ΚΕΦΑΛΑΙΟ 1–ΑΝΑΛΥΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

ΟΡΙΣΜΟΣ ΠΡΟΒΛΗΜΑΤΟΣ

Με τον όρο Πρόβλημα εννοείται μια κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, η δε λύση της δεν είναι γνωστή, ούτε προφανής.

ΚΑΤΑΝΟΗΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

Η κατανόηση ενός προβλήματος αποτελεί συνάρτηση δυο παραγόντων: της σωστής διατύπωσης εκ μέρους του δημιουργού του και της σωστής ερμηνείας από τη μεριά εκείνου που καλείται να το αντιμετωπίσει.

ΔΕΔΟΜΕΝΟ, ΠΛΗΡΟΦΟΡΙΑ, ΕΠΕΞΕΡΓΑΣΙΑ ΔΕΔΟΜΕΝΩΝ

Με τον όρο **δεδομένο** δηλώνεται οποιοδήποτε στοιχείο μπορεί να γίνει αντιληπτό από έναν τουλάχιστον παρατηρητή με μία από τις πέντε αισθήσεις του.
Με τον όρο **πληροφορία** αναφέρεται οποιοδήποτε γνωσιακό στοιχείο προέρχεται από επεξεργασία δεδομένων.
Ο όρος **επεξεργασία δεδομένων** δηλώνει εκείνη τη διαδικασία κατά την οποία ένας "μηχανισμός" δέχεται δεδομένα, τα επεξεργάζεται σύμφωνα με έναν προκαθορισμένο τρόπο και αποδίδει πληροφορίες.

Επί χιλιετίες ο "μηχανισμός" επεξεργασίας των δεδομένων ήταν και εξακολουθεί να είναι ο ανθρώπινος εγκέφαλος. Στις μέρες μας, ένας άλλος "μηχανισμός" επεξεργασίας δεδομένων είναι ο υπολογιστής.

ΔΟΜΗ ΠΡΟΒΛΗΜΑΤΟΣ

Με τον όρο δομή ενός προβλήματος αναφερόμαστε στα συστατικά του μέρη, στα επιμέρους τμήματα που το αποτελούν καθώς επίσης και στον τρόπο που αυτά τα μέρη συνδέονται μεταξύ τους.

ΚΑΘΟΡΙΣΜΟΣ ΑΠΑΙΤΗΣΕΩΝ ΠΡΟΒΛΗΜΑΤΟΣ

Η σωστή επίλυση ενός προβλήματος προϋποθέτει τον επακριβή προσδιορισμό των δεδομένων που παρέχει το πρόβλημα. Απαιτεί επίσης τη λεπτομερειακή καταγραφή των ζητούμενων που αναμένονται σαν αποτελέσματα της επίλυσης του προβλήματος.

ΣΤΑΔΙΑ ΑΝΤΙΜΕΤΩΠΙΣΗΣ ΠΡΟΒΛΗΜΑΤΟΣ

Τα στάδια αντιμετώπισης ενός προβλήματος είναι τρία:

- κατανόηση, όπου απαιτείται η σωστή και πλήρης αποσαφήνιση των δεδομένων και των ζητούμενων του προβλήματος
- ανάλυση, όπου το αρχικό πρόβλημα διασπάται σε άλλα επιμέρους απλούστερα προβλήματα
- επίλυση, όπου υλοποιείται η λύση του προβλήματος, μέσω της λύσης των επιμέρους προβλημάτων.

ΚΕΦΑΛΑΙΟ 2-ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΑΛΓΟΡΙΘΜΩΝ
ΚΕΦΑΛΑΙΟ 7-ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ
ΚΕΦΑΛΑΙΟ 8-ΕΠΙΛΟΓΗ ΚΑΙ ΕΠΑΝΑΛΗΨΗ

ΟΡΙΣΜΟΣ ΑΛΓΟΡΙΘΜΟΥ

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

ΚΡΙΤΗΡΙΑ ΠΟΥ ΠΡΕΠΕΙ ΝΑ ΙΚΑΝΟΠΟΙΕΙ ΚΑΘΕ ΑΛΓΟΡΙΘΜΟΣ

- ✓ **ΕΙΣΟΔΟΣ** (καμία, μία ή περισσότερες τιμές)
- ✓ **ΕΞΟΔΟΣ** (τουλάχιστον μια τιμή)
- ✓ **ΚΑΘΟΡΙΣΤΙΚΟΤΗΤΑ** (κάθε εντολή πρέπει να καθορίζεται χωρίς καμιά αμφιβολία για τον τρόπο εκτέλεσής της)
- ✓ **ΠΕΡΑΤΟΤΗΤΑ** (πρέπει να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του)
- ✓ **ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ** (κάθε μια εντολή πρέπει να είναι απλή και εκτελέσιμη)

ΑΝΑΠΑΡΑΣΤΑΣΗ ΑΛΓΟΡΙΘΜΩΝ

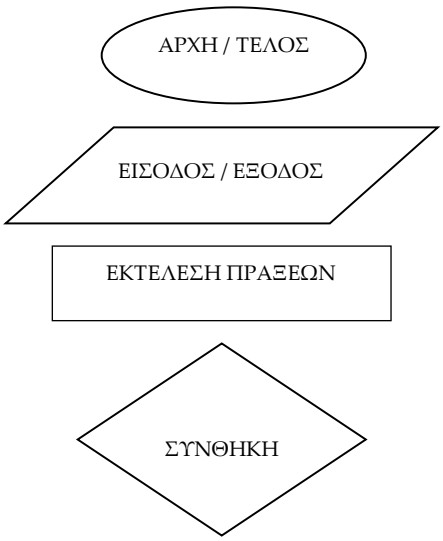
Γίνεται με τους παρακάτω τρόπους:

- ✓ **ΕΛΕΥΘΕΡΟ ΚΕΙΜΕΝΟ**
- ✓ **ΔΙΑΓΡΑΜΜΑΤΙΚΕΣ ΤΕΧΝΙΚΕΣ (διάγραμμα ροής)**
- ✓ **ΦΥΣΙΚΗ ΓΛΩΣΣΑ ΚΑΤΑ ΒΗΜΑΤΑ**
- ✓ **ΚΩΔΙΚΟΠΟΙΗΣΗ (ψευδογλώσσα)**

Εμείς θα σχεδιάζουμε αλγορίθμους με διαγραμματικές τεχνικές και κωδικοποίηση.

ΔΙΑΓΡΑΜΜΑΤΑ ΡΟΗΣ

Χρησιμοποιούν τα παρακάτω σχήματα:



ΈΛΛΕΙΨΗ

ΠΛΑΓΙΟ ΠΑΡΑΛΛΗΛΟΓΡΑΜΜΟ

ΟΡΘΟΓΩΝΙΟ

ΡΟΜΒΟΣ

ΠΩΣ ΔΗΜΙΟΥΡΓΟΥΜΕ ΠΡΟΓΡΑΜΜΑΤΑ ΜΕ ΚΩΔΙΚΟΠΟΙΗΣΗ (ΓΛΩΣΣΑ)

1.Κάθε πρόγραμμα ξεκινάει με την λέξη **ΠΡΟΓΡΑΜΜΑ** ακολουθούμενη από ένα όνομα που δίνουμε στον πρόγραμμα. Το όνομα που δίνουμε στο πρόγραμμα μπορεί να αποτελείται από γράμματα της ελληνικής ή λατινικής αλφαβήτου και αριθμούς. Το όνομα **ΔΕΝ** πρέπει να ξεκινάει με αριθμό
π.χ. δεν είναι σωστό να γράψουμε ΠΡΟΓΡΑΜΜΑ 1ΑΣΚΗΣΗ
Το σωστό θα ήταν να γράψουμε ΠΡΟΓΡΑΜΜΑ ΑΣΚΗΣΗ1
Επίσης αν το όνομα του προγράμματος αποτελείται από περισσότερες από μία λέξη **ΔΕΝ** επιτρέπεται να αφήνουμε κενό ανάμεσα σε αυτές αλλά χρησιμοποιούμε σαν διαχωριστικό την κάτω παύλα (_)
Π.χ. δεν είναι σωστό να γράψουμε ΠΡΟΓΡΑΜΜΑ Γ ΤΑΞΗ ΛΥΚΕΙΟΥ
Το σωστό θα ήταν να γράψουμε ΠΡΟΓΡΑΜΜΑ Γ_ΤΑΞΗ_ΛΥΚΕΙΟΥ

2.Κάθε πρόγραμμα χρησιμοποιεί τις **ΜΕΤΑΒΛΗΤΕΣ** δηλ. μεγέθη η τιμή των οποίων μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης του αλγορίθμου και τις **ΣΤΑΘΕΡΕΣ** δηλ. μεγέθη των οποίων η τιμή παραμένει αμετάβλητη κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Οι μεταβλητές και οι σταθερές έχουν κι αυτές ονόματα που τις δίνουμε εμείς και για τα ονόματά τους ισχύουν οι περιορισμοί που προαναφέραμε παραπάνω για τα ονόματα των προγραμμάτων.

3. Οι **ΣΤΑΘΕΡΕΣ** και οι **ΜΕΤΑΒΛΗΤΕΣ** διακρίνονται σε 4 κατηγορίες ανάλογα με το είδος των τιμών που μπορούν να πάρουν:

- **ΑΚΕΡΑΙΕΣ** π.χ 0, 15, -25
- **ΠΡΑΓΜΑΤΙΚΕΣ** π.χ 0.2, 12.12, -3.966
- **ΧΑΡΑΚΤΗΡΕΣ** π.χ. 'μ', 'μπαταρία', '15'. Οι μεταβλητές τύπου χαρακτήρα μπαίνουν πάντοτε μέσα σε μονά εισαγωγικά '…'
- **ΛΟΓΙΚΕΣ** όταν παίρνουν δυο τιμές ΑΛΗΘΗΣ (TRUE) ή ΨΕΥΔΗΣ (FALSE)

Η δήλωση των σταθερών και των μεταβλητών γίνεται πάντοτε μετά την δήλωση του ονόματος του προγράμματος, γράφοντας π.χ.:

ΣΤΑΘΕΡΕΣ
Π=3.14
ΜΕΤΑΒΛΗΤΕΣ
ΑΚΕΡΑΙΕΣ: Χ
ΠΡΑΓΜΑΤΙΚΕΣ: Υ

4.Στην αμέσως επόμενη σειρά γράφουμε **ΠΑΝΤΑ** την λέξη **ΑΡΧΗ**

5.Στις επόμενες σειρές γράφουμε τις **ΕΝΤΟΛΕΣ** φροντίζοντας να ξεκινούν πιο δεξιά από την **ΑΡΧΗ.** Οι ΕΝΤΟΛΕΣ χωρίζονται σε τρείς κατηγορίες:

- **ΕΝΤΟΛΗ ΕΙΣΟΔΟΥ** που είναι η **ΔΙΑΒΑΣΕ**
- **ΕΝΤΟΛΗ ΕΞΟΔΟΥ** που είναι η **ΓΡΑΨΕ**, η **ΕΚΤΥΠΩΣΕ** και η **ΕΜΦΑΝΙΣΕ**
- **ΕΝΤΟΛΗ ΕΚΧΩΡΗΣΗΣ** όπου εκχωρούμε σε μια μεταβλητή μια τιμή ή το αποτέλεσμα μιας πράξης. Κατά την εκχώρηση η μεταβλητή χάνει το περιεχόμενο που πιθανώς είχε πριν. Η εντολή εκχώρησης συντάσσετε με το σύμβολο ← .Αριστερά του συμβόλου υπάρχει **μόνο μια μεταβλητή** ενώ δεξιά μπορεί να υπάρχει αριθμός, μεταβλητή, μαθηματική παράσταση.

