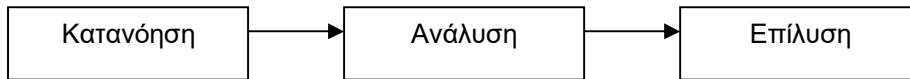


1.1 Η έννοια πρόβλημα

Πρόβλημα : Εννοούμε μία κατάσταση που χρήζει αντιμετώπισης (επίλυσης), η δε λύση της δεν είναι γνωστή και ούτε προφανής.

Στάδια αντιμετώπισης ενός προβλήματος :



1.2 Κατανόηση προβλήματος

Για την αντιμετώπιση κάθε προβλήματος πρέπει προηγουμένως να έχει προηγηθεί η **κατανόησή** του. Αποτελεί συνάρτηση δυο παραγόντων:

1. Σωστή διατύπωση εκ μέρους του δημιουργού του,
2. Σωστή ερμηνεία από αυτόν που θα το επιλύσει.

Ορισμοί:

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

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

Ο όρος **επεξεργασία δεδομένων** δηλώνει εκείνη τη διαδικασία κατά την οποία ένας "μηχανισμός" δέχεται δεδομένα, τα επεξεργάζεται σύμφωνα με έναν προκαθορισμένο τρόπο και αποδίδει πληροφορίες.

1.3 Ανάλυση προβλήματος

Σημαίνει ότι ξεκινάμε να αποκαλύπτουμε τη δομή του προβλήματος, δηλαδή να χωρίσουμε το πρόβλημα σε μικρότερα και απλούστερα υπο-προβλήματα, καθένα από τα οποία λύνεται ευκολότερα.

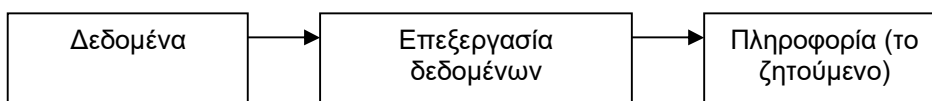
Δομή προβλήματος: Εννοούμε τα συστατικά μέρη από τα οποία συντίθεται, δηλαδή τα επιμέρους τμήματα που το αποτελούν καθώς και τον τρόπο με τον οποίο αυτά συνδέονται μεταξύ τους.

Η ανάλυση μπορεί να γίνει με δομημένο λεκτικό τρόπο ή καλύτερα χρησιμοποιώντας ένα ιεραρχικό διάγραμμα.

1.4 Επίλυση προβλήματος

Λύση του προβλήματος, με επίλυση των αρχικών προβλημάτων. Για τη σωστή επίλυση του προβλήματος βασική προϋπόθεση είναι ο καθορισμός απαιτήσεων.

Καθορισμός απαιτήσεων : Σημαίνει να α) προσδιορίσουμε τα **δεδομένα** που παρέχονται β) να προσδιορίσουμε τα **ζητούμενα**, δηλαδή τι περιμένουμε σαν αποτέλεσμα.



- Τα δεδομένα είναι οτιδήποτε στοιχείο μπορεί να γίνει αντιληπτό από τον άνθρωπο.
- Η πληροφορία είναι γνωσιακό στοιχείο που προκύπτει από την επεξεργασία δεδομένων.
- Η επεξεργασία δεδομένων είναι μία διαδικασία κατά την οποία ένας μηχανισμός (π.χ. ο Η/Υ) κάνει πράξεις και υπολογισμούς στα δεδομένα ώστε να προκύψει μία χρήσιμη πληροφορία.

Για παράδειγμα, η μέση θερμοκρασία κατά το μήνα Ιανουάριο (πληροφορία) προκύπτει από τον υπολογισμό του μέσου όρου (επεξεργασία) των ημερήσιων θερμοκρασιών του μήνα (δεδομένα).

1.5 Κατηγορίες προβλημάτων

A) Με κριτήριο τη δυνατότητα επίλυσης τα διακρίνουμε σε :

Επιλύσιμα

Η λύση τους είναι γνωστή κι έχει διατυπωθεί (π.χ το πρόβλημα του υπολογισμού του εμβαδού του κύκλου).

Ανοικτά

Η λύση τους δεν έχει βρεθεί αλλά παράλληλα δεν έχει αποδειχτεί ότι δεν επιδέχονται λύση (π.χ. το πρόβλημα της ενοποίησης των 4 δυνάμεων της φυσικής).

Άλυτα

Έχουμε παραδεχτεί ότι δεν υπάρχει λύση (πχ. Ο τετραγωνισμός του κύκλου ταξίδι στο χρόνο).

B) Με κριτήριο το βαθμό δόμησης των λύσεών τους τα διακρίνουμε σε:

Δομημένα

Η λύση τους είναι μία αυτοματοποιημένη διαδικασία. (π.χ. η επίλυση της δευτεροβάθμιας εξίσωσης)

Ημιδομημένα

Η λύση τους μπορεί να προέλθει από πλήθος πιθανών λύσεων που εμείς επιλέγουμε (πχ. Το πρόβλημα του πώς θα πάμε εκδρομή σε ένα μέρος. Μπορούμε να διαλέξουμε ένα πλήθος από πιθανούς τρόπους-τραίνο, αεροπλάνο κλπ).

Αδόμητα

Η λύση τους δεν είναι αυτοματοποιημένη, δηλαδή δεν μπορούμε να βρούμε ένα συγκεκριμένο τρόπο λύσης. Για την επίλυση βασίζομαστε στην ανθρώπινη διαίσθηση και εμπειρία (πχ ο τρόπος οργάνωσης ενός πάρτι).

Γ) Με κριτήριο το είδος επίλυσης τα χωρίζουμε σε:

Απόφασης

Η λύση σε αυτά τα προβλήματα είναι του τύπου «Ναι» και «Όχι». (π.χ. ο αριθμός 101 είναι πρώτος;)

Υπολογιστικά

Για την επίλυση απαιτείται η διενέργεια υπολογισμών. (π.χ. για τον αριθμό N να βρεθεί το παραγοντικό του N!).

Βελτιστοποίησης

Η λύση που ζητάμε είναι το βέλτιστο αποτέλεσμα για τα συγκεκριμένα δεδομένα. (π.χ. ποιος είναι ο συντομότερος δρόμος για να πάμε σε ένα μέρος).

1.6 Πρόβλημα και υπολογιστές

Οι λόγοι που αναθέτουμε την επίλυση ενός προβλήματος σε Η/Υ είναι:

- Η πολυπλοκότητα των υπολογισμών.
- Η ταχύτητα εκτέλεσης των πράξεων.
- Ο μεγάλος όγκος των δεδομένων.
- Η επαναληπτικότητα των διαδικασιών.

Ο Η/Υ, στα κυκλώματά του, εκτελεί 3 μόνο λειτουργίες:

- Πρόσθεση (όλες οι άλλες αριθμητικές πράξεις γίνονται μέσω πρόσθεσης!).
- Σύγκριση (που αποτελεί βασική λειτουργία των λογικών πράξεων).
- Μεταφορά των δεδομένων.

Γενικώς η ικανότητα του Η/Υ εκφράζεται σε ποσοτικό επίπεδο ενώ η ικανότητα του ανθρώπου σε ποιοτικό επίπεδο (Φανταστείτε πόσους υπολογισμούς κάνει ο Η/Υ στο σκάκι για να βρει μία καλή κίνηση ενώ ο άνθρωπος το κάνει με πολύ λιγότερους).

2.1 Τι είναι αλγόριθμος

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

Τα πέντε κριτήρια που πρέπει να ικανοποιεί ένας αλγόριθμος είναι:

Είσοδος

Καμία, μία ή περισσότερες τιμές δεδομένων δίνονται ως είσοδοι.

Έξοδος

Πρέπει να δημιουργεί μία τιμή δεδομένων ως αποτέλεσμα.

Καθοριστικότητα

Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσης της. (σαφήνεια, λεπτομερής και πληρότητα)

Περατότητα

Να τελειώνει, φτάνοντας στο επιθυμητό αποτέλεσμα, σε πεπερασμένο αριθμό βημάτων ή χρόνο. Αν δεν συμβαίνει αυτό τότε είναι απλώς υπολογιστική διαδικασία.

Αποτελεσματικότητα

Μετά το πέρας της εκτέλεσης των βημάτων να έχει επιτευχθεί ο στόχος - το ζητούμενο. (αποτελεσματικότητα και πραγματοποιήσιμος)

2.3 Περιγραφή και αναπαράσταση αλγορίθμων

Οι τρόποι αναπαράστασης ενός αλγορίθμου είναι:

- 1) **Με ελεύθερο κείμενο:** Αποτελεί έναν αδόμητο τρόπο περιγραφής, που μπορεί να καταστρατηγήσει εύκολα τα κριτήρια του σωστού αλγορίθμου (ασάφειες, μη εκτελεσιμότητα κλπ).
- 2) **Με διαγραμματικές τεχνικές:** Αποτελεί έναν γραφικό τρόπο παρουσίασης των εντολών. Η πιο γνωστή τεχνική είναι το **διάγραμμα ροής** (flow chart) που παλαιότερα ήταν αρκετά δημοφιλής.
- 3) **Με φυσική γλώσσα κατά βήματα:** Χρειάζεται προσοχή διότι μπορεί να παραβιαστεί το κριτήριο της πληρότητας. Δεν είναι προτιμητέος τρόπος.
- 4) **Με κωδικοποίηση:** Στην περίπτωση αυτή, δημιουργούμε τις εντολές κατευθείαν σε μία γλώσσα προγραμματισμού.

Σύμβολα διαγράμματος 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

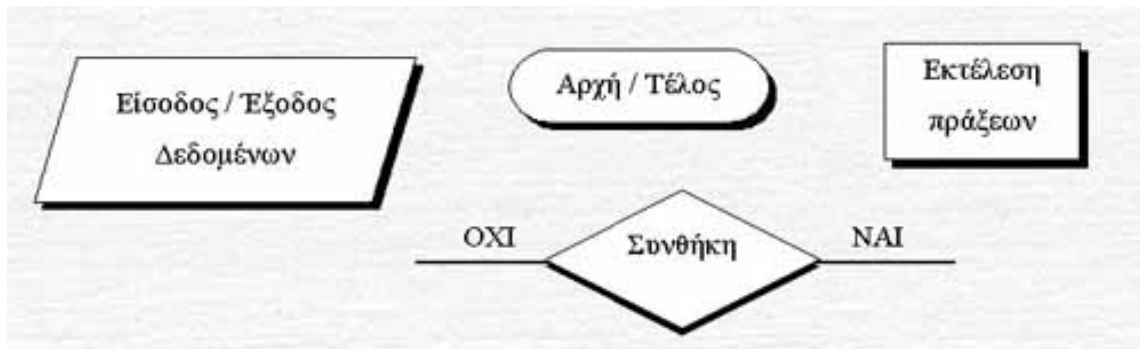
Ορθογώνιο

που δηλώνει την εκτέλεση μίας ή περισσότερων πράξεων

πλάγιο παραλληλόγραμμο

δηλώνει είσοδο ή έξοδο στοιχείων

Το επόμενο σχήμα αποτυπώνει όλα αυτά τα σύμβολα.



6.1 Η έννοια του προγραμματισμού

Η επίλυση ενός προβλήματος με τον υπολογιστή περιλαμβάνει, όπως έχει ήδη αναφερθεί, τρία εξίσου σημαντικά στάδια.

- Τον ακριβή προσδιορισμό του προβλήματος.
- Την ανάπτυξη του αντίστοιχου αλγορίθμου.
- Τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή.

Ο προγραμματισμός ασχολείται με το τρίτο αυτό στάδιο, τη δημιουργία του προγράμματος δηλαδή του συνόλου των εντολών που πρέπει να δοθούν στον υπολογιστή, ώστε να υλοποιηθεί ο αλγόριθμος για την επίλυση του προβλήματος.

Ο **προγραμματισμός** είναι η διατύπωση του αλγορίθμου σε μορφή κατανοητή από τον Η/Υ ώστε να τον εκτελέσει («τρέξει» όπως λέμε στην ορολογία της Πληροφορικής).

Η διατύπωση γίνεται χρησιμοποιώντας μία γλώσσα προγραμματισμού.

6.2 Ιστορική αναδρομή

Κατηγορίες γλωσσών προγραμματισμού

Γλώσσα μηχανής

Το πρόγραμμα περιέχει εντολές που είναι σε δυαδική μορφή, άμεσα κατανοητή από τον Η/Υ (όχι όμως από τον άνθρωπο). Δηλαδή, το πρόγραμμα αποτελείται από ακολουθίες 0 και 1 π.χ.

```
10101000 00001010
11000000 00000001
.....
```

Πλεονεκτήματα:

- Ταχύτατη εκτέλεση των εντολών.
- Δεν απαιτείται μεταφραστικό πρόγραμμα.

Μειονεκτήματα:

- Το γράψιμο του προγράμματος είναι μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία.
- Απαιτείται βαθιά γνώση του υλικού και της αρχιτεκτονικής του Η/Υ.

Το πρόγραμμα «τρέχει» μόνο στο συγκεκριμένο τύπο του Η/Υ.

Γλώσσες χαμηλού επιπέδου ή Συμβολικές γλώσσες

Οι εντολές που είναι σε μορφή 0 και 1 αντικαθίστανται από μνημονικά (συμβολικά) ονόματα. Για παράδειγμα, η εντολή 100001100 αντικαθίστανται από το ADD. Ένα δείγμα χρήσης θα ήταν:

INDEX =\$01	{βάλει στην INDEX την τιμή 1}
ADD INDEX	{πρόσθεσε την τιμή της INDEX στον συσσωρευτή}
LDA #10	{φόρτωσε στο συσσωρευτή την τιμή 10}
CLA	{καθάρισε το συσσωρευτή}
.....	

Πλεονεκτήματα:

- Ταχύτατη εκτέλεση των εντολών.
- Η μορφή του προγράμματος είναι καλύτερα κατανοητή από τον άνθρωπο σε σχέση με τη γλώσσα μηχανής.

Μειονεκτήματα:

- Η αντιστοιχία 1 προς 1 με τις εντολές της γλώσσας παρέμεινε.
Δε διαθέτει εντολές πιο σύνθετων λειτουργιών οδηγώντας σε μακροσκελή προγράμματα
- Απαιτείται η χρήση ενός μεταφραστικού προγράμματος ώστε οι συμβολικές εντολές να μετατραπούν στις αντίστοιχες δυαδικές. Το ειδικό αυτό πρόγραμμα ονομάζεται **συμβολομεταφραστής (assembler)**.
- Το γράψιμο του προγράμματος εξακολουθεί να είναι μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία.
- Απαιτείται βαθιά γνώση της αρχιτεκτονικής του Η/Υ.
- Το πρόγραμμα «τρέχει» μόνο στο συγκεκριμένο τύπο του Η/Υ.

Γλώσσες υψηλού επιπέδου

Λέγονται έτσι διότι τα προγράμματα διατυπωμένα σε μία τέτοια γλώσσα είναι άμεσα κατανοητά από τον άνθρωπο (αλλά όχι από τον Η/Υ) αφού χρησιμοποιείται μία γλώσσα που είναι αρκετά περιγραφική όπως μία φυσική γλώσσα. Παράδειγμα,

```
INPUT "Δώσε την τελική τιμή" ; N
SUM = 0
For INDEX = 1 to N
    SUM = SUM + INDEX
Next
```

Πλεονεκτήματα:

- Η μορφή του προγράμματος είναι εύκολα κατανοητή από τον άνθρωπο σε σχέση με τη γλώσσα μηχανής ή τη συμβολική γλώσσα.
- Το γράψιμο του προγράμματος δεν είναι πλέον μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία όπως συμβαίνει με τη γλώσσα μηχανής ή τη συμβολική γλώσσα.
- Δεν απαιτείται σχεδόν καμία γνώση της αρχιτεκτονικής του Η/Υ. Συνεπώς, είναι ανεξάρτητα από την αρχιτεκτονική του Η/Υ.
- Το πρόγραμμα «τρέχει» σε όλους τους τύπους Η/Υ αρκεί να υπάρχει το κατάλληλο μεταφραστικό πρόγραμμα. Συνεπώς ένα χαρακτηριστικό τους είναι η **μεταφερισιμότητα**, δηλαδή ένα πρόγραμμα υψηλού επιπέδου να εκτελείται, με ελάχιστες μετατροπές, σε πολλούς τύπους Η/Υ.
- Η εκμάθηση της γλώσσας είναι εύκολη.
- Η διόρθωση λαθών και η συντήρηση των προγραμμάτων είναι ευκολότερη.

Μειονεκτήματα:

- Απαιτείται η χρήση ενός μεταφραστικού προγράμματος ώστε οι εντολές να μετατραπούν σε πολλές δυαδικές εντολές (δεν έχουμε εδώ αντιστοιχία 1 προς 1). Έχουμε δύο ειδών μεταφραστικά προγράμματα: τους **μεταγλωττιστές (compilers)** και τους **διερμηνείς (interpreters)**.

Το πρόγραμμα «τρέχει» πιο αργά σε σχέση με τα προγράμματα των συμβολικών γλωσσών ή της γλώσσας μηχανής.

Κατηγοριοποίηση γλωσσών υψηλού επιπέδου:

- ο **Διαδικασιακές ή αλγοριθμικές γλώσσες (Procedural)**: λέγονται έτσι διότι επιτρέπουν την εύκολη υλοποίηση αλγορίθμων π.χ. Pascal, Basic.
- ο **Αντικειμενοστραφείς γλώσσες (object – oriented)** π.χ. C++
- ο **Συναρτησιακές γλώσσες (functional)** π.χ. LISP
- ο **Μη-διαδικασιακές γλώσσες** π.χ. PROLOG
- ο **Γλώσσες ερωταπαντήσεων (Query languages) ή 4ης γενιάς** π.χ. SQL

Μία άλλη κατηγοριοποίηση των γλωσσών υψηλού επιπέδου είναι η εξής:

- ο **Γλώσσες γενικής χρήσης**: Σκοπός τους είναι να επιλύουν πάσης φύσεως προβλήματα (αριθμητικά, εμπορικά, επιστημονικά). Τέτοιες είναι η Basic, Pascal. Μερικές γλώσσες, όμως, έχουν δημιουργηθεί αποκλειστικά για να επιλύουν ευκολότερα συγκεκριμένους τύπους προβλημάτων όπως:
 - **Γλώσσες επιστημονικής κατεύθυνσης**: π.χ. FORTRAN
 - **Γλώσσες εμπορικής κατεύθυνσης**: π.χ. COBOL.
- ο **Γλώσσες προγραμματισμού συστημάτων** π.χ. C
- ο **Γλώσσες τεχνητής νοημοσύνης** π.χ. LISP, PROLOG.
- ο **Γλώσσες ειδικής χρήσης**. Σκοπός τους είναι να επιλύουν ειδικού τύπου προβλήματα όπως διαχείριση Βάσεων Δεδομένων κ.α. π.χ. SQL.

A. Διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης

FORTRAN (1957)

Κατάλληλη για επίλυση επιστημονικών προβλημάτων (αριθμητικές εφαρμογές).

COBOL (1960)

Κατάλληλη για επίλυση εμπορικών προβλημάτων (εφαρμογές μισθοδοσίας, λογιστικές κλπ).

ALGOL (1960)

Δημιουργήθηκε με σκοπό την ανάπτυξη προγραμμάτων κυρίως για επιστημονικές εφαρμογές ως ανταγωνιστική της Fortran. Δεν χρησιμοποιήθηκε, όμως, ευρέως στην πράξη. Πρωτοπαρουσίασε τις αρχές του δομημένου προγραμματισμού.

PL/1 (μέσα '60)

Προσπάθησε να συνδυάσει τις δυνατότητες των γλωσσών προσανατολισμένων για εμπορικές και επιστημονικές εφαρμογές χωρίς όμως να γνωρίσει επιτυχία.

BASIC (μέσα '60)

Δημιουργήθηκε με σκοπό την εκπαίδευση των αρχάριων στον προγραμματισμό. Σκοπός της BASIC είναι να γράφονται μικρά προγράμματα που κατόπιν εκτελούνται με τη βοήθεια διερμηνέα (interpreter). Στις μέρες μας αποτελεί μία πανίσχυρη, γενικής χρήσης γλώσσα.

PASCAL (1970)

Γλώσσα γενικής χρήσης. Στηρίχθηκε στην ALGOL. Είναι η καταλληλότερη γλώσσα για να μάθει κάποιος δομημένο προγραμματισμό.

C (αρχές '70)

Περιέχει αρκετά κοινά χαρακτηριστικά με τη Pascal για την ανάπτυξη δομημένων εφαρμογών αλλά παράλληλα ενσωματώνει και χαρακτηριστικά γλώσσας χαμηλού επιπέδου. Θεωρείται κατάλληλη για την ανάπτυξη λειτουργικών συστημάτων (π.χ. Unix)

dBase (τέλη '70)

Διαχείριση αρχείων Βάσεων Δεδομένων

Ada (1979)

Γλώσσα γενικής χρήσης που δίνει έμφαση στο θέμα της αξιοπιστίας των προγραμμάτων. Γι' αυτό χρησιμοποιήθηκε πρωτίστως για στρατιωτικές εφαρμογές.

B. Αντικειμενοστραφείς γλώσσες κατά χρονολογική σειρά εμφάνισης

Smalltalk (αρχές '80)

Η πρώτη αντικειμενοστραφής γλώσσα με ολοκληρωμένο μάλιστα περιβάλλον ανάπτυξης προγραμμάτων.

C++ (τέλη '80)

Αποτελεί μία μετεξέλιξη της C στο χώρο του αντικειμενοστραφούς προγραμματισμού και χρησιμοποιείται αρκετά στην ανάπτυξη λειτουργικών συστημάτων (π.χ. Windows) αλλά και άλλου τύπου εφαρμογών. Θεωρείται σήμερα μία κορυφαία γλώσσα.

JAVA (μέσα '90)

Γλώσσα ειδικά σχεδιασμένη για την ανάπτυξη εφαρμογών στο Internet. Σκοπός της είναι να γράφονται προγράμματα που θα εκτελούνται, χωρίς μετατροπές, σε Η/Υ με διαφορετικά λειτουργικά συστήματα. Περιέχει αρκετά στοιχεία από τη C++.

C# (2002)

Δημιουργήθηκε ως ανταγωνιστική της JAVA. Σκοπός της είναι να συνδυάσει την ευχρηστία της Basic και τη δυναμική της C++ για την ανάπτυξη εφαρμογών που θα εκτελούνται σε Η/Υ με διαφορετικά λειτουργικά συστήματα.

Γ. Συναρτησιακές γλώσσες κατά χρονολογική σειρά εμφάνισης

LISP (μέσα '60)

Μη διαδικαστική. Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης για επεξεργασία συμβολικών δεδομένων.

Δ. Μη - διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης

PROLOG (αρχές '70)

Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης.

Ε. Γλώσσες 4^{ης} γενιάς ή ερωταπαντήσεων

SQL

Δεν απευθύνεται μόνο σε προγραμματιστές αλλά και χρήστες. Ο χρήστης μπορεί, σχετικά εύκολα, να υποβάλει ερωτήσεις στο σύστημα ή να αναζητά πληροφορίες από μία Βάση Δεδομένων.

Παράδειγμα,
SELECT Επώνυμο, Όνομα
FROM Μαθητές
WHERE Τάξη = 'Γ2'

Η παραπάνω πρόταση της γλώσσας SQL θα κάνει μία αναζήτηση στη βάση δεδομένων των μαθητών και θα εμφανίσει τα ονοματεπώνυμα των μαθητών του Γ2 μόνο.

Από τι εξαρτάται η επιλογή μίας γλώσσας προγραμματισμού;

- Από το είδος του προβλήματος (εμπορικό, αριθμητικό κ.α.)
- Από τον Η/Υ στον οποίο θα εκτελεστεί το πρόγραμμα.
- Από τη διαθέσιμη γλώσσα ή προγραμματιστικό περιβάλλον στο οποίο θα αναπτυχθεί το πρόγραμμα.
- Από τις γνώσεις και την εμπειρία του προγραμματιστή.

Με την ευρεία διάδοση των γραφικών περιβαλλόντων επικοινωνίας (π.χ. Windows, Mac OS κλπ) δημιουργήθηκαν παραλλαγές κάποιων γλωσσών που απευθύνονται σε αυτά. Τέτοιες είναι η Visual Basic, Visual C++, Delphi (Visual Pascal), C# κ.α. Αυτές οι γλώσσες ακολουθούν τη φιλοσοφία του οπτικού και του καθοδηγούμενου-από-γεγονότα προγραμματισμού χωρίς να απορρίπτουν τις αρχές του δομημένου προγραμματισμού.

6.3 Φυσικές και τεχνητές γλώσσες

Οι γλώσσες προγραμματισμού είναι τεχνητές γλώσσες που απευθύνονται σε ανθρώπους που επιθυμούν να επικοινωνήσουν με τον Η/Υ.

Κάθε γλώσσα προγραμματισμού προσδιορίζεται από:

- ο Το αλφάβητό της
- ο Το λεξιλόγιό της
- ο Τη γραμματική της
- ο Τη σημασιολογία της (Semantics)

Αλφάβητο γλώσσας

Ως αλφάβητο ορίζουμε το σύνολο των αποδεκτών χαρακτήρων της γλώσσας. Από τους χαρακτήρες αυτούς σχηματίζονται οι λέξεις της γλώσσας.

Σε μία φυσική γλώσσα, όπως τα Ελληνικά, οι αποδεκτοί χαρακτήρες είναι τα γράμματα Α-Ω (κεφαλαία και πεζά), τα ψηφία 0-9 και τα σημεία στίξης.

Λεξιλόγιο γλώσσας

Το λεξιλόγιο μίας γλώσσας περιλαμβάνει όλες τις έγκυρες και αποδεκτές λέξεις. Στην ουσία, είναι ένα υποσύνολο από όλες τις δυνατές ακολουθίες που μπορούμε να σχηματίσουμε από τα στοιχεία του αλφαβήτου.

Σε μία φυσική γλώσσα, όπως τα Ελληνικά, η λέξη ΔΙΑΒΑΖΩ είναι αποδεκτή ενώ η λέξη ΖΩΒΑΓΩ όχι.

Η Γραμματική της γλώσσας

Η γραμματική περιλαμβάνει το **τυπολογικό** και το **συντακτικό**.

Το **τυπολογικό** ορίζει τους κανόνες σύμφωνα με του οποίους μία λέξη θα είναι αποδεκτή.

Για παράδειγμα, στην ελληνική γλώσσα για τη λέξη "δίνω", αποδεκτές μορφές είναι και το "δίνεις", "δίνουν" αλλά όχι το "δίνουη".

Το **συντακτικό** είναι ένα σύνολο κανόνων που ορίζει το πώς πρέπει να σχηματίζονται οι προτάσεις από τις λέξεις της γλώσσας ώστε οι προτάσεις αυτές να είναι έγκυρες και αποδεκτές. Σε μία γλώσσα προγραμματισμού αυτό που ενδιαφέρει είναι η σωστή σύνταξη των εντολών.

Σημασιολογία της γλώσσας

Είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και προτάσεων της γλώσσας. Σε μία γλώσσα προγραμματισμού αυτό καθορίζεται από το δημιουργό της ενώ σε μία φυσική γλώσσα από αυτόν που εκφέρει την πρόταση.

Διαφορές μεταξύ φυσικών και τεχνητών γλωσσών

Φυσικές

Χρησιμοποιούνται για την επικοινωνία μεταξύ ανθρώπων.

Έχουν μεγάλες δυνατότητες εξέλιξης. Νέες λέξεις μπορεί να εισαχθούν, κανόνες γραμματικής και σύνταξης να αλλάξουν κλπ.

Τεχνητές

Χρησιμοποιούνται για την επικοινωνία μεταξύ ανθρώπου και Η/Υ.

Οι δυνατότητες εξέλιξης είναι περιορισμένες. Τις περισσότερες φορές η εξέλιξη αυτή αφορά την επέκταση του ρεπερτορίου των εντολών της γλώσσας (π.χ. Basic και Visual Basic).

6.4 Τεχνικές σχεδίασης προγραμμάτων

Δεν αρκεί κάποιος να γνωρίζει απλά μία γλώσσα προγραμματισμού. Θα πρέπει να ακολουθήσει και μία τεχνική σχεδίασης του προγράμματός του.

Για την σύνταξη σωστών, κατανοητών και εύκολα συντηρήσιμων προγραμμάτων ακολουθήθηκαν διάφορες μεθοδολογίες ανάπτυξης που παρουσιάζονται παρακάτω:

6.4.1 Ιεραρχική σχεδίαση ή Top down σχεδίαση

Η τεχνική αυτή συνιστά στον καθορισμό των βασικών λειτουργιών του προγράμματος σε ανώτερο επίπεδο και στη συνέχεια τη διάσπαση καθεμιάς σε μικρότερες και απλούστερες μέχρι του σημείου να είναι τόσο απλές που μπορούν να επιλυθούν άμεσα.

Συνεπώς, σκοπός της ιεραρχικής σχεδίασης είναι η διάσπαση του προβλήματος σε μικρότερα και απλούστερα υπο-προβλήματα τα οποία είναι ευκολότερο να επιλυθούν.

Για την Top-down σχεδίαση χρησιμοποιούμε το **ιεραρχικό διάγραμμα**.

6.4.2 Τμηματικός προγραμματισμός

Η ιεραρχική σχεδίαση υλοποιείται με τον τμηματικό προγραμματισμό.

Μετά την ανάλυση του προβλήματος σε μικρότερα και απλούστερα υπο-προβλήματα, κάθε υπο-πρόβλημα αποτελεί μία ξεχωριστή και ανεξάρτητη **ενότητα** (module). Τώρα, για κάθε ενότητα θα γραφτεί το κατάλληλο πρόγραμμα ή τμήμα προγράμματος.

6.4.3 Δομημένος προγραμματισμός

Είναι η μεθοδολογία που έχει επικρατήσει σήμερα. Εμπεριέχει τις αρχές της ιεραρχικής σχεδίασης και του τμηματικού προγραμματισμού. Επιπλέον, αναφέρει ότι για τη δημιουργία σωστών προγραμμάτων να χρησιμοποιούμε μόνο τις 3 στοιχειώδεις δομές: Ακολουθίας, Επιλογής και Επανάληψης.

Όλα τα προγράμματα, οσοδήποτε μεγέθους, μπορούν να γραφτούν στηριζόμενα στη χρήση μόνο αυτών των δομών, αποφεύγοντας πλήρως τη χρήση τη δομής GOTO. Επίσης, κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Πλεονεκτήματα του δομημένου προγραμματισμού

- Άμεση υλοποίηση των αλγορίθμων σε πρόγραμμα.
- Διευκόλυνση της ανάλυσης του προγράμματος σε τμήματα (ενότητες). Το κάθε τμήμα μπορεί να γραφτεί από διαφορετικές ομάδες προγραμματιστών.
- Ευκολότερη και συντομότερη ανάπτυξη προγραμμάτων.
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση του προγράμματος.

Οπτικός προγραμματισμός

Είναι η δυνατότητα να δημιουργούμε, με γραφικό τρόπο, ολόκληρο το περιβάλλον της εφαρμογής όπως για παράδειγμα τα μενού και τα πλαίσια διαλόγου και άλλα παράθυρα της εφαρμογής.

Ο οπτικός προγραμματισμός εκμεταλλεύεται τις δυνατότητες των γραφικών περιβαλλόντων επικοινωνίας (π.χ Windows, Mac Os κλπ).

Καθοδηγούμενος από γεγονότα προγραμματισμός

Είναι η δυνατότητα να εκτελούνται οι διάφορες λειτουργίες του προγράμματος με την ενεργοποίηση ενός γεγονότος. Για παράδειγμα, αν κάνουμε κλικ σε κάποια εντολή ενός μενού ή σε κάποιο κουμπί σε ένα παράθυρο της εφαρμογής τότε θα εκτελεστεί μία λειτουργία..

Τα σύγχρονα προγραμματιστικά περιβάλλοντα είναι κτισμένα πάνω στις αρχές του οπτικού και καθοδηγούμενου από γεγονότα προγραμματισμού.

6.5 Αντικειμενοστρεφής προγραμματισμός

Η φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού είναι η εξής: Να θεωρήσουμε τα δεδομένα και τις αποδεκτές ενέργειες που γίνονται πάνω σε αυτά ως ένα ενιαίο αντικείμενο (object). Αυτό το αντικείμενο θα μπορεί να χρησιμοποιηθεί πολύ εύκολα οπουδήποτε αλλού, δηλαδή θα είναι επαναχρησιμοποιήσιμο.

Ο αντικειμενοστρεφής προγραμματισμός ακολουθεί τις αρχές του δομημένου προγραμματισμού.

6.7 Προγραμματιστικά περιβάλλοντα

Ένα πρόγραμμα που φτιάχνεται σε μία γλώσσα υψηλού επιπέδου δεν είναι άμεσα κατανοητό από τον Η/Υ. Θα πρέπει να μεταφραστεί σε ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (δυαδική μορφή). Την διαδικασία μετάφρασης την πραγματοποιούν τα μεταφραστικά προγράμματα. Είναι δύο ειδών:

- Μεταγλωττιστές (Compilers)
- Διερμηνευτές (Interpreters)

Πηγαίο πρόγραμμα (Source program): Το πρόγραμμα που είναι φτιαγμένο σε γλώσσα υψηλού επιπέδου.

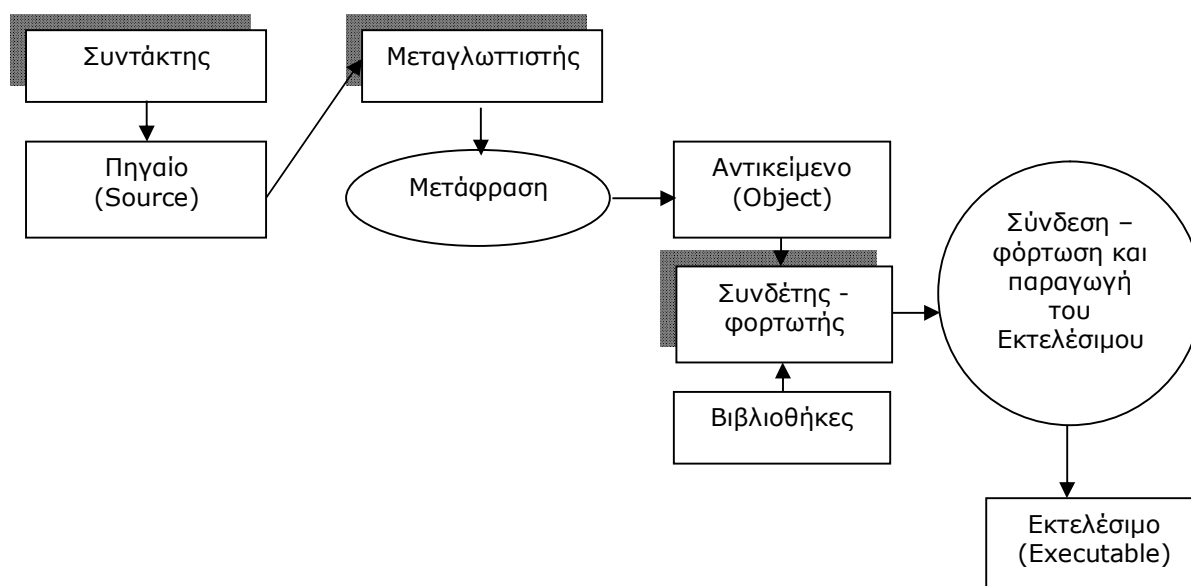
Αντικείμενο πρόγραμμα (Object program): Το πρόγραμμα που είναι μεταφρασμένο σε γλώσσα μηχανής αλλά όχι άμεσα εκτελέσιμο.

Εκτελέσιμο πρόγραμμα (Executable program): Το πρόγραμμα σε γλώσσα μηχανής που είναι έτοιμο πλέον να εκτελεστεί.

Μεταγλωττιστής: Παίρνει, ως είσοδο, το πηγαίο πρόγραμμα και αναλαμβάνει να το μεταφράσει εξ' ολοκλήρου παράγοντας το αντικείμενο πρόγραμμα.

Διερμηνευτής: Παίρνει, ως είσοδο, μία-μία εντολή του πηγαίου, την μεταφράζει και την εκτελεί αμέσως. Η λειτουργία του μοιάζει με τον άνθρωπο-διερμηνέα που μεταφράζει επί τόπου κάθε πρόταση.

Η διαδικασία μεταγλώττισης και σύνδεσης προγράμματος φαίνεται παρακάτω:



Βιβλιοθήκες (Libraries): Έτοιμες ενότητες (modules) αντικείμενου προγράμματος της γλώσσας, απαραίτητες για την παραγωγή του εκτελέσιμου προγράμματος.

Συνδέτης – Φορτωτής (Linker-Loader): Ειδικό πρόγραμμα που αναλαμβάνει να συνδέσει το αντικείμενο πρόγραμμα με τις βιβλιοθήκες και να παράγει το εκτελέσιμο.

Κατά τη δημιουργία ενός προγράμματος σχεδόν πάντα ενυπάρχουν λάθη. Τα λάθη τα χωρίζουμε σε δύο κατηγορίες :

- Συντακτικά λάθη
- Λογικά λάθη

Τα **συντακτικά λάθη** ανιχνεύονται κατά την διαδικασία της μεταγλώττισης ή διερμηνεύσης. Αφορούν παραβιάσεις του τυπολογικού και συντακτικού της γλώσσας. (π.χ. μία εντολή έχει γραφτεί συντακτικά λάθος). Στην περίπτωση αυτή ο προγραμματιστής πρέπει να επιστρέψει στο πηγαίο πρόγραμμα, να διορθώσει τα λάθη και να το ξανά υποβάλλει για μεταγλώττιση.

Τα **λογικά λάθη** είναι και τα πλέον δύσκολα στην ανίχνευσή τους. Έχουν να κάνουν με σφάλματα στη λογική επίλυσης του προβλήματος ή λανθασμένης διατύπωσης του αλγορίθμου (π.χ. το πρόγραμμα παράγει άλλα αποτελέσματα κι όχι τα ζητούμενα!).

Για την σύνταξη των προγραμμάτων χρησιμοποιούμε ένα ειδικό πρόγραμμα που ονομάζεται **συντάκτης (editor)**. Μοιάζει με επεξεργαστή κειμένου με επιπλέον δυνατότητες που διευκολύνουν την γρήγορη σύνταξη των πηγαίων προγραμμάτων.

Τα σύγχρονα προγραμματιστικά περιβάλλοντα περιέχουν οπωσδήποτε 3 ειδών προγράμματα : Τον συντάκτη, το μεταγλωττιστή και το συνδέτη - φορτωτή. Πέραν αυτών, περιέχουν όλα εκείνα τα εργαλεία που διευκολύνουν την εύκολη, ταχύτατη σύνταξη, διόρθωση και συντήρηση των προγραμμάτων γι' αυτό και πολλές φορές ονομάζονται ως **RAD περιβάλλοντα (Rapid Application Development)**.

Όλα αυτά παρέχονται με έναν ενιαίο και λειτουργικό τρόπο στον προγραμματιστή.

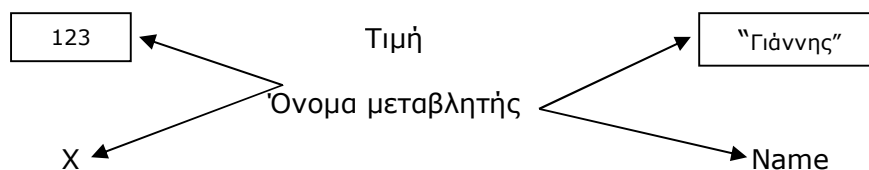
2.4 Βασικές συνιστώσες/εντολές ενός αλγόριθμου

Βασικά στοιχεία αλγορίθμου

1) Σταθερές: Αφορούν ποσότητες που δεν μεταβάλλονται κατά τη διάρκεια εκτέλεσης του αλγόριθμου. Υπάρχουν 3 ειδών :

- **Αριθμητικές** π.χ. 123, -8, 3.14
- **Αλφαριθμητικές** π.χ. "Γιάννης", "Μακεδονίας 12". Λέγονται και λεκτικά ή συμβολοσειρές. Περικλείονται σε " "
- **Λογικές** π.χ. Αληθής, Ψευδής.

2) Μεταβλητές: Αφορούν ποσότητες που μεταβάλλονται κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Χονδρικά, παριστάνουν μία θέση μνήμης που περιέχει μία τιμή. Το περιεχόμενο αυτής της θέσης (η τιμή της) αλλάζει.



Στις μεταβλητές δίνουμε ένα όνομα (αναγνωριστικό) και αυτό χρησιμοποιούμε στον αλγόριθμο. **(Τα ονόματα των μεταβλητών ακολουθούν τον εξής κανόνα : Ελληνικά ή Αγγλικά, κεφαλαία ή μικρά, από σύμβολα μόνο η κάτω παύλα _ , οι αριθμοί επιτρέπονται αλλά όχι σαν πρώτος χαρακτήρας, και απαγορεύονται ονόματα που είναι δεσμευμένες λέξεις, π.χ απαγορεύεται όνομα μεταβλητής Διάβασε ή αλγόριθμος).**

Η τιμή της αλλάζει με μία **εντολή εκχώρησης τιμής** ← :
π.χ. $X \leftarrow 123$ ή $Name \leftarrow \text{"Γιάννης"}$.

Κι εδώ, ανάλογα με το είδος δεδομένου που εκχωρείται η μεταβλητή διακρίνεται σε:

- **Αριθμητική** π.χ. $X \leftarrow 123$
- **Αλφαριθμητική** π.χ. $Name \leftarrow \text{"Γιάννης"}$
- **Λογική** π.χ. $\text{Έγγαμος} \leftarrow \text{Ψευδής}$.

Επιπλέον, οι **αριθμητικές** διακρίνονται σε

- **Ακέραιες,** αν δέχονται ακέραια τιμή π.χ. 5, -12
- **Πραγματικές,** αν δέχονται πραγματική τιμή π.χ. 3.14, -4.56

3) Τελεστές: Είναι τα γνωστά σύμβολα. Είναι 3 ειδών:

- **Αριθμητικοί:** + (πρόσθεση), - (αφαίρεση), * (πολλ/σμός), / (διαίρεση), ^ (ύψωση σε δύναμη), div (ακέραια διαίρεση), mod (ακέραιο υπόλοιπο). **(Σειρά εκτέλεσης αριθμητικών πράξεων : 1^ο το ^, 2^α και με ίδια προτεραιότητα τα *, /, div, mod και από αριστερά προς τα δεξιά και 3^α τα +, - και φυσικά πρώτα οι παρενθέσεις)**

Αριθμητικός τελεστής	Αριθμητική πράξη
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
div	Ακέραιο πηλίκιο
mod	Το υπόλοιπο της ακέραιας διαίρεσης
^	Υψωση σε δύναμη

- **Λογικοί:** και, ή, όχι.

(Σειρά εκτέλεσης λογικών πράξεων : 1^ο το **όχι**, 2^α το **και** και 3^α το **ή** από αριστερά προς τα δεξιά και φυσικά πρώτα οι παρενθέσεις)

Λογικός τελεστής	Λογική Πράξη
όχι	Άρνηση (OXI)
και	Σύζευση ή τομή (NAI)
ή	Διάζευξη ή ένωση (H)

π.χ. Έγκυρος $\leftarrow (X \geq 1) \text{ και } (X \leq 20)$

- **Συγκριτικοί:** $> , \geq , < , \leq , = , \neq$ (διάφορο)

Συγκριτικός Τελεστής	Συνθήκη
=	Ισότητα
$\neq$	Ανισότητα
>	Μεγαλύτερο από
$\geq$	Μεγαλύτερο ή ίσο
<	Μικρότερο από
$\leq$	Μικρότερο ή ίσο

4) **Εκφράσεις:** Συνδυάζουν όλα τα παραπάνω:

Προτεραιότητα τελεστών	
Τελεστής	Περιγραφή
$\wedge$	Υψωση σε δύναμη
$*, /, \text{div}, \text{mod}$	Πολλαπλασιασμός, Διαίρεση, Ακέραια Διαίρεση, Υπόλοιπο
$+, -$	Πρόσθεση και αφαίρεση
$> , \geq , < , \leq , = , \neq$	Συγκρίσεις
not	Λογικό OXI
and	Λογικό ΚΑΙ
or	Λογικό Ή

5) **Βασικές εντολές:**

- **Εντολή εισόδου:** **Διάβασε**
π.χ. **Διάβασε** X . Στη μεταβλητή X εισάγεται μία τιμή.

Συντάσσεται με την λέξη **Διάβασε** και δεξιά από αυτή το **όνομα μιας μεταβλητής (ή περισσότερων)**, είναι **εντολή εισόδου**, που σημαίνει ότι χρησιμοποιείται για την **είσοδο δεδομένων** από το πληκτρολόγιο.

π.χ **Διάβασε** X ή **Διάβασε** X,Y,Z (για περισσότερες μεταβλητές χωρίζω με κόμμα)

- **Εντολή εξόδου:** **Εκτύπωσε ή Εμφάνισε** (Γράψε στα προγράμματα)

π.χ. **Εκτύπωσε** X

Συντάσσεται με την λέξη **Εμφάνισε** (ή **Εκτύπωσε** όταν θέλουμε να πάει στο χαρτί αντί στην οθόνη) και στα δεξιά της μπορεί να μπει ή όνομα μιας μεταβλητής (ή περισσότερων), ή ένα μήνυμα μέσα σε εισαγωγικά, ή συνδυασμός μεταβλητής/των και μηνυμάτων χωρισμένων μεταξύ τους με κόμμα. Είναι **εντολή εξόδου**, που σημαίνει ότι χρησιμοποιείται για εμφάνιση πληροφοριών στην οθόνη (Εμφάνισε) ή στο χαρτί (Τύπωσε).

π.χ **Εκτύπωσε** X ή **Εκτύπωσε** X,Y ή **Εκτύπωσε** 'Καλημέρα' ή **Εκτύπωσε** 'Το εμβαδό του τριγώνου είναι: ', X

- **Εντολή εκχώρησης τιμής:** Μορφή: **Μεταβλητή** ← **Έκφραση**.
Αριστερά βάζουμε το όνομα της μεταβλητής και δεξιά μία τιμή ή έκφραση
Π.χ. $X \leftarrow (5 * 2) / 100$.
Το αποτέλεσμα εκχωρείται στη μεταβλητή.

Με δύο τρόπους λοιπόν, τοποθετούμε τιμές σε μεταβλητές:

- I. Με εντολή εισόδου Διάβασε.
- II. Κατευθείαν με μία εντολή εκχώρησης,

7.9 Εντολές εισόδου εξόδου

Η **αλγόριθμος** υποστηρίζει για την εισαγωγή δεδομένων από το πληκτρολόγιο την εντολή **Διάβασε** και για την εμφάνιση των αποτελεσμάτων την εντολή **Γράψε**.

- **Εντολή εισόδου:** **Διάβασε**

Η εντολή **Διάβασε** ακολουθείται πάντοτε από ένα ή περισσότερα ονόματα μεταβλητών. Αν υπάρχουν περισσότερες από μία μεταβλητές τότε αυτές χωρίζονται με κόμμα (,).

Π.χ. **Διάβασε** x .

Στη μεταβλητή x εισάγεται μία τιμή.

Συντάσσεται με την λέξη **Διάβασε** και δεξιά από αυτή το **όνομα μιας μεταβλητής (ή περισσότερων)**, είναι **εντολή εισόδου**, που σημαίνει ότι χρησιμοποιείται για την **είσοδο δεδομένων** από το πληκτρολόγιο.

π.χ **Διάβασε** x ή **Διάβασε** x, y, z (για περισσότερες μεταβλητές χωρίζω με κόμμα)

- **Εντολή εξόδου:** **Γράψε**

Η εντολή **Γράψε** έχει ως αποτέλεσμα την εμφάνιση τιμών στη μονάδα εξόδου. Συσκευή εξόδου μπορεί να είναι η οθόνη του υπολογιστή, ο εκτυπωτής, βοηθητική μνήμη ή γενικά οποιαδήποτε συσκευή εξόδου έχει οριστεί στο πρόγραμμα.

Π.χ. **Γράψε** x

Συντάσσεται με την λέξη **Γράψε** και στα δεξιά της μπορεί να μπει ή **όνομα μιας μεταβλητής (ή περισσότερων)**, ή **ένα μήνυμα μέσα σε εισαγωγικά**, ή **συνδυασμός μεταβλητής/των και μηνυμάτων χωρισμένων μεταξύ τους με κόμμα**. Είναι **εντολή εξόδου**, που σημαίνει ότι χρησιμοποιείται για εμφάνιση πληροφοριών στην οθόνη (Εμφάνισε) ή στο χαρτί (Τύπωσε).

π.χ **Γράψε** x ή **Γράψε** x, y ή **Γράψε** 'Καλημέρα' ή **Γράψε** 'Το εμβαδό του τριγώνου είναι: ', x

2.4.1 Πως γράφουμε ένα αλγόριθμο ή ένα πρόγραμμα στην γλωσσολογία.

Η **πρώτη γραμμή** ενός αλγορίθμου ξεκινά με την λέξη **Αλγόριθμος** και δεξιά το **όνομα του αλγορίθμου**, το οποίο όνομα ακολουθεί τον κανόνα των ονομάτων που μάθαμε και για τις μεταβλητές.

Η τελευταία γραμμή του αλγορίθμου ξεκινά με την λέξη **Τέλος** και δεξιά το **όνομα του αλγορίθμου** το οποίο βάλαμε και στην αρχή.

7.7 Αριθμητικές εκφράσεις

Συνδυάζουν όλα τα παραπάνω

Όταν μια τιμή προκύπτει από υπολογισμό, τότε αναφερόμαστε σε **εκφράσεις (expressions)**. Για τη σύνταξη μιας αριθμητικής έκφρασης χρησιμοποιούνται αριθμητικές σταθερές, μεταβλητές, συναρτήσεις, αριθμητικοί τελεστές και παρενθέσεις. 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 προηγουμένα, δηλαδή να έχουν κάποια τιμή.

Ιεραρχία

1. Ύψωση σε δύναμη.
2. Πολλαπλασιασμός και διαίρεση.
3. Πρόσθεση και αφαίρεση.

7.8 Εντολή εκχώρησης

Η εντολή εκχώρησης χρησιμοποιείται για την απόδοση τιμών στις μεταβλητές κατά τη διάρκεια εκτέλεσης του προγράμματος.

Μορφή: **Μεταβλητή ← Έκφραση.**

Αριστερά βάζουμε το όνομα της μεταβλητής και δεξιά μία τιμή ή έκφραση

Π.χ. $X \leftarrow (5 * 2) / 100$.

Το αποτέλεσμα εκχωρείται στη μεταβλητή.

Με δύο τρόπους λοιπόν, τοποθετούμε τιμές σε μεταβλητές:

- III.** Με εντολή εισόδου Διάβασε.
- IV.** Κατευθείαν με μία εντολή εκχώρησης,

2.4.2 Δομή ακολουθίας

Όταν για την εκτέλεση ενός αλγορίθμου απαιτείται σειριακή εκτέλεση εντολών

```

Αλγόριθμος όνομα
  Διάβασε α, β           ή  Δεδομένα // α, β //
  Εντολή 1
  ...
  Εντολή N
  Εκτύπωσε γ           ή  Αποτελέσματα // άθροισμα //
Τέλος όνομα
  
```

Παρατηρήσεις

- Οι εντολές εκτελούνται σειριακά όπως παρατίθενται στον αλγόριθμο.
- Στην αρχή του αλγορίθμου πραγματοποιείται η είσοδος των δεδομένων και τελευταία ενέργεια είναι η έξοδος – εκτύπωσή τους.
- Πάντοτε ξεκινάμε με τη λέξη Αλγόριθμος και ακολουθεί το όνομα του αλγορίθμου. Το όνομα ενός αλγορίθμου ακολουθεί τους ίδιους κανόνες με το όνομα μιας μεταβλητής. Επιπλέον, δεν μπορεί να χρησιμοποιείται ίδιο όνομα για μια μεταβλητή και για τον αλγόριθμο. Αυτές οι λέξεις καθώς και όλες όσες είναι έντονα γραμμένες καλούνται **δεσμευμένες** λέξεις και δεν επιτρέπεται να παρουσιαστούν αλλιώς, ούτε να χρησιμοποιηθούν με λάθος τρόπο.
- Για την χρησιμοποίηση της τιμής κάποιας μεταβλητής πρέπει οπωσδήποτε να έχει προηγηθεί εκχώρηση τιμής σε αυτή τη μεταβλητή είτε με εντολή εκχώρησης είτε με είσοδο δεδομένων από το χρήστη. Πιο πρακτικά, δεν μπορούμε να χρησιμοποιήσουμε μια μεταβλητή στο δεξί τμήμα μιας εντολής εκχώρησης ή σε μια εκτύπωση αν προηγουμένως δεν έχει λάβει τιμή στον αλγόριθμο – **αρχικοποίηση**.
- Η πρώτη ή οι πρώτες εντολές πραγματοποιούν είσοδο δεδομένων. Αυτό μπορεί να γίνει με την εντολή Διάβασε ακολουθούμενη από τα ονόματα μεταβλητών ή με την εντολή Δεδομένα (δες αλγόριθμο).

Όταν η εκφώνηση αναφέρει ρητώς την ανάγνωση των δεδομένων από το χρήστη τότε χρησιμοποιείται η εντολή **Διάβασε**. Σε διαφορετική περίπτωση π.χ. όταν η εκφώνηση αναφέρει «έστω μεταβλητή N που περιέχει το πλήθος αριθμών», τότε μπορούμε να εισάγουμε την τιμή της μεταβλητής N στον αλγόριθμό μας με την εντολή **Δεδομένα**. Όμοια για την έξοδο και τις εντολές **Εκτύπωσε** (για αποστολή εκτύπωσης στον εκτυπωτή), **Εμφάνισε** (για εμφάνιση των στοιχείων στην οθόνη) και της εντολής **Αποτελέσματα**.

8.1 Εντολές Επιλογής

A. Λογική Έκφραση

Για τη σύνταξη μιας **λογικής έκφρασης ή συνθήκης** χρησιμοποιούνται **σταθερές, μεταβλητές, αριθμητικές, αλφαριθμητικές και λογικές τιμές και 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. Το αποτέλεσμα είναι μία λογική τιμή **Αληθής** ή **Ψευδής**.

B. Σύνθετες Εκφράσεις

Σε πολλά προβλήματα οι επιλογές δεν αρκεί να γίνονται με απλές λογικές παραστάσεις όπως αυτές οι οποίες αναφέρθηκαν, αλλά χρειάζεται να συνδυαστούν μία ή περισσότερες λογικές παραστάσεις. Αυτό επιτυγχάνεται με τη χρήση των τριών βασικών λογικών τελεστών **όχι, και, ή**.

2.4.3 Δομή επιλογής

Πολλές φορές για να λυθεί ένα πρόβλημα πρέπει να ελεγχθεί αν ισχύει κάποια συνθήκη.

Η γενική σύνταξη της δομής επιλογής είναι:

Αν συνθήκη **τότε**
 Εντολή a_1
 ...
 Εντολή a_n
Τέλος_αν

Παρατηρήσεις

Η εντολή $x \leftarrow x + 1$ δεν είναι μαθηματική έκφραση αλλά εντολή εκχώρησης. Επιτρέπεται λοιπόν, η παρουσία της ίδιας μεταβλητής αριστερά και δεξιά του $\leftarrow$. Έτσι, προσθέτουμε το περιεχόμενο της μεταβλητής x με το 1 και το αποτέλεσμα καταχωρείται εκ νέου στη μεταβλητή x

2.4.3.1 Δομή Σύνθετης επιλογής

Η δομή σύνθετης επιλογής χρησιμοποιείται στην περίπτωση που επιθυμούμε να εκτελέσουμε εναλλακτικά δυο ομάδες εντολών. Κριτήριο για το ποιο σενάριο εντολών θα εκτελεστεί αποτελεί κάποια συνθήκη. Στην περίπτωση που η συνθήκη ισχύει θα εκτελεστεί η πρώτη ομάδα εντολών διαφορετικά θα εκτελεστεί η δεύτερη

Η γενική σύνταξη της σύνθετης δομής επιλογής είναι:

```

Αν συνθήκη τότε
    Εντολή  $a_1$ 
    ...
    Εντολή  $a_n$ 
αλλιώς
    Εντολή  $b_1$ 
    ...
    Εντολή  $b_n$ 
Τέλος_αν
    
```

2.4.3.2 Δομή πολλαπλής επιλογής

Κάποια προβλήματα απαιτούν την επιλογή μεταξύ περισσότερων από δυο περιπτώσεις

- Η γενική σύνταξη της πολλαπλής δομής επιλογής είναι:

```

Αν συνθήκη1 τότε
    Εντολή  $a_1$ 
    ...
    Εντολή  $a_n$ 
αλλιώς_αν συνθήκη2 τότε
    Εντολή  $b_1$ 
    ...
    Εντολή  $b_n$ 
    ...
αλλιώς
    Εντολή  $c_1$ 
    ...
    Εντολή  $c_n$ 
Τέλος_αν
    
```

Παρατηρήσεις

- Μια από τις περιπτώσεις της δομής επιλογής εκτελούνται σε κάθε περίπτωση
- Τα σύμβολα ' και " χρησιμοποιούνται ισοδύναμα
- Στις περιπτώσεις που θα ελεγχθούν περιλαμβάνουμε και αυτήν της λανθασμένης εισαγωγής δεδομένων
- Τα παραπάνω θα μπορούσαν να υλοποιηθούν και με πολλές δομές σύνθετης επιλογής

2.4.4 Εμφωλευμένες Δομές

Στην υλοποίηση ενός αλγορίθμου μπορούμε να χρησιμοποιούμε κάποιες από τις παραπάνω δομές ως τμήμα εντολών σε άλλες δομές.

2.4.5 Δομή Επανάληψης

Πολλές φορές πρέπει να εκτελεστεί μια ομάδα εντολών περισσότερες από μια φορές. Σ' αυτές τις περιπτώσεις χρησιμοποιούνται οι δομές επανάληψης. Υπάρχουν δυο περιπτώσεις: το πλήθος των επαναλήψεων να είναι γνωστό εξ αρχής ή όχι.

Η δομή επανάληψης Για

Όταν είναι γνωστό το πλήθος των επαναλήψεων χρησιμοποιούμε την δομή επανάληψης Για

Η γενική σύνταξη της δομής επανάληψης Για είναι:

Για *i* από *k* **μέχρι** *n* **με_βήμα** *j*

...

Τέλος_επανάληψης

Παρατηρήσεις

- Οι εντολές του βρόχου εκτελούνται για όλες τις τιμές της μεταβλητής από την αρχική τιμή μέχρι την τελική τιμή, αυξανόμενες με την τιμή του βήματος. Αν το βήμα είναι ίσο με 1, τότε παραλείπεται.
- Για τη περίπτωση $k=1$ και $j=1$: Η μεταβλητή *i* θα αυξηθεί από την τιμή 1 μέχρι την τιμή *n* (την ονομάζουμε μετρητή) κατά μία μονάδα κάθε φορά.
- Το τμήμα εντολών που βρίσκεται εντός του **Για** ονομάζεται βρόχος.
- Η μεταβλητή του **Για** δεν επιτρέπεται να αλλάζει τιμή μέσα στο βρόχο.
Η μεταβλητή *i* θα αυξηθεί $n+1-k$ φορές για βήμα +1
Η μεταβλητή *i* θα μειωθεί $k+1-n$ φορές για βήμα -1
Αριθμός επαναλήψεων = $[\text{Απόλυτη_Τιμή}(\text{Τελική_τιμή}-\text{Αρχική_τιμή})] \text{ div } [\text{Απόλυτη_Τιμή}(\text{Βήμα})+1]$

Στην περίπτωση που θα χρησιμοποιήσουμε μία μεταβλητή για να υπολογίσουμε το άθροισμα πολλών τιμών χρησιμοποιώντας επαναληπτική δομή : Πρέπει έξω από την επαναληπτική δομή να μηδενίσουμε την μεταβλητή για το άθροισμα, διαφορετικά δεν μπορούμε να γνωρίζουμε την τιμή της κατά την πρώτη επανάληψη

Γενικώς, δεν μπορούμε να χρησιμοποιήσουμε μια μεταβλητή, αν προηγούμενα δεν έχει λάβει τιμή – αρχικοποίηση. Η αρχικοποίηση μπορεί να πραγματοποιηθεί είτε με την είσοδο δεδομένων σε μια μεταβλητή (εντολή **Διάβασε**) είτε με εκχώρηση τιμής

Οι τιμές των *k, n, j* δεν είναι απαραίτητο να είναι ακέραιοι αριθμοί, αλλά μπορεί να λάβουν και δεκαδική τιμή.

Ειδικές μορφές της **Για**:

- Για** δείκτη **από** μικρή τιμή σε μεγάλη τιμή **με βήμα** αρνητικό καμία επανάληψη,
- Για** δείκτη **από** Μεγάλη τιμή σε μικρή τιμή **με βήμα** θετικό καμία επανάληψη,
- Για** δείκτη **από** μία τιμή στον εαυτό της **με βήμα** αρνητικό ή θετικό μία επανάληψη και
- Για** δείκτη **από** μία τιμή σε μία άλλη **με βήμα** μηδέν άπειρες επαναλήψεις.

Η δομή επανάληψης Όσο...Επανάλαβε

Όταν δεν είναι καθορισμένο το πλήθος των επαναλήψεων εξ' αρχής τότε δεν μπορεί να χρησιμοποιηθεί η δομή επανάληψης **Για** που παρουσιάστηκε στην προηγούμενη παράγραφο. Σε αυτήν την περίπτωση χρησιμοποιείται η δομή **Όσο...επανάλαβε**, η οποία θα περιγραφεί με τη βοήθεια ενός ακόμη παραδείγματος

Η γενική σύνταξη της δομής επανάληψης **Όσο...επανάλαβε** είναι:

- **Όσο** συνθήκη **επανάλαβε**
Εντολή a_1
...
Εντολή a_n
- **Τέλος_επανάληψης**

Παρατηρήσεις

- Πρέπει να τονιστεί ότι ο έλεγχος της συνθήκης στη δομή **Όσο...επανάλαβε** πραγματοποιείται στην αρχή του βρόχου. Η δομή αυτή μπορεί να μην εκτελεστεί καμία φορά.
- Χρησιμοποιείται στην περίπτωση που έχουμε τιμή φρουρό δηλ. θέλουμε η επανάληψη να σταματά όταν ο χρήστης δώσει συγκεκριμένη τιμή.

Η δομή επανάληψης Αρχή_Επανάληψης...Μέχρις_Ότου

Συμπληρωματικά της δομής **Όσο...επανάλαβε** χρησιμοποιείται η δομή **Αρχή_Επανάληψης...Μέχρις_Ότου**

Η γενική σύνταξη της δομής επανάληψης **Αρχή_Επανάληψης...Μέχρις_Ότου** είναι:

Αρχή_επανάληψης

Εντολή a_1

...

Εντολή a_n

Μέχρις_ότου συνθήκη

Παρατηρήσεις

- Παρατηρούμε ότι σε αυτή τη δομή ο έλεγχος της συνθήκης πραγματοποιείται στο τέλος του βρόχου
- Η δομή αυτή εκτελείται τουλάχιστον 1 φορά.
- Χρησιμοποιείται στις περιπτώσεις που έχουμε Μενού επιλογής, Έλεγχο δεδομένων από το πληκτρολόγιο και Επανάληψη απαντώντας σε συγκεκριμένη ερώτηση.

Παρατηρήσεις για τις δομές επανάληψης

Η δομή επανάληψης **Για** χρησιμοποιείται όταν είναι γνωστό το πλήθος των επαναλήψεων ενώ οι δομές **Όσο...επανάλαβε** και **Αρχή_επανάληψης...Μέχρις_ότου** στις άλλες περιπτώσεις

Ένα ζήτημα που μπορεί να μας απασχολήσει στις ασκήσεις είναι η μετατροπή από την μία δομή στην άλλη. Μια δομή **Για** μπορεί να μετατραπεί πάντα στις **Όσο...επανάλαβε** και **Αρχή_επανάληψης...Μέχρις_ότου**, το αντίθετο δεν ισχύει πάντα. Οι δομές **Όσο...επανάλαβε** και **Αρχή_επανάληψης...Μέχρις_ότου** μπορούν πάντα να μετατραπούν από τη μία μορφή στην άλλη. Κάθε **Για** μετατρέπεται σε **Όσο** και **Μέχρις_ότου**, το ανάποδο όχι.

Σε εμφωλευμένους βρόχους δεν επιτρέπεται να χρησιμοποιείται η ίδια μεταβλητή ως μετρητής δύο ή περισσότερων βρόχων που ο ένας βρίσκεται στο εσωτερικό του άλλου.

Εμφωλευμένες Δομές

Πολύ συχνά για την επίλυση των προβλημάτων απαιτείται η χρήση **εμφωλευμένων βρόχων**. Σε αυτή την περίπτωση ο ένας βρόχος βρίσκεται μέσα στον άλλο. Στη χρήση των **εμφωλευμένων βρόχων** ισχύουν συγκεκριμένοι κανόνες που πρέπει να ακολουθούνται αυστηρά για την σωστή λειτουργία των προγραμμάτων.

Συγκεκριμένα:

- Ο εσωτερικός βρόχος πρέπει να βρίσκεται ολόκληρος μέσα στον εξωτερικό. Ο βρόχος που ξεκινάει τελευταίος, πρέπει να ολοκληρώνεται πρώτος.
- Η είσοδος σε κάθε βρόχο υποχρεωτικά γίνεται από την αρχή του.
- Δεν μπορεί να χρησιμοποιηθεί η ίδια μεταβλητή ως μετρητής δύο ή περισσότερων βρόχων που ο ένας βρίσκεται στο εσωτερικό του άλλου.

3.1 Δεδομένα

Όπως η Πληροφορική ορίζεται ως επιστήμη σε συνάρτηση με την έννοια του αλγορίθμου, κατά τον ίδιο τρόπο η Πληροφορική ορίζεται και σε σχέση με την έννοια των δεδομένων. Έτσι, Πληροφορική θεωρείται η επιστήμη που μελετά τα δεδομένα από τις ακόλουθες σκοπιές:

Υλικού

Το υλικό (hardware), δηλαδή η μηχανή, επιτρέπει στα δεδομένα ενός προγράμματος να αποθηκεύονται στην κύρια μνήμη και στις περιφερειακές συσκευές του υπολογιστή με διάφορες αναπαραστάσεις (representations). 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 επιπέδου (high level programming languages) επιτρέπουν τη χρήση διάφορων τύπων (types) μεταβλητών (variables) 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 (data structure) είναι ένα σύνολο δεδομένων μαζί με ένα σύνολο επιτρεπτών λειτουργιών επί αυτών. Για παράδειγμα, μία τέτοια δομή είναι η εγγραφή (record), που μπορεί να περιγράφει ένα είδος, ένα πρόσωπο κλπ. Η εγγραφή αποτελείται από τα πεδία (fields) που αποθηκεύουν χαρακτηριστικά (attributes) διαφορετικού τύπου, όπως για παράδειγμα ο κωδικός, η περιγραφή κλπ. 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

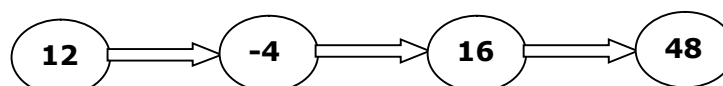
Τρόποι καταγραφής και αλληλοσυσχέτισης των δεδομένων μελετώνται έτσι ώστε να αναπαρασταθεί η γνώση για πραγματικά γεγονότα. Οι τεχνολογίες των Βάσεων Δεδομένων (Databases), της Μοντελοποίησης Δεδομένων (Data Modeling) και της Αναπαράστασης Γνώσης (Knowledge Representation) ανήκουν σε αυτή τη σκοπιά μελέτης των δεδομένων.

3.2 Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

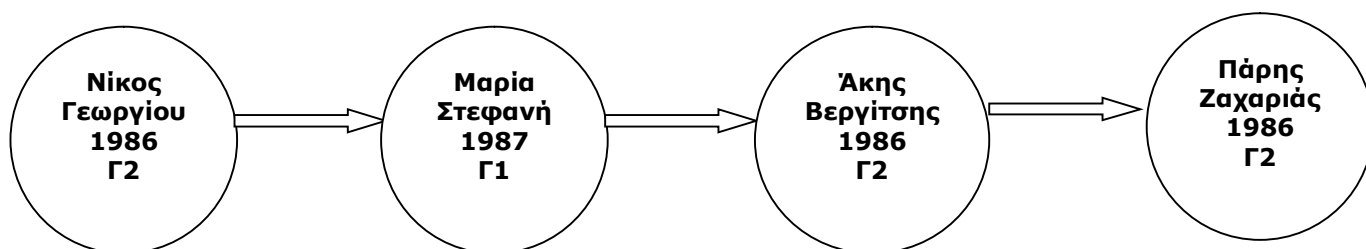
Δομή Δεδομένων: Είναι ένας τρόπος αποθήκευσης (οργάνωσης) δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.

Η δομές αποτελείται από ένα σύνολο κόμβων. Κάθε κόμβος περιέχει μία απλή τιμή (αριθμό, αλφάβητο κλπ) ή μία πιο σύνθετη τιμή (π.χ. μία εγγραφή).

π.χ. δομή λίστας που περιέχει ακέραιους αριθμούς:



Π.χ. δομή λίστας που περιέχει εγγραφές (records. Εγγραφή = Μία ομάδα τιμών διαφορετικού τύπου, που περιγράφουν ένα αντικείμενο. Π.χ. μία εγγραφή μαθητή περιέχει το όνομα, επώνυμο, έτος γέννησης, τάξη κλπ).



Βασικές λειτουργίες (ή πράξεις) επί των δομών δεδομένων:

- **Προσπέλαση (access)**
Είναι η δυνατότητα πρόσβαση σε ένα κόμβο με σκοπό την ανάγνωση ή τροποποίηση του περιεχομένου του.
- **Εισαγωγή (insertion)**
Προσθήκη νέου κόμβου στη δομή.
- **Διαγραφή (deletion)**
Αφαιρούμε έναν κόμβο από τη δομή.
- **Αναζήτηση (searching)**
Προσπελαύνονται οι κόμβοι μίας δομής με σκοπό να εντοπιστεί κάποιος που περιέχει μία συγκεκριμένη τιμή.
- **Ταξινόμηση (sorting)**
Διατάσσουμε τους κόμβους της δομής σε αύξουσα ή φθίνουσα σειρά (π.χ. αλφάβητα κατά επώνυμο, όνομα).
- **Αντιγραφή (copying)**
Όλοι ή μερικοί κόμβοι αντιγράφονται σε μία άλλη δομή.
- **Συγχώνευση (merging)**
Δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή.
- **Διαχωρισμός (separation)**
Μία δομή διασπάται σε δύο ή περισσότερες. Είναι το αντίγραφο της συγχώνευσης.

Για κάθε λειτουργία δημιουργείται και ένας αλγόριθμος (π.χ. αλγόριθμος αναζήτησης κάποιου στοιχείου). Αρκετές φορές υπάρχουν διαφορετικοί αλγόριθμοι που υλοποιούν μία συγκεκριμένη λειτουργία (π.χ. υπάρχουν αρκετοί αλγόριθμοι για τη λειτουργία της ταξινόμησης). Όταν συμβαίνει αυτό, επιλέγουμε τον αλγόριθμο που είναι πιο αποδοτικός (π.χ. πιο γρήγορος) για τα συγκεκριμένα δεδομένα.

Κανόνας: **Προγράμματα = Δομές δεδομένων + Αλγόριθμοι**

Κατηγορίες δομών δεδομένων:

Στατικές

- Το μέγεθος τους είναι καθορισμένο (σταθερό) και δεν μεταβάλλεται κατά τη διάρκεια εκτέλεσης του προγράμματος.
- Οι κόμβοι αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη.

Δυναμικές

- Το μέγεθος τους δεν είναι καθορισμένο (δεν είναι σταθερό) και μεταβάλλεται κατά τη διάρκεια εκτέλεσης του προγράμματος. Μπορούν να εισάγονται νέοι κόμβοι και να διαγράφονται υπάρχοντες.
- Το μέγεθος τους δεν είναι καθορισμένο (δεν είναι σταθερό) και μεταβάλλεται κατά τη διάρκεια εκτέλεσης του προγράμματος. Μπορούν να εισάγονται νέοι κόμβοι και να διαγράφονται υπάρχοντες.
- Οι κόμβοι δεν αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη.

3.3 Πίνακες

Χαρακτηριστικός εκπρόσωπος των στατικών δομών. Περιέχει ένα σταθερό σύνολο κόμβων (θέσεις) που αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη. Επίσης, όλα τα στοιχεία είναι του ίδιου τύπου (δηλαδή ακέραιοι, πραγματικοί κλπ).

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. Αναφορά σε ατομικά στοιχεία του πίνακα γίνεται με το όνομα του πίνακα ακολουθούμενο από ένα δείκτη.

9.1 Μονοδιάστατοι πίνακες (μίας γραμμής ή μίας στήλης)

Οι πίνακες που χρησιμοποιούν ένα μόνο δείκτη για την αναφορά των στοιχείων τους, ονομάζονται **μονοδιάστατοι** πίνακες. Το όνομα του πίνακα μπορεί να είναι οποιοδήποτε δεκτό όνομα του **αλγόριθμο** και ο δείκτης είναι μία ακέραια έκφραση, σταθερή ή μεταβλητή που περικλείεται μέσα στα σύμβολα **[]**.

Οι αριθμοί πάνω από τις θέσεις του πίνακα δηλώνουν τον αριθμό θέσης του πίνακα. Ο αριθμός θέσης λέγεται και **δείκτης θέσης**. Ο πίνακας ορίζεται ως εξής: **όνομα_πίνακα [διαστάσεις]**. π.χ. **ακέραιος Π [5]** δηλώνει μονοδιάστατο πίνακα μίας γραμμής πέντε θέσεων (στήλης) που περιέχει ακραίους.

Γενικά, ορίζεται ως **Π [N]**

π.χ. Πίνακας ακεραίων **Π [5]**

1	2	3	4	5
12	-4	22	45	-7

π.χ. Πίνακας αλφαριθμητικών **Π [4]**

1	2	3	4
"Γιάννης"	"Αλέκα"	"Μάριος"	"Ακης"

Για να αναφερθούμε στο περιεχόμενο μίας θέσης του πίνακα βάζουμε το **όνομα[θέση]**.

Για παράδειγμα, **Π[1]** = το περιεχόμενο της θέσης 1, **Π[6]** = το περιεχόμενο της θέσης 6.

Για να αναφερθούμε στο περιεχόμενο μίας θέσης του πίνακα βάζουμε το **όνομα[θέση]**.

Για παράδειγμα, **Π[1]** = το περιεχόμενο της θέσης 1, **Π[6]** = το περιεχόμενο της θέσης 6.

9.3 Πολυδιάστατοι πίνακες (πολλών γραμμών και στηλών)

Οι μονοδιάστατοι πίνακες, ως γνωστό, έχουν μια γραμμή και πολλές στήλες. Όταν, έχουμε περισσότερες από μια γραμμές σε έναν πίνακα, τότε ο πίνακας αυτός ονομάζεται πολυδιάστατος. Για τους πίνακες 2-διαστάσεων απαιτούνται 2 δείκτες (στους μονοδιάστατους πίνακες έχουμε ένα δείκτη). Ο πρώτος δείκτης αναφέρεται στις γραμμές και ο δεύτερος στις στήλες.

Εδώ έχουμε περισσότερες από μία γραμμές. Κάθε γραμμή έχει ένα σύνολο θέσεων (στήλες).

Στο παρακάτω παράδειγμα ο πίνακας ορίζεται ως εξής: **ακέραιος Π [3,4]**. Η πρώτη διάσταση αναφέρεται στις γραμμές και η δεύτερη στις στήλες.

Γενικά, ορίζεται ως $\Pi [M, N]$ για έναν πίνακα $M \times N$.

	1	2	3	4
1	$\Pi[1,1]$	$\Pi[1,2]$	$\Pi[1,3]$	$\Pi[1,4]$
2	$\Pi[2,1]$	$\Pi[2,2]$	$\Pi[2,3]$	$\Pi[2,4]$
3	$\Pi[3,1]$	$\Pi[3,2]$	$\Pi[3,3]$	$\Pi[3,4]$

Για να αναφερθούμε στο **περιεχόμενο μίας θέσης** του δισδιάστατου πίνακα βάζουμε το **όνομα[γραμμή, στήλη]**. Δηλαδή, η θέση προσδιορίζεται από τον αριθμό γραμμής και στήλης. Για παράδειγμα, $\Pi[1,2]$ = το περιεχόμενο της θέσης στη γραμμή 1 και στήλη 2, $\Pi[3,2]$ = το περιεχόμενο της θέσης στη γραμμή 3 και στήλη 2.

Τυπικές επεξεργασίες (λειτουργίες) σε έναν πίνακα:

- **Διάβασμα** των στοιχείων του πίνακα, δηλαδή εισαγωγή τιμών στις θέσεις του.
- **Εκτύπωση** των στοιχείων του πίνακα.
- Υπολογισμός του **αθροίσματος** των στοιχείων του.
- Υπολογισμός του **μέσου όρου** των στοιχείων του.
- Εύρεση του **ελάχιστου** ή **μέγιστου** στοιχείου του.
- **Αναζήτηση** ενός στοιχείου.
- **Ταξινόμηση** του πίνακα.
- **Συγχώνευση** δύο πινάκων.

Στους αλγόριθμους του πίνακα χρησιμοποιούμε, ως επί το πλείστον, τη δομή **Για...από...μέχρι...**

9.1 Μονοδιάστατοι πίνακες

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.

Οι πίνακες που χρησιμοποιούν ένα μόνο δείκτη για την αναφορά των στοιχείων τους, ονομάζονται **μονοδιάστατοι** πίνακες. Το όνομα του πίνακα μπορεί να είναι οποιοδήποτε δεκτό όνομα του **Αλγόριθμου** και ο δείκτης είναι μία ακέραια έκφραση, σταθερή ή μεταβλητή που περικλείεται μέσα στα σύμβολα [και].

9.2 Πότε πρέπει να χρησιμοποιούνται πίνακες

Όταν τα δεδομένα που εισάγουμε, απαιτείται να βρίσκονται στη μνήμη RAM καθ' όλη τη διάρκεια εκτέλεσης του προγράμματος.

Πλεονεκτήματα

Η χρήση πινάκων είναι ένας βολικός τρόπος για τη διαχείριση πολλών δεδομένων ίδιου τύπου, αλλά συχνά η χρήση τους είναι περιττή και επιζήμια στην ανάπτυξη του προγράμματος.

Μειονεκτήματα

1. **Οι πίνακες απαιτούν μνήμη.** Κάθε πίνακας δεσμεύει από την αρχή του προγράμματος πολλές θέσεις μνήμης. Σε ένα μεγάλο και σύνθετο πρόγραμμα η άσκοπη χρήση μεγάλων πινάκων μπορεί να οδηγήσει ακόμη και σε αδυναμία εκτέλεσης του προγράμματος.
2. **Οι πίνακες περιορίζουν τις δυνατότητες του προγράμματος.** Στο πρόγραμμα υπάρχει ανώτατο όριο στο πλήθος των αριθμών ίσο με ένα σταθερό αριθμό. Αυτό γιατί οι πίνακες είναι στατικές δομές και το μέγεθος τους πρέπει να δηλώνεται στην αρχή του προγράμματος, ενώ παραμένει υποχρεωτικά σταθερό κατά την εκτέλεση του προγράμματος.

9.4 Τυπικές επεξεργασίες πινάκων

Τα προγράμματα τα οποία χρησιμοποιούν πίνακες πολύ συχνά απαιτούν συγκεκριμένες επεξεργασίες στα στοιχεία του πίνακα. Οι τυπικές αυτές επεξεργασίες είναι:

- **Υπολογισμός αθροισμάτων στοιχείων του πίνακα**
Πολύ συχνά απαιτείται ο υπολογισμός του αθροίσματος στοιχείων του πίνακα που έχουν κοινά χαρακτηριστικά για παράδειγμα βρίσκονται στην ίδια στήλη ή στην ίδια γραμμή.
- **Εύρεση του μέγιστου ή του ελάχιστου στοιχείου**
Αν ο πίνακας δεν είναι ταξινομημένος, τότε πρέπει να συγκριθούν τα στοιχεία ένα προς ένα, για να βρεθεί το μέγιστο ή το ελάχιστο. Αν ο πίνακας είναι ταξινομημένος, τότε προφανώς το μέγιστο και το ελάχιστο βρίσκονται στα δύο ακριανά στοιχεία του πίνακα.
- **Αναζήτηση ενός στοιχείου του πίνακα**
Δύο είναι οι πλέον διαδεδομένοι αλγόριθμοι αναζήτησης:
 - **Η σειριακή αναζήτηση**
 - **Η δυαδική αναζήτηση**
 Η **σειριακή μέθοδος** αναζήτησης είναι η πιο απλή, αλλά και η λιγότερη αποτελεσματική μέθοδος. Χρησιμοποιείται όμως υποχρεωτικά για πίνακες που δεν είναι ταξινομημένοι. Αντίθετα η **δυαδική αναζήτηση** χρησιμοποιείται μόνο σε ταξινομημένους πίνακες και είναι σαφώς αποδοτικότερη από τη σειριακή μέθοδο.
- **Ταξινόμηση των στοιχείων του πίνακα**
Η επιλογή του καλύτερου αλγόριθμου εξαρτάται κυρίως από το πλήθος των στοιχείων του πίνακα και την αρχική τους διάταξη, αν δηλαδή ο πίνακας είναι τελείως αταξινόμητος ή μερικώς ταξινομημένος.
- **Διαχωρισμός ενός πίνακα σε δύο ή περισσότερους**
- **Συγχώνευση δύο πινάκων**
Σκοπός της είναι η δημιουργία από τα στοιχεία δύο (ή περισσότερων) ταξινομημένων πινάκων ενός άλλου, που είναι και αυτός ταξινομημένος.

Παράλληλοι πίνακες

Όταν έχουμε εγγραφές με πεδία διαφορετικού τύπου οι οποίες καταχωρούνται σε πίνακες τότε κάθε πεδίο είναι ένας διαφορετικός πίνακας, λόγω διαφορετικού τύπου. Οι πίνακες αυτοί συνδέονται μεταξύ τους με τέτοιο τρόπο ώστε η κάθε εγγραφή να έχει όλα τα στοιχεία της με την ίδια τιμή δείκτη, δηλ. να βρίσκονται

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.

Όταν ταξινομούμε παράλληλους πίνακες, κάνουμε τον έλεγχο ως προς τον πίνακα που ζητείται η ταξινόμηση, αλλά στην αντιμετάθεση των στοιχείων αντιμετωπίζουμε τις αντίστοιχες θέσεις όλων των παράλληλων πινάκων ως προς αυτόν που γίνεται η ταξινόμηση.

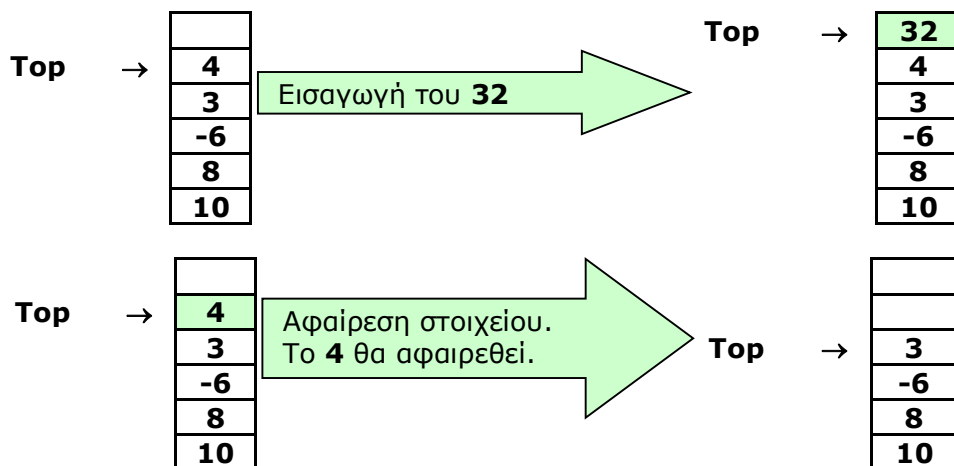
3.4 Στοιίβα

Η στοιίβα υλοποιεί τη λογική **LIFO (Last In First Out)**. Τα δεδομένα **εισάγονται** στην **κορυφή** της στοιίβας ενώ η **αφαίρεση ενός στοιχείου** γίνεται **πάντα** από την **κορυφή** της στοιίβας. Η στατική στοιίβα υλοποιείται με μονοδιάστατο πίνακα και χρησιμοποιεί ένα **δείκτη Top** που δείχνει στην κορυφή της στοιίβας.

Οι λειτουργίες είναι δύο:

- Η **εισαγωγή-ώθηση (Push)** ενός στοιχείου στην κορυφή της στοιίβας.
- Η **εξαγωγή-απώθηση (Pop)** ενός στοιχείου από την κορυφή της στοιίβας.

Παράδειγμα:



Σημείωση: Κατά την εισαγωγή ενός στοιχείου (ώθηση), πρέπει να γίνεται έλεγχος μήπως η στοιίβα είναι γεμάτη (δηλαδή υπάρχει άλλη ελεύθερη θέση στον πίνακα). Δηλαδή, ελέγχει μήπως συμβεί **υπερχείλιση (overflow)**. Αντίστοιχα, κατά την αφαίρεση ενός στοιχείου (απώθηση), πρέπει να γίνεται έλεγχος μήπως η στοιίβα είναι άδεια. Δηλαδή, ελέγχει μήπως συμβεί **υποχείλιση (underflow)**.

3.5 Ουρά

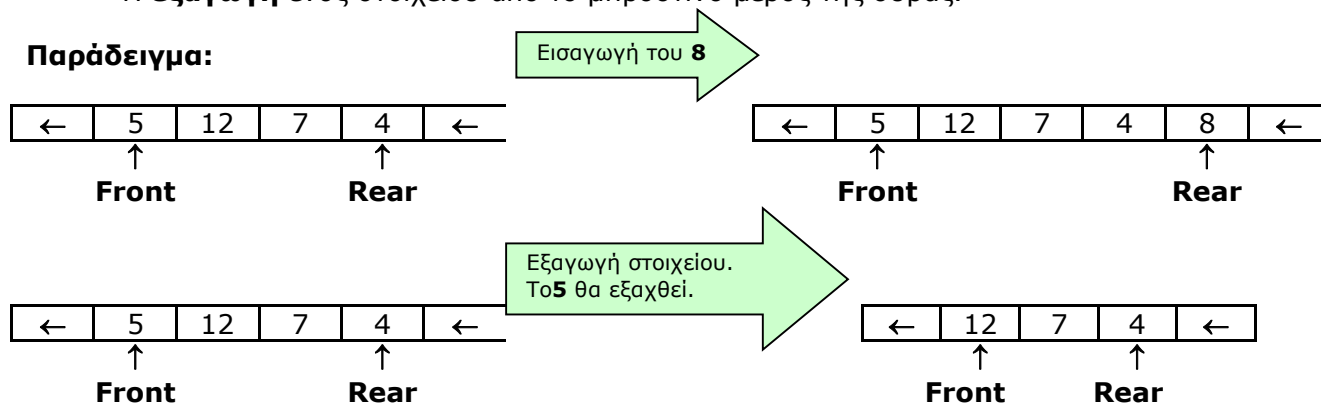
Η ουρά υλοποιεί τη λογική **FIFO (First In First Out)**. Τα δεδομένα εισάγονται στο πίσω μέρος της ουράς ενώ η εξαγωγή ενός στοιχείου γίνεται πάντα από το μπροστινό μέρος της ουράς (όπως συμβαίνει σε μία ουρά σε τράπεζα).

Η στατική ουρά υλοποιείται με μονοδιάστατο πίνακα και χρησιμοποιεί δύο δείκτες: Τον **Front** που δείχνει στο μπροστινό μέρος της ουράς και τον **Rear** που δείχνει στο πίσω μέρος της ουράς.

Οι λειτουργίες είναι δύο:

- Η **εισαγωγή** ενός στοιχείου στο πίσω μέρος της ουράς.
- Η **εξαγωγή** ενός στοιχείου από το μπροστινό μέρος της ουράς.

Παράδειγμα:



Σημείωση: Κατά την εισαγωγή ενός στοιχείου, πρέπει να γίνεται έλεγχος μήπως η ουρά είναι γεμάτη (δηλαδή δεν υπάρχει άλλη θέση στον πίνακα). Αντίστοιχα, κατά την εξαγωγή ενός στοιχείου, πρέπει να γίνεται έλεγχος μήπως η ουρά είναι άδεια.

3.6 Αναζήτηση

Αλγόριθμος σειριακής αναζήτησης

Εδώ θα δούμε την **σειριακή μέθοδο**. Η λογική είναι η εξής: Σαρώνουμε τον πίνακα και εξετάζουμε σε κάθε θέση αν το στοιχείο της θέσης αυτής είναι ίσο με αυτό που ψάχνουμε. Αν ναι, τότε σταματάμε.

Πότε χρησιμοποιούμε την μέθοδο αυτή;

- Όταν ο πίνακας είναι μη ταξινομημένος
- Ο πίνακας είναι μικρού μεγέθους (π.χ. $N \leq 20$)
- Η αναζήτηση να πραγματοποιείται σπάνια, διότι η μέθοδος αυτή είναι σχετικά αργή.

Θα χρειαστούμε μία **μεταβλητή λογικού τύπου**, που θα γίνει αληθής αν το στοιχείο βρεθεί. Επίσης, δεν γνωρίζουμε σε ποια θέση βρίσκεται. Μπορεί να είναι στην 1^η θέση, αλλά μπορεί να είναι στη N^{τη} θέση. Θα χρησιμοποιήσουμε και **μία μεταβλητή θέση** που θα κρατήσει τη θέση όπου βρέθηκε.

Αλγόριθμος Σειριακή_Αναζήτηση_Στοιχείου_Πίνακα

! σελίδα 64 του βιβλίου

! σελίδα 47 του βιβλίου Β' Λυκείου

Δεδομένα // n, table //

Γράψε "Δώσε αριθμό για αναζήτηση στον πίνακα"

Διάβασε key

done ← Ψευδής

position ← 0

i ← 1

Όσο (done = Ψευδής) **και** (i ≤ n) **επανάλαβε**

! όσο δεν είμαστε στο τέλος του πίνακα και δεν βρέθηκε

Αν table[i] = key **τότε**

done ← Αληθής

position ← i

αλλιώς

i ← i + 1

Τέλος_αν

Τέλος_επανάληψης

Αν (done = Αληθής) **τότε**

Γράψε "Το στοιχείο ", key, " βρέθηκε στη θέση ", position

αλλιώς

Γράψε "Το στοιχείο ", key, " δεν βρέθηκε στον δοθέντα πίνακα"

Τέλος_αν

Τέλος Σειριακή_Αναζήτηση_Στοιχείου_Πίνακα

Παρατηρήσεις

1. Η αναζήτηση σταματά μόλις εντοπίσει κάποιο στοιχείο που είναι ίσο με την αναζητούμενη τιμή. Ο αλγόριθμος **Παραλλαγή Νο1** στη συνέχεια αποτελεί τροποποίηση του αλγορίθμου σειριακής αναζήτησης ώστε να καλύπτει την αδυναμία αυτή και συνεχίζει την αναζήτηση ώστε να εντοπίσει όλες τις τιμές του πίνακα table που έχουν τιμή ίση με τη μεταβλητή key.

Παραλλαγή Νο1: Τροποποίηση του αλγορίθμου σειριακής αναζήτησης ώστε να αναζητά όλες τις θέσεις που βρίσκεται η αναζητούμενη τιμή

Για την αποφυγή τερματισμού του βρόχου αναζήτησης, εξαλείφουμε την μεταβλητή done από ολόκληρο το πρόγραμμα. Είναι προφανές ότι η τιμή που αναζητούμε στον πίνακα (μεταβλητή key) μπορεί να βρίσκεται σε περισσότερες από μια θέσεις του πίνακα table. Αν δεν επιθυμούμε περαιτέρω επεξεργασία των θέσεων αυτών απλά τις εκτυπώνουμε

Ωστόσο, υπάρχει η περίπτωση οι θέσεις αυτές να πρέπει να αποθηκευτούν ώστε να χρησιμοποιηθούν σε κάποιο άλλο σημείο του αλγορίθμου. Πώς πρέπει να αποθηκεύσουμε τις θέσεις αυτές; Δεδομένου ότι δεν γνωρίζουμε εξ αρχής το πλήθος των δεδομένων δεν μπορούμε

να χρησιμοποιήσουμε μεταβλητές για το σκοπό αυτό. Κατά συνέπεια, πρέπει να χρησιμοποιήσουμε έναν άλλο πίνακα (έστω με όνομα POSITION_TABLE) ώστε να καταχωρούνται σε αυτόν οι θέσεις του πίνακα table που εντοπίστηκε η τιμή της μεταβλητής key

Το μέγεθος του πίνακα POSITION_TABLE είναι η καθώς πρέπει να καλύψουμε την ακραία περίπτωση να υπάρχουν και στις η θέσεις του πίνακα table η τιμή που έχει η μεταβλητή key. Η μεταβλητή count_found θα περιέχει το πλήθος των θέσεων που ικανοποιούν την αναζήτηση. Ο αλγόριθμος παρουσιάζεται εναλλακτικά με δυο τρόπους. Αφού επιθυμούμε να σαρώσουμε ολόκληρο τον πίνακα αυτό μπορεί να γίνει με την δομή επανάληψης Για...από...μέχρι αντί για την Όσο...επανάλαβε που χρησιμοποιήθηκε στα προηγούμενα παραδείγματα.

Αλγόριθμος Σειριακή_Αναζήτηση_στοιχείου_σε_όλο_τον_πίνακα

! σελίδα 64 του βιβλίου

! Δε σταματά την αναζήτηση με την εύρεση του στοιχείου

! αλλά συνεχίζει για να εντοπίσει και άλλες εμφανίσεις του

Δεδομένα // n, table //

Γράψε "Δώσε αριθμό για αναζήτηση στον πίνακα"

Διάβασε key

i ← 1

count_found ← 0

Όσο (i ≤ n) **επανάλαβε** *! όσο δεν είμαστε στο τέλος του πίνακα*

Αν table[i] = key **τότε** *! αν το i στοιχείο ισούται με αυτό που ψάχνουμε τότε*
 count_found ← count_found + 1 *! βρέθηκε*

! position_table[count_found] ← i

! αποθηκεύει τους αριθμούς των θέσεων στις οποίες βρέθηκε το στοιχείο

Γράψε "Το στοιχείο ", key, " βρέθηκε στη θέση ", i

Τέλος_αν

 i ← i + 1

! προχωρά στην επόμενη θέση

Τέλος_επανάληψης

Αν (count_found ≠ 0) **τότε**

Γράψε "Το στοιχείο ", key, " βρέθηκε σε ", count_found, " θέσεις"

αλλιώς

Γράψε "Το στοιχείο ", key, " δεν βρέθηκε στον δοθέντα πίνακα"

Τέλος_αν

Τέλος Σειριακή_Αναζήτηση_στοιχείου_σε_όλο_τον_πίνακα

ή εναλλακτικά

Αλγόριθμος Σειριακή_Αναζήτηση_στοιχείου_σε_όλο_τον_πίνακα

! σελίδα 64 του βιβλίου

! Δε σταματά την αναζήτηση με την εύρεση του στοιχείου

! αλλά συνεχίζει για να εντοπίσει και άλλες εμφανίσεις του

Δεδομένα // n, table //

Γράψε "Δώσε αριθμό για αναζήτηση στον πίνακα"

Διάβασε key

count_found ← 0

Για i **από** 1 **μέχρι** n

! όσο δεν είμαστε στο τέλος του πίνακα

Αν table[i] = key **τότε**

! αν το i στοιχείο ισούται με αυτό που ψάχνουμε τότε

 count_found ← count_found + 1

! βρέθηκε

! position_table[count_found] ← i

! αποθηκεύει τους αριθμούς των θέσεων στις οποίες βρέθηκε το στοιχείο

Γράψε "Το στοιχείο ", key, " βρέθηκε στη θέση ", i

Τέλος_αν

Τέλος_επανάληψης

Αν (count_found ≠ 0) **τότε**

Γράψε "Το στοιχείο ", key, " βρέθηκε σε ", count_found, " θέσεις"

αλλιώς

Γράψε "Το στοιχείο ", key, " δεν βρέθηκε στον δοθέντα πίνακα"

Τέλος_αν

Τέλος Σειριακή_Αναζήτηση_στοιχείου_σε_όλο_τον_πίνακα

Παρατηρήσεις

ο παραπάνω Αλγόριθμος χρησιμοποιείται όταν ο πίνακας table δεν είναι ταξινομημένος.

Ο αλγόριθμος **Παραλλαγή No2** στη συνέχεια αποτελεί τροποποίηση του αλγορίθμου σειριακής αναζήτησης ώστε να καλύπτει την αδυναμία αυτή και σταματά την αναζήτηση μόλις συναντήσει κάποιο στοιχείο του πίνακα table που είναι μεγαλύτερο από το ζητούμενο (μεταβλητή key) σε ταξινομημένο πίνακα.

Παραλλαγή No2: Τροποποίηση του αλγορίθμου σειριακής αναζήτησης ώστε να λειτουργεί βέλτιστα σε ταξινομημένο πίνακα

Αλγόριθμος Σειριακή_Αναζήτηση_στοιχείου_σε_Ταξινομημένο_Πίνακα

! σελίδα 64 του βιβλίου

! Για ταξινομημένο πίνακα με αύξουσα διάταξη

Δεδομένα // n, table //

Γράψε "Δώσε αριθμό για αναζήτηση στον πίνακα"

Διάβασε key

done ← **Ψευδής**

position ← 0

i ← 1

Όσο (done = **Ψευδής**) **και** ($i \leq n$) **επανάλαβε** *! όσο δεν είμαστε στο τέλος του πίνακα και δεν βρέθηκε*

Αν table[i] > key **τότε** *! σταμάτα την επανάληψη, δε θα βρεθεί το στοιχείο*

done ← **Αληθής**

αλλιώς_αν table[i] = key **τότε** *! αν το i στοιχείο ισούται με αυτό που ψάχνουμε τότε*

done ← **Αληθής**

position ← i

αλλιώς

i ← i + 1

Τέλος_αν

Τέλος_επανάληψης

Αν (position ≠ 0) **τότε**

Γράψε "Το στοιχείο ", key, " βρέθηκε στη θέση ", position

αλλιώς

Γράψε "Το στοιχείο ", key, " δεν βρέθηκε στον δοθέντα πίνακα"

Τέλος_αν

Τέλος Σειριακή_Αναζήτηση_στοιχείου_σε_Ταξινομημένο_Πίνακα

Παρατηρήσεις:

Η αναζήτηση σταματά όταν η τιμή προς αναζήτηση (μεταβλητή key) είναι μικρότερη από την τρέχουσα τιμή του πίνακα table.

Σε αυτήν την περίπτωση δεν έχει νόημα να συνεχιστεί η αναζήτηση καθώς δεν αναμένεται να έχει επιτυχές αποτέλεσμα.

4	9	12	19	23	45	53	67
---	---	----	----	----	----	----	----

Για παράδειγμα η αναζήτηση για την τιμή 14 στον παραπάνω πίνακα δεν έχει νόημα να συνεχιστεί μετά την 4^η επανάληψη ($i=4$, table[i]=19) καθώς ο αριθμός 14 που είναι μικρότερος του 19 αποκλείεται να βρίσκεται σε κάποια από τις επόμενες θέσεις του πίνακα. Έτσι, αποφεύγονται άσκοπες επαναλήψεις. Το αν εντοπίστηκε ή όχι η τιμή key το ανιχνεύουμε με τη μεταβλητή position και όχι με την done καθώς η τελευταία λαμβάνει την τιμή αληθής ακόμη κι αν δεν ευρέθη η key.

Αλγόριθμος δυαδικής αναζήτησης

Σημείωση: Όταν ο πίνακας είναι **ταξινομημένος** είναι προτιμότερο να χρησιμοποιείται μία πιο αποδοτική (γρήγορη) μέθοδο που ονομάζεται **δυαδική αναζήτηση**.

Αλγόριθμος Δυαδική_Αναζήτηση_Στοιχείου_σε_Ταξινομημένο_Πίνακα

! σελίδα 64 του βιβλίου

! Για ταξινομημένο πίνακα με αύξουσα διάταξη

Δεδομένα // n, table //

Γράψε "Δώσε αριθμό για αναζήτηση στον πίνακα"

Διάβασε key

done ← **Ψευδής**

position ← 0

first ← 1

end ← n

Όσο (first < end) **και** (done = **Ψευδής**) **επανάλαβε**

 i ← (first + end) div 2

Αν table[i] = key **τότε**

 done ← **Αληθής**

 position ← i

αλλιώς

Αν table[i] > key **τότε**

 end ← i - 1

αλλιώς

 first ← i + 1

Τέλος_αν

Τέλος_αν

Τέλος_επανάληψης

Αν table[i + 1] = key **τότε** *! αναζήτηση μίας επιπλέον θέσης για την περίπτωση*

ζυγού αριθμού στοιχείων

 done ← **Αληθής**

 position ← i + 1

Τέλος_αν

Αν (done = **Αληθής**) **τότε**

Γράψε "Το στοιχείο ", key, " βρέθηκε στη θέση ", position

αλλιώς

Γράψε "Το στοιχείο ", key, " δεν βρέθηκε στον δοθέντα πίνακα "

Τέλος_αν

Τέλος Δυαδική_Αναζήτηση_Στοιχείου_σε_Ταξινομημένο_Πίνακα

3.7 Ταξινόμηση

Η μέθοδος της φυσαλίδας (Bubble Sort) ή ευθείας ανταλλαγής. Βασίζεται στην αρχή της σύγκρισης γειτονικών στοιχείων του πίνακα και ανταλλαγής τους μέχρι να διαταχθούν όλα σε μία σειρά (αύξουσα ή φθίνουσα).

Η λογική είναι η εξής: Κάνουμε διαδοχικές σαρώσεις στον πίνακα. Σε κάθε σάρωση, το μικρότερο στοιχείο (για αύξουσα) μετακινείται προς την κορυφή του πίνακα αφού γίνουν όλες οι σαρώσεις θα έχει επιτευχθεί η ταξινόμηση.

Αλγόριθμος Ταξινόμηση_Στοιχείων_Πίνακα_Φυσαλίδας

! σελίδα 67 του βιβλίου

! σελίδα 56 των οδηγιών μελέτης

! Ξεκινώντας από το τελευταίο στοιχείο του πίνακα

! σε κάθε πέρασμα αν κάποιο στοιχείο είναι μικρότερο από το προηγούμενο του

! πηγαίνει προς τα μπροστά με αντιμετάθεση τους

Δεδομένα // n, table //

Για i από 2 μέχρι n

! το i σαρώνει από τη 2η θέση έως το τέλος

Για j από n μέχρι i με_βήμα -1

! το j από το τέλος προς τη θέση i

Αν table[j] < table[j - 1] **τότε**

! αν το j στοιχείο είναι μικρότερο από το j-1 στοιχείο

! Αντιμετάθεσε table[j - 1], table[j]

! τότε κάνε αντιμετάθεση στοιχείων.

! Δηλαδή, το j στοιχείο θα πάει στη θέση του j-1 και το j-1 στη θέση του j.

temp ← table[j - 1]

table[j - 1] ← table[j]

table[j] ← temp

Τέλος_αν

Τέλος_επανάληψης

Τέλος_επανάληψης

Αποτελέσματα // table //

Τέλος Ταξινόμηση_Στοιχείων_Πίνακα_Φυσαλίδας

Παρατηρήσεις:

- Η εντολή αντιμετάθεσε ορίζεται στον αλγόριθμο και μπορούμε να τη χρησιμοποιήσουμε σε αλγορίθμους αλλά καλό να την αποφεύγουμε καθώς δεν ορίζεται στη ΓΛΩΣΣΑ και δεν μπορούμε να τη χρησιμοποιήσουμε σε προγράμματα.
- Η διάταξη της ταξινόμησης είναι **αύξουσα**. Για φθίνουσα ταξινόμηση, αρκεί να αντιστραφεί η φορά της συνθήκης της δομής επιλογής Αν
- Ο πίνακας table είναι μονοδιάστατος

Παρατηρούμε στο επόμενο σχήμα την «εικόνα» του πίνακα στο τέλος κάθε επανάληψης της εξωτερικής δομής Για. Από την τελευταία θέση του πίνακα έως την i, αν η τιμή κάποιου κελιού του πίνακα είναι μικρότερη από αυτή του επόμενου κελιού, τότε αντιμεταθέτουμε τις τιμές τους. Έτσι, αν υπάρχει κάποιος μικρός αριθμός σε "χαμηλή" θέση στον πίνακα διαδοχικά "ανεβαίνει" σε "υψηλότερες" θέσεις όπως μια φυσαλίδα στο υγρό.

Αρχικά κλειδιά									Τελικά κλειδιά
i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9		
52	5	5	5	5	5	5	5	5	
12	52	10	10	10	10	10	10	10	
71	12	52	12	12	12	12	12	12	
56	71	12	52	19	19	19	19	19	
5	56	71	19	52	45	45	45	45	
10	10	56	71	45	52	52	52	52	
19	19	19	56	71	56	56	56	56	
90	45	45	45	56	71	71	71	71	
45	90	90	90	90	90	90	90	90	

Ας δούμε όμως και ένα άλλο παράδειγμα, θέλουμε να ταξινομήσουμε τον πίνακα:

3	16	11	1	8
---	----	----	---	---

$i = 2$

$j = 5 \quad j = 4 \quad j = 3 \quad j = 2$

3	3	3	1
16	16	1	3
11	1	16	16
1	11	11	11
8	8	8	8

Το εξωτερικό Για επιβάλλει την εκτέλεση 4 βημάτων. Αυτά είναι τα εξής:

Αλλαγή: Όχι Ναι Ναι Ναι

$i = 3$

$j = 5 \quad j = 4 \quad j = 3$

1	1	1
3	3	3
16	8	8
8	16	16
11	11	11

Παρατηρούμε, ότι ο μικρότερος αριθμός έχει βρεθεί στην πρώτη θέση του πίνακα. Στη συνέχεια:

Αλλαγή: Ναι Ναι Όχι

$i = 4$

$j = 5 \quad j = 4$

1	1
3	3
8	8
11	11
16	16

Παρατηρούμε, ότι οι δυο πρώτες θέσεις περιέχουν τους δυο μικρότερους αριθμούς του πίνακα. Στη συνέχεια:

Αλλαγή: Ναι Όχι

$i = 5$

$j = 5$

1
3
8
11
16

Παρατηρούμε, ότι οι έχει ολοκληρωθεί η ταξινόμηση αλλά πρέπει να εκτελέσουμε ακόμη ένα βήμα που δεν θα προκαλέσει αλλαγές στον πίνακα

Αλλαγή: Όχι

Με την κατάλληλη τροποποίηση του αλγορίθμου της φουσαλίδας όπως παρουσιάζεται στο βιβλίο μπορούμε να αποφύγουμε την εκτέλεση περιττών βημάτων. Αυτό μπορεί να επιτευχθεί με τη βοήθεια μίας λογικής μεταβλητής, που σε κάθε νέα επανάληψη του εξωτερικού βρόχου

αρχικοποιείται ως ψευδής και αλλάζει ως αληθής αν σε κάποιο πέρασμα γίνει έστω και μία ανταλλαγή. Έτσι, αν σε κάποιο πέρασμα δεν εκτελεσθεί καμία ανταλλαγή, τότε η σημαία παραμένει ψευδής και αυτομάτως τελειώνει ο αλγόριθμος.

Παραλλαγή Νο1: Τροποποίηση του προγράμματος ταξινόμησης ευθείας ανταλλαγής ώστε να σταματάει με την ολοκλήρωση της ταξινόμησης

Αλγόριθμος Ταξινόμηση_Στοιχείων_Πίνακα_Φυσαλίδας

! σελίδα 67 του βιβλίου

! Ξεκινώντας από το τελευταίο στοιχείο του πίνακα

! σε κάθε πέρασμα αν κάποιο στοιχείο είναι μικρότερο από το προηγούμενο του

! πηγαίνει προς τα μπροστά με αντιμετάθεση τους

Δεδομένα // n, table //

i ← 2

! το i σαρώνει από τη 2η θέση έως το τέλος

Αρχή_επανάληψης

done ← ψευδής

Για j από n μέχρι i με_βήμα -1

! το j από το τέλος προς τη θέση i

Αν table[j] < table[j - 1] τότε

! αν το j στοιχείο είναι μικρότερο από το j-1 στοιχείο

temp ← table[j - 1]

! τότε κάνε αντιμετάθεση στοιχείων.

table[j - 1] ← table[j]

! Δηλαδή, το j στοιχείο θα πάει στη θέση του j-1 και το j-1 στη θέση του j.

table[j] ← temp

done ← Αληθής

Τέλος_αν

Τέλος_επανάληψης

i ← i + 1

Μέχρις_ότου (i > n) ή (done = Αληθής)

Αποτελέσματα // table //

Τέλος Ταξινόμηση_Στοιχείων_Πίνακα_Φυσαλίδας

Αλγόριθμος ταξινόμησης με επιλογή

Ο αλγόριθμος βασίζεται στην επιλογή του μικρότερου στοιχείου από αυτά που δεν έχουν ταξινομηθεί μέχρι τώρα.

Αλγόριθμος Ταξινόμηση_Στοιχείων_Πίνακα_με_Επιλογή

! σελίδα 48 του βιβλίου Β' Λυκείου

! Βρίσκει με Επιλογή το μικρότερο στοιχείο

! σε κάθε πέρασμα από αυτά που δεν έχουν ταξινομηθεί ακόμα

! και το τοποθετεί αμέσως μετά τα ήδη ταξινομημένα στοιχεία

Δεδομένα // n, table //

Για i από 1 μέχρι n

j ← i *! θεωρούμε στην αρχή ότι η θέση i περιέχει το μικρότερο στοιχείο*

min ← table[j]

Για k από i + 1 μέχρι n

Αν table[k] < min τότε

j ← k

! στο τέλος η θέση j περιέχει το μικρότερο στοιχείο

min ← table[j]

Τέλος_αν

Τέλος_επανάληψης

temp ← table[j]

table[j] ← table[i]

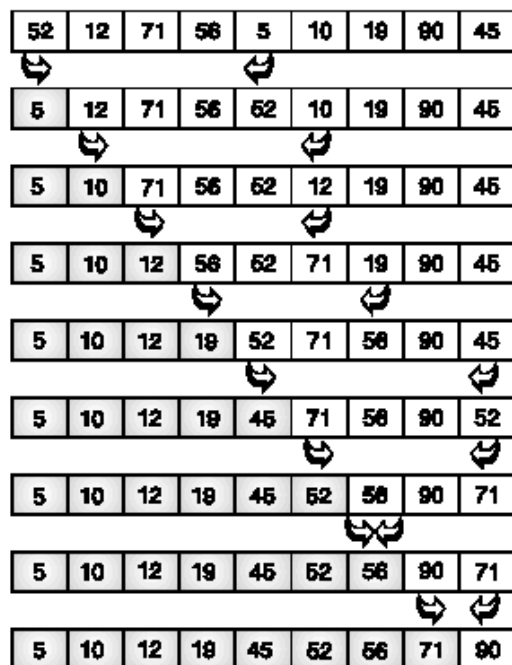
table[i] ← temp

Τέλος_επανάληψης

Αποτελέσματα // table //

Τέλος Ταξινόμηση_Στοιχείων_Πίνακα_με_Επιλογή

Για κάθε στοιχείο δηλαδή από το πρώτο μέχρι το τελευταίο, ελέγχεται ποιο από τα στοιχεία που ακολουθούν είναι μικρότερο και αν υπάρχει τέτοιο τα περιεχόμενα των δυο θέσεων αντιμετατίθενται. Η μέθοδος αυτή παρουσιάζεται στο επόμενο σχήμα καθώς εφαρμόζεται σε μονοδιάστατο πίνακα. Το ταξινομημένο τμήμα του πίνακα εμφανίζεται με σκίαση, ενώ με τα βέλη εμφανίζονται τα στοιχεία που ανταλλάσσονται αμοιβαία. Λόγου χάριν, στην πρώτη σειρά βρίσκουμε ότι το στοιχείο 5 είναι το μικρότερο και αντιμετατίθεται με το πρώτο στοιχείο του πίνακα, το 52. Έτσι προκύπτει η μορφή του πίνακα στη δεύτερη σειρά. Στη συνέχεια η διαδικασία προχωρεί με την ίδια λογική μέχρι την τελική ταξινόμηση του πίνακα



Αλγόριθμος ταξινόμησης ευθείας εισαγωγής

Ο αλγόριθμος αυτός είναι ιδανικός για περιπτώσεις δεδομένων "περίπου" ταξινομημένων.

Αλγόριθμος Ταξινόμηση_Στοιχείων_Πίνακα_με_Ευθεία_Εισαγωγή

! δραστηριότητα ΔΣ3 του Τεραδίου Μαθητή
! Για περίπου ταξινομημένους πίνακες
! Τοποθετεί κάθε φορά ένα στοιχείο
! σε κάθε πέρασμα από αυτά που δεν έχουν ταξινομηθεί ακόμα
! στη σωστή θέση με Ευθεία Εισαγωγή του στα ήδη ταξινομημένα στοιχεία

Δεδομένα // n, table //

Για i από 2 μέχρι n

temp ← table[i] *! κρατάμε στο temp το στοιχείο με το οποίο ασχολούμαστε κάθε φορά*

j ← i - 1

done ← Ψευδής

Όσο done = Ψευδής **επανάλαβε**

Αν j = 0 **τότε**

! φτάσαμε στην αρχή του πίνακα, άρα πρέπει να σταματήσουμε

done ← Αληθής

αλλιώς_αν temp < table[j] **τότε**

! βρήκαμε ένα μεγαλύτερο στοιχείο, άρα πρέπει να το μετακινήσουμε πιο πίσω

table[j+1] ← table[j]

j ← j - 1

αλλιώς

! πρέπει να σταματήσουμε την επανάληψη γιατί το στοιχείο θα μείνει σε αυτήν τη θέση

done ← Αληθής

Τέλος_αν

Τέλος_επανάληψης

table[j + 1] ← temp

Τέλος_επανάληψης

Αποτελέσματα // table //

Τέλος Ταξινόμηση_Στοιχείων_Πίνακα_με_Ευθεία_Εισαγωγή

Η τεχνική είναι η εξής: συγκρίνουμε τον δεύτερο με τον πρώτο αριθμό και αν χρειαστεί τους αντιμεταθέτουμε ώστε ο πρώτος να είναι μικρότερος. Στη συνέχεια ελέγχουμε τον τρίτο και τον τοποθετούμε στη σωστή θέση σε σχέση με τους 2 πρώτους (αν χρειαστεί μετακινούμε μια θέση τους αριθμούς που είναι μεγαλύτεροι από αυτόν ώστε να τοποθετηθεί). Το ίδιο επαναλαμβάνουμε για όλους τους αριθμούς. Ακολουθεί παράδειγμα:

52	12	71	56	5	10	19	90	45
12	52	71	56	5	10	19	90	45
12	52	71	56	5	10	19	90	45
12	52	56	71	5	10	19	90	45
5	12	52	56	71	10	19	90	45
5	10	12	52	56	71	19	90	45
5	10	12	19	52	56	71	90	45
5	10	12	19	52	56	71	90	45
5	10	12	19	45	52	56	71	90

Διαχωρισμός Πινάκων

Ο παρακάτω αλγόριθμος διαχωρίζει ένα πίνακα σε δύο άλλους από τους οποίους στον έναν τοποθετούνται οι θετικοί αριθμοί και στον άλλο οι αρνητικοί

Αλγόριθμος Διαχωρισμός_Πινάκων

! Διαχωρισμός Πινάκα σε 2 πίνακες :

! 1 πίνακα με τους αρνητικούς και 1 με τους θετικούς αριθμούς

Δεδομένα // n, table //

n1 ← 0

n2 ← 0

Για i από 1 μέχρι n

Αν (table[i] > 0) **τότε**

 n1 ← n1 + 1

 pos_table[n1] ← table[i]

αλλιώς

 n2 ← n2 + 1

 neg_table[n2] ← table[i]

Τέλος_αν

Τέλος_επανάληψης

Αποτελέσματα // pos_table, neg_table //

Τέλος Διαχωρισμός_Πινάκων

Συνένωση Πινάκων

Απλή Συνένωση

Αλγόριθμος Συνένωση_Πινάκων

! σελίδα 61 των οδηγιών μελέτης

Δεδομένα // a, b, A, B //

Για i από 1 μέχρι a

 Γ[i] ← A[i]

Τέλος_επανάληψης

k ← a

Για i από 1 μέχρι b

 Γ[i + k] ← B[i]

Τέλος_επανάληψης

Αποτελέσματα // Γ //

Τέλος Συνένωση_Πινάκων

Συγχώνευση Ταξινομημένων Πινάκων

Αλγόριθμος Συγχώνευση_Ταξινομημένων_Πινάκων

! σελίδα 62 των οδηγιών μελέτης

! σελίδα 85 του τετραδίου

Δεδομένα // a, b, A, B //

```

i ← 1
j ← 1
k ← 1
Όσο i ≤ a και j ≤ b επανάλαβε
  Αν A[i] < B[j] τότε
    Γ[k] ← A[i]
    i ← i + 1
  αλλιώς
    Γ[k] ← B[j]
    j ← j + 1
  Τέλος_αν
  k ← k + 1
Τέλος_επανάληψης

```

```

Αν i > a τότε
  Για h από j μέχρι b
    Γ[k] ← B[h]
    k ← k + 1
  Τέλος_επανάληψης
αλλιώς
  Για h από i μέχρι a
    Γ[k] ← A[h]
    k ← k + 1
  Τέλος_επανάληψης
Τέλος_αν

```

Αποτελέσματα // Γ //

Τέλος Συγχώνευση_Ταξινομημένων_Πινάκων

Δομές δεδομένων δευτερεύουσας μνήμης

Επειδή η RAM δεν επαρκεί για την αποθήκευση των δεδομένων καθώς και το ότι δεν αποθηκεύει μόνιμα, χρησιμοποιούμε τους μαγνητικούς δίσκους (ταινίες, σκληροί δίσκοι κ.ά.) και οπτικούς δίσκους (CD-ROM, DVD-ROM κ.ά.). Αυτά αποτελούν την **περιφερειακή** ή **δευτερεύουσα μνήμη**. Για την αποθήκευση δεδομένων σε αυτή κάνουμε χρήση ειδικών δομών που λέγονται **αρχεία (files)**. Κάθε αρχείο περιέχει μία συλλογή εγγραφών (π.χ. ένα αρχείο με εγγραφές μαθητών). Μία **εγγραφή (record)** αποτελεί τη συλλογή στοιχειωδών πληροφοριών σχετικά με ένα αντικείμενο (π.χ. μαθητής, ένα βιβλίο κλπ). Μία στοιχειώδης πληροφορία συνιστά ένα **πεδίο (field)**. Π.χ. μία εγγραφή μαθητή περιέχει στοιχειώδεις πληροφορίες (πεδία) όπως: Επώνυμο, Όνομα, Πατρώνυμο, Έτος γέννησης, Τάξη κλπ.

Πεδία					← Αρχείο
↓					
Επώνυμο	Ευθυμίου	Παπαδάκης	Αγγελάκη		← Αρχείο
Όνομα	Νίκος	Στέλιος	Δήμητρα		
Έτος	1988	1988	1987		
Τμήμα	Γ2	Γ1	Β2		
	↑	↑	↑		
Εγγραφές					

.