Κεφάλαιο 1: Ανάλυση Προβλήματος

- **Πρόβλημα**: Με τον όρο πρόβλημα εννοείται μία κατάσταση η οποία χρήζει αντιμετώπισης και απαιτεί λύση ή οποία δεν είναι γνωστή ούτε προφανής.
- **Αντιμετώπιση:** Για να αντιμετωπίσουμε σωστά ένα πρόβλημα χρειάζεται να ακολουθήσουμε με τη σειρά τα παρακάτω στάδια:

- **Από τι εξαρτάται η κατανόηση του προβλήματος:**

1ο Στάδιο : Κατανόηση
Σωστή διατύπωση του Προβλήματος εκ μέρους του δημιουργού του.
Σωστή ερμηνεία του Προβλήματος εκ μέρους εκείνου που καλείται να το επιλύσει.

2ο Στάδιο : Ανάλυση
Ξεκινάμε να ανακαλύπτουμε την δομή του προβλήματος, δηλαδή να το χωρίσουμε σε μικρότερα και απλούστερα υποπροβλήματα, καθένα από τα οποία έχει απλούστερη λύση.

3ο Στάδιο : Επίλυση
Για την σωστή επίλυση του προβλήματος βασική προϋπόθεση αποτελεί ο καθορισμός των απαιτήσεων. Αυτό σημαίνει:
Α. Να προσδιορίσουμε τα δεδομένα που παρέχονται.
Β. Να προσδιορίσουμε τα ζητούμενα, δηλαδή τι αποτελέσματα περιμένουμε.

- **Δεδομένο:** Είναι οτιδήποτε μπορεί να γίνει αντιληπτό από έναν τουλάχιστον παρατηρητή με μία από τις 5 αισθήσεις.
- **Πληροφορία:** Είναι οποιοδήποτε γνωστικό στοιχείο που προέρχεται από επεξεργασία δεδομένων.
- **Επεξεργασία Δεδομένων**: Είναι μία διαδικασία κατά την οποία ένας μηχανισμός δέχεται δεδομένα, τα επεξεργάζεται σύμφωνα με έναν προκαθορισμένο τρόπο και αποδίδει χρήσιμες πληροφορίες.
- **Δομή ενός Προβλήματος**: Είναι τα συστατικά του μέρη καθώς και ο τρόπος που συνδέονται μεταξύ τους.
- **Τρόποι αναπαράστασής ενός προβλήματος :**
  1) Φραστικά
  2) Διαγραμματική Αναπαράσταση

Κεφάλαιο 2,8 : Βασικές Έννοιες Αλγορίθμων

- **Αλγόριθμος:** Είναι μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος.
- **Εντολές:** Είναι καθένα από τα βήματα του αλγορίθμου. Οι εντολές πρέπει να είναι κατανοητές από αυτόν που θα τις εκτελέσει.
- **Πρόγραμμα:** Είναι ένας αλγόριθμος γραμμένος με τέτοιο τρόπο ώστε να μπορεί να εκτελεστεί από τον υπολογιστή.
- **Κριτήρια που πρέπει να ικανοποιεί ένας Αλγόριθμος:**
  1) Είσοδος (input) : Σε κάθε αλγόριθμο πρέπει να δίνονται μία ή περισσότερες τιμές δεδομένων. Στη συνέχεια ο αλγόριθμος επεξεργάζεται τα δεδομένα με κάποια συγκεκριμένη διαδικασία, για να παραχθούν αποτελέσματα. Υπάρχει και η περίπτωση κατά την οποία ο αλγόριθμος δε δέχεται καμία τιμή δεδομένου σαν είσοδο. Αυτό συμβαίνει όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες αρχικές τιμές με τη βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια απλών εντολών.
  2) Έξοδο (output): Ο αλγόριθμος πρέπει να δημιουργεί αποτελέσματα προς τον χρήστη ή προς έναν άλλο αλγόριθμο.
  3) Καθοριστικότητα (definiteness): Τα βήματα (εντολές) του αλγορίθμου πρέπει να είναι κατανοητά από αυτόν που τα εκτελεί. Δηλαδή κάθε εντολή του πρέπει να καθορίζει πλήρως τον τρόπο που θα εκτελεστεί.
  4) Περατότητα (finiteness): Ο αλγόριθμος να τελειώνει μετά από πεπερασμένο αριθμό βημάτων εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο.
  5) Αποτελεσματικότητα (effectiveness): Κάθε μεμονωμένη εντολή του αλγορίθμου πρέπει να είναι απλή, δηλαδή να έχει αφενός μεν ορισθεί και αφετέρου να μπορεί να εκτελεστεί.
- **Η Πληροφορική μελετά τους αλγορίθμους από τις ακόλουθες σκοπιές:**
  1) Υλικού(Hardware): Η ταχύτητα εκτέλεσης ενός αλγορίθμου επηρεάζεται από τις διάφορες τεχνολογίες υλικού, δηλαδή από τον τρόπο που είναι δομημένα σε μία ενιαία αρχιτεκτονική τα διάφορα συστατικά του υπολογιστή .
  2) Γλωσσών Προγραμματισμού(Programming Languages): Το είδος της γλώσσας προγραμματισμού που χρησιμοποιείται αλλάζει τη δομή και τον αριθμό των εντολών ενός αλγορίθμου. Γενικά μία γλώσσα που είναι χαμηλότερου επιπέδου είναι ταχύτερη από μία άλλη γλώσσα που είναι υψηλότερου επιπέδου. Ακόμη, σημειώνεται ότι διαφορές συναντώνται μεταξύ των γλωσσών σε σχέση με το πότε εμφανίσθηκαν. Για παράδειγμα, παλαιότερα μερικές γλώσσες προγραμματισμού δεν υποστήριζαν την αναδρομή.
  3) Θεωρητική(Theoretical): Το ερώτημα που συχνά τίθεται είναι αν πράγματι υπάρχει ή όχι κάποιος αποδοτικός αλγόριθμος για την επίλυση ενός προβλήματος. Η εξέταση αυτού του ερωτήματος είναι δύσκολο να σχολιασθεί στα πλαίσια του βιβλίου αυτού, επειδή απαιτεί μεγάλη θεωρητική κατάρτιση. Ωστόσο η προσέγγιση αυτή είναι ιδιαίτερα σημαντική, γιατί προσδιορίζει τα όρια της λύσης που θα βρεθεί σε σχέση με ένα συγκεκριμένο πρόβλημα.
  4) Αναλυτική(Analytical): Μελετώνται οι υπολογιστικοί πόροι που απαιτούνται από έναν αλγόριθμο, όπως για παράδειγμα το μέγεθος της κύριας και της δευτερεύουσας μνήμης, ο χρόνος για λειτουργίες CPU και για λειτουργίες εισόδου/εξόδου κ.λ.π.
- **Περιγραφή και αναπαράσταση αλγορίθμων**
  1) Ελεύθερο κείμενο (free text): Αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Έτσι εγκυμονεί τον κίνδυνο ότι μπορεί εύκολα να οδηγήσει σε μη εκτελέσιμη παρουσίαση παραβιάζοντας το τελευταίο χαρακτηριστικό των αλγορίθμων, δηλαδή την αποτελεσματικότητα.
  2) Διαγραμματικές τεχνικές (diagramming techniques): Συνιστούν ένα γραφικό τρόπο παρουσίασης του αλγορίθμου. Από τις διάφορες διαγραμματικές τεχνικές που έχουν επινοηθεί, η πιο παλιά και η πιο γνωστή ίσως, είναι το διάγραμμα ροής. Ωστόσο η χρήση διαγραμμάτων ροής για την παρουσίαση αλγορίθμων δεν αποτελεί την καλύτερη λύση, για αυτό και εμφανίζονται όλο και σπανιότερα στη βιβλιογραφία και στην πράξη.

3) Φυσική γλώσσα (natural language) κατά βήματα: Στην περίπτωση αυτή χρειάζεται προσοχή, γιατί μπορεί να παραβιασθεί το τρίτο βασικό χαρακτηριστικό ενός αλγορίθμου, δηλαδή το κριτήριο του καθορισμού.

4) Κωδικοποίηση (coding): Δηλαδή με ένα πρόγραμμα γραμμένο είτε σε μία ψευδογλώσσα είτε σε κάποια γλώσσα προγραμματισμού που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

- **Διαγράμματα ροής**

Είναι μία διαγραμματική τεχνική. Αποτελείται από ένα σύνολο γεωμετρικών σχημάτων, όπου το καθένα δηλώνει μία συγκεκριμένη ενέργεια ή λειτουργία. Τα σχήματα ενώνονται μεταξύ τους με βέλη που δηλώνουν την σειρά εκτέλεσης των ενεργειών αυτών.

Τα κυριότερα σχήματα είναι τα εξής:

1. **Έλλειψη**: Δηλώνει την αρχή και το τέλος του αλγορίθμου.
2. **Ρόμβος**: Δηλώνει τον έλεγχο μίας συνθήκης με δύο εξόδους ανάλογα με την τιμή της.
3. **Ορθογώνιο**: Δηλώνει την εκτέλεση μίας ή περισσοτέρων πράξεων.
4. **Πλάγιο παραλληλόγραμμο**: Η είσοδος δεδομένων και η έξοδος αποτελεσμάτων του αλγορίθμου.

- **Δομή Ακολουθίας:** Η ακολουθιακή δομή εντολών χρησιμοποιείται πρακτικά για την αντιμετώπιση απλών προβλημάτων, όπου είναι δεδομένη η σειρά εκτέλεσης ενός συνόλου ενεργειών.
- **Χαρακτηριστικά της δομής ακολουθίας είναι ότι:**
  1. Οι εντολές εκτελούνται η μία μετά την άλλη με την σειρά που είναι γραμμένες.
  2. Χρησιμοποιείται πρακτικά για την αντιμετώπιση πολύ απλών προβλημάτων όπου είναι δεδομένη η σειρά εκτέλεσης ενός συνόλου ενεργειών.
  3. Ο αλγόριθμος αποτελείται μόνο από εντολές εισόδου, εξόδου και εντολές εκχώρησης τιμής.
  4. Εκτελούνται υποχρεωτικά όλες οι εντολές του.
- **Σταθερές (constants) :** Προκαθορισμένες τιμές που παραμένουν αμετάβλητες σε όλη τη διάρκεια της εκτέλεσης ενός αλγορίθμου. Οι σταθερές διακρίνονται σε:
  1. Αριθμητικές
  2. Αλφαριθμητικές
  3. Λογικές
- **Μεταβλητές (variables) :** Μία μεταβλητή είναι ένα γλωσσικό αντικείμενο, που χρησιμοποιείται για να παραστήσει ένα στοιχείο δεδομένου. Στη μεταβλητή εκχωρείται μια τιμή, η οποία μπορεί να αλλάζει κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Ανάλογα με το είδος της τιμής που μπορούν να λάβουν, χωρίζονται σε:
  1. Αριθμητικές
  2. Αλφαριθμητικές
  3. Λογικές
- **Τελεστές (operators):** Πρόκειται για γνωστά σύμβολα που χρησιμοποιούνται στις διάφορες πράξεις. Οι τελεστές διακρίνονται σε αριθμητικούς, λογικούς και συγκριτικούς.

Βασικές Συναρτήσεις	Ενέργεια
ΗΜ(Χ)	Ημίτονο
ΣΥΝ(Χ)	Συνημίτονο
ΕΦ(Χ)	Εφαπτομένη
Τ_Ρ(Χ)	Τετραγωνική ρίζα
ΛΟΓ(Χ)	Λογάριθμος
Α_Τ(Χ)	Απόλυτη τιμή
Ε(Χ)	Εκθετική συνάρτηση
Α_Μ(Χ)	Ακέραιο μέρος αριθμού

- **Εκφράσεις (expressions):** Οι εκφράσεις διαμορφώνονται από τους τελεστέους, που είναι σταθερές και μεταβλητές και από τους τελεστές. Η διεργασία αποτίμησης μιας έκφρασης συνίσταται στην απόδοση τιμών στις μεταβλητές και στην εκτέλεση των πράξεων. Η τελική τιμή μιας έκφρασης εξαρτάται από την ιεραρχία των πράξεων και τη χρήση των παρενθέσεων. Μια έκφραση μπορεί να αποτελείται από μια μόνο μεταβλητή ή σταθερά μέχρι μια πολύπλοκη μαθηματική παράσταση.
- **Εντολές Ψευδογλώσσας**
  1. Δηλωτικές
     - **Αλγόριθμος** : Με αυτήν την εντολή ξεκινάει πάντα ο αλγόριθμος
     - **Τέλος** : Με αυτήν την εντολή τερματίζει ο κάθε αλγόριθμος
     - **Δεδομένα** : Εντολή για να δηλώσουμε δεδομένα εισόδου
     - **Αποτελέσματα** : Για την δήλωση αποτελεσμάτων
  2. Εκτελεστέες
     - **Εμφάνισε** : Έξοδο αποτελεσμάτων στην οθόνη
     - **Εκτύπωσε**: Έξοδο αποτελεσμάτων στον εκτυπωτή
     - **Διάβασε** : Εισαγωγή δεδομένων από το πληκτρολόγιο
     - Εντολές εκχώρησης (←)
  3. Σχόλια (!) : δεν επηρεάζουν τον αλγόριθμο ή το πρόγραμμα.

Αλγόριθμος αλά ρωσικά και Ολίσθηση

```
Αλγόριθμος πολλαπλασιασμός_αλά_ρωσικά
     Δεδομένα // Μ1, Μ2 ακέραιοι //
     Ρ ← 0
     Όσο Μ2 >0 επανάλαβε
        Αν Μ2 mod 2=1 τότε
           Ρ ← Ρ+Μ1
           Μ1 ← Μ1 * 2
           Μ2 ← Μ2 div 2
        Τέλος_αν
     Τέλος_επανάληψης
     Αποτελέσματα // Ρ, το γινόμενο των ακεραίων Μ1,Μ2 //
Τέλος πολλαπλασιασμός_αλά_ρωσικά
```

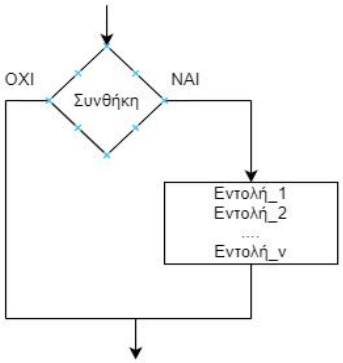
- **Ολίσθηση:** Όταν ο προγραμματιστής ορίζει ένα δεδομένο τότε αυτό αποθηκεύεται στα κυκλώματα του υπολογιστή σε δυαδική μορφή δηλαδή σε ακολουθίες από 0 και 1. Ολίσθηση είναι η μετακίνηση όλων των ψηφίων ενός αριθμού κατά μια θέση.

  Διακρίνουμε δύο ολισθήσεις:

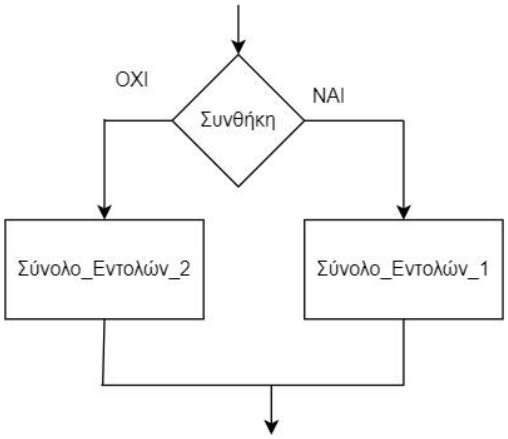
  Ολίσθηση προς τα αριστερά: Ισοδυναμεί με πολλαπλασιασμό επί δύο. Μετακινούμε όλα τα ψηφία του προς τα αριστερά προσθέτοντας ένα μηδέν στο τέλος και αγνοώντας το αρχικό μηδέν.

  Ολίσθηση προς τα δεξιά: Ισοδυναμεί με την ακέραια διαίρεση δια δύο. Μετακινούμε όλα τα ψηφία του προς τα δεξιά, αποκόπτοντας το τελευταίο και προσθέτοντας ένα μηδενικό στην αρχή.

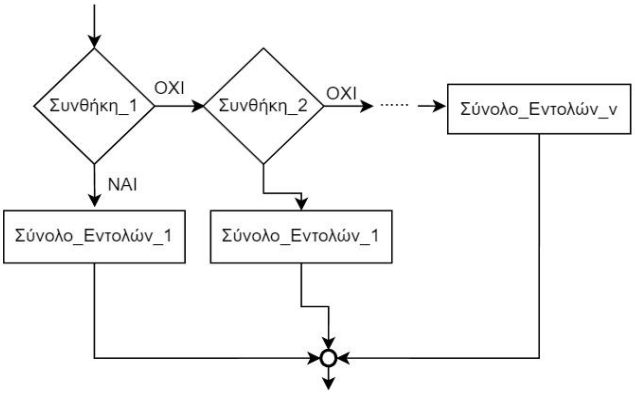
Δομές επιλογής

Δομή Απλής επιλογής	Διάγραμμα Ροής
1 **Αν** «Συνθήκη» **τότε** 2 Εντολή_1 3 Εντολή_2 4 ..... 5 Εντολή_ν 6 **Τέλος_αν**	

- **Λειτουργία:** Αν η συνθήκη ισχύει τότε εκτελούνται οι εντολές που βρίσκονται ανάμεσα στο τότε και στο Τέλος_αν, αλλιώς αγνοούνται. Η εκτέλεση συνεχίζεται με την εντολή μετά το Τέλος_αν

Δομή σύνθετης επιλογής	Διάγραμμα Ροής
1 **Αν** «Συνθήκη» **τότε** 2 Σύνολο_Εντολών_1 3 **Αλλιώς** 4 Σύνολο_Εντολών_2 5 **Τέλος_αν**	

- **Λειτουργία:** Αν η συνθήκη ισχύει, δηλαδή είναι αληθής, τότε εκτελούνται οι εντολές που βρίσκονται μεταξύ των δεσμευμένων λέξεων τότε και Αλλιώς, διαφορετικά εκτελούνται οι εντολές μεταξύ των δεσμευμένων λέξεων Αλλιώς και Τέλος_αν. Η εκτέλεση συνεχίζεται με την εντολή που ακολουθεί το Τέλος_αν.

Δομή πολλαπλής επιλογής	Διάγραμμα Ροής
1 **Αν** «Συνθήκη_1» **τότε** 2 Σύνολο_εντολών_1 3 **Αλλιώς_αν** «Συνθήκη_2» **τότε** 4 Σύνολο_εντολών_2 5 **Αλλιώς_αν** «Συνθήκη_3»**τότε** 6 Σύνολο_εντολών_3 7 …. 8 **Αλλιώς** 9 Σύνολο_εντολών_v 10 **Τέλος_αν**	

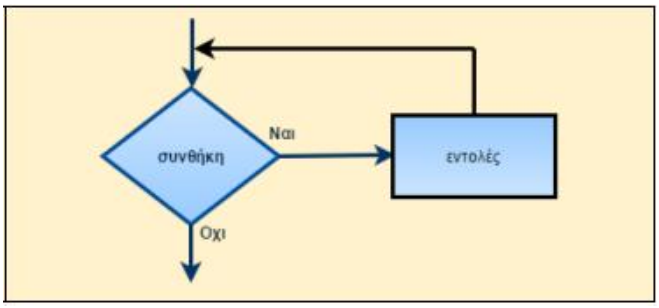
- **Λειτουργία:** Στη δομή της πολλαπλής επιλογής μπορούμε να έχουμε στην ίδια επιλογή περισσότερες από μια λογικές συνθήκες. Πρέπει να επισημάνουμε ότι στην δομή αυτή κάθε φορά ικανοποιείται μόνο μια λογική συνθήκη. Έτσι μόλις βρεθεί η λογική συνθήκη η οποία ικανοποιείται αμέσως εκτελείται η ομάδα εντολών που ανήκει στη συνθήκη αυτή και τερματίζεται η επιλογή, αυτό άμεσα σημαίνει ότι όλες οι άλλες συνθήκες είναι ψευδής.
- **Εμφωλευμένες δομές επιλογής:** Ονομάζονται δύο ή περισσότερες εντολές Αν…Αλλιώς που περιέχονται η μία μέσα στην άλλη. Η χρήση εμφωλευμένων αν οδηγεί συνήθως σε πολύπλοκες δομές που αυξάνουν την πιθανότητα λάθους και κάνουν το πρόγραμμα δυσνόητο.

Επίλεξε	Διάγραμμα Ροής
1 **ΕΠΙΛΕΞΕ** έκφραση 2 **ΠΕΡΙΠΤΩΣΗ** λίστα_τιμών_1 3 Εντολές1 4 **ΠΕΡΙΠΤΩΣΗ** λίστα_τιμών_2 5 Εντολές2 6 … 7 **ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ** 8 Εντολές_Αλλιώς 9 **ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ**	Δεν αναφέρει το βιβλίο θεωρούμε πως είναι το ίδιο με της πολλαπλής επιλογής

Παρατηρήσεις:

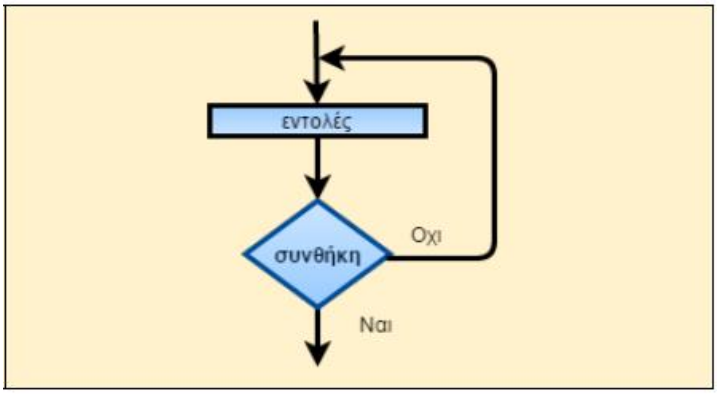
1) Η έκφραση που βρίσκεται δίπλα από τη δεσμευμένη λέξη ΕΠΙΛΕΞΕ μπορεί να είναι: μία μεταβλητή, πχ ΕΠΙΛΕΞΕ x, μία αριθμητική πράξη, πχ ΕΠΙΛΕΞΕ x mod 2, ή μία συγκριτική πράξη, πχ ΕΠΙΛΕΞΕ x < ψ.
   a. Μία η περισσότερες διακριτές τιμές χωρισμένες με κόμμα, είτε αριθμητικές διακριτές τιμές, πχ ΠΕΡΙΠΤΩΣΗ 1, 2, 3, 4, 5 είτε διακριτές τιμές τύπου χαρακτήρα, πχ ΠΕΡΙΠΤΩΣΗ ‘α’, ‘β’, ‘γ’.
   b. Να είναι περιοχή τιμών «από - έως», η οποία προσδιορίζεται με δύο τελείες(..), πχ για τιμές από 10 έως 100, γράφουμε ΠΕΡΙΠΤΩΣΗ 10..100.
   c. Να υπακούουν σε μία συνθήκη, δηλαδή να υπάρχει μία μόνο συνθήκη, πχ ΠΕΡΙΠΤΩΣΗ < 50.
2) Δεν υπάρχει περιορισμός στον αριθμό ΠΕΡΙΠΤΩΣΗ που μπορούν να υπάρχουν σε μία ΕΠΙΛΕΞΕ.
3) Υπάρχει περίπτωση η εντολή ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ να μην υπάρχει σε μία ΕΠΙΛΕΞΕ.
4) Η γραφή στην Ψευδογλώσσα(Αλγόριθμο) είναι παρόμοια απλά χωρίς κεφαλαία.

Δομές επανάληψης

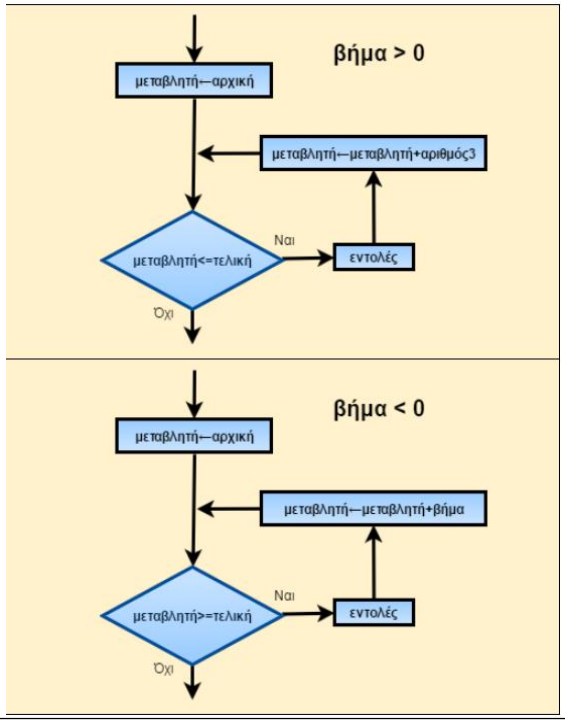
Δομή επανάληψης ΌΣΟ	Διάγραμμα Ροής
1 **ΟΣΟ** συνθήκη **ΕΠΑΝΑΛΑΒΕ** 2 Εντολή_1 3 .... 4 Εντολή_v 5 **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ**	

- **Λειτουργία:** Αρχικά ελέγχεται η συνθήκη και αν είναι αληθής εκτελούνται οι εντολές που βρίσκονται ανάμεσα στις ΟΣΟ και ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ. Στην συνέχεια ελέγχεται πάλι η συνθήκη και αν ισχύει εκτελούνται πάλι οι ίδιες εντολές. Όταν η συνθήκη γίνει Ψευδής τότε σταματά η επανάληψη και εκτελείται η εντολή μετά το ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ.

Παρατηρήσεις: Με τη δομή ΌΣΟ μπορούν να εκφραστούν όλες οι άλλες δομές επανάληψης. Χαρακτηριστικό της είναι ότι ο αριθμός των επαναλήψεων πρέπει υποχρεωτικά μέσα στον βρόχο να υπάρχει μία εντολή η οποία κάνει την συνθήκη Ψευδή.

Δομή επανάληψης ΜΕΧΡΙΣ_ΟΤΟΥ	Διάγραμμα Ροής
1 **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** 2 Εντολή_1 3 .... 4 Εντολή_v 5 **ΜΕΧΡΙΣ_ΟΤΟΥ** συνθήκη	

- **Λειτουργία:** Εκτελούνται οι εντολές μεταξύ των ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ και ΜΕΧΡΙΣ_ΟΤΟΥ. Στην συνέχεια ελέγχεται η συνθήκη και αν είναι Ψευδής τότε εκτελούνται πάλι οι εντολές. Όταν η συνθήκη βρεθεί αληθής τότε σταματά η επανάληψη και εκτελείται η εντολή μετά το ΜΕΧΡΙΣ_ΟΤΟΥ. Η δομή αυτή εκτελείται τουλάχιστον μία φορά.

Δομή επανάληψης ΓΙΑ	Διάγραμμα Ροής
1 **ΓΙΑ** μεταβλητή **ΑΠΟ** τιμή_1 **ΜΕΧΡΙ** τιμή_2 **ΜΕ_ΒΗΜΑ** τιμή_3 2 Εντολή_1 3 …. 4 Εντολή_v 5 **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ**	

- **Λειτουργία:** Οι εντολές του βρόχου εκτελούνται για όλες τις τιμές της μεταβλητής από την τιμή_1 μέχρι την τιμή_2 αυξανόμενες κατά την τιμή_3. Αν το βήμα είναι 1 τότε παραλείπεται το κομμάτι ΜΕ_ΒΗΜΑ. Χρησιμοποιείται όταν γνωρίζουμε εκ των προτέρων τον αριθμό των επαναλήψεων.
- **Τιμή φρουρός:** Η χρήση τιμών για τον τερματισμό μιας επαναληπτικής διαδικασίας είναι συνήθης στον προγραμματισμό. Η τιμή αυτή ορίζεται από τον προγραμματιστή και αποτελεί μια σύμβαση για το τέλος του προγράμματος. Η τιμή αυτή είναι τέτοια ώστε να μην είναι λογικά σωστή για το πρόβλημα.
- **Εμφωλευμένοι βρόχοι:** Ονομάζονται δύο ή περισσότεροι βρόχοι που βρίσκονται ο ένας μέσα στον άλλον.
- **Κανόνες χρήσης εμφωλευμένων βρόχων:**
  1) Ο εσωτερικός βρόχος πρέπει να βρίσκεται ολόκληρος μέσα στον εξωτερικό, έτσι ο βρόχος που ξεκινάει τελευταίος ολοκληρώνεται πρώτος.
  2) Η είσοδος σε ένα βρόχο υποχρεωτικά γίνεται από την αρχή του.
  3) Δεν μπορεί να χρησιμοποιηθεί η ίδια μεταβλητή ως μετρητής.

Κεφάλαια 3,9

- **Δομή δεδομένων:** Είναι ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.
- **Τα δεδομένα μελετούνται από τις σκοπιές:**
  1) **Υλικού:** Το υλικό (hardware), δηλαδή η μηχανή, επιτρέπει στα δεδομένα ενός προγράμματος να αποθηκεύονται στην κύρια μνήμη και στις περιφερειακές συσκευές του υπολογιστή με διάφορες αναπαραστάσεις (representations). Τέτοιες μορφές είναι η δυαδική, ο κώδικας ASCII (βλ. παράρτημα), ο κώδικας EBCDIC, το συμπλήρωμα του 1 ή του 2 κ.λ.π.
  2) **Γλωσσών προγραμματισμού:** Οι γλώσσες προγραμματισμού υψηλού επιπέδου (high level programming languages) επιτρέπουν τη χρήση διάφορων τύπων (types) μεταβλητών (variables) για να περιγράψουν ένα δεδομένο. Ο μεταφραστής κάθε γλώσσας φροντίζει για την αποδοτικότερη μορφή αποθήκευσης, από πλευράς υλικού, κάθε μεταβλητής στον υπολογιστή.
  3) **Δομών δεδομένων:** Δομή δεδομένων (data structure) είναι ένα σύνολο δεδομένων μαζί με ένα σύνολο επιτρεπτών λειτουργιών επί αυτών. Για παράδειγμα, μία τέτοια δομή είναι η εγγραφή (record), που μπορεί να περιγράφει ένα είδος, ένα πρόσωπο κ.λπ. Η εγγραφή αποτελείται από τα πεδία (fields) που αποθηκεύουν χαρακτηριστικά (attributes) διαφορετικού τύπου, όπως για παράδειγμα ο κωδικός, η περιγραφή κ.λ.π. Άλλη μορφή δομής δεδομένων είναι το αρχείο που αποτελείται από ένα σύνολο

εγγραφών. Μία επιτρεπτή λειτουργία σε ένα αρχείο είναι η σειριακή προσπέλαση όλων των εγγραφών του.

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

- **Βασικές λειτουργίες(ή πράξεις) επί των δομών δεδομένων:**

  1) **Προσπέλαση:** Πρόσβαση σε έναν κόμβο με σκοπό να εξεταστεί ή να αλλάξει το περιεχόμενο του.
  2) **Εισαγωγή:** Προσθήκη νέων κόμβων σε μία δομή.
  3) **Διαγραφή:** Αντίθετο της εισαγωγής, δηλαδή αφαίρεση ενός κόμβου από μία δομή.
  4) **Αναζήτηση:** Γίνεται προσπέλαση των κόμβων μίας δομής προκειμένου να εντοπιστούν ένας ή περισσότεροι που έχουν μία δεδομένη ιδιότητα.
  5) **Ταξινόμηση**: όπου οι κόμβοι διατάσσονται κατά αύξουσα ή φθίνουσα σειρά.
  6) **Αντιγραφή**: όλοι ή μερικοί από τους κόμβους μίας δομής αντιγράφονται σε μία άλλη.
  7) **Συγχώνευση**: δύο ή περισσότερες δομές ενώνονται σε μία δομή.
  8) **Διαχωρισμός**: (αντίστροφα από την συγχώνευση)

- **Νόμος του Wirth:** Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα
- **Κατηγορίες δομών δεδομένων**
  1. **Στατικές:** Το ακριβές μέγεθος τους είναι σταθερό και καθορίζεται κατά την στιγμή του προγραμματισμού τους και όχι κατά την στιγμή της εκτέλεσης του προγράμματος. Οι κόμβοι τους αποθηκεύονται σε συνεχόμενες θέσεις μνήμης. Στην πράξη οι στατικές δομές υλοποιούνται με πίνακες.
  2. **Δυναμικές:** Οι δομές αυτές δεν έχουν σταθερό μέγεθος αλλά ο αριθμός των κόμβων τους μεγαλώνει καθώς εισάγονται νέα δεδομένα, και μικραίνει καθώς διαγράφονται δεδομένα από αυτές. Δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης αλλά στηρίζονται στην τεχνική της δυναμικής παραχώρησης μνήμης. Όλες οι σύγχρονες γλώσσες προγραμματισμού τις υποστηρίζουν. (δυναμικές δομές είναι το αρχείο, η εγγραφή κ.α).
- **Πίνακας(ορισμός):** Είναι ένα σύνολο αντικειμένων ίδιου τύπου, τα οποία αναφέρονται με ένα κοινό όνομα. Κάθε ένα από τα αντικείμενα που απαρτίζουν τον πίνακα λέγεται στοιχείο του πίνακα. Η αναφορά σε ατομικά στοιχεία του πίνακα γίνεται με το όνομα του πίνακα ακολουθούμενο από έναν δείκτη.

- **Μειονεκτήματα Πινάκων:**

  1) Απαιτούν πολύ μνήμη.
  2) Περιορίζουν τις δυνατότητες του προγράμματος.

- **Τυπικές επεξεργασίες πινάκων:**

  1) Υπολογισμός αθροισμάτων στοιχείου πίνακα.
  2) Εύρεση ελαχίστου ή μεγίστου στοιχείου.
  3) Ταξινόμηση των στοιχείων του πίνακα.
  4) Αναζήτηση ενός στοιχείου του πίνακα
  5) Συγχώνευση δύο πινάκων.

Αναζήτηση

Αλγόριθμος Σειριακής-Γραμμικής αναζήτησης

Είναι η πιο απλή αλλά και η πιο αναποτελεσματική μέθοδος αναζήτησης σε πίνακα.

Η χρήση της δικαιολογείται σε περιπτώσεις όπου ο πίνακας:

1) Είναι μη ταξινομημένος,
2) Είναι μικρού μεγέθους (v<=20),
3) Η αναζήτηση γίνεται σπάνια.

Παρατηρήσεις:

Μια αποτελεσματικότερη μέθοδος αναζήτησης σε ταξινομημένο πίνακα είναι η δυαδική αναζήτηση.

Ο αλγόριθμος για την δυαδική αναζήτηση επιτρέπεται μόνο όταν ο πίνακας είναι ταξινομημένος

```
Αλγόριθμος Σειριακή_αναζήτηση
    Δεδομένα //n, table, key//
    Done ← Ψευδής
    I ←1
    Position ← 0
    Όσο i<=n ΚΑΙ Done=Ψευδής επανάλαβε
        Αν table[i] = key τότε
            Done ← Αληθής
            Position ← i
        Αλλιώς
            I ← i+1
        Τέλος_αν
    Τέλος_επανάληψης
    Αποτελέσματα //position, done //
Τέλος Σειριακή_αναζήτηση
```

Ταξινόμηση

- **Ταξινόμηση** (Ορισμός για την αύξουσα ταξινόμηση): Δοθέντων των στοιχείων α_1, α_2, … α_n η ταξινόμηση ορίζεται ως μετάθεση της θέσης των στοιχείων ώστε να τοποθετηθούν σε μια σειρά ακ_1, ακ_2, … ακ_n έτσι ώστε δοθείσης μίας συνάρτησης διάταξης (ordering function), F να ισχύει : F(ακ_1) ≤ F(ακ_2) ≤ … ≤ F(ακ_n).

Παρατηρήσεις: Ταξινόμηση είναι η διαδικασία κατά την οποία τα στοιχεία μιας δομής διατάσσονται κατά φθίνουσα ή αύξουσα σειρά. Σκοπός της ταξινόμησης είναι η εύκολη αναζήτηση ενός στοιχείου.

- **Ταξινόμηση ευθείας ανταλλαγής-φυσαλίδας**

Η εντολή **Αντιμετάθεσε** ανταλλάσσει το περιεχόμενο δυο θέσεων μνήμης με την βοήθεια μιας τρίτης θέσης. Αυτό μπορεί να γίνει με τις εξής τρεις εντολές:

```
temp← table[ j ]
table[ j ]← table [ j-1 ]
table [ j-1] ← temp
```

Η ταξινόμηση φυσαλίδας είναι ο πιο απλός και ταυτόχρονα ο πιο αργός αλγόριθμος ταξινόμησης.

```
Αλγόριθμος Φυσαλίδα
  Δεδομένα // table,n //
  Για i από 2 μέχρι n
    Για j από n μέχρι i με_βήμα -1
      Αν table[j]>table [j-1] τότε
        Αντιμετάθεσε table[j], table [j-1]
      Τέλος_αν
    Τέλος_επανάληψης
  Τέλος_επανάληψης
  Αποτελέσματα // table //
Τέλος Φυσαλίδα
```

Άλλοι αλγόριθμοι ταξινόμησης

Υπάρχουν πολλοί αλγόριθμοι ταξινόμησης. Άλλοι σχετικά απλοί αλγόριθμοι είναι

1) Η ταξινόμηση με επιλογή
2) Η ταξινόμηση με παρεμβολή.
3) Η γρήγορη ταξινόμηση (ο πιο γρήγορος αλγόριθμος ταξινόμησης).

Δομές Δεδομένων δευτερεύουσας μνήμης

- **Περιγράψτε τις δομές δεδομένων σε δευτερεύουσα μνήμη:** Τα στοιχεία ενός αρχείου ονομάζονται εγγραφές(records), όπου κάθε εγγραφή αποτελείται από ένα ή περισσότερα πεδία(fields). Ένα σύνολο από αρχεία αποτελούν μία βάση δεδομένων.

- **Τί γνωρίζεται για τα κλειδιά στα αρχεία;**

Τα πεδία χωρίζονται σε αυτά που ταυτοποιούν την εγγραφή και σε άλλα που περιγράφουν διάφορα χαρακτηριστικά της εγγραφής.
Το πεδίο που ταυτοποιεί την εγγραφή ονομάζεται πρωτεύων κλειδί(primary key) ή απλά κλειδί. Αν υπάρχει πρωτεύων κλειδί και υπάρχει και άλλο πεδίο που ταυτοποιεί την εγγραφή, αυτό αποκαλείται δευτερεύον κλειδί(secondary key).

- **Ποιες οι διαφορές στις δομές δεδομένων στην κύρια και δευτερεύουσα μνήμη;**

Στην δευτερεύουσα μνήμη(σκληρός δίσκος κλπ.) αποθηκεύονται τα δεδομένα που έχουν μεγάλο όγκο και το μέγεθος της κύριας μνήμης δεν επαρκεί για την αποθήκευση τους.
Επίσης τα δεδομένα δεν χάνονται αν διακοπεί η ηλεκτρική παροχή και διατηρούνται ακόμη και μετά τον τερματισμό ενός προγράμματος, κάτι που δεν συμβαίνει στην περίπτωση των δομών της κύριας μνήμης, όπως είναι οι πίνακες, όπου τα δεδομένα χάνονται όταν τελειώσει το πρόγραμμα.

ΣΤΟΙΒΑ

- **Τι ονομάζεται 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**).

- **Ποιες είναι οι κύριες λειτουργίες σε μια στοίβα;**

  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.

- **Πως υλοποιείται η στοίβα με χρήση μονοδιάστατου πίνακα;**

Χρησιμοποιούμε μια μεταβλητή (top), που δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας, η οποία έχει τιμή 0, όταν η στοίβα είναι άδεια.
Η ώθηση ενός νέου στοιχείου στη στοίβα (εισαγωγή στοιχείου στον πίνακα) γίνεται πάντα στην κορυφή της. Συγκεκριμένα, η μεταβλητή top αυξάνεται κατά ένα: top ← 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 ← top-1.

```
ΠΡΟΓΡΑΜΜΑ ΣΤΟΙΒΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: Α[10] , στοιχείο
  ΑΚΕΡΑΙΕΣ: top, απάντηση, i
ΑΡΧΗ
  top ← 0
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      ΓΡΑΨΕ 'Δώστε: | 1.  Ώθηση | 2.  Απώθηση | 9.  Έξοδος | '
      ΔΙΑΒΑΣΕ απάντηση
      ΑΝ απάντηση = 1 ΤΟΤΕ
! ******ώθηση******
    ΓΡΑΨΕ 'Δώστε τιμή για ώθηση : '
          ΔΙΑΒΑΣΕ στοιχείο
          ΑΝ top < 10 ΤΟΤΕ
              top ← top + 1
              Α[top] ← στοιχείο
          ΑΛΛΙΩΣ
              ΓΡΑΨΕ 'Γεμάτη στοίβα'
          ΤΕΛΟΣ_ΑΝ
      ΑΛΛΙΩΣ_ΑΝ απάντηση = 2 ΤΟΤΕ
! ******Απώθηση******
          ΑΝ top > 0 ΤΟΤΕ
              ΓΡΑΨΕ 'Απώθηση : ', Α[top]
              top ← top – 1
          ΑΛΛΙΩΣ
              ΓΡΑΨΕ 'Αδεια στοίβα'
         ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ απάντηση = 9
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

ΟΥΡΑ

❖ **Τι ονομάζεται 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).

❖ **Ποιες είναι οι κύριες λειτουργίες σε μια ουρά;**

1) Η **εισαγωγή** (enqueue) στοιχείου στο πίσω άκρο της ουράς.
2) Η **εξαγωγή** (dequeue) στοιχείου από το εμπρός άκρο της ουράς.

❖ **Πως υλοποιείται η ουρά 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. Αυτό υπό την προϋπόθεση ότι υπάρχει χώρος (rear < max).
Αν το στοιχείου που βάζουμε είναι το πρώτο (rear=1) τότε θέτουμε και front ← 1
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). Αυτό υπό την προϋπόθεση ότι υπάρχει στοιχείο (front <= rear και front > 0).
Όταν γίνεται εξαγωγή του τελευταίου στοιχείου της ουρά (front = rear) οι δείκτες παίρνουν πάλι τις αρχικές τιμές (0).

```
ΠΡΟΓΡΑΜΜΑ ΟΥΡΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: Α[10], στοιχείο
  ΑΚΕΡΑΙΕΣ: f, r, απάντηση, i
ΑΡΧΗ
  f ← 0
  r ← 0
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Δώστε: | 1.  Εισαγωγή | 2.  Εξαγωγή | 9. Έξοδος | '
    ΔΙΑΒΑΣΕ απάντηση
 ! ******Εισαγωγή******
    ΑΝ απάντηση = 1 ΤΟΤΕ
      ΓΡΑΨΕ 'Δώστε τιμή για εισαγωγή : '
      ΔΙΑΒΑΣΕ στοιχείο
      ΑΝ r = 10 ΤΟΤΕ
        ΓΡΑΨΕ 'Γεμάτη ουρά'
      ΑΛΛΙΩΣ_ΑΝ (r = 0 ΚΑΙ f = 0) ΤΟΤΕ
        f ← 1
        r ← 1
        A[r] ← στοιχείο
      ΑΛΛΙΩΣ
        r ← r + 1
        A[r] ← στοιχείο
      ΤΕΛΟΣ_ΑΝ
 ! ******Εξαγωγή******
    ΑΛΛΙΩΣ_ΑΝ απάντηση = 2
      ΑΝ  (r = 0 ΚΑΙ f = 0) ΤΟΤΕ
        ΓΡΑΨΕ 'Άδεια ουρά'
      ΑΛΛΙΩΣ_ΑΝ f = r ΤΟΤΕ
        ΓΡΑΨΕ 'Εξαγωγή : ', A[f]
        f  ← 0
        r ← 0
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'Εξαγωγή : ', A[f]
        f ← f + 1
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ απάντηση = 9
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Κεφάλαιο 6: Εισαγωγή στον προγραμματισμό

- **Στάδια επίλυσης προβλήματος μέσω υπολογιστή**
  1. Ακριβής προσδιορισμός του προβλήματος
  2. Η ανάπτυξη του αντίστοιχου αλγορίθμου
  3. Η διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή

- **Φυσικές και τεχνητές γλώσσες**
  - Οι **φυσικές γλώσσες** είναι αυτές που χρησιμοποιούν οι άνθρωποι προκειμένου να επικοινωνήσουν μεταξύ τους. Χαρακτηριστικό τους είναι ότι εξελίσσονται συνεχώς, αφού συνέχεια δημιουργούνται νέες λέξεις και οι κανόνες γραμματικής και σύνταξης αλλάζουν.
  - Οι **τεχνητές γλώσσες** είναι οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την επικοινωνία ανθρώπου-μηχανής. Αυτές χαρακτηρίζονται από στασιμότητα, αφού κατασκευάζονται συνειδητά για έναν συγκεκριμένο λόγο.

- **Από τι προσδιορίζεται μια γλώσσα;**

  1) Αλφάβητο: Είναι το σύνολο των στοιχείων που χρησιμοποιείται από την γλώσσα. (Παράδειγμα: Η ελληνική γλώσσα έχει τα εξής στοιχεία: Τα γράμματα του αλφαβήτου πεζά και κεφαλαία [Α – Ω και α – ω] τα ψηφία [0 – 9] και όλα τα σημεία στίξης [( ) / : κλπ.]).

  2) Λεξιλόγιο: Το λεξιλόγιο αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από την γλώσσα. (Παράδειγμα: Στην ελληνική γλώσσα η ακολουθία των γραμμάτων ΑΒΓΑ είναι δεκτή αφού αποτελεί λέξη, αλλά η ακολουθία ΑΒΓΔΑ δεν αποτελεί λέξη της ελληνικής γλώσσας, άρα δεν είναι δεκτή).

  3) Γραμματική: Η γραμματική αποτελείται από το τυπικό ή τυπολογικό και το συντακτικό.

- **Τυπικό:** είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία λέξη είναι αποδεκτή (Παράδειγμα: Στην ελληνική γλώσσα οι λέξεις γλώσσα, γλώσσας, γλώσσες είναι δεκτές, ενώ η λέξη γλωσσατ δεν είναι αποδεκτή)
- **Συντακτικό**: Είναι το σύνολο των κανόνων που καθορίζει την νομιμότητα της διάταξης και της σύνδεσης των λέξεων της γλώσσας για την δημιουργία προτάσεων. (Η γνώση του συντακτικού επιτρέπει τη δημιουργία σωστών προτάσεων στις φυσικές γλώσσες, ενώ στις γλώσσες προγραμματισμού τη δημιουργία σωστών εντολών)

4) Σημασιολογία: Είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα. (Στην γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας)

❖ **Διαφορές φυσικών και τεχνητών γλωσσών**

Οι φυσικές γλώσσες εξελίσσονται συνεχώς (νέες λέξεις δημιουργούνται, καινούργιοι κανόνες γραμματικής και σύνταξης αλλάζουν με την πάροδο του χρόνου και αυτό γιατί η γλώσσα χρησιμοποιείται για την επικοινωνία μεταξύ ανθρώπων, που εξελίσσονται και αλλάζουν ανάλογα με τις εποχές και τον κοινωνικό περίγυρο) ενώ οι τεχνητές γλώσσες χαρακτηρίζονται από στασιμότητα αφού κατασκευάζονται συνειδητά για έναν συγκεκριμένο σκοπό.

❖ **Τεχνικές σχεδίασης προγραμμάτων**

1) **Ιεραρχική σχεδίαση προγράμματος:** Η τεχνική της ιεραρχικής σχεδίασης προγράμματος περιλαμβάνει τον καθορισμό των βασικών λειτουργιών ενός προγράμματος, σε ανώτερο επίπεδο, και στη συνέχεια τη διάσπαση των λειτουργιών αυτών σε όλο και μικρότερες λειτουργίες, μέχρι το τελευταίο επίπεδο που οι λειτουργίες είναι πολύ απλές, ώστε να επιλυθούν εύκολα.
(Σκοπός της ιεραρχικής σχεδίαση είναι δηλαδή η διάσπαση του κεντρικού προβλήματος σε μία σειρά από απλούστερα υποπροβλήματα τα οποία είναι εύκολο να επιλυθούν οδηγώντας έτσι στην επίλυση του αρχικού προβλήματος)
2) **Τμηματικός προγραμματισμός:** Μετά την ανάλυση του προβλήματος σε αντίστοιχα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί ανεξάρτητη ενότητα, που γράφεται ξεχωριστά από τα υπόλοιπα τμήματα του προγράμματος.
(Σημείωση: Ο τμηματικός προγραμματισμός διευκολύνει τη δημιουργία του προγράμματος, μειώνει τα λάθη και επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους)
3) **Δομημένος προγραμματισμός:** Ο δομημένος είναι μία μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση των προγραμμάτων και να διευκολύνει τις διορθώσεις και τις αλλαγές σε αυτά.
Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφτούν χρησιμοποιώντας μόνο αυτές τις τρείς δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Πλεονεκτήματα Δομημένου Προγραμματισμού

1. Δημιουργία απλούστερων προγραμμάτων.
2. Άμεση μεταφορά των αλγορίθμων σε προγράμματα.
3. Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα.
4. Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος
5. Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους
6. Ευκολότερη διόρθωση και συντήρηση.

4) **Αντικειμενοστραφής Προγραμματισμός**

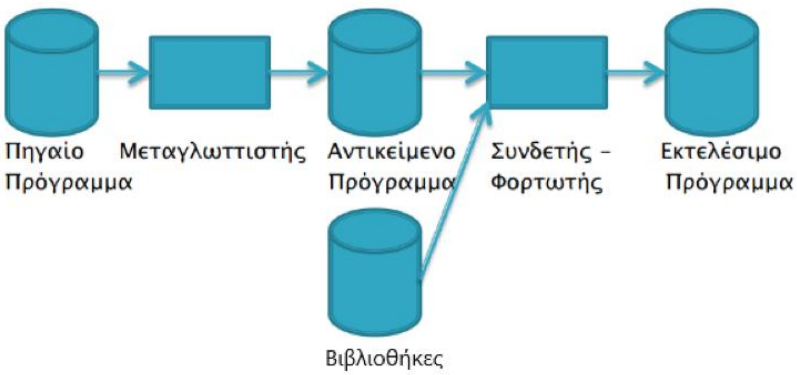
❖ **Τι γνωρίζεται για την εντολή GOTO**

Η εντολή GOTO (Πήγαινε) χρησιμοποιείται για την αλλαγή ροής ενός προγράμματος και τη διακλάδωση σε μία άλλη εντολή μέσα στο πρόγραμμα, εκτός από την επόμενη, πχ από τη $2^η$ εντολή να μετακινηθούμε στη $10^η$ εντολή και όχι στην $3^η$ όπως θα γίνονταν αν εκτελούνταν σε σειρά.
Η χρήση της εντολής GOTO κάνει το πρόγραμμα δυσνόητο και δύσκολη την παρακολούθηση του και για αυτό, όλες οι γλώσσες προγραμματισμού υποστηρίζουν τον δομημένο προγραμματισμό και διαθέτουν κατάλληλες εντολές, που η χρήση τους κάνει την εντολή GOTO περιττή.
Ορισμένες γλώσσες προγραμματισμού (Cobol - Fortran) διατηρούν ακόμη στο ρεπερτόριο των εντολών τους και την εντολή GOTO, κυρίως για λόγους συντήρησης παλιών προγραμμάτων και για συμβατότητα με παλαιότερες εκδόσεις τους.

❖ **Προγραμματιστικά Περιβάλλοντα**

Κάθε πρόγραμμα που γράφτηκε σε οποιοδήποτε γλώσσα προγραμματισμού πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές γλώσσας μηχανής. Η μετατροπή αυτή επιτυγχάνεται με τη χρήση ειδικών μεταφραστικών προγραμμάτων. Υπάρχουν 2 μεγάλες κατηγορίες τέτοιων προγραμμάτων, οι μεταγλωττιστές (compilers) και οι διερμηνευτές (interpreters)

- **Μεταγλωττιστής:** Είναι ένα μεταφραστικό πρόγραμμα που χρησιμοποιείται για την μετατροπή ενός προγράμματος γραμμένου σε γλώσσα προγραμματισμού υψηλού επιπέδου, σε εντολές γλώσσας μηχανής. Δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου, το οποίο ονομάζεται πηγαίο πρόγραμμα και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής το οποίο ονομάζεται αντικείμενο πρόγραμμα. Το πρόγραμμα που προκύπτει από τον μεταγλωττιστή είναι ανεξάρτητο από το αρχικό πρόγραμμα και μπορεί να εκτελείται οποτεδήποτε από τον υπολογιστή.
- **Διερμηνευτής:** Είναι ένα μεταφραστικό πρόγραμμα που χρησιμοποιείται για την μετατροπή ενός προγράμματος γραμμένου σε γλώσσα προγραμματισμού υψηλού επιπέδου, σε εντολές γλώσσας μηχανής. Διαβάζει 1 προς 1 τις εντολές του αρχικού προγράμματος και για κάθε μία εντολή εκτελεί αμέσως μία ισοδύναμη ακολουθία εντολών γλώσσα μηχανής.
- **Συντάκτης:** Είναι ένας επεξεργαστής κειμένου, που χρησιμοποιείται για την σύνταξη του προγράμματος (πηγαίου προγράμματος) και την διόρθωση του εάν αυτή απαιτείται.
- **Πηγαίο Πρόγραμμα (source)** λέγεται το αρχικό πρόγραμμα που έχουμε γράψει στον συντάκτη.
- **Αντικείμενο Πρόγραμμα (object):** Είναι το πρόγραμμα που παράγεται από τον μεταγλωττιστή.
- **Βιβλιοθήκες της γλώσσας:** Είναι τμήματα προγράμματος που είτε τα γράφει ο προγραμματιστής είτε υπάρχουν ήδη στην γλώσσα κατά την δημιουργία της.
- **Συνδέτης-Φορτωτής (linker-loader):** Πρόγραμμα που επιτρέπει την σύνδεση βιβλιοθηκών με το αντικείμενο πρόγραμμα ώστε να παραχθεί το εκτελέσιμο πρόγραμμα.
- **Εκτελέσιμο Πρόγραμμα (executable):** Είναι το τελικό πρόγραμμα που εκτελείται από τον υπολογιστή. Για τον λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση
  (Η δημιουργία του εκτελέσιμου προγράμματος γίνεται μόνο στην περίπτωσής που το αρχικό δεν περιέχει συντακτικά λάθη)



❖ **Διαφορές Διερμηνευτή και μεταγλωττιστή:**

1. Ο μεταγλωττιστής μεταφράζει όλες τις εντολές σε γλώσσα μηχανής και μετά τις εκτελεί. Ο διερμηνευτής μεταφράζει και εκτελεί επί τόπου μία εντολή κάθε φορά

2. Ο διερμηνευτής παρέχει τη δυνατότητα πιο γρήγορου ελέγχου και διόρθωσης των εντολών σε σχέση με το μεταγλωττιστή. Ο μεταγλωττιστής ωστόσο εξασφαλίζει ταχύτερη εκτέλεση του προγράμματος από το διερμηνευτή

❖ **Ποια προγράμματα και εργαλεία περιέχει ένα προγραμματιστικό περιβάλλον;**

Τα προγραμματιστικά περιβάλλοντα περιέχουν όλα τα προγράμματα και τα εργαλεία που απαιτούνται και βοηθούν τη συγγραφή, την εκτέλεση και κυρίως, τη διόρθωση των προγραμμάτων, δηλαδή περιέχουν τουλάχιστον 3 προγράμματα: τον **συντάκτη** , τον **μεταγλωττιστή** και τον **συνδέτη** .

❖ **Ποια είδη λαθών υπάρχουν σε ένα πρόγραμμα;**

- **Συντακτικά λάθη:** Δημιουργούνται από τους αναγραμματισμούς ονομάτων εντολών, παραλείψεις δήλωσης δεδομένων κατά τη διάρκεια υλοποίησης του προγράμματος. Τα συντακτικά λάθη εντοπίζονται στο στάδιο της μεταγλώττισης και διακόπτουν την εκτέλεση του προγράμματος. Για αυτό πρέπει οπωσδήποτε να διορθωθούν, ώστε να δημιουργηθεί το τελικό εκτελέσιμο πρόγραμμα.
- **Λογικά λάθη:** Είναι τα πιο σοβαρά και η διόρθωσης τους αρκετά δύσκολη. Δημιουργούνται από σφάλματα που γίνονται από τον προγραμματιστή κατά την υλοποίηση του προγράμματος. Τα λογικά λάθη δεν εντοπίζονται κατά την μεταγλώττιση του προγράμματος παρά μόνο κατά την εκτέλεση του. Είναι συνήθως λάθη σχεδιασμού και δεν προκαλούν διακοπή της εκτέλεσης του προγράμματος.
- **Λάθη κατά την εκτέλεση.**

Κεφάλαιο 4.1

❖ **Ποιος ο ρόλος των διαφορετικών προσεγγίσεων στη λύση των προβλημάτων;**

Η λύση σε ένα πρόβλημα μπορεί να προέλθει από ποικίλες και **διαφορετικές προσεγγίσεις**, τεχνικές και μεθόδους. Έτσι, είναι απαραίτητο να γίνεται μία **καλή ανάλυση** του κάθε προβλήματος και να προτείνεται συγκεκριμένη μεθοδολογία και ακολουθία βημάτων. Βασικός στόχος μας είναι η πρόταση έξυπνων και αποδοτικών λύσεων.

❖ **Ποια βήματα περιλαμβάνει η ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον;**

- την **καταγραφή της υπάρχουσας πληροφορίας** για το πρόβλημα,
- την **αναγνώριση των ιδιαιτεροτήτων** του προβλήματος,
- την **αποτύπωση των συνθηκών και προϋποθέσεων υλοποίησής** του
  - την **πρόταση επίλυσης με χρήση κάποιας μεθόδου**,
  - την **τελική επίλυση με χρήση υπολογιστικών συστημάτων**.

❖ **Γιατί είναι απαραίτητη η ανάλυση του προβλήματος** ;

Η ανάλυση κάθε προβλήματος είναι απαραίτητη, έτσι ώστε να αναζητηθεί **η πλέον κατάλληλη μέθοδος** που να παρέχει τη ζητούμενη λύση, όσο γίνεται **ταχύτερα** και με το λιγότερο δυνατό **κόστος σε υπολογιστικούς πόρους**.

❖ **Γιατί οι μέθοδοι ανάλυσης και επίλυσης των προβλημάτων παρουσιάζουν ιδιαίτερο ενδιαφέρον;**

- Παρέχουν ένα γενικό πρότυπο κατάλληλο για την επίλυση προβλημάτων ευρείας κλίμακας,
- Μπορούν να αναπαρασταθούν με κοινές δομές δεδομένων και ελέγχου (που υποστηρίζονται από τις περισσότερες σύγχρονες γλώσσες προγραμματισμού),
- Παρέχουν τη δυνατότητα καταγραφής των χρονικών και "χωρικών" απαιτήσεων της μεθόδου επίλυσης, έτσι ώστε να μπορεί να γίνει επακριβής εκτίμηση των αποτελεσμάτων.

Μέθοδος Διαίρει και Βασίλευε

❖ **Τι είναι η μέθοδος σχεδίασης αλγορίθμων «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).

Κεφάλαια 10: Υποπρογράμματα

- **Τμηματικός προγραμματισμός** ονομάζεται η τεχνική σχεδίασης και ανάπτυξης των προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων.

- Όταν ένα τμήμα προγράμματος επιτελεί ένα αυτόνομο έργο και έχει γραφεί χωριστά από το υπόλοιπο πρόγραμμα, τότε αναφερόμαστε σε **υποπρόγραμμα (subprogram).**

- **Υπάρχουν πάντως τρεις ιδιότητες που πρέπει να διακρίνουν τα υποπρογράμματα:**
  - Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο.
  - Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα.
  - Κάθε υποπρόγραμμα πρέπει να μην είναι πολύ μεγάλο.
- **Ο σωστός χωρισμός ενός σύνθετου προγράμματος σε υποπρογράμματα εξασφαλίζει τέσσερα βασικά χαρακτηριστικά του σωστού προγραμματισμού:**
  - Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος.
  - Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.
  - Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.
  - Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.
- **Παράμετρος:** είναι μία μεταβλητή που επιτρέπει το πέρασμα της τιμής της από ένα τμήμα προγράμματος σε ένα άλλο.
- **Συνάρτηση:** είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή με το όνομα της(όπως οι μαθηματικές συναρτήσεις)
- **Διαδικασία:** είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος.

Διαφορές των Διαδικασιών από τις Συναρτήσεις

Διαδικασίες	Συναρτήσεις
Οι διαδικασίες μπορούν να εκτελέσουν μια οποιαδήποτε λειτουργία, π.χ. να εισάγουν δεδομένα, να εκτελούν υπολογισμούς, να τυπώνουν αποτελέσματα και να αλλάζουν τις τιμές των μεταβλητών.	Οι συναρτήσεις υπολογίζουν μόνο μία τιμή και μόνο αυτή επιστρέφουν στο κύριο πρόγραμμα ή στο υποπρόγραμμα που τις κάλεσε. Η τιμή αυτή μπορεί να είναι ΑΡΙΘΜΗΤΙΚΗ (ΑΚΕΡΑΙΑ ή ΠΡΑΓΜΑΤΙΚΗ), ΧΑΡΑΚΤΗΡΑΣ ή ΛΟΓΙΚΗ.
Οι διαδικασίες μεταφέρουν τα αποτελέσματα στα άλλα υποπρογράμματα με τη χρήση παραμέτρων.	Οι συναρτήσεις μεταφέρουν το αποτέλεσμα στο κύριο πρόγραμμα ή στο υποπρόγραμμα που τις κάλεσε με το όνομά τους και όχι με τη χρήση παραμέτρων. Μοιάζουν με τις μαθηματικές συναρτήσεις.
Οι διαδικασίες εκτελούνται αν γράψουμε την εντολή ΚΑΛΕΣΕ και μετά το όνομα της διαδικασίας.	Οι συναρτήσεις εκτελούνται με τη χρήση του ονόματος τους μέσα σε οποιαδήποτε εντολή.

Δομή Συνάρτησης

```
1  ΣΥΝΑΡΤΗΣΗ Όνομα(Λίστα_παραμέτρων): Τύπος_Συνάρτησης
2  Τμήμα_δηλώσεων !Όπως και στο πρόγραμμα
3  ΑΡΧΗ
4    ΕΝΤΟΛΕΣ
5    Όνομα ← έκφραση
6    ΕΝΤΟΛΕΣ
7  ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Παρατηρήσεις:
Ο τύπος της συνάρτησης μπορεί να είναι: **ΑΚΕΡΑΙΑ, ΠΡΑΓΜΑΤΙΚΗ, ΧΑΡΑΚΤΗΡΑΣ, ΛΟΓΙΚΗ.**

Δομή Διαδικασίας

```
1  ΔΙΑΔΙΚΑΣΙΑ Όνομα(λίστα_παραμέτρων)
2  Τμήμα_δηλώσεων !Όπως και στο πρόγραμμα
3  ΑΡΧΗ
4    ΕΝΤΟΛΕΣ
5  ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

❖ **Η λίστα παραμέτρων**

Η λίστα των **τυπικών παραμέτρων** καθορίζει τις παραμέτρους στη **δήλωση** του υποπρογράμματος. (ορίσματα)
Η λίστα των **πραγματικών παραμέτρων** καθορίζει τις παραμέτρους στην **κλήση** του υποπρογράμματος. (παράμετροι)

❖ **Στοίβα χρόνου εκτέλεσης**

Όταν μία διαδικασία ή συνάρτηση καλείται από το κύριο πρόγραμμα, τότε η αμέσως επόμενη διεύθυνση του κύριου προγράμματος, που ονομάζεται διεύθυνση επιστροφής, αποθηκεύεται από το μεταφραστή σε μία στοίβα που ονομάζεται στοίβα χρόνου εκτέλεσης. Μετά την εκτέλεση της διαδικασίας ή της συνάρτησης η διεύθυνση επιστροφής απωθείται από τη στοίβα και έτσι ο έλεγχος του προγράμματος μεταφέρεται και πάλι στο κύριο πρόγραμμα.

❖ **Κανόνες για τη χρήση παραμέτρων**

1) Ο αριθμός των πραγματικών και των τυπικών παραμέτρων πρέπει να είναι ίδιος.
2) Κάθε πραγματική παράμετρος αντιστοιχεί στην τυπική παράμετρο που βρίσκεται στην **αντίστοιχη θέση**. Για παράδειγμα η πρώτη της λίστας των τυπικών παραμέτρων στην πρώτη της λίστας των πραγματικών παραμέτρων κοκ.
3) Η τυπική παράμετρος και η αντίστοιχη της πραγματική πρέπει να είναι του **ιδίου τύπου**.

❖ **Τι ονομάζεται εμβέλεια;**

Το τμήμα του προγράμματος που ισχύουν οι μεταβλητές λέγεται **εμβέλεια** (scope) μεταβλητών.

❖ **Τι ονομάζεται Απεριόριστη εμβέλεια;**

Απεριόριστη εμβέλεια: Σύμφωνα με αυτή την αρχή όλες οι μεταβλητές και όλες οι σταθερές είναι γνωστές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, άσχετα που δηλώθηκαν. Όλες οι μεταβλητές είναι καθολικές.

❖ **Ποιες μεταβλητές ονομάζονται καθολικές;**

Καθολικές ονομάζονται οι μεταβλητές που ισχύουν **σε οποιοδήποτε τμήμα** του προγράμματος, άσχετα που δηλώθηκαν και είναι γνωστές στο κύριο πρόγραμμα και όλα τα υποπρογράμματα.

❖ **Ποια τα μειονεκτήματα της απεριόριστης εμβέλειας;**

Η απεριόριστη εμβέλεια καταστρατηγεί την αρχή της αυτονομίας των υποπρογραμμάτων, δημιουργεί πολλά προβλήματα και τελικά είναι αδύνατη για μεγάλα προγράμματα με πολλά υποπρογράμματα, αφού ο καθένας που γράφει κάποιο υποπρόγραμμα πρέπει να γνωρίζει τα ονόματα όλων των μεταβλητών που χρησιμοποιούνται στα υπόλοιπα υποπρογράμματα.

❖ **Τι είναι η περιορισμένη εμβέλεια**

Η περιορισμένη εμβέλεια υποχρεώνει όλες τις μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, να δηλώνονται σε αυτό το τμήμα. Όλες οι μεταβλητές είναι τοπικές, ισχύουν δηλαδή για το υποπρόγραμμα στο οποίο δηλώθηκαν. Στη ΓΛΩΣΣΑ έχουμε περιορισμένη εμβέλεια.

❖ **Ποιες μεταβλητές ονομάζονται τοπικές;**

Τοπικές ονομάζονται οι μεταβλητές που ισχύουν για το υποπρόγραμμα στο οποίο δηλώθηκαν.

❖ **Ποια τα πλεονεκτήματα της περιορισμένης εμβέλειας;**

1) Απόλυτη αυτονομία όλων των υποπρογραμμάτων και
2) Δυνατότητα να χρησιμοποιείται οποιοδήποτε όνομα, χωρίς να ενδιαφέρει αν το ίδιο χρησιμοποιείται σε άλλο υποπρόγραμμα.

❖ **Τι είναι η Μερικώς περιορισμένη εμβέλεια;**

Σύμφωνα με τη μερικώς περιορισμένη εμβέλεια άλλες μεταβλητές είναι τοπικές και άλλες καθολικές. Κάθε γλώσσα προγραμματισμού έχει τους δικούς της κανόνες και μηχανισμούς για τον τρόπο και τις προϋποθέσεις που ορίζονται οι μεταβλητές ως τοπικές ή καθολικές.

❖ **Πλεονεκτήματα και μειονεκτήματα της Μερικώς περιορισμένης εμβέλειας;**

Η μερικώς περιορισμένη εμβέλεια προσφέρει μερικά πλεονεκτήματα στον πεπειραμένο προγραμματιστή, αλλά για τον αρχάριο περιπλέκει το πρόγραμμα δυσκολεύοντας την ανάπτυξή του.

Κεφάλαια 13: Εκσφαλμάτωση

❖ **Ποιες είναι οι βασικές κατηγορίες λαθών που είναι δυνατό να παρουσιαστούν στα προγράμματα;**

1) Λάθη κατά την υλοποίηση (Συντακτικά λάθη)
2) Λάθη κατά την εκτέλεση (Λάθη που οδηγούν σε αντικανονικό τερματισμό του προγράμματος)
3) Λογικά λάθη (Λογικά λάθη που παράγουν λανθασμένα αποτελέσματα)

❖ **Που οφείλονται συνήθως τα λάθη κατά την υλοποίηση και πότε γίνονται αντιληπτά;**

Τα λάθη κατά το χρόνο υλοποίησης προκαλούνται κυρίως από **λανθασμένη σύνταξη** εντολών προγράμματος. Τέτοια λάθη μπορεί να είναι η λανθασμένη συγγραφή μιας **δεσμευμένης λέξης** της γλώσσας προγραμματισμού ή η χρήση μιας **δομής ελέγχου χωρίς την εντολή τερματισμού** της.
Ανιχνεύονται από τον μεταγλωττιστή, ο οποίος εμφανίζει προς τον προγραμματιστή κάποιο προειδοποιητικό μήνυμα και δεν επιτρέπεται η εκτέλεσή του προγράμματος μέχρι να το διορθώσει ο προγραμματιστής.

❖ **Που οφείλονται συνήθως τα λάθη κατά την εκτέλεση και πότε γίνονται αντιληπτά;**

Τα λάθη που προκαλούν τον αντικανονικό τερματισμό της εφαρμογής και το κρέμασμα του συστήματος εμφανίζονται σε πραγματικό περιβάλλον εκτέλεσης. Τέτοια λάθη είναι δυνατό να προκληθούν από την προσπάθεια διαίρεσης ενός αριθμού με το μηδέν, την κλήση μιας διαδικασίας με δεδομένα που δεν μπορεί να χειριστεί, όπως η αναζήτηση διαγραμμένων αρχείων, η υπερχείλιση μιας αριθμητικής μεταβλητής ή από δυσλειτουργία του υλικού μέρους του υπολογιστή, όπως η καταστροφή του σκληρού δίσκου του συστήματος, ο τερματισμός μιας σύνδεσης δικτύου και η αποσύνδεση του εκτυπωτή.

❖ **Που οφείλονται συνήθως τα λογικά λάθη και πότε γίνονται αντιληπτά;**

Τα λογικά λάθη είναι συνήθως λάθη σχεδιασμού και δεν προκαλούν τη διακοπή της εκτέλεσης του προγράμματος. Ενώ ο μεταγλωττιστής της γλώσσας προγραμματισμού δεν ανιχνεύει κανένα συντακτικό λάθος και κατά την εκτέλεση του προγράμματος δεν παρουσιάζονται ανεπιθύμητες καταστάσεις σφαλμάτων, τελικά δεν παράγονται τα επιθυμητά αποτελέσματα. Η ανίχνευση τέτοιων λαθών δεν είναι δυνατό να πραγματοποιηθεί από κάποιο εργαλείο του υπολογιστή και διαπιστώνονται μόνο με τη διαδικασία ελέγχου (testing) και την ανάλυση των αποτελεσμάτων των προγραμμάτων.

❖ **Τι ονομάζεται εκσφαλμάτωση και ποιος ο στόχος της;**

Η διαδικασία ελέγχου, εντοπισμού και διόρθωσης των σφαλμάτων ενός προγράμματος καλείται εκσφαλμάτωση (debugging). Στόχος της διαδικασίας εκσφαλμάτωσης είναι ο εντοπισμός των σημείων του προγράμματος που προκαλούν προβλήματα στη λειτουργία του.

❖ **Ποιος ο ρόλος των σχολίων στην εκσφαλμάτωση;**

Η εισαγωγή γραμμών με σχόλια σε ένα πρόγραμμα υποβοηθά σημαντικά την εκσφαλμάτωση.
Χρησιμοποιούνται προκειμένου να διαχωρίζονται οι επεξηγηματικές φράσεις από τις λέξεις-κλειδιά του αλγορίθμου.

❖ **Εκσφαλμάτωση λογικών λαθών σε δομή 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 σε δομή 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.

❖ **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.

❖ **Τι προβλέπει ο έλεγχος «μαύρου κουτιού» (black-box testing).**

Κατά τον έλεγχο «μαύρου κουτιού», εκτελούνται σενάρια ελέγχου με **δεδομένα εισόδου** που προκύπτουν από τις προδιαγραφές του προγράμματος, **αγνοώντας εντελώς τον κώδικα**.

❖ **Τι είναι ένα σενάριο ελέγχου;**

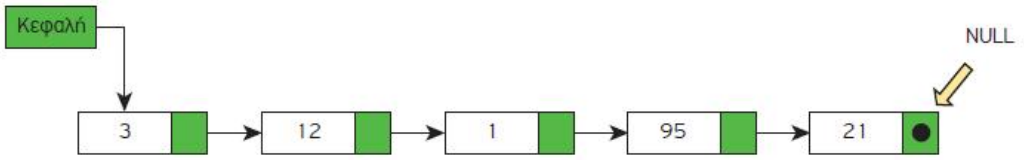
Ένα σενάριο ελέγχου περιγράφει τα δεδομένα εισόδου ολόκληρου του προγράμματος ή τμήματος του προγράμματος(διαδικασία,συνάρτηση) και τα αναμενόμενα αποτελέσματα.

❖ **Πώς γίνεται η επιλογή των δεδομένων στον έλεγχο μαύρου κουτιού;**

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.

❖ **Τι ονομάζεται συνδεδεμένη λίστα;**

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**.



❖ **Πώς γίνεται η προσπέλαση σε μια συνδεδεμένη λίστα;**

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.

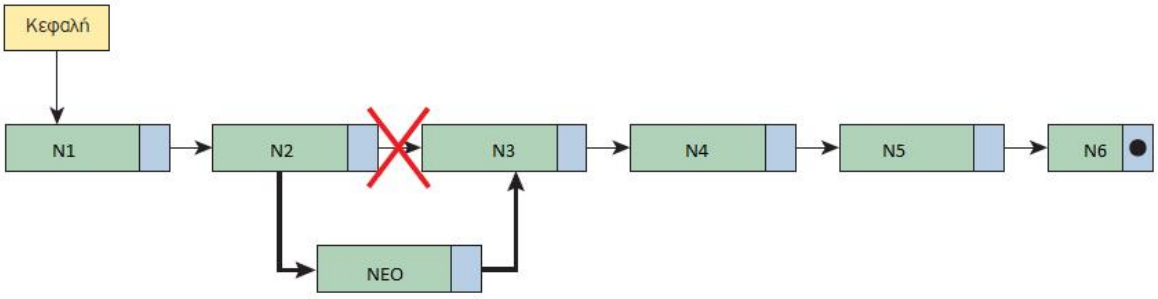
Έχουμε άμεση πρόσβαση μόνο στον πρώτο κόμβο της λίστας. Επομένως, για να εντοπίσουμε κάποιον από τους ενδιάμεσους κόμβους, πρέπει να ξεκινήσουμε από τον πρώτο κόμβο της λίστας και να ακολουθήσουμε τους δείκτες με τη σειρά, μέχρι να φτάσουμε στον επιθυμητό κόμβο.

❖ **Πώς προσθέτω κόμβο σε μια συνδεδεμένη λίστα;**

Οι απαιτούμενες ενέργειες για την εισαγωγή (παρεμβολή) του νέου κόμβου είναι :

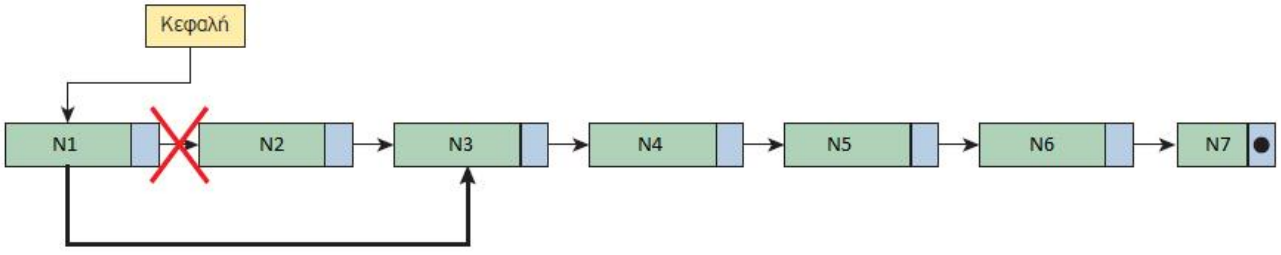
- ο δείκτης του κόμβου μετά τον οποίο θα γίνει η παρεμβολή (N2), να δείχνει τον νέο κόμβο (ΝΕΟ)
- και ο δείκτης του νέου κόμβου (ΝΕΟ), να δείχνει αυτό που έδειχνε ο προηγούμενος (στο N3) (δηλαδή, να πάρει την τιμή που είχε πριν την εισαγωγή ο δείκτης του N2).

Με αυτόν τον τρόπο, οι κόμβοι της λίστας διατηρούν τη λογική τους σειρά, αλλά οι φυσικές θέσεις στη μνήμη μπορεί να είναι τελείως διαφορετικές.



❖ **Πώς διαγράφω κόμβο από μια συνδεδεμένη λίστα;**

Για τη διαγραφή ενός κόμβου αρκεί ν' αλλάξει τιμή ο δείκτης του προηγούμενου κόμβου και να δείχνει πλέον στον επόμενο αυτού που διαγράφεται. Ο κόμβος που διαγράφηκε (ο δεύτερος) αποτελεί «άχρηστο δεδομένο» και ο χώρος μνήμης που καταλάμβανε, παραχωρείται για άλλη χρήση.



❖ **Τι είναι η διπλά συνδεδεμένη λίστα;**

Η διπλά συνδεδεμένη λίστα είναι μια λίστα στην οποία μπορούμε να τη διατρέξουμε και προς τις δύο κατευθύνσεις. Αυτό επιτυγχάνεται με τη χρήση του δεύτερου δείκτη που δείχνει τον προηγούμενο κόμβο. Προσφέρει έτσι τη δυνατότητα ξεκινώντας από οποιοδήποτε κόμβο της λίστας να μπορούμε να διατρέξουμε τη λίστα και προς τις δυο κατευθύνσεις.

❖ **Γιατί χρησιμοποιούνται οι συνδεδεμένες λίστες στην υλοποίηση της στοίβας και της ουράς;**

Οι συνδεδεμένες λίστες αξιοποιούνται για την υλοποίηση της στοίβας και της ουράς, λόγω της **δυνατότητάς αυξομείωσης του μεγέθους τους**.

❖ **Πότε θα χρησιμοποιήσουμε λίστα και πότε πίνακα για την υλοποίηση μιας στοίβας;**

Καλύτερη είναι η υλοποίηση στοίβας με συνδεδεμένη λίστα εκτός αν δεν **γνωρίζουμε** εκ των προτέρων τον **μέγιστο αριθμό** στοιχείων και θέλουμε μια εύκολη και γρήγορη λύση, οπότε επιλέγουμε την υλοποίηση με πίνακα.

❖ **Πώς μια ουρά μπορεί να υλοποιηθεί με μία διπλά συνδεδεμένη λίστα.**

Στην ουρά έχουμε τις εξής δύο κύριες λειτουργίες. Εισαγωγή στοιχείου στο πίσω άκρο της ουράς και εξαγωγή στοιχείου από το εμπρός άκρο της ουράς. Η ουρά μπορεί να υλοποιηθεί με μία διπλά συνδεδεμένη λίστα, αφού μπορούμε να κάνουμε **εισαγωγή** κόμβου στο **τέλος της διπλά συνδεδεμένης λίστας**. Επίσης μπορούμε να κάνουμε **διαγραφή** κόμβου στην **αρχή** της διπλά συνδεδεμένης λίστας (εξαγωγή στοιχείου από το εμπρός άκρο της ουράς).

❖ **Πώς μια στοίβα μπορεί να υλοποιηθεί με μία απλά συνδεδεμένη λίστα;**

Οι κόμβοι (στοιχεία) εισέρχονται από την αρχή της απλά συνδεδεμένης λίστας και εξέρχονται πάλι από την αρχή της λίστας τότε ο κόμβος (στοιχείο) που εισήλθε τελευταίος εξέρχεται και πρώτος (LIFO). Επομένως η στοίβα μπορεί να υλοποιηθεί με μία απλά συνδεδεμένη λίστα αφού μπορεί να γίνει **εισαγωγή κόμβου (στοιχείου) στην αρχή της λίστας και διαγραφή κόμβου (στοιχείου) από την αρχή της λίστας** .

❖ **Ποιες οι διαφορές ανάμεσα σε λίστες και πίνακες;**

1) 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.
2) 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.
3) Οι κόμβοι της **λίστας** αποθηκεύονται σε **μη συνεχόμενες θέσεις** μνήμης σε αντιδιαστολή με τους **πίνακες**, όπου τα στοιχεία αποθηκεύονται σε **συνεχόμενες** θέσεις μνήμης.

❖ **Ποια είναι τα πλεονεκτήματα των λιστών (έναντι των πινάκων) ;**

1) Το **δυναμικό** τους μέγεθος,
2) η ευκολία **εισαγωγής** και **διαγραφής** από οποιοδήποτε μέρος της λίστας, καθώς και
3) η **μη αναγκαιότητα δήλωσης του μεγέθους** τους

❖ **Ποια τα μειονεκτήματα των λιστών (έναντι των πινάκων);**

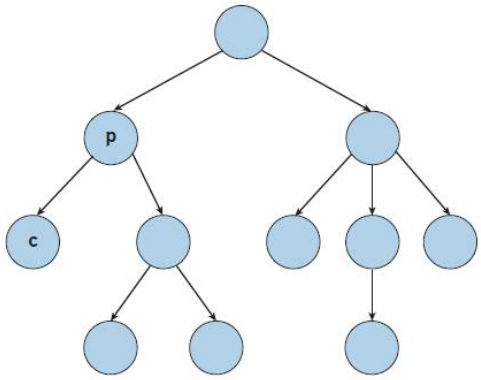
1) Η **τυχαία πρόσβαση** στη λίστα **δεν επιτρέπεται**. Είναι αδύνατο να φτάσετε στον n-οστό κόμβο μιας απλά συνδεδεμένης λίστας χωρίς πρώτα να περάσετε από όλους τους κόμβους διαδοχικά μέχρι τον συγκεκριμένο κόμβο ξεκινώντας από τον πρώτο κόμβο. Εναλλακτικά, στην περίπτωση της διπλά συνδεδεμένης λίστας μπορείτε να ξεκινήσετε και από τον τελευταίο κόμβο. Επομένως**, δεν μπορούμε** να πραγματοποιήσουμε με αποτελεσματικό τρόπο **δυαδική αναζήτηση** σε συνδεδεμένες λίστες.
2) 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.

❖ **Ποιες είναι οι βασικές πράξεις των συνδεδεμένων λιστών;**

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

❖ **Τι ονομάζεται δέντρο;**

Ένα **δένδρο** (tree) είναι μία **δομή** που αποτελείται από ένα **σύνολο κόμβων** και ένα **σύνολο ακμών** μεταξύ των κόμβων με βάση τους εξής **κανόνες**:
1) Υπάρχει ένας ξεχωριστός κόμβος που ονομάζεται **ρίζα**. Αυτός είναι ένας κόμβος **χωρίς γονέα**.
2) 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.
3) 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.



❖ **Σε ένα δέντρο ποιοι κόμβοι ονομάζονται φύλλα και ποιοι αδέλφια;**

Οι κόμβοι χωρίς παιδιά ονομάζονται «**φύλλα**»
Κόμβοι με τον ίδιο γονέα ονομάζονται «**αδέλφια**».

❖ **Ποια δέντρα ονομάζονται διατεταγμένα;**

Τα δέντρα στα οποία για κάθε κόμβο του υπάρχει μία **γραμμική σχέση μεταξύ των παιδιών του κόμβου αυτού**, αναφερόμαστε σε ένα διατεταγμένο δένδρο.

❖ **Τι ονομάζεται δένδρο του παιχνιδιού (game tree);**

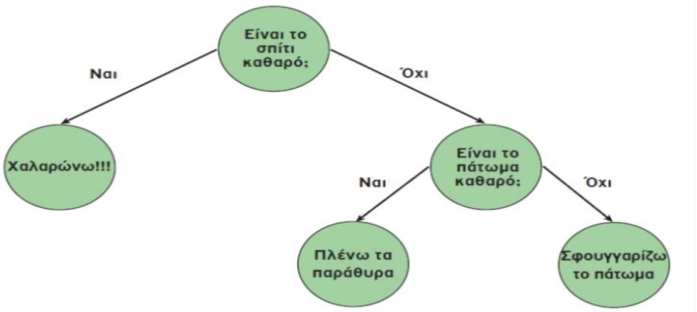
Στα παιχνίδια ο υπολογιστής χρησιμοποιεί ένα ειδικό δένδρο, που ονομάζεται **δένδρο του παιχνιδιού** (game tree), το οποίο **μοντελοποιεί όλες τις πιθανές κινήσεις των παικτών** για να νικήσει. Κάθε **κόμβος** στο δένδρο, αντιπροσωπεύει μία συγκεκριμένη κατάσταση παιχνιδιού και περιέχει **πληροφορίες** σχετικά με το ποιος παίκτης έχει τη **μεγαλύτερη πιθανότητα** να κερδίσει από οποιαδήποτε πιθανή **κίνηση**.

❖ **Τι ονομάζεται δένδρο απόφασης; Δώστε παράδειγμα.**

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.

Ένα απλό πρόβλημα λήψης απόφασης μπορεί να είναι αυτό της καθαριότητας του φοιτητικού σπιτιού σας.

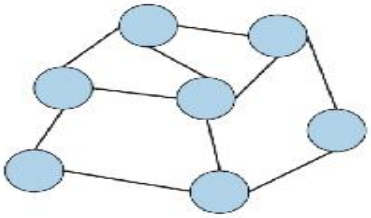


❖ **Πώς χρησιμοποιούνται τα δέντρα στον υπολογισμό αριθμητικών εκφράσεων;**
Για τον υπολογισμό αριθμητικών εκφράσεων στα φύλλα του δέντρου μπαίνουν οι τελεστέοι, ενώ στους εσωτερικούς κόμβους οι τελεστές.

❖ **Τι ονομάζεται δυαδικό δέντρο;**
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.

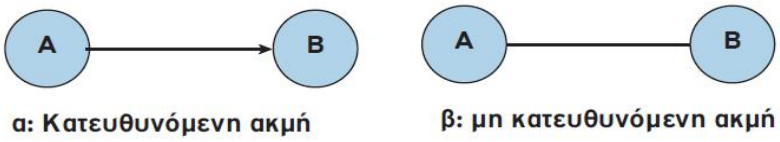
❖ **Τι ονομάζεται δυαδικό δέντρο αναζήτησης;**
Ένα δυαδικό δένδρο αναζήτησης (binary search tree) είναι ένα **δυαδικό δένδρο**, όπου για **κάθε κόμβο** u, **όλοι οι κόμβοι του αριστερού υποδένδρου** έχουν τιμές **μικρότερες** της τιμής του κόμβου **u** και όλοι οι κόμβοι του **δεξιού** υποδένδρου έχουν τιμές **μεγαλύτερες** (ή ίσες) της τιμής του κόμβου u.

❖ **Τι ονομάζεται γράφος;**
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**.



❖ **Ποια η συσχέτιση του γράφου με τις λίστες και τα δέντρα;**
Ο γράφος αποτελεί την πιο γενική δομή δεδομένων, με την έννοια ότι **όλες οι προηγούμενες δομές** που παρουσιάστηκαν μπορούν να θεωρηθούν περιπτώσεις **γράφων**.

❖ **Ποιοι οι τύποι του γράφου;**
1. Εάν όλες οι **ακμές** σε έναν γράφο έχουν **κατεύθυνση**, ο γράφος ονομάζεται **κατευθυνόμενος γράφος** (directed graph).
2. Εάν όλες οι ακμές σε έναν γράφο δεν έχουν κατεύθυνση, ο γράφος ονομάζεται **μη κατευθυνόμενος** γράφος (undirected graph) και η διαδρομή μεταξύ των δύο κόμβων είναι αμφίδρομη.



Αντικειμενοστραφής Προγραμματισμός

❖ **Τι ονομάζεται Αντικειμενοστραφής προγραμματισμός και τι τα Αντικείμενα;**

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.

❖ **Τι ονομάζουμε ιδιότητες και τι μεθόδους στα αντικείμενα;**

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).**

❖ **Ποια είναι τα βήματα στη μεθοδολογία επίλυσης προβλημάτων στην αντικειμενοστραφή προσέγγιση;**

Για να αναλύσουμε το πρόβλημα το οποίο θέλουμε να επιλύσουμε, πρέπει να αναγνωρίσουμε και να καταγράψουμε 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, δηλ. τα σχετικά με το συγκεκριμένο πρόβλημα χαρακτηριστικά του

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.

❖ **Τι περιλαμβάνει η διαγραμματική αναπαράσταση των αντικειμένων;**

Αφού εντοπίσουμε τα συστατικά επίλυσης του προβλήματος, μπορούμε να τα οργανώσουμε σε μια απλή **διαγραμματική αναπαράσταση** χρησιμοποιώντας **παραλληλόγραμμα** για την αποτύπωση των **αντικειμένων**, των **ιδιοτήτων** και των **μεθόδων** τους και **γραμμές σύνδεσης** για την περιγραφή του είδους της μεταξύ τους συνεργασίας .

❖ **Ποια η δομή ενός αντικειμενοστραφούς προγράμματος;**

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.

❖ **Τι ονομάζεται ενθυλάκωση;**

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).

❖ **Τι ονομάζεται 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**.

- **Τι ονομάζεται κληρονομικότητα και ποια τα χαρακτηριστικά από τις κλάσεις προγόνοι-απόγονοι;**

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).

- **Πώς αποτυπώνεται η σχέση κληρονομικότητας στη διαγραμματική αναπαράσταση;**

Η διαγραμματική αναπαράσταση της σχέσης κληρονομικότητας που μόλις περιγράψαμε γίνεται με τη βοήθεια του ειδικού συμβόλου γενίκευσης(βελάκι).

- **Πότε μπορεί να θεωρηθεί ως έγκυρη μια υποκλάση;**

Μια κλάση Α μπορεί να είναι έγκυρη υποκλάση της Β αν έχει νόημα να πούμε «**ένα Α είναι ένα (is_a) Β»** (προσοχή το αντικείμενο της κλάσης δεν αποτελεί υποκλάση!)

- **Τι ονομάζεται πολυμορφισμός;**

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

- **Ποια είναι τα πρωτεύοντα δομικά στοιχεία ενός προγράμματος στην αντικειμενοστραφή σχεδίαση**;

Η αντικειμενοστραφής σχεδίαση εκλαμβάνει ως πρωτεύοντα δομικά στοιχεία ενός προγράμματος τα **δεδομένα**, από τα οποία δημιουργούνται με κατάλληλη μορφοποίηση τα **αντικείμενα** (objects).

- **Ποια τα πλεονεκτήματα της αντικειμενοστραφούς σχεδίασης;**

Η αντικειμενοστραφής σχεδίαση αποδείχθηκε ότι επιφέρει καλύτερα αποτελέσματα, αφού τα προγράμματα που δημιουργούνται είναι περισσότερο ευέλικτα και επαναχρησιμοποιήσιμα.

- **Ο αντικειμενοστραφής προγραμματισμός χρησιμοποιεί τεχνικές του κλασσικού διαδικασιακού προγραμματισμού;**

Ο αντικειμενοστραφής προγραμματισμός εκτός από τον τρόπο που χειρίζεται τα δεδομένα, συνεχίζει να χρησιμοποιεί την ιεραρχική σχεδίαση, τον τμηματικό προγραμματισμό και να ακολουθεί τις αρχές του δομημένου προγραμματισμού.