν συγκεκριμένους αλγόριθμους και μετά, τη συνένωση αυτών με τον τρόπο που έχει καθορίσει ο προγραμματιστής σε ένα μεγαλύτερο πρόγραμμα που θα επιτελεί συγκεκριμένες λειτουργίες. Μέσα στις υποενότητες ή υποπρογράμματα, μπορούμε να δομήσουμε το πρόγραμμα έτσι ώστε να αποτελείται από ευανάγνωστα "μπλοκ" εντολών που υλοποιούν στοιχειώδεις δομές (διαδοχή, επιλογή, επανάληψη). Εξ άλλου, ήδη ορισμένοι αλγόριθμοι που εξυπηρετούν τη χρήση του ΗΥ και των περιφερειακών του (π.χ. λήψη πληροφοριών από το πληκτρολόγιο, εκτύπωση αποτελεσμάτων στην οθόνη) είναι προ-μεταγλωττισμένοι και συνδέονται και αυτοί με το κύριο πρόγραμμα για να αποτελέσουν μια ενιαία πρακτική εφαρμογή.

Στην ιδανική περίπτωση, τόσο τα διάφορα υποπρογράμματα, όσο και τα επιμέρους τμήματα που τα αποτελούν έχουν μία "είσοδο" δεδομένων και μία "έξοδο" αποτελεσμάτων, ώστε αν κάποιος διαβάσει ένα τέτοιο πρόγραμμα να μη χρειαστεί να κάνει άλματα και πισωγυρίσματα στο εσωτερικό κάθε ενότητας ή μεταξύ των διαφόρων ενοτήτων. Αυτού του είδους τα "άλματα" ήταν κάτι που εμφανιζόταν συχνά σε παλιότερα προγράμματα, κατά τα πρώτα χρόνια του προγραμματισμού σε γλώσσες υψηλού επιπέδου, καθιστώντας τα δύσκολα στην κατανόηση όσο και στην αναβάθμισή τους. Γι' αυτό το λόγο αναπτύχθηκαν γλώσσες που επιτρέπουν ένα άλλο στυλ προγραμματισμού, το οποίο ευνοεί την υποδιαίρεση κάθε προγράμματος σε "αυτάρκεις" ενότητες με την έννοια που περιγράψαμε, οι οποίες κατόπιν μπορούν να συνδυαστούν εύκολα μεταξύ τους για να δώσουν μια ποικιλία εφαρμογών. Αυτό το ιδανικό προσπαθεί να προσεγγίσει η φιλοσοφία του ***δομημένου προγραμματισμού***.

Μπορούμε να διακρίνουμε "ισχυρή" και "ασθενή" έννοια του δομημένου προγραμματισμού (ή κατά Dijkstra και Knuth, αντιστοίχως). Στην πρώτη περίπτωση προσπαθούμε να κάνουμε υποπρογράμματα και τμήματα κώδικα που να έχουν πάντα μόνο μία είσοδο και μόνο μία έξοδο. Στη δεύτερη περίπτωση, χαλαρώνουμε αυτό τον περιορισμό και επιτρέπουμε περισσότερες εξόδους και ακόμη και την εντολή goto σε ορισμένα πλαίσια (θα παρουσιαστεί σε επόμενο μάθημα) που είναι προγενέστερη των δομημένων γλωσσών και η υπερβολική χρήση της οδηγούσε συχνά σε πολύπλοκα και ακατανόητα προγράμματα. Η ανάγκη για ένα πιο "χαλαρό" ορισμό του δομημένου προγραμματισμού προέκυψε από το γεγονός ότι η απόλυτη προσήλωση στους αρχικούς κανόνες κάποιες φορές οδηγούσε σε στριφνό κώδικα που ακύρωνε τα επιδιωκόμενα πλεονεκτήματα.

Η θεωρητική δικαιολόγηση του δομημένου προγραμματισμού ως εφικτού τρόπου προγραμματισμού στηρίζεται στο Θεώρημα Jacopini. Αυτό πρακτικά λέει ότι κάθε πρόγραμμα μπορεί να αναπτυχθεί αναλυόμενο σε υποπρογράμματα και αυτά με τη σειρά τους σε "μπλοκ" εντολών που βασίζονται στις "τρεις θεμελιώδεις δομές" (διαδοχή, επιλογή, επανάληψη).
Επίσης από πρακτική άποψη μπορούμε να πούμε ότι ο Ιδανικός δομημένος κώδικας είναι αυτός που μπορούμε να διαβάσουμε και να καταλάβουμε χωρίς ούτε μια φορά να γυρίσουμε πίσω ή να προτρέξουμε.

Λογισμικό
Στον ΗΥ δεν αποθηκεύονται μόνο προγράμματα γραμμένα από τους χρήστες σε μια γλώσσα προγραμματισμού και δεδομένα προς επεξεργασία. Όπως ξέρουμε από την εμπειρία μας υπάρχουν πολλά άλλα προγράμματα τα οποία καθιστούν τον ΗΥ κατάλληλο προς χρήση διότι βοηθούν την επικοινωνία με τους χρήστες, αλλά και εξασφαλίζουν την αποδοτικότερη αξιοποίηση των περιφερειακών. Το σύνολό τους αποτελεί το **λογισμικό** (software) του ΗΥ κατ' αντιδιαστολή προς το υλικό (hardware). Αυτό μπορούμε να το κατατάξουμε σε λογισμικό συστήματος και λογισμικό εφαρμογών.
Λογισμικό Συστήματος:
Λειτουργικά Συστήματα: multi-tasking, ενός χρήστη, πολλών χρηστών, γραφικά περιβάλλοντα. Αρμοδιότητα τους είναι η επικοινωνία με το χρήστη και η αποδοτική χρήση των πόρων (μνήμη, επεξεργαστική ισχύς) και περιφερειακών συσκευών.
Προγράμματα εξυπηρέτησης (utilities): Βοηθητικά προγράμματα που εκτελούν βασικές λειτουργίες υπό μορφή εντολών ή κλήσεων με άλλα μέσα (π.χ. ποντίκι), όπως αντιγραφή ή μεταφορά αρχείων, αποθήκευση, σβήσιμο,ταξινόμηση και πολλά άλλα. Συχνά οι χρήστες μπορούν να προμηθευτούν και νέα ή να δημιουργήσουν τα δικά τους βοηθητικά προγράμματα εξυπηρέτησης.
Μεταφραστές γλωσσών: Εκτελούν τη μεταγλώττιση σε γλώσσα μηχανής. Πριν την καθαυτό μεταγλώττιση υπάρχει το στάδιο της **προεπεξεργασίας** (preprocessing). Σε αυτό, το αρχικό κείμενο με τις εντολές, δηλαδή ο **πηγαίος** κώδικας υφίσταται προσθήκες αρχείων που είτε έχουν κατασκευαστεί από το χρήστη είτε προϋπάρχουν στο σύστημα, καθώς επίσης και γίνονται τροποποιήσεις στον κώδικα, σύμφωνα με οδηγίες που έχουν δοθεί με κατάλληλο συμβολισμό από το χρήστη, π.χ. για να κάνουν τον κώδικα ταχύτερο και αποτελεσματικότερο. Μετά ακολουθεί η μεταγλώττιση του τροποποιημένου πηγαίου κώδικα σε δυαδικό κώδικα (**αντικειμενικός**, object, κώδικας), σύνδεση από τον **linker** των επί μέρους ενοτήτων (συναρτήσεων), αλλά και **βιβλιοθηκών** του συστήματος που εκτελούν βασικές λειτουργίες κοινές σε πολλά προγράμματα (χρήση περιφερειακών, στοιχειώδεις μαθηματικές συναρτήσεις κλπ) και για τις οποίες δε θα ήταν σκόπιμο ο προγραμματιστής να απασχολείται κάθε φορά και τέλος, δημιουργία του **εκτελέσιμου αρχείου**.

Λάθη κατά τον προγραμματισμό
Στην ανάπτυξη ενός προγράμματος μπορεί να παρουσιαστούν τριών ειδών λάθη:

- Συντακτικά (αναγνωρίζονται κατά τη μεταγλώττιση και εμποδίζουν την ολοκλήρωσή της),
- Λάθη χρόνου εκτέλεσης (run time errors – διαφεύγουν της μεταγλώττισης και δεν την εμποδίζουν αλλά αναγνωρίζονται κατά την εκτέλεση),
- Λογικά (δε μπορεί να τα εντοπίσει ο μεταγλωττιστής γιατί έχουν να κάνουν με τον ίδιο τον αλγόριθμο, την υλοποίησή του και κατά πόσον αυτός είναι σωστός για το πρόβλημα που μας ενδιαφέρει).

Ο compiler μπορεί να βρει τα δύο πρώτα είδη και να μας δώσει ανάλογα μηνύματα, αλλά το τρίτο είναι ευθύνη του προγραμματιστή. Πρώτα ελέγχονται τα δεδομένα αν είναι σωστά και αν δεν έχουν πρόβλημα τότε το πρόγραμμα έχει λάθος και πρέπει να τροποποιηθεί. Εδώ μπορεί να βοηθήσει και ένα πρόγραμμα διόρθωσης ή αποσφαλμάτωσης (**debugger**) που επιτρέπει τη βήμα-με-βήμα παρακολούθηση της εκτέλεσης.

Σχεδίαση ή Τεχνολογία Λογισμικού (Software Engineering)
Η ανάπτυξη ενός προγράμματος έχει τυποποιηθεί σε βαθμό που να μπορεί να θεωρηθεί και αυτή η ίδια

ένας αλγόριθμος που περιλαμβάνει τα βασικά στοιχεία των διαδοχικών εκτελέσεων, επαναλήψεων και επιλογών, όπως θα εξηγηθεί στη συνέχεια. Ένας από τους πιο καθιερωμένους τέτοιους αλγόριθμους ανάπτυξης λογισμικού βασίζεται στο **ιεραρχικό μοντέλο**. Σύμφωνα με αυτό, πρώτα ορίζουμε σε αδρές γραμμές τη συνολική λειτουργία του προγράμματος. Μετά καθορίζουμε τις υπολειτουργίες ως επιμέρους ενότητες (modules) που μπορούν να σχεδιαστούν χωριστά. Για κάθε ενότητα, η ίδια διαδικασία μπορεί να επαναληφθεί διασπώντας την σε ακόμη βασικότερες και ευκολώτερες στην υλοποίηση υποενότητες. Σε αυτή τη διαδικασία, βασικά εργαλεία είναι τα *διαγράμματα δομής* που απεικονίζουν την ιεραρχική δομή του λογισμικού με το κύριο πρόγραμμα στην κορυφή και από κάτω σε διάφορα επίπεδα, τις υποενότητες αυτού` και τα *διαγράμματα ροής* που δείχνουν τη ροή και τους μετασχηματισμούς των δεδομένων από ενότητα σε ενότητα. Τέλος, χρησιμότατη είναι και η *τεκμηρίωση* του προγράμματος, δηλαδή όλα τα έγγραφα που εξηγούν το πρόβλημα και τη μεθοδολογία επίλυσής του, τη δομή του προγράμματος και του αλγορίθμου που αυτό υλοποιεί, τα διαγράμματα δομής και ροής, ακόμη και τα σχόλια που παρεμβάλλονται στο ίδιο το πρόγραμμα ώστε να είναι ευανάγνωστο για κάθε τρίτον.
Βάσει των παραπάνω, ο αλγόριθμος ανάπτυξης λογισμικού σύμφωνα με το ιεραρχικό μοντέλο είναι σε γενικές γραμμές ο εξής:
1. Ορισμός προβλήματος και απαιτήσεων (τι είσοδο έχουμε και τι έξοδο θέλουμε)
2. Δομή προγράμματος και ροή δεδομένων
3. Ανάλυση σε υποενότητες με τις δικές τους απαιτήσεις σε όρους εξόδου για δεδομένη είσοδο που συνεργάζονται μεταξύ τους αλλά μπορούν να αναπτυχθούν ανεξάρτητα, και μάλιστα από διαφορετικά πρόσωπα που θα εργάζονται παράλληλα.
4. Όσο δεν έχει ολοκληρωθεί το πρόγραμμα ώστε να λειτουργεί σύμφωνα με τις απαιτήσεις:
4-1. Όσο δεν έχουν ολοκληρωθεί οι υποενότητες ώστε να λειτουργούν σύμφωνα με τις επιμέρους απαιτήσεις τους:
4-1-1. Ανάπτυξη κώδικα κάθε ενότητας
4-1-2. Δοκιμή κάθε υποενότητας
4-2. Ολοκλήρωση ανάπτυξης κάθε υποενότητας με τεκμηρίωση
4-3. Συνένωση των ενοτήτων και ανάπτυξη κώδικα κύριου προγράμματος για να συνεργαστούν
4-4. Δοκιμή όλου του προγράμματος.
5. Ολοκλήρωση ανάπτυξης τελικού προγράμματος με τεκμηρίωση.
6. Χρήση προγράμματος

Αυτός ο αλγόριθμος ανάπτυξης μπορεί με τη σειρά του να επαναλαμβάνεται για την ανάπτυξη καινούριων εκδόσεων του προγράμματος με βάση τις προτάσεις για βελτίωση και αναβάθμιση που προκύπτουν κατά τη χρήση του και που επιβάλλουν επιστροφή στο βήμα 1 για επανακαθορισμό του προβλήματος.

Η γλώσσα C
Υψηλού επιπέδου, αλλά με δυνατότητα πρόσβασης σε λειτουργίες χαμηλού επιπέδου. Από τις γλώσσες υψηλού επιπέδου, η C είναι η πιο “κοντινή” στη γλώσσα μηχανής. Ωστόσο, είναι γενική και portable. Δεν εξαρτάται σχεδόν καθόλου από τη συγκεκριμένη πλατφόρμα. Τα προγράμματα γραμμένα σε αυτή, γενικά αναμένεται να τρέχουν παντού. Για τους λόγους αυτούς, είναι ιδανική για την ανάπτυξη εφαρμογών συστήματος που ήταν άλλωστε και ο αρχικός σκοπός για τη δημιουργία της (λειτουργικό σύστημα UNIX).
Οι υποενότητες του προγράμματος που θέλουμε να κάνουμε, στη C ονομάζονται συναρτήσεις (functions). Κάθε πρόγραμμα σε C έχει τουλάχιστον μία συνάρτηση και γενικά μία συνάρτηση είναι το κύριο πρόγραμμα που καλεί και τις υπόλοιπες, σύμφωνα με το ιεραρχικό μοντέλο. Γενικά, οι συναρτήσεις στη C χωρίζονται σε συναρτήσεις βιβλιοθήκης και συναρτήσεις ορισμένες από το χρήστη. Οι πρώτες είναι υπεύθυνες για συχνά επαναλαμβανόμενες τυπικές λειτουργίες ώστε να απαλλαχθεί ο προγραμματιστής από το σχετικό βάρος και να ασχοληθεί αποκλειστικά με τις δεύτερες που θα οδηγήσουν και στη λύση του εκάστοτε προβλήματος.

Ένα απλό πρόγραμμα σε C – γενική μορφή:

```
#include <stdio.h> ← οδηγία προς τον "προεπεξεργαστή" για να περιληφθούν τα περιεχόμενα
                                από αρχείο σχετικό με λειτουργίες εισόδου/εξόδου
int main(void)  ← όνομα συνάρτησης
{       ← έναρξη τμήματος (μπλοκ) κώδικα
     δηλώσεις μεταβλητών;   ← δέσμευση περιοχών μνήμης για αποθήκευση πληροφορίας
   εντολή;                  ← σώμα συνάρτησης· κάθε δήλωση και εντολή τελειώνει με ;
   εντολή;
     εντολή;
} ← τέλος τμήματος (μπλοκ) κώδικα
```

Συγκεκριμένο πρόγραμμα:

```
#include <stdio.h>

void main(void)
{
/* this is a comment! */
    printf ("\nHello world!");
}
```

Αν και δεν ορίσαμε καμία δική μας συνάρτηση, η printf είναι μια συνάρτηση βιβλιοθήκης εντελώς ανάλογη με αυτές που θα μπορούσαμε να ορίσουμε εμείς και με συγκεκριμένο σκοπό (μορφοποιημένη εκτύπωση). Γενικά, όπου υπάρχει κάτι στη μορφή *όνομα*(*κάτι*) είναι κλήση της συνάρτησης *όνομα* με όρισμα *κάτι*.

Στυλ γραψίματος: ισχύει ελεύθερο format αρκεί να μη σπάνε τα tokens (δεσμευμένες λέξεις και ονόματα μεταβλητών και συναρτήσεων). Αλλά πρέπει να προτιμούμε ένα συνεπές και τακτοποιημένο στυλ για να είναι ευανάγνωστο και κατανοητό και να βρίσκονται πιο εύκολα τυχόν σφάλματα (δεν αρκεί ο δομημένος προγραμματισμός! Πρέπει να βοηθάμε κι εμείς!)

Προσοχή! Case sensitive!!! Τα κεφαλαία και τα μικρά γράμματα θεωρούνται διαφορετικά. Αν γράψουμε κάτι, π.χ. όνομα μιας μεταβλητής με συγκεκριμένο συνδυασμό κεφαλαίων και μικρών, αυτό το συνδυασμό πρέπει να διατηρούμε σε όλο το πρόγραμμα, όποτε αναφερόμαστε σε αυτή τη μεταβλητή και όχι να το αλλάζουμε όπως μας έρθει!

Το τμήμα κώδικα αρχίζει με { και τελειώνει με } και δεν πρέπει να συγχέεται με τις ενότητες (συναρτήσεις) γιατί είναι πιο γενικό. Μία συνάρτηση μπορεί να έχει περισσότερα του ενός τμήματα κώδικα που περικλείονται στα δικά τους { και }.

```
/* Σχόλια */
```

Σε ένα πρόγραμμα έχουμε τη δυνατότητα να παρεμβάλλουμε επεξηγηματικά σχόλια ώστε όποιος το διαβάζει να καταλαβαίνει καλύτερα τα σημεία του προγράμματος που η λειτουργία τους δεν είναι προφανής. Αυτά τα σχόλια αγνοούνται από τον υπολογιστή σα να μην υπάρχουν.

Απλός κανόνας: ανοίγουν με /*, κλείνουνε με */ και από εκεί και πέρα, τελείως ελεύθερο φορμάτ και ελευθερία τοποθέτησης με μόνους περιορισμό να μη σπάσουν κάποιο token και να μην είναι ένθετα σε άλλα σχόλια. Π.χ. μπορούμε να γράψουμε σε μια σειρά το σύμβολο αρχής σχολίου /*, να γράψουμε... την ιστορία της ζωής μας σε πολλές επόμενες σειρές και στο τέλος να κλείσουμε το σχόλιο με το σύμβολο τέλους σχολίου */. Απλά δεν επιτρέπεται να βάζουμε σχόλια μέσα σε σχόλια.

Για να αποφύγουμε μερικά συχνά λάθη προσέχουμε τα εξής:

- οι εντολές να τελειώνουν με ;
- να κλείνουν τα άγκιστρα (για κάθε { να υπάρχει και ένα })
- Παρομοίως να κλείνουν τα σχόλια (/* και */) και να μην υπάρχουν ένθετα σχόλια