Πληροφορική

Περίληψη Θεωρίας ΙΙ

(Νέα ύλη)

Δημήτρης Παπαδάκης (697 460 0499)
dimitrisp@easylearn.gr

Πίνακας Περιεχομένων

Δομές Δεδομένων και Αλγόριθμοι....4
Στοίβα....4
Ουρά....4
Λίστες....5
Διαφορές Λίστας / Πίνακα (Πλεονεκτήματα – Μειονεκτήματα)....6
Βασικές πράξεις των συνδεδεμένων λιστών....6
Δένδρα....7
Χρήσεις των δένδρων....7
Πλεονεκτήματα των δένδρων....7
Παραδείγματα χρήσης δένδρων....7
Δένδρα απόφασης....8
Δένδρα παιχνιδιού....8
Δυαδικά δένδρα....8
Δυαδικά δένδρα αναζήτησης....8
Γράφοι....8
Διαίρει και Βασίλευε....9
Αντικειμενοστραφής προγραμματισμός....10
Μεθοδολογία σχεδίασης....10
Κλάσεις (αφαιρετικότητα και ενθυλάκωση)....10
Ενθυλάκωση....10
Κλάση....10
Κληρονομικότητα....11
Σχέση «είναι ένα» (is_a)....11
Πολυμορφισμός....11
Εκσφαλμάτωση Προγράμματος....12
Κατηγορίες λαθών....12
Συντακτικά λάθη....12
Λάθη που οδηγούν σε αντικανονικό τερματισμό του προγράμματος....12
Λογικά λάθη....12
Εκσφαλμάτωση....12
5.2.1 Εκσφαλμάτωση λογικών λαθών στις δομές επιλογής....12
5.2.2 Εκσφαλμάτωση λογικών λαθών στις δομές επανάληψης....13
Δομές επανάληψης....13
Εκσφαλμάτωση λογικών λαθών σε πίνακες....13
Εκσφαλμάτωση λογικών λαθών στα υποπρογράμματα....13
Μέθοδος ελέγχου «Μαύρο Κουτί»....14
Σενάριο ελέγχου....14
Μαύρο κουτί....14

Δομές Δεδομένων και Αλγόριθμοι

Στοίβα

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).

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.

```
Διαδικασία Ώθηση(Π, top, Χ, done)
Μεταβλητές
    Χαρακτήρες: Π[10], Χ
    Ακέραιες: top
    Λογικές: done
Αρχή
Αν top < 10 Τότε
    top ← top + 1
    Π[top] ← Χ
    done ← Αληθής
Αλλιώς
    done ← Ψευδής
Τέλος_Αν
Τέλος_Διαδικασίας
```

```
Διαδικασία Απώθηση(Π, top, Χ, done)
Μεταβλητές
    Χαρακτήρες: Π[10], Χ
    Ακέραιες: top
    Λογικές: done
Αρχή
Αν top > 0 Τότε
    Χ ← Π[top]
    top ← top – 1
    done ← Αληθής
Αλλιώς
    done ← Ψευδής
Τέλος_Αν
Τέλος_Διαδικασίας
```

Ουρά

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).

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.

```
Διαδικασία Εισαγωγή(Π, front, rear, Χ, done)
```

```
Διαδικασία Εξαγωγή(Π, front, rear, Χ, done)
```

```
Μεταβλητές
    Χαρακτήρες: Π[10], Χ
    Ακέραιες: front, rear
    Λογικές: done
Αρχή
Αν rear = 10 Τότε
    done ← Ψευδής
Αλλιώς_Αν front = 0 και rear = 0 Τότε
    front ← 1
    rear ← 1
    Π[rear] ← Χ
    done ← Αληθής
Αλλιώς
    rear ← rear + 1
    Π[rear] ← Χ
    done ← Αληθής
Τέλος_Αν
Τέλος_Διαδικασίας
```

```
Μεταβλητές
    Χαρακτήρες: Π[10], Χ
    Ακέραιες: front, rear
    Λογικές: done
Αρχή
Αν front = 0 και rear = 0  Τότε
    done ← Ψευδής
Αλλιώς_Αν front = rear Τότε
    Χ ← Π[front]
    front ← 0
    rear ← 0
    done ← Αληθής
Αλλιώς
    Χ ← Π[front]
    front ← front + 1
    done ← Αληθής
Τέλος_Αν
Τέλος_Διαδικασίας
```

Λίστες

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.