6.Τέλος, ένα πρόγραμμα κλείνει **ΠΑΝΤΑ** με την εντολή **ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ**

Μια γενική λοιπόν δομή ενός προγράμματος είναι η ακόλουθη:

ΠΡΟΓΡΑΜΜΑ ΑΣΚΗΣΗ_1
ΣΤΑΘΕΡΕΣ
ΟΝΟΜΑ=ΣΤΑΘΕΡΗ ΤΙΜΗ
ΜΕΤΑΒΛΗΤΕΣ
ΑΚΕΡΑΙΕΣ:ΟΝΟΜΑΤΑ ΜΕΤΑΒΛΗΤΩΝ
ΠΡΑΓΜΑΤΙΚΕΣ:ΟΝΟΜΑΤΑ ΜΕΤΑΒΛΗΤΩΝ
ΑΡΧΗ
    ΔΙΑΒΑΣΕ ΜΕΤΑΒΛΗΤΗ1
    ΔΙΑΒΑΣΕ ΜΕΤΑΒΛΗΤΗ2
    ΓΡΑΨΕ ΜΕΤΑΒΛΗΤΗ 3
ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

7.Τα σύμβολα των αριθμητικών πράξεων που εκτελούν οι αλγόριθμοι/προγράμματα είναι τα παρακάτω:

ΠΡΟΣΘΕΣΗ	+
ΑΦΑΙΡΕΣΗ	-
ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ	*
ΔΙΑΙΡΕΣΗ	/
ΥΨΩΣΗ ΣΕ ΔΥΝΑΜΗ	^
ΠΗΛΙΚΟ ΑΚΕΡΑΙΗΣ ΔΙΑΙΡΕΣΗΣ	div
ΥΠΟΛΟΙΠΟ ΑΚΕΡΑΙΗΣ ΔΙΑΙΡΕΣΗΣ	mod

Όταν πρέπει να υπολογίσουμε σύνθετες μαθηματικές παραστάσεις κάνουμε σωστή χρήση παρενθέσεων με προτεραιότητα εκτέλεσης όπως γνωρίζουμε από τα Μαθηματικά.

8.Κάποιες λέξεις οι οποίες έχουν αυστηρά προκαθορισμένη σημασία ονομάζονται **δεσμευμένες** και δεν μπορούμε να τις χρησιμοποιήσουμε σαν όνομα του προγράμματος ή των μεταβλητών. Τέτοιες λέξεις είναι οι ΑΛΓΟΡΙΘΜΟΣ, ΑΡΧΗ, ΤΕΛΟΣ, ΔΙΑΒΑΣΕ, ΓΡΑΨΕ.

ΔΟΜΕΣ ΑΛΓΟΡΙΘΜΩΝ / ΠΡΟΓΡΑΜΜΑΤΩΝ

Υπάρχουν τρεις βασικές δομές που χρησιμοποιούμε στους αλγορίθμους και στα προγράμματα:

- ✓ ΔΟΜΗ ΑΚΟΛΟΥΘΙΑΣ
- ✓ ΔΟΜΗ ΕΠΙΛΟΓΗΣ
- ✓ ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ

ΔΟΜΗ ΑΚΟΛΟΥΘΙΑΣ

Είναι η πιο απλή δομή, κατά την οποία η σειρά εκτέλεσης ενός συνόλου ενεργειών είναι δεδομένη δηλαδή οι εντολές ακολουθούν η μια την άλλη.

ΔΟΜΗ ΕΠΙΛΟΓΗΣ

Η δομή της επιλογής περιλαμβάνει τον έλεγχο κάποιας **λογικής συνθήκης** που μπορεί να έχει δυο τιμές (ΑΛΗΘΗΣ ή ΨΕΥΔΗΣ) και ακολουθεί η απόφαση εκτέλεσης κάποιας ενέργειας με βάση την τιμή αυτής της λογικής συνθήκης π.χ. αν βρέχει θα πάρω μαζί μου ομπρέλα διαφορετικά δεν θα πάρω δηλ. αν η συνθήκη(βροχή) είναι ΑΛΗΘΗΣ θα πάρω μαζί μου ομπρέλα ενώ αν είναι ΨΕΥΔΗΣ δεν θα πάρω.

Οι **λογικές συνθήκες** χρησιμοποιούν συνήθως τους παρακάτω **τελεστές σύγκρισης**:

>	Μεγαλύτερο	Π.χ. Χ>10
<	Μικρότερο	Π.Χ Χ<Υ
>=	Μεγαλύτερο ή ίσο	Π.χ. Χ>=20
=<	Μικρότερο ή ίσο	Π.χ Χ=<0
=	Ίσο	Π.χ. Χ=Υ
<>	διάφορο	Π.χ Χ<>Υ

Όταν αριθμητικοί και συγκριτικοί τελεστές τελεστές συνδυάζονται σε μια έκφραση, οι αριθμητικές πράξεις εκτελούνται πρώτες.

Υπάρχουν όμως και πιο σύνθετες λογικές συνθήκες, οι οποίες σχηματίζονται με την χρήση των παρακάτω **λογικών τελεστών:**

ΚΑΙ	ΣΥΖΕΥΞΗ	Π.χ Χ<10 **ΚΑΙ** Χ>20
Η	ΔΙΑΖΕΥΞΗ	Π.χ Χ<>0 **Η** Χ>3
ΟΧΙ	ΑΡΝΗΣΗ	Π.χ **ΟΧΙ** Χ>2

Έστω ότι έχουμε δυο συνθήκες, την ΣΥΝΘΗΚΗ Α και την ΣΥΝΘΗΚΗ Β. Ο παρακάτω πίνακας μας δείχνει τις τιμές των λογικών πράξεων για όλους τους συνδυασμούς τιμών:

ΣΥΝΘΗΚΗ Α	**ΣΥΝΘΗΚΗ Β**	Α **Η** Β	Α **ΚΑΙ** Β	**ΟΧΙ** Α
ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ
ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ
ΨΕΥΔΗΣ	ΑΛΗΘΗΣ	ΑΛΗΘΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ
ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΨΕΥΔΗΣ	ΑΛΗΘΗΣ

Δηλαδή η λογική πράξη **Η** είναι ΑΛΗΘΗΣ όταν οποιαδήποτε από τις δυο συνθήκες είναι ΑΛΗΘΗΣ.
Η λογική πράξη **ΚΑΙ** είναι ΑΛΗΘΗΣ μόνο όταν και οι δυο συνθήκες είναι ΑΛΗΘΕΙΣ.
Η λογική πράξη **ΟΧΙ** είναι ΑΛΗΘΗΣ όταν η πρόταση που την ακολουθεί είναι ΨΕΥΔΗΣ

Οι λογικοί τελεστές έχουν χαμηλότερη ιεραρχία από τους συγκριτικούς.

Η πιο απλή σύνταξη της δομής επιλογής είναι η ΑΠΛΗ ΕΠΙΛΟΓΗ:

ΑΝ <λογική συνθήκη> ΤΟΤΕ
ΕΝΤΟΛΗ/ΕΣ
ΤΕΛΟΣ_ΑΝ

ΠΑΡΑΔΕΙΓΜΑ:

```
ΑΝ Χ>0 ΤΟΤΕ
        ΓΡΑΨΕ 'ΘΕΤΙΚΟΣ'
ΤΕΛΟΣ_ΑΝ
```

Η γενική σύνταξη της σύνθετης δομή επιλογής είναι η ΣΥΝΘΕΤΗ ΕΠΙΛΟΓΗ:

ΑΝ <λογική συνθήκη> ΤΟΤΕ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ1
ΑΛΛΙΩΣ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ2
ΤΕΛΟΣ_ΑΝ

ΠΑΡΑΔΕΙΓΜΑ:
```
ΑΝ Χ>0 ΤΟΤΕ
        ΕΜΦΑΝΙΣΕ 'ΘΕΤΙΚΟΣ'
ΑΛΛΙΩΣ
        ΕΜΦΑΝΙΣΕ 'ΜΗ ΘΕΤΙΚΟΣ'
ΤΕΛΟΣ_ΑΝ
```

Υπάρχουν ωστόσο περιπτώσεις όπου μπορεί να ληφθούν παραπάνω από δυο διαφορετικές αποφάσεις ανάλογα με την τιμή που παίρνει μια έκφραση. Σε αυτές τις περιπτώσεις χρησιμοποιούμε την **ΠΟΛΛΑΠΛΗ ΕΠΙΛΟΓΗ.** Υπάρχουν δυο τρόποι για να συντάξουμε πολλαπλές επιλογές:

Α' τρόπος

ΑΝ <λογική συνθήκη1> ΤΟΤΕ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ1
ΑΛΛΙΩΣ_ΑΝ <λογική συνθήκη2> ΤΟΤΕ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ2
ΑΛΛΙΩΣ_ΑΝ <λογική συνθήκη3> ΤΟΤΕ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ3
.
.
ΑΛΛΙΩΣ
ΟΜΑΔΑ ΕΝΤΟΛΩΝν
ΤΕΛΟΣ_ΑΝ

ΠΑΡΑΔΕΙΓΜΑ
```
ΑΝ Χ>0 ΤΟΤΕ
        ΕΜΦΑΝΙΣΕ 'ΘΕΤΙΚΟΣ'
ΑΛΛΙΩΣ_ΑΝ Χ<0 ΤΟΤΕ
        ΕΜΦΑΝΙΣΕ 'ΑΡΝΗΤΙΚΟΣ'
ΑΛΛΙΩΣ
        ΕΜΦΑΝΙΣΕ 'ΜΗΔΕΝ'
ΤΕΛΟΣ_ΑΝ
```

Β' τρόπος

ΕΠΙΛΕΞΕ <έκφραση>
ΠΕΡΙΠΤΩΣΗ <λίστα τιμών 1>
ΟΜΑΔΑ ΕΝΤΟΛΩΝ1
ΠΕΡΙΠΤΩΣΗ <λίστα τιμών 2>
ΟΜΑΔΑ ΕΝΤΟΛΩΝ2
...
ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
ΟΜΑΔΑ ΕΝΤΟΛΩΝ3
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ

ΠΑΡΑΔΕΙΓΜΑ
```
ΕΠΙΛΕΞΕ Χ
ΠΕΡΙΠΤΩΣΗ >0
        ΕΜΦΑΝΙΣΕ 'ΘΕΤΙΚΟΣ'
ΠΕΡΙΠΤΩΣΗ <0
        ΕΜΦΑΝΙΣΕ 'ΑΡΝΗΤΙΚΟΣ'
ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
        ΕΜΦΑΝΙΣΕ 'ΜΗΔΕΝ'
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

Τέλος, μια άλλη μορφή που μπορεί να έχει η δομή επιλογής είναι αυτή της **εμφωλευμένης δομής,** όπου μια ή περισσότερες δομές επιλογής περιέχονται μέσα σε μια δομή επιλογής. Π.χ.

