ΣΤΟΙΒΑ

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

Η παραπάνω μέθοδος ονομάζεται **Τελευταίο Μέσα, Πρώτο Έξω ή LIFO** (=Last In First Out).
Μπορούμε να φανταστούμε την τοποθέτηση των στοιχείων μιας στοίβας σε κατακόρυφη σειρά.
Χαρακτηριστικό παράδειγμα είναι μια στοίβα από πιάτα. Παίρνουμε προς χρήση το πιάτο που τοποθετήσαμε τελευταίο.

Οι **κύριες λειτουργίες** σε μια στοίβα είναι δύο:

1. Η **ώθηση** (push) στοιχείου στην κορυφή της στοίβας.
Στη διαδικασία της ώθησης ελέγχουμε αν η στοίβα είναι γεμάτη. Στην περίπτωση που προσπαθήσουμε να «προσθέσουμε» ένα στοιχείο σε μια ήδη γεμάτη στοίβα, έχουμε **υπερχείλιση** (overflow) της στοίβας.

2. Η **απώθηση** (pop) στοιχείου από τη στοίβα.
Στη διαδικασία της απώθησης ελέγχουμε αν υπάρχει ένα τουλάχιστον στοιχείο στη στοίβα. Στην περίπτωση που προσπαθήσουμε να «αφαιρέσουμε» ένα στοιχείο από μία κενή στοίβα, έχουμε **υποχείλιση** (underflow) της στοίβας.

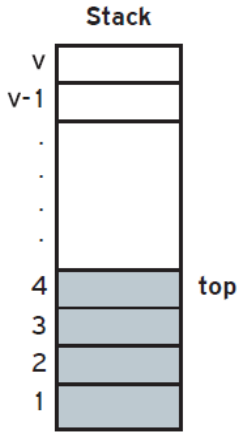
ΥΛΟΠΟΙΗΣΗ ΣΤΟΙΒΑΣ ΜΕ ΧΡΗΣΗ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

- Χρησιμοποιούμε μια βοηθητική μεταβλητή (top), που δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας.
- Η **ώθηση** ενός νέου στοιχείου στη στοίβα (εισαγωγή στοιχείου στον πίνακα) γίνεται πάντα στην κορυφή της. Συγκεκριμένα, η μεταβλητή top αυξάνεται κατά ένα:

$$top \leftarrow top+1$$

και στη συνέχεια γίνεται η ώθηση του στοιχείου.
- Η **απώθηση** ενός στοιχείου από τη στοίβα (εξαγωγή από τον πίνακα) γίνεται πάντα από την κορυφή της στοίβας. Συγκεκριμένα, εξάγεται το στοιχείο που δείχνει η μεταβλητή top και στη συνέχεια η μεταβλητή top μειώνεται κατά ένα:

$$top \leftarrow top-1$$



Σε μια κενή στοίβα/πίνακα θεωρούμε ότι η αρχική τιμή της μεταβλητής top είναι μηδέν (top←0). Η μεταβλητή top είναι η μεταβλητή που δείχνει τη θέση που τοποθετήθηκε το τελευταίο στοιχείο στη στοίβα/πίνακα (δηλ. δείχνει την κορυφή της στοίβας).
Κατά την **ώθηση** ενός στοιχείου στη στοίβα (εισαγωγή ενός στοιχείου στον πίνακα), πρώτα αυξάνεται η τιμή της μεταβλητής top κατά ένα, δηλ. top←top+1, και στη συνέχεια γίνεται η ώθηση του στοιχείου στην κορυφή της στοίβας.
Κατά την **απώθηση** ενός στοιχείου από τη στοίβα (εξαγωγή στοιχείου από τον πίνακα) μειώνεται η τιμή της μεταβλητής top κατά ένα, δηλ. top← top-1. Στην απώθηση δε διαγράφεται το στοιχείο, στην πραγματικότητα δε γίνεται καμία παρέμβαση στα περιεχόμενα του πίνακα. Απλώς ο δείκτης κορυφή δείχνει στην προηγούμενη θέση.

ΩΘΗΣΗ ΣΤΟΙΧΕΙΟΥ ΣΤΗΝ ΚΟΡΥΦΗ ΤΗΣ ΣΤΟΙΒΑΣ ΜΕ ΧΡΗΣΗ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

Να αναπτύξετε τμήμα προγράμματος σε ΓΛΩΣΣΑ, που πραγματοποιεί ώθηση στοιχείου στην κορυφή της στοίβας με χρήση μονοδιάστατου πίνακα Α, 10 θέσεων.

```
1    ΓΡΑΨΕ 'Δώσε στοιχείο για να εισαχθεί στη στοίβα Α:'
2    ΔΙΑΒΑΣΕ στοιχείο
3    ΑΝ top<10 ΤΟΤΕ
4          top<- top+1
5          A[top]<- στοιχείο
6    ΑΛΛΙΩΣ
7          ΓΡΑΨΕ 'Υπερχείλιση στοίβας'
8    ΤΕΛΟΣ_ΑΝ
```

ΑΠΩΘΗΣΗ ΣΤΟΙΧΕΙΟΥ ΑΠΟ ΣΤΟΙΒΑ ΜΕ ΧΡΗΣΗ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

Να αναπτύξετε τμήμα προγράμματος σε ΓΛΩΣΣΑ που πραγματοποιεί την απώθηση στοιχείου από στοίβα με χρήση ενός μονοδιάστατου πίνακα Α, 10 θέσεων.

```
1    ΑΝ top>=1  ΤΟΤΕ
2          ΓΡΑΨΕ A[top]
3          top<- top-1
4    ΑΛΛΙΩΣ
5          ΓΡΑΨΕ 'Υποχείλιση στοίβας'
6    ΤΕΛΟΣ_ΑΝ
```

ΟΥΡΑ

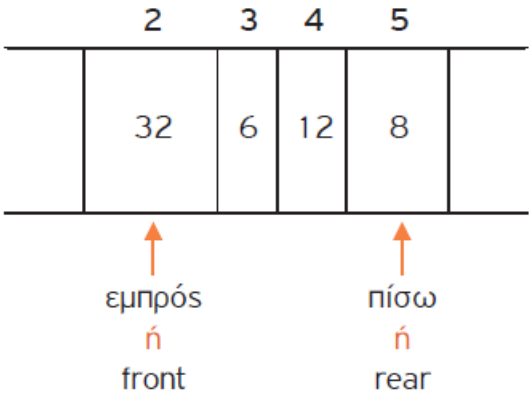
Ουρά (Queue), ονομάζεται μια δομή δεδομένων το σύνολο των στοιχείων της οποίας είναι διατεταγμένο με τέτοιο τρόπο, ώστε τα στοιχεία που τοποθετήθηκαν πρώτα στην ουρά να λαμβάνονται επίσης πρώτα.

Η παραπάνω μέθοδος ονομάζεται **Πρώτο Μέσα, Πρώτο Έξω ή FIFO** (=First In First Out).
Μπορούμε να φανταστούμε την τοποθέτηση των στοιχείων μιας ουράς σε οριζόντια σειρά. Ο πελάτης μιας τράπεζας που μπαίνει πρώτος σε μια ουρά για να εξυπηρετηθεί, είναι αυτός που εξυπηρετείται και πρώτος, με την έξοδό του από την ουρά αναμονής. Άλλα παραδείγματα είναι η ουρά στα λεωφορεία ή η ουρά στα ταμεία, όπου ο πρώτος που στέκεται
στην ουρά εξυπηρετείται και πρώτος.