Οι κόμβοι μιας λίστας δεν έχουν ονόματα. Γνωρίζουμε μόνο τις διευθύνσεις τους, που είναι αποθηκευμένες στους προηγούμενους κόμβους.

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).

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.

Μια λίστα μπορεί να είναι **απλά συνδεδεμένη**, δηλαδή να μπορούμε να κινηθούμε προς μία μόνο κατεύθυνση, ξεκινώντας από τον πρώτο κόμβο και μετακινούμενοι προς τον τελευταίο 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.

Σε μια διπλά συνδεδεμένη λίστα διευκολύνεται η ταξινόμηση και η αναζήτηση, ωστόσο, αυξάνεται η πολυπλοκότητα στη διαχείριση των κόμβων, καθώς απαιτείται επιπλέον χώρος για τον δεύτερο δείκτη (επιπρόσθετη μνήμη για κάθε κόμβο).

Διαφορές Λίστας / Πίνακα (Πλεονεκτήματα – Μειονεκτήματα)

Διαφορές λίστας σε σχέση με τον πίνακα

- 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 ν-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

- 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).

Δένδρα

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[1], εκτός από τη ρίζα, υπάρχει μόνο μια ακμή που καταλήγει στον κόμβο αυτόν ξεκινώντας από κάποιον άλλον κόμβο p[2]. 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.

Μπορούμε να έχουμε ένα **απλό δένδρο**, το οποίο να απαρτίζεται από έναν μόνο κόμβο. Αυτός ο κόμβος είναι και ρίζα του απλού αυτού δένδρου, διότι δεν έχει γονέα και φύλλο, και διότι δεν έχει παιδιά.

Κάθε κόμβος ενός δένδρου μπορεί να θεωρηθεί ως η ρίζα ενός υποδένδρου.

Διατεταγμένο δένδρο λέγεται το δένδρο, στο οποία για κάθε κόμβο υπάρχει μία γραμμική σχέση μεταξύ των παιδιών του κόμβου αυτού.

Χρήσεις των δένδρων

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.

Πλεονεκτήματα των δένδρων

Είναι αποτελεσματικά στην οργάνωση και διαχείριση δεδομένων

1. Τα δένδρα είναι δυναμικά

   Είναι πολύ εύκολο να προσθέσουμε, να αφαιρέσουμε ή να αναζητήσουμε ένα στοιχείο σε ένα δένδρο.

2. Η δομή των δένδρων μεταφέρει πληροφορίες.

Παραδείγματα χρήσης δένδρων

Τα δένδρα είναι χρήσιμα στην αναπαράσταση δεδομένων που διέπονται από ένα είδος φυσικής ιεραρχίας, όπως:

1. Οικογενειακό δένδρο

1 Child (c) – παιδί

2 Parent (p) – γονέας

2. Η δομή ενός οργανισμού ή μιας εταιρείας
3. Ο πίνακας περιεχομένων ενός βιβλίου
4. Τα αρχεία και οι φάκελοι ενός υπολογιστή
5. Ένα λεξικό
6. Τα μέρη που απαρτίζουν τη μηχανή ενός αυτοκινήτου
7. Τα συστατικά μιας πρότασης
8. Αναπαράστασης και υπολογισμού μαθηματικών εκφράσεων

Επίσης, τα δένδρα αποτελούν τη βάση αρκετών αλγορίθμων επίλυσης προβλήματος, όπως:

1. η συμπίεση εικόνων,
2. η ταξινόμηση,
3. η αυτόματη συμπλήρωση λέξεων σε συσκευές κινητών τηλεφώνων,
4. η μεταγλώττιση ενός προγράμματος και
5. η λήψη αποφάσεων

Δένδρα απόφασης

Τα δένδρα απόφασης 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 και είναι ιδιαίτερα δημοφιλή σε περιπτώσεις ιατρικών διαγνώσεων.

Δένδρα παιχνιδιού

Τα **δένδρα του παιχνιδιού** (game tree) είναι δένδρα, με τα οποία ο υπολογιστής μοντελοποιεί όλες τις πιθανές κινήσεις των παικτών. Κάθε κόμβος στο δένδρο αντιπροσωπεύει μία συγκεκριμένη κατάσταση παιχνιδιού και περιέχει πληροφορίες σχετικά με το ποιος παίκτης έχει τη μεγαλύτερη πιθανότητα να κερδίσει από οποιαδήποτε πιθανή κίνηση.

