ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Α.Τ.Ε.Ι ΠΑΤΡΩΝ

ΣΧΟΛΗ: ΔΙΟΙΚΗΣΗΣ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ: ΔΙΟΙΚΗΣΗ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ:

« ΑΦΗΡΗΜΕΝΕΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ »

ΕΙΣΗΓΗΤΡΙΑ: ΑΝΤΩΝΟΠΟΥΛΟΥ ΗΡΑ

ΕΠΙΜΕΛΗΤΕΣ: ΜΩΡΟΥ ΑΙΚΑΤΕΡΙΝΗ
ΦΡΑΓΚΗ ΜΑΡΙΛΕΝΑ

ΠΑΤΡΑ 2004

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ ΤΡΙΤΟ
ΓΕΝΙΚΕΣ ΛΙΣΤΕΣ

ΚΕΦΑΛΑΙΟ ΤΕΤΑΡΤΟ
ΕΙΔΙΚΕΣ ΛΙΣΤΕΣ

ΚΕΦΑΛΑΙΟ ΠΕΜΠΤΟ
ΔΕΝΤΡΑ

1ο ΚΕΦΑΛΑΙΟ

Δομές δεδομένων και αλγόριθμοι

Εισαγωγικές παρατηρήσεις

Στο πρώτο μέρος αυτού του κεφαλαίου θα γνωρίσουμε τους Αφηρημένους Τύπους Δεδομένων καθώς επίσης και τους τρόπους που μπορούμε να περιγράψουμε έναν Αφηρημένο Τύπο Δεδομένων.

Στην συνέχεια θα αναφερθούμε στην έννοια του αλγορίθμου, στην περιγραφή της αλγοριθμικής γλώσσας και στην έννοια της αποδοτικότητας ενός αλγορίθμου. Ενώ συνοπτικά θα αναφερθούμε στην συνάρτηση πολυπλοκότητας.

1.1 Αφηρημένοι τύποι δεδομένων

1.1.1 Εισαγωγικά

Για κάθε τύπο δεδομένων (μπορεί να) υπάρχει ένας αντίστοιχος μαθηματικός ορισμός που προσδιορίζει με αυστηρό τρόπο τα χαρακτηριστικά του. Το μαθηματικό αυτό μοντέλο χρησιμοποιεί αφηρημένους όρους και έννοιες για την περιγραφή του αντίστοιχου τύπου δεδομένων και δεν αναφέρεται καθόλου σε θέματα υλοποίησης. Ο μαθηματικός αυτός ορισμός ονομάζεται *αφηρημένος τύπος δεδομένων* (abstract data type) ή συντομογραφικά ΑΤΔ (ATD).

Η υλοποίηση ενός ΑΤΔ σε συγκεκριμένο λογισμικό (γλώσσα προγραμματισμού) συνιστά ένα τύπο δεδομένων. Ένας ΑΤΔ όμως είναι δυνατόν να μη μπορεί να υλοποιηθεί σε όλη του τη γενικότητα ή και καθόλου σε κάποιο συγκεκριμένο λογισμικό ή υλικό Η/Υ. Π.χ. ο ΑΤΔ 'ακέραιος' δεν μπορεί να υλοποιηθεί σε όλη του τη γενικότητα λόγω των περιορισμών του υλικού και του λογισμικού των Η/Υ. Θυμηθείτε (Παράδειγμα1.2) ότι το πεδίο τιμών του στη γλώσσα Pascal είναι το [-maxint, maxint], όπου το maxint

εξαρτάται από τη συγκεκριμένη υλοποίηση της γλώσσας, δηλαδή τον μεταφραστή και τον τύπο Η/Υ που απευθύνεται, και όχι το $(-\infty, \infty)$ όπως απαιτεί ο αντίστοιχος ΑΤΔ.

Οι ΑΤΔ αποτελούν αυστηρές περιγραφές των ιδιοτήτων των τύπων δεδομένων, χωρίς να εμπλέκουν θέματα υλοποίησης ή αποδοτικότητας.. Είναι δυνατόν να υπάρχουν ΑΤΔ που να μην μπορεί να υλοποιηθούν σε κάποιο λογισμικό ή υλικό, αλλά να είναι χρήσιμοι για την κατανόηση κάποιων εννοιών. Η περιγραφή των ΑΤΔ γίνεται με κάποια «γλώσσα» ή «μέθοδο» και συνήθως συγκροτείται από δύο μέρη, το πρώτο αναφέρεται στον ορισμό του πεδίου τιμών και το δεύτερο τους ορισμούς των πράξεων. Έτσι οι ΑΤΔ αποτελούν χρήσιμο οδηγό και για υλοποιητές γλωσσών προγραμματισμού, ώστε να γίνεται έλεγχος σωστής υλοποίησης, και για προγραμματιστές, ώστε να γίνεται σωστή χρήση και εκμετάλλευση των δυνατοτήτων των τύπων δεδομένων.

1.1.2 Κατηγορίες δομών

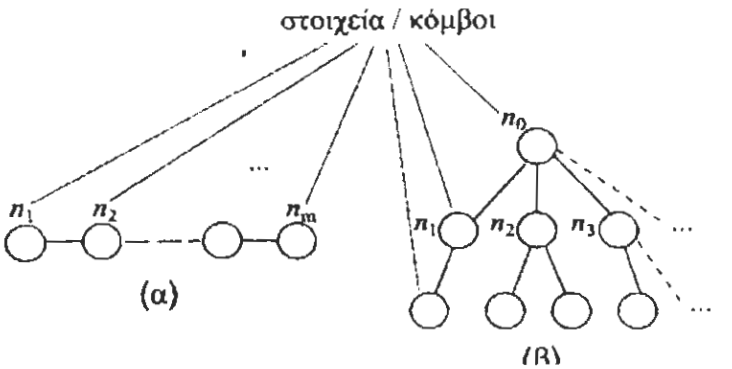
Οι δομές δεδομένων με τις οποίες θα ασχοληθούμε στα επόμενα κεφάλαια είναι οι εξής: *πίνακες*, *εγγραφές*, *γενικές λίστες (συνεχόμενες, συνδεδεμένες)*, *ειδικές λίστες (στοίβες, ουρές)* και *δέντρα*. Προς το παρόν μας είναι αρκετό να τις γνωρίσουμε μόνο ονομαστικά και να αναφερθούμε σε ορισμένα γενικά χαρακτηριστικά τους με βάση τα οποία τις κατατάσσουμε σε διάφορες κατηγορίες. Δεν θα ασχοληθούμε με *γραφήματα (ή γράφους)* και αρχεία, δύο άλλες δομές δεδομένων, διότι αποτελούν αντικείμενα άλλων θεματικών ενοτήτων.

Οι *πίνακες* και οι *εγγραφές* είναι οι πιο απλές δομές δεδομένων και υπάρχουν συνήθως ενσωματωμένες στις γλώσσες υψηλού επιπέδου. Οι δομές αυτές ονομάζονται *θεμελιώδεις δομές* (fundamental structures). Οι θεμελιώδεις δομές δεδομένων συνήθως προσδιορίζονται στατικά μέσα σένα πρόγραμμα, δηλαδή το μέγεθος τους και το οργανωτικό σχήμα τους παραμένουν αμετάβλητα στη διάρκεια εκτέλεσης ενός προγράμματος, γι' αυτό ονομάζονται και *στατικές δομές* (static structures).

Όμως, για τη λύση πολλών προβλημάτων απαιτούνται πιο πολύπλοκες δομές, που θεωρούνται *ανώτερες δομές* (higher-level structures). Οι ανώτερες δομές δεδομένων χρησιμοποιούν τις θεμελιώδεις δομές για την υλοποίηση τους και συνήθως

προσδιορίζονται δυναμικά σένα πρόγραμμα, δηλαδή το μέγεθος τους μεταβάλλεται κατά την εκτέλεση του προγράμματος, γι' αυτό αποκαλούνται και *δυναμικές δομές* (dynamic structures). τέτοιες δομές δεδομένων είναι οι διάφορες λίστες (συνεχόμενες, συνδεδεμένες, στοίβες και ουρές) και δέντρα.

Μια άλλη διάκριση που αφορά τις δομές δεδομένων είναι σε *γραμμικές* (linear) και *μη γραμμικές* (non-linear). Μια δομή είναι γραμμική αν το οργανωτικό της σχήμα μπορεί να απεικονισθεί σαν μία γραμμή. Αυτό είναι δυνατόν όταν κάθε στοιχείο σχετίζεται μόνο με ένα άλλο στοιχείο της, υπάρχει δηλαδή μια σχέση ένα- προς-ένα μεταξύ των στοιχείων της. Παρατηρήστε το Σχήμα.1.α όπου παρουσιάζεται η γενική μορφή μιας γραμμικής δομής, όπου κάθε στοιχείο / κόμβος της δομής σχετίζεται μόνο με ένα άλλο στοιχείο της δομής π.χ. με τη σχέση 'επόμενο'. Αντίθετα, σε μια μη γραμμική δομή υπάρχουν σχέσεις ένα – προς-πολλά ή πολλά –προς-πολλά μεταξύ των στοιχείων της. Στο Σχήμα 1.β παρουσιάζεται η γενική μορφή μιας κατηγορίας μη γραμμικών δομών, των δέντρων. Παρατηρήστε ότι κάθε κόμβος δυνατόν να σχετίζεται με περισσότερους του ενός άλλους κόμβους (σχέση ένα –προς-πολλά) π.χ. με τη σχέση 'παιδιά'. Γραμμικές δομές είναι οι πίνακες και οι λίστες, ενώ μη γραμμικές οι εγγραφές και τα δέντρα.



Σχήμα 1

Οργανωτικό σχήμα (α) γραμμικής (β) μη γραμμικής (δεντρικής) δομής

1.1.3 Υλοποίηση δομών δεδομένων

Οι ανώτερες δομές δεν υπάρχουν ενσωματωμένες στις γλώσσες προγραμματισμού, αλλά υλοποιούνται χρησιμοποιώντας τις θεμελιώδεις δομές τους. Η υλοποίηση μιας δομής έχει δύο πτυχές. Η μια αφορά στην *αναπαράσταση της δομής* στον Η/Υ, ενώ η άλλη στην *υλοποίηση των πράξεων* στην δομή. Αναπαράσταση μιας δομής σημαίνει την απεικόνιση της δομής στη μνήμη του Η/Υ, δηλαδή τον τρόπο αποθήκευσης των στοιχείων /κόμβων της δομής στη μνήμη του Η/Υ ώστε να ανταποκρίνεται στο οργανωτικό σχήμα της. Συνήθως υπάρχουν περισσότεροι του ενός τρόποι αναπαράστασης μιας δομής δεδομένων.

Όταν μιλάμε για μνήμη Η/Υ, αναφερόμαστε στην *κύρια μνήμη* (main memory) του. Η κύρια μνήμη ενός Η/Υ θεωρείται σαν ένα σύνολο από θέσεις μνήμης (memory cells). Κάθε θέση μνήμης προσδιορίζεται από μία διεύθυνση (address), που παριστάνεται με ένα θετικό ακέραιο αριθμό. Επίσης οι θέσεις μνήμης χαρακτηρίζονται από το μέγεθος τους, δηλαδή το πλήθος των δυαδικών ψηφίων πληροφορίας που μπορεί να αποθηκεύσει η κάθε μια. Το μέγεθος μιας θέσης μνήμης ονομάζεται λέξη (word) του Η/Υ. Στα επόμενα, για λόγους απλότητας, θεωρούμε Η/Υ με μέγεθος θέσης μνήμης (λέξη) 1 byte, δηλαδή πληροφορίας 8 δυαδικών ψηφίων (bits) (1 byte=8 bits). Μια μεταβλητή αντιπροσωπεύει ένα σύνολο θέσεων στη μνήμη, που το πλήθος του εξαρτάται από τον τύπο της μεταβλητής.

Η κύρια μνήμη ενός Η/Υ είναι *μνήμη τυχαίας προσπέλασης* (RAM: *Random Access Memory*). Ο όρος 'τυχαίας προσπέλασης' αναφέρεται στο ότι ο χρόνος που απαιτείται για πρόσβαση σε μια τυχαία επιλεγείσα θέση μνήμης είναι ο ίδιος για όλες τις θέσεις, υπόθεση που δεν ισχύει για δευτερεύουσες μνήμες (π.χ. μαγνητικό δίσκο, μαγνητική ταινία). Στο κεφάλαιο αυτό θα ασχοληθούμε με δομές δεδομένων που έχουν σχέση με την κύρια μνήμη του Η/Υ και όχι με κάποια δευτερεύουσα (όπως π.χ. αρχεία).

Υλοποίηση μιας πράξης σε μια δομή δεδομένων σημαίνει την εύρεση του τρόπου πραγματοποίησης της στον Η/Υ με βάση την αναπαράσταση της δομής και τις διαθέσιμες πράξεις των θεμελιωδών δομών και των ατομικών τύπων δεδομένων. Από τις

πράξεις στις δομές δεδομένων αυτές που κυρίως μας ενδιαφέρουν αναφέρονται παρακάτω.

Κυριότερες Πράξεις στις Δομές Δεδομένων

(α) διαπέραση (traversal):προσπέλαση και επεξεργασία κάθε στοιχείο ή κόμβου.
(β) αναζήτηση (search):εύρεση στοιχείου ή κόμβου με κάποια δεδομένη τιμή.
(γ) εισαγωγή (insertion):πρόσθεση ενός νέου στοιχείου ή κόμβου.
(δ) διαγραφή (deletion):αφαίρεση ενός υπάρχοντος στοιχείου ή κόμβου.
(ε) διάταξη (sorting):τακτοποίηση των στοιχείων ή κόμβων σε κάποια σειρά.

Από τις πράξεις αυτές οι δύο πρώτες δεν επιφέρουν μεταβολή στις δομές, σε αντίθεση με τις υπόλοιπες. Τις τέσσερις πρώτες πράξεις, που θεωρούνται πιο βασικές, τις εξετάζουμε μαζί με τις αντίστοιχες δομές.

1.2 Αλγοριθμική γλώσσα

1.2.1 Εισαγωγικές έννοιες

Αλγόριθμος είναι μια υπολογιστική διαδικασία αποτελούμενη από μια πεπερασμένη ακολουθία αυστηρά καθορισμένων υπολογιστικών βημάτων με σκοπό τη λύση ενός (γενικού) προβλήματος. Κάθε αλγόριθμος δέχεται κάποια δεδομένα σαν είσοδο και παράγει κάποια αποτελέσματα (λύση) σαν έξοδο.

Η περιγραφή ενός αλγορίθμου γίνεται με κάποια αλγοριθμική γλώσσα. Σαν τέτοια μπορεί να χρησιμοποιηθεί είτε η φυσική μας γλώσσα ή μία γλώσσα προγραμματισμού ή και μια διαγραμματική γλώσσα (όπως π.χ. το γνωστό διάγραμμα ροής) Το προαπαιτούμενο μιας αλγοριθμικής γλώσσας είναι η καθαρή σημασιολογία της σημειογραφίας και των συμβάσεων που χρησιμοποιεί, ώστε να μπορούν να περιγραφούν τα βήματα του αλγορίθμου με αυστηρό τρόπο, δηλαδή με τρόπο που επιδέχεται μόνο μια ερμηνεία.

Για την περιγραφή των αλγορίθμων χρησιμοποιούμε μια *γλώσσα ψευδοκώδικα*. Μια αυστηρά καθορισμένη γλώσσα που μας αποδεσμεύει από τις λεπτομέρειες μιας

γλώσσας προγραμματισμού. Αυτό που μας ενδιαφέρει σένα ψευδοκώδικα είναι η όσο το δυνατόν πιο κατανοητή και ταυτόχρονα συνοπτική περιγραφή βασικών βημάτων ενός αλγόριθμου.

Η γλώσσα ψευδοκώδικα που θα χρησιμοποιήσουμε διαθέτει στοιχεία που βασίζονται στις αρχές του δομημένου προγραμματισμού και είναι παρόμοια με στοιχεία των γλωσσών προγραμματισμού Pascal (κυρίως) και C. Η μεταφορά ενός αλγορίθμου από ψευδοκώδικα σε πραγματικό κώδικα είναι σχετικά εύκολη. Η έκφραση ενός αλγορίθμου σε μια γλώσσα προγραμματισμού συνιστά ένα πρόγραμμα.

1.2.2 Περιγραφή της γλώσσας

Η περιγραφή της αλγοριθμικής γλώσσας βασίζεται στο παρακάτω παράδειγμα.

Σχεδίαση αλγορίθμου από την περιγραφή του σε φυσική γλώσσα

Τίθεται το *πρόβλημα*: <<Να γράφει αλγόριθμος που να βρίσκει τη θέση και την τιμή του μικρότερου στοιχείου ενός πίνακα που περιέχει *n* αριθμούς.>>

Προτείνεται η εξής διαδικασία *λύσης* σε φυσική γλώσσα: <<Θεωρούμε το πρώτο στοιχείο του πίνακα σαν το μικρότερο και κρατούμε στη μνήμη την τιμή του και τη θέση του. Στη συνέχεια συγκρίνουμε την τιμή που κρατήσαμε με την τιμή του δεύτερου στοιχείου του πίνακα. Αν η τιμή του δεύτερου στοιχείου είναι μικρότερη, τότε κρατούμε στη μνήμη αυτή και τη θέση του, αλλιώς προχωρούμε και κάνουμε το ίδιο με το τρίτο στοιχείο κ.ο.κ ως το τελευταίο. Η τελευταία τιμή και θέση που κρατήσαμε στη μνήμη είναι τα ζητούμενα.>>

Ζητείται ο αντίστοιχος *αλγόριθμος*, γραμμένος στην αλγοριθμική μας γλώσσα. Κάθε αλγόριθμος παρουσιάζεται μέσα σε ένα πλαίσιο όπως και στον Αλγόριθμο 1.1. Μέσα στο πλαίσιο υπάρχουν σύντομες περιγραφές της εισόδου και εξόδου του αλγόριθμου, όπου δίδονται πληροφορίες για τα δεδομένα εισόδου και εξόδου, καθώς και από ποιες μεταβλητές αντιπροσωπεύονται. Ακολουθεί κυρίως περιγραφή του αλγόριθμου σαν μια υπολογιστική διαδικασία.

Ο Αλγόριθμος αυτός παρουσιάζεται στο επόμενο πλαίσιο.

1.1 **ΑΛΓΟΡΙΘΜΟΣ:** ΕΥΡΕΣΗ ΜΙΚΡΟΤΕΡΟΥ ΣΤΟΙΧΕΙΟΥ ΠΙΝΑΚΑ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (Α) και το πλήθος (Ν)των στοιχείων του, που είναι αριθμοί.

Έξοδος: Η θέση (Χ) και η τιμή (MIN) του μικρότερου στοιχείου του πίνακα.

```
MINSTOIXEIO (A, N)
1  K←1, X← 1, MIN←A[1]          {Αρχικοποίηση μεταβλητών.}
2  while K ≤ N                  {Έλεγχος τέλους επανάληψης }
3     if MIN> A[K]              {Σύγκριση με τη μικρότερη τιμή}
4        then X← K, MIN ← A[K]  {Ενημέρωση των Χ και MIN}
       endif
5     K← K + 1                  {Ενημέρωση μετρητή}
   endwhile
6  print X, MIN                 {Επιστροφή αποτελεσμάτων}
```

Η υπολογιστική διαδικασία έχει μια *κεφαλίδα* που περιλαμβάνει το *όνομα* (MINSTOIXEIO) του αλγορίθμου ακολουθούμενο από τις *παραμέτρους εισόδου* (A, N), που λειτουργούν σαν μεταβλητές, σε παρένθεση. Το *σώμα* της διαδικασίας περιγράφεται σαν μια ακολουθία (αριθμημένων) υπολογιστικών βημάτων (1-6) κωδικοποιημένων στην αλγοριθμική μας γλώσσα, που περιγράφουμε αμέσως παρακάτω. Λέξεις ή σύμβολα που παραμένουν αμετάβλητα εμφανίζονται με έντονη γραφή.

Βήματα και εκτέλεση

• Κάθε βήμα αποτελεί ένα *στοιχειώδες εκτελέσιμο τμήμα* αλγορίθμου και περιλαμβάνει, είτε μια έκφραση καταχώρησης (βήμα 5) είτε μια συνθήκη μιας διάταξης ελέγχου (βήματα 2,3) είτε μια έκφραση εξόδου (βήμα6).

- Σε ένα βήμα επιτρέπεται να υπάρχουν περισσότερες της μιας εκφράσεις καταχώρησης χωριζόμενες μεταξύ τους με κόμμα, οπότε θεωρούμε ότι εκτελούνται διαδοχικά από αριστερά προς τα δεξιά (π.χ. βήματα 1,4). Τα βήματα αυτά θεωρούνται ως *σύνθετα βήματα*, και χρησιμοποιούνται απλώς και μόνο σαν μέσα συντόμευσης της περιγραφής ενός αλγορίθμου.
- Τα βήματα εκτελούνται διαδοχικά, σύμφωνα με τη σειρά αναγραφής. Ο έλεγχος ροής της εκτέλεσης μπορεί να μεταφέρεται από ένα βήμα σε κάποιο άλλο που δεν είναι το επόμενο μόνο σαν αποτέλεσμα της ενέργειας κάποιας διάταξης ελέγχου ροής. (Οι διατάξεις ελέγχου ροής περιγράφονται στη συνέχεια της υποενότητας αυτής κάτω από τον ομώνυμο τίτλο).
- Σε κάθε βήμα μπορεί να υπάρχουν *σχόλια* μέσα σε άγκιστρα (π.χ. {Αρχικοποίηση μεταβλητών} στο βήμα 1 του αλγορίθμου).

Είσοδος και Έξοδος

- Τα δεδομένα εισόδου εισάγονται με τις παραμέτρους εισόδου της κεφαλίδας. Δεν απαιτείται είσοδος δεδομένων στο σώμα του αλγορίθμου.
- Η έξοδος των αποτελεσμάτων γίνεται με μια έκφραση εξόδου, που αποτελείται από την λέξη **print** ακολουθούμενη από καμία, μια ή περισσότερες μεταβλητές ή σταθερές, που αντιπροσωπεύουν τα (τελικά) αποτελέσματα (βήμα 6).

Μεταβλητές

- Για ονόματα μεταβλητών χρησιμοποιούνται συνδυασμοί κεφαλαίων γραμμάτων (MIN,K, X).
- Κάθε μεταβλητή έχει κάποιο ατομικό τύπο δεδομένων που δεν αναφέρεται ρητά στο σώμα του αλγορίθμου, αλλά υπονοείται από τα στοιχεία του προβλήματος.
- Η προσπέλαση σ'ένα στοιχείο μιας μεταβλητής τύπου πίνακα γίνεται με τον αντίστοιχο δείκτη, που μπορεί να είναι ακέραιος αριθμός ή μεταβλητή, σε τετραγωνικές παρενθέσεις, π.χ. A[1] (βήμα 1), A[K] (βήματα 3, 4).

- Όλες οι μεταβλητές είναι τοπικές, δηλαδή έχουν νόημα μόνο μέσα στην υπολογιστική διαδικασία του αλγορίθμου. Οι παράμετροι εισόδου μπορεί να παίζουν το ρόλο καθολικών μεταβλητών, δηλαδή να έχουν νόημα και εκτός του αλγορίθμου.

Σταθερές

- Σταθερές είναι οι αριθμοί (οποιουδήποτε τύπου), οι λογικές σταθερές (TRUE, FALSE) και χαρακτήρες ή λέξεις σε απλά εισαγωγικά (π.χ.'ΛΑΘΟΣ')

Τελεστές και Παραστάσεις

- Για πράξεις μεταξύ υπολογιστικών οντοτήτων (μεταβλητών, σταθερών) χρησιμοποιούνται οι γνωστοί από τα μαθηματικά τελεστές: *αριθμητικοί* (+,-,*,/, ύψωση σε δύναμη κλπ), *σύγκρισης* ($<, \leq, =, \neq, \geq, >$) και *λογικοί* (**and, or, not**).
- Με τη χρήση των τελεστών αυτών δημιουργούνται αντίστοιχα *αριθμητικές παραστάσεις* (βήμα 5 δεξί μέλος), *παραστάσεις σύγκρισης* (βήματα 2,3) και *λογικές παραστάσεις* (π.χ.(A>1) **and** (B ≠ 0)) κατά τους γνωστούς από τα βασικά μαθηματικά τρόπους. Το αποτέλεσμα μιας αριθμητικής παράστασης είναι ένας αριθμός, ενώ των άλλων δύο τύπων παραστάσεων μια λογική σταθερά.

Έκφραση Καταχώρησης

- Χρησιμοποιούμε ένα βέλος προς τα αριστερά (←) για να υποδηλώσουμε την καταχώρηση (ή απόδοση) τιμής σε μία μεταβλητή:

<μεταβλητή>← <τιμή>

όπου η <τιμή> μπορεί να είναι είτε μια σταθερά είτε μια μεταβλητή είτε μια οποιαδήποτε παράσταση (π.χ. στο βήμα 5 είναι μια αριθμητική παράσταση) ή μια συνάρτηση. (Η έννοια της συνάρτησης εξηγείται στο τελευταίο τμήμα αυτής της υποενότητας κάτω από τον τίτλο <<Μερικοί Αλγόριθμοι>>)

Διατάξεις Ελέγχου Ροής

- Οι διατάξεις *ελέγχου ροής* είναι υπολογιστικά σχήματα που επιτρέπουν την διαφοροποίηση από την ακολουθιακή εκτέλεση ενός αλγορίθμου. Χρησιμοποιούμε δύο

τέτοιες διατάξεις: *διάταξη ελέγχου με συνθήκη* (ή *επιλογής*) και *διάταξη ελέγχου επανάληψης*.

- Στο σώμα του αλγορίθμου, για ευκολότερη διάκριση των διατάξεων ελέγχου, χρησιμοποιούνται και *εσοχές* (βήμα 4, βήματα 3-5) και *ενδείξεις τέλους* (endif, endwhile). Γραμμές που περιέχουν ενδείξεις τέλους δεν αριθμούνται, διότι είναι μη εκτελέσιμα βήματα.
- *Διάταξη ελέγχου με Συνθήκη*

  Η διάταξη ελέγχου με συνθήκη ή διάταξη *if* επιτρέπει την επιλογή εκτέλεσης κάποιου τμήματος αλγορίθμου με βάση την αλήθεια κάποιας ή κάποιων συνθηκών και έχει την ακόλουθη μορφή:

If <συνθήκη>
then <τμήμα Α>
[else <τμήμα Β>]
endif

Η <συνθήκη> μπορεί να είναι είτε μια παράσταση σύγκρισης (όπως π.χ. στο βήμα 3) είτε μια λογική παράσταση. Το καθένα από τα <τμήμα Α> και <τμήμα Β> μπορεί να περιλαμβάνει ένα ή περισσότερα αλγοριθμικά βήματα. Οι τετραγωνικές παρενθέσεις στην περίπτωση αυτή δηλώνουν το προαιρετικό της γραμμής **else**. Το **endif** είναι απλώς η ένδειξη τέλους της διάταξης ελέγχου if και δεν αποτελεί εκτελέσιμο βήμα.

Η λογική της διάταξης αυτής είναι η εξής: αν η συνθήκη είναι αληθής τότε εκτελείται το <τμήμα Α>, αλλιώς εκτελείται το <τμήμα Β>, αν υπάρχει. Κατόπιν εκτελείται το επόμενο της διάταξης βήμα. Στο παράδειγμα, αν η παράσταση <<MIN> A[K]>> (βήμα 3) είναι αληθής, τότε εκτελείται το βήμα 4 (καταχώρηση των K, A[K] στα X, MIN αντίστοιχα) και στη συνέχεια η εκτέλεση προχωρά στο επόμενο της διάταξης βήμα 5 (αύξηση του μετρητή K κατά ένα), αλλιώς εκτελείται κατευθείαν το βήμα 5.

- Το <τμήμα Β> μπορεί να είναι μια άλλη διάταξη if, επίσης το αντίστοιχο τμήμα αυτής μπορεί να είναι μια άλλη διάταξη if κ.ο.κ., οπότε δημιουργείται μια *σύνθετη διάταξη if*, δηλαδή μια διάταξη που περιέχει περισσότερες της μιας συνθήκες και περισσότερα των

δύο τμήματα, οπότε για να εκτελεστεί ένα τμήμα πρέπει να αληθεύει ένας ορισμένος συνδυασμός συνθηκών.

- *Διάταξη Ελέγχου Επανάληψης*

Η διάταξη ελέγχου επανάληψης ή διάταξη *while* επιτρέπει την επανάληψη της εκτέλεσης ενός τμήματος αλγορίθμου εφόσον ισχύει μια συνθήκη και έχει μορφή:

While <συνθήκη>
<τμήμα Χ>
endwhile

Η λογική εδώ είναι η εξής: Εξετάζεται κάθε φορά η «συνθήκη» και εκτελείται το «τμήμα Χ», που λέγεται *σώμα* της διάταξης, εφόσον η συνθήκη είναι αληθής. Η εκτέλεση σταματά όταν η συνθήκη γίνει ψευδής, τότε ο έλεγχος μεταφέρεται στο επόμενο βήμα της διάταξης, το «τμήμα Χ» μπορεί να είναι μια άλλη διάταξη ελέγχου, είτε με συνθήκη ή επανάληψης. Το **endwhile** αποτελεί την ένδειξη τέλους και δεν είναι εκτελέσιμο. Π.χ στον αλγόριθμο «συνθήκη» είναι μια παράσταση σύγκρισης (βήμα 2) και το σώμα περιλαμβάνει μία διάταξη if (βήματα 3-4) και μια έκφραση καταχώρησης (βήμα 5). Παρατηρήστε ότι η τιμή του Κ, δηλαδή της μεταβλητής που επηρεάζει το αποτέλεσμα της συνθήκης, μεταβάλλεται σε κάθε εκτέλεση (βήμα 5), ώστε κάποια φορά θα ξεπεράσει την τιμή του Ν και θα πάψει να εκτελείται το σώμα της διάταξης. Αν δεν προβλέψουμε να συμβεί αυτό τότε θα έχουμε εκτέλεση του σώματος της διάταξης άπειρες φορές.