Οι **κύριες λειτουργίες** που εκτελούνται σε μια ουρά είναι δύο:
1. Η **εισαγωγή** (enqueue) στοιχείου στο πίσω άκρο της ουράς.
2. Η **εξαγωγή** (dequeue) στοιχείου από το εμπρός άκρο της ουράς.

Υλοποίηση ουράς με χρήση μονοδιάστατου πίνακα

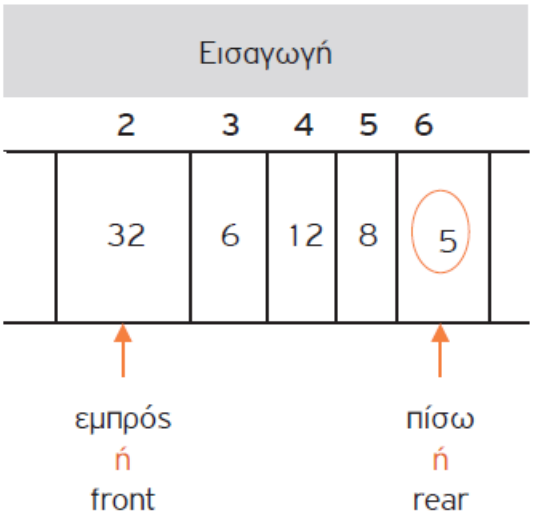
- Χρησιμοποιούμε δύο μεταβλητές, την front (ή εμπρός) που δείχνει τη θέση του 1ου στοιχείου της ουράς και την rear (ή πίσω) που δείχνει τη θέση του τελευταίου στοιχείου. Ως αρχικές τιμές των μεταβλητών rear και front θεωρούμε το μηδέν.



- Η **εισαγωγή** ενός νέου στοιχείου γίνεται από το πίσω άκρο της ουράς και η τιμή της μεταβλητής rear αλλάζει ως εξής:

  rear← rear+1

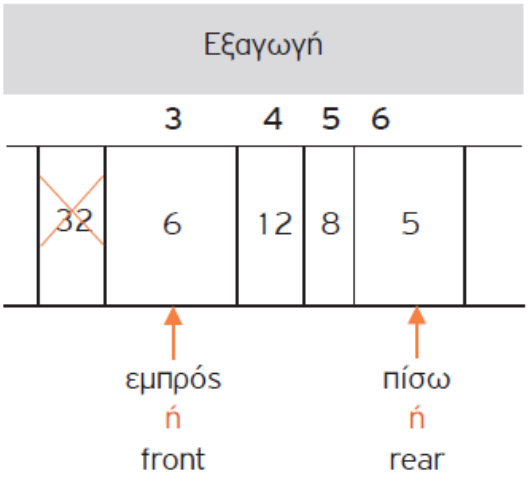
  Κατά την εισαγωγή, πρώτα αυξάνουμε τον δείκτη rear κατά ένα και μετά εισάγουμε το στοιχείο στον πίνακα.



- Η **εξαγωγή** ενός στοιχείου γίνεται από το εμπρός άκρο της ουράς και η τιμή της μεταβλητής front αλλάζει ως εξής:

  front ← front +1

  Κατά την εξαγωγή ενός στοιχείου, αυξάνεται ο δείκτης front κατά ένα (δείχνει στην επόμενη θέση του πίνακα) χωρίς στην πραγματικότητα να γίνεται καμία παρέμβαση στα περιεχόμενα του πίνακα (χωρίς να διαγράφεται κάποιο στοιχείο).



ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ ΣΕ ΟΥΡΑ ΜΕ ΧΡΗΣΗ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

Να αναπτύξετε τμήμα προγράμματος σε ΓΛΩΣΣΑ που υλοποιεί την εισαγωγή στοιχείου σε ουρά, με χρήση μονοδιάστατου πίνακα Α, 10 θέσεων.

```
ΓΡΑΨΕ 'Δώσε στοιχείο για εισαγωγή στην ουρά Α:'
ΔΙΑΒΑΣΕ στοιχείο
ΑΝ rear = 10 ΤΟΤΕ
   ΓΡΑΨΕ 'Γεμάτη ουρά'
ΑΛΛΙΩΣ_ΑΝ (front = 0 ΚΑΙ rear = 0) ΤΟΤΕ
   front<- 1
   rear<-1
   A[rear]<-στοιχείο
ΑΛΛΙΩΣ
      rear <- rear + 1
      A[rear]<-στοιχείο
ΤΕΛΟΣ_ΑΝ
```

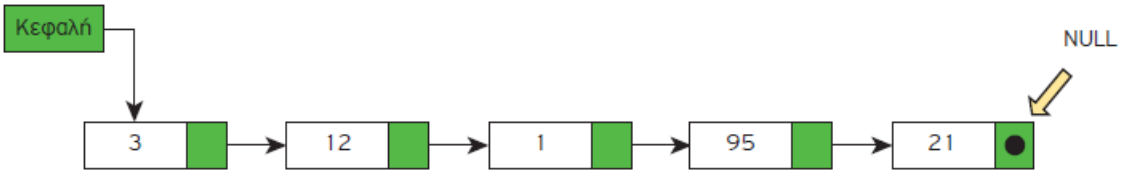
ΕΞΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ ΑΠΟ ΟΥΡΑ ΜΕ ΧΡΗΣΗ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

Να αναπτύξετε τμήμα προγράμματος σε ΓΛΩΣΣΑ που υλοποιεί την εξαγωγή στοιχείου από ουρά, με χρήση μονοδιάστατου πίνακα Α, 10 θέσεων.

```
ΑΝ (front = 0 ΚΑΙ rear = 0) ΤΟΤΕ
    ΓΡΑΨΕ 'Άδεια ουρά'
ΑΛΛΙΩΣ ΑΝ (front = rear ) ΤΟΤΕ
    ΓΡΑΨΕ 'Εξάγεται το στοιχείο:',A[front]
    front <-  0
    rear  <-  0
ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Εξάγεται το στοιχείο:',A[front]
    front <- front + 1
ΤΕΛΟΣ_ΑΝ
```

ΛΙΣΤΕΣ

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



Εικόνα 1.3.3. Μία απλά συνδεδεμένη λίστα με 5 κόμβους.