Δυαδικά δένδρα

Ένα δυαδικό δένδρο (binary tree) είναι ένα διατεταγμένο δένδρο, στο οποίο κάθε κόμβος έχει το πολύ δύο παιδιά, το αριστερό και το δεξί παιδί.

Δυαδικά δένδρα αναζήτησης

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**.

Γράφοι

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 (λίστες και δένδρα) μπορούν να θεωρηθούν περιπτώσεις γράφων.

Εάν όλες οι ακμές σε έναν γράφο έχουν κατεύθυνση, ο γράφος ονομάζεται **κατευθυνόμενος γράφος**. Εάν όλες οι ακμές σε έναν γράφο δεν έχουν κατεύθυνση, ο γράφος ονομάζεται **μη κατευθυνόμενος γράφος**.

Διαίρει και Βασίλευε

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.

Αντικειμενοστραφής προγραμματισμός

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).

Σύμφωνα με την αντικειμενοστραφή θεωρία ανάπτυξης εφαρμογών, η προσέγγιση κάθε προβλήματος πρέπει να γίνεται με φυσική ερμηνεία και να μη στηρίζεται σε πολύπλοκα τεχνικά ζητήματα.

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.

Μεθοδολογία σχεδίασης

Αναλύουμε το πρόβλημα, το οποίο θέλουμε να επιλύσουμε, δηλαδή αναγνωρίζουμε και καταγράφουμε τα βασικά συστατικά στοιχεία της διαδικασίας επίλυσής του που είναι:

1. τα αντικείμενα που συμμετέχουν με βάση τον ρόλο τους στο συγκεκριμένο σενάριο
2. οι ιδιότητες κάθε αντικειμένου, δηλαδή τα σχετικά με το συγκεκριμένο πρόβλημα χαρακτηριστικά του, και
3. οι υπηρεσίες που προσφέρει ή οι ενέργειες που υλοποιεί κάθε αντικείμενο (μέθοδοι) προς αξιοποίηση από άλλες, ώστε να αναπτυχθούν οι απαραίτητες συνεργασίες μεταξύ των αντικειμένων για την επίλυση του προβλήματος.

Κλάσεις (αφαιρετικότητα και ενθυλάκωση)

Σε μια αντικειμενοστραφή εφαρμογή κάθε αντικείμενο αποτελεί ξεχωριστή οντότητα και περιέχει ενσωματωμένες τις ιδιότητες (δεδομένα) και τους κανόνες συμπεριφοράς του (μεθόδους).

Ενθυλάκωση

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.

Κλάση

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.

Κληρονομικότητα

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.

Σχέση «είναι ένα» (is_a)

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

Πολυμορφισμός

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

Πολυμορφισμός σημαίνει πολλές διαφορετικές μορφές ή πολλές διαφορετικές συνθήκες.

Ο πολυμορφισμός μας επιτρέπει να επαναπροσδιορίσουμε τον τρόπο με τον οποίο λειτουργούν κάποια πράγματα, είτε αλλάζοντας τον τρόπο λειτουργίας τους είτε αλλάζοντας τα εργαλεία τα οποία χρησιμοποιούνται για την επίτευξη του στόχου.

Εκσφαλμάτωση Προγράμματος

Κατηγορίες λαθών

1. Συντακτικά λάθη
2. Λάθη που οδηγούν σε αντικανονικό τερματισμό του προγράμματος
3. Λογικά λάθη που παράγουν λανθασμένα αποτελέσματα

Συντακτικά λάθη

Ένα πρόγραμμα δεν μπορεί να εκτελεστεί, όταν κατά τη μετάφραση εντοπίζονται συντακτικά λάθη.

- Δεν γράψαμε σωστά μια δεσμευμένη λέξη,
- Παραλείψαμε μια δεσμευμένη λέξη
- Παραλείψαμε να δηλώσουμε μια μεταβλητή.

Τα συντακτικά λάθη ανιχνεύονται από τον μεταγλωττιστή (και τον διερμηνευτή)

Λάθη που οδηγούν σε αντικανονικό τερματισμό του προγράμματος

Ένα πρόγραμμα μπορεί να τερματίσει αντικανονικά λόγω διαφόρων λαθών.

- Αν επιχειρήσουμε να διαιρέσουμε με το μηδέν
- Αν κατά την ανάγνωση ενός ακεραίου αριθμού εισαχθεί ένα γράμμα[3]

Λογικά λάθη

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.