Οι περισσότερες γλώσσες προγραμματισμού διαθέτουν και μια άλλη διάταξη ελέγχου επανάληψης, τη διάταξη for, που χρησιμοποιείται όταν είναι γνωστός (ή μπορεί να υπολογιστεί) ο αριθμός των επαναλήψεων. Η διάταξη αυτή ουσιαστικά είναι πλεονασμός, διότι η λειτουργία της μπορεί
να περιγραφεί από τη διάταξη while. Όμως, δημιουργεί συνοπτικότερη και καλύτερα αναγνώσιμη περιγραφή των αλγορίθμων και οδηγεί σε αποδοτικότερη εκτέλεση. Γι' αυτό ορίζουμε μια αντίστοιχη διάταξη και στην αλγοριθμική μας γλώσσας, που έχει την μορφή

for < μετρητής > ← < αρχική τιμή > **to** < τελική τιμή >
< τμήμα χ >
endfor

Το < μετρητής > είναι μια μεταβλητή. Τα <αρχική τιμή>, <τελική τιμή> είναι ακέραιοι αριθμοί ή ακέραιες μεταβλητές ή αριθμητικές παραστάσεις ακεραίων. Το **endfor** είναι η ένδειξη τέλους και δεν είναι εκτελέσιμο.

Η λογική της διάταξη είναι η εξής: Ο μετρητής παίρνει σαν τιμή την <αρχική τιμή> και την αυξάνει διαδοχικά κατά ένα. Πριν από κάθε αύξηση εκτελείται το <τμήμα Χ>, το σώμα της διάταξης. Αυτό επαναλαμβάνεται έως ότου η τιμή του μετρητή να γίνει μεγαλύτερη της <τελικής τιμής>, οπότε σταματά. Τότε ο έλεγχος μεταφέρεται στο επόμενο της διάταξης βήμα του αλγορίθμου.

Με χρήση της διάταξης for αντί για της while, ο Αλγόριθμος 1.1 «Εύρεση μικρότερου στοιχείου πίνακα» γίνεται:

ΑΛΓΟΡΙΘΜΟΣ1.2: ΕΥΡΕΣΗ ΜΙΚΡΟΤΕΡΟΥ ΣΤΟΙΧΕΙΟΥ ΠΙΝΑΚΑ

```
MINSTOIXEIO(A, N)
1.  X ← 1, MIN ← A[1]                    {Αρχικοποίηση μεταβλητών}
2.  for K ← 1 to N                       {Καθορισμός επαναλήψεων}
3.      if MIN > A[K]                    {Σύγκριση με την μικρότερη τιμή}
4.          then X ← K, MIN ← A[K]       {Ενημέρωση των X και MIN}
        endif
        endfor
5.  print X, MIN                         {Επιστροφή αποτελεσμάτων}
```

Παρατηρούμε ότι ο μετρητής Κ είναι πλέον μέρος της διάταξης if και ότι τα βήματα του αλγορίθμου ελλατώθηκαν κατά ένα. Εφεξής, θα χρησιμοποιούμε τη διάταξη for σαν διάταξη επανάληψης όταν γνωρίζουμε τον αριθμό των επαναλήψεων, και την διάταξη while στις υπόλοιπες περιπτώσεις.

Μερικοί Αλγόριθμοι

Ένας *μερικός αλγόριθμος* είναι ένας ανεξάρτητος αλγόριθμος που καλείται για εκτέλεση μέσα από ένα *κύριο αλγόριθμο* ή από άλλον μερικό αλγόριθμο. Η κλήση ενός μερικού αλγορίθμου γίνεται με την κεφαλίδα του. Διακρίνουμε δύο είδη (μερικών) αλγορίθμων, την συνάρτηση (function) και την διαδικασία (procedure). Η πρώτη επιστρέφει πάντα μια τιμή που μπορεί να αποδοθεί σε μια μεταβλητή, ενώ η δεύτερη εκτελεί κάποιες υπολογιστικές πράξεις (π.χ καταχωρήσεις, ενημερώσεις κ.λ. π.) και ενδεχομένως να επιστρέφει και κάποια ή κάποιες τιμές.

1.3 Πολυπλοκότητα αλγορίθμων

Η υλοποίηση μιας πράξης επεξεργασίας μπορεί να γίνει συνήθως με περισσότερους του ενός τρόπους, δηλαδή αλγορίθμους. Επομένως τίθεται θέμα επιλογής του «καλύτερου» αλγορίθμου. Όταν λέμε «καλύτερος» αλγόριθμος, συνήθως εννοούμε «πιο αποδοτικός». Η αποδοτικότητα είναι το βασικό μέτρο σύγκρισης της επίδοσης δύο αλγορίθμων.

1.3.1 Η έννοια της αποδοτικότητας

Η αποδοτικότητα ενός αλγορίθμου έχει δύο παραμέτρους, την αποδοτικότητα χώρου. Η αποδοτικότητα χρόνου σχετίζεται με τον χρόνο επεξεργασίας που απαιτείται κατά την εκτέλεση του αλγορίθμου. δύο παραμέτρους, αυτή που είναι δυσκολότερο να

εκτιμηθεί και συνήθως σημαντικότερη είναι η αποδοτικότητα χρόνου. Η αποδοτικότητα χώρου είναι απλούστερη στην εκτίμηση και λιγότερο σημαντική συνήθως. Γι' αυτό, εφεξής θα επικεντρώσουμε την προσοχή μας στην αποδοτικότητα χρόνου.

Το θέμα που τίθεται τώρα είναι πως θα μετράμε την αποδοτικότητα χρόνου, ποιο μέτρο θα χρησιμοποιούμε για την μέτρησή της. Ένας τρόπος θα ήταν να υλοποιήσουμε τον αλγόριθμο σε μία γλώσσα προγραμματισμού, να τον εκτελέσουμε και να μετρήσουμε την απόδοσή του μετρώντας τον χρόνο εκτέλεσής του. Όμως μια τέτοια μέτρηση επηρεάζουν α) η γλώσσα στην οποία θα υλοποιηθεί β) οι ικανότητες του προγραμματιστή και γ) οι δυνατότητες του συγκεκριμένου Η/Υ στον οποίο θα τρέξει. Εμείς όμως ενδιαφερόμαστε για κάποιο τρόπο μέτρησης ανεξάρτητο από αυτούς τους παράγοντες.

Εξαιτίας αυτού , για τον υπολογισμό της αποδοτικότητας ενός αλγορίθμου, στηριζόμαστε σ' αυτές που ονομάζουμε βασικές υπολογιστικές πράξεις (basic operations) του αλγορίθμου. Πιο συγκεκριμένα, η αποδοτικότητα εκτιμάται με τον αριθμό εκτελέσεων των βασικών υπολογιστικών πράξεων, χωρίς να μας ενδιαφέρει ο πραγματικός χρόνος εκτέλεσής τους. Βασικές υπολογιστικές πράξεις είναι αυτές που χαρακτηρίζουν την χρονική εκτέλεση ενός αλγορίθμου. Πολλές φορές επιλέγουμε μία σαν βασική υπολογιστική πράξη. Π.χ. στους αλγορίθμους αναζήτησης και διάταξης θεωρούμε συνήθως σαν βασική
υπολογιστική πράξη την σύγκριση δύο στοιχείων, και επομένως η αποδοτικότητα τους εκτιμάται από τον αριθμό των συγκρίσεων που ενεργούνται κατά την εκτέλεση. Αντίθετα, σε ένα αλγόριθμο διαγραφής θεωρούμε συνήθως σαν βασική πράξη την μετακίνηση ενός στοιχείου, οπότε η αποδοτικότητα εκτιμάται από τον αριθμό των μετακινήσεων που πραγματοποιούνται.

Είναι προφανές επίσης ότι η αποδοτικότητα εξαρτάται και από το μέγεθος των δεδομένων εισόδου. Π.χ. αν έχω να διαγράψω ένα στοιχείο από μία λίστα, ο αριθμός των μετακινήσεων που θα απαιτηθούν για να καλυφθεί το κενό, εξαρτάται εν γένει από το πλήθος των στοιχείων της λίστας

1.3.2 Συνάρτηση πολυπλοκότητας

Η εκτίμηση της αποδοτικότητας ενός αλγορίθμου αναφέρεται ως πολυπλοκότητα του αλγορίθμου και εκφράζεται με μια συνάρτηση πολυπλοκότητας f(n), όπου n το μέγεθος / πλήθος των δεδομένων εισόδου. Η f(n) εκφράζει την απαίτηση του αλγορίθμου σε χρόνο εκτέλεσης.

Σύνοψη

Ένας τύπος καθορίζει το είδος των τιμών που μπορεί να πάρει μια μεταβλητή, δηλαδή το **πεδίο τιμών της**, και τις **πράξεις** που μπορούν να γίνουν στις τιμές αυτές. Σε ένα **ατομικό τύπο δεδομένων** οι τιμές είναι απλές (π.χ αριθμοί) και οι πράξεις στοιχειώδεις (π.χ. αριθμητικές). Μια **δομή δεδομένων** είναι ένας τύπων δεδομένων που προσδιορίζει μεταβλητές που αντιπροσωπεύουν σύνθετες τιμές. Οι σύνθετες τιμές περιλαμβάνουν επί μέρους τιμές που ονομάζονται **στοιχεία** ή **κόμβοι** και μπορεί να είναι είτε απλές τιμές είτε άλλες σύνθετες τιμές. Μία δομή δεδομένων λοιπόν εκτός από το πεδίο τιμών και τις αντίστοιχες πράξεις προσδιορίζει κα το **οργανωτικό σχήμα** των (σύνθετων) τιμών, δηλαδή των τρόπο οργάνωσης ή συσχετισμού των στοιχείων της δομής. Αυτό το χαρακτηριστικό δεν υπάρχει σε ένα ατομικό τύπο δεδομένων διότι στερείται επιμέρους τιμών και επομένως οργάνωσης. Το πλήθος των στοιχείων (ή κόμβων) μιας δομής συνιστά το **μέγεθος** της δομής.

Με βάση την απλότητα της δομής και την μεταβλητότητα του μεγέθους της κατά τη διάρκεια εκτέλεσης του προγράμματος, οι δομές δεδομένων διακρίνονται σε **θεμελιώδεις-στατιστικές** (πίνακες, εγγραφές) και **ανώτερες-δυναμικές** (λίστες, δένδρα). Επιπλέον, με βάση το είδος του οργανωτικού σχήματος διακρίνονται σε **γραμμικές** (πίνακες, λίστες) και μη γραμμικές (εγγραφές, δένδρα).

Ο αλγόριθμος είναι μια πεπερασμένη ακολουθία υπολογιστικών βημάτων που δίνει κάποιο επιδιωκόμενο αποτέλεσμα. Κάθε αλγόριθμος περιγράφεται με μια **γλώσσα ψευδοκώδικα,** δηλαδή μια αυστηρά καθορισμένη γλώσσα που είναι όμως απλούστερη από μια γλώσσα προγραμματισμού υψηλού επιπέδου και μας αποδεσμεύει από μη αναγκαίες λεπτομέρειες.

Οι αλγόριθμοι χρησιμοποιούνται ως μέσα περιγραφής μιας δομής δεδομένων, η οποία κυρίως αναφέρεται στην υλοποίηση των πράξεων στη δομή. Είναι δυνατόν να υπάρχουν περισσότεροι του ενός αλγόριθμοι υλοποίησης μιας πράξης. Η σύγκριση τους στηρίζεται κυρίως στην εκτίμηση της αποδοτικότητας χρόνου που εκφράζεται με την **συνάρτηση πολυπλοκότητας**.

2ο ΚΕΦΑΛΑΙΟ

Πίνακες και εγγραφές

Εισαγωγικές Παρατηρήσεις

Στην πρώτη ενότητα αυτού του κεφαλαίου εξετάζουμε τους πίνακες, μονοδιάστατους και δισδιάστατους, και αναφερόμαστε στην αναπαράστασή τους στην μνήμη του Η/Υ. Επίσης γίνεται μικρή αναφορά για τους πολυδιάστατους πίνακες. Οι πίνακες είναι από τις πιο βασικές δομές δεδομένων.

Στην δεύτερη ενότητα αναφέρουμε το θέμα της αναζήτησης ενός στοιχείου σε ένα (μονοδιάστατο) πίνακα. Εξετάζουμε δύο βασικές μεθόδους, την γραμμική αναζήτηση (για μη διατεταγμένους πίνακες) και την δυαδική αναζήτηση (για διατεταγμένους πίνακες). Οι βασικοί αλγόριθμοι που παρουσιάζονται εδώ, μπορούν να χρησιμοποιηθούν και σε άλλες δομές, όπως οι λίστες με τις απαιτούμενες συντακτικές κυρίως αλλαγές. Η λογική παραμένει ίδια.

Τέλος, στην Τρίτη ενότητα παρουσιάζουμε βασικά στοιχεία της δομής 'εγγραφή'. Οι εγγραφές, συχνά αποτελούν στοιχεία άλλων δομών. Στην περίπτωση του πίνακα, η παραγόμενη δομή ονομάζεται «πίνακας εγγραφών», στην οποία αναφερόμαστε συνοπτικά.

2.1 ΠΙΝΑΚΕΣ

Στον προγραμματισμό η έννοια αυτή συνδέεται με αριθμημένες θέσεις μνήμης όπου καταχωρούμε πληροφορίες του ίδιου τύπου δεδομένων. Η αναφορά μας σε κάθε μία από τις θέσεις αυτές γίνεται με το κοινό όνομα που τις χαρακτηρίζει σαν σύνολο (το όνομα του πίνακα) συνοδευμένο από την ακριβή τους τοποθέτηση μέσα στο σύνολο αυτό.

Ο λογικός τρόπος οργάνωσης των διαφόρων πληροφοριών σε πίνακα καθορίζει και την μορφή του πίνακα. Έτσι μπορούμε να έχουμε μονοδιάστατους πίνακες όπου τα δεδομένα έχουν διαταχθεί μόνο ως προς μία κατεύθυνση (γραμμή) ή δυσδιάστατους πίνακες με τα δεδομένα τους διατεταγμένα ως προς περισσότερες από μία κατευθύνσεις (γραμμές-στήλες). Η μορφή βέβαια με την οποία καταχωρείται ένας πίνακας εσωτερικά στη μνήμη του Η/Υ είναι απόλυτα γραμμική.

2.1.1 Μονοδιάστατος πίνακας

Ο μονοδιάστατος πίνακας, είναι μία θεμελιώδης δομή δεδομένων που υπάρχει ενσωματωμένη σε όλες σχεδόν τις γλώσσες προγραμματισμού υψηλού επιπέδου. Πριν αναφερθούμε λοιπόν στις ανώτερες δομές, κρίνουμε σκόπιμο να αναφερθούμε στον πίνακα, δεδομένου ότι αποτελεί τη βάση υλοποίησης για πολλές από τις ανώτερες δομές.

Πίνακας είναι μια συλλογή ενός πεπερασμένου πλήθους στοιχείων του ίδιου τύπου, που ονομάζεται τύπος βάσης (base type). Λόγω της τελευταίας ιδιότητας ,λέμε ότι ένας πίνακας αποτελεί μια ομογενή δομή (homogeneous structure). Κάθε πίνακας σχετίζεται με ένα σύνολο δεικτών (index type) I, που αποτελείται από τόσους δείκτες όσα και τα στοιχεία του πίνακα. Οι δείκτες είναι και αυτοί ίδιου τύπου δεδομένων, που ονομάζεται τύπος δείκτη (index type).Κάθε δείκτης αντιστοιχεί σ'ένα και μόνο στοιχείο του πίνακα.

Παρόλο που οι δείκτες δεν είναι απαραίτητα ακέραιοι αριθμοί, εφεξής, για λόγους απλότητας και δεδομένου ότι δεν βλάπτεται η γενικότητα, θα θεωρούμε δείκτες μόνο ακέραιους και μάλιστα διαδοχικούς. Σε αυτήν την περίπτωση οι δείκτες είναι συνήθως διατεταγμένοι.

Το πεδίο τιμών ενός πίνακα μεγέθους Ν είναι ένα σύνολο που περιλαμβάνει πλειάδες μήκους Ν, που είναι όλοι οι δυνατοί συνδυασμοί τιμών των Ν στοιχείων του πίνακα. Οι βασικές πράξεις σε έναν πίνακα είναι η προσπέλαση (access) ή ανάκτηση (retrieval) ενός στοιχείου και η καταχώρηση

(assignment) ή ενημέρωση (update) ενός στοιχείου. Στην αλγοριθμική γλώσσα, αυτές οι πράξεις εκφράζονται ως εξής (όπου X είναι μια μεταβλητή):

X ← A[K] (προσπέλαση ή ανάκτηση του στοιχείου με δείκτη K)

A[K] ← X (κ α τ α χ ώρηση ή ενημέρωση του στοιχείου με δείκτη K).

Κάθε γλώσσα προγραμματισμού έχει το δικό της τρόπο για τη δήλωση (δημιουργία) ενός πίνακα. Πάντως, κάθε τέτοια δήλωση πρέπει να περιλαμβάνει τρία στοιχεία:

I) Το όνομα του πίνακα

II) Τον τύπο βάσης του πίνακα

III) Το σύνολο (ή τον τύπο) δεικτών του πίνακα

Το μέγεθος ενός πίνακα συνήθως παραμένει σταθερό κατά τη διάρκεια εκτέλεσης ενός προγράμματος, γι αυτό και ο πίνακας αποτελεί στατική δομή. Στις περισσότερες γλώσσες υψηλού επιπέδου , το μέγεθος ενός πίνακα καθορίζεται στατικά κατά την μετάφραση του προγράμματος. Υπάρχουν όμως και γλώσσες που επιτρέπουν τον καθορισμό του μεγέθους ενός πίνακα και δυναμικά, δηλαδή κατά την εκτέλεση του προγράμματος μέσω μιας μεταβλητής. Και στις δύο περιπτώσεις όμως το μέγεθος ενός πίνακα παραμένει σταθερό μετά τον καθορισμό του.

Ένα από τα βασικά χαρακτηριστικά ενός πίνακα είναι ότι είναι μια *δομή τυχαίας προσπέλασης*, δηλαδή ο χρόνος εντοπισμού (προσπέλασης) ενός στοιχείου του ενός ανεξάρτητος από τη θέση του στοιχείου στον πίνακα. Αυτό είναι μια ιδιότητα μοναδική για τους πίνακες που δεν υπάρχει σε άλλες δομές.

2.1.1 Αναπαράσταση μονοδιάστατου πίνακα

Για να είναι δυνατό να επιτευχθεί τυχαία προσπέλαση στα στοιχεία ενός πίνακα πρέπει να υπάρχει ένας τρόπος υπολογισμού της διεύθυνσης της θέσης μνήμης κάθε στοιχείου με βάση μόνο το δείκτη του στοιχείου, ανεξάρτητα από τα υπόλοιπα στοιχεία. Ας θεωρήσουμε ένα πίνακα A με $I = \{nl \ldots, nu\}$.

Θυμηθείτε, από την υποενότητα «1.1.5 Υλοποίηση Δομών Δεδομένων», ότι η μνήμη ενός Η/Υ θεωρείται σαν ένα σύνολο θέσεων μνήμης (bytes) με διαδοχικές διευθύνσεις. Τα στοιχεία ενός πίνακα αποθηκεύονται σε γειτονικές (διαδοχικές) θέσεις μνήμης. Κάθε στοιχείο ενός πίνακα καταλαμβάνει εν γένει L θέσεις μνήμης. Το L καλείται μήκος στοιχείου (element length) του πίνακα και εξαρτάται από τον τύπο των στοιχείων και τη γλώσσα υλοποίησης. Ας θεωρήσουμε τη διεύθυνση της πρώτης θέσης μνήμης

του πρώτου στοιχείου του πίνακα, πού καλείται διεύθυνση βάσης (base address) του πίνακα, και ας τη συμβολίσουμε με bA. Τότε, το πρώτο στοιχείο του πίνακα θα αποθηκευτεί σε L θέσεις με διευθύνσεις

bA, bA+1, bA+2,....., bA + l.-1.

Επομένως, η διεύθυνση της πρώτης θέσης του δεύτερου στοιχείου, πού θα είναι η αμέσως επόμενη από την τελευταία του πρώτου στοιχείου, θα είναι bA+L, αυτής του τρίτου στοιχείου θα είναι bA+2L, του τέταρτου bA+3L κ.ο.κ., όπου οι συντελεστές 1,2,3,..... του L φράζουν τον αριθμό των στοιχείων του πίνακα πού προηγούνται των στοιχείων με δείκτες nl+1, nl+2, nl+3.....αντίστοιχα. (δηλαδή 1 = ((nl+1) –nl), 2 = ((nl+2)-nl), 3=((nl+3)-nl),....). Άρα, κατ΄ αντιστοιχία, η διεύθυνση της πρώτης θέσης μνήμης του στοιχείου με δείκτη i (nl ≤ i ≤ nu) θα είναι bA +(i-nl) L, αφού υπάρχουν (i-nl) στοιχεία πριν το στοιχείο με δείκτη i, δηλαδή έχουν καταληφθεί (i- nl) L θέσεις από αυτά ξεκινώντας από τη θέση bA. Επομένως, η διεύθυνση loc (Ai) του πρώτου byte του στοιχείου με δείκτη i του πίνακα A είναι :

$$loc(Ai) = bA + (i - nl)\,L \qquad (2.2)$$

πού μπορεί να γραφεί και ως

$$loc(Ai) = c0 + c1\,i \qquad (2.3)$$

όπου

$$c0 = bA - nl\,L,\ c1 = L \qquad (2.4)$$

Από την σχέση (2.3) είναι φανερό ότι η διεύθυνση του στοιχείου με δείκτη i είναι μια γραμμική συνάρτηση του i. Η σχέση αυτή αποτελεί τη συνάρτηση απεικόνισης πίνακα (array mapping function) ή συντομογραφικά ΣΑΠ (AMF). Τα c0, c1 είναι οι σταθερές ΣΑΠ και υπολογίζονται εσωτερικά από τον μεταφραστή της γλώσσας υλοποίησης για κάθε πίνακα.

Παρατηρήστε ότι ο χρόνος υπολογισμού της loc (Ai)είναι ο ίδιος για κάθε i, δεδομένου ότι μετά τον υπολογισμό των c0 και c1 απαιτούνται ένας πολλαπλασιασμός (c1L)και μια πρόσθεση (c0+c1L) για τον υπολογισμό της διεύθυνσης οποιουδήποτε στοιχείου του πίνακα. Επομένως, αν δοθεί ο δείκτης i μπορούμε να προσπελάσουμε το περιεχόμενο του Ai χωρίς να περάσουμε από κανένα άλλο στοιχείο του A.

Παράδειγμα 2.1 Καθορισμός χαρακτηριστικών, εύρεση ΣΑΠ και διεύθυνσης στοιχείου μονοδιάστατου πίνακα.

Ας υποθέσουμε ότι μια Υπηρεσία του Υπουργείου Παιδείας για να καταχωρίσει τον αριθμό των μαθηματικών πού διορίστηκαν στη μέση εκπαίδευση για κάθε έτος από το 1956 ως το 1998 χρησιμοποιεί ένα πίνακα MATH. Στη περίπτωση αυτή, είναι χρήσιμο να θεωρήσουμε σαν σύνολο δεικτών του πίνακα

το I = {1956, 1957,....,1998}. Έτσι, το στοιχείο MATH1965 παριστάνει το πλήθος των μαθηματικών πού διορίστηκαν το έτος 1965. Το μέγεθος του πίνακα υπολογίζεται από τον τύπο 2.1 ως εξής:

N=nu – nl + 1 = 1998 – 1956 + 1 = 43 στοιχεία.

Αν μας δίνεται ότι το μήκος στοιχείου του πίνακα είναι L = 2 θέσεις μνήμης (bytes) και ότι η διεύθυνση βάσης είναι b MATH = 500, τότε αυτό σημαίνει ότι loc ($MATH_{1956}$) = 500, loc ($MATH_{1957}$) = 502, loc ($MATH_{1958}$) = 504 κ.ο.κ., δηλαδή η διεύθυνση του πρώτου byte μνήμης πού καταλαμβάνουν τα στοιχεία $MATH_{1956}$, $MATH_{1957}$, $MATH_{1958}$ κ.ο.κ. είναι 500,502,504 κ.ο.κ.. Η κατάσταση της περιοχής της μνήμης πού χρησιμοποιεί ο πίνακας απεικονίζεται στο σχήμα 1α.

Για να βρούμε τη ΣΑΠ υπολογίζουμε τις σταθερές της από τους τύπους (2.4),

$$c0 = 500 - 1956 \times 2 = -3412,\ c1 = 2$$

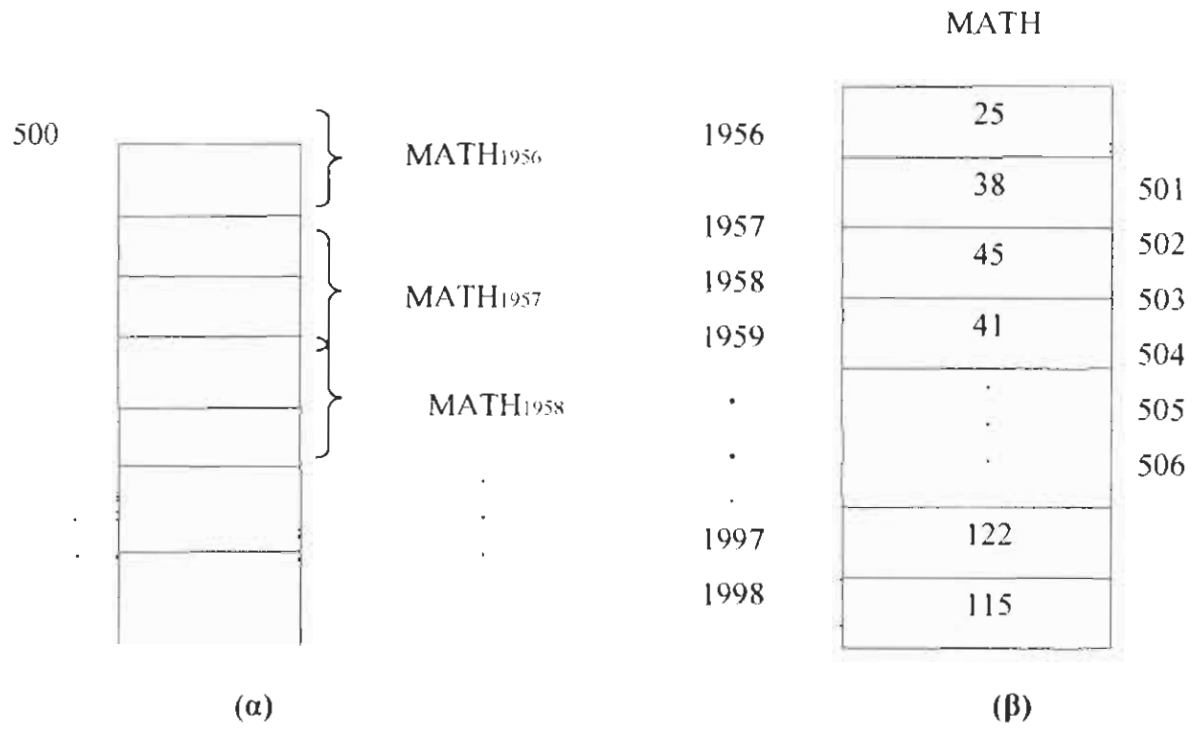
οπότε η ΣΑΠ (τύπος 2.3) είναι

$$loc(MATH_i) = -3412 + 2i.$$

Η διεύθυνση της πρώτης θέσης μνήμης του στοιχείου του πίνακα MATH πού αναφέρεται στο έτος π.χ. 1985 υπολογίζεται από τη ΣΑΠ για i = 1985:

$$Loc(MATH_{1985}) = -3412 + 2 \times 1985 = 558.$$

Στο σχήμα 1β παρουσιάζεται ένας τρόπος γραφικής απεικόνισης ενός μονοδιάστατου πίνακα, εδώ του MATH με κάποιες εικονικές τιμές στοιχείων.



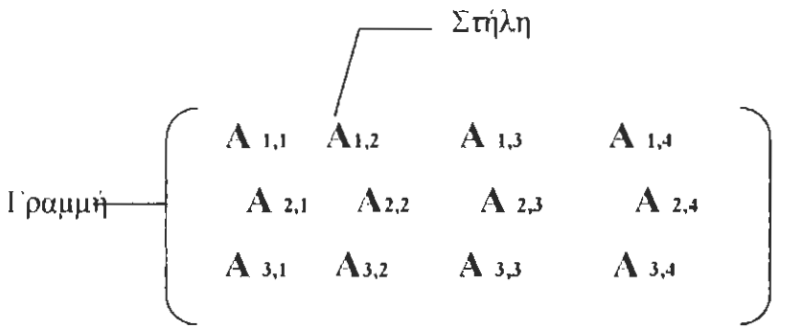
Σχήμα 1

(α) Αναπαράσταση του πίνακα MATH στη μνήμη (β) Απεικόνιση του πίνακα MATH

2.1.3 Δισδιάστατοι πίνακες.

Ένας δισδιάστατος πίνακας (two-dimensional array)είναι ένα σύνολο πεπερασμένου πλήθους στοιχείων του ίδιου τύπου, όπου κάθε στοιχείο προσδιορίζεται από ένα ζεύγος δεικτών. Δηλαδή, το σύνολο δεικτών I ενός δισδιάστατου πίνακα είναι ένα σύνολο διατεταγμένων ζευγών (θεωρούμε ακεραίων) δεικτών ή με άλλα λόγια το καρτεσιανό γινόμενο δυο συνόλων (θεωρούμε ακεραίων)δεικτών, $I = I_1 \times I_2$. Τα πλήθη των στοιχείων N_1 και N_2 των I_1 και I_2 αντίστοιχα καλούνται διαστάσεις (dimensions) του πίνακα, και λέμε ότι ο πίνακας είναι διαστάσεων $N_1 \times N_2$. Προφανώς, το μέγεθος του πίνακα είναι $N=N_1 \times N_2$

Ας θεωρήσουμε ένα δισδιάστατο πίνακα Α διαστάσεων $N_1 \times N_2$, με I_1 = {1,2,...., nu1}και I_2 ={1,2,...., nu2}, οπότε N_1 = nu1 και N_2 = nu2 τότε, θα αναφερόμαστε στα στοιχεία του πίνακα αυτού με τα σύμβολα $A_{1,1}$, $A_{1,2}$, ... $A_{2,1}$, $A_{2,2}$, ... Για καλύτερη εποπτεία και κατανόηση των δυο διαστάσεων, απεικονίζουμε ένα δισδιάστατο πίνακα διαστάσεων 3X4, όπως φαίνεται στο σχήμα 2.

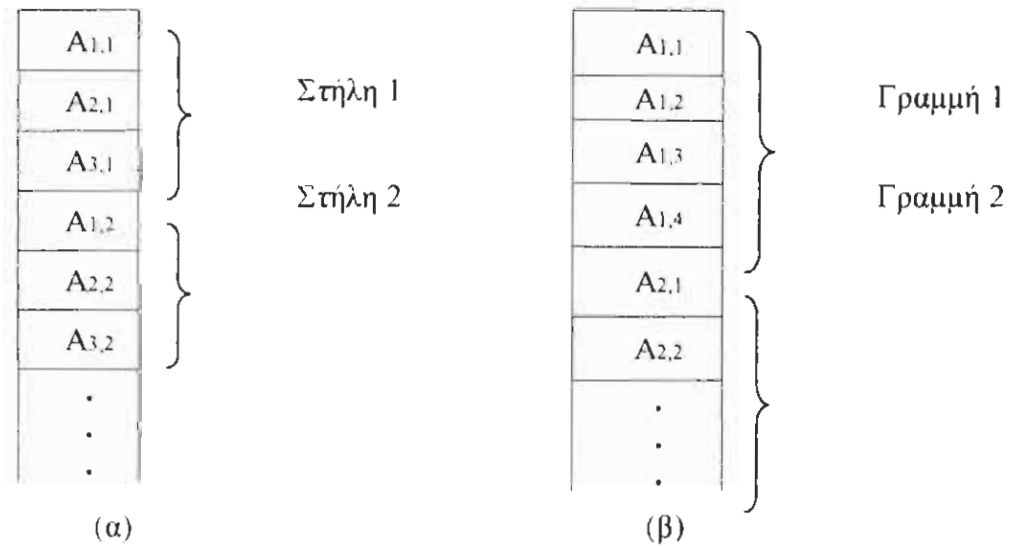


Σχήμα 2
Δισδιάστατος πίνακας διαστάσεων 3 X 4

Με βάση αυτή την απεικόνιση διακρίνουμε σ΄ ένα δισδιάστατο πίνακα γραμμές (rows), δηλαδή οριζόντιες συλλογές στοιχείων, και στήλες (colymns), δηλαδή κατακόρυφες συλλογές στοιχείων. Ο παραπάνω πίνακας Α έχει 3 γραμμές και 4 στήλες, πού αντιστοιχούν στις διαστάσεις του πίνακα. Κάθε ζεύγος δεικτών προσδιορίζει ένα και μόνο ένα στοιχείο του πίνακα. Γενικά το σύμβολο $A_{i,j}$, όπου $i \in I_1$ ($1 \le i \le N_1$) και $j \in I_2$ ($1 \le j \le N2$), παριστάνει το στοιχείο πού ορίζεται από την γραμμή i και την στήλη j ή με άλλες λέξεις, το στοιχείο πού βρίσκεται στη θέση j της γραμμής i ή με άλλες λέξεις, το στοιχείο πού βρίσκεται στη θέση i της στήλης j.

2.1.4 Αναπαράσταση δισδιάστατου πίνακα

Η αναπαράσταση ενός δισδιάστατου πίνακα διαστάσεων N X N, στη μνήμη του Η/Υ μπορεί να γίνει κατά δυο τρόπους: (α) με βάση τις στήλες του, οπότε ονομάζεται διάταξη κατά στήλες (column - major order), (β) με βάση τις γραμμές του, οπότε ονομάζεται διάταξη κατά γραμμές (row – major order). Οι δυο μέθοδοι απεικονίζονται στο σχήμα 3, όπου θεωρούμε ότι κάθε ορθογώνιο κελί αντιστοιχεί σε τόσες θέσεις μνήμης όσο και το μήκος στοιχείου του πίνακα.



Σχήμα 3

Ένας δισδιάστατος πίνακας, όπως και ένας μονοδιάστατος, αποθηκεύεται σε γειτονικές θέσεις μνήμης, ώστε ο Η/Υ δεν είναι υποχρεωμένος να κρατά τις διευθύνσεις όλων των στοιχείων του, αλλά μόνο του πρώτου στοιχείου του πίνακα, δηλαδή τη διεύθυνση της πρώτης θέσης μνήμης πού καταλαμβάνει το στοιχείο $A_{1,1}$ πού, όπως και στην περίπτωση του μονοδιάστατου πίνακα, ονομάζεται διεύθυνση βάσης και τη συμβολίζουμε με bA. Αν θεωρήσουμε ένα δισδιάστατο πίνακα A με I_1 = {ni1,.....ni1}και I_2 {ni2,....ni2}, οι συναρτήσεις απεικόνισης για τους δυο τρόπους αναπαράστασης του αποδεικνύεται, με βάση παρόμοιο συλλογισμό με αυτόν στην περίπτωση του μονοδιάστατου πίνακα, ότι είναι οι εξής:

Διάταξη κατά στήλες

$$loc(Aij)=bA + (N1(j - nl2) + (I - nl1))L \quad (2.5)$$

η οποία μπορεί να γραφεί

$$loc(Ai,j)=c0+c1i+c2j \quad (2.6)$$

όπου

$$c1 = L,\ c2 = N1L,\ c0 = bA - c1nl1 - c2nl2 \quad (2.7)$$

οι σταθερές ΣΑΠ, και ως γνωστόν N1 = nu1 – nl1 +1.

Διάταξη κατά γραμμές

$$loc(Aij) = bA + (N2(i - nl1) + (j - nl2))L \quad (2.8)$$

η οποία μπορεί να γραφεί

$$loc(Ai,j) = c0+c1i+c2j \quad (2.9)$$

όπου

$$c1 = N2L,\ c2 = L,\ c0 = bA - c1nl1 - c2nl2 \quad (2.10)$$

και ως γνωστόν N2 =nl2 – nl2 +1.

Ας σημειωθεί ότι, η διεύθυνση ενός στοιχείου είναι γραμμική συνάρτηση των δεικτών I,j, σχέσεις (2.6) και (2.9). Επίσης, ότι ο χρόνος υπολογισμού είναι ανεξάρτητος από το συγκεκριμένο ζεύγος δεικτών.

Ένας άλλος τρόπος υπολογισμού της θέσης ενός στοιχείου πίνακα στη μνήμη του Η/Υ, εκτός της συνάρτησης απεικόνισης, είναι ο πίνακας προσπέλασης (access table), πού είναι λιγότερο χρονοβόρος, αλλά απαιτεί περισσότερο χώρο μνήμης. Για περισσότερες λεπτομέρειες, αναφερθείτε στον οδηγό για περαιτέρω μελέτη, στο τέλος του κεφαλαίου.

Οι γλώσσες υψηλού επιπέδου δεν χρησιμοποιούν όλες τον ίδιο τρόπο αποθήκευσης των δυσδιάστατων πινάκων. Π.χ., η C και η πλειονότητα των γλωσσών υψηλού επιπέδου χρησιμοποιούν την αποθήκευση κατά γραμμές, ενώ η FORTRAN την κατά στήλες.

Καθορισμός χαρακτηριστικών, εύρεση ΣΑΠ και διεύθυνσης στοιχείου δισδιάστατου πίνακα.

Σ΄ ένα τμήμα 15 μαθητών δόθηκαν τέσσερα διαγωνίσματα, τα αποτελέσματα των οποίων (βαθμοί) παρουσιάζονται στον παρακάτω πίνακα.

Για την αποθήκευσή τους χρησιμοποιούμε ένα δισδιάστατο πίνακα ΒΑΤΗ διαστάσεων 15 Χ 4 (N1 =15, N2 = 4), με I1 = {1,2,....,15} και I2 = {I,....,4}. Κάθε ακέραιος στο I1 αντιπροσωπεύει ένα μαθητή, ενώ κάθε ακέραιος στο I2 ένα διαγώνισμα. Κάθε στοιχείο του πίνακα παριστάνει το βαθμό κάποιου μαθητή σε κάποιο διαγώνισμα. Π.χ., το BATH5,3 παριστάνει το βαθμό του μαθητή 5 στο διαγώνισμα 3. Συνολικά, ο πίνακας έχει 15 Χ 4 = 60 στοιχεία.

		Διαγωνίσματα				
		1	2	3	4	
	1	15	11	13	14	
Μαθητές	2	12	14	15	12	Βαθμοί
	3	17	16	14	13	
	.	.	.	.	.	
	.	.	.	.	.	
	.	.	.	.	.	
	15	14	18	15	13	

Αν μας δίνεται ότι η διεύθυνση βάσης του πίνακα είναι b ΒΑΤΗ = 300, το μήκος στοιχείου του πίνακα L=3 και ότι η αποθήκευση γίνεται κατά γραμμές, τότε η ΣΑΠ προκύπτει από τον τύπο (2.9), αφού υπολογιστούν οι σταθερές από τους τύπους (2.10):

$$c1 = N2L = 3 \times 3 = 9,$$

$$c2 = L = 3,\ c0 = bA - c1\ nl1 - c2\ nl2 = 300 - 9 \times 1 - 1 - 3 \times 1 = 288$$

$$loc(BATHij) = 288 + 9i + 3j$$

Αν θέλουμε τώρα να βρούμε τη διεύθυνση του πρώτου byte του στοιχείου BATH12,3, υπολογίζουμε την τιμή της ΣΑΠ για i = 12 και j = 3 :

$$loc = (BATH12,3) = 288 + 9 \times 12 + 3 \times 3 = 288 + 108 + 9 = 405.$$

2.1.5 Πολυδιάστατοι πίνακες

Γενικά, σ΄ έναν πολυδιάστατο πίνακα (multidimensional array) m διαστάσεων N1 X N2 X.....X Nm, κάθε στοιχείο του συνόλου Α προσδιορίζεται από μια m-άδα δεικτών, δηλαδή είναι I = I1 X I2 X....X Im. Η ΣΑΠ είναι κι εδώ γραμμική συνάρτηση των m δεικτών. Π.χ., στην περίπτωση της διάταξης κατά γραμμές αποδεικνύεται ότι είναι

$$Loc(Ai1, i2, \ldots\ldots im) = c0 + c1i1 + c2i2 + \ldots\ldots + cmim \qquad (2.11)$$

όπου

$$cm = L$$

$$c1\text{-}1 = N1c1 \quad i = 2, \ldots, m$$

$$c0 = bA - c1nl1 - c2nl2 - \ldots - cmnlm. \qquad (2.12)$$

Οι διάφορες γλώσσες προγραμματισμού, στην καλύτερη περίπτωση, προσφέρουν πίνακες μέχρι και επτά διαστάσεις.

2.2 Αναζήτηση σε πίνακες

Η αναζήτηση (search) σ΄ έναν πίνακα αναφέρεται στο εξής πρόβλημα: Δοθείσης μιας τιμής να βρεθεί η θέση του στοιχείου στον πίνακα πού έχει την ίδια τιμή με τη δοθείσα, αν υπάρχει, ή να τυπωθεί κάποιο μήνυμα αποτυχίας, αν δεν υπάρχει. Πολλές φορές, η αναζήτηση συνδυάζεται με την διαγραφή ενός στοιχείου. Υπάρχουν διάφοροι μέθοδοι αναζήτησης σε έναν πίνακα. Εδώ, θα αναφερθούμε σε δυο βασικές μεθόδους, τη σειριακή ή γραμμική αναζήτηση (sequential or linear search) και τη δυαδική αναζήτηση (binary search).

2.2.1 Γραμμική αναζήτηση

Η λύση πού δίνει στο παραπάνω πρόβλημα η γραμμική αναζήτηση, είναι η προφανής. Συγκρίνουμε, δηλαδή, τη δοθείσα τιμή με την τιμή κάθε στοιχείου του πίνακα, ξεκινώντας από το πρώτο στοιχείο. Αν βρούμε κάποιο στοιχείο, με τιμή ίση με τη δοθείσα (επιτυχημένη αναζήτηση), τότε σταματάμε και επιστρέφουμε τη θέση του στοιχείου, αλλιώς (αποτυχημένη αναζήτηση) επιστρέφουμε ένα μήνυμα αποτυχίας.

ΑΛΓΟΡΙΘΜΟΣ 2.1: ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος : Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).

Έξοδος : Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα σε περίπτωση επιτυχίας ή μήνυμα σε

περίπτωση αποτυχίας. Εσωτερικά, δεν συμβαίνει τίποτα.

GRAMM – ΑΝΑΖΗΤΗΣΗ (A,N,X)

```
1     I←1                          {Αρχικοποίηση δείκτη θέσης }
2     while (I ≤ N)and (A[I] ≠ X)  {Έλεγχος τέλους επανάληψης}
3     I← I +1                            {Ενημέρωση δείκτη θέσης}

      endwhile
         if A[I] = X               {Αναγνώριση αποτελέσματος}
         then print I              {Επιστροφή θέσης στοιχείου}
6.       else print 'ΑΠΟΤΥΧΙΑ'     {Επιστροφή μηνύματος αποτυχίας}
endif
```

Ο αντίστοιχος αλγόριθμος, είναι ο Αλγόριθμος 2.1 (GRAM- ΑΝΑΖΗΤΗΣΗ) και παρουσιάζεται στο προηγούμενο πλαίσιο, όπου θεωρούμε ένα πίνακα Α μεγέθους Ν. Η μεταβλητή (δείκτης) Ι χρησιμοποιείται για καταχώρηση της θέσης του στοιχείου. Η διάταξη while (βήματα 2 - 3) αυξάνει διαδοχικά την τιμή του δείκτη Ι κατά ένα(βήμα 3), ελέγχοντας κάθε φορά (στη συνθήκη της), αν το στοιχείο του πίνακα στην παρούσα θέση, δηλ. το Α[Ι], είναι ίσο με Χ. Οι επαναλήψεις (συγκρίσεις) σταματούν είτε όταν δεν υπάρχουν άλλα στοιχεία του πίνακα για εξέταση (Ι >Ν) είτε όταν βρεθεί το ζητούμενο στοιχείο (Α[Ι] = Χ). Σημειώστε ότι η σειρά των όρων του τελεστή **end** έχει σημασία. Αν ήταν αντίστροφη, τότε στην περίπτωση αποτυχίας, όταν θα συμβεί Ι > Ν, θα αναζητηθεί το ανύπαρκτο στοιχείο Α[Ι], λόγω της σύγκρισης (Α[Ι]≠ Χ), πριν εκτιμηθεί ο δεύτερος όρος (Ι ≤ Ν) και σταματήσει η παραπέρα διαδικασία. Στη συνέχεια, μια διάταξη if (βήματα 4 - 6) επιστρέφει, ανάλογα με την περίπτωση, είτε την θέση του στοιχείου (βήμα 5) είτε το μήνυμα 'ΑΠΟΤΥΧΙΑ' (βήμα 6).

Η πολυπλοκότητα του αλγόριθμου υπολογίζεται με βάση τον αριθμό των συγκρίσεων πού γίνονται στο βήμα 2. Στη χειρότερη περίπτωση, δηλαδή όταν το ζητούμενο στοιχείο είναι το τελευταίο ή δεν υπάρχει, είναι f (n) = n, διότι θα γίνουν τόσες συγκρίσεις όσα και τα στοιχεία του πίνακα. Στη μέση περίπτωση, αποδεικνύεται ότι,

$$f(n) = \frac{n+1}{2}$$

Παράδειγμα 2.3

Γραμμική αναζήτηση σε διατεταγμένο πίνακα

Θεωρούμε ένα διατεταγμένο πίνακα, δηλαδή έναν πίνακα πού τα στοιχεία του είναι τοποθετημένα κατά αύξουσα (ή αλφαβητική) σειρά. Το ζητούμενο είναι να σχεδιάσουμε ένα αλγόριθμο γραμμικής αναζήτησης για ένα τέτοιο πίνακα.

Το γεγονός ότι ο πίνακας είναι διατεταγμένος μας δίνει τη δυνατότητα για αποδοτικότερη εφαρμογή της γραμμικής αναζήτησης. Η βασική διαφορά από την περίπτωση του μη διατεταγμένου πίνακα είναι ότι τώρα δεν χρειάζεται να συνεχίσουμε την αναζήτηση πέραν του πρώτου στοιχείου πού θα έχει τιμή μεγαλύτερη από τη δοθείσα, διότι δεν έχει νόημα.

ΑΛΓΟΡΙΘΜΟΣ Π2.3 : ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΙΑΤΕΤΑΓΜΕΝΟ ΠΙΝΑΚΑ

Είσοδος : Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).
Έξοδος : Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα σε περίπτωση επιτυχίας ή μήνυμα αποτυχίας. Εσωτερικά, δεν συμβαίνει τίποτα.

```
DIAT – GRAM – ANAZHTHΣH (A,N,X)
1   I ← 1                          {Αρχικοποίηση δείκτη θέσης}
2   while (I ≤ N) and (A[I] ≠ X)   {Έλεγχος τέλους επανάληψης}
3   I ← I +1                       {Ενημέρωση δείκτη θέσης}
    endwhile
4   if A[I] = X                    {Αναγνώριση αποτελέσματος}
5   then print I                   {Επιστροφή θέσης στοιχείου}
6   else print 'ΑΠΟΤΥΧΙΑ'          {Επιστροφή μηνύματος αποτυχίας}
    endif
```

Αυτή η διαφορά αντικατοπτρίζεται στον παραπάνω Αλγόριθμο Π2.3 (DIAT – GRAM – ΑΝΑΖΗΤΗΣΗ), όπου η επανάληψη του σώματος του while (βήμα 3) εξακολουθεί, ενόσω η τιμή του τρέχοντος στοιχείου του πίνακα παραμένει μεγαλύτερη από τη ζητούμενη τιμή (A [I] > X) και δεν έχουμε φθάσει στο τέλος του πίνακα (I ≤ N). Δηλαδή, η επανάληψη σταματά όταν η τιμή του τρέχοντος στοιχείου είναι ίση με τη δοθείσα (επιτυχία) ή μεγαλύτερη από τη δοθείσα (αποτυχία) ή έχουμε εξαντλήσει τον πίνακα (αποτυχία). Το ποια από αυτές τις περιπτώσεις συνέβη, ανιχνεύεται από τη διάταξη if (βήματα 4 – 6).

2.2.2 Δυαδική αναζήτηση

Η δυαδική αναζήτηση προϋποθέτει ότι τα στοιχεία του πίνακα είναι διατεταγμένα, δηλαδή βρίσκονται κατά αύξουσα (ή αλφαβητική) σειρά. Η λύση του προβλήματος κατά τη δυαδική αναζήτηση έχει ως εξής: Συγκρίνουμε την τιμή του δοθέντος στοιχείου με την τιμή του 'μεσαίου' στοιχείου του

πίνακα. Αν το στοιχείο είναι το ζητούμενο τερματίζεται η αναζήτηση με επιτυχία. Αν όχι, τότε, αν η δοθείσα τιμή είναι μεγαλύτερη από αυτή του 'μεσαίου' στοιχείου, ερευνούμε με τον ίδιο τρόπο το τμήμα του πίνακα πού είναι στα δεξιά του, αλλιώς αυτό πού είναι αριστερά του. Αυτή η διαδικασία συνεχίζεται μέχρις ότου εντοπιστεί το ζητούμενο στοιχείο ή εξαντληθούν τα στοιχεία του πίνακα. Ο αντίστοιχος αλγόριθμος είναι ο Αλγόριθμος 2.2 (DIAD – ΑΝΑΖΗΤΗΣΗ), πού παρουσιάζεται στο επόμενο πλαίσιο.

Οι Βοηθητικές μεταβλητές NL και NU, πού αρχικοποιούνται στο βήμα 1, αντιπροσωπεύουν κάθε φορά τον μικρότερο και μεγαλύτερο δείκτη αντίστοιχα, δηλ. τα όρια του τμήματος του πίνακα πού είναι υπό έρευνα, ενώ η μεταβλητή M τη μέση τιμή τους (βήμα 2). Οι τετραγωνικές παρενθέσεις '[ ]' σημαίνουν το ακέραιο μέρος (του αποτελέσματος) της παράστασης πού βρίσκεται ανάμεσά τους (Στην Pascal, αυτό υλοποιείται με τη χρήση του συντελεστή div.) Η διάταξη while (βήματα 3-7) ξεκινά μια επαναληπτική διαδικασία, σε κάθε επανάληψη της οποίας, ερευνάται και ένα τμήμα του πίνακα πού είναι το ένα από τα δύο μισά του προηγουμένου. Η διάταξη if (βήματα 4-6) μέσα στην while, καθόσον τα όρια του μισού του τρέχοντος τμήματος πού θα γίνει το νέο υπό έρευνα τμήμα στην επόμενη επανάληψη. Η επανάληψη σταματά όταν, είτε δεν υπάρχει άλλο στοιχείο για εξέταση (NL > NU) (αποτυχία), είτε βρεθεί το ζητούμενο στοιχείο (A [I] = X) (επιτυχία). Η δεύτερη διάταξη if (βήματα 8-10) επιστρέφει το κατάλληλο αποτέλεσμα.

ΑΛΓΟΡΙΘΜΟΣ 2.2 : ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος : Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).

Έξοδος : Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα, σε περίπτωση επιτυχίας ή μήνυμα αποτυχίας. Εσωτερικά, δεν συμβαίνει τίποτα.

DIAT – ΑΝΑΖΗΤΗΣΗ (A,N,X)

```
1     NL ← 1, NU← N                    {Αρχικοποίηση μεταβλητών}
2     M ← [(NL + NU) /2]               {Εύρεση δείκτη μεσαίου στοιχείου}
3     while (A[M] ≠ X) and (NL ≤ NU)   {Έλεγχος τέλους επανάληψης}
4     if A[M] < X                      {Σύγκριση στοιχείων}
5        then NL ← M + 1               {Ενημέρωση κάτω ορίου}
6        else NU ← M – 1               {Ενημέρωση πάνω ορίου}
endif
7     M ← [(NL +NU)/2]                 {Ενημέρωση δείκτη μεσαίου στοιχείου}
      endwhile
8     if A[M] = X                      {Αναγνώριση αποτελέσματος}
9     then print M                     {Επιστροφή θέσης στοιχείου}
10    else print 'ΑΠΟΤΥΧΙΑ'            {Επιστροφή μηνύματος αποτυχίας}
      endif
```

Η πολυπλοκότητα της δυαδικής αναζήτησης, στη χειρότερη περίπτωση, είναι καλύτερη από αυτή της γραμμικής, δεδομένου ότι, σε κάθε επανάληψη ο αριθμός των συγκρίσεων μειώνεται στο μισό. Αποδεικνύεται ότι είναι f(n) = O (logn) και στη χειρότερη και στη μέση περίπτωση.

Παράδειγμα 2.4

Αναλυτική περιγραφή δυαδικής αναζήτησης με παράδειγμα

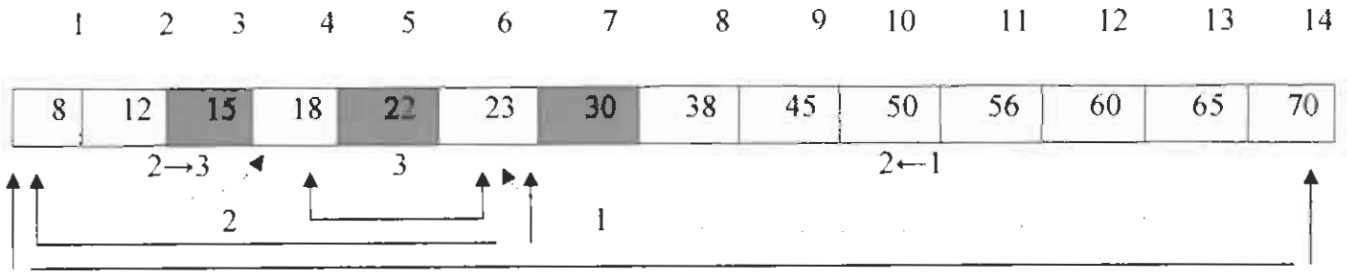
Θεωρούμε ότι μας δίνεται ο παρακάτω πίνακας Α με διατεταγμένα στοιχεία και ζητείται να περιγράψουμε τα βήματα της εφαρμογής της δυαδικής αναζήτησης, στην περίπτωση αναζήτησης της τιμής X = 22. Επίσης, να μετρήσουμε τον αριθμό των συγκρίσεων πού γίνονται και να τον συγκρίνουμε με αυτόν της γραμμικής αναζήτησης.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Α	8	12	15	18	22	23	30	38	45	50	56	60	65	70

Για την παρουσίαση των βημάτων του αλγόριθμου δυαδικής αναζήτησης, χρησιμοποιούμε τον παρακάτω πίνακα όπου φαίνονται οι τιμές των εμπλεκομένων στοιχείων (μεταβλητών κλπ) του Αλγόριθμου 2.2.

Βήμα	A[M]	A[M]≠X	NL≤NU	NL	NU	M
0	-	-	-	1	14	7
1	30	TRUE	TRUE	1	6	3
2	15	TRUE	TRUE	4	6	5
3	22	FALSE	TRUE	4	6	5

Το βήμα 0 αναφέρεται στην αρχικοποίηση των μεταβλητών. Η κάθε σειρά, στον παραπάνω πίνακα, περιέχει τις τιμές των αντίστοιχων στοιχείων (εκφράσεων ή μεταβλητών) μετά την εκτέλεση του αντίστοιχου βήματος, δηλαδή της αντίστοιχης επανάληψης του σώματος της διάταξης while του Αλγόριθμου 2.2. Παρατηρήστε ότι, μετά την εκτέλεση του τρίτου βήματος σταματάς η επανάληψη, διότι τότε A[M] = X (A [5] = X = 22). Η διαδικασία απεικονίζεται γραφικά στο παρακάτω σχήμα όπου, με συνεχή αριθμημένα διπλά βέλη καταδεικνύονται τα υπό εξέταση τμήματα του πίνακα στα αντίστοιχα βήματα, και με διακεκομμένα απλά βέλη οι μεταβολές των άκρων (ορίων) των τμημάτων στις αλλαγές βήματος. Επίσης, κάθε ▢ δείχνει το μεσαίο στοιχείο ενός τμήματος.



A

Όπως είναι φανερό, γίνονται μόνο τρεις συγκρίσεις, ενώ για την εφαρμογή του αλγορίθμου γραμμικής αναζήτησης θα απαιτούνταν πέντε συγκρίσεις, δηλαδή τόσες όσος είναι ο αριθμός των στοιχείων από την αρχή του πίνακα, θα ήταν μεγαλύτερη. Γενικά, όσο πιο κοντά στην αρχή του πίνακα βρίσκεται ένα στοιχείο, τόσο πιο πλεονεκτική γίνεται η γραμμική αναζήτηση.

Ο αλγόριθμος δυαδικής αναζήτησης στηρίζεται στη διχοτόμηση του πίνακα ή του υπό εξέταση τμήματός του. Με βάση τον αλγόριθμο δυαδικής αναζήτησης σχεδιάστε ένα αλγόριθμο `τριαδικής` αναζήτησης, δηλαδή έναν αλγόριθμο πού να τριχοτομεί τον πίνακα ή το υπό εξέταση τμήμα του. Στην περίπτωση αυτή, επιλέγουμε ένα από τα τρία τμήματα και προχωρούμε τριχοτομώντας το, έως ότου είτε βρούμε το ζητούμενο στοιχείο είτε αποτύχουμε.

2.3 Εγγραφές

Όπως αναφέραμε και στο Κεφάλαιο 1, η εγγραφή (record) είναι μια θεμελιώδης δομή δεδομένων πού υπάρχει ενσωματωμένη σε πολλές γλώσσες προγραμματισμού υψηλού επιπέδου. Είναι μια σημαντική δομή πού χρησιμοποιείται σαν δομικό στοιχείο για άλλες ανώτερες δομές.

2.3.1 Ορισμός και αναπαράσταση

Εγγραφή είναι μια συλλογή ενός πεπερασμένου πλήθους αλληλοσχετιζόμενων στοιχείων πού, συνήθως, δεν είναι του ιδίου τύπου, αντίθετα με ότι συμβαίνει σ΄ ένα πίνακα. Λόγω αυτού του χαρακτηριστικού, η εγγραφή θεωρείται ως ετερογενής δομή (heterogeneous structure).

Κάθε στοιχείο μιας εγγραφής περιγράφεται από ένα πεδίο (field). Κάθε πεδίο έχει ένα όνομα (identifier) και ένα τύπο (type). Κάθε στοιχείο μπορεί να συνίσταται από άλλα, μερικά στοιχεία. Τότε καλείται σύνθετο στοιχείο, ενώ στην αντίθετη περίπτωση απλό στοιχείο. Κατ΄ αντιστοιχία, διακρίνουμε απλά πεδία και σύνθετα πεδία.

Έστω μια εγγραφή (ή μεταβλητή τύπου εγγραφής) R και (fd1,fd2,.....,fdn) η αντίστοιχη n-άδα πεδίων με τύπους (δηλ. Πεδία τιμών) T1, T2,.....,Tn, αντίστοιχα. Τότε, θα αναφερόμαστε στα στοιχεία

της R με τους συμβολισμούς fd1(R), fd2(R), …., fdn(R). Δηλαδή, τα πεδία λειτουργούν σαν δείκτες των στοιχείων μιας εγγραφής και, δεδομένου ότι είναι διαφόρων τύπων, δεν τίθεται θέμα διάταξης των στοιχείων μιας εγγραφής με βάση την τιμή τους, παρά μόνο με βάση τη θέση τους. Το πεδίο τιμών της εγγραφής R είναι το σύνολο T = T1 X T2 X.....X Tn, δηλαδή το καρτεσιανό γινόμενο των πεδίων τιμών των πεδίων της εγγραφής. Αν κάποιο πεδίο fd_χ, είναι ένα σύνθετο πεδίο με μερικά (απλά) πεδία (fd_x^1, fd_x^2,….,fd_x^m), τότε θα αναφερόμαστε στα αντίστοιχα απλά στοιχεία με τους συμβολισμούς $fd_x^1(fd_x(R))$, $fd_x^2(fd_x(R))$,…… $fd_x^m(fd_x(R))$.

Ο καθένας από τους παραπάνω συμβολισμούς λειτουργεί ως μεταβλητή. Τα πεδία fd1, fd2,……,fdn καλούνται πεδία πρώτου επιπέδου, τα fd_x^1, fd_x^2,….., fd_x^m δεύτερου επιπέδου κ.ο.κ.
Λόγω της ύπαρξης σύνθετων πεδίων διαφόρου βάθους επιπέδων, το οργανωτικό σχήμα μιας εγγραφής απεικονίζεται σαν μια ιεραρχία πεδίων.

Παράδειγμα 2.5

Παράδειγμα απεικόνιση εγγραφής

Ένα σχολείο κρατά τα στοιχεία κάθε μαθητή σε μια εγγραφή με τα εξής πεδία: ΑΡ. ΜΗΤΡΩΟΥ, ΟΝΟΜΑΤΕΠΩΝΥΜΟ, ΟΝ. ΠΑΤΡΟΣ, ΦΥΛΛΟ, ΗΜ. ΓΕΝΝΗΣΕΩΣ, ΤΑΞΗ. Το ΟΝΟΜΑΤΕΠΩΝΥΜΟ είναι ένα σύνθετο πεδίο με μερικά πεδία τα ΟΝΟΜΑ και ΕΠΩΝΥΜΟ. Επίσης, το πεδίο ΗΜΕΡ. ΓΕΝΝΗΣΕΩΣ είναι ένα σύνθετο πεδίο με μερικά πεδία τα ΗΜΕΡΑ, ΜΗΝΑΣ, ΕΤΟΣ. Η εγγραφή αυτή απεικονίζεται παρακάτω με δυο τρόπους, από όπου φαίνεται η ιεραρχική δομή της.

ΜΑΘΗΤΗΣ
- ΑΡ. ΜΗΤΡΩΟΥ
- ΟΝΟΜΑΤΕΠΩΝΥΜΟ
  - ΟΝΟΜΑ
  - ΕΠΩΝΥΜΟ
- ΟΝ. ΠΑΤΡΟΣ
- ΦΥΛΛΟ
- ΗΜ. ΓΕΝΝΗΣΕΩΣ
  - ΗΜΕΡΑ
  - ΜΗΝΑΣ
  - ΕΤΟΣ
- ΤΑΞΗ

ΜΑΘΗΤΗΣ

ΑΡ. ΜΗΤΡΩΟΥ	ΟΝΟΜΑΤΕΠΩΝΥΜΟ		ΟΝ. ΠΑΤΡΟΣ	ΦΥΛΛΟ	ΗΜ. ΓΕΝΝΗΣΕΩΣ			ΤΑΞΗ
	ΟΝΟΜΑ	ΕΠΩΝΥΜΟ			ΗΜΕΡΑ	ΜΗΝΑΣ	ΕΤΟΣ	

Τα πεδία ΑΡ. ΜΗΤΡΩΟΥ, ΟΝΟΜΑΤΕΠΩΝΥΜΟ, ΟΝ. ΠΑΤΡΟΣ, ΦΥΛΛΟ, ΗΜ. ΓΕΝΝΗΣΗΣ και ΤΑΞΗ είναι πεδία πρώτου επιπέδου, ενώ τα ΟΝΟΜΑ, ΕΠΩΝΥΜΟ, ΗΜΕΡΑ, ΜΗΝΑΣ και ΕΤΟΣ, δεύτερου επιπέδου. Αν θέλουμε να αναφερθούμε στην τάξη ενός μαθητή, τότε χρησιμοποιούμε το συμβολισμό ΤΑΞΗ (ΜΑΘΗΤΗΣ), ενώ για να αναφερθούμε στο όνομα του μαθητή, τον συμβολισμό ΟΝΟΜΑ (ΟΝΟΜΑΤΕΠΩΝΥΜΟ(ΜΑΘΗΤΗΣ)). Κάθε ένα από τα απλά πεδία έχει σαν τιμή το αντίστοιχο στοιχείο (πληροφορία) για τον μαθητή.

Οι βασικές πράξεις σε μια εγγραφή, όπως και σε έναν πίνακα, είναι η προσπέλαση (ή ανάκτηση) στοιχείου και η καταχώρηση (ή ενημέρωση)στοιχείου. Στην αλγοριθμική μας γλώσσα αυτές εκφράζονται ως εξής:

X ← FDX(R) (προσπέλαση ή ανάκτηση)

FDX(R) ← X (καταχώρηση ή ενημέρωση)

Όπου FDX είναι ένα πεδίο πρώτου επιπέδου.

Παράδειγμα 2.6α

Ορισμός εγγραφής στην Pascal

Ας θεωρήσουμε μια απλοποιημένη εκδοχή της εγγραφής του Παραδείγματος 2.5, όπου ένας μαθητής χαρακτηρίζεται μόνο από τον αριθμό μητρώου, το ονοματεπώνυμό του και το όνομα του πατέρα του. Η εγγραφή αυτή μπορεί να υλοποιηθεί στην Pascal, όπως παρουσιάζεται παρακάτω.

```
type
 onomateponrec = record
                   onoma:array [1...12] of char;
                   eponymo: array [1...24] of char;
                 end;

mathitisrec = record
                armitrwou: integer;
               onomatepon: onomateponrec;
               onpatros:array [1...12] of char:
                end;

var mathitis: mathitisrec;
```

Με τις παραπάνω εκφράσεις, ορίζουμε δύο τύπους δεδομένων με ονόματα onepontec και mathitisrec που η κάθε μία είναι μια εγγραφή (record). Η δεύτερη, είναι η κυρίως εγγραφή που χρησιμοποιεί την πρώτη για να δηλώσει τον τύπο του ενός σύνθετου πεδίου της (onepon). Η πρώτη δομή εγγραφής (oneponrec) έχει δύο απλά πεδία (onoma, eponymo). Το πρώτο, δηλώνει σας ένας πίνακας 12 χαρακτήρων, ενώ το δεύτερο, σαν πίνακας 24 χαρακτήρων. Αυτός είναι ο τρόπος που δηλώνουμε ακολουθίες χαρακτήρων (π.χ. λέξεις) στην Pascal. Η κυρίως εγγραφή έχει δύο απλά πεδία (armitrwou και onpatrow) και ένα σύνθετο (onomatepon). Το πρώτο δηλώνεται τύπου ακέραιος (integer), ενώ το δεύτερο σαν ακολουθία 12 χαρακτήρων. Το σύνθετο πεδίο δηλώνεται τύπου (εγγραφής) "onomateponrec". Τέλος, δηλώνουμε μια μεταβλητή (mathitis) τύπου (εγγραφής) mathitisrec, στην οποία μπορούμε να εισάγουμε τιμές για τα διάφορα πεδία της εγγραφής. Στην Pascal για να προσπελάσουμε το πεδίο "armitrwou" της μεταβλητής "mathitis" γράφουμε mathitis. Atmitrwou. Επίσης προκειμένου για το πεδίο "onoma" γράφουμε mathitis. Onepon. Onoma. Αυτές οι εκφράσεις λειτουργούν σαν μεταβλητές, στις οποίες μπορούμε να καταχωρήσουμε μια τιμή ή να καταχωρήσουμε την τιμή τους σε κάποια άλλη μεταβλητή του ίδιου τύπου.

Ορισμός εγγραφής στη C

Η εγγραφή του Παραδείγματος 2. 6α μπορεί να υλοποιηθεί στην C ως εξής:

```
Typedef
    Struct {
        Char onoma [12];
        Char eponymo [24];
            } onomateponrec;
                typedef                                    Παράδειγμα 2.6β
                    struct {
                        int armitrwou;
                        onomateponrec onomatepon;
                        char onratros [12];
                            } mathitisrec;
                mathitisrec mathitis;
```

Ορίζονται δύο τύποι (onomateponrec, mathitisrec) που είναι δύο δομές (struct) της C και η καθεμία παριστάνει μια εγγραφή. Η δεύτερη, είναι η κυρίως εγγραφή (δομή) που χρησιμοποιεί την πρώτη στη δήλωση του τύπου ενός σύνθετου πεδίου (μέλους) της (onomatepon). Το int δηλώνει τον ακέραιο τύπο. Για τα υπόλοιπα, ισχύουν παρόμοια με αυτά στην Pascal.

2.3.2 Πινάκες εγγραφών

Πολλές φορές, στην πράξη χρησιμοποιούνται δομές που είναι συνδυασμοί δύο άλλων δομώ. Αυτές οι δομές ονομάζονται *συνδυασμένες ή μεικτές δομές (composite structures).* Μία συνήθης και χρήσιμη περίπτωση μεικτής δομής είναι ο *πίνακας εγγράφων (array of records),* δηλαδή ένας πίνακας που τα στοιχεί του είναι εγγραφές. Συνήθως. Κάθε εγγραφή έχει ένα πεδίο που ονομάζεται *κλειδί (Key)* και προσδιορίζει με μοναδικό τρόπο κάθε στιγμιότυπο της εγγραφής. Δηλαδή, δεν υπάρχουν δύο στιγμιότυπα μιας εγγραφής στις οποίες το κλειδί να έχει την ίδια τιμή. Στιγμιότυπο λέγεται κάθε συγκεκριμένη εγγραφή. Π.χ. κάθε εγγραφή που έχει στοιχεία ενός συγκεκριμένη εγγραφή. Π.Χ κάθε εγγραφή που έχει στοιχεία ενός συγκεκριμένου μαθητή είναι στιγμιότυπο της εγγραφής ΜΑΘΗΤΗΣ (Παράδειγμα 2.5). Ένα κλειδί στην εγγραφή ΜΑΘΗΤΗΣ είναι το ΑΡ. ΜΗΤΡΩΟΥ. Η αναζήτηση στα στοιχεία ενός τέτοιου πίνακα με βάση το κλειδί των εγγράφων – στοιχείων του πίνακα.

Παράδειγμα 2.7 *Γραμμική αναζήτηση σε διαταγμένο πίνακα εγγράφων*

Θεωρούμε έναν πίνακα εγγράφων, όπου συμβολίζουμε με KLEIDI το πεδίο που αποτελεί το κλειδί κάθε εγγραφής. Επίσης θεωρούμε ότι ο πίνακας είναι διατεταγμένος με βάση το κλειδί κάθε εγγραφής. Το ζητούμενο είναι να σχεδιάσουμε έναν αλγόριθμο γραμμικής αναζήτησης για ένα τέτοιο πίνακα.

Ο ζητούμενος αλγόριθμος αποτελεί μια παραλλαγή του Αλγόριθμου Π2.3. Το σημείο παραλλαγής προερχόμενο από το γεγονός ότι τα στοιχεία του πίνακα τώρα είναι εγγραφές και η τιμή που αναζητούμε αφορά το κλειδί εγγραφής. Έτσι, ο νέος αλγόριθμος προκύπτει από τον Π2.3, αν όπου A[I] θέσουμε KLEIDI (A[I]).

Σύνοψη

Ο πίνακας και η ***εγγραφή*** είναι δύο θεμελιώδεις δομές, στις οποίες στηρίζεται η υλοποίηση άλλων ανώτερων δομών. Και οι δύο δομές αποτελούν συλλογές στοιχείων. Κάθε στοιχείο ενός πίνακα αντιστοιχεί σ΄ ένα ***δείκτη,*** από ένα σύνολο δεικτών, ενώ κάθε στοιχείο μια εγγραφής σ΄ ένα ***πεδίο*** από μια πλειάδα πεδίων. Τα στοιχεία ενός πίνακα είναι του ίδιου τύπου και η σχέση μεταξύ τους γραμμική. Γι΄ αυτό, ο πίνακας είναι μια ***ομογενής*** και ***γραμμική δομή***. Αντίθετα, τα στοιχεία μιας εγγραφής δεν είναι, συνήθως, του ίδιου τύπου και η σχέση μεταξύ τους είναι ιεραρχική. Γι αυτό, η εγγραφή είναι μια ***ετερογενής*** και ***μη γραμμική δομή.*** Και οι δύο, όμως, είναι ***στατικές δομές***, δηλαδή το μέγεθος τους δεν μεταβάλλεται κατά την εκτέλεση του προγράμματος.

Τα στοιχεία ενός πίνακα αποθηκεύονται σε γειτονικές θέσεις στη μνήμη του Η/Υ. Αυτό δίνει τη δυνατότητα στον πίνακα να είναι ***δομή τυχαίας προσπέλαση***, δηλαδή ο χρόνος προσπέλασης ενός στοιχείου είναι ανεξάρτητος από τη θέση του στοιχείου. Ο υπολογισμός της διεύθυνσης ενός στοιχείου στη μνήμη, από ένα μεταφραστικό πρόγραμμα, γίνεται με τη ***συνάρτηση απεικόνισης πίνακα*** (ΣΑΠ), που είναι μία γραμμική συνάρτηση του δείκτη του στοιχείου. Το ίδιο συμβαίνει και στους ***δισδιάστατους*** και στους ***τρισδιάστατους*** και , εν γένει, στους ***πολυδιάστατους πίνακες***, μόνο που τώρα αντί για ένα δείκτη έχουμε ζεύγος, τριάδα, ….., πλειάδα δεικτών αντίστοιχα για κάθε στοιχείο.

Και σε μία εγγραφή όμως, δεδομένου ότι τα πεδία λειτουργούν σαν δείκτες έχουμε τυχαία προσπέλαση σε κάθε πεδίο.

Η αναζήτηση είναι μια από τις πράξεις σε ένα πίνακα που μας ενδιαφέρει. Η γραμμική αναζήτηση είναι μια μέθοδος αναζήτησης που ψάχνει ένα – ένα τα στοιχεία ενός πίνακα από την αρχή μέχρι το τέλος, έως ότου είτε βρει το ζητούμενο στοιχείο ή αποτύχει. Η δυαδική αναζήτηση, που εφαρμόζεται μόνο σε διατεταγμένες πίνακες, στηρίζεται σε διαδοχικές διχοτομήσεις του πίνακα. Εν γένει, η δυαδική αναζήτηση έχει καλύτερη χρονική απόδοση (μικρότερη πολυπλοκότητα) από τη γραμμική.

Ο πίνακας εγγράφων είναι μια ενδιαφέρουσα περίπτωση μεικτής δομής, όπου κάθε στοιχείο του πίνακα είναι ένα στιγμιότυπο μιας εγγραφής. Εδώ, η αναζήτηση γίνεται με βάση το κλειδί των εγγράφων, το πεδίο δηλαδή που έχει μοναδική τιμή σε κάθε στιγμιότυπο.

3ο ΚΕΦΑΛΑΙΟ

ΓΕΝΙΚΕΣ ΛΙΣΤΕΣ

Εισαγωγή

Στο κεφάλαιο αυτό διαπραγματευόμαστε τις γενικές λίστες, μια κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη ενότητα, προσδιορίζεται η έννοια της λίστας σε αντιπαράθεση με αυτή του πίνακα. Επίσης, παρουσιάζονται οι διάφοροι τύποι λιστών και περιγράφονται οι αναπαραστάσεις των δύο βασικών τύπων, τις συνεχόμενης και της απλά συνδεδεμένης λίστας.

Στη δεύτερη και Τρίτη ενότητα, παρουσιάζουμε τους βασικούς αλγόριθμους για τις πράξεις διαπέραση, εισαγωγή και διαγραφή σε συνεχόμενη και συνδεδεμένη λίστα αντίστοιχα. Επίσης, δίνονται υποδείξεις για την πράξη της αναζήτησης στις λίστες αυτές. Εδώ θεωρούμε σκόπιμο να κάνουμε την εξής διευκρίνιση : Υπάρχει ένα σύνολο πράξεων σε μια λίστα που αποτελεί μέρος του ορισμού της ως δομής δεδομένων, όπως εξηγήσαμε στο κεφάλαιο « Τύποι και Δομές Δεδομένων». Τέτοιες πράξεις, είναι η δημιουργία λίστας, ο έλεγχος κενής λίστας, ο έλεγχος γεμάτης λίστας, η εισαγωγή στοιχείου, κ.λ.π. Πράξεις σαν τις τρεις πράξεις, θα μπορούσαμε να τις χαρακτηρίσουμε ως 'στοιχειώδεις'.Η υλοποίησή τους είναι σχετική απλή και συνδέεται άμεσα με την αντίστοιχη γλώσσα προγραμματισμού. Εμείς θα ασχοληθούμε, όχι με στοιχειώδεις πράξεις αλλά με τις υπόλοιπες που τις χαρακτηρίσαμε ως κυριότερες.

3.1 Ορισμός, αναπαράσταση και τύποι λιστών

3.1.1 Έννοια και τύποι λίστας

Παράδειγμα έννοιας της λίστας

Στη καθημερινή μας ζωή, πολλές φορές, χρησιμοποιούμε την έννοια της λίστας Για παράδειγμα, χρησιμοποιούμε λίστες αγορών, εργασιών κ.λ.π. Ας θεωρήσουμε μια λίστα εργασιών που πρέπει να διεκπεραιώσουμε σε μια ημέρα. Οι εργασίες αυτές συνήθως αναγράφονται με κάποια σειρά, π.χ. τέτοια, ώστε να ελαχιστοποιεί το συνολικό χρόνο της διεκπεραίωσης. Έτσι, υπάρχει η πρώτη, η δεύτερη, η Τρίτη κ.λ.π και η τελευταία εργασία. Έχουμε δε κατά νου, να διέλθουμε όλη τη λίστα ξεκινώντας από την πρώτη εργασία και καταλήγοντας στην τελευταία για να τις διεκπεραιώσουμε. Κατά τη διάρκεια της ημέρας όμως, θα χρειαστεί είτε να σβήσουμε κάποιες εργασίες που πραγματοποιήθηκαν είτε να εισάγουμε κάποιες νέες που προέκυψαν εν τω μεταξύ είτε και να μετακινήσουμε κάποιες εργασίες σε άλλες θέσεις. Μια λίστα είναι μια πεπερασμένη ακολουθία στοιχείων του ίδιου τύπου (τύπος βάσης).

Η ακολουθία (sequence) είναι μια μαθηματική έννοια που αναφέρεται σε μια συλλογή θεωρητικά άπειρων στοιχείων. Μια πεπερασμένη ακολουθία L,n στοιχείων, είναι μια πλειάδα n στοιχείων (ή αλλιώς n-άδα στοιχείων) $(L_1,L_2,\ldots,L_n)$, όπου L_1 είναι το πρώτο στοιχείο και L_n το τελευταίο. Κάθε στοιχείο L_i έχει ένα προηγούμενο(predecessor) στοιχείο $L_{(i-1)}$ και ένα επόμενο (successor) στοιχείο (L_{i+1}), πλην του πρώτου, που έχει μόνο επόμενο, και του τελευταιου, που έχει μόνο προηγούμενο.

Η θέση (position) ενός στοιχείου σε μια λίστα (ακολουθία) είναι χαρακτηριστική ιδιότητα του στοιχείου. Υπάρχει, δηλαδή, μια διάταξη σε μια λίστα που δεν καθορίζεται από την τιμή του στοιχείου αλλά από τη θέση του. Η θέση του στοιχείου L_i είναι η 'i '. Το πρώτο στοιχείο μιας λίστας ονομάζεται κεφαλή (head). Μια λίστα με κανένα στοιχείο ονομάζεται κενή λίστα (null list).

Από τα παραπάνω, εκ πρώτης όψεως, ίσως να συμπεραίνεται ότι η λίστα L δεν είναι κάτι διαφορετικό από ένα πίνακα L με I={1,...,n}. Όμως, δεν είναι έτσι. Υπάρχουν σημαντικές διαφορές μεταξύ τους, που πηγάζουν από το γεγονός ότι ο πίνακας στηρίζεται στις έννοιες του συνόλου και της απεικόνισης (συνάρτησης), ενώ μια λίστα στηρίζεται στην έννοια της ακολουθίας. Έτσι ένας πίνακας θεωρείται σαν μια δομή τυχαίας προσπέλασης, όπως είδαμε , ενώ μια λίστα είναι στην ουσία μια δομή ακολουθιακής ή σειριακής προσπέλασης (sequential access). Για να φθάσουμε δηλαδή, σε ένα στοιχείο μιας λίστας πρέπει να περάσουμε από όλα τα προηγούμενα ξεκινώντας από το πρώτο. Δεύτερον, το μέγεθος ενός πίνακα παραμένει σταθερό, ενώ το μέγεθος μιας λίστας όχι, διότι συνήθως απαιτούνται διαγραφές παλαιών και εισαγωγές νέων στοιχείων. Δηλαδή, μια λίστα είναι μια δυναμική δομή και όχι στατική, όπως ο πίνακας.

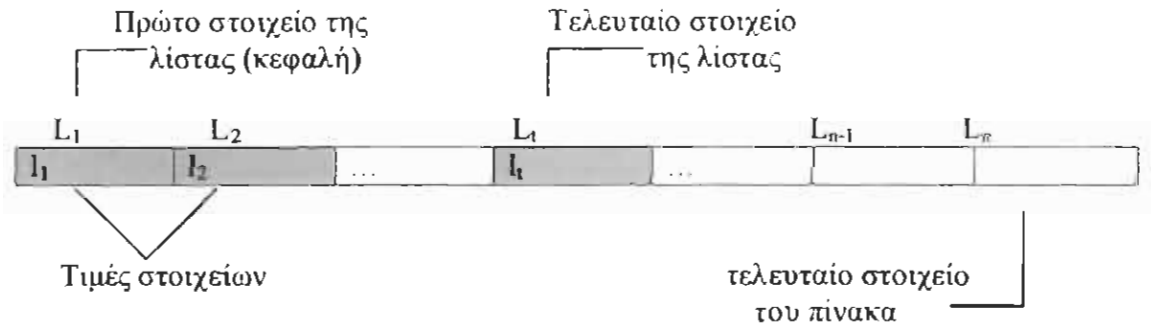
Τις λίστες μπορούμε να τις διακρίνουμε σε δύο μεγάλες κατηγορίες, ανάλογα με το βαθμό ελευθερίας που παρέχουν στην εφαρμογή των διαφόρων πράξεων σε αυτές, τις γενικές λίστες (general lists) και τις ειδικές λίστες (special lists).Οι γενικές λίστες επιτρέπουν την εισαγωγή και διαγραφή στοιχείων / κόμβων σε οποιαδήποτε θέση στη λίστα, ενώ οι ειδικές θέτουν περιορισμούς. Ειδικές λίστες είναι η 'στοίβα' και η 'ουρά', που εξετάζονται αργότερα.

Υπάρχουν δύο τρόποι αναπαράστασης μιας λίστας είτε γενικής είτε ειδικής. Με τον πρώτο τρόπο, την συνεχόμενη αναπαράσταση (contiguous representation), τα στοιχεία της λίστας καταλαμβάνουν γειτονικές ή συνεχόμενες (contiguous) θέσεις στη μνήμη. Εδώ, η μετάβαση από το ένα στοιχείο στο επόμενο γίνεται με την μετάβαση στην επόμενη θέση μνήμης. Με το δεύτερο τρόπο, τη συνδεδεμένη αναπαράσταση (linked representation), τα στοιχεία της λίστας αποθηκεύονται σε μη διαδοχικές θέσεις στη μνήμη του Η/Υ. Τώρα η μετάβαση από το ένα στοιχείο στο επόμενο, πραγματοποιείται με τη χρήση ενός συνδέσμου (link) μεταξύ των στοιχείων. Αντίστοιχα, διακρίνουμε δύο κατηγορίες γενικών λιστών, τις συνεχόμενες λίστες (contiguous lists) και τις συνδεδεμένες λίστες (linked lists).

3.1.2 Αναπαράσταση συνεχόμενης λίστας

Η αναπαράσταση μιας συνεχόμενης λίστας L γίνεται μέσω ενός μονοδιάστατου πίνακα. Στη περίπτωση αυτή, τα στοιχεία της λίστας καταλαμβάνουν μέρος του πίνακα, έτσι, ώστε να είναι δυνατόν να μεταβάλλεται το μέγεθος της λίστας, όπως απεικονίζεται στο σχήμα 1, όπου χρησιμοποιούμε ένα πίνακα L με I ={1,..., n }. Το μέγεθος πρέπει να είναι μεγαλύτερο από ή ίσο με το πιθανό μέγιστο μέγεθος της λίστας που αναπαριστά. Δηλαδή, το τελευταίο στοιχείο της λίστας (L_t) πρέπει να βρίσκεται κάθε φορά πριν από την τελευταία θέση του πίνακα (n), ώστε να υπάρχει περιθώριο μεταβολής του μεγέθους της λίστας, λόγω εισαγωγής νέων στοιχείων.

Σχήμα 1: Αναπαράσταση συνεχόμενης λίστας



Η δυσκολία στη περίπτωση αυτή είναι ο καθορισμός του μεγέθους του πίνακα. Αν είναι σχετικά μικρό, τότε ενδεχομένως θα έχουμε υπερχείλιση (overflow),δηλαδή χάσιμο, στοιχείων της λίστας σε κάποιες μετακινήσεις. Αν είναι σχετικά μεγάλο, τότε θα έχουμε σπατάλη χώρου στη μνήμη, διότι μέρος του πίνακα θα μένει αχρησιμοποίητο. Επίσης, πρέπει να υπάρχει κάποια μεταβλητή που να περιέχει κάθε φορά το δείκτη του τελευταίου στοιχείου της λίστας για να γνωρίζουμε το τέλος της. Τέλος, αξίζει να

σημειωθεί ότι, λόγω της υλοποίησης της συνεχόμενης λίστας μέσω ενός πίνακα, έχουμε δυνατότητα άμεσης προσπέλασης των στοιχείων της.

Συνήθως σε μια γλώσσα υψηλού επιπέδου μια συνεχόμενη λίστα ορίζεται σαν μια εγγραφή με δύο πεδία, από τα οποία το ένα είναι ένας πίνακας κάποιου μεγέθους και το άλλο είναι ένας μετρητής που αποθηκεύει το δείκτη του τελευταίου στοιχείου της λίστας. Στο παρακάτω Παράδειγμα 3.1α, παρουσιάζεται μια τέτοια υλοποίηση στην Pascal και στο Παράδειγμα 3.1β στη C.

ΠΑΡΑΔΕΙΓΜΑ 3.1α Ορισμός συνεχόμενης λίστας στη Pascal

Ο ορισμός της δομής μιας συνεχόμενης λίστας μπορεί να γίνει στη Pascal ως εξής :

```
type
  slista= record
            stoixeia : array [1..n] of typestoixeiou;
            telos: 0...n
          end;
```

Ορίζεται, δηλαδή, ένας νέος τύπος δεδομένων (type) με όνομα slista που είναι μια εγγραφή (record) με δύο πεδία.Το πρώτο πεδίο (stoixeia) είναι ένας πίνακας (array) με δείκτες από το σύνολο ακεραίων {1,2,...,n},δηλ. μεγέθους n, και στοιχεία κάποιου τύπου, έστω 'typostoixeiou'.Το δεύτερο πεδίο (telos) είναι μια ακέραια μεταβλητή που παίρνει τιμές από το σύνολο ακεραίων {0,1,2,...,n}.

Στο πεδίο 'stoixeia'αποθηκεύονται τα στοιχεία της λίστας.Το πεδίο 'telos' παίζει το ρόλο του δείκτη του τέλους της λίστας και ενημερώνεται κάθε φορά που γίνεται κάποια αλλαγή στο μέγεθος της λίστας ,λόγω εισαγωγής ή διαγραφής κάποιου στοιχείου της. Επομένως ,το πεδίο αυτό μας δείχνει το μέγεθος της λίστας. Η τιμή του πεδίου 'telos' δεν πρέ[ει να υπερβεί την τιμή 'n', διότι τότε έχουμε υπερχείλιση. Επίσης, παίρνει την τιμή '0' στην περίπτωση κενής λίστας.

ΠΑΡΑΔΕΙΓΜΑ 3.1β Ορισμός συνεχόμενης λίστας στη C

Ο ορισμός της δομής μιας συνεχόμενης λίστα μπορεί να γίνει στην C ως εξής :

```
Typedef
    struct{
        typostoixeiou stoixeia [n];
        int telos;
          }slista;
```

Ορίζεται,δηλαδή, ένας νέος τύπος δεδομένων (typedef) με όνομα slista που είναι μια δομή/αγγραφή (struct) με δύο μέλη/πεδία. Το πρώτο πεδίο (stoixeia) είναι ένας πίνακας μεγέθους n, και στοιχεια κάποιου τύπου, έστω 'typostoixeiou'.Το δεύτερο πεδίο (telos) είναι μια ακέραια (int)μεταβλητή (στη C δεν υπάρχει τύπος αντίστοιχος της υποπεριοχής ακεραίων ([1..n] της Pascal). Για τα υπόλοιπα ισχύουν παρόμοια, όπως και στο Παράδειγμα 3.1α.

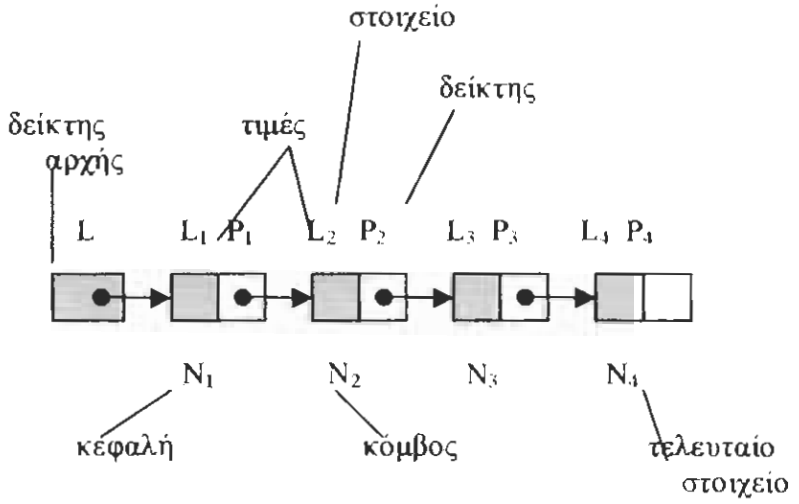
3.1.3 Αναπαρασταση Συνδεδεμενης Λιστας

Για την αναπαράσταση μιας συνδεδεμένης λίστας μαζί με την τιμή κάθε στοιχείου αποθηκεύεται και ένας σύνδεσμος (link) ή δείκτης (pointer) που δείχνει στην (πρώτη) θέση μνήμης του επόμενου στοιχείου της λίστας. Επειδή λοιπόν τα στοιχεία μιας συνδεδεμένης λίστας είναι πιο σύνθετα από αυτά μιας συνεχόμενης λίστας, αποκαλούνται κόμβοι (nodes). Ένας κόμβος αποτελείται από δύο τμήματα. Τα αριστερό περιέχει τη πληροφορία (τιμή) του κόμβου και συνιστά το στοιχείο του κόμβου. Το δεξιό τμήμα περιέχει τη διεύθυνση της πρώτης θέσης μνήμης του επόμενου κόμβου, είναι δηλαδή ένας δείκτης στον επόμενο κόμβο της λίστας. Μια λίστα με κόμβους τέτοιας δομής ονομάζεται απλά συνδεμένη λίστα (one-way or singly linked list).

Στο σχήμα 2 παρουσιάζεται η γραφική απεικόνιση μιας απλά συνδεδεμένης λίστας τεσσάρων στοιχείων $L=(l_1, l_2, l_3, l_4)$ που αποτελείται δηλαδή από τέσσερις κόμβους

(N_1, N_2 , N_3 , N_4). Κάθε κόμβος N_i παριστάνεται από ένα επίμηκες ορθογώνιο παραλληλόγραμμο χωρισμένο σε δύο τμήματα. Το αριστερό τμήμα παριστάνει το αντίστοιχο στοιχείο L_i της λίστας και περιέχει την αντίστοιχη τιμή l_i. Το δεξιό τμήμα παριστάνει το δείκτη P_i του κόμβου που περιέχει τη διεύθυνση του επόμενου κόμβου της λίστας. Το γεγονός ότι ο δείκτης αυτός δείχνει στον επόμενο κόμβο απεικονίζεται με ένα ειδικό βέλος (→). Το τελευταίο στοιχείο της λίστας (κόμβος N_4) δεν περιέχει βέλος, αλλά μόνο την έντονη τελεία του βέλους , διότι εφόσον είναι ο τελευταίος κόμβος της λίστας δε δείχνει πουθενά. Ο κενός δείκτης θεωρείται ότι έχει σαν τιμή την ειδική τιμή 'NIL'. Η μεταβλητή δείκτη L ονομάζεται δείκτης αρχής, δείχνει τη κεφαλή της λίστας (κόμβος N1) και ουσιαστικά αντιπροσωπεύει τη λίστα. Σε κενή λίστα, ο δείκτης αυτός έχει τιμή NIL.

Σχήμα 2: Σχηματική παράσταση απλά συνδεδεμένης λίστας



Η υλοποίηση μιας απλά συνδεδεμένης λίστας μπορεί να γίνει κατά δύο τρόπους. Ο ένας τρόπος είναι με τη χρήση πινάκων. Συνήθως, χρησιμοποιούνται δύο πίνακες, εκ των οποίων ο ένας περιέχει τα στοιχεία των κόμβων και ο δεύτερος τους δείκτες των κόμβων. Δηλαδή, τα τμήματα κάθε κόμβου (στοιχείο, δείκτης) αντιστοιχούν σε ομοθέσια στοιχεία των δύο πινάκων. Ο τρόπος αυτός χρησιμοποιείται για την υλοποίηση

συνδεδεμένων λιστών στις παλαιότερες γλώσσες υψηλού επιπέδου όπως η FORTRAN, η BASIC και η COBOL, που δε διαθέτουν μηχανισμό δημιουργίας δεικτών (συνδέσμων).

Ο δεύτερος τρόπος χρησιμοποιεί μεταβλητές δείκτη (pointer variables). Ο δείκτης (pointer) είναι ένας τύπος δεδομένων που έχει πεδίο τιμών το σύνολο των διευθύνσεων της μνήμης του Η /Υ. Μια μεταβλητή δείκτη παίρνει σαν τιμές διευθύνσεις θέσεων μνήμης που αντιστοιχούν σε κόμβους, δηλαδή σε σύνθετες τιμές. Έτσι, για κάθε κόμβο χρησιμοποιείται και μια μεταβλητή δείκτη (ή ένας δείκτης).

Ο τρόπος αυτός χρησιμοποιείται για υλοποίηση με γλώσσες υψηλού επιπέδου που παρέχουν κάποιο μηχανισμό δημιουργίας δεικτών, όπως η Pascal και η C.Στο κεφάλαιο αυτό, ασχολούμαστε με αυτό το δεύτερο τρόπο υλοποίησης συνδεδεμένων λιστών. Συνήθως, σε μια τέτοια γλώσσα, ο κάθε κόμβος ορίζεται σαν μια εγγραφή με δύο πεδία, το ένα περιέχει το στοιχείο και το άλλο τον δείκτη.

Παράδειγμα 3.2[α]: *Ορισμός απλά συνδεδεμένης λίστας στην Pascal*

Στην Pascal, ο ορισμός μιας απλά συνδεδεμένης λίστας ως τύπου δεδομένων, μπορεί να γίνει ως εξής :

```
Type
   deiktiskombou= ^kombos;
   kombos= record
              stoixeio: typostoixeiou;
              deiktis: deiktiskombou
           end;
var
   l: deiktiskombou;
```

Το σύμβολο '^' χρησιμοποιείται στην Pascal για τη δήλωση δεικτών ακολουθούμενο από τον τύπο δεδομένων για στιγμιότυπα του οποίου θέλουμε να δημιουργήσουμε δείκτες / μεταβλητές δείκτη. Στις παραπάνω δηλώσεις, ορίζεται ένας

τύπος δείκτη με το όνομα 'deiktiskombou'. Μεταβλητές του τύπου αυτού δείχνουν σε δεδομένα τύπου 'kombos', δηλαδή του τύπου που ακολουθεί το σύμβολο '^' στην πρώτη από τις παραπάνω δηλώσεις. Στη δεύτερη δήλωση, δηλώνεται ο τύπος δεδομένων 'kombos' σαν μια εγγραφή (record) με δύο πεδία. Το πρώτο, 'stoixeio', αντιπροσωπεύει το στοιχείο ενός κόμβου και είναι κάποιου τύπου, έστω 'typostoixeiou'. Το δεύτερο πεδίο, 'deiktis', είναι τύπου 'deiktiskombou', δηλαδή ένας δείκτης σε ένα άλλο κόμβο. Τέλος, δηλώνεται μια μεταβλητή L τύπου 'deiktiskombou', που αντιπροσωπεύει τον δείκτη αρχή της λίστας. Με βάση το δείκτη αυτό, μπορούμε να προσπελάσουμε όλους τους κόμβους μιας συνδεδεμένης λίστας.

Παράδειγμα 3.2β: *Ορισμός απλά συνδεδεμένης λίστας στη C*

Στην C ο ορισμός μιας απλά συνδεδεμένης λίστας μπορεί να γίνει ως εξής:

```
typedef struct kombos *Deiktiskombou;
typedef struct kombos{
            typestoixeiou stoixeio;
           deiktiskombou; deiktis;
              }KOMBOS;
deiktiskombou; L;
```

Εδώ το '*' παίζει ένα ρόλο αντίστοιχο του '^' της Pascal. Έτσι, ορίζεται ένας τύπος δείκτη με το όνομα 'Deiktiskombou'. Μεταβλητές του δείκτη αυτού, όπως η L, δείχνουν σε δεδομένα τύπου 'KOMBOS', δηλαδή σε μια εγγραφή που περιγράφεται από τη δομή (struct) 'kombos'. Τα υπόλοιπα ισχύουν όπως στο Παράδειγμα 3.2α.

3.2 Πράξεις σε συνεχόμενη λίστα

3.2.1 Διαπέραση

Η διαπέραση (traversal) είναι μια από τις συνήθεις πράξεις στις δομές δεδομένων. Διαπέραση σε μια λίστα (ή έναν πίνακα) σημαίνει να «περάσουμε» από όλα τα στοιχεία της, το ένα μετά το άλλο, και να εφαρμόσουμε κάποιο είδος επεξεργασίας σ'αυτά. Αυτό συνήθως ονομάζεται και επίσκεψη (visiting). Το είδος της επεξεργασίας εξαρτάται από το συγκεκριμένο πρόβλημα. Π.χ., αύξηση της τιμής των στοιχείων κατά ένα.

ΑΛΓΟΡΙΘΜΟΣ 3.1: ΔΙΑΠΕΡΑΣΗ ΣΥΝΕΧΟΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος : ένας πίνακας (L) και το τρέχον μέγεθος της αντίστοιχης λίστας (T).
Έξοδος : Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται μεταβολές στις τιμές των στοιχείων της λίστας.

```
SL-DIAPERASH (L, T)
1   for k      1 to T                {Καθορισμός αριθμού επαναλήψεων}
2       PROCCESS (L[k])              {Εφαρμογή διαδικασίας}
    endfor
```

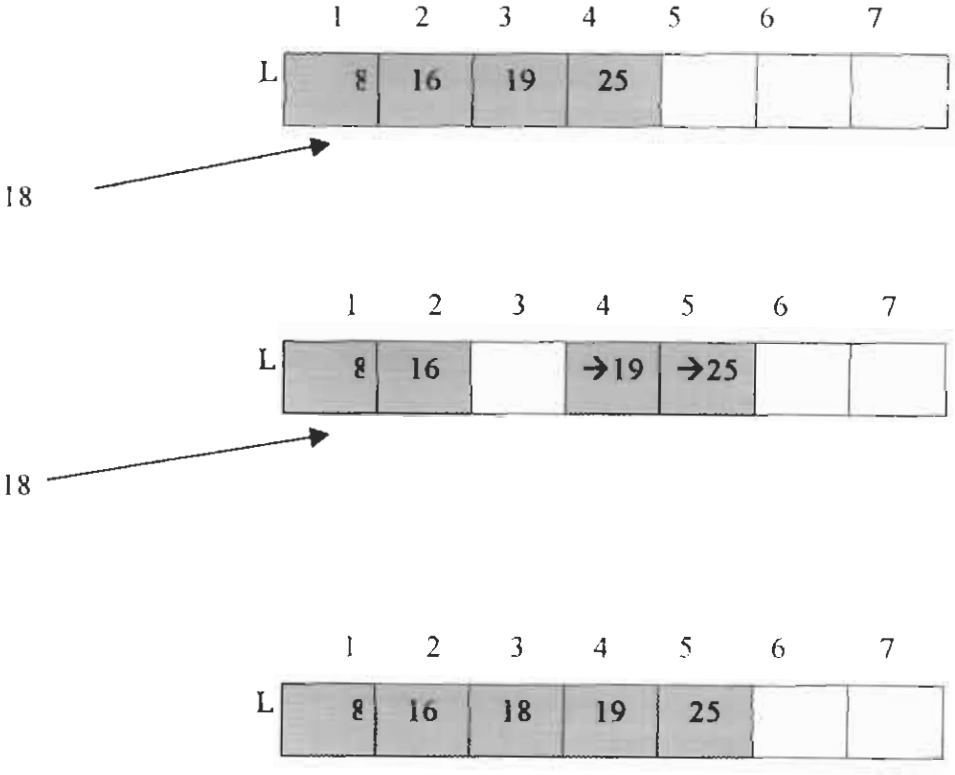
Ο αλγόριθμος για τη διαπέραση μιας συνεχόμενης λίστας παρουσιάζεται στο παραπάνω πλαίσιο, ως Αλγόριθμος 3.1 (SL-DIAPERASH).
Την επεξεργασία που υφίσταται κάθε στοιχείο αντιπροσωπεύει η διαδικασία PROCCESS, που δέχεται σαν παράμετρο το τρέχον στοιχείο. Η παράμετρος T είναι ο δείκτης τέλους της λίστας, δηλαδή αντιπροσωπεύει το τρέχον μέγεθος της λίστας. Ο Αλγόριθμος 3.1 μπορεί να εφαρμοστεί και για την διαπέραση ενός πίνακα, με τη μόνη διαφορά ότι το T τότε αντιπροσωπεύει το (σταθερό) μέγεθος του πίνακα. Η λογική του αλγόριθμου είναι πολύ απλή. Η διαδικασία PROCCESS μπορεί να έχει και άλλες παραμέτρους εκτός του τρέχοντος στοιχείου.

3.2.2 Εισαγωγή

Η εισαγωγή (insertion) είναι πράξη με την οποία εισάγουμε ένα νέο στοιχείο (τιμή) σε μια λίστα. Η εισαγωγή ενός στοιχείου στο τέλος μιας λίστας να μην υπερβεί το μέγεθος του πίνακα που την αναπαριστά. Η διαδικασία δυσκολεύει όταν θέλουμε να εισάγουμε ένα στοιχείο σε μια ενδιάμεση θέση μεταξύ των στοιχείων της λίστας. Τότε, πρέπει τα στοιχεία που βρίσκονται δεξιά της θέσης εισαγωγής να μετακινηθούν μια θέση δεξιότερα. Στο σχήμα 3 περιγράφονται οι φάσεις της διαδικασίας εισαγωγής ενός στοιχείου σε μια λίστα με τρέχον μέγεθος 4, που παριστάνεται μέσω ενός πίνακα μεγέθους 7.

Σχήμα 3:

(α) Πριν την εισαγωγή (β) Μετακίνηση στοιχείων (γ) Εισαγωγή στοιχείου



Αλγόριθμος 3.2: Εισαγωγή σε συνεχόμενη λίστα

Είσοδος : Ένας πίνακας (L), το μέγεθος του (N),ο δείκτης τέλους της αντίστοιχης λίστας (T), η θέση εισαγωγής (L) και η προς εισαγωγή τιμή (X).

Έξοδος : Εξωτερικά, επιστρέφει μήνυμα υπερχείλισης στην περίπτωση γεμάτου πίνακα, αλλιώς τίποτα. Εσωτερικά, γίνονται οι απαραίτητες μετακινήσεις, καταχώρηση της τιμής και ενημέρωση του δείκτη τέλους σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
SL-EISAGWGH (L, N, T, I, X)
1   ifT=N                    {Έλεγχος γεμάτης λίστας}
2    then print              {Μήνυμα υπερχείλισης}
3    else K ← T              {Αρχικοποίηση μετρητή}
4       while K≥ 1           {Έλεγχος τέλους επανάληψης}
5          L[K+1]  L[K]      {Μετακίνηση στοιχείου μια θέση}
6          K  ←  K-1         {Ενημέρωση μετρητή}
        endwhile
7       L[1]  ←  X           {Εισαγωγή στοιχείου}
8       T  ←  T+1            {Ενημέρωση δείκτη τέλους}
    endif
```

Ο Αλγόριθμος 3.2 (SL- EISAGWGH) είναι αυτός που υλοποιεί την εισαγωγή ενός στοιχείου σε μια συνεχόμενη λίστα. Η λογική του Αλγόριθμου 3.2 έχει ως εξής: Μετά από τον έλεγχο για το αν ο πίνακας είναι γεμάτος, οπότε επιστρέφει μήνυμα υπερχείλισης (βήματα 1-2), Αρχίζει ο κύριος αλγόριθμος. Γίνεται μετακίνηση κατά μια θέση δεξιά των στοιχείων της λίστας που βρίσκονται στη θέση I και δεξιά της , αρχίζοντας από το τέλος της λίστας (θέση T) και προχωρώντας προς τα αριστερά (μέχρι και τη θέση I). Αυτό επιτυγχάνεται με τη διάταξη while (βήματα 4-6) με κατάλληλη αρχικοποίηση του μετρητή K (βήμα 3). Στη συνέχεια, γίνεται καταχώρηση του στοιχείου X στην κενή θέση I του πίνακα (βήμα 7) και αυξάνεται ο

δείκτης τέλους της λίστας κατά ένα (βήμα 8), αφού το μέγεθος της λίστας αυξήθηκε κατά ένα, λόγω της εισαγωγής του στοιχείου.

3.2.3 Διαγραφή

Η διαγραφή (deletion) ενός στοιχείου από μια λίστα πραγματοποιείται με τη μετακίνηση των στοιχείων της λίστας που είναι στα δεξιά του, κατά μια θέση αριστερά. Ο αντίστοιχος αλγόριθμος είναι ο Αλγόριθμος 3.3 (SL-DIAGRAFH).

Αλγόριθμος 3.3: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), ο δείκτης τέλους της αντίστοιχης λίστας (T) και η θέση του προς διαγραφή στοιχείου (I).

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται οι απαραίτητες μετακινήσεις και ενημερώνεται ο δείκτης τέλους της λίστας.

```
SL-DIAGRAFH (L, T, I)
1    for K = I to T-1                {Καθορισμός ορίων επανάληψης}
2       L[K] ← L[K+1]                {Μετακίνηση στοιχείου μια θέση αριστερά}
     endfor
3    T ← T-1                         {Ενημέρωση του δείκτη τέλους της
λίστας}
```

Η λογική του αλγόριθμου αυτού έχει ως εξής: Γίνεται μετακίνηση των στοιχείων που βρίσκονται δεξιά της θέσης I του προς διαγραφή στοιχείου κατά μια θέση αριστερά, αρχίζοντας από το πρώτο δεξιά στοιχείο, με τη διάταξη for (βήματα 1-2). Στη συνέχεια, ελαττώνεται ο δείκτης τέλους της λίστας Τα κατά ένα (βήμα 3), αφού το μέγεθος της λίστας ελαττώθηκε.

3.2.4 Αναζήτηση

Οι μέθοδοι γραμμικής και δυαδικής αναζήτησης σε πίνακα εφαρμόζεται με τον ίδιο ακριβώς τρόπο και σε συνεχόμενη λίστα. Οι αντίστοιχοι αλγόριθμοι (2.1 και 2.2) παραμένουν οι ίδιοι με της εξής μετονομασίες παραμέτρων μόνο: L αντί A και T αντί N.

Παράδειγμα 3.3: Αλγόριθμος Αναζήτησης και Διαγραφής Στοιχείου

Θεωρούμε μια συνεχόμενη λίστα και μια τιμή. Το ζητούμενο είναι να σχεδιάσουμε ένα αλγόριθμο που να βρίσκει αν υπάρχει στοιχείο με αυτή την τιμή στη λίστα και, αν υπάρχει, να το διαγράφει.

ΑΛΓΟΡΙΘΜΟΣ Π3.3: ΑΝΑΖΗΤΗΣΗ-ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), ο δείκτης τέλους της αντίστοιχης λίστας (T) και η τιμή του προς διαγραφή στοιχείου (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση που το X δεν υπάρχει, αλλιώς τίποτα. Εσωτερικά, διαγράφεται το στοιχείο, γίνονται οι απαραίτητες μετακινήσεις και ενημερώνεται ο δείκτης τέλους σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
SL-ANAZHTHSH-DIAGRAFH (L, T, X)
1    I ← 1                          {Αρχικοποίηση μεταβλητών}
2    while (I≤T) and (L[I] ≠ X)     {Έλεγχος τέλους επανάληψης}
3    I ← I+1                        {Ενημέρωση δείκτη θέσης}
     endwhile
3      If L[I] = X                  {Αναγνώριση αποτελέσματος}
4        Then for K= I to T-1       {Καθορισμός ορίων επανάληψης}
5                L[K] ← L[K+1       {Μετακίνηση μια θέση αριστερά}
               Endfor
7              T ← T – 1            {Ενημέρωση δείκτη τέλους}
         else print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'
8        endif                      {Μήνυμα λάθους}
```

Ο ζητούμενος αλγόριθμος, που είναι ένας συνδυασμός των αλγορίθμων αναζήτησης (Αλγόριθμος 2.1) και διαγραφής (Αλγόριθμος 3.3), παρουσιάζεται στο παραπάνω πλαίσιο ως Αλγόριθμος Π3.3. Η λογική του αλγόριθμου είναι η ακόλουθη: Κατ' αρχήν, αναζητούμε το στοιχείο με τη δοθείσα τιμή (βήματα 1-3) και, όταν το

βρούμε (βήμα 4), το διαγράφουμε (βήματα 5-7), αλλιώς επιστρέφουμε ένα μήνυμα αποτυχίας (βήμα 8)

3.3 Πράξεις σε απλά συνδεδεμένη λίστα

Κάθε γλώσσα προγραμματισμού που παρέχει δείκτες, διαθέτει και αντίστοιχες διαδικασίες δημιουργίας και απαλοιφής δεικτών. Π.χ., η Pascal διαθέτει τις διαδικασίες 'new(p)' και 'dispose(p)' για τη δημιουργία και απαλοιφή αντίστοιχα, ενός κόμβου στον οποίο «δείχνει» ο δείκτης 'p'. Στους αλγόριθμους που σχετίζονται με τη συνδεδεμένη αναπαράσταση, δε μας απασχολούν τέτοια θέματα. Θεωρούμε ότι οι κόμβοι έχουν ήδη με κάποιο τρόπο παραχθεί και στη διαγραφή τους δεν μας απασχολεί θέμα απελευθέρωσης μνήμης.

Επίσης, χρησιμοποιούμε τη βοηθητική συνάρτηση KOMBOS. Η KOMBOS(P), όπου P δείκτης, επιστρέφει τον κόμβο στον οποίο «δείχνει ο δείκτης . Ακόμη, τα STOIXEIO (KOMBOS(P)), DEIKTHS (KOMBOS(P)) ενεργούν ως τρόποι (μεταβλητές) προσπέλασης των τμημάτων του κόμβου KOMBOS(P).

Τέλος, πρέπει να διευκρινίσουμε τις διαφορές στις έννοιες των εκφράσεων «κόμβος του δείκτη P_i», «δείκτης του κόμβου N_i» και δείκτης στον κόμβο N_i», που χρησιμοποιούνται στη συνέχεια. Αναφερόμενοι στο Σχήμα 3.2, ο «κόμβος του P_1», που εκφράζεται ως KOMBOS(P1), είναι ο κόμβος N_2, δηλαδή ο κόμβος στον οποίο «δείχνει» ο δείκτης P_1. Ο «δείκτης του κόμβου N_2», που εκφράζεται ως DEIKTHS (KOMBOS(P1)),είναι ο P_2, δηλαδή ο «δείκτης στον κόμβο N_3».

3.3.1 Διαπέραση

Ο αλγόριθμος για τη διαπέραση μιας απλά συνδεδεμένης λίστας παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.4 (DL-DIAPERASH). Η μεταβλητή L είναι ο δείκτης αρχής της λίστας, δηλαδή περιέχει τη διεύθυνση του πρώτου κόμβου της λίστας, ενώ η δεύτερη μεταβλητή P είναι μια βοηθητική (τοπική) μεταβλητή δείκτη, που αντιπροσωπεύει τη διεύθυνση του τρέχοντος κόμβου. Επίσης, χρησιμοποιείται πάλι η διαδικασία PROCESS.

Η λογική του Αλγόριθμου 3.4 έχει ως ακολούθως: Κατ' αρχήν, αρχικοποιούμε τη βοηθητική (τοπική) μεταβλητή P, δίνοντας της σαν τιμή τη διεύθυνση του πρώτου κόμβου της λίστας. Στη συνέχεια, ξεκινά μια επαναληπτική διαδικασία που έχει δυο βήματα στο σώμα της. Στο πρώτο (βήμα 3), εφαρμόζεται η διαδικασία PROCESS στο στοιχείο του τρέχοντος κόμβου. Στο δεύτερο (βήμα 4), ενημερώνεται ο δείκτης P ώστε να δείχνει τον επόμενο κόμβο της λίστας. Η εκτέλεση των βημάτων αυτών τερματίζεται όταν ο δείκτης P πάρει την τιμή NIL,που σημαίνει ότι δεν υπάρχει επόμενος κόμβος.

ΑΛΓΟΡΙΘΜΟΣ 3.4: ΔΙΑΠΕΡΑΣΗ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος: Ο δείκτης αρχής (L) μιας συνδεδεμένης λίστας.

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται μεταβολές στις τιμές των στοιχείων της λίστας.

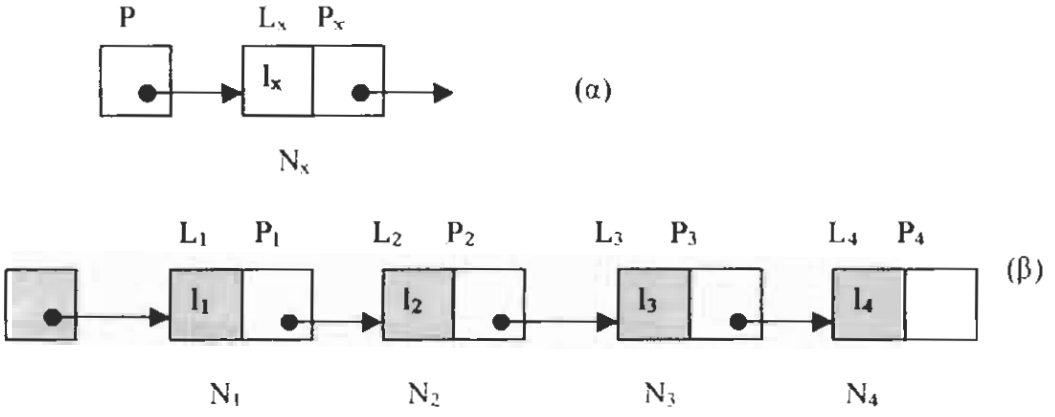
```
DL-DIAPERASH(L)
1  P ← L                                  {Αρχικοποίηση δείκτη}
2  while P ≠ NIL                          {Συνθήκη τερματισμού}
3     ROCESS (STOIXEIO(KOMBOS(P)))        {Εφαρμογή διαδικασίας}

4     P← DEIKTHS(KOMBOS(P))               {Ενημέρωση τρέχοντος δείκτη}
   endwhile
```

3.3.2 Εισαγωγή

Η εισαγωγή σε μια συνδεδεμένη λίστα σημαίνει την παρεμβολή ενός κόμβου σε κάποια θέση μεταξύ δύο ήδη υπαρχόντων κόμβων της λίστας. Η αρχική κατάσταση της διαδικασίας αυτής απεικονίζεται στο σχήμα 4, ενώ η τελική στο σχήμα 5.

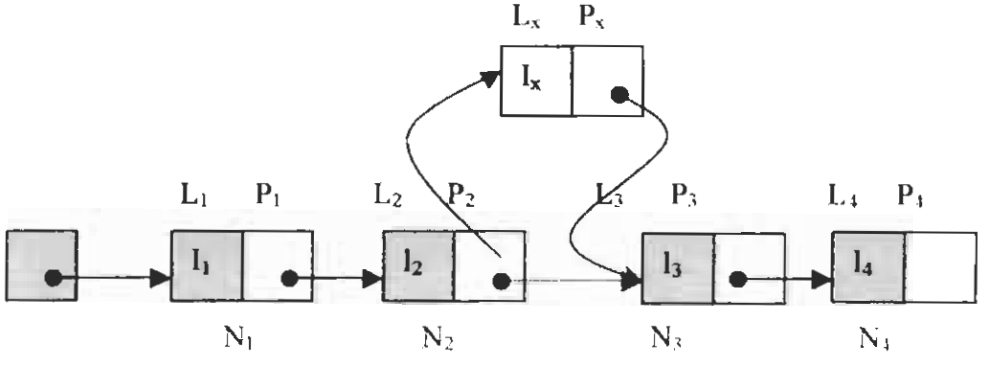
Σχήμα 4: Κατάσταση πριν την εισαγωγή κόμβου σε συνδεδεμένη λίστα:

(α) προς εισαγωγή κόμβος (β) αρχική συνδεδεμένη λίστα.



Ο αλγόριθμος που πραγματοποιεί την εισαγωγή αυτή, παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.5 (DL-EISAGWGH-META). Δέχεται δύο παραμέτρους εισόδου, τον δείκτη στον προς εισαγωγή κόμβο (P)και τον δείκτη στον κόμβο μετά τον οποίο θα γίνει η εισαγωγή (PI). Μετά τον έλεγχο για την ύπαρξη των δύο δεικτών και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), αρχίζει ο κυρίως αλγόριθμος που αποτελείται από δύο βήματα. Στο βήμα 3, ο δείκτης του (προς εισαγωγή) κόμβου P γίνεται δείκτης στον κόμβο που είναι μετά τον κόμβο PI. Με το σχήμα 4, για το οποίο P=P και PI=P1, ο δείκτης (του κόμβου Nx) Px παίρνει τη τιμή του P2, δηλαδή δείχνει στον κόμβο N3. Στο βήμα 4, ο δείκτης του κόμβου PI παίρνει τη τιμή του δείκτη στον προς εισαγωγή κόμβο ενώ ο δείκτης P2 παίρνει την τιμή του P.

Σχήμα 5: Κατάσταση μετά την εισαγωγή του κόμβου σε συνδεδεμένη λίστα.



Τα βήματα 3 και 4 του αλγόριθμου παριστάνονται διαγραμματικά, για τα δεδομένα των σχημάτων 4 και 5, στο Σχήμα.6

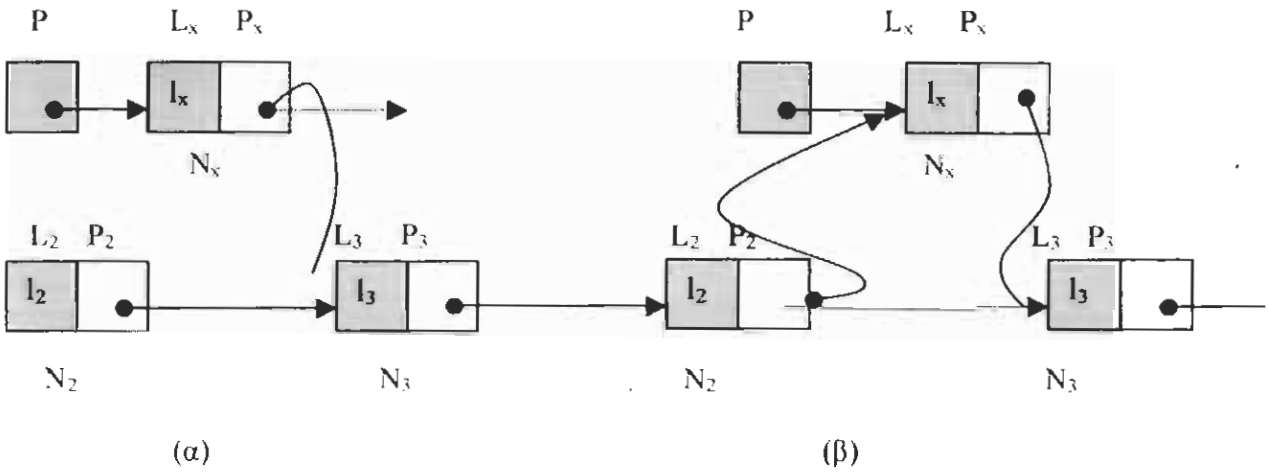
ΑΛΓΟΡΙΘΜΟΣ 3.5: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης στον προς εισαγωγή κόμβο (P) και ο δείκτης στον κόμβο της λίστας μετά τον οποίο θα γίνει η εισαγωγή του νέου κόμβου (PI).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση μη έγκυρων δεικτών, αλλιώς τίποτα. Εσωτερικά, γίνονται μεταβολές στη λίστα σε περίπτωση επιτυχούς εισαγωγής, αλλιώς τίποτα.

```
DL-EISAGWGH-META (P, PI)
1   if(P= NIL) or (PI = NIL)                    {Έλεγχος δεικτών}
2   then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'              {Μήνυμα λάθους}
3   else DEIKTHS(KOMBOS(P))← DEIKTHS(KOMBOS(PI)) {ενημέρωση
    δείκτη}
4       DEIKTHS(KOMBOS(PI))← P                  {Ενημέρωση δείκτη}
    Endif
```

Σχήμα 6: Διαγραμματική αναπαράσταση (α) βήμα 3 (β) βήμα 4 του Αλγόριθμου 3.5



3.3.3 Διαγραφή

Η διαγραφή ενός κόμβου από μια συνδεδεμένη λίστα απεικονίζεται διαγραμματικά στο Σχήμα 7. Ο αντίστοιχος αλγόριθμος παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.6 (DL-DIAGRAFH-META). Δέχεται μια παράμετρο (PI) που αντιπροσωπεύει την τιμή του δείκτη που «δείχνει στον προηγούμενο, του προς διαγραφή, κόμβο (π.χ. την τιμή του P_1 στο Σχήμα 7, που δείχνει N_2). Η παράμετρος αυτή, δεν αντιπροσωπεύει τον δείκτη που «δείχνει» στον προς διαγραφή αλλά στον προηγούμενο κόμβο, διότι, όπως φαίνεται από το Σχήμα 7, απαιτείται ο δείκτης του προηγούμενου κόμβου να αλλάξει τιμή και να «δείχνει» στον επόμενο, του προς διαγραφή, κόμβο. Αν μας δίνονταν μόνο η τιμή του δείκτη στον προς διαγραφή κόμβο, δε θα ήταν δυνατό να προσπελάσουμε το δείκτη του προηγούμενου κόμβου, ενώ το αντίστροφο είναι δυνατό.

ΑΛΓΟΡΙΘΜΟΣ 3.6:ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης (PI) στον προηγούμενο, του προς διαγραφή, κόμβο.

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση μη έγκυρων δεικτών, αλλιώς τίποτα. Εσωτερικά, γίνονται μεταβολές στη λίστα σε περίπτωση επιτυχούς διαγραφής, αλλιώς τίποτα.

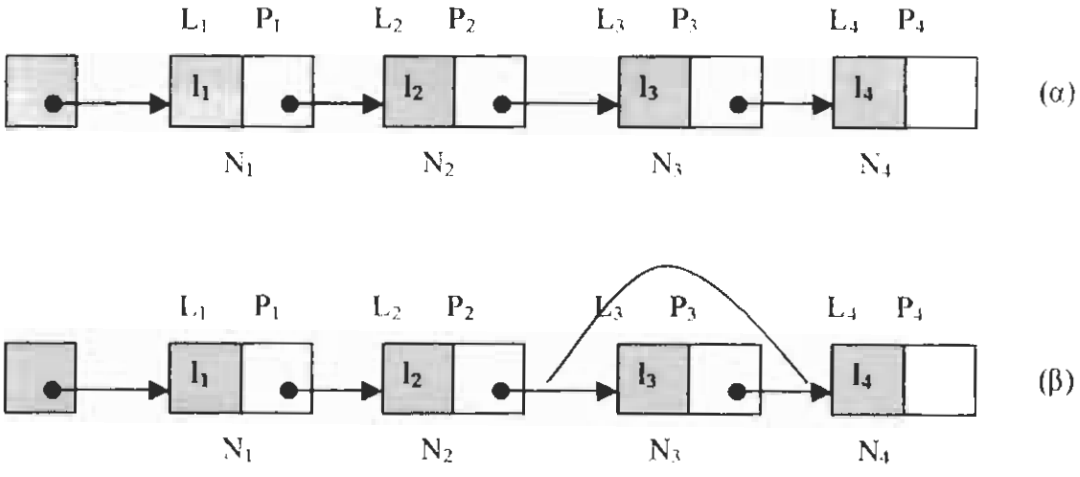
```
DL-DIAGRAFH(PI)
1  if (PI=NIL) or (DEIKTHS(KOMBOS(PI))=NIL) {Έλεγχος δεικτών}
2  then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'          {Μήνυμα λάθους}
3  else P← DEIKTHS(KOMBOS(PI))              {Ενημέρωση δεικτών}
4  DEIKTHS(KOMBOS(PI))←DEIKTHS(KOMBOS(P))
   endif
```

Στο βήμα 1 γίνεται έλεγχος ύπαρξης του προηγούμενου (PI = NIL) και του προς διαγραφή (DEIKTHS(KOMBOS(PI)) = NIL) κόμβου. Αν κάποιος από αυτούς δεν υπάρχει, τότε επιστρέφει το αντίστοιχο μήνυμα. Αλλιώς, προχωρά στον κυρίως αλγόριθμο που είναι τα βήματα 3 και 4. Στο βήμα 3 κάνουμε την τιμή μιας βοηθητικής μεταβλητής δείκτη P ίση με την τιμή του δείκτη στον υπό διαγραφή κόμβο (ή, με άλλα λόγια, του δείκτη του προηγούμενού του υπό διαγραφή κόμβου), δηλαδή αποθηκεύουμε

την τρέχουσα τιμή του «δείκτη στον υπό διαγραφή κόμβο». Σε αναφορά με το Σχήμα 7, η P παίρνει την τιμή του P_2. Στο βήμα 4 ενημερώνουμε το δείκτη του προηγούμενού του υπό διαγραφή του κόμβου (δηλαδή του κόμβου N_2 στο Σχήμα 7) με την τιμή του δείκτη του προς διαγραφή κόμβου (ή με άλλα λόγια του δείκτη στον επόμενο του προς διαγραφή κόμβου) ώστε να «δείχνει» στον επόμενο του προς διαγραφή κόμβου. Δηλαδή, ο P_2 παίρνει τη τιμή του P_3, ώστε να «δείχνει» στον N_4.

Μια ειδική περίπτωση διαγραφής, είναι εκείνη όπου θέλουμε να διαγράψουμε την κεφαλή μιας λίστας. Στην περίπτωση αυτή, ο Αλγόριθμος 3.6 μετατρέπεται ως εξής: α) τη θέση του PI παίρνει ο L, β) ο έλεγχος στο βήμα 1 περιλαμβάνει μόνο το L και γ) τα βήματα 3,4 συνοψίζονται στο L ← DEIKTHS(KOMBOS(L)).

Σχήμα 7: (α) Κατάσταση πριν, (β) κατάσταση μετά τη διαγραφή κόμβου



Παράδειγμα 3.4: Διαγραφή Κ-οστού κόμβου συνδεδεμένης λίστας

Δίνεται μια απλά συνδεδεμένη λίστα και ένας αριθμός Κ. Ζητείται να σχεδιαστεί αλγόριθμος που να διαγράφει το Κ-οστό κόμβο της λίστας.

Ενώ σε μια συνεχόμενη λίστα η πρόσβαση σ'ένα στοιχείο που δίνεται η σειρά του είναι εύκολη, λόγω της άμεσης προσπέλασης που προσφέρει η αναπαράσταση μέσω πίνακα, σε μια συνδεδεμένη λίστα δεν είναι. Για να γίνει αυτό, πρέπει να γίνει μια διαπέραση της λίστας κατά την οποία, αντί επεξεργασίας, θα γίνεται καταμέτρηση των κόμβων μέχρι τη θέση Κ. Αυτό γίνεται στο πρώτο τμήμα του Αλγόριθμου Π3.4 (DL-DIAGRAFH-K), που είναι ο ζητούμενος και παρουσιάζεται στο παρακάτω πλαίσιο.

Το πρώτο αυτό τμήμα αποτελείται βασικά από τη διάταξη επανάληψης while (βήματα 2-5). Το βήμα 1 δίνει αρχικές τιμές σε δύο τοπικές μεταβλητές, την PK και τη I. Η πρώτη, αντιπροσωπεύει το δείκτη του Κ-οστού κόμβου, ενώ η δεύτερη, τη θέση του κόμβου. Μέσα στη διάταξη while χρησιμοποιείται και η μεταβλητή PI που αντιπροσωπεύει το δείκτη στον προηγούμενο, του τρέχοντος, κόμβο. Θυμηθείτε ότι, η διαγραφή ενός στοιχείου από συνδεδεμένη λίστα απαιτεί και αυτόν το δείκτη (Αλγόριθμος 3.6).

Η επανάληψη του σώματος του while σταματά όταν φθάσουμε είτε στον Κ-οστό είτε στο τέλος της λίστας (P= NIL). Στην πρώτη περίπτωση, γίνεται διαγραφή του κόμβου, ενώ στη δεύτερη, επιστρέφει ένα μήνυμα λάθους. Αυτό υλοποιείται από το δεύτερο τμήμα του αλγόριθμου, που είναι μια διάταξη if (βήματα 6-8)

ΑΛΓΟΡΙΘΜΟΣ Π3.4:ΔΙΑΓΡΑΦΗ Κ-ΟΣΤΟΥ ΚΟΜΒΟΥ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος: Ο δείκτης αρχής μιας απλά συνδεδεμένης λίστας (L) και ένας ακέραιος αριθμός (K).

Έξοδος: Εξωτερικά, επιστρέφει ένα μήνυμα λάθους σε περίπτωση ανυπαρξίας του K-οστού κόμβου, αλλιώς τίποτα. Εσωτερικά, γίνεται διαγραφή κόμβου σε περίπτωση επιτυχίας.

```
DL-DIAGRAFH-K(L, K)
1   PK← L, I← 1                                  {Αρχικοποίηση μεταβλητών}
2   While (PK≠NIL) and (I≤K)                     {Έλεγχος τέλους επανάληψης}
3       PI← P                                    {Ενημέρωση δείκτη}
4       PK←DEIKTHS(KOMBOS(PK))                   {Ενημέρωση δείκτη}
5       I←I+1                                    {Ενημέρωση μετρητή}
    Endwhile
6   if(PK≠NIL)                                   {Έλεγχος αποτελέσματος}
7     then DEIKTHS(KOMBOS(PI)) ← DEIKTHS(KOMBOS(PK))   {Επιτυχία}
8     Else print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'            {Αποτυχία}
    endif
```

3.3.4 Αναζήτηση

Για την αναζήτηση ενός στοιχείου σε μια συνδεδεμένη λίστα μπορεί να εφαρμοστεί μόνο η μέθοδος της γραμμικής αναζήτησης. Η δυαδική αναζήτηση δεν μπορεί να εφαρμοστεί, διότι δεν υπάρχει απ' ευθείας τρόπος να υπολογιστεί ο δείκτης του 'μεσαίου' στοιχείου. Επομένως , είτε η λίστα είναι διατεταγμένη είτε όχι, χρησιμοποιούμε τη γραμμική αναζήτηση, ψάχνουμε δηλαδή τα στοιχεία της λίστας ένα-ένα. Βέβαια, όταν η λίστα είναι διατεταγμένη, έχουμε διαφορετική συνθήκη τερματισμού του αλγόριθμου.

***Παράδειγμα 3.5**: Εύρεση Μηδενικών Στοιχείων σε Απλά Συνδεδεμένη Λίστα*

Θεωρούμε μια απλά συνδεδεμένη λίστα στην οποία αποθηκεύουμε αριθμούς. Το ζητούμενο είναι να σχεδιάσουμε έναν αλγόριθμο που να μετρά τον αριθμό των στοιχείων με μηδενική τιμή.

Ο ζητούμενος αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως αλγόριθμος Π3.5 (DL-EURESH-MHDEN). Θα μπορούσε να θεωρηθεί σαν μια τροποποίηση του αλγόριθμου διαπέρασης (Αλγόριθμος 3.4). Η βασική ιδέα είναι ότι περνάμε από όλους τους κόμβους της λίστας, μέσω μιας επαναληπτικής διάταξης while (βήματα 2-4), και σε κάθε κόμβο εξετάζουμε αν η τιμή του στοιχείου του είναι μηδενική (βήμα 3). Αν είναι, καταγράφουμε τη διαπίστωση με την αύξηση της τιμής του μετρητή I (βήμα 4). Αν όχι, προχωρούμε στον επόμενο κόμβο (βήμα 5). Στο τέλος, επιστρέφουμε την τιμή του μετρητή που αντιπροσωπεύει τον αριθμό των μηδενικών στοιχείων της λίστας (βήμα 6).

ΑΛΓΟΡΙΘΜΟΣ Π3.5:ΕΥΡΕΣΗ ΜΗΔΕΝΙΚΩΝ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης αρχής (L) μιας λίστας.

Έξοδος: Εξωτερικά, επιστρέφει τον αριθμό των μηδενικών στοιχείων της λίστας. Εσωτερικά, δεν γίνεται τίποτα.

```
DL-EURESH-MHDEN(L)
1   P ← L, I ← 0                          {Αρχικοποίηση μεταβλητών}
2   while P ≠ NIL                         {Συνθήκη επανάληψης}
3     if STIXEIO(KOMBOS(P))=0             {Έλεγχος μηδενικού στοιχείου}
4       then I ← I+1                      {Ενημέρωση μετρητή}
      endif
5     P ← DEIKTHS(KOMBOS(P))              {Ενημέρωση τρέχοντος δείκτη}
    Endwhile

6   print I                               {Επιστροφή τιμής μετρητή}
```

3.4 Εφαρμογή Προγράμματος σε Γλώσσα Pascal

```
program list1;     {ΟΛΟΚΛΗΡΩΜΕΝΟ ΠΡΟΓΡΑΜΜΑ ΓΙΑ ΛΙΣΤΕΣ}

uses wincrt;

type
list=^node;

node=record
    next:list;
    value:integer;
    end;

var
p,q,w,listbase:list;
ans2,ans3:char;
j:integer;

procedure listcreate;  {Η διαδικασία αυτή καλείται για τη δημιουργία της λίστας}
var
ans:char;
i,num:integer;
begin
 listbase:=nil;
 ans:='y';
 i:=1;  {Το i είναι βοηθητική μεταβλητή για την εισαγωγή των αριθμών}
 j:=0   {Το j μετράει τους κόμβους της λίστας}.
 while ans='y' do
     begin
      writeln ('Δώσε την ',i,' τιμή ');
      readln (num);
```

```
new(p); {Η εντολή new δημιουργεί ένα νέο κόμβο και κάνει το δείκτη
p να δείχνει σε αυτόν. Ο κόμβος είναι αρχικά κενός, κατάλληλα όμως
προετοιμασμένος να δεκτεί δείκτη και αριθμό.}
p^.value:=num; {Στο πεδίο value του κόμβου αυτού (το οποίο μπορεί να
προσπελαστεί ως πεδίο εγγραφής με την εντολή p^.value) θέτουμε ως περιεχόμενο
την τιμή που διαβάσαμε στο num δηλαδή 1 αριθμό}
p^.next:=listbase; {Στο πεδίο next του κόμβου αυτού (το οποίο μπορεί να
προσπελαστεί ως πεδίο εγγραφής με την εντολή p^.next) θέτουμε ως περιεχόμενο τη
διεύθυνση που δείχνει ο δείκτης listbase. Αρχικά ο listbase δεν δείχνει πουθενά, αλλά
στη συνέχεια δείχνει πάντα στον αρχικό κόμβο της λίστας.}
listbase:=p; {Ο δείκτης listbase δείχνει όπου και ο δείκτης p δηλαδή στον
καινούριο κόμβο της λίστας που μόλις δημιουργήσαμε.}
j:=j+1;
writeln ('Θέλεις να συνεχίσεις y n? ');
readln (ans);
i:=i+1; {αυξάνεται ο μετρητής των κόμβων}
end;
end; {Με τον τρόπο αυτό η λίστα δημιουργείται αντίστροφα από τη σειρά με την οποία
εισάγουμε τους αριθμούς. Ο δείκτης listbase δείχνει τελικά την αρχή της λίστας δηλαδή
πάντα τον 1ο κόμβο που περιέχει τον τελευταίο εισαχθέντα αριθμό}

procedure printlist; {Η διαδικασία αυτή καλείται για την εκτύπωση της λίστας}
begin
p:=listbase; {Για να εκτυπώσω μια λίστα πρέπει να ξεκινήσω από την αρχή της. Βάζω
στο δείκτη p τη διεύθυνση του δείκτη listabase ο οποίος όπως αναφέραμε δείχνει πάντα
στον 1ο κόμβο της λίστας, άρα και ο p δείχνει στην αρχή της λίστας}
writeln ('Η λίστα είναι:');
while p<>nil do {Για όσο χρόνο δεν έχω φτάσει στο τέλος της λίστας επανέλαβε}
begin
write(p^.value,'->'); {τυπώνουμε τον αριθμό του κάθε κόμβου και μετά το - >}
```

```
      p:=p^.next;    {Με τον τρόπο αυτό κάνουμε το δείκτη p να δείχνει κάθε φορά στον επόμενο κόμβο της λίστας}
    end;
 writeln;
end;

procedure append;    {Η διαδικασία αυτή καλείται για την προσθήκη νέων κόμβων στην αρχή της λίστας}
var
ans:char;
i,num:integer;
begin
 p:=listbase;  {Βάζω στο δείκτη p τη διεύθυνση του δείκτη listabase ο οποίος όπως αναφέραμε δείχνει πάντα στον 1ο κόμβο της λίστας, άρα και ο p δείχνει στην αρχή της λίστας}
 ans:='y';

 while ans='y' do
    begin
     writeln   ('Δώσε την ',j,' τιμή '); {Το j φυλάει το συνολικό αριθμό κόμβων όλης της λίστας π.χ. αν η αρχή λίστα είχε 3 κόμβους τώρα θα έγραφε 'Δώσε την 4 τιμή'}
     readln(num);
     new(p);
     p^.value:=num;
     p^.next:=listbase;   {Βάζω το δείκτη του νέου κόμβου που δημιούργησα να δείχνει όπου και ο listbase δηλαδή στην αρχή της λίστας έτσι ώστε ο νέος κόμβος να γίνει αυτός η αρχή της λίστας}
     listbase:=p;         {Βάζω το δείκτη listbase να δείχνει τον νεοεισαχθέντα κόμβο. Συνεπώς η λίστα αρχίζει από τον κόμβο αυτό. Ο listbase δείχνει πάντα στην αρχή της λίστας}
     writeln   ('Θέλεις να συνεχίσεις y/n? ');
```

```
      readln(ans);
      j:=j+1;
     end;
end;

procedure append2;   {Η διαδικασία αυτή καλείται για την προσθήκη νέων κόμβων στο τέλος της λίστας}
var
num:integer;
begin
 p:=listbase;  {Βάζω στο δείκτη p τη διεύθυνση του δείκτη listabase ο οποίος όπως αναφέραμε δείχνει πάντα στον 1ο κόμβο της λίστας, άρα και ο p δείχνει στην αρχή της λίστας}
 while p^.next<> nil do  {Για όσο χρόνο δεν έχω φτάσει στο τέλος της λίστας επανέλαβε}
     p:=p^.next;     {Με τον τρόπο αυτό κάνουμε το δείκτη p να δείχνει κάθε φορά στον επόμενο κόμβο της λίστας μέχρι να φτάσουμε στο τέλος της}
 writeln   ('Δώσε τη νέα τιμή'),
 readln(num);
 new(q);        {Επειδή χρειάζομαι το δείκτη p να ξακολουθεί να δείχνει στο τέλος της λίστας, δημιουργώ το νέο κόμβο και κάνω ένα νέο δείκτη, τον q, να δείχνει σε αυτόν}
 q^.value:=num;    {Θέτω τιμή στον νέο κόμβο}
 p^.next:=q;      {Κάνω τον τελευταίο κόμβο της υπάρχουσας λίστας να δείχνει στον νεοεισαχθέντα}
 q^.next:=nil;       {Για να καθορίσουμε το τέλος της νέας λίστας βάζω το δείκτη του τελευταίου κόμβου να δείχνει όπως πάντα στο nil}
 j:=j+1;
end;

procedure sort;   {Η διαδικασία αυτή καλείται για την ταξινόμηση των κόμβων της λίστας}
var
```

```
n,temp,k,i:integer;
begin
 for k:=1 to j-1 do      {Το k είναι ο μετρητής βημάτων (περασμάτων) όταν η λίστα έχει συνολικά j στοιχεία}
    begin
     p:=listbase;    {Βάζω στο δείκτη p τη διεύθυνση του δείκτη listabase ο οποίος όπως αναφέραμε δείχνει πάντα στον 1ο κόμβο της λίστας, άρα και ο p δείχνει στην αρχή της λίστας}
     q:=p^.next;
     for i:=1 to j-k do    {Σε κάθε βήμα ταξινομείται 1 κόμβος της λίστας, άρα σε κάθε επόμενο βήμα θα γίνεται 1 σύγκριση λιγότερη}
       begin
        if p^.value>q^.value then
          begin
           temp:=p^.value;
           p^.value:=q^.value;
           q^.value:=temp;
          end;
        p:=p^.next;
        q:=q^.next;
       end;
    end;
end;

procedure delete;      {Η διαδικασία αυτή καλείται για την διαγραφή ενός κόμβου από τη λίστα}
var
num:integer;
f:boolean;
begin
 writeln      ('Δώσε το περιεχόμενο (αριθμό) του κόμβου που θέλεις να διαγράψεις');
```

```
readln(num);
p:=listbase;    {Βάζω το δείκτη p να δείχνει στην αρχή της λίστας}
q:=p^.next;  {Βάζω το δείκτη q να δείχνει όπου και ο p^.next δηλαδή στο δεύτερο κόμβο της λίστας}
f:=false;    {Η μεταβλητή αυτή ελέγχει την ύπαρξη του κόμβου που θέλουμε να διαγράψουμε. Υποθέτουμε αρχικά ότι ο κόμβος που θέλουμε να διαγράψουμε δεν υπάρχει}
if p^.value=num then    {Δηλαδή αν ο αριθμός που θέλουμε να διαγράψουμε υπάρχει στον τρέχοντα κόμβο που δείχνει το p}
  begin
   listbase:=p^.next;    {Βάζω το δείκτη listbase να δείχνει όπου και ο p^.next δηλαδή στο δεύτερο κόμβο της λίστας}
   dispose(p);    {Είναι η αντίθετη εντολή της new με την οποία απελευθερώνουμε τη μνήμη που δεσμεύει ο κόμβος που διαγράφουμε. Συγκεκριμένα η εντολή αυτή διαγράφει τον κόμβο που δείχνει ο p δηλαδή τον 1ο κόμβο}
   writeln    ('Το στοιχείο βρέθηκε και διαγράφηκε!');
  end
else    {δηλαδή αν η τιμή που πληκτρολογήσαμε δεν υπάρχει στον 1ο κόμβο}
   begin
    while (q<>nil) and (f=false) do    {δηλαδή όσο ο δείκτης q δεν έχει φτάσει στο τέλος της λίστας και η τιμή που θέλουμε να διαγράψουμε δεν έχει ακόμα βρεθεί}
       if q^.value<>num then    {αν η τιμή του κόμβου που δείχνει ο δείκτης q είναι διάφορη από τον αριθμό που πληκτρολογήσαμε τότε βάζουμε και τους 2 δείκτες, τον p και τον q να προχωρήσουν ταυτόχρονα στην επόμενη θέση}
         begin
          p:=p^.next;    {Βάλε το δείκτη p να δείχνει στον επόμενο κόμβο της λίστας}
          q:=q^.next;    {Βάλε το δείκτη q να δείχνει στον επόμενο κόμβο του p}
         end
       else    {δηλαδή αν το q^.value = num δηλαδή αν η τιμή βρεθεί στη λίστα}
         begin
          f:=true;    {Η μεταβλητή f παίρνει την τιμή true ως ένδειξη ότι το στοιχείο που θέλουμε να διαγράψουμε υπάρχει στη λίστα}
```

```
            p^.next:=q^.next;    {Βάζουμε το δείκτη του προηγούμενου κόμβου από
            αυτόν που βρέθηκε το στοιχείο να δείχνει στον μεθεπόμενο κόμβο. Έτσι
            ουσιαστικά παρακάμπτουμε τον κόμβο που έχει το στοιχείο}
            dispose(q);    {διαγραφή του κόμβου που περιέχει το στοιχείο}
          end;
 if f=true then
   writeln    ('το στοιχείο βρέθηκε και διαγράφηκε!')
 else
   writeln    ('το στοιχείο δεν βρέθηκε και συνεπώς δεν διαγράφηκε!');
 end;
end;

procedure search;    {Η διαδικασία αυτή καλείται για την αναζήτηση ενός κόμβου στη
λίστα}
var
num,i:integer;
f:boolean;
begin
 writeln    ('Δώσε το περιεχόμενο (αριθμό) του κόμβου που θέλεις να βρεις');
 readln(num);
 p:=listbase;    {Βάζω το δείκτη p να δείχνει στην αρχή της λίστας}
 i:=1;
 f:=false;
 while (p<>nil) and (f=false) do
     begin
      if p^.value=num then
        begin
          writeln    ('Το στοιχείο ',num,' βρέθηκε στη θέση ',i,' της λίστας');
          f:=true;    {Θέτω στην f την τιμή true για να σταματήσω την αναζήτηση}
        end
      else
```

```
        p:=p^.next;
       i:=i+1;
     end;
 if (f=false) then
    writeln    ('Το στοιχείο ',num,' δεν υπάρχει στη λίστα');
end;

procedure insert;    {Η διαδικασία αυτή καλείται για την παρεμβολή ενός κόμβου στη
λίστα}
var
num,pos,i:integer;
begin
 writeln    ('Δώσε τη θέση στην οποία θα παρεμβληθεί ο νέος κόμβος.');
 writeln    ('Η θέση αυτή πρέπει να είναι από 1-',j-1);
 readln(pos);
 p:=listbase;    {Βάζω το δείκτη p να δείχνει στον πρώτο κόμβο της λίστας}
 q:=p^.next;    {Βάζω το δείκτη q να δείχνει στο δεύτερο κόμβο της λίστας}

 while (pos<=1) or (pos>=j) do
    begin
     writeln    ('Δώσε άλλη θέση για παρεμβολή');
     readln(pos);    {Έτσι εξασφαλίζουμε ότι δεν θα βάλουμε το νέο κόμβο στα άκρα της
     λίστας αλλά μόνο ενδιάμεσα στη λίστα}
    end;

 writeln    ('Δώσε το περιεχόμενο (αριθμό) του κόμβου που θέλεις να παρεμβάλεις');
 readln(num);
 for i:=1 to pos-1 do
   begin
    p:=p^.next;
    q:=q^.next;
```

```
    end;

 new(w);
 w^.value:=num;
 p^.next:=w;
 w^.next:=q;
 j:=j+1;    {Αυξάνουμε το μέγεθος της λίστας κατά 1}
end;

begin
 repeat
  clrscr;
  writeln  ('1.Δημιουργία Λίστας');
  writeln  ('2.Εκτύπωση Λίστας');
  writeln  ('3.Προσθήκη 1 στοιχείου στην αρχή της λίστας');
  writeln  ('4.Προσθήκη 1 στοιχείου στο τέλος της λίστας');
  writeln  ('5.Προσθήκη 1 στοιχείου σε τυχαία θέση της λίστας');
  writeln  ('6.Διαγραφή 1 στοιχείου από τη λίστα');
  writeln  ('7.Αναζήτηση στοιχείου στη λίστα');
  writeln  ('8.Ταξινόμηση λίστας');
  ans2:=readkey;

  if ans2='1' then
    listcreate;
  if ans2='2' then
    printlist;
  if ans2='3' then
     append;
  if ans2='4' then
     append2;
  if ans2='5' then
```

```
      insert;
    if ans2='6' then
       delete;
    if ans2='7' then
       search;
    if ans2='8' then
       sort;
    writeln  ('Συνέχεια προγράμματος; y/n '):
    ans3:=readkey;
  until ans3='n';
end.
```

Σύνοψη

Η λίστα είναι μια διαφορετική δομή αυτή του πίνακα, αν και φαίνεται να μοιάζουν. Η λίστα είναι δομή σειριακής προσπέλασης, ενώ ο πίνακας τυχαίας. Η λίστα είναι δυναμική δομή, δηλαδή το μέγεθος της μεταβάλλεται κατά την εκτέλεση του αντίστοιχου προγράμματος, ενώ ο πίνακας στατική. Η λίστα, τέλος, στηρίζεται στην έννοια της ακολουθίας, ενώ ο πίνακας σ' αυτές του συνόλου και της συνάρτησης.
Διακρίνουμε δύο βασικά είδη λίστας ως προς τον τρόπο αναπαράστασής της, τις συνεχόμενες λίστες και τις συνδεδεμένες λίστες. Οι συνεχόμενες λίστες αποθηκεύουν τα στοιχεία τους σε γειτονικές (συνεχόμενες) θέσεις μνήμης και υλοποιούνται, συνήθως, μέσω πίνακα, ενώ οι συνδεδεμένες λίστες σε, συνήθως, μη γειτονικές θέσεις μνήμης και υλοποιούνται μέσω συνδέσμων ή δεικτών. Τα στοιχεία μιας συνδεδεμένης λίστας είναι πιο σύνθετα από αυτά μιας συνεχόμενης, δεδομένου ότι κρατούν και πληροφορία για τη θέση του προηγούμενου στοιχείου και, ενδεχομένως, και του προηγούμενου στοιχείου, και ονομάζονται κόμβοι. Υπάρχουν πολλά είδη συνδεδεμένων λιστών. Το πιο διαδεδομένο είναι η απλά συνδεδεμένη λίστα.

Η δυναμική φύση μιας λίστας συνιστάται στη συχνή διαγραφή και εισαγωγή στοιχείων. Στην υλοποίηση των πράξεων αυτών σε μια συνεχόμενη λίστα σαν κύριο χαρακτηριστικό έχουμε την μετακίνηση στοιχείων ενός πίνακα. Εκτός από την εισαγωγή και διαγραφή στοιχείων η διαπέραση μιας λίστας και η αναζήτηση στοιχείου σε μια λίστα είναι άλλες δύο από τις κυριότερες πράξεις σε μια λίστα.

4ο ΚΕΦΑΛΑΙΟ

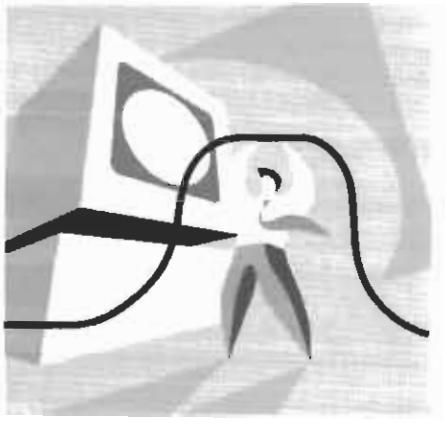
Ειδικές λίστες

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό εξετάζουμε τις ειδικές λίστες, μια άλλη κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη ενότητα παρουσιάζουμε τη στοίβα, ένας τύπος ειδικής λίστας. Δίνεται ο ορισμός της, περιγράφουμε τους τρόπους αναπαράστασής της, συνεχόμενη και συνδεδεμένη, και εξηγούμε τους αλγόριθμους για τις δύο κυριότερες πράξεις σ' αυτήν, την εισαγωγή και της διαγραφή.

Στη δεύτερη ενότητα παρουσιάζουμε δύο εφαρμογές της στοίβας στην επιστήμη των υπολογιστών, την χρήση της στην κλήση υποπρογραμμάτων και στην εκτίμηση αριθμητικών εκφράσεων.

Στην Τρίτη ενότητα ασχολούμαστε με τον άλλο τύπο ειδικής λίστας, την 'ουρά'. Δίνεται ο ορισμός της και παρουσιάζουμε τους τρόπους αναπαράστασής της, συνεχόμενη και συνδεδεμένη, και τους αλγόριθμους για την εισαγωγή και διαγραφή στοιχείων/ κόμβων.



4.1 Στοίβα

4.1.1 Ορισμός και αναπαράσταση

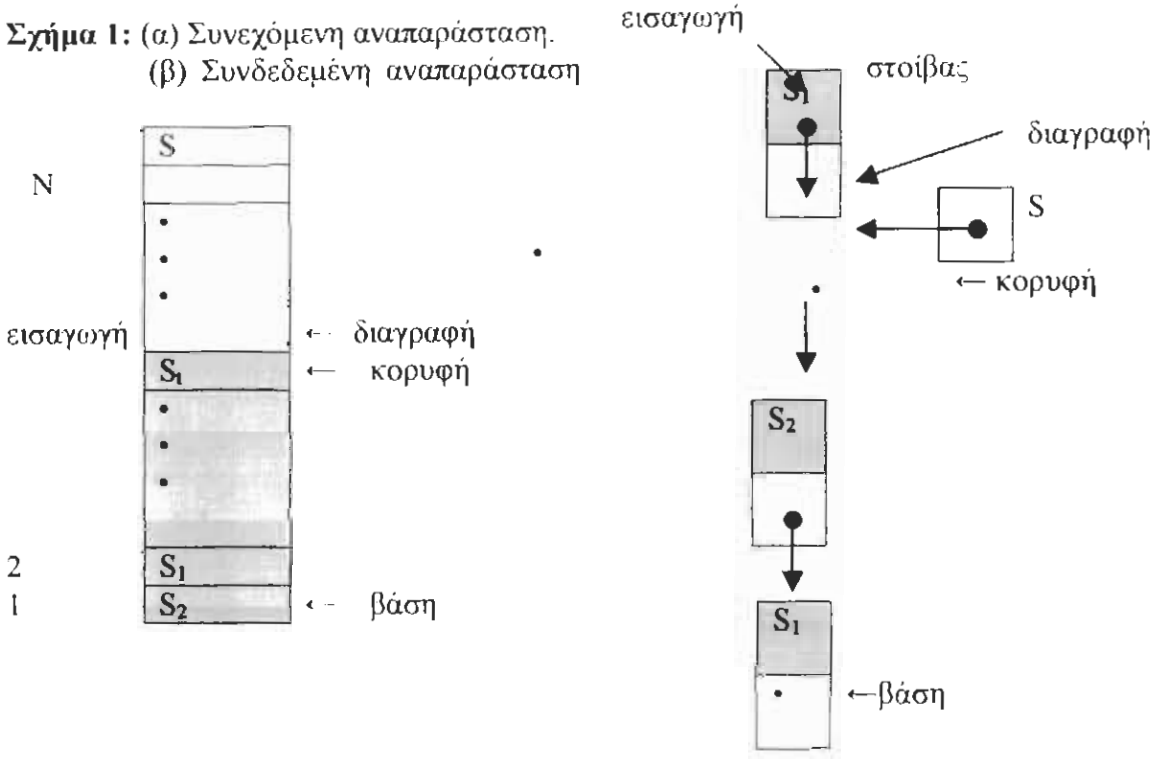
Μια στοίβα (stack) είναι ένας ειδικός τύπος λίστας, δηλαδή μια λίστα που έχει ορισμένους περιορισμούς ως προς την εφαρμογή των διαφόρων πράξεων σε αυτή. Πιο συγκεκριμένα μια στοίβα επιτρέπει εισαγωγές και διαγραφές στοιχείων /κόμβων μόνο στο ένα άκρο της, που ονομάζεται κορυφή (top) της στοίβας και χαρακτηρίζεται ως το 'ελεύθερο' άκρο της. Το άλλο άκρο το μη ελεύθερο, ονομάζεται βάση (bottom).

Δηλαδή, σε κάθε στιγμή σε μια στοίβα μπορούμε είτε να εισάγουμε ένα στοιχείο στη κορυφή της είτε να διαγράψουμε (ή να εισάγουμιε) ένα στοιχείο από την κορυφή της.

Στην τελευταία περίπτωση, το στοιχείο που εισήχθη τελευταίο είναι αυτό που διαγράφεται. Μια τέτοιου είδους λογική ονομάζεται λογική LIFO (Last-In-First-Out). Από την καθημερινή μας ζωή, ένα σύνηθες παράδειγμα στοίβας, είναι αυτό της στοίβας πιάτων σ'ένα εστιατόριο, όπου τα πιάτα που πλένονται και τοποθετούνται πρώτα, βρίσκονται στο πάτο της στοίβας και αυτό που τοποθετήθηκε τελευταίο είναι αυτό που θα πάρει πρώτο ο σερβιτόρος για σερβίρισμα.

Η αναπαράσταση μιας στοίβας γίνεται είτε με τον τρόπο της αναπαράστασης είτε με αυτόν της συνδεδεμένης αναπαράστασης. Έτσι, διακρίνουμε μεταξύ *συνεχόμενης στοίβας (contiguous stack)* και *συνδεδεμένης στοίβας (linked stack)*. Στην πρώτη περίπτωση, η στοίβα υλοποιείται μέσω ενός πίνακα, ενώ στη δεύτερη, μέσω συνδέσμων ή δεικτών. Μια συνεχόμενη στοίβα σε μια γλώσσα υψηλού επιπέδου μπορεί να οριστεί σαν μια εγγραφή με δύο πεδία, που το ένα ένας πίνακας που αποθηκεύονται τα στοιχεία της και το άλλο ένας δείκτης στην κορυφή της. Μια συνδεδεμένη στοίβα μπορεί να οριστεί σαν μια μεταβλητή δείκτη που αποθηκεύει τον δείκτη στην κορυφή της στοίβας. Κάθε κόμβος της στοίβας ορίζεται σαν μια εγγραφή με δύο πεδία, που το ένα περιέχει το στοιχείο και το άλλο το δείκτη στον επόμενο κόμβο. Στο Σχήμα 1 απεικονίζονται οι δύο τρόποι αναπαράστασης μιας στοίβας.

Σχήμα 1: (α) Συνεχόμενη αναπαράσταση.
(β) Συνδεδεμένη αναπαράσταση



4.1.2 Πράξεις σε στοίβα

Όπως είναι προφανές, δύο είναι οι κυριότερες πράξεις που μας ενδιαφέρουν σε μια στοίβα, η 'εισαγωγή' και η 'διαγραφή' (ή 'εξαγωγή') ενός στοιχείου / κόμβου. Στη συνέχεια, παρουσιάζονται οι αλγόριθμοι για τις δύο αυτές πράξεις και για τους δύο τρόπους αναπαράστασης της στοίβας.

Συνεχόμενη Αναπαράσταση

Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου από μια συνεχόμενη στοίβα παρουσιάζονται στα παρακάτω πλαίσια.

ΑΛΓΟΡΙΘΜΟΣ 4.1 : ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ένας πίνακας (S), το μέγεθος του (N), ο δείκτης κορυφής της αντίστοιχης στοίβας (T) και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά επιστρέφει μήνυμα υπερχείλισης στην περίπτωση γεματης στοίβας, αλλιώς τίποτα. Εσωτερικά, καταχωρείται η τιμή στη στοίβα σε περίπτωση επιτυχίας.

```
SS-EISAGWGH (S,N,T,X)
1    if T=N                          {Έλεγχος γεμάτης στοίβας}
2      then print 'ΥΠΕΡΧΕΙΛΙΣΗ'      {Μήνυμα υπερχείλισης}
3      else T ← T+1                  {Ενημέρωση κορυφής}
4           S[T] ← X                 {Καταχώρηση στοιχείου}
     endif
```

ΑΛΓΟΡΙΘΜΟΣ 4.2: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ένας πίνακας (S) και ο δείκτης κορυφής της αντίστοιχης στοίβας (T).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση άδειας στοίβας, αλλιώς το εξαγχθέν στοιχείο. Εσωτερικά, διαγράφεται η κορυφή της στοίβας σε περίπτωση επιτυχίας.

```
SS-DIAGRAFH (S,T)
1    if T=0                          {Έλεγχος άδειας στοίβας}
2      then print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'      {Μήνυμα άδειας στοίβας}
3      else print S[T]                {Επιστροφή στοιχείου}
4           T ← T-1                   {Ενημέρωση κορυφής}
     endif
```

Η λογική των δύο αλγόριθμων είναι απλή. Στον Αλγόριθμο 4.1 (SS-EISAGWGH), μετά τον έλεγχο για το αν η στοίβα είναι γεμάτη και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) αυξάνεται

ο δείκτης κορυφής της στοίβας κατά ένα, οπότε «δείχνει» την επόμενη κενή θέση στον πίνακα (βήμα 3) και κατόπιν καταχωρείται το στοιχείο στην κενή αυτή θέση (βήμα 4). Στον Αλγόριθμο 4.2 (SS-DIAGRAFH), μετά τον έλεγχο για το αν η στοίβα είναι άδεια και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) επιστρέφει στην τιμή του στοιχείου της κορυφής (βήμα 3) και κατόπιν μειώνεται ο δείκτης κορυφής της στοιβας κατά ένα, οπότε «δείχνει» την προηγούμενη θέση στη στοίβα. Η επιστροφή της τιμής γίνεται, διότι συνήθως ένα στοιχείο διαγράφεται από μια στοίβα για να χρησιμοποιηθεί κάπου.

Συνδεδεμένη Αναπαράσταση

Οι Αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου από μια απλά συνδεδεμένη στοίβα παρουσιάζονται στα παρακάτω πλαίσια.

ΑΛΓΟΡΙΘΜΟΣ 4.3: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ο δείκτης στον προς εισαγωγή κόμβο (P) και ο δείκτης στην κορυφή μιας συνδεδεμένης λίστας (S).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση ανυπαρξίας του προς εισαγωγή κόμβου, αλλιώς τίποτα. Εσωτερικά, εισάγεται ο νέος κόμβος στην στοίβα, σε περίπτωση επιτυχίας.

```
DS-EISAGWGH (P,S)
1   if P=NIL                                  {Έλεγχος δείκτη}
2     then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'          {Μήνυμα λάθους}
3     else DEIKTHS(KOMBOS(P)) ← S              {Καταχώρηση δείκτη}
4          S ← P                               {Ενημέρωση δείκτη}
    endif
```

ΑΛΓΟΡΙΘΜΟΣ 4.4: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ο δείκτης στην κορυφή μιας συνδεδεμένης λίστας (S).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα σε περίπτωση άδειας στοίβας, αλλιώς το στοιχείο του διαγραφέντος κόμβου. Εσωτερικά, διαγράφεται η κορυφή της στοίβας σε περίπτωση επιτυχίας.

```
DS-DIAGRAFH (S)
1  if S = NIL                          {Έλεγχος κενής στοίβας}
2    then print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'          {Μήνυμα κενής στοίβας}
3    else print STOIXEIO(KOMBOS(S))     {Επιστροφή στοιχείου κορυφής}
4         S ← DEIKTHS(KOMBOS(S))        {Ενημέρωση δείκτη κορυφής}
   endif
```

Η λογική και αυτών των δύο αλγορίθμων είναι απλή.

Στον Αλγόριθμο 4.3 (DS-EISAGWGH), μετά τον έλεγχο για το αν ο δείκτης στον προς εισαγωγή κόμβο είναι έγκυρος και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) γίνεται εναλλαγή των δεικτών. Ο δείκτης του προς εισαγωγή κόμβου παίρνει την τιμή του S, ώστε να «δείχνει» στον προηγούμενο κορυφαίο κόμβο (βήμα 3), ενώ ο δείκτης κορυφής παίρνει την τιμή του P, ώστε να δείχνει στον νέο κόμβο που είναι η νέα κορυφή.

Στον Αλγόριθμο 4.4 (DS-DIAGRAFH), μετά τον έλεγχο για το αν η στοίβα είναι άδεια και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα3-4) επιστρέφει την τιμή του στοιχείου της κορυφής (βήμα 3) και κατόπιν ενημερώνεται ο δείκτης κορυφής της στοίβας, οπότε «δείχνει» στον προηγούμενο κόμβο στη στοίβα που τώρα γίνεται η νέα κορυφή.

Παράδειγμα 4.1: *Διαγραφή Κ πρώτων στοιχείων συνεχόμενης στοίβας*

Θεωρούμε μια συνεχόμενη στοίβα. Το ζητούμενο είναι να σχεδιαστεί ένας αλγόριθμος που να διαγράφει (εξάγει) τα Κ πρώτα στοιχεία της στοίβας και να επιστρέφει το Κ-οστό στοιχείο. Αν δεν υπάρχουν Κ στοιχεία στη στοίβα, τότε να επιστρέφει το τελευταίο διαγραφέν στοιχείο και μήνυμα άδειας στοίβας. Ο αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος Π4.1 (SS-DIAGRAFH-KP).

Το βασικό βήμα (βήμα 3) και η ενημέρωση του μετρητή (βήμα 4) βρίσκονται μέσα στη διάταξη επανάληψης while (βήματα 2-4). Η επανάληψη συνεχίζεται, εφόσον ο δείκτης κορυφής (T) παραμένει θετικός και ο μετρητής (I) όχι μεγαλύτερος του K. Σταματά επομένως, όταν είτε T=0 είτε I>K (πιο συγκεκριμένα I=K+1). Αυτές είναι και οι δύο περιπτώσεις στις οποίες βασίζεται η σύνθετη διάταξη if (βήματα 5-8) που ακολουθεί. Αν συμβαίνει η δεύτερη περίπτωση (επιτυχία) (βήμα 5), τότε επιστρέφει το K-οστό διαγραφέν στοιχείο (βήμα 6). Αν, όχι, τότε επιστρέφει το τελευταίο διαγραφέν στοιχείο και το μήνυμα άδειας στοίβας.

ΑΛΓΟΡΙΘΜΟΣ Π4.1: ΔΙΑΓΡΑΦΗ Κ ΣΤΟΙΧΕΙΩΝ ΣΥΝΕΧΟΜΕΝΗΣ ΣΤΟΙΒΑΣ

Είσοδος: Ένας πίνακας (S), ο δείκτης κορυφής της αντίστοιχης στοίβας (T) και ένας ακέραιος αριθμός (K).

Έξοδος: εξωτερικά, επιστρέφει μήνυμα σε περίπτωση άδειας στοίβας ή στοίβας με λιγότερα από K στοιχεία, αλλιώς το τελευταίο από τα διαγραφέντα στοιχεία. Εσωτερικά, διαγράφονται τα K πρώτα ή όλα στοιχεία της στοίβας, σε περίπτωση επιτυχίας.

```
SS-DIAGRAFH- KP (S, T, K)
1   I ← 1                          {Αρχικοποίηση μετρητή}
2   while (T > 0) and (I < K)      {Συνθήκη επανάληψης}
3      T ← T-I                     {Ενημέρωση δείκτη κορυφής}
4      I ← I+1                     {Ενημέρωση μετρητή}
    endwhile
5   if I = K+1                     {Έλεγχος επιτυχίας}
6     then print S[T+1]            {Επιστροφή K-οστού στοιχείου}
7     else print S[1]              {Επιστροφή τελευταίου στοιχείου}
8          print 'ΑΔΕΙΑ ΛΙΣΤΑ'     {Μήνυμα άδειας στοίβας}
    endif
```

4.2 Εφαρμογές στοίβας

4.2.1 Κλήση υποπρογραμμάτων

Μια εφαρμογή της στοίβας στην επιστήμη των υπολογιστών σχετίζεται με την κλήση υποπρογραμμάτων. Ας υποθέσουμε ότι έχουμε ένα κυρίως πρόγραμμα α και τρία υποπρογράμματα, τα Β, Γ και Δ, και ότι το Α καλεί το Β, το Β καλεί το Γ και το Γ καλεί το δ κατά την εκτέλεσή του. Δηλαδή, για την εκτέλεση του Α απαιτείται η εκτέλεση όλων των υποπρογραμμάτων και επομένως, ενώ ξεκινά πρώτο τελειώνει τελευταίο. Το Β για να εκτελεστεί χρειάζεται η εκτέλεση των Γ και Δ και επομένως, ενώ ξεκινά δεύτερο, τελειώνει προτελευταίο κ.ο.κ. Το Δ που ξεκινά τελευταίο, τελειώνει πρώτο. Αυτή η κατάσταση, προφανώς, συνιστά μια λογική LIFO και επομένως ευνοεί τη χρήση μιας στοίβας για τον χειρισμό των κλήσεων των υποπρογραμμάτων.

Για το σκοπό αυτό χρησιμοποιείται μια στοίβα στην οποία καταχωρούνται οι διευθύνσεις των περιοχών της μνήμης όπου κρατούνται τα δεδομένα (η κατάσταση) κάθε (υπο)προγράμματος, όταν εγκαταλείπεται προσωρινά για να εκτελεστεί κάποιο άλλο υποπρόγραμμα. Τα δεδομένα αυτά, που συχνά ονομάζονται *εγγραφή ενεργοποίησης (activation record)* ή *πλαίσιο στοίβας (stack frame)*, περιλαμβάνουν όλα τα απαραίτητα στοιχεία για να επανέλθει η εκτέλεση του (υπο)προγράμματος στην κατάσταση που ήταν όταν διακόπηκε.

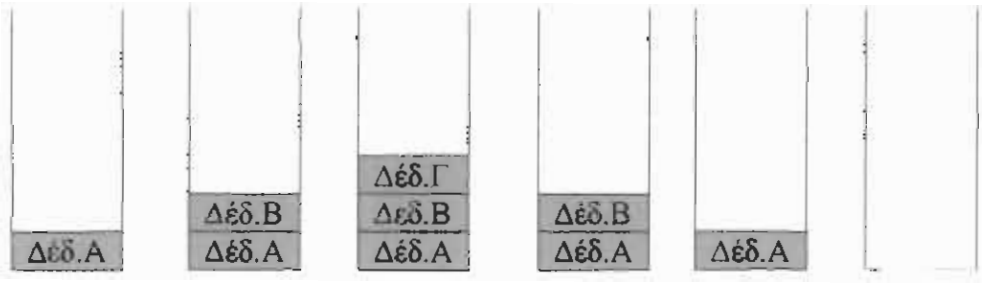
Μια εφαρμογή της στοίβας στην επιστήμη των υπολογιστών σχετίζεται με την κλήση υποπρογραμμάτων. Ας υποθέσουμε ότι έχουμε ένα κυρίως πρόγραμμα και τρία υποπρογράμματα, τα Β, Γ και Δ, και ότι το Α καλεί το Β, το Β καλεί το Γ και το Γ καλεί το Δ κατά την εκτέλεσή του. Δηλαδή, για την εκτέλεση του Α απαιτείται η εκτέλεση όλων των υποπρογραμμάτων και επομένως, ενώ ξεκινά πρώτο τελειώνει τελευταίο. Το Β για να εκτελεστεί χρειάζεται η εκτέλεση των Γ και Δ και επομένως, ενώ ξεκινά δεύτερο, τελειώνει προτελευταίο κ.ο.κ. το Δ που ξεκινά τελευταίο, τελειώνει πρώτο. Αυτή η κατάσταση, προφανώς, συνιστά μια λογική LIFO και επομένως ευνοεί τη χρήση μιας στοίβας για τον χειρισμό των κλήσεων των υποπρογραμμάτων.

Για το σκοπό αυτό χρησιμοποιείται μια στοίβα στην οποία καταχωρούνται οι διευθύνσεις των περιοχών της μνήμης όπου κρατούνται τα δεδομένα (η κατάσταση) κάθε (υπο)προγράμματος, όταν εγκαταλείπεται προσωρινά για να εκτελεστεί κάποιο άλλο υποπρόγραμμα. Τα δεδομένα αυτά, που συχνά ονομάζονται εγγραφή ενεργοποίησης (activation record) ή πλαίσιο στοίβας (stack frame), περιλαμβάνουν όλα τα απαραίτητα

στοιχεία για να επανέλθει η εκτέλεση του (υπο)προγράμματος στην κατάσταση που ήταν όταν διακόπηκε.

Στο Σχήμα 2 απεικονίζονται οι καταστάσεις της στοίβας κατά την εκτέλεση του προγράμματος Α. Πρώτο ξεκινά το πρόγραμμα Α και σε κάποια στιγμή καλεί το υποπρόγραμμα Β. Τότε τα δεδομένα του Α κατά εκείνη τη στιγμή, τοποθετούνται σε μια προσωρινή περιοχή της μνήμης και η διεύθυνση της περιοχής αυτής εισάγεται στη στοίβα (Σχήμα 2α) και ξεκινά η εκτέλεση του Β. Σε κάποια στιγμή, το Β καλεί το υποπρόγραμμα Γ και τότε τα δεδομένα του Β τοποθετούνται στη μνήμη και η αντίστοιχη διεύθυνση στη στοίβα (Σχήμα 2β) κ.ο.κ. μέχρι να κληθεί το Δ από το Γ. Μόλις τελειώσει το Δ, εξάγεται (διαγράφεται) η διεύθυνση των δεδομένων του Γ από τη στοίβα (Σχήμα 2δ), ανακαλείται το περιεχόμενό της και συνεχίζεται η εκτέλεση του υποπρογράμματος Γ από εκεί που σταμάτησε κ.ο.κ. μέχρι να τελειώσει και η εκτέλεση του Β, οπότε εξάγεται και η διεύθυνση των δεδομένων του Α και η στοίβα μένει κενή (Σχήμα 2στ).

Σχήμα 2: Στοίβα για την κλήση υποπρογραμμάτων.



4.2.2Εκτίμηση αριθμητικών εκφράσεων

Μια άλλη εφαρμογή της στοίβας στην επιστήμη των υπολογιστών αφορά την εκτίμηση αριθμητικών εκφράσεων. Ο συνήθης τρόπος (συμβολισμός) με τον οποίο γράφουμε τις αριθμητικές εκφράσεις ονομάζεται ενδοθεματική μορφή (infix notation). Για παράδειγμα, τέτοιας μορφής είναι οι εκφράσεις (α+β), (α+β)*γ, α*β+γ*δ, στις

οποίες οι αριθμητικοί τελεστές (operators), (π.χ. +,*, -, /) βρίσκονται ανάμεσα στα στοιχεία τα οποία εφαρμόζονται, δηλαδή ανάμεσα στους τελεστέους (operands).

Εκτός όμως της ενδοθεματικής μορφής χρησιμοποιούνται και άλλες μορφές γραφής αριθμητικών εκφράσεων. Μια τέτοια είναι η λεγόμενη προθεματική μορφή (prefix notation) ή Πολωνική μορφή (Polish notation), στην οποία οι τελεστές προηγούνται των τελεστέων και δεν απαιτούνται παρενθέσεις. Οι προηγούμενες εκφράσεις σε προθεματική μορφή γράφονται, +αβ,* +αβγ, +*αβ*γδ.

Μια τρίτη μορφή είναι η μεταθεματική μορφή (postfix notation) ή αντίστροφη Πολωνική μορφή (reserve Polish notation), στην οποία οι τελεστές γράφονται μετά από τους τελεστέους (αριθμούς). Π.χ., οι παραπάνω αριθμητικές εκφράσεις σε μεταθεματική μορφή γράφονται, αβ+, αβ+γ*, αβ*γδ*+.

Οι μορφές αυτές βρίσκουν εφαρμογή στα μεταφραστικά προγράμματα (μεταγλωττιστές, διερμηνευτές) των γλωσσών προγραμματισμού υψηλού επιπέδου. Μερικά μεταφραστικά προγράμματα χρησιμοποιούν εσωτερικά την μεταθεματική μορφή. Έτσι , ο υπολογισμός της τιμής μιας αριθμητικής έκφρασης γίνεται σε δύο στάδια. Στο πρώτο γίνεται μετατροπή της ενδοθεματικής μορφής σε μεταθεματική, ενώ στο δεύτερο στάδιο γίνεται υπολογισμός της τιμής (εκτίμηση) της έκφρασης από την μεταθεματική της μορφή. Και οι δύο αλγόριθμοι που υλοποιούν τα δύο αυτά στάδια χρησιμοποιούν σαν βάση μια στοίβα. Στη συνέχεια, θα αναφερθούμε στο δεύτερο αλγόριθμο, αυτόν που υπολογίζει την τιμή μιας παράστασης από την μεταθεματική της μορφή.

Πίνακα 1

Βήμα	Στοιχείο	Στοίβα	Υπολογισμοί
1	6	6	
2	3	3 6	
3	4	4 3 6	4 * 3 = 12
4	*	12 6	12 + 6 = 18
5	+	18	
6	2	2 18	18 / 2 = 9
7	/	9	

Θα περιγράψουμε τα βήματα του αλγόριθμου για μια συγκεκριμένη αριθμητική έκφραση, την 6 3 4 * + 2 / (δηλαδή την α β γ * + δ / με α=6, β=3, γ=4, και δ=2), που είναι η μεταθεματική μορφή της (6 + 3*4)/2. Τα βήματα και οι αντίστοιχες ενέργειες φαίνονται στον Πίνακα 1, όπου βάση της στοίβας είναι ο δεξιότερα ευρισκόμενος αριθμός (3η στήλη). Ο αλγόριθμος βασίζεται στην εξής λογική: Σαρώνεται η μεταθεματική έκφραση και (α) όταν συναντάτε αριθμός, τότε προωθείται (εισάγεται) στη στοίβα, (β) όταν συναντάτε τελεστής τότε εξάγουμε (διαγράφουμε) τα δύο τελευταία στοιχεία (αριθμούς) της στοίβας, εφαρμόζουμε την πράξη του τελεστή αυτά, υπολογίζουμε το αποτέλεσμα και το εισάγουμε στη στοίβα. Όταν έχουμε διέλθει όλα τα στοιχεία της έκφρασης τότε αυτό που μένει στη στοίβα είναι το αποτέλεσμα, η τιμή της αριθμητικής έκφρασης.

4.3 Ουρά

4.3.1 Ορισμός και αναπαράσταση

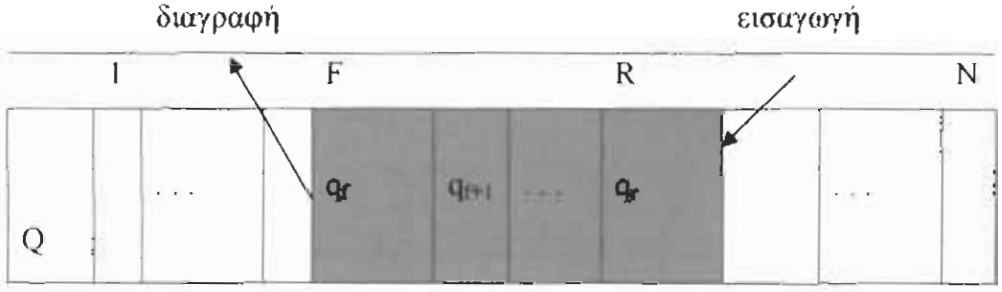
Στην καθημερινή ζωή υπάρχουν καταστάσεις που δεν μπορούν να περιγράφουν με βάση την έννοια της στοίβας. Τέτοιες καταστάσεις είναι οι ουρές αναμονής σε μια τράπεζα, στη στάση λεωφορείου, για τις θέσεις μιας αεροπορικής πτήσης κλπ. Στις περιπτώσεις αυτές, αυτός που προσήλθε πρώτος στην ουρά εξυπηρετείται και πρώτος. Ακολουθείται δηλαδή μια διαφορετική λογική από εκείνη στις στοίβες, που ονομάζεται FIFO(First-In-First-Out). Η δομή δεδομένων που μπορεί να αναπαραστήσει αυτή τη λογική είναι ένας ειδικός τύπος λίστας που ονομάζεται ουρά (queue). Οι περιορισμοί στην ουρά, σε σχέση με μια γενική λίστα, είναι οι εξής: α) Εισαγωγές στοιχείων μπορούν να γίνονται μόνο από το ένα άκρο της, που ονομάζεται πίσω (rear) ή τέλος και β) διαγραφές μπορούν να γίνονται μόνο από το άλλο άκρο της, που ονομάζεται εμπρός (front) ή αρχή.

Η αναπαράσταση μιας ουράς μπορεί να γίνει, όπως και στην περίπτωση της στοίβας, με δύο τρόπους, είτε με τον συνεχόμενο είτε με τον συνδεδεμένο, δηλαδή είτε με πίνακα είτε με δείκτες , οπότε διακρίνουμε δύο τύπους ουράς, συνεχόμενη ουρά (contiguous queue) και συνδεδεμένη ουρά (linked queue). Στο Σχήμα 3 απεικονίζεται μια ουρά και με τους δύο τρόπους αναπαράστασης.

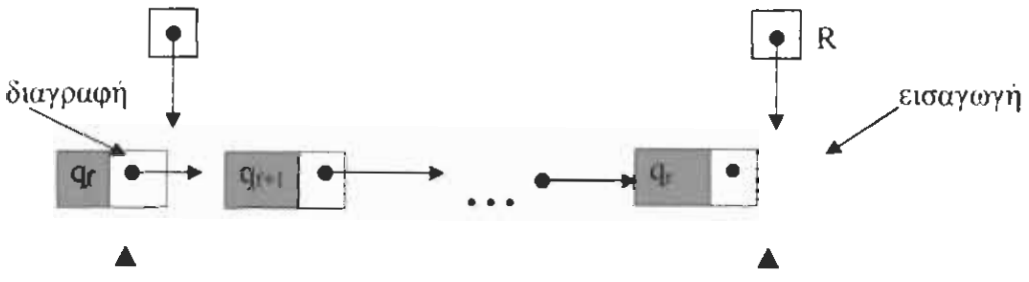
Σχήμα 3

(α) Συνεχόμενη αναπαράσταση,

(β) συνδεδεμένη αναπαράσταση ουράς



εμπρός (αρχή) πίσω (τέλος) (α)



εμπρός (αρχή) πίσω (τέλος) (β)

Μια συνεχόμενη ουρά σε μια γλώσσα υψηλού επιπέδου, μπορεί να οριστεί σαν μια εγγραφή με τρία πεδία : δύο μετρητές που αντιπροσωπεύουν την αρχή και το τέλος της και ένας πίνακας για την αποθήκευση των στοιχείων της . Μια συνδεδεμένη ουρά μπορεί να οριστεί σαν μια εγγραφή με δύο πεδία που είναι μεταβλητές δείκτη. Η μια δείχνει στην αρχή και η άλλη στο τέλος της ουράς. Ο κάθε κόμβος ορίζεται όπως και σε μια συνδεδεμένη λίστα.

4.3.2 Πράξεις σε ουρά

Όπως και στη στοίβα, έτσι και στην ουρά δύο είναι οι κυριότερες πράξεις, η εισαγωγή και η διαγραφή. Οι αντίστοιχοι αλγόριθμοι και για τις δύο αναπαραστάσεις παρουσιάζονται στη συνέχεια.

Συνεχόμενη Αναπαράσταση

Στη συνεχόμενη αναπαράσταση η 'άδεια ουρά' ορίζεται από την συνθήκη F=R−0, ενώ η ουρά με ένα στοιχείο από τη συνθήκη F=R−1. Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου παρουσιάζονται στα παρακάτω πλαίσια ως Αλγόριθμοι 4.5 και 4.6

ΑΛΓΟΡΙΘΜΟΣ 4.5: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΟΥΡΑ

Είσοδος: Ένας πίνακας (Q) , το μέγεθος του (N), ο δείκτης αρχής (F) και τέλους (R) της αντίστοιχης ουράς και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά επιστρέφει μήνυμα γεμάτης ουράς, αλλιώς τίποτα. Εσωτερικά, καταχωρείται η τιμή στο τέλος της ουράς, σε περίπτωση επιτυχίας.

```
SQ-EISAGWGH (Q, N, F, R, X)
1       if R ≥ N                          {Έλεγχος γεμάτης ουράς }
2        then print 'ΓΕΜΑΤΗ ΟΥΡΑ'          {Μήνυμα γεμάτης ουράς}
3        else R ← R+1                      {Ενημέρωση δείκτη τέλους}
4             Q[R] ← X                     {Καταχώρηση τιμής}
5              if F = 0                    {Έλεγχος άδειας ουράς}
6               then F ← 1                 {Ενημέρωση δείκτη αρχής}
              endif
        endif
```

Η λογική του Αλγόριθμου 4.5 (SQ-EISAGWGH) είναι η εξής: Μετά τον έλεγχο για γεμάτη ουρά (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-6), καταρχήν (βήμα 3) αυξάνεται ο δείκτης τέλους της ουράς κατά ένα, ώστε να δείχνει στην επόμενη κενή θέση του πίνακα, και κατόπιν καταχωρείται η τιμή X στη θέση αυτή (βήμα 4). Στη συνέχεια, ελέγχουμε αν η ουρά ήταν κενή (βήμα 5) και ενημερώνουμε τον δείκτη αρχής (βήμα 6).

Στον αλγόριθμο 4.6 (SQ-DIAGRAFH) έχουμε τα εξής: Μετά τον έλεγχο για άδεια ουρά (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-6), καταρχήν (βήμα 3) επιστρέφει την αρχή της ουράς (για όποια χρήση). Κατόπιν (βήμα 4), αυξάνεται ο δείκτης αρχής της ουράς κατά ένα, ώστε να δείχνει στο επόμενο στοιχείο του πίνακα (της ουράς).Στη συνέχεια, ελέγχουμε αν η ουρά γίνεται άδεια (βήμα 5), δηλαδή αν υπήρχε ένα μόνο στοιχείο πριν τη διαγραφή (F=R=1), και ενημερώνουμε τους δείκτες αρχής και τέλους (βήμα 6).

ΑΛΓΟΡΙΡΙΘΜΟΣ 4.6: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΟΥΡΑ

Είσοδος: Ένας πίνακας (Q), ο δείκτης αρχής (F) και τέλους (R) της αντίστοιχης ουράς.
Έξοδος: Εξωτερικά, επιστρέφει μήνυμα άδειας ουράς, αλλιώς το διαγραφέν στοιχείο. Εσωτερικά, διαγράφεται το στοιχείο από την αρχή της ουράς, σε περίπτωση επιτυχίας.

```
SQ-DIAGRAFH (Q, F, R)
1     if F = 0                          {Έλεγχος άδειας ουράς}
2       then print 'ΑΔΕΙΑ ΟΥΡΑ'         {Μήνυμα άδειας ουράς}
3       else print Q[F]                 {Επιστροφή διαγραφέντος}
4            F ← F+1                    {Ενημέρωση δείκτη αρχής}
5            if F > R                   {Έλεγχος άδειας ουράς}
6              then F ← 0, R ← 0        {Ενημέρωση δεικτών}
       endif
     endif
```

Συνδεδεμένη Αναπαράσταση

Στη συνδεδεμένη αναπαράσταση η 'άδεια ουρά' ορίζεται από την συνθήκη F = R – N I L, ενώ η ουρά με ένα στοιχείο από τη συνθήκη F = R ≠ NIL. Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου παρουσιάζονται στα παρακάτω πλαίσια ως Αλγόριθμοι 4.7 και 4.8

ΑΛΓΟΡΙΘΜΟΣ 4.7: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΟΥΡΑ

Είσοδος: Ο δείκτης αρχής (F) και τέλους (R) μιας ουράς και ο δείκτης (P) της αντίστοιχης ουράς.

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, εισάγεται ο κόμβος στο τέλος της ουράς.

```
DQ-EISAGWGH (F, R, P)
1       if F=NIL                          {Έλεγχος άδειας ουράς}
2         then F ← P                      {Ενημέρωση δείκτη αρχής}
3         else DEIKTHS(KOMBOS(R)) ← P     {Ενημέρωση τελευταίου κόμβου}
4              DEIKTHS(KOMBOS(P)) ← NIL   {Δημιουργία τελευταίου κόμβου}
    endif
    5   R ← P                             {Ενημέρωση δείκτη τέλους}
```

Η λογική του Αλγόριθμου 4.7 (DQ-EISAGWGH) έχει ως ακολούθως: Κατ' αρχήν, εξετάζεται αν η ουρά είναι άδεια (βήμα 1). Αν είναι τότε απλώς ο δείκτης αρχής παίρνει τη τιμή του δείκτη στον προς εισαγωγή κόμβο (βήμα 2). Αλλιώς, ο δείκτης του τελευταίου κόμβου της ουράς παίρνει τη τιμή του P (βήμα 3), οπότε «δείχνει» στον προς εισαγωγή κόμβο, που γίνεται ο τελευταίος της ουράς (βήμα 4). Τέλος, σε κάθε περίπτωση, ενημερώνεται ο δείκτης τέλους (βήμα 5).

ΑΛΓΟΡΙΘΜΟΣ 4.8: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΟΥΡΑ

Είσοδος: Ο δείκτης αρχής (F) και τέλους (R) μιας ουράς.

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση άδειας ουράς, αλλιώς το διαγραφέν στοιχείο. Εσωτερικά, διαγράφεται το στοιχείο από την αρχή της ουράς, σε περίπτωση επιτυχίας.

```
DQ-DIAGRAFH (F, R, X)
1     if F = NIL                          {Έλεγχος άδειας ουράς}
2       then print 'ΑΔΕΙΑ ΟΥΡΑ'           {Μήνυμα άδειας ουράς}
3       else print ΣΤΟΙΧΕΙΟ(ΚΟΜΒΟΣ(F))    {Επιστροφή στοιχείου}
4            F ← ΔΕΙΚΤΗΣ(ΚΟΜΒΟΣ(F))     {Ενημέρωση δείκτη αρχής}
5            if F = NIL                  {Έλεγχος άδειας ουράς}
6              then R ← NIL              {Ενημέρωση δείκτη τέλους}
           endif
       endif
```

Στον Αλγόριθμου 4.8 (DQ-DIAGRAFH), κατ' αρχήν εξετάζεται αν η ουρά είναι κενή (βήμα 1) και επιστρέφει κατάλληλο μήνυμα (βήμα 2). Αλλιώς, επιστρέφει την τιμή του στοιχείου του πρώτου κόμβου της ουράς (βήμα 3) και ενημερώνεται ο δείκτης F, ώστε να «δείχνει» στο δεύτερο κόμβο, που τώρα γίνεται πρώτος (βήμα 4). Τέλος, ελέγχουμε αν με τη διαγραφή άδειασε η ουρά (βήμα 5), οπότε ενημερώνουμε τον δείκτη R.

Παράδειγμα 4.2

Άδειασμα συνδεδεμένης ουράς σε άδεια συνεχόμενη στοίβα

Θεωρούμε μια απλά συνδεδεμένη ουρά και ζητείται να σχεδιαστεί αλγόριθμος που να την αδειάζει και να τοποθετεί τα στοιχεία των κόμβων της, σε μια άδεια συνεχόμενη στοίβα.

Η λογική του ζητούμενου αλγόριθμου είναι σχετικά απλή. Γίνονται διαδοχικές διαγραφές των κόμβων της ουράς και κάθε φορά τοποθετούμε το στοιχείο του προς διαγραφή κόμβου στη συνεχόμενη στοίβα. Επομένως, η βάση για τη σχεδίαση είναι ο βασικός αλγόριθμος διαγραφής (Αλγόριθμος 4.8), ο ζητούμενος αλγόριθμος δίνεται στο παρακάτω πλαίσιο ως Αλγόριθμος Π4.2 (DQ-ADEIASMA-SS).

Στο βήμα 1 τίθεται η τιμή του δείκτη κορυφής της στοίβας στο μηδέν (άδεια στοίβα). Στη συνέχεια, ενεργοποιείται η επαναληπτική διαδικασία διαγραφής από την ουρά και καταχώρησης στη στοίβα μέσω της διάταξης while (βήματα 2-5). Σε κάθε επανάληψη, ο δείκτης κορυφής αυξάνει κατά ένα, ώστε να δείχνει στην επόμενη κενή θέση της στοίβας (βήμα 3), καταχωρείται το στοιχείο του τρέχοντος κόμβου αρχής της ουράς στη στοίβα (βήμα 4) και ενημερώνεται ο δείκτης αρχής της ουράς, ώστε να δείχνει στον επόμενο κόμβο (βήμα 5). Τέλος, ενημερώνεται και ο δείκτης τέλους της ουράς (άδεια ουρά) (βήμα 6). Επειδή θεωρούμε στοίβα με ικανό μέγεθος πίνακα, δεν τίθεται θέμα ελέγχου υπερχείλισης της στοίβας.

ΑΛΓΟΡΙΘΜΟΣ Π4.2: ΑΔΕΙΑΣΜΑ ΣΥΝΔΕΔΕΜΕΝΗΣ ΟΥΡΑΣ ΣΕ ΣΤΟΙΒΑ

Είσοδος: Οι δείκτες αρχής (F) και τέλους (R) μιας συνδεδεμένης ουράς, ένας πίνακας (S) και ο δείκτης κορυφής της αντίστοιχης στοίβας (T).

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, αδειάζει η ουρά και γεμίζει μερικώς η στοίβα.

```
DQ-ADEIASMA-SS (F, R, S, T)
1       T ← 0                              {Αρχικοποίηση δείκτη κορυφής}
2       while F ≠ NIL                      {Συνθήκη ελέγχου επανάληψης}
3         T ← T+1                          {Ενημέρωση δείκτη κορυφής}
4         S[T] ← STOIXEIO(KOMBOW(F))       {Καταχώρηση στη στοίβας}
5         F ← DEIKTHS(KOMBOS(F))           {Ενημέρωση δείκτη αρχής}
Endwhile
   6    R ← NIL                            {Ενημέρωση δείκτη τέλους}
```

Σύνοψη

Οι ***ειδικές λίστες*** είναι λίστες που θέτουν περιορισμούς όσον αφορά τις πράξεις σε αυτές. Η ***στοίβα*** είναι ένας τύπος ειδικής λίστας στην οποία επιτρέπονται εισαγωγές και διαγραφές (εξαγωγές) μόνο από το ένα άκρο της, που ονομάζεται ***κορυφή***. Η ***ουρά***, ένας άλλος τύπος ειδικής λίστας, επιτρέπει εισαγωγές από το ένα άκρο, που ονομάζεται ***πίσω*** ή ***τέλος***, και διαγραφές (εξαγωγές) από το άλλο, που ονομάζεται ***εμπρός*** ή ***αρχή***.

Η στοίβα υπακούει στη ***λογική LIFO***, δηλαδή όποιο στοιχείο εισάγεται τελευταίο, αυτό διαγράφεται πρώτο. Η ουρά υπακούει στη ***λογική FIFO***, δηλαδή όποιο στοιχειό εισάγεται πρώτο, αυτό και εξάγεται πρώτο.

Η στοίβα και η ουρά μπορούν να αναπαρασταθούν είτε με συνεχόμενη είτε με συνδεδεμένη αναπαράσταση και βρίσκουν διάφορες εφαρμογές στην επιστήμη των Η/Υ. Κυριότερες πράξεις στις δομές αυτές είναι η εισαγωγή και η διαγραφή.

5ο ΚΕΦΑΛΑΙΟ

ΔΕΝΤΡΑ

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό παρουσιάζουμε τα δέντρα, ίσως τη σπουδαιότερη κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη κατηγορία δίνεται η περιγραφή ενός γενικού δέντρου, καθώς και ορισμένων βασικών του στοιχείων.

Στη δεύτερη ενότητα εξετάζουμε τα δυαδικά δέντρα, τη σπουδαιότερη κατηγορία δέντρων. Λίνεται ο ορισμός ενός δυαδικού δέντρου, περιγράφονται οι τρόποι αναπαράστασης του και οι διάφοροι τρόποι διαπέρασης, καθώς και ο αλγόριθμος για την προδιατεταγμένη διαπέραση.

5.1 Γενικά στοιχεία και ορισμοί

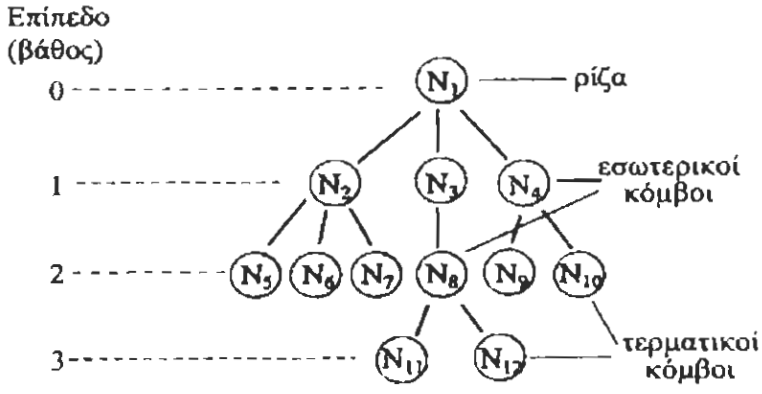
Τα δέντρα είναι από τις σπουδαιότερες δομές δεδομένων και ανήκουν στην κατηγορία των μη γραμμικών δομών. Ένα δέντρο είναι μια συλλογή από στοιχεία (κόμβους) του ίδιου τύπου που συνδέονται (σχετίζονται) μεταξύ τους κατά τρόπο πιο πολύπλοκο απ' ότι στους πίνακες και στις λίστες, που είναι γραμμικές δομές. Η συνήθης απεικόνιση του τρόπου αυτού συσχέτισης των κόμβων μοιάζει με δέντρο. Στο σχήμα 1 παρουσιάζεται ένα δέντρο γενικής μορφής.

Κάθε δέντρο αποτελείται από κόμβους (notes) και ακμές (edges).Οι ακμές είναι ευθύγραμμα τμήματα που συνδέουν τους κόμβους μεταξύ τους. Σε κάθε δέντρο υπάρχει ένας και μοναδικός κόμβος που ονομάζεται ρίζα(root) του δέντρου. Από τη ρίζα μόνο ξεκινούν ακμές. Δεν υπάρχει ακμή που να καταλήγει στη ρίζα. Οι κόμβοι στους οποίους μόνο καταλήγουν ακμές ονομάζονται τερματικοί κόμβοι(terminal notes) ή φύλλα (leaves).Οι υπόλοιποι κόμβοι, στους οποίους καταλήγουν και από τους οποίους ξεκινούν

ακμές, ονομάζονται εσωτερικοί(internal) ή μη τερματικοί(non terminal) κόμβοι. Το κενό δέντρο(null tree) δεν έχει κανέναν κόμβο και καμία ακμή

Για παράδειγμα στο δέντρο του σχήματος 1 ο κόμβος N_1 είναι η ρίζα του δέντρου, οι N_2,N_3,N_4 και N_8 είναι εσωτερικοί κόμβοι και οι υπόλοιποι τερματικοί κόμβοι. Τέλος, υπόδεντρο (subtree) ενός δέντρου είναι κάθε δέντρο που σχηματίζεται αν θεωρήσουμε ως ρίζα έναν οποιοδήποτε κόμβο του δέντρου.

ΣΧΗΜΑ 1



Σ' ένα δέντρο ορίζουμε τις παρακάτω «οικογενειακές» σχέσεις μεταξύ των κόμβων του. Ένας κόμβος N_X ονομάζεται γονέας(parent) ενός άλλου κόμβου N_Y αν από τον N_X ξεκινά μια ακμή που καταλήγει στον N_Y.Ο N_Y τότε ονομάζεται παιδί (child) του N_X. Τα φύλλα ενός δέντρου δεν έχουν παιδιά. Π.χ. στο σχήμα 5.1 ο κόμβος N_4 είναι γονέας των N_9,N_{10} και οι N_9,N_{10} είναι παιδιά του. Τα παιδιά ενός κόμβου, π.χ. τα N_9,N_{10}είναι μεταξύ τους αδέρφια(siblings). Κατ'επέκταση,
έχουμε και τους όρους απογόνος (descendant) και πρόγονος(ancestor).Π.χ. ο κόμβος N_{12} είναι απόγονος του N_3,διότι είναι παιδί του παιδιού του (δηλαδή του N_8). Αντίστοιχα, ο κόμβος N_3 είναι πρόγονος του N_{12}.

Επίσης, σ'ένα δέντρο ορίζουμε και τις έννοιες που παρουσιάζονται στον Πίνακα 1.Τα παραδείγματα αναφέρονται στο δέντρο του Σχήματος 1.

Πινακας 1

Ορισμός	**Παράδειγμα –Παρατηρήσεις**
Διαδρομή (path) από έναν κόμβο N_x Σε έναν άλλο κόμβο N_y είναι μια ακολουθία κόμβων, όπου πρώτος είναι ο N_x, τελευταίος ο N_y και κάθε άλλος κόμβος είναι παιδί του προηγουμένου.	Η διαδρομή από τον κόμβο N_1 στον κόμβο N_{12} είναι η N_1-N_3-N_8-N_{12}. Σε ένα δέντρο υπάρχει μία και μοναδική διαδρομή από τη ρίζα σε οποιονδήποτε κόμβο.
Μήκος (length) μιας διαδρομής είναι ο αριθμός των ακμών που περιλαμβάνονται σ'αυτήν.	Το μήκος της παραπάνω διαδρομής είναι τρία (3), διότι περιλαμβάνει τρεις ακμές.
Επίπεδο (level) ή *βάθος (depth)* ενός κόμβου είναι το μήκος της μοναδικής διαδρομής από τη ρίζα στον κόμβο αυτό.	Το επίπεδο (βάθος) του κόμβου N_8 είναι δύο (2), διότι η διαδρομή από τη ρίζα στον κόμβο έχει μήκος δύο (2). Οι κόμβοι είναι οργανωμένοι σε επίπεδα. Η ρίζα έχει βάθος μηδέν (0), δηλαδή βρίσκεται στο επίπεδο μηδέν (0).
Ύψος (height) ενός δέντρου είναι το μέγιστο βάθος των (τερματικών) κόμβων του.	Το ύψος του δέντρου του σχήματος είναι τρία (3), διότι αυτό είναι το μέγιστο βάθος των κόμβων του. Το ύψος του κενού δέντρου είναι -1.
Βαθμός (degree) ενός κόμβου είναι ο αριθμός των παιδιών του.	Ο βαθμός του κόμβου N_2 είναι τρία (3), ενώ του N_3 είναι ένα (1). Ο βαθμός ενός τερματικού κόμβου είναι μηδέν.

5.2 Δυαδικά δέντρα

5.2.1 Ορισμοί και αναπαράσταση

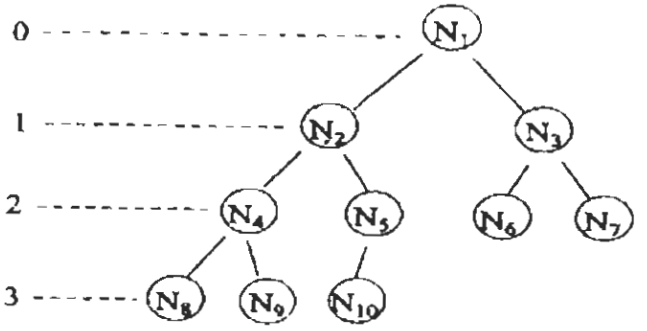
Ένα δυαδικό δέντρο είναι ένα δέντρο με βαθμό δύο (2) για κάθε κόμβο του.

Δηλαδή κάθε κόμβος σ'ένα δυαδικό δέντρο έχει, κατά μέγιστο, δύο παιδιά, το αριστερό παιδί και το δεξιό παιδί, που είναι αντίστοιχα οι ρίζες του αριστερού και του δεξιού υπόδεντρου του κόμβου. Αν κάποιο από τα δύο παιδιά είναι το κενό δέντρο, τότε λέμε ότι το αντίστοιχο παιδί δεν υπάρχει ή είναι κενό.

Ένα δυαδικό δέντρο απεικονίζεται στο σχήμα 2. Το αριστερό παιδί του κόμβου N_1 είναι ο κόμβος N_2 και το δεξιό παιδί ο κόμβος N_3.Το αριστερό παιδί του N_5 είναι ο N_{10}, αλλά το δεξιό του παιδί είναι το κενό δέντρο. Οι τερματικοί κόμβοι έχουν ως αριστερό και δεξιό παιδί τους το κενό δέντρο.

Στο επίπεδο 0 (ρίζα) ο μέγιστος αριθμός κόμβων είναι $2^0=1$, στο επίπεδο 1 είναι $2^1=2$, στο επίπεδο 2 είναι $2^2=4$κ.ο.κ., στο επίπεδο(βάθος) K είναι 2^K. Επομένως, ένα δυαδικό δέντρο ύψους h μπορεί να έχει κατά μέγιστο $2^0+2^1+2^2+...+2^h = \Sigma\ 2^K=2^{h+1}-1$ κόμβους.

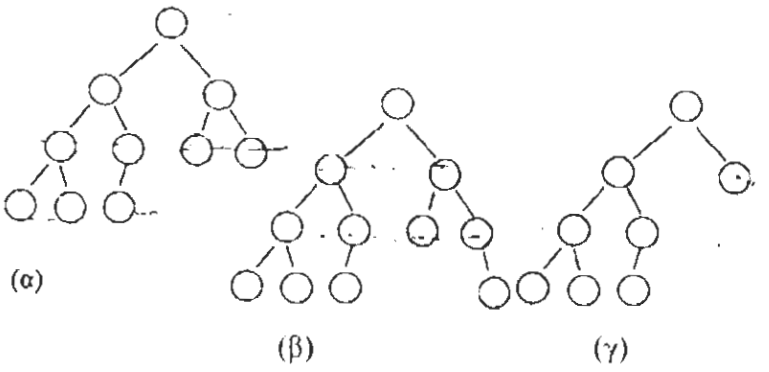
ΣΧΗΜΑ 2



Ένα δυαδικό δέντρο λέγεται πλήρες(complete), αν σε όλα τα επίπεδά του, εκτός από το τελευταίο, έχει το μέγιστο δυνατό αριθμό κόμβων και όλοι οι κόμβοι στο τελευταίο επίπεδο βρίσκονται όσο το δυνατό στα αριστερά του δέντρου. Επομένως,

υπάρχει ένα μοναδικό πλήρες δέντρο με ακριβώς n κόμβους. Π.χ. από τα δέντρα του σχήματος 3 μόνο το (α) είναι πλήρες. Το (β) δεν είναι πλήρες, διότι οι κόμβοι στο τελευταίο επίπεδο (τερματικοί) δε βρίσκονται όσο το δυνατόν αριστερά. Το (γ) δεν είναι επίσης πλήρες, διότι στο πρώτο επίπεδο, που δεν είναι τελευταίο, δεν έχει το μέγιστο δυνατό αριθμό κόμβων.

ΣΧΗΜΑ 3



Για το ύψος h ενός πλήρους δέντρου n κόμβων ισχύει (πάνω όριο) η σχέση:

$$h \leq \log n.$$

Η απόδειξη έχει ως ακολούθως. Ο ελάχιστος αριθμός κόμβων σε ένα πλήρες δέντρο ύψους h είναι:

$$(2^0+2^1+2^2+\ldots+2^{h-1})+1,$$

δηλαδή συμπληρωμένα τα επίπεδα μέχρι ύψος (h-1) συν ένα κόμβο από το τελευταίο επίπεδο. Αλλά $(2^0+2^1+2^2+\ldots+2^{h-1})=2^h-1$, οπότε ο ελάχιστος αριθμός κόμβων γίνεται τελικά $(2^h-1)+1=2^h$. Δηλαδή για ένα δέντρο n στοιχείων, είναι $n \geq 2^h <=> \log n \geq \log(2^h) <=> h \leq \log n$.

Επίσης, ισχύει(κάτω όριο) η σχέση:

$$h \geq \log(n+1)-1.$$

Η απόδειξη έχει ως ακολούθως: Αφού ο μέγιστος αριθμός κόμβων ενός πλήρους δέντρου ύψους h είναι $2^{h+1}-1$, για ένα δέντρο n στοιχείων θα ισχύει $n<2^{h+1}-1 <=> n+1 \leq 2^{h+1} <=> \log(n+1) \leq \log(2^{h+1}) <=> h+1 \geq \log(n+1) <=> h \geq \log(n+1)-1$.

Όπως και οι προηγούμενες δομές, έτσι και ένα δυαδικό δέντρο μπορεί να αναπαρασταθεί χρησιμοποιώντας είτε συνεχόμενη είτε συνδεδεμένη αναπαράσταση. Κοινές απαιτήσεις και στις δύο περιπτώσεις είναι α) να υπάρχει άμεση προσπέλαση της ρίζας του δέντρου και β) δεδομένου ενός κόμβου να υπάρχει άμεση προσπέλαση στα παιδιά του.

Σύνοψη

Τα δέντρα είναι από τις σπουδαιότερες δομές δεδομένων και ανήκουν στην κατηγορία των μη γραμμικών δομών. Κάθε δέντρο αποτελείται από κόμβους (notes) και ακμές (edges).

Δυαδικά δέντρα είναι τα δέντρα με βαθμό (2) για κάθε κόμβο τους. Υπάρχουνε δύο κατηγορίεσ δυαδικών δέντρων τα πλήρες και τα μη πλήρες

ΒΙΒΛΙΟΓΡΑΦΙΑ

Α. ΕΛΛΗΝΙΚΗ

- Μανωλόπουλος Ι., « Δομές Δεδομένων », τόμος Α 2η έκδοση Θεσσαλονίκη 1992.
- Κοίλιας Χ., « Δομές Δεδομένων και Οργανώσεις Αρχείων» Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1993.
- Πεσεξίδης Χ. Νικόλαος, « Turbo Pascal », εκδόσεις Κονιδάρη Αθήνα 1994.

Β. ΞΕΝΗ

- Stubbs, F. D. and Webre, W.N., « Data structures with Abstract Data Types and Pascal », 2nd Edition, Brooks/Cole, Pasific Grove, CA 1989.
- Kruse, L. R., « Data Structures and Program Design », 2nd Edition, Prentice Hall, Englewood Cliffs, NJ 1991.
- T.A. Standish, « Data Structures, Algorithms and Software Principles », Addison- Wesley, 1994.
- M. A. Weiss, « Data Structures and algorithm Analysis », Benjamin/Cummings, 2nd Edition, 1995.
- M. Main and W. Savitch, « Data Structures and Other Objects » Benjamin/Cummings, 1995.
- C. A. Shaffer, « A Practical Introduction to Data Structures and Algorithm Analysis », Prentice Hall, 1997.