```
ΑΝ Χ>=10 ΤΟΤΕ
        ΑΝ Χ>18 ΤΟΤΕ
                ΓΡΑΨΕ 'ΠΑΙΡΝΕΙΣ ΑΡΙΣΤΕΙΟ'
        ΑΛΛΙΩΣ
                ΓΡΑΨΕ 'ΔΕΝ ΠΑΙΡΝΕΙΣ ΑΡΙΣΤΕΙΟ'
        ΤΕΛΟΣ_ΑΝ
ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΕΠΑΝΑΛΑΜΒΑΝΕΙΣ ΤΗΝ ΤΑΞΗ'
ΤΕΛΟΣ_ΑΝ
```

ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ

Η δομή επανάληψης εφαρμόζεται όταν μια ομάδα εντολών πρέπει να εκτελεστεί πολλές φορές, ανάλογα με την τιμή μιας συνθήκης.
Υπάρχουν τρεις δομές επανάληψης:

1η ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ

ΟΣΟ <συνθήκη> ΕΠΑΝΑΛΑΒΕ
Εντολές
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

Οι εντολές εκτελούνται όσο η συνθήκη είναι αληθής. Η δομή επανάληψης τερματίζεται όταν η συνθήκη γίνει ψευδής(αυτό μπορεί να συμβεί από την αρχή κι η δομή επανάληψης να μην εκτελεστεί ούτε μια φορά)

2η ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ

ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
Εντολές
ΜΕΧΡΙΣ_ΟΤΟΥ <συνθήκη>

Οι εντολές εκτελούνται μέχρι η συνθήκη να γίνει αληθής.

Οι διαφορές της δομής ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ …ΜΕΧΡΙΣ_ΟΤΟΥ με την δομή ΟΣΟ…ΕΠΑΝΑΛΑΒΕ είναι οι εξής:

- ✓ **Η δομή ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ …ΜΕΧΡΙΣ_ΟΤΟΥ εκτελείται τουλάχιστον μια φορά ενώ η δομή ΟΣΟ…ΕΠΑΝΑΛΑΒΕ υπάρχει περίπτωση να μην εκτελεστεί ούτε μια φορά**
- ✓ **Στη δομή ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ …ΜΕΧΡΙΣ_ΟΤΟΥ η συνθήκη βρίσκεται στο τέλος ενώ στη δομή ΟΣΟ…ΕΠΑΝΑΛΑΒΕ η συνθήκη βρίσκεται στην αρχή**
- ✓ **Η δομή ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ …ΜΕΧΡΙΣ_ΟΤΟΥ εκτελείται μέχρι η συνθήκη να γίνει αληθής ενώ η δομή ΟΣΟ…ΕΠΑΝΑΛΑΒΕ εκτελείται όσο η συνθήκη είναι αληθής.**

3η ΔΟΜΗ ΕΠΑΝΑΛΗΨΗΣ

ΓΙΑ <μεταβλητή> ΑΠΟ <αρχική τιμή> ΜΕΧΡΙ <τελική τιμή> με_βήμα<μεταβολή τιμής>
Εντολές
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

Η δομή αυτή χρησιμοποιείται όταν γνωρίζουμε εκ των προτέρων τον αριθμό των επαναλήψεων.

Το βήμα μπορεί να είναι θετικός ή αρνητικός αριθμός και μας δείχνει κατά πόσο θα μεταβληθεί η τιμή της μεταβλητής. Το βήμα δεν μπορεί να είναι μηδέν γιατί ο αλγόριθμος δεν θα είχε νόημα εφόσον η επανάληψη δεν θα τερματιζόταν ποτέ.

ΕΛΕΓΧΟΣ ΕΓΚΥΡΟΤΗΤΑΣ

Γίνεται οταν μας ζητείται κάποιες μεταβλητές να παίρνουν ένα συγκεκριμένο εύρος τιμών κι αν δοθεί τιμή εκτός αυτού του εύρους τότε η τιμή αυτή να μη γινεται δεκτή και να ζητείται να δοθεί μια αποδεκτή τιμή. Για να κάνουμε έλεγχο εγκυρότητας χρησιμοποιούμε είτε τη δομή επανάληψης ΟΣΟ...ΕΠΑΝΑΛΑΒΕ είτε τη δομή επανάληψης ΜΕΧΡΙΣ_ΟΤΟΥ.
Έστω οτι θέλουμε να διαβάσουμε τον μισθό ενός υπαλλήλου και να κάνουμε έλεγχο εγκυρότητας ετσι ώστε ο μισθός να παίρνει μόνο θετικές τιμές. Σε περίπτωση που δοθεί μια μη θετική τιμή,αυτή να μη γίνεται δεκτή και να ζητήται να ξαναδοθεί άλλη τιμή.

ΕΛΕΓΧΟΣ ΕΓΚΥΡΟΤΗΤΑΣ ΜΕ ΟΣΟ

```
ΓΡΑΨΕ 'ΔΩΣΕ ΘΕΤΙΚΗ ΤΙΜΗ'
ΔΙΑΒΑΣΕ ΜΙΣΘ
ΟΣΟ ΜΙΣΘ<=0 ΕΠΑΝΑΛΑΒΕ
    ΓΡΑΨΕ 'ΔΩΣΕ ΘΕΤΙΚΗ ΤΙΜΗ'
    ΔΙΑΒΑΣΕ ΜΙΣΘ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

ΕΛΕΓΧΟΣ ΕΓΚΥΡΟΤΗΤΑΣ ΜΕ ΜΕΧΡΙΣ_ΟΤΟΥ

```
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'ΔΩΣΕ ΘΕΤΙΚΗ ΤΙΜΗ'
    ΔΙΑΒΑΣΕ ΜΙΣΘ
ΜΕΧΡΙΣ_ΟΤΟΥ ΜΙΣΘ > 0
```

ΕΜΦΩΛΕΥΜΕΝΟΙ ΒΡΟΧΟΙ ΕΠΑΝΑΛΗΨΗΣ

Στην περίπτωση που ένας βρόχος βρίσκεται μέσα σε άλλον, τότε έχουμε εμφωλευμένους βρόχους. Για τη σωστή λειτουργία των εμφωλευμένων βρόχων πρέπει να ισχύουν 3 συγκεκριμένοι κανόνες:

- Ο εσωτερικός βρόχος πρέπει να βρίσκεται ολόκληρος μέσα στον εξωτερικό βρόχο.
- Η είσοδος σε βρόχο υποχρεωτικά γίνεται από την αρχή του.
- Δεν πρέπει να χρησιμοποιηθεί η ίδια <μεταβλητή> ως μετρητής δύο ή περισσοτέρων βρόχων που ο ένας βρίσκεται στο εσωτερικό του άλλου.

ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ Α ΛΑ ΡΩΣΙΚΑ

Είναι η μέθοδος με την οποία κάνει τους πολλαπλασιασμούς ένας υπολογιστής. Εδώ θα μας απασχολήσει μόνο η περίπτωση πολλαπλασιασμού δυο ακέραιων θετικών αριθμών.

```
ΑΛΓΟΡΙΘΜΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ_Α_ΛΑ_ΡΩΣΙΚΑ
ΔΙΑΒΑΣΕ Μ1, Μ2
Ρ←0
ΟΣΟ Μ2>0 ΕΠΑΝΑΛΑΒΕ
ΑΝ (Μ2 mod 2 = 1)ΤΟΤΕ
Ρ←Ρ+Μ1
ΤΕΛΟΣ_ΑΝ
Μ1←Μ1*2
Μ2←Μ2 div 2
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ Ρ
ΤΕΛΟΣ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ_Α_ΛΑ_ΡΩΣΙΚΑ
```

ΟΛΙΣΘΗΣΗ

Στα κυκλώματα του υπολογιστή τα δεδομένα αποθηκεύονται με δυαδική μορφή δηλ. με 0 και 1. Για παράδειγμα ο αριθμός 17 ισούται στο δυαδικό σύστημα με τον αριθμό 00010001.

ΟΛΙΣΘΗΣΗ ΠΡΟΣ ΤΑ ΑΡΙΣΤΕΡΑ: μετακινώ τα δυαδικά ψηφία προς τα αριστερά κατά μια θέση. Αυτό γίνεται αν προσθέσω ένα 0 στο τέλος του αριθμού και αγνοήσω το αρχικό 0. Έτσι ο αριθμός 00010001(ο 17 στο δεκαδικό) με ολίσθηση προς τα αριστερά γίνεται 00100010 (34 στο δεκαδικό). Παρατηρώ λοιπόν ότι ο αριθμός διπλασιάστηκε (από 17 έγινε 34), άρα **η ολίσθηση προς τα αριστερά ισοδυναμεί με πολλαπλασιασμό επί 2.**

ΟΛΙΣΘΗΣΗ ΠΡΟΣ ΤΑ ΔΕΞΙΑ: : μετακινώ τα δυαδικά ψηφία προς τα δεξιά κατά μια θέση. Αυτό γίνεται αν προσθέσω ένα 0 στην αρχή του αριθμού και αγνοήσω το τελικό 1. Έτσι ο αριθμός 00010001(ο 17 στο δεκαδικό) με ολίσθηση προς τα δεξιά γίνεται 00001000 (8 στο δεκαδικό). Παρατηρώ λοιπόν ότι ο αριθμός υποδιπλασιάστηκε-αγνοώντας το δεκαδικό μέρος- (από 17 έγινε 8), άρα **η ολίσθηση προς τα δεξιά ισοδυναμεί με ακέραια διαίρεση δια 2.**

ΚΕΦΑΛΑΙΟ 3-ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΛΓΟΡΙΘΜΟΙ
ΚΕΦΑΛΑΙΟ 9-ΠΙΝΑΚΕΣ

ΑΛΓΟΡΙΘΜΟΙ + ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ = ΠΡΟΓΡΑΜΜΑΤΑ

Τα δεδομένα ενός προβλήματος αποθηκεύονται στην κύρια και δευτερεύουσα μνήμη του υπολογιστή, όχι με τυχαίο τρόπο αλλά με την χρήση μιας δομής.