Μερικές φορές το λογικό λάθος δεν υπάρχει στην εντολή που εμφανίζεται το λανθασμένο αποτέλεσμα, αλλά σε προηγούμενη εντολή.

Εκσφαλμάτωση

5.2.1 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.

Στην ανίχνευση ενός λογικού λάθους στις δομές επιλογής δεν αρκεί η μεμονωμένη μελέτη των συνθηκών και των ομάδων εντολών που εκτελούνται όταν μια συνθήκη είναι αληθής ή ψευδής, αλλά

3 Αν ο χρήστης αντί να δώσει μια αριθμητική τιμή, εισαγάγει ένα γράμμα, τότε το πρόγραμμα θα τερματίσει αντικανονικά λόγω λάθους του χρήστη. Στις σύγχρονες γλώσσες προγραμματισμού υπάρχουν τρόποι διαχείρισης τέτοιων λαθών, οι οποίοι δε θα μας απασχολήσουν στο πλαίσιο του μαθήματος αυτού.

χρειάζεται να μελετηθεί το αποτέλεσμα που παράγει ο συνδυασμός των συνθηκών και των ομάδων εντολών.

Μερικές φορές κατά την εκτέλεση της δομής επιλογής εμφανίζονται λανθασμένα αποτελέσματα, τα οποία σχετίζονται με λογικά λάθη σε προηγούμενες εντολές, που επηρεάζουν την τιμή που λαμβάνει η συνθήκη.

5.2.2 Εκσφαλμάτωση λογικών 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.

Πολλές φορές τα πιθανά σενάρια ελέγχου ενός προγράμματος είναι πάρα πολλά.

Στους υπολογισμούς που γίνονται εντός των δομών επανάληψης χρειάζεται να δοθεί ιδιαίτερη προσοχή στο αν θα συμπεριλάβουμε στον υπολογισμό την τιμή που λαμβάνει κάποια μεταβλητή στην τελευταία επανάληψη.

Δομές επανάληψης

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

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

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.

Μέθοδος ελέγχου «Μαύρο Κουτί»

Σενάριο ελέγχου

- Ένα **σενάριο ελέγχου** 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.

Μαύρο κουτί

Το μαύρο κουτί είναι μια δημοφιλής τεχνική ελέγχου που 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.

Επειδή είναι αδύνατο να ελέγξουμε όλες τις τιμές εισόδου και όλα τα πιθανά αποτελέσματα προσπαθούμε να βρούμε αντιπροσωπευτικές τιμές για τα δεδομένα εισόδου που θα παράγουν αντιπροσωπευτικά αποτελέσματα.

1. Το πρώτο βήμα είναι η **δημιουργία ισοδύναμων διαστημάτων τιμών** 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.

2. 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 <u>έλεγχος των ακραίων τιμών</u> κάθε διαστήματος, καθώς η εμπειρία έχει δείξει ότι τα περισσότερα λάθη γίνονται σε αυτά τα σημεία.

3. Το τελευταία βήμα είναι να δημιουργήσουμε ένα σενάριο ελέγχου για κάθε ακραία τιμή.

   Κάθε σενάριο πρέπει κατ' ελάχιστο να περιλαμβάνει την τιμή εισόδου, το αναμενόμενο αποτέλεσμα (σύμφωνα με την εκφώνηση του προβλήματος) και περιγραφή της περίπτωσης που ελέγχεται.

<u>Παρατηρήσεις:</u>

- Στα υποπρογράμματα, πρώτα ελέγχεται κάθε υποπρόγραμμα μεμονωμένα. Αφού διαπιστωθεί η ορθή λειτουργία του καθενός, μόνο τότε πραγματοποιείται έλεγχος ολόκληρου του προγράμματος.
- Αν οι είσοδοι είναι πολλές, τότε κάθε μία πρέπει να ελεγχθεί ανεξάρτητα από τις άλλες. Για να γίνει αυτό, δημιουργούνται σενάρια ελέγχου όπου μία από τις εισόδους λαμβάνει όλες τις ακραίες τιμές ενώ οι υπόλοιπες διατηρούνται σταθερές δίνοντας μια οποιαδήποτε έγκυρη τιμή. Αυτό επαναλαμβάνεται για όλες τις εισόδους.

  Η αύξηση των δεδομένων εισόδου οδηγεί σε μεγάλη αύξηση των σεναρίων ελέγχου, ενώ η διαδικασία αρχίζει να γίνεται περίπλοκη.