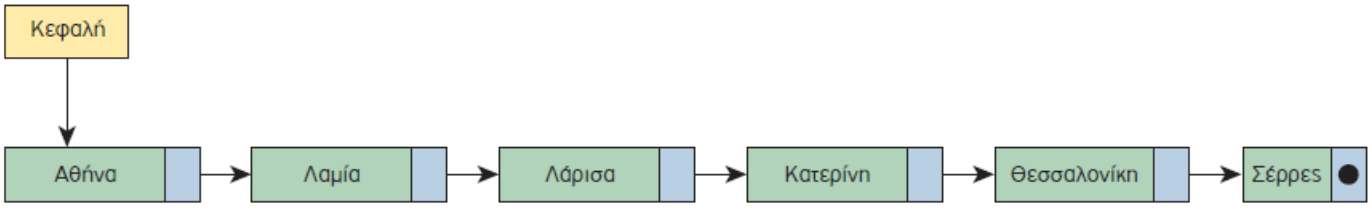
Οι κόμβοι μιας λίστας δεν έχουν ονόματα. Γνωρίζουμε μόνο τις διευθύνσεις τους, που είναι αποθηκευμένες στους προηγούμενους κόμβους και αυτές θα αξιοποιήσουμε για να τους προσπελάσουμε. Επομένως, αν θελήσουμε να έχουμε πρόσβαση στον τέταρτο κόμβο μιας λίστας, για να επεξεργαστούμε τα δεδομένα που περιέχει, θα πρέπει να ξεκινήσουμε από τον πρώτο κόμβο της λίστας, η διεύθυνση του οποίου περιέχεται στον δείκτη **Κεφαλή**. Ξεκινώντας από τον πρώτο κόμβο της λίστας μπορούμε να έχουμε πρόσβαση στη διεύθυνση μνήμης του δεύτερου κόμβου. Από τον δεύτερο κόμβο μπορούμε να έχουμε πρόσβαση στη διεύθυνση μνήμης του τρίτου κόμβου. Και συνεχίζουμε έτσι μέχρι να φτάσουμε στον τέταρτο κόμβο. Αυτός είναι ο μόνος τρόπος για να διασχίσουμε μία συνδεδεμένη λίστα.

ΠΡΟΣΒΑΣΗ ΣΤΟΥΣ ΚΟΜΒΟΥΣ ΜΙΑΣ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ

Οι κόμβοι μιας (απλά) συνδεδεμένης λίστας είναι διατεταγμένοι σε μια συγκεκριμένη σειρά, χωρίς αυτό να σημαίνει ότι αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη. Αντίθετα, είναι διασκορπισμένοι σε όλη τη μνήμη και η σύνδεση μεταξύ τους γίνεται μέσω των δεικτών. Έχουμε άμεση πρόσβαση μόνο στον πρώτο κόμβο της λίστας. Επομένως, για να εντοπίσουμε κάποιον από τους ενδιάμεσους κόμβους, πρέπει να ξεκινήσουμε από τον πρώτο κόμβο της λίστας και να ακολουθήσουμε τους δείκτες με τη σειρά, μέχρι να φτάσουμε στον επιθυμητό κόμβο.

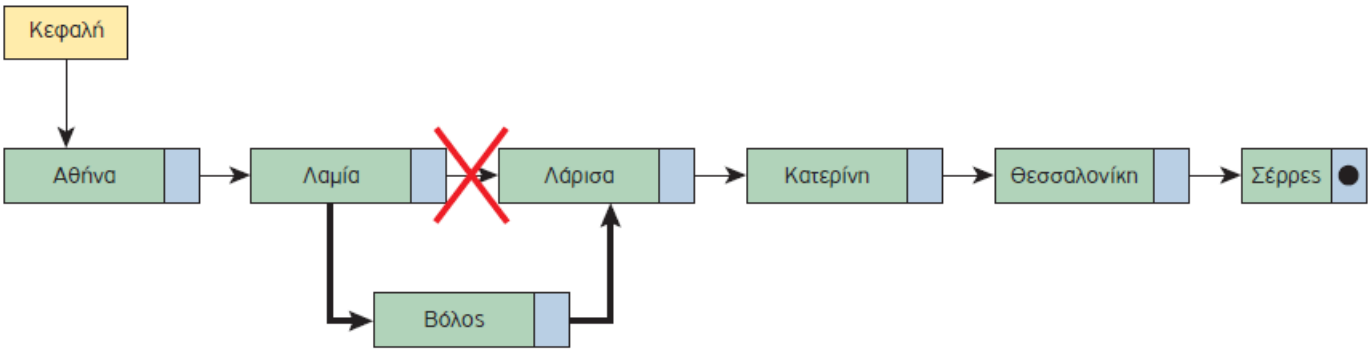
ΕΙΣΑΓΩΓΗ ΚΑΙ ΔΙΑΓΡΑΦΗ ΚΟΜΒΟΥ ΣΕ ΜΙΑ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Έστω ότι σχεδιάζουμε στη συνέχεια ένα ταξίδι στην Κεντρική Μακεδονία. Η αφετηρία μας είναι η Αθήνα και το τέρμα του ταξιδιού είναι οι Σέρρες. Έχουμε αποθηκεύσει σε μια συνδεδεμένη λίστα τις πόλεις που είναι πιθανό να επισκεφθούμε (Εικόνα 1.3.4).



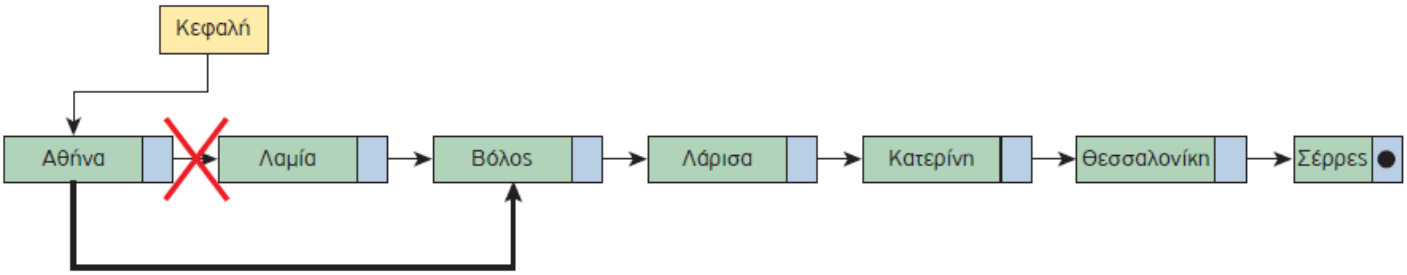
Εικόνα 1.3.4. Μια απλά συνδεδεμένη λίστα

Λίγο πριν το ταξίδι, αποφασίζουμε να επισκεφθούμε τον Βόλο, αμέσως μετά τη Λαμία και πριν φτάσουμε στη Λάρισα. Επομένως, χρειάζεται να εισαγάγουμε ένα νέο κόμβο στη λίστα μας, κάνοντας την κατάλληλη διευθέτηση των δεικτών (Εικόνα 1.3.5).