ΔΟΜΗ ΔΕΔΟΜΕΝΩΝ: είναι ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.

Κάθε μορφή δομής δεδομένων αποτελείται από ένα σύνολο κόμβων.

Οι βασικές λειτουργίες επί των δομών δεδομένων είναι οι εξής οκτώ:

- **ΠΡΟΣΠΕΛΑΣΗ**, δηλ. πρόσβαση σε ένα κόμβο με σκοπό να εξετασθεί ή να τροποποιηθεί το περιεχόμενό του.
- **ΕΙΣΑΓΩΓΗ**, δηλ. προσθήκη νέων κόμβων σε μια δομή.
- **ΔΙΑΓΡΑΦΗ**, δηλ αφαίρεση ενός κόμβου από μια δομή.
- **ΑΝΑΖΗΤΗΣΗ**, δηλ. προσπέλαση των κόμβων μιας δομής με σκοπό τον εντοπισμό ενός ή περισσοτέρων κόμβων που έχουν κάποια συγκεκριμένη ιδιότητα.
- **ΤΑΞΙΝΟΜΗΣΗ**, δηλ. οι κόμβοι μιας δομής ταξινομούνται κατά αύξουσα ή φθίνουσα σειρά.
- **ΑΝΤΙΓΡΑΦΗ**, δηλ. κάποιοι ή όλοι οι κόμβοι μιας δομής αντιγράφονται σε μια άλλη δομή.
- **ΣΥΓΧΩΝΕΥΣΗ**, δηλ. δυο ή περισσότερες δομές ενώνονται σε μια ενιαία δομή.
- **ΔΙΑΧΩΡΙΣΜΟΣ**, δηλ. μια δομή διασπάται σε δυο ή περισσότερες διαφορετικές δομές.
-

Στην πράξη σπάνια και οι 8 λειτουργίες χρησιμοποιούνται από μια δομή. Συνήθως μια δομή δεδομένων είναι πιο αποδοτική από κάποια άλλη με βάση κάποια λειτουργία π.χ. μια δομή δεδομένων μπορεί να είναι πιο αποτελεσματική για την λειτουργία της εισαγωγής ενώ κάποια άλλη πιο αποδοτική για τη λειτουργία της αναζήτησης. Γι' αυτόν ακριβώς το λόγο υπάρχουν πολλές διαφορετικές δομές δεδομένων κι είναι σημαντικό να επιλέγουμε την κατάλληλη δομή κάθε φορά ανάλογα με τη λειτουργία που θα εκτελέσουμε.

Συμπερασματικά μπορούμε να πούμε(όπως διατύπωσε και ο Wirth) ότι:

ΑΛΓΟΡΙΘΜΟΙ + ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ = ΠΡΟΓΡΑΜΜΑΤΑ

Δηλ. υπάρχει μεγάλη εξάρτηση μεταξύ του αλγορίθμου και της δομής δεδομένων ώστε το πρόγραμμα να τα θεωρεί ως μια αδιάσπαστη ενότητα.

Οι Δομές Δεδομένων χωρίζονται στις εξής δυο κατηγορίες:

- ΣΤΑΤΙΚΕΣ, τα στοιχεία των οποίων αποθηκεύονται σε συνεχόμενες θέσεις μνήμης και το ακριβές μέγεθος της απαιτούμενης κύριας και δευτερεύουσας μνήμης καθορίζεται κατά τη στιγμή του προγραμματισμού τους δηλ. όταν μεταφράζεται κι όχι όταν εκτελείται το πρόγραμμα.
- ΔΥΝΑΜΙΚΕΣ, τα στοιχεία των οποίων δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης αλλά στηρίζονται στην τεχνική της λεγόμενης *δυναμικής παραχώρησης μνήμης*. Οι δομές αυτές δεν έχουν σταθερό μέγεθος αλλά ο αριθμός των κόμβων τους μεγαλώνει και μικραίνει καθώς στη δομή εισάγονται νέα δεδομένα ή διαγράφονται κάποια άλλα αντίστοιχα.

ΠΙΝΑΚΕΣ

Οι στατικές δομές δεδομένων υλοποιούνται με ΠΙΝΑΚΕΣ, όπως τους γνωρίζουμε από τα Μαθηματικά.
Ως πίνακα ορίζουμε μια δομή που περιέχει στοιχεία του ίδιου τύπου (ακέραιους, πραγματικούς αριθμούς, χαρακτήρες).
Ένας πίνακας μπορεί να είναι μονοδιάστατος, δισδιάστατος, τρισδιάστατος και γενικά ν-διάστατος.
Εμείς θα ασχοληθούμε με μονοδιάστατους και δισδιάστατους πίνακες.

ΜΟΝΟΔΙΑΣΤΑΤΟΣ ΠΙΝΑΚΑΣ

Παρακάτω φαίνεται ένας μονοδιάστατος πίνακας:

ι=1	ι=2	ι=3	ι=4	ι=5	ι=6	ι=7	ι=8	ι=9	ι=10

Παρατηρούμε ότι ο συγκεκριμένος πίνακας έχει 10 θέσεις.
Σε έναν αλγόριθμο ή σε ένα πρόγραμμα για να συμβολίσουμε έναν μονοδιάστατο πίνακα του δίνουμε ένα (επιτρεπτό) όνομα και μέσα σε αγκύλη τον αριθμό των θέσεών του. Έτσι, ο παραπάνω πίνακας θα μπορούσε να ονομαστεί table[10].
Για να διαβάσει τον παραπάνω πίνακα ένας αλγόριθμος ή ένα πρόγραμμα(δηλ. για να τον «γεμίσει» με δεδομένα) χρησιμοποιεί την επαναληπτική δομή ΓΙΑ…ΑΠΟ…ΜΕΧΡΙ. Έτσι έχω:

```
ΓΙΑ ι ΑΠΟ 1 ΜΕΧΡΙ 10
    ΔΙΑΒΑΣΕ table[ι]
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

όπου το ι είναι ο λεγόμενος «δείκτης», δηλ. μας δείχνει σε ποια θέση του πίνακα βρισκόμαστε.
Έστω ότι «γεμίζω» τον πίνακα με τους αριθμούς 10,20,30,40,50,60,70,80,90,100 οπότε γίνεται:

10	20	30	40	50	60	70	80	90	100
ι=1	ι=2	ι=3	ι=4	ι=5	ι=6	ι=7	ι=8	ι=9	ι=10

ΔΙΣΔΙΑΣΤΑΤΟΣ ΠΙΝΑΚΑΣ

Παρακάτω φαίνεται ένας δισδιάστατος πίνακας:

Παρατηρούμε ότι ο πίνακας αποτελείται από 6 γραμμές και 5 στήλες, γι αυτό και λέμε ότι ο πίνακας έχει διαστάσεις 6 χ 5. Στην περίπτωση που ο αριθμός των γραμμών και στηλών ενός πίνακα είναι ίδιος, τότε ο πίνακας λέγεται τετραγωνικός.
Σε έναν αλγόριθμο ή σε ένα πρόγραμμα για να συμβολίσουμε έναν δισδιάστατο πίνακα του δίνουμε ένα (επιτρεπτό) όνομα και μέσα σε αγκύλη τον αριθμό των γραμμών και των στηλών του, χωρισμένους με κόμμα. Έτσι, ο παραπάνω πίνακας θα μπορούσε να ονομαστεί table[6,5].
Για να διαβάσει τον παραπάνω πίνακα ένας αλγόριθμος ή ένα πρόγραμμα(δηλ. για να τον «γεμίσει» με δεδομένα) χρησιμοποιεί την επαναληπτική δομή ΓΙΑ…ΑΠΟ…ΜΕΧΡΙ. Έτσι έχω:

```
ΓΙΑ ι ΑΠΟ 1 ΜΕΧΡΙ 6
    ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 5
        ΔΙΑΒΑΣΕ table[ι, j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

όπου τα ι, j είναι οι λεγόμενοι «δείκτες», δηλ. μας δείχνουν σε ποια θέση (γραμμή και στήλη αντίστοιχα) του πίνακα βρισκόμαστε.
Έστω ότι γεμίζω τον πίνακα με τους παρακάτω αριθμούς:

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25
30	29	28	27	26

οπότε για ι=1 και j=1 έχω table[1,1]=1, για ι=1 και j=2 έχω table[1,2]=2 κ.ο.κ.

Αν στο πρόγραμμα χρησιμοποιήσω πίνακες θα τους δηλώσω κι αυτούς στο τμήμα δήλωσης μεταβλητών ανάλογα με τις τιμές που δέχονται. Π.χ. αν έχω έναν μονοδιάστατο πίνακα Α 30 θέσεων που περιέχει ονόματα μαθητών, έναν μονοδιάστατο πίνακα Β 12 θέσεων που περιέχει ονόματα μαθημάτων κι έναν δισδιάστατο πίνακα Γ 30χ12 που περιέχει τους βαθμούς των 30 μαθητών στα 12 μαθήματα, το αντίστοιχο τμήμα δήλωσης μεταβλητών θα είναι:

ΜΕΤΑΒΛΗΤΕΣ
ΧΑΡΑΚΤΗΡΕΣ : Α[30], Β[12]
ΑΚΕΡΑΙΕΣ : Γ[30,12]

ΠΟΤΕ ΠΡΕΠΕΙ ΝΑ ΧΡΗΣΙΜΟΠΟΙΟΥΝΤΑΙ ΠΙΝΑΚΕΣ

Η χρήση πινάκων είναι ένας εύκολος τρόπος για τη διαχείριση πολλών δεδομένων ίδιου τύπου, αλλά πολλές φορές η χρήση τους είναι περιττή. Η απόφαση για τη χρήση ή μη πίνακα είναι θέμα εμπειρίας στον προγραμματισμό. Γενικά, αν τα δεδομένα που εισάγονται στο πρόγραμμα πρέπει να διατηρούνται στη μνήμη μέχρι το τέλος της εκτέλεσης του προγράμματος τότε η χρήση πινάκων βοηθάει ή είναι απαραίτητη για την επίλυση του προβλήματος.

