5 ΕΙΣΑΓΩΓΗ ΣΤΗ ΘΕΩΡΙΑ ΑΛΓΟΡΙΘΜΩΝ

5.1 Εισαγωγή στους αλγορίθμους

5.1.1 Εισαγωγή και ορισμοί

Αλγόριθμος (algorithm) είναι ένα πεπερασμένο σύνολο εντολών οι οποίες εκτελούν κάποιο ιδιαίτερο έργο. Κάθε αλγόριθμος πρέπει να ικανοποιεί 5 κριτήρια:

1. ***Είσοδος***: Πρέπει να υπάρχουν μηδέν ή περισσότερα δεδομένα/στοιχεία τα οποία να δίνονται στον αλγόριθμο από το εξωτερικό περιβάλλον (δεδομένα εισόδου).
2. ***Έξοδος***: Θα πρέπει ο αλγόριθμος να παράγει δεδομένα (αποτελέσματα).
3. ***Σαφήνεια***: Κάθε εντολή του αλγορίθμου θα πρέπει να είναι σαφής (π.χ. a = b ± c, δεν είναι σαφής εντολή).
4. ***Πεπεραστηκότα***: Εάν ανιχνεύσουμε τις εντολές ενός αλγορίθμου, σ' όλες τις περιπτώσεις ο αλγόριθμος θα πρέπει να τελειώνει μετά από ένα πεπερασμένο αριθμό βημάτων.
5. ***Αποτελεσματικότα***: Όλες οι εντολές του αλγορίθμου θα πρέπει να είναι αρκετά βασικές και να εκτελούνται σε πεπερασμένο χρόνο από άνθρωπο με τη χρήση μολυβιού και χαρτιού.

Ένας αλγόριθμος μπορεί να περιγραφεί με πολλούς τρόπους.

1. Στη γλώσσα την οποία ομιλούμε (όπως η Ελληνική), δηλαδή σε φυσική γλώσσα. Σ' αυτήν την περίπτωση θα υπάρχουν ασάφειες επειδή οι ομιλούμενες γλώσσες από τη φύση τους είναι ασαφείς.
2. Ένας πιο βελτιωμένος τρόπος είναι το διάγραμμα ροής (flowchart) στο οποίο διαφορετικά σχήματα παριστούν διάφορα είδη πράξεων. Βέλη δείχνουν την ακολουθία εκτέλεσης των πράξεων. Το διάγραμμα ροής έχει το μειονέκτημα ότι όσο μεγαλώνει γίνεται πιο δυσνόητο λόγω των πολλών σχημάτων και βελών.
3. Ένας πιο βελτιωμένος τρόπος περιγραφής των αλγορίθμων είναι ο ψευδοκώδικας. Οι εντολές του ψευδοκώδικα είναι παρόμοιες μ' αυτές των γλωσσών προγραμματισμού. Βέβαια το ερώτημα που τίθεται είναι γιατί ένας αλγόριθμος να μην γραφτεί κατευθείαν σε κάποια γλώσσα προγραμματισμού αλλά σε ψευδοκώδικα.

5.1.2 Αλγόριθμοι και γλώσσες προγραμματισμού

Οι αλγόριθμοι περιγράφουν τη λύση των προβλημάτων. Όμως οι Η/Υ δεν καταλαβαίνουν τη φυσική γλώσσα την οποία χρησιμοποιήσαμε για την περιγραφή των αλγορίθμων. Πρέπει να γράψουμε τους αλγορίθμους σε γλώσσα η οποία να είναι κατανοητή από τον υπολογιστή. Για τον σκοπό αυτό έχουν κατασκευαστεί πολλές

γλώσσες προγραμματισμού. Οι γλώσσες προγραμματισμού σε αντίθεση με τη φυσική γλώσσα έχουν αυστηρό συντακτικό. Για παράδειγμα, στη φυσική γλώσσα τα παρακάτω έχουν την ίδια σημασία τέλος_για, τέλος για, τέλος για. Σε μια γλώσσα προγραμματισμού αυτό δεν ισχύει. Υπάρχει αυστηρός καθορισμός του συντακτικού. Επίσης, στους αλγορίθμους χρησιμοποιούμε τις μεταβλητές χωρίς να τις έχουμε ορίσει. Δηλαδή, χωρίς να καθορίσουμε αν είναι ακέραιες, πραγματικές, πίνακες ακεραίων κτλ. Σε μια γλώσσα προγραμματισμού πριν χρησιμοποιήσουμε μια μεταβλητή πρέπει να έχει οριστεί ο τύπος της μεταβλητής (εξαίρεση αποτελούν κάποιες script γλώσσες, όπως η php).

Οι γλώσσες προγραμματισμού έχουν ένα μεταγλωττιστή ή ένα μεταφραστή. Αυτός μετατρέπει το πρόγραμμα το οποίο γράφουμε σύμφωνα με το συντακτικό της γλώσσας σε ακολουθία από 0 και 1, την οποία μπορεί στη συνέχεια να εκτελέσει ο υπολογιστής. Η διαδικασία για να γίνει αυτό είναι:

1. Έλεγχος για συντακτικά και σημασιολογικά λάθη.
2. Αν υπάρχουν τέτοια λάθη εμφανίζονται τα κατάλληλα μηνύματα και δεν προχωράει η διαδικασία.
3. Αλλιώς το πρόγραμμα μετατρέπεται σε ακολουθία από 0 και 1. Αυτή η ακολουθία είναι η *γλώσσα μηχανής* την οποία καταλαβαίνει ο υπολογιστής.

5.1.3 Επίλυση προβλημάτων με τον υπολογιστή

Οι υπολογιστές μπορούν να επιλύσουν μόνο τα προβλήματα τα οποία μπορεί να επιλύσει και ο άνθρωπος. Ο υπολογιστής δεν μπορεί να έχει την κρίση που έχει ο άνθρωπος. Το μόνο που μπορεί να κάνει είναι να εκτελεί μια ακολουθία από εντολές. Από την άλλη πλευρά, όμως, έχει τη δυνατότητα να κάνει πολλές πράξεις και να αποθηκεύει και να επεξεργάζεται τεράστιο όγκο δεδομένων.

- Γιατί επιλέγουμε επίλυση με τον υπολογιστή;

- Λόγω των δυνατοτήτων που προσφέρει όσον αφορά:

1. την πολυπλοκότητα των υπολογισμών
2. την επαναληπτικότητα των διαδικασιών
3. την ταχύτητα εκτέλεσης
4. το μεγάλο πλήθος των δεδομένων

Παράδειγμα αλγορίθμου: Θέλουμε να περιγράψουμε έναν αλγόριθμο για το πρόβλημα της απόφασης αν ένας αριθμός είναι πρώτος ή όχι.

Βήμα 1: Ξεκίνα από το 2 και αν το 2 διαιρεί τον αριθμό τότε δεν είναι πρώτος
Βήμα 2: Κάνε το ίδιο για το 3
Βήμα 3: Επανέλαβε το βήμα 2 για όλους τους αριθμούς μέχρι να φτάσεις στον ίδιο τον αριθμό
Βήμα 4: Αν κανείς αριθμός δε διαιρεί τον αριθμό, τότε αυτός είναι πρώτος

5.2 Αλγόριθμοι

5.2.1 Τυπικός Ορισμός

Δίνουμε έναν πιο λεπτομερή ορισμό για το τι είναι αλγόριθμος:

ΟΡΙΣΜΟΣ: *Ο αλγόριθμος είναι μία αλληλουχία από σαφείς και μη διφορούμενες εντολές, που όλες δύναται να εκτελεστούν και να περατωθούν. Με την εκτέλεση του αλγορίθμου παράγεται το επιθυμητό αποτέλεσμα και ο τερματισμός της όλης διαδικασίας επέρχεται σε πεπερασμένο χρόνο.*

Για να μπορεί το κάθε βήμα του αλγορίθμου να είναι σαφή και απλό, θα πρέπει να καθορίσουμε ακριβώς τι μπορεί να είναι το κάθε βήμα. Δηλαδή να καθορίσουμε με ακρίβεια ποιες εντολές μπορεί να περιέχει ένας αλγόριθμος. Ο αλγόριθμος μπορεί να εκτελεί τις εντολές:

1. Σειριακά
2. Αν ισχύει κάποια συνθήκη
3. Επαναληπτικά

5.2.2 Εντολές που χρειαζόμαστε για να περιγράψομε έναν αλγόριθμο

- Εντολές ανάθεσης, π.χ. a = 3, b = 3 + 7, b = c + d
- Αριθμητικές πράξεις, π.χ. a + 31 * c, u * 3
- Λογικές πράξεις, π.χ. a > b, b > 7 + 8, (a > d) || (b < c)
- Εντολές εισόδου εξόδου, π.χ. “Διάβασε a”, “Γράψε d”
- Εντολές αριθμητικών και λογικών πράξεων
- Σειριακή εκτέλεση εντολών, π.χ.:
  1. a = 3
  2. b = 7
- Εντολές επιλογής, π.χ. “Αν (a > 3) τότε k = 9”
- Επαναληπτικές εντολές, π.χ.:
  1. Όσο (a > 0) επανέλαβε
  2. Διάβασε b
  3. c = c + b
  4. Τέλος

5.2.3 Μεταβλητές και σταθερές

Οι μεταβλητές είναι αυτές που μπορούν αλλάξουν τιμή κατά τη διάρκεια εκτέλεσης ενός προγράμματος, π.χ a = 4, a = 8, το α αλλάζει τιμή. Οι σταθερές δεν μπορούν να αλλάξουν τιμή κατά την διάρκεια της εκτέλεσης ενός προγράμματος.

Η κάθε μεταβλητή έχει έναν τύπο, π.χ. ακέραιο, και μπορεί να πάρει οποιαδήποτε τιμή (που ανήκει στον τύπο αυτόν). Δηλαδή μία ακέραια μεταβλητή μπορεί να πάρει μόνο ακέραιες τιμές. Π.χ. a=4 (σωστό), a=54 (σωστό), a='η' (λάθος, το 'η' είναι χαρακτήρας).

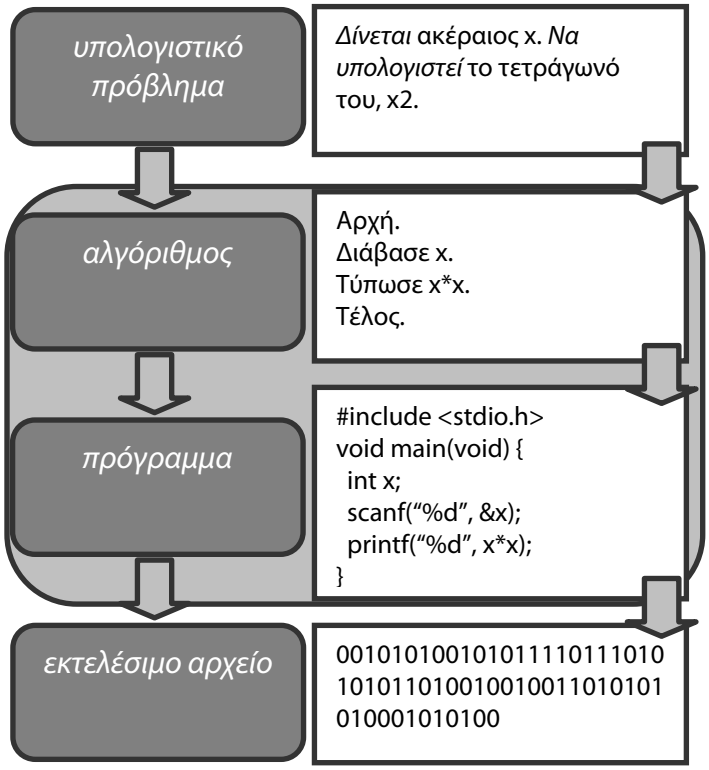
5.2.4 Προγραμματισμός

Προγραμματισμός είναι η διαδικασία του να επινοήσουμε αλγόριθμο και να συντάξουμε πρόγραμμα. **Πρόγραμμα** είναι ο αλγόριθμος σε *τυπική* μορφή (δηλαδή σε μορφή έτοιμη να μεταφραστεί σε μορφή κατανοητή από τον υπολογιστή).

Μετάφραση (compilation) είναι η μετατροπή του πηγαίου κώδικα σε εκτελέσιμο αρχείο.

Εκτελέσιμο αρχείο είναι ο αλγόριθμος σε μορφή κατανοητή από τον υπολογιστή.

Το σύνολο των γραμματικών και συντακτικών κανόνων που ακολουθούμε κατά την συγγραφή ενός προγράμματος λέγεται **γλώσσα προγραμματισμού (programming language)**. Το ειδικό λογισμικό που επιτελεί την μετάφραση λέγεται **μεταφραστής (compiler)**.



Εικόνα 1: Διαδικασία επίλυσης ενός προβλήματος με προγραμματιστικό τρόπο

5.2.5 Κύκλος ανάπτυξης του προγράμματος

Γενικά, η πορεία μέχρι το εκτελέσιμο αρχείο είναι πιο σύνθετη από την απλή ακολουθία «*προγραμματισμός + μετάφραση*». Είναι μια διαδικασία που αποτελείται από πολλά στάδια και ενδέχεται να μην τερματίσει ποτέ. Η διαδικασία αυτή ονομάζεται **κύκλος ανάπτυξης προγράμματος (program development cycle)**. Τον κύκλο ανάπτυξης συνιστούν τα εξής 6 στάδια:

5.2.5.1 Στάδιο 1: Περιγραφή

Τις περισσότερες φορές, το υπολογιστικό πρόβλημα δε μας δίνεται στην απλή μορφή: «*Δίνεται… . Να βρεθεί… .*». Αντιθέτως, αυτό που έχουμε μπροστά μας είναι ένα πρόβλημα του πραγματικού κόσμου. Πρέπει οι ίδιοι να διακρίνουμε ποια συστατικά αυτού του προβλήματος είναι σημαντικά και να τα καταγράψουμε με σαφήνεια.

- Ποια είναι τα δεδομένα (***είσοδος***, *input*);
- Ποια αποτελέσματα πρέπει να προκύψουν (***έξοδος***, *output*);
- Εκτός από τον προφανή περιορισμό (να προκύπτουν πάντα τα σωστά αποτελέσματα που αντιστοιχούν στα δεδομένα), μήπως πρέπει να ικανοποιούνται και άλλοι, πρόσθετοι περιορισμοί;

Παράδειγμα: Μόλις τελείωσε το εξάμηνο και θέλουμε να μάθουμε ποιος ήταν ο μέγιστος βαθμός στο ΕΠΠ004. Ποιο ακριβώς είναι το υπολογιστικό πρόβλημα;

- ***Είσοδος***:
  - Μια ακολουθία *μη αρνητικών* αριθμών x1, x2, …, xN από το πληκτρολόγιο (σήμα τέλους: αριθμός < 0).
- ***Έξοδος***:
  - Ο μεγαλύτερος από τους δεδομένους αριθμούς, δηλ. ο max(x1, x2, …, xN), τυπωμένος στην οθόνη.
- ***Περιορισμοί***: Το πρόγραμμα θα πρέπει να λειτουργεί γρήγορα.

5.2.5.2 Στάδιο 2: Ανάλυση

Τώρα που το υπολογιστικό πρόβλημα είναι απολύτως σαφές, προσπαθούμε να επινοήσουμε κάποια λύση. Λύση υπολογιστικού προβλήματος = αλγόριθμος που λύνει το υπολογιστικό πρόβλημα. Ίσως υπάρχουν πολλές λύσεις, δηλαδή πολλοί αλγόριθμοι που λύνουν το ίδιο πρόβλημα. Για κάθε λύση που σκεφτόμαστε, προσπαθούμε να βεβαιωθούμε ότι σε κάθε είσοδο θα παράγει τη σωστή έξοδο και ότι θα ικανοποιεί τους περιορισμούς. Διερευνούμε το ενδεχόμενο να υπάρχουν περισσότερες από μία λύσεις. Αν όντως επινοήσουμε περισσότερες από μία, τότε πρέπει να επιλέξουμε τη βέλτιστη.

Στο παράδειγμά μας, μια λύση είναι η εξής:

Βήμα 1: Διαβάζουμε τους x_1, x_2, ..., x_N από το πληκτρολόγιο και τους αποθηκεύουμε

Βήμα 2: Έπειτα, για κάθε x_i ελέγχουμε μήπως είναι μέγιστος, δηλαδή:
i. Διατρέχουμε τους άλλους και ελέγχουμε αν είναι όλοι μικρότεροι ή ίσοι του x_i
Βήμα 3: Αν ναι, τότε τυπώνουμε τον x_i στην οθόνη και τερματίζουμε

Είναι αυτή η μέθοδος ορθή λύση; Παράγει το σωστό αποτέλεσμα για οποιεσδήποτε τιμές των $x_1, x_2, \ldots, x_N$; Η απάντηση είναι ΝΑΙ.

Μια άλλη λύση είναι η εξής:

Βήμα 1: Διαβάζομε τους $x_1, x_2, ..., x_N$ από το πληκτρολόγιο
Βήμα 2: Ανά πάσα στιγμή παρακολουθούμε ποιός ήταν μέγιστος από τους μέχρι τώρα διαβασμένους
Βήμα 3: Στο τέλος τυπώνουμε στην οθόνη αυτόν που θυμόμαστε για μέγιστο

Είναι αυτή η μέθοδος ορθή λύση; Παράγει το σωστό αποτέλεσμα για οποιεσδήποτε τιμές των $x_1, x_2, \ldots, x_N$; Η απάντηση είναι ΝΑΙ.

Ποια από τις δύο λύσεις είναι καλύτερη;

Η δεύτερη λύση διατρέχει τους αριθμούς μόνο 1 φορά. Αντιθέτως, η πρώτη λύση ενδέχεται να τους διατρέχει πολλές φορές (στη χειρότερη περίπτωση, αν ο μέγιστος έχει δοθεί τελευταίος, οι αριθμοί διατρέχονται Ν φορές!). Άρα, η δεύτερη λύση είναι ταχύτερη και επομένως πρέπει να επιλέξουμε αυτήν.

5.2.5.3 Στάδιο 3: Σχεδίαση

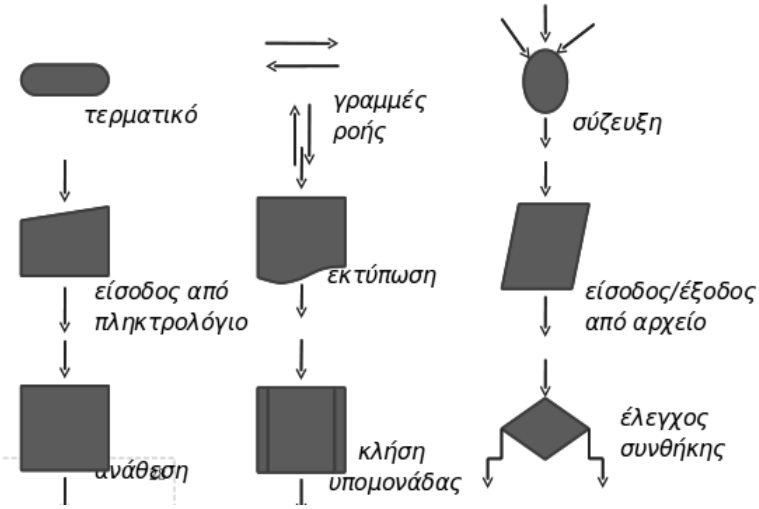
Τώρα που γνωρίζουμε ποια λύση θέλουμε να υλοποιήσουμε, αναπτύσσουμε τον αλγόριθμο λεπτομερώς:

- Μετατρέπουμε τη λύση που έχουμε περιγράψει σε ***διάγραμμα ροής*** **(*flow chart*)**. Αυτό αναπαριστά σχηματικά τη ροή του ελέγχου στον αλγόριθμό μας.
- Μετατρέπουμε το διάγραμμα ροής σε ***ψευδοκώδικα*** **(*pseudocode*)**. Αυτός αναπαριστά τον αλγόριθμό μας ως ακολουθία βημάτων, καθένα από τα οποία περιγράφεται μέσω ενός μείγματος λέξεων της ελληνικής ή της αγγλικής γλώσσας και κάποιων εντολών που είναι κοινές σε πολλές γλώσσες προγραμματισμού.

Ακολουθεί ο ψευδοκώδικας για τη δεύτερη λύση:

```
1. Αρχή.
2. max ← -1 .
3. Επανάληψη:
4.       Διάβασε x .
5.       Αν max < x τότε max ← x .
6. Όσο x ≥ 0 .
7. Τύπωσε max .
8. Τέλος.
```

Στην Εικόνα 2 απεικονίζονται τα βασικά σύμβολα που χρησιμοποιούνται για το σχεδιασμό των διαγραμμάτων ροής, ενώ στον Πίνακα Ι οι βασικές εντολές σε ψευδοκώδικα.



Εικόνα 1: Βασικά Σύμβολα του Διαγράμματος Ροής

Αρχή / Τέλος	τερματικό
Διάβασε …	είσοδος από πληκτρολόγιο
Τύπωσε …	εκτύπωση
Διάβασε/Γράψε …	είσοδος /έξοδος από αρχείο
… ← …	ανάθεση
Αν … Τότε …	έλεγχος συνθήκης
Αν … Τότε … Αλλιώς …	έλεγχος συνθήκης
Επανάληψη … Όσο …	βρόχος επανάληψης
Όσο … Επανέλαβε …	βρόχος επανάληψης
Για … Από … Μέχρι … Επανέλαβε … Τέλος_για	βρόχος επανάληψης

Πίνακας Ι: Βασικές εντολές σε ψευδοκώδικα

5.2.5.4 Στάδιο 4: Κωδικοποίηση

Από τον ψευδοκώδικα, συντάσσουμε πρόγραμμα σε κάποια γλώσσα προγραμματισμού (π.χ. C, Java, κ.α.). Μέσω του μεταφραστή μετατρέπουμε το πρόγραμμα σε γλώσσα, η οποία είναι αναγνωρίσιμη από τη μηχανή. Στο στάδιο αυτό γίνεται ο έλεγχος και η διόρθωση των ***συντακτικών***, δηλαδή των λαθών που οφείλονται σε λανθασμένη χρήση των κανόνων της γλώσσας προγραμματισμού.

Ο μεταφραστής δέχεται στην είσοδο ένα αρχείο κειμένου (π.χ.: findmax.c) που περιέχει το πρόγραμμα και παράγει στην έξοδο ένα εκτελέσιμο αρχείο (findmax.exe), ή μία αναφορά με όλα τα συντακτικά λάθη που εντόπισε.

Παρακάτω ακολουθεί ο κώδικας σε C για τη δεύτερη λύση και το εκτελέσιμο αρχείο:

```
1. #include <stdio.h>
2. void main(void) {
3. int x, max = -1;
4.    do {
5.       scanf("%d", &x);
6.       if (max<x)  max=x;
7.    } while (x>=0);
8.    printf("%d\n", max);
9. }
```

Μετάφραση

```
00101010010101101111011101
10010110101100100101100110
10110010100101001001001010
10101001000000011111101011
11001001110010011001001001
00000001111110101111011010
01001001110010010010010010
01110010010101010010001000
00011111111111011111110111
10101001010101001001001011
10000100100101010101010100
00001111110101111100110000
01111101010100101100 1001
```

5.2.5.5 Στάδιο 5 Έλεγχος

Στο παράδειγμά μας, ίσως κατά την κωδικοποίηση να πληκτρολογήσαμε “max>x” αντί για “max<x”, δηλαδή:

```
5.  scanf("%d", &x);
6.  if (max>x)  max=x;
```

Ο μεταφραστής δεν μπορεί να ανιχνεύσει τέτοια λογικά σφάλματα. Εντοπίζονται μόνο στο στάδιο του ελέγχου.

5.2.5.6 Στάδιο 6: Συντήρηση

Συντάσσουμε ***τεκμηρίωση*** **(*documentation*)**, δηλαδή ένα εγχειρίδιο που να εξηγεί την χρήση του προγράμματος. Εγκαθιστούμε το πρόγραμμα και αρχίζουμε να το χρησιμοποιούμε. Παρακολουθούμε τη χρήση του προγράμματος και ενημερωνόμαστε για τυχόν νέες απαιτήσεις ή σφάλματα που ανακύπτουν. Όποτε είναι απαραίτητο, τροποποιούμε το πρόγραμμα καταλλήλως.