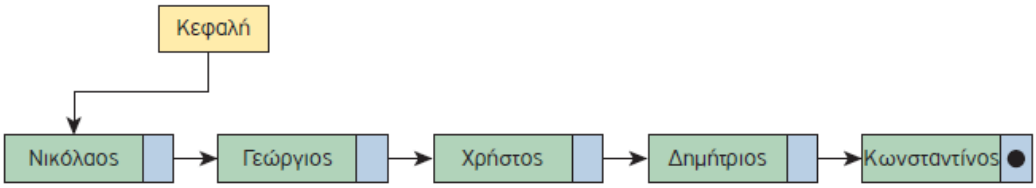
Εικόνα 1.3.5. Εισαγωγή νέου κόμβου στη λίστα

Όπως φαίνεται και στην Εικόνα 1.3.5, οι απαιτούμενες ενέργειες για την εισαγωγή (παρεμβολή) του νέου κόμβου είναι ο δείκτης του δεύτερου κόμβου να δείχνει τον νέο κόμβο και ο δείκτης του νέου κόμβου να δείχνει τον τέταρτο κόμβο (δηλαδή, να πάρει την τιμή που είχε πριν την εισαγωγή ο δείκτης του τρίτου κόμβου). Με αυτόν τον τρόπο, οι κόμβοι της λίστας διατηρούν τη λογική τους σειρά, αλλά οι φυσικές θέσεις στη μνήμη μπορεί να είναι τελείως διαφορετικές.
Αν, επιπλέον, αποφασίσουμε να μην επισκεφθούμε τη Λαμία, θα πρέπει να διαγράψουμε τον παραπάνω κόμβο από τη λίστα κάνοντας μια εκ νέου διευθέτηση των αντίστοιχων δεικτών (Εικόνα 1.3.6). Έτσι, για τη διαγραφή ενός κόμβου αρκεί ν' αλλάξει τιμή ο δείκτης του προηγούμενου κόμβου και να δείχνει πλέον στον επόμενο αυτού που διαγράφεται, όπως φαίνεται στην Εικόνα 1.3.6. Ο κόμβος που διαγράφηκε (ο δεύτερος) αποτελεί «άχρηστο δεδομένο» και ο χώρος μνήμης που καταλάμβανε, παραχωρείται για άλλη χρήση.



Εικόνα 1.3.6. Διαγραφή κόμβου από τη λίστα

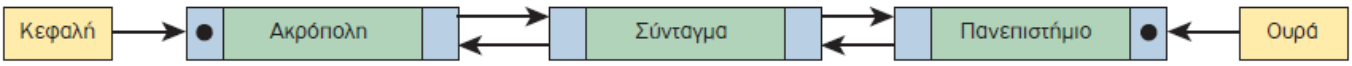
Ας δούμε ένα δεύτερο παράδειγμα με μία συνδεδεμένη λίστα. Ας φανταστούμε έναν αγώνα σκυταλοδρομίας (Εικόνα 1.3.7). Οι αθλητές που λαμβάνουν μέρος είναι τοποθετημένοι με μια συγκεκριμένη σειρά και ο κάθε αθλητής για να ξεκινήσει να τρέχει πρέπει να περιμένει τον προηγούμενο του. Σε περιπτώσεις όπως αυτή, που κάθε στοιχείο συνδέεται με το αμέσως επόμενο, χρησιμοποιούμε τη δομή δεδομένων της συνδεδεμένης λίστας για να αποθηκεύσουμε τις τιμές.



Εικόνα 1.3.7. Το παράδειγμα των αθλητών της σκυταλοδρομίας - Απλά συνδεδεμένη λίστα

Οι συνδεδεμένες λίστες αξιοποιούνται για την υλοποίηση της στοίβας και της ουράς, λόγω της δυνατότητάς αυξομείωσης του μεγέθους τους. Μπορείτε να εξηγήσετε γιατί;

Μια λίστα μπορεί να είναι **απλά συνδεδεμένη**, όπως στο παράδειγμα της σκυταλοδρομίας, δηλαδή να μπορούμε να κινηθούμε προς μία μόνο κατεύθυνση, ξεκινώντας από τον πρώτο κόμβο και μετακινούμενοι προς τον τελευταίο, όπως δείχνουν τα βέλη του σχήματος ή να είναι **διπλά συνδεδεμένη (doubly linked list)**, δηλαδή, να μπορούμε να τη διατρέξουμε και προς τις δύο κατευθύνσεις. Η χρήση του δεύτερου δείκτη προσφέρει τη δυνατότητα ξεκινώντας από οποιοδήποτε κόμβο της λίστας να μπορούμε να διαβάσουμε τη λίστα και προς τις δυο κατευθύνσεις. Ένα τέτοιο παράδειγμα είναι οι σταθμοί του μετρό.



Εικόνα 1.3.8. Οι σταθμοί του μετρό - Μια διπλά συνδεδεμένη λίστα

Όπως φαίνεται και στην Εικόνα 1.3.8., κάθε κόμβος της διπλά συνδεδεμένης λίστας, συνδέεται με τον αμέσως επόμενο και τον αμέσως προηγούμενο κόμβο της λίστας. Εκτός, βέβαια, από τον αρχικό και τον τελευταίο κόμβο της λίστας.

Διαφορές Λίστας σε σχέση με τον Πίνακα – Πλεονεκτήματα – Μειονεκτήματα

Συνοψίζοντας, θα μπορούσε κάποιος πρόχειρα να συμπεράνει ότι η λίστα δε διαφέρει από τον πίνακα. Παρατηρώντας όμως πιο προσεκτικά, είναι εύκολο να εντοπίσει σημαντικές **διαφορές** μεταξύ τους, οι οποίες παρατίθενται παρακάτω:

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

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

- Το δυναμικό τους μέγεθος,
- η ευκολία εισαγωγής και διαγραφής από οποιοδήποτε μέρος της λίστας, καθώς και
- η μη αναγκαιότητα δήλωσης του μεγέθους τους.

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

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

Βασικές πράξεις των συνδεδεμένων λιστών

Οι βασικές πράξεις των συνδεδεμένων λιστών είναι οι παρακάτω:

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

ΔΕΝΤΡΑ

Τα δένδρα είναι μία μη-γραμμική ευέλικτη δομή δεδομένων που χρησιμοποιούνται σε πολλούς τομείς της επιστήμης των υπολογιστών, συμπεριλαμβανομένων των λειτουργικών συστημάτων, των γραφικών, των συστημάτων βάσεων δεδομένων, των παιχνιδιών, της τεχνητής νοημοσύνης και της δικτύωσης υπολογιστών.

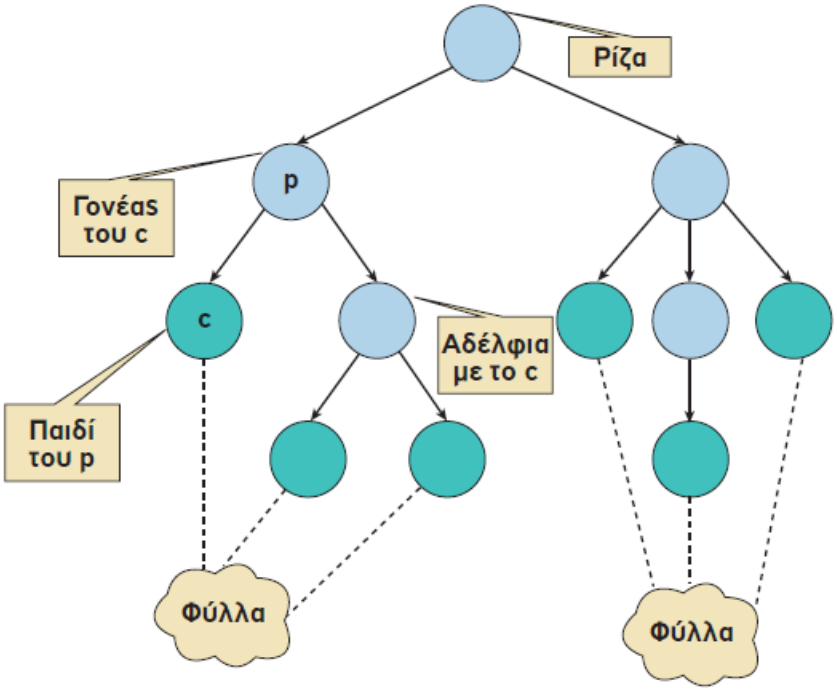
Ένα **δένδρο (tree)** είναι μία δομή που αποτελείται από ένα σύνολο κόμβων και ένα σύνολο ακμών μεταξύ των κόμβων με βάση τους εξής κανόνες:

• Υπάρχει ένας ξεχωριστός κόμβος που ονομάζεται ρίζα. Αυτός είναι ένας κόμβος χωρίς γονέα.

• Για κάθε κόμβο c, εκτός από τη ρίζα, υπάρχει μόνο μια ακμή που καταλήγει στον κόμβο αυτόν ξεκινώντας από κάποιον άλλον κόμβο p. Ο κόμβος p ονομάζεται γονέας του c και ο κόμβος c παιδί του p.

• Για κάθε κόμβο υπάρχει μία μοναδική διαδρομή, δηλαδή, μια ακολουθία διαδοχικών ακμών, που ξεκινάει από τη ρίζα και τερματίζει σε αυτόν τον κόμβο.

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

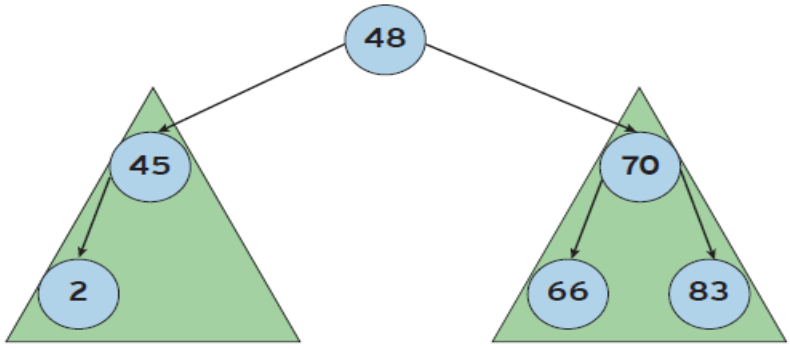


Εικόνα 1.3.11. Σχέσεις μεταξύ των κόμβων ενός δένδρου

Όταν δύο κόμβοι συνδέονται μεταξύ τους με μία ακμή, τότε ονομάζουμε «**γονέα**» τον κόμβο από τον οποίο ξεκινάει η ακμή και «**παιδί**» τον κόμβο στον οποίο καταλήγει η ακμή. Στην Εικόνα 1.3.11 ο κόμβος p είναι γονέας του κόμβου c και ο κόμβος c είναι παιδί του κόμβου p. Ένας κόμβος μπορεί να έχει κανένα, ένα ή περισσότερα παιδιά. Όλοι οι κόμβοι, εκτός από έναν, έχουν ακριβώς έναν γονέα. Ο κόμβος χωρίς γονέα ονομάζεται «**ρίζα**» (root) και βρίσκεται στην κορυφή του δένδρου. Κόμβοι με τον ίδιο γονέα ονομάζονται «**αδέλφια**». Οι κόμβοι χωρίς παιδιά ονομάζονται «**φύλλα**».

ΥΠΟΔΕΝΤΡΑ

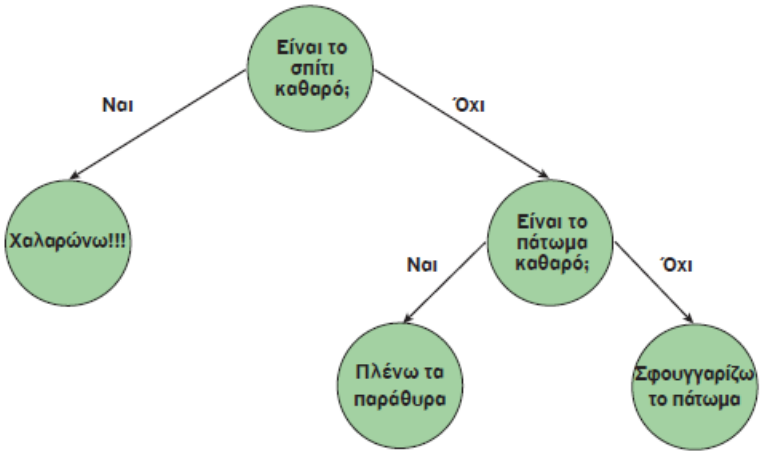
Μέσα σε ένα δένδρο μπορούμε να εντοπίσουμε και άλλα μικρότερα δένδρα, που ονομάζονται υποδένδρα. Πιο συγκεκριμένα, κάθε κόμβος ενός δένδρου μπορεί να θεωρηθεί ως ρίζα ενός υποδένδρου, δηλαδή ενός άλλου μικρότερου δένδρου, που ξεκινάει από τον κόμβο αυτόν. Στο δένδρο της Εικόνας 1.3.14, όπως φαίνεται και από το σχήμα, ο κόμβος 48 είναι ρίζα και έχει δύο υποδένδρα που ξεκινούν από τους κόμβους 45 και 70 αντίστοιχα. Ο κόμβος 45 έχει ένα υποδένδρο που αποτελείται από τον κόμβο 2. Ο κόμβος 70 έχει δύο υποδένδρα που αποτελούνται από τους κόμβους 66 και 83 αντίστοιχα. Τα υποδένδρα των κόμβων 2, 66 και 83 είναι κενά.



Εικόνα 1.3.14. Αριστερό και δεξιό υποδένδρο του κόμβου 48

ΔΕΝΤΡΑ ΑΠΟΦΑΣΗΣ

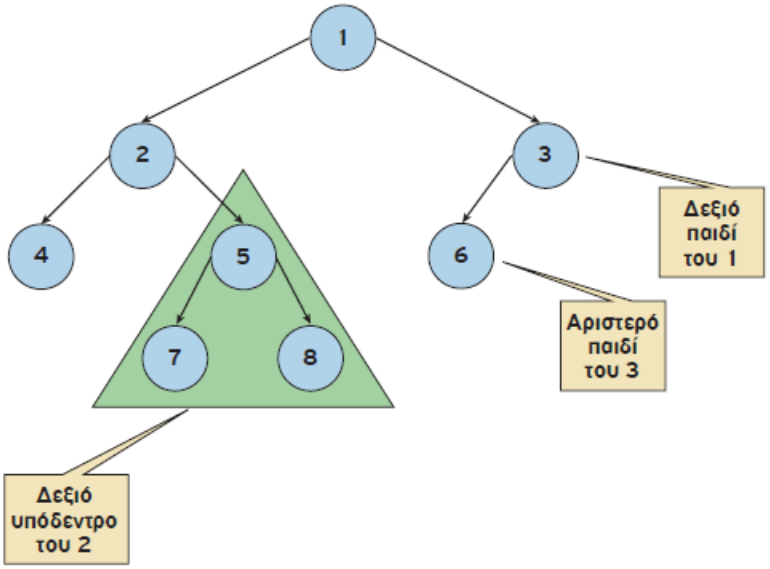
Τα δένδρα απόφασης, όπως πολύ εύκολα μπορείτε να συμπεράνετε από την Εικόνα 1.3.19, είναι δένδρα στα οποία κάθε κόμβος αντιπροσωπεύει ένα χαρακτηριστικό (ιδιότητα), κάθε ακμή αντιπροσωπεύει μια απόφαση (κανόνα) και κάθε φύλλο αντιπροσωπεύει ένα αποτέλεσμα. Στους αλγορίθμους μηχανικής μάθησης (machine learning) τα δένδρα απόφασης έχουν πρωτεύοντα ρόλο.



Εικόνα 1.3.19. Δένδρο Απόφασης

ΔΥΑΔΙΚΑ ΔΕΝΤΡΑ

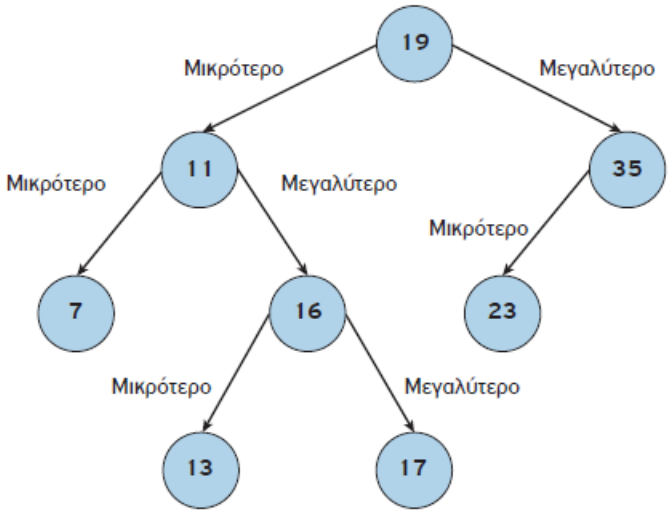
Ένα δυαδικό δένδρο (binary tree) είναι ένα διατεταγμένο δένδρο, στο οποίο κάθε κόμβος έχει το πολύ δύο παιδιά, το αριστερό και το δεξί παιδί. Μπορούμε, συνεπώς, να μιλάμε για αριστερό και δεξιό υποδένδρο ενός κόμβου. Στο δυαδικό δένδρο της Εικόνας 1.3.21, ο κόμβος 3 έχει ως αριστερό υποδένδρο, το δένδρο με μοναδικό κόμβο το 6 και ως δεξιό υποδένδρο, το κενό δένδρο. Προφανώς, αν ανταλλάξουμε το αριστερό με το δεξιό υποδένδρο ενός κόμβου παίρνουμε ένα διαφορετικό δένδρο.



Εικόνα 1.3.21. Δυαδικό δένδρο

ΔΥΑΔΙΚΑ ΔΕΝΤΡΑ ΑΝΑΖΗΤΗΣΗΣ

Ένα δυαδικό δένδρο αναζήτησης (binary search tree) είναι ένα δυαδικό δένδρο, όπου για κάθε κόμβο u, όλοι οι κόμβοι του αριστερού υποδένδρου έχουν τιμές μικρότερες της τιμής του κόμβου u και όλοι οι κόμβοι του δεξιού υποδένδρου έχουν τιμές μεγαλύτερες (ή ίσες) της τιμής του κόμβου u. Για λόγους απλούστευσης θεωρούμε ότι δεν υπάρχουν τιμές ίσες με την τιμή του κόμβου u. Στην Εικόνα 1.3.22 παρουσιάζεται το παράδειγμα ενός δυαδικού δένδρου αναζήτησης.



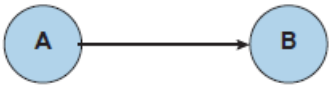
Εικόνα 1.3.22. Δυαδικό δένδρο αναζήτησης

ΓΡΑΦΟΙ

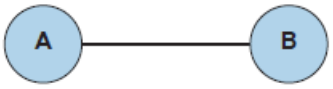
Ένας **γράφος (graph)** είναι μία δομή που αποτελείται από ένα σύνολο κόμβων (ή σημείων ή κορυφών) και ένα σύνολο γραμμών (ή ακμών ή τόξων) που ενώνουν μερικούς ή όλους τους κόμβους. Ο γράφος αποτελεί την πιο γενική δομή δεδομένων, με την έννοια ότι όλες οι προηγούμενες δομές που παρουσιάστηκαν μπορούν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

ΤΥΠΟΙ ΓΡΑΦΩΝ

Εάν όλες οι ακμές σε έναν γράφο έχουν κατεύθυνση, ο γράφος ονομάζεται **κατευθυνόμενος γράφος (directed graph)**.
Εάν όλες οι ακμές σε έναν γράφο δεν έχουν κατεύθυνση, ο γράφος ονομάζεται **μη κατευθυνόμενος γράφος (undirected graph).**



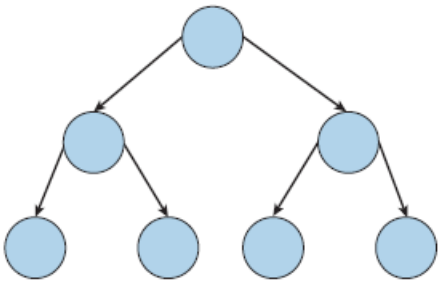
α: Κατευθυνόμενη ακμή

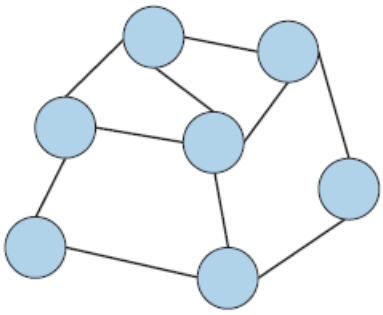


β: μη κατευθυνόμενη ακμή

ΔΙΑΦΟΡΕΣ ΔΕΝΤΡΟΥ ΑΠΟ ΓΡΑΦΟ

Ένα δένδρο μπορεί μόνο να ρέει προς μία κατεύθυνση, από τον κόμβο ρίζας σε κόμβους φύλλων ή κόμβους παιδιών. Ένα δένδρο μπορεί να έχει μόνο μονόδρομες συνδέσεις - ένας κόμβος παιδιού μπορεί να έχει μόνο έναν γονέα και ένα δένδρο δεν μπορεί να έχει βρόχους ή κυκλικούς δεσμούς (Εικόνα 1.3.27.α).
Με τους γράφους, όλοι αυτοί οι περιορισμοί δεν υπάρχουν. Οι γράφοι δεν έχουν την έννοια ενός κόμβου «ρίζας». Οι κόμβοι μπορούν να συνδεθούν με οποιονδήποτε τρόπο. Για παράδειγμα, ένας κόμβος μπορεί να συνδεθεί με άλλους πέντε (Εικόνα 1.3.27.β)! Οι γράφοι, επίσης, δεν έχουν «μονοκατευθυντική» ροή - αντ' αυτού, μπορεί να έχουν κατεύθυνση ή να μην έχουν καμιά κατεύθυνση.





Εικόνα 1.3.27.

α: Δένδρο

β: Γράφος

ΜΕΘΟΔΟΣ ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

Η **«Διαίρει και Βασίλευε»** (divide and conquer) αποτελεί μια μέθοδο σχεδίασης αλγορίθμων στην οποία εντάσσονται οι τεχνικές που υποδιαιρούν ένα πρόβλημα σε μικρότερα υποπροβλήματα, που έχουν την ίδια τυποποίηση με το αρχικό πρόβλημα, αλλά είναι μικρότερα σε μέγεθος. Με όμοιο τρόπο, τα υποπροβλήματα αυτά μπορούν να διαιρεθούν σε ακόμη μικρότερα υποπροβλήματα κ.ο.κ. Έτσι η επίλυση ενός προβλήματος έγκειται στη σταδιακή επίλυση των όσο το δυνατόν μικρότερων υποπροβλημάτων, ώστε τελικά να προκύψει η συνολική λύση του αρχικού ευρύτερου προβλήματος.
Η προσέγγιση αυτή ονομάζεται «από πάνω προς τα κάτω» (top-down).

Η μέθοδος σχεδίασης αλγορίθμων «Διαίρει και Βασίλευε» μπορεί να αποδοθεί με τα επόμενα βήματα:
1. Δίνεται για επίλυση ένα στιγμιότυπο ενός προβλήματος.
2. Το στιγμιότυπο του προβλήματος υποδιαιρείται σε υπο-στιγμιότυπα του ίδιου προβλήματος.
3. Δίνεται ανεξάρτητη λύση σε κάθε ένα υπο-στιγμιότυπο.
4. Συνδυάζονται όλες οι μερικές λύσεις που βρέθηκαν για τα υπο-στιγμιότυπα, έτσι ώστε να δοθεί η συνολική λύση του προβλήματος.

Ένας κλασικός αλγόριθμος που ακολουθεί τη φιλοσοφία της μεθόδου «Διαίρει και Βασίλευε» είναι η «Δυαδική αναζήτηση».

ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

Η δυαδική αναζήτηση στηρίζει τη λειτουργία της στο γεγονός ότι ο πίνακας είναι ταξινομημένος.
Βρίσκουμε τη μεσαία θέση του πίνακα κι ελέγχουμε αν το στοιχείο που αναζητάμε είναι σε αυτήν την θέση. Αν ναι, η αναζήτηση τελειώνει. Αν το στοιχείο που υπάρχει στη μεσαία θέση είναι μικρότερο από αυτό που ψάχνουμε τότε το τελευταίο αποκλείεται να βρίσκεται αριστερά από τη μεσαία θέση εφόσον ο πίνακας είναι ταξινομημένος, άρα περιορίζουμε την αναζήτηση στο δεύτερο μισό του πίνακα δηλ. από τη μέση του πίνακα και μετά. Ξαναβρίσκουμε τη μεσαία θέση του δεύτερου μισού του πίνακα και κάνουμε τον ίδιο έλεγχο κ.ο.κ.
Το αντίστοιχο συμβαίνει αν το στοιχείο που περιέχεται στη μεσαία θέση του πίνακα είναι μεγαλύτερο από αυτό που ψάχνουμε, οπότε περιορίζουμε την αναζήτηση στο πρώτο μισό του πίνακα.

Με τη χρήση δυο δεικτών του L (Left) και του R (Right) καθορίζουμε αρχικά την αρχή και το τέλος του πίνακα. Από τις τιμές των δυο αυτών δεικτών υπολογίζουμε τη μεσαία θέση του πίνακα M<--(L+R) DIV 2 και εκεί κάνουμε τη σύγκριση με το στοιχείο που ψάχνουμε.

Αν το στοιχείο που υπάρχει στη μεσαία θέση είναι μικρότερο από αυτό που ψάχνουμε τότε το τελευταίο αποκλείεται να βρίσκεται αριστερά από τη θέση του M , άρα μετακινούμε τον δείκτη L μια θέση δεξιά από το M ορίζοντας έτσι μια άλλη περιοχή του πίνακα στην οποία θα γίνει η αναζήτηση . Στην πραγματικότητα αφαιρούμε από την αναζήτηση το τμήμα του πίνακα που βρίσκεται αριστερά του M .

Το αντίστοιχο συμβαίνει αν το στοιχείο που περιέχεται στη θέση M είναι μεγαλύτερο από αυτό που ψάχνουμε.

Όλη αυτή η διαδικασία έχει ως αποτέλεσμα την ταχύτατη εύρεση του υπο αναζήτηση στοιχείου.

Έστω ότι αναζητάμε την τιμή key στον πίνακα table ο οποίος έχει n θέσεις. Ο αλγόριθμος της δυαδικής αναζήτησης φαίνεται παρακάτω:

```
Αλγόριθμος δυαδική_αναζήτηση
L←1
R←n
DONE←ΨΕΥΔΗΣ
position<--0
Όσο DONE=ΨΕΥΔΗΣ και L<=R επανάλαβε
        M←(L+R) DIV 2
        Αν table[M] = key τότε
                DONE← ΑΛΗΘΗΣ
                        position<--M
        Αλλιώς_αν table[M]<key τότε
                L←M+1
        Αλλιώς
                R←M-1
        Τέλος_αν
Τέλος_επανάληψης
Αν DONE = ΑΛΗΘΗΣ τότε
        Εμφάνισε 'Βρέθηκε το στοιχείο στη θέση:',position
Αλλιώς
        Εμφάνισε 'Δεν βρέθηκε το στοιχείο'
Τέλος_αν
Τέλος δυαδική_αναζήτηση
```

ΔΥΑΔΙΚΗ Ή ΣΕΙΡΙΑΚΗ ΑΝΑΖΗΤΗΣΗ;

Η Δυαδική αναζήτηση σε αντίθεση με την Σειριακή εφαρμόζεται ΜΟΝΟ σε ταξινομημένους πίνακες και θεωρείται πολύ πιο γρήγορη.

Ακόμη και στην περίπτωση που το υπό αναζήτηση στοιχείο δεν υπάρχει στον πίνακα, η σάρωση του πίνακα γίνεται πολύ γρήγορα. Για παράδειγμα σε ένα πίνακα με 200 θέσεις αν δεν υπάρχει το υπό αναζήτηση στοιχείο η σειριακή αναζήτηση θα πραγματοποιήσει 200 επαναλήψεις για να καταλήξει στο αποτέλεσμα, αντίστοιχα η δυαδική δεν θα χρειαστεί περισσότερες από 7 με 8 επαναλήψεις.

Η Σειριακή αναζήτηση είναι γρηγορότερη από την Δυαδική μόνο όταν το υπό αναζήτηση στοιχείο βρίσκεται σε μια από τις πρώτες θέσεις του πίνακα.

ΕΝΤΟΛΗ ΕΠΙΛΕΞΕ

Γενική μορφή της εντολής ΕΠΙΛΕΞΕ

```
ΕΠΙΛΕΞΕ <έκφραση>
 ΠΕΡΙΠΤΩΣΗ <λίστα_τιμών_1>
  <εντολές_1>
  ΠΕΡΙΠΤΩΣΗ <λίστα_τιμών_2>
  <εντολές_2>
  .....
  ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
  <εντολές_αλλιώς>
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

Όπου:

- **<έκφραση> :**
  είναι μια μεταβλητή, η τιμή της οποίας θα ελεγχθεί με τις τιμές που δίνονται στις ΠΕΡΙΠΤΩΣΕΙΣ και ανάλογα σε ποια ΠΕΡΙΠΤΩΣΗ ανήκει θα εκτελεστούν οι αντίστοιχες εντολές **ή** η πράξη, που υπολογίζει την τιμή της.
  Δηλαδή, η **<έκφραση>** μπορεί να είναι:
  - Μεταβλητή
  - Αριθμητική πράξη
  - Συγκριτική πράξη

- **<λίστα_τιμών_Ν>:**
  οι τιμές που μπορεί να πάρει μια έκφραση. Οι τιμές αυτές μπορεί να είναι διακριτές τιμές, περιοχή τιμών από...έως ή να υπακούν σε μια συνθήκη.

Τρόπος εκτέλεσης

Κατά την εκτέλεση της εντολής υπολογίζεται η τιμή της έκφρασης και στη συνέχεια εκτελούνται οι εντολές που ανήκουν στην αντίστοιχη περίπτωση τιμών. Στην περίπτωση που η τιμή έκφρασης δεν αντιστοιχεί σε καμία περίπτωση, τότε εκτελούνται οι εντολές της ΠΕΡΙΠΤΩΣΗΣ_ΑΛΛΙΩΣ. Η ΠΕΡΙΠΤΩΣΗ_ΑΛΛΙΩΣ είναι προαιρετική.

Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή που ακολουθεί μετά το ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ.

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

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

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

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

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

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

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

Μια κλάση Α μπορεί να είναι έγκυρη υποκλάση της Β αν έχει νόημα να πούμε «ένα Α είναι ένα (is_a) Β»

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

ΚΑΤΗΓΟΡΙΕΣ ΛΑΘΩΝ

Μπορούμε να διακρίνουμε τις εξής κατηγορίες λαθών:

- Συντακτικά λάθη
- Λάθη που οδηγούν σε αντικανονικό τερματισμό του προγράμματος
- Λογικά λάθη που παράγουν λανθασμένα αποτελέσματα

ΣΥΝΤΑΚΤΙΚΑ ΛΑΘΗ

Κάποιες φορές ένα πρόγραμμα δεν μπορεί να εκτελεστεί, επειδή κατά τη μετάφραση εντοπίζονται συντακτικά λάθη. Π.χ. δε γράψαμε σωστά μια δεσμευμένη λέξη, παραλείψαμε μια δεσμευμένη λέξη ή παραλείψαμε να δηλώσουμε μια μεταβλητή.

ΛΑΘΗ ΠΟΥ ΟΔΗΓΟΥΝ ΣΕ ΑΝΤΙΚΑΝΟΝΙΚΟ ΤΕΡΜΑΤΙΣΜΟ

Ένα πρόγραμμα μπορεί να τερματίσει αντικανονικά λόγω διαφόρων λαθών. Για παράδειγμα, αν επιχειρήσουμε να διαιρέσουμε με το μηδέν ή αν κατά την ανάγνωση ενός ακεραίου αριθμού εισαχθεί ένα γράμμα.

ΛΟΓΙΚΑ ΛΑΘΗ

Ακόμη κι αν το πρόγραμμά μας δεν περιέχει συντακτικά λάθη και μπορεί να εκτελεστεί, πρέπει οπωσδήποτε να ελεγχθεί, ώστε να διαπιστώσουμε αν κατά την εκτέλεσή του εμφανίζονται λογικά λάθη. Τα λογικά λάθη έχουν ως συνέπεια το πρόγραμμα σε κάποιες περιπτώσεις να εξάγει λανθασμένα αποτελέσματα. Για να εντοπίσουμε τα λογικά λάθη μπορούμε να κάνουμε δοκιμαστικές εκτελέσεις του προγράμματός μας και να ελέγξουμε αν για συγκεκριμένες τιμές εισόδου, το πρόγραμμά μας εξάγει σωστά αποτελέσματα.

Εκσφαλμάτωση λογικών λαθών στις δομές επιλογής

Σε μια δομή επιλογής μπορεί να εμφανιστούν λογικά λάθη που σχετίζονται με:

- τη συνθήκη ή τις συνθήκες
- τις ομάδες εντολών που εκτελούνται όταν μια συνθήκη είναι αληθής ή ψευδής.

Εκσφαλμάτωση λογικών λαθών στις δομές επανάληψης

Στην ενότητα αυτή θα ασχοληθούμε με την εκσφαλμάτωση κάποιων συνηθισμένων λαθών στις δομές επανάληψης. Σε μια δομή επανάληψης μπορεί να εμφανιστούν λογικά λάθη που σχετίζονται με:

- τη συνθήκη επανάληψης ή τερματισμού,
- την αρχικοποίηση της συνθήκης,
- την ενημέρωση της συνθήκης εντός του βρόχου επανάληψης,
- τις εντολές που περιλαμβάνονται εντός του βρόχου.

Συμβουλή: Κατά την εκσφαλμάτωση των δομών επανάληψης χρειάζεται να δίνετε προσοχή στα εξής:

- στους συγκριτικούς και τους λογικούς τελεστές των συνθηκών επανάληψης ή τερματισμού
- στην αρχικοποίηση της συνθήκης
- στην ενημέρωση της συνθήκης εντός του βρόχου
- στην αλληλουχία των εντολών του βρόχου και στη σειρά εκτέλεσής τους
- στο κριτήριο της περατότητας
- στην πρώτη επανάληψη και στην περίπτωση που ο βρόχος επανάληψης δεν πρέπει να εκτελεστεί ούτε μία φορά
- στην τελευταία επανάληψη

Εκσφαλμάτωση λογικών λαθών σε πίνακες

Συμβουλή: Κατά την εκσφαλμάτωση προγραμμάτων που χρησιμοποιούν πίνακες χρειάζεται να δίνετε ιδιαίτερη προσοχή:

- στο μέγεθος των πινάκων κατά τη δήλωσή τους,
- στους δείκτες των πινάκων κατά την προσπέλασή τους,
- στη μη υπέρβαση των ορίων του πίνακα.

Εκσφαλμάτωση λογικών λαθών στα υποπρογράμματα

Συμβουλή: Κατά την εκσφαλμάτωση προγραμμάτων που χρησιμοποιούν υποπρογράμματα χρειάζεται να δίνεται προσοχή στον εντοπισμό λογικών λαθών που σχετίζονται με:

- την κλήση του υποπρογράμματος και το πέρασμα των παραμέτρων
- τα λοιπά λογικά λάθη που εμφανίζονται και στα προγράμματα.

ΜΕΘΟΔΟΣ ΕΛΕΓΧΟΥ «ΜΑΥΡΟ ΚΟΥΤΙ»

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

Μια δημοφιλής τεχνική ελέγχου είναι ο **έλεγχος μαύρου κουτιού (black-box testing)**. Ονομάζεται έτσι επειδή τα δεδομένα εισόδου στα σενάρια ελέγχου προκύπτουν από τις προδιαγραφές του προγράμματος, αγνοώντας εντελώς τον κώδικα. Δηλαδή το πρόγραμμα μοιάζει σαν να βρίσκεται μέσα σε ένα μαύρο κουτί που κρύβει το περιεχόμενό του.

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

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

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