Δυο μειονεκτήματα από την χρήση πινάκων είναι τα εξής:

1. **οι πίνακες απαιτούν πολλή μνήμη, η οποία δεσμεύεται από την αρχή του προγράμματος.**
2. **οι πίνακες περιορίζουν τις δυνατότητες του προγράμματος, καθώς είναι στατικές δομές και το μέγεθος τους πρέπει να καθορίζεται στην αρχή του προγράμματος και παραμένει σταθερό καθ' όλη τη διάρκεια του προγράμματος.**

ΤΥΠΙΚΕΣ ΕΠΕΞΕΡΓΑΣΙΕΣ ΠΙΝΑΚΩΝ

Είναι οι εξής:

1. υπολογισμός αθροισμάτων στοιχείων του πίνακα
2. εύρεση μέγιστου ή ελάχιστου στοιχείου
3. ταξινόμηση των στοιχείων του (ταξινόμηση ευθείας ανταλλαγής)
4. αναζήτηση στοιχείου του πίνακα (σειριακή και δυαδική)
5. συγχώνευση δυο πινάκων

ΣΕΙΡΙΑΚΗ ΑΝΑΖΗΤΗΣΗ ΠΙΝΑΚΑ

Όταν χρησιμοποιούμε πίνακες σε κάποιο αλγόριθμο μπορεί να μας ζητηθεί να εξετάσουμε αν κάποιο από τα στοιχεία του έχει μια συγκεκριμένη τιμή. Αν αυτό συμβαίνει, θέλουμε να εντοπίσουμε τη θέση του στοιχείου αυτού. Γι αυτό χρησιμοποιούμε αλγόριθμους αναζήτησης, ένας από τους οποίους είναι αυτός της **σειριακής αναζήτησης** (ο μόνος που θα εξετάσουμε). Σημειώνεται ότι η αναζήτηση γίνεται είτε σε μονοδιάστατο πίνακα είτε σε συγκεκριμένη γραμμή ή στήλη δισδιάστατου πίνακα. Ο αλγόριθμος αυτός χρησιμοποιείται όταν:

Α) ο πίνακας είναι μη ταξινομημένος.
Β) ο πίνακας έχει μικρό μέγεθος (λιγότερο από 20 στοιχεία).
Γ) η αναζήτηση στοιχείων του συγκεκριμένου πίνακα γίνεται σπάνια.

Εξετάζουμε ένα προς ένα όλα τα στοιχεία του πίνακα ξεκινώντας από αυτό που βρίσκεται στην πρώτη θέση μέχρι να βρεθεί στοιχείο με τιμή ίση με αυτή που ζητάμε. Όταν το στοιχείο βρεθεί η αναζήτηση σταματάει και η μεταβλητή DONE (τύπου λογική) γίνεται ΑΛΗΘΗΣ. Σε περίπτωση που η τιμή που αναζητούμε δεν υπάρχει η μεταβλητή DONE θα είναι ΨΕΥΔΗΣ, οπότε συμπεραίνουμε ότι η τιμή που

θέλουμε δεν υπάρχει στον πίνακα. Παρακάτω δίνεται ο αντίστοιχος αλγόριθμος αναζήτησης σε έναν πίνακα TABLE με Ν θέσεις, κι έστω ότι η ζητούμενη τιμή είναι το ΚΕΥ, κι η θέση που πιθανόν θα βρεθεί το ζητούμενο στοιχείο είναι η POSITION:

```
ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗ
ΔΙΑΒΑΣΕ Ν
ΓΙΑ Ι ΑΠΟ 1 ΜΕΧΡΙ Ν
        ΔΙΑΒΑΣΕ TABLE[I]
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΔΙΑΒΑΣΕ ΚΕΥ
DONE← ΨΕΥΔΗΣ
POSITION←0
I←1
ΟΣΟ (DONE=ΨΕΥΔΗΣ) ΚΑΙ (I<=N) ΕΠΑΝΑΛΑΒΕ
        ΑΝ TABLE[I]=ΚΕΥ ΤΟΤΕ
                POSITION←I
                DONE← ΑΛΗΘΗΣ
        ΑΛΛΙΩΣ
                I←I+1
        ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΑΝ DONE=ΑΛΗΘΗΣ ΤΟΤΕ
        ΓΡΑΨΕ POSITION
ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΔΕΝ ΒΡΕΘΗΚΕ'
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ ΑΝΑΖΗΤΗΣΗ
```

ΤΑΞΙΝΟΜΗΣΗ

Ταξινόμηση είναι η τακτοποίηση των κόμβων μιας δομής με μια συγκεκριμένη σειρά. Συνήθως η σειρά αυτή είναι η αύξουσα τάξη της τιμής των στοιχείων προς ταξινόμηση.

Σκοπός της ταξινόμησης είναι να διευκολυνθεί στη συνέχεια η αναζήτηση ενός στοιχείου στον πίνακα.

ΟΡΙΣΜΟΣ ΤΑΞΙΝΟΜΗΣΗΣ

Δοθέντων των στοιχείων $\alpha_1, \alpha_2, \ldots \alpha_v$, η ταξινόμηση συνίσταται στη μετάθεση της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μια σειρά $\alpha_{κ1}, \alpha_{κ2}, \ldots \alpha_{κv}$ έτσι ώστε, δοθείσης μιας συνάρτησης διάταξης f να ισχύει:

$$f(\alpha_{κ1}) \leq f(\alpha_{κ2}) \leq \ldots \leq f(\alpha_{κv})$$

ΤΑΞΙΝΟΜΗΣΗ ΕΥΘΕΙΑΣ ΑΝΤΑΛΛΑΓΗΣ (ΤΑΞΙΝΟΜΗΣΗ ΦΥΣΣΑΛΙΔΑΣ) ΣΕ ΕΝΑΝ ΜΟΝΟΔΙΑΣΤΑΤΟ ΠΙΝΑΚΑ Α n ΘΕΣΕΩΝ

Υπάρχουν πολλοί αλγόριθμοι ταξινόμησης εμάς όμως θα μας απασχολήσει μόνο ο αλγόριθμος ταξινόμησης *ευθείας ανταλλαγής* (είναι ο πιο απλός αλλά κι ο πιο αργός αλγόριθμος ταξινόμησης). Ο αλγόριθμος αυτός βασίζεται στην αρχή της σύγκρισης και ανταλλαγής ζευγών γειτονικών στοιχείων, μέχρι να διαταχθούν όλα τα στοιχεία. Κάθε φορά γίνονται διαδοχικές προσπελάσεις στον πίνακα και το μικρότερο στοιχείο μετακινείται προς τα αριστερά (αν πρόκειται για πίνακα-γραμμή) ή μετακινείται προς τα πάνω (αν πρόκειται για πίνακα-στήλη). Στην περίπτωση αυτή (πίνακα-

στήλη) μπορούμε να φανταστούμε τα στοιχεία του πίνακα σαν «φυσσαλίδες» όπου η πιο ελαφριά ανεβαίνει πιο ψηλά για αυτό κι ο αλγόριθμος ταξινόμησης ευθείας ανταλλαγής λέγεται και ***ταξινόμηση φυσσαλίδας(bubblesort).***

Παρακάτω δίνεται ο αλγόριθμος ταξινόμησης ενός μονοδιάστατου πίνακα Α με n θέσεις(η ταξινόμηση είναι αύξουσα):

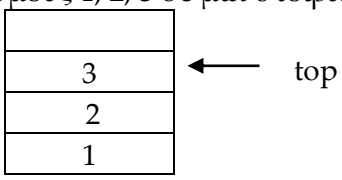
```
ΑΛΓΟΡΙΘΜΟΣ ΦΥΣΣΑΛΙΔΑ
ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ n
        ΓΙΑ j ΑΠΟ n ΜΕΧΡΙ i ΜΕ_ΒΗΜΑ -1
                ΑΝ A[j-1]>A[J] ΤΟΤΕ
                        temp←A[j-1]
                        A[j-1]←A[j]
                        A[j]←temp
                ΤΕΛΟΣ_ΑΝ
        ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ n
        ΕΜΦΑΝΙΣΕ Α[i]
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ ΦΥΣΣΑΛΙΔΑ
```

ΣΤΟΙΒΑ-ΟΥΡΑ

Δυο άλλες σημαντικές δομές είναι η στοίβα και η ουρά

ΣΤΟΙΒΑ (stack)

Μια στοίβα δεδομένων είναι σαν μια στοίβα πιάτων, όπου το ένα τοποθετείται πάνω στο άλλο. Για παράδειγμα αν θέλω να εισάγω τους αριθμούς 1, 2, 3 σε μια στοίβα θα γίνει ως εξής:



ενώ για να κάνω εξαγωγή θα πρέπει να βγει πρώτο το στοιχείο που βρίσκεται στην κορυφή(top) της στοίβας , στο παράδειγμά μας ο αριθμός 3.

Αυτή η μέθοδος επεξεργασίας ονομάζεται *Τελευταίο μέσα, πρώτο έξω,* ή πιο απλά **LIFO** (Last-In-First-Out).

Δυο είναι οι βασικές λειτουργίες μιας στοίβας:

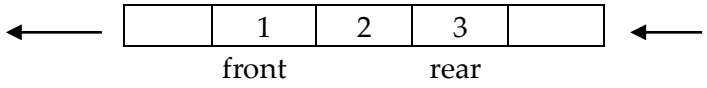
1. **ώθηση (push) στοιχείου στην κορυφή της στοίβας.**
2. **απώθηση (pop) στοιχείου από την στοίβα.**

Στην ώθηση πρέπει να ελέγχεται αν η στοίβα είναι γεμάτη για να μη συμβεί **υπερχείλιση (overflow)** ενώ στην απώθηση να ελέγχεται αν υπάρχει τουλάχιστον ένα στοιχείο στην στοίβα για να μη συμβεί **υποχείλιση (underflow)** της στοίβας.

Η στοίβα μπορεί να υλοποιηθεί με τη βοήθεια ενός μονοδιάστατου πίνακα, χρησιμοποιώντας μια βοηθητική μεταβλητή με όνομα **top** η οποία δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας. Αν θέλω να εισάγω ένα νέο στοιχείο στη στοίβα η μεταβλητή top αυξάνεται κατά ένα ενώ αν θέλω να εξάγω ένα στοιχείο, εξάγεται το στοιχείο και μετά η μεταβλητή top μειώνεται κατά ένα για να δείξει τη νέα κορυφή.

ΟΥΡΑ (queue)

Η ουρά μπορεί να παρομοιαστεί με την ουρά στο ταμείο μιας τράπεζας. Έτσι, το στοιχείο που εισέρχεται πρώτο εξέρχεται και πρώτο. Για παράδειγμα αν θέλω να εισάγω τους αριθμούς 1, 2, 3 σε μια στοίβα θα γίνει ως εξής:



Αυτή η μέθοδος επεξεργασίας λέγεται *Πρώτο μέσα, πρώτο έξω,* ή πιο απλά **FIFO** (First-In-First-Out).

Δυο είναι οι βασικές λειτουργίες μιας ουράς:

1. **εισαγωγή στοιχείου στο πίσω άκρο της ουράς.**
2. **εξαγωγή στοιχείου από το εμπρός άκρο της ουράς.**

Στην εισαγωγή πρέπει να ελέγχεται αν η ουρά είναι γεμάτη ενώ στην εξαγωγή να ελέγχεται αν υπάρχει τουλάχιστον ένα στοιχείο στην ουρά .

Η ουρά μπορεί να υλοποιηθεί με τη βοήθεια ενός μονοδιάστατου πίνακα χρησιμοποιώντας δυο δείκτες: τον δείκτη **εμπρός (front),** ο οποίος μας δείχνει τη θέση του στοιχείου που θα εξαχθεί πρώτο και τον δείκτη **πίσω (rear),** ο οποίος μας δείχνει τη θέση του στοιχείου που μόλις εισήλθε.

Για την εισαγωγή ενός στοιχείου στην ουρά, ο δείκτης rear αυξάνεται κατά ένα και στη θέση αυτή τοποθετείται το στοιχείο ενώ για την εξαγωγή στοιχείου από την ουρά, εξέρχεται το στοιχείο που δείχνει ο δείκτης front, ο οποίος στη συνέχεια αυξάνεται κατά ένα για να δείχνει το επόμενο στοιχείο που θα εξαχθεί.

ΚΕΦΑΛΑΙΟ 4-ΤΕΧΝΙΚΕΣ ΣΧΕΔΙΑΣΗΣ ΑΛΓΟΡΙΘΜΩΝ

ΑΝΑΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ

Η ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον περιλαμβάνει:

- την καταγραφή της υπάρχουσας πληροφορίας για το πρόβλημα
- την αναγνώριση των ιδιαιτεροτήτων του προβλήματος
- την αποτύπωση των συνθηκών και προϋποθέσεων υλοποίησής του

και στη συνέχεια:

- την πρόταση επίλυσης με χρήση κάποιας μεθόδου
- την τελική επίλυση με χρήση υπολογιστικών συστημάτων.

Έτσι, κατά την ανάλυση ενός προβλήματος θα πρέπει να δοθεί απάντηση σε καθεμία από τις επόμενες ερωτήσεις:

1. Ποια είναι τα δεδομένα και το μέγεθος του προβλήματος
2. Ποιες είναι οι συνθήκες που πρέπει να πληρούνται για την επίλυσητου προβλήματος
3. Ποια είναι η πλέον αποδοτική μέθοδος επίλυσής τους (σχεδίαση αλγορίθμου)
4. Πώς θα καταγραφεί η λύση σε ένα πρόβλημα (π.χ. σε ψευδογλώσσα)
5. Ποιος είναι ο τρόπος υλοποίησης στο συγκεκριμένο υπολογιστικό σύστημα (π.χ. επιλογή γλώσσας προγραμματισμού).

ΚΕΦΑΛΑΙΟ 6-ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

ΤΙ ΟΝΟΜΑΖΟΥΜΕ ΦΥΣΙΚΕΣ ΚΑΙ ΤΙ ΤΕΧΝΗΤΕΣ ΓΛΩΣΣΕΣ;

Φυσικές είναι οι γλώσσες που χρησιμοποιούνται για την επικοινωνία μεταξύ των ανθρώπων ενώ τεχνητές αυτές που χρησιμοποιούνται για την επικοινωνία ανθρώπου με υπολογιστή.

ΠΟΙΑ ΤΑ ΚΟΙΝΑ ΚΑΙ ΠΟΙΕΣ ΟΙ ΔΙΑΦΟΡΕΣ ΜΕΤΑΞΥ ΦΥΣΙΚΩΝ ΚΑΙ ΤΕΧΝΗΤΩΝ ΓΛΩΣΣΩΝ;

Τα κοινά είναι τα ακόλουθα:

- Αλφάβητο
- Λεξιλόγιο
- Γραμματική (τυπικό και συντακτικό)
- Σημασιολογία

Η βασική διαφορά τους είναι ότι οι φυσικές γλώσσες εμπλουτίζονται συνεχώς ενώ οι τεχνητές παραμένουν στάσιμες.

ΤΙ ΕΙΝΑΙ Η ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ ΕΝΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ;

Πρόκειται για τη συνεχή διαίρεση του προβλήματος σε υποπροβλήματα, τα οποία είναι ευκολότερα να επιλυθούν, οδηγώντας έτσι στην επίλυση του αρχικού προβλήματος.

ΤΙ ΕΙΝΑΙ Ο ΤΜΗΜΑΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ;

Η ιεραρχική σχεδίαση ενός προγράμματος υλοποιείται με τον τμηματικό προγραμματισμό, δηλ. τη διαίρεση ενός προβλήματος σε υποπροβλήματα, καθένα από τα οποία αποτελεί μια ξεχωριστή ενότητα. Ο τμηματικός προγραμματισμός διευκολύνει τη δημιουργία του προβλήματος, μειώνει τα λάθη και κάνει ευκολότερη την παρακολούθηση , κατανόηση και συντήρηση του προγράμματος.

ΤΙ ΕΙΝΑΙ Ο ΔΟΜΗΜΕΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ;

Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο δομών, τη δομή ακολουθίας, τη δομή επιλογής και τη δομή επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα έχει μόνο μια είσοδο και μόνο μια έξοδο. Ο δομημένος προγραμματισμός αποτελεί σήμερα την βασική μεθοδολογία προγραμματισμού.

ΠΟΙΑ ΤΑ ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΟΥ ΔΟΜΗΜΕΝΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ;

Είναι συνοπτικά τα παρακάτω:

- Δημιουργία απλούστερων προγραμμάτων
- Άμεση μεταφορά των αλγορίθμων σε προγράμματα
- Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα
- Περιορισμός των λαθών
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους
- Ευκολότερη διόρθωση και συντήρηση

Η ΕΝΤΟΛΗ GOTO (ΠΗΓΑΙΝΕ)

Η εντολή αυτή έχει ως αποτέλεσμα την αλλαγή της ροής του προγράμματος, δηλ. με την εντολή αυτή εκτελείται μια άλλη εντολή του προγράμματος εκτός από την επόμενη. Η χρήση αυτής της εντολής χώρισε τους προγραμματιστές σε 2 στρατόπεδα: στους φανατικούς υποστηρικτές της, οι οποίοι έλυναν εύκολα τα προβλήματα με την χρήση της και στους πολέμιούς της, οι οποίοι έβλεπαν οτι εντολή αυτή ήταν υπεύθυνη για τη δυσκολία της επίλυσης, παρακολούθησης, της κατανόησης και συντήρησης του προγράμματος.

Στις μέρες μας όλες οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν τον δομημένο προγραμματισμό και διαθέτουν εντολές που καθιστούν την χρήση της ΠΗΓΑΙΝΕ (GOTO) περιττή. (ο δομημένος προγραμματισμός αναπτύχθηκε από την ανάγκη να υπάρχει μια κοινή μεθοδολογία στην ανάπτυξη προγραμμάτων και τη μείωση των εντολών GOTO που χρησιμοποιούνται στο πρόγραμμα).

Παρακάτω δίνεται ένα τμήμα αλγορίθμου με την χρήση της ΠΗΓΑΙΝΕ και για λόγους σύγκρισης δίνεται το ίδιο παράδειγμα με χρήση του δομημένου προγραμματισμού (συγκεκριμένα, της δομής επιλογής) :

ΜΕ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ:

```
ΑΝ Χ>0 ΤΟΤΕ
        ΓΡΑΨΕ 'ΘΕΤΙΚΟΣ'
ΑΛΛΙΩΣ_ΑΝ Χ=0 ΤΟΤΕ
        ΓΡΑΨΕ 'ΜΗΔΕΝ'
ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΑΡΝΗΤΙΚΟΣ'
ΤΕΛΟΣ_ΑΝ
```

ΜΕ ΧΡΗΣΗ ΤΗΣ ΠΗΓΑΙΝΕ(GOTO)

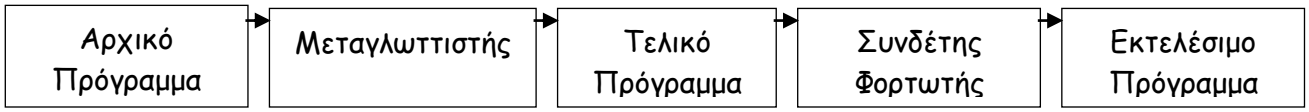
```
ΑΝ Χ>0 ΤΟΤΕ ΠΗΓΑΙΝΕ 1
ΑΝ Χ=0 ΤΟΤΕ ΠΗΓΑΙΝΕ 2
        ΓΡΑΨΕ 'ΑΡΝΗΤΙΚΟΣ'
ΠΗΓΑΙΝΕ 4
1: ΓΡΑΨΕ 'ΘΕΤΙΚΟΣ'
ΠΗΓΑΙΝΕ 4
2: ΓΡΑΨΕ 'ΜΗΔΕΝ'
ΠΗΓΑΙΝΕ 4
4: !ΣΥΝΕΧΕΙΑ,
```

ΤΙ ΕΙΝΑΙ ΟΙ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ ΚΑΙ ΟΙ ΔΙΕΡΜΗΝΕΥΤΕΣ;

Είναι ειδικά μεταφραστικά προγράμματα που μετατρέπουν το πρόγραμμα που γράφουμε σε υψηλή γλώσσα προγραμματισμού σε γλώσσα μηχανής, δηλ. σε μορφή αναγνωρίσιμη κι εκτελέσιμη από τον υπολογιστή.

ΠΩΣ ΛΕΙΤΟΥΡΓΟΥΝ ΟΙ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ;

Οι μεταγλωττιστές δέχονται στην είσοδο το πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου και παράγουν ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής. Το αρχικό πρόγραμμα λέγεται πηγαίο ενώ το πρόγραμμα που παράγεται από τους μεταγλωττιστές λέγεται αντικείμενο πρόγραμμα. Το αντικείμενο πρόγραμμα είναι κατανοητό από τον υπολογιστή αλλά για να εκτελεστεί πρέπει να συνδεθεί με άλλα τμήματα προγράμματος απαραίτητα για την εκτέλεσή του. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση λέγεται συνδέτης-φορτωτής, και παράγει το εκτελέσιμο πρόγραμμα. Για να παραχθεί το εκτελέσιμο πρόγραμμα πρέπει να μην υπάρχουν λάθη στο αρχικό πρόγραμμα. Τα λάθη μπορεί να είναι είτε λογικά π.χ. σφάλμα στην υλοποίηση αλγορίθμου είτε συντακτικά π.χ. παράλειψη δήλωσης δεδομένων. Τα λογικά λάθη εμφανίζονται μόνο στην εκτέλεση ενώ τα συντακτικά στο στάδιο της μεταγλώττισης. Η λειτουργία των μεταγλωττιστών φαίνεται σχηματικά παρακάτω:



ΠΩΣ ΛΕΙΤΟΥΡΓΟΥΝ ΟΙ ΔΙΕΡΜΗΝΕΥΤΕΣ;

Οι διερμηνευτές, σε αντίθεση με τους μεταγλωττιστές, διαβάζουν μια προς μια τις εντολές του αρχικού προγράμματος και για κάθε μια εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής.

ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΚΑΙ ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΩΝ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ ΚΑΙ ΤΩΝ ΔΙΕΡΜΗΝΕΥΤΩΝ

ΠΛΕΟΝΕΚΤΗΜΑΤΑ

- Μεταγλωττιστές: γρήγορη εκτέλεση προγράμματος.
- Διερμηνευτές: άμεση εκτέλεση και διόρθωση λαθών.

ΜΕΙΟΝΕΚΤΗΜΑΤΑ

- Μεταγλωττιστές: πριν χρησιμοποιηθεί ένα πρόγραμμα πρέπει να περάσει από το στάδιο μεταγλώττισης και σύνδεσης.
- Διερμηνευτές: αργή εκτέλεση προγράμματος.

Σήμερα τα προγραμματιστικά περιβάλλοντα παρουσιάζουν μεικτές υλοποιήσεις χρησιμοποιώντας διερμηνευτές κατά τη φάση δημιουργίας του προγράμματος και μεταγλωττιστές για την τελική έκδοση.

ΤΙ ΕΙΝΑΙ Ο ΣΥΝΤΑΚΤΗΣ (EDITOR);

Είναι ουσιαστικά ένας μικρός επεξεργαστής κειμένου που χρησιμοποιείται για την αρχική σύνταξη και στη συνέχεια την διόρθωση των προγραμμάτων.

ΚΕΦΑΛΑΙΟ 10-ΥΠΟΠΡΟΓΡΑΜΜΑΤΑ

Τα **υποπρογράμματα** αποτελούν τμήματα (ενότητες) αυτόνομων προγραμμάτων και υλοποιούν το λεγόμενο τμηματικό προγραμματισμό.

Τμηματικός προγραμματισμός ονομάζεται μία τεχνική σχεδίασης και ανάπτυξης προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων.

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ (ΙΔΙΟΤΗΤΕΣ) ΤΩΝ ΥΠΟΠΡΟΓΡΑΜΜΑΤΩΝ:

1. **Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο**. Δηλαδή, ενεργοποιείται με την είσοδο σε αυτό που γίνεται πάντα από την αρχή του, εκτελεί κάποιες ενέργειες και απενεργοποιείται με την έξοδο που γίνεται πάντα από το τέλος του.

2. **Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα.** Αυτό σημαίνει ότι μπορεί να σχεδιαστεί, αναπτυχθεί και ελεγχθεί αυτόνομα σε σχέση με τα άλλα υποπρογράμματα. Έτσι, διευκολύνεται η ταχεία ανάπτυξη ενός μεγάλου προγράμματος το οποίο χρησιμοποιεί τα υποπρογράμματα.
   Πάντως στην πράξη, η *απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί*.

3. **Κάθε υποπρόγραμμα πρέπει να εκτελεί μία μόνο λειτουργία και να μην είναι πολύ μεγάλο**. Έτσι, διευκολύνεται η κατανόηση και ο έλεγχός του. Αν εκτελεί περισσότερες λειτουργίες τότε μάλλον πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΟΥ ΤΜΗΜΑΤΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ :

- **Διευκολύνει την ανάπτυξη του αλγορίθμου και το αντίστοιχου προγράμματος**. Αντί να προσπαθούμε να επιλύσουμε μεμιάς το συνολικό πρόβλημα είναι προτιμότερο να το σπάσουμε σε απλούστερα υπο-προβλήματα για τα οποία θα γράψουμε αντίστοιχους αλγόριθμους και συνεπώς τμήματα προγραμμάτων και κατόπιν να προβούμε στη σύνθεσή τους η οποία λύνει και το συνολικό πρόβλημα.

- **Διευκολύνει την κατανόηση και διόρθωση του προγράμματος**. Είναι πολύ ευκολότερο από κάποιον να κατανοήσει, ελέγξει και τροποποιήσει το μικρότερο υποπρόγραμμα

- **Απαιτεί λιγότερο χρόνο και κόπο στη συγγραφή του προγράμματος**. Αν η ίδια λειτουργία χρειάζεται σε διαφορετικά σημεία του συνολικού προγράμματος τότε είναι προτιμότερο να γραφτεί ένα υποπρόγραμμα που την υλοποιεί. Κατόπιν, θα μπορούμε να *καλούμε* το υποπρόγραμμα στο αντίστοιχο σημείο. Έτσι, διευκολύνεται η ταχεία ανάπτυξη του συνολικού προγράμματος μειώνοντας το μέγεθός του και τις πιθανότητες λαθών.

- **Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού**. Αν μία λειτουργία δεν υποστηρίζεται απευθείας από τη γλώσσα τότε φτιάχνουμε ένα υποπρόγραμμα που την υλοποιεί. Το υποπρόγραμμα τότε μπορεί να είναι διαθέσιμο σε όσα προγράμματα απαιτούν την λειτουργία. Π.χ. ένα υποπρόγραμμα που υπολογίζει το εμβαδόν ενός τριγώνου. Όσα προγράμματα χρειάζονται να υπολογίζουν εμβαδά τριγώνων μπορούν να καλούν αυτό το υποπρόγραμμα.
  Συγγράφοντας πολλά υποπρογράμματα μπορούμε να δημιουργήσουμε ολόκληρες *βιβλιοθήκες (Libraries)* και ουσιαστικά να επεκτείνουμε την ίδια τη γλώσσα προγραμματισμού για λειτουργίες που δεν τις υποστηρίζει απευθείας.

ΠΑΡΑΜΕΤΡΟΙ

Τα υποπρογράμματα ενεργοποιούνται από κάποιο άλλο πρόγραμμα (π.χ. το *κύριο πρόγραμμα*) ή υποπρόγραμμα για να εκτελέσουν τη λειτουργία τους.

Η επικοινωνία μεταξύ του υποπρογράμματος και αυτού που το καλεί γίνεται διαμέσου κάποιων **ειδικών μεταβλητών** που ονομάζονται **παράμετροι.** Το καλούν πρόγραμμα «περνάει» (μεταβιβάζει) τιμές στο υποπρόγραμμα μέσω των παραμέτρων του. Ενδεχομένως, να επιστρέφει και τιμές στο καλούν μέσω αυτών.

Παράμετρος είναι μία μεταβλητή που επιτρέπει το πέρασμα τιμής από ένα τμήμα προγράμματος σε ένα άλλο.

ΕΙΔΗ ΥΠΟΠΡΟΓΡΑΜΜΑΤΩΝ

Διαδικασίες. Μία διαδικασία είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελέσει οποιαδήποτε λειτουργία. Μπορεί να επιστρέψει μηδέν, μία ή περισσότερες τιμές ως αποτελέσματα μέσω των παραμέτρων της.

Συναρτήσεις. Μία συνάρτηση είναι ένας τύπος υποπρογράμματος που **υπολογίζει μία μόνο τιμή** και την επιστρέφει στο καλούν μέσω του ονόματός της.

ΟΡΙΣΜΟΣ ΚΑΙ ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ

ΣΥΝΑΡΤΗΣΗ όνομα (λίστα παραμέτρων) : τύπος συνάρτησης
Τμήμα δηλώσεων μεταβλητών
Αρχή
    ….
    όνομα ← έκφραση
    …

Τέλος_συνάρτησης

ΟΡΙΣΜΟΣ ΚΑΙ ΚΛΗΣΗ ΔΙΑΔΙΚΑΣΙΩΝ

ΔΙΑΔΙΚΑΣΙΑ όνομα (λίστα παραμέτρων)
Τμήμα δηλώσεων μεταβλητών
Αρχή
    ….
    εντολές
    ….

Τέλος_διαδικασίας

Σημείωση : Οι λίστες παραμέτρων στα υποπρογράμματα δεν είναι υποχρεωτικές, δηλαδή να μην υπάρχουν παράμετροι για μεταβίβαση τιμών ή επιστροφή αποτελεσμάτων.

ΠΑΡΑΔΕΙΓΜΑ ΟΡΙΣΜΟΥ ΣΥΝΑΡΤΗΣΗΣ ΚΑΙ ΠΩΣ ΚΑΛΕΙΤΑΙ

Να γραφτεί μία συνάρτηση που υπολογίζει το εμβαδόν ενός τριγώνου δοθέντος της βάσης και του ύψους.
Επειδή η λειτουργία που θα εκτελεί είναι ένας υπολογισμός και συνεπώς θα επιστρέφει μία τιμή είναι καταλληλότερο να χρησιμοποιηθεί συνάρτηση. Η τιμή που επιστρέφει είναι ένας πραγματικός αριθμός.

ΣΥΝΑΡΤΗΣΗ Εμβαδόν_τριγώνου (βάση, ύψος) : Πραγματική
Μεταβλητές
    Ακέραιες: βάση, ύψος ! οι ειδικές μεταβλητές για τις παραμέτρους.
Αρχή

    Εμβαδόν_τριγώνου ← (βάση * ύψος) / 2

Τέλος_συνάρτησης

Πώς θα κληθεί από ένα κύριο πρόγραμμα για να υπολογίζει διάφορα εμβαδά τριγώνων:

Πρόγραμμα Υπολογισμός_εμβαδών_διαφόρων_τριγώνων
Μεταβλητές
    Ακέραιες: Β,Υ
    Πραγματικές: Ε
Αρχή
    ! Τα δεδομένα για το 1ο τρίγωνο
    Β ←12
    Υ ← 4
    Ε ← Εμβαδόν_τριγώνου (Β,Υ)
    Τύπωσε "Το εμβαδόν είναι ", Ε

    ! Τα δεδομένα για το 2ο τρίγωνο
    Β ← 18
    Υ ← 6
    Ε ← Εμβαδόν_τριγώνου (Β,Υ)
    Τύπωσε "Το εμβαδόν είναι ", Ε

Τέλος_προγράμματος

Εδώ καλείται ! Περνά τις τιμές των *Β, Υ* του κύριου προγράμματος στις αντίστοιχες παραμέτρους *βάση, ύψος* του υποπρογράμματος. Κατόπιν, η συνάρτηση υπολογίζει το εμβαδόν και επιστρέφει την τιμή μέσω του ονόματός της. Αυτή, εκχωρείται στην Ε

Κι εδώ καλείται όπου περνά άλλες τιμές.

Σημείωση:
Η συνάρτηση καλείται μέσω του ονόματός της

ΠΑΡΑΔΕΙΓΜΑ ΟΡΙΣΜΟΥ ΔΙΑΔΙΚΑΣΙΑΣ ΚΑΙ ΠΩΣ ΚΑΛΕΙΤΑΙ

Θα χρησιμοποιήσουμε το ίδιο παράδειγμα για να δείξουμε τη διαφορά στην υλοποίηση με μία διαδικασία.
Θα φτιάξουμε μία διαδικασία που υπολογίζει το εμβαδόν ενός τριγώνου δοθέντος της βάσης και του ύψους. Εδώ, θα έχουμε δύο παραμέτρους για το πέρασμα τιμών (βάση, ύψος) και μία παράμετρο για την επιστροφή του αποτελέσματος (Εμβαδόν) :

```
ΔΙΑΔΙΚΑΣΙΑ Εμβαδόν_τριγώνου (βάση, ύψος, Εμβαδόν)
Μεταβλητές
        Ακέραιες:  βάση, ύψος        ! οι ειδικές μεταβλητές για τις παραμέτρους – πέρασμα τιμών.
        Πραγματικές:  Εμβαδόν        ! παράμετρος για την επιστροφή του αποτελέσματος
Αρχή                                 ! Η επιστροφή αποτελέσματος σε μία διαδικασία γίνεται μέσω
                                     ! παραμέτρου κι όχι του ονόματος όπως συμβαίνει σε !συνάρτηση.
        Εμβαδόν ← (βάση * ύψος) / 2

Τέλος_διαδικασίας
```

Πώς θα κληθεί από ένα κύριο πρόγραμμα για να υπολογίζει διάφορα εμβαδά τριγώνων:

```
Πρόγραμμα Υπολογισμός_εμβαδών_διαφόρων_τριγώνων
Μεταβλητές
        Ακέραιες:  Β,Υ
        Πραγματικές:  Ε
Αρχή
        ! Τα δεδομένα για το 1ο τρίγωνο
        Β ←12
        Υ ← 4
        Κάλεσε Εμβαδόν_τριγώνου (Β,Υ,Ε)
        Τύπωσε "Το εμβαδόν είναι ", Ε

        ! Τα δεδομένα για το 2ο τρίγωνο
        Β ← 18
        Υ ← 6
        Κάλεσε Εμβαδόν_τριγώνου (Β,Υ,Ε)
        Τύπωσε "Το εμβαδόν είναι ", Ε

Τέλος_προγράμματος
```

Εδώ καλείται ! Περνά τις τιμές των *Β, Υ* του κύριου προγράμματος στις αντίστοιχες παραμέτρους *βάση, ύψος* του υποπρογράμματος. Κατόπιν, η διαδικασία υπολογίζει το εμβαδόν και επιστρέφει την τιμή μέσω της παραμέτρου *Ε*.

Κι εδώ καλείται όπου περνά άλλες τιμές.

Σημείωση:
Η διαδικασία καλείται με την **εντολή Κάλεσε**

ΠΡΑΓΜΑΤΙΚΕΣ ΚΑΙ ΤΥΠΙΚΕΣ ΠΑΡΑΜΕΤΡΟΙ

Πραγματικές παράμετροι είναι οι μεταβλητές που χρησιμοποιεί το κυρίως πρόγραμμα και συνδέονται με το υποπρόγραμμα.

Τυπικές παράμετροι είναι οι μεταβλητές που χρησιμοποιεί το υποπρόγραμμα και συνδέονται με το κυρίως πρόγραμμα.

ΠΑΡΑΔΕΙΓΜΑ
Έστω το παρακάτω πρόγραμμα και υποπρόγραμμα:

Πρόγραμμα Υπολογισμός_εμβαδών_διαφόρων_τριγώνων
Μεταβλητές
    Ακέραιες: Β,Υ
    Πραγματικές: Ε
Αρχή
    ! Τα δεδομένα για το 1ο τρίγωνο
    Β ←12
    Υ ← 4
    Ε ← Εμβαδόν_τριγώνου (Β,Υ)
    Τύπωσε "Το εμβαδόν είναι ", Ε

    ! Τα δεδομένα για το 2ο τρίγωνο
    Β ← 18
    Υ ← 6
    Ε ← Εμβαδόν_τριγώνου (Β,Υ)
    Τύπωσε "Το εμβαδόν είναι ", Ε
Τέλος_προγράμματος

ΣΥΝΑΡΤΗΣΗ Εμβαδόν_τριγώνου (βάση, ύψος) : Πραγματική
Μεταβλητές
    Ακέραιες: βάση, ύψος ! οι ειδικές μεταβλητές για τις παραμέτρους.
Αρχή

    Εμβαδόν_τριγώνου ← (βάση * ύψος) / 2
Τέλος_συνάρτησης

Στο παραπάνω παράδειγμα πραγματικές παράμετροι είναι οι μεταβλητές Β, Υ ενώ τυπικές παράμετροι είναι οι μεταβλητές βάση, ύψος.
Μερικές γλώσσες προγραμματισμού ονομάζουν **ορίσματα** τις τυπικές παραμέτρους και απλά **παραμέτρους** τις πραγματικές παραμέτρους.

Η **λίστα των τυπικών παραμέτρων** καθορίζει τις παραμέτρους **στη δήλωση του υποπρογράμματος** και η **λίστα των πραγματικών παραμέτρων** καθορίζει τις παραμέτρους **στην κλήση του υποπρογράμματος.**

Γενικά, όλες οι μεταβλητές είναι γνωστές ή όπως συνηθίζεται να λέμε έχουν ισχύ μόνο για το τμήμα προγράμματος στο οποίο έχουν δηλωθεί, δηλαδή ισχύουν **τοπικά** για το συγκεκριμένο υποπρόγραμμα ή κυρίως πρόγραμμα.

ΟΙ ΛΙΣΤΕΣ ΤΩΝ ΠΑΡΑΜΕΤΡΩΝ ΠΡΕΠΕΙ ΝΑ ΑΚΟΛΟΥΘΟΥΝ ΤΟΥΣ ΕΞΗΣ ΚΑΝΟΝΕΣ:

1.Ο αριθμός των τυπικών και πραγματικών παραμέτρων πρέπει να είναι ίδιος.

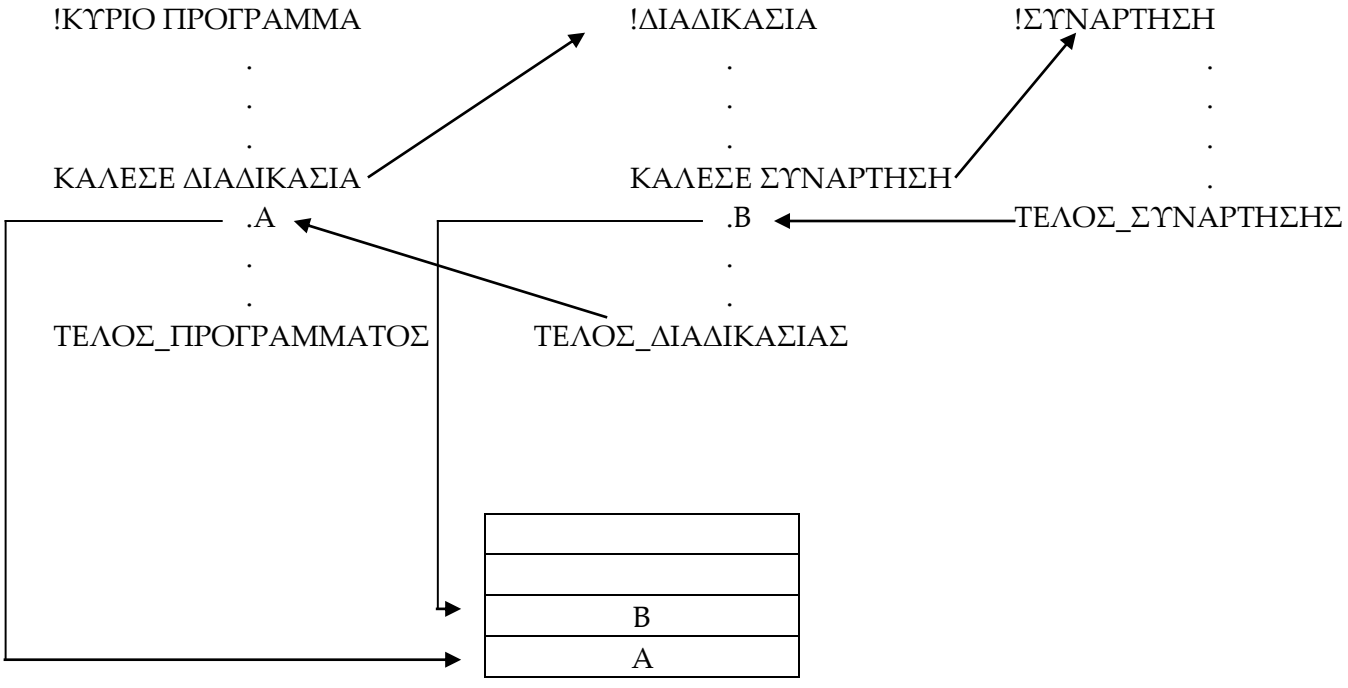
2.Κάθε πραγματική παράμετρος αντιστοιχεί στην τυπική παράμετρο που βρίσκεται στην αντίστοιχη θέση.

3.Η τυπική παράμετρος και η αντίστοιχή της πραγματική πρέπει να είναι του ίδιου τύπου.

Η ΧΡΗΣΗ ΣΤΟΙΒΑΣ ΣΤΗΝ ΚΛΗΣΗ ΥΠΟΠΡΟΓΡΑΜΜΑΤΩΝ

Όταν ένα υποπρόγραμμα καλείται από το κυρίως πρόγραμμα, τότε η αμέσως επόμενη διεύθυνση του κυρίως προγράμματος, η οποία λέγεται *διεύθυνση επιστροφής*, αποθηκεύεται από τον μεταφραστή σε μια στοίβα που λέγεται *στοίβα χρόνου εκτέλεσης*. Μετά την εκτέλεση του υποπρογράμματος η *διεύθυνση επιστροφής* απωθείται από τη στοίβα κι έτσι ο έλεγχος του προγράμματος μεταφέρεται και πάλι στο κυρίως πρόγραμμα.
Η τεχνική αυτή εφαρμόζεται και γενικότερα, δηλαδή οποτεδήποτε ένα υποπρόγραμμα καλεί ένα άλλο υποπρόγραμμα.

ΠΑΡΑΔΕΙΓΜΑ
Έστω ότι ένα πρόγραμμα καλεί μια διαδικασία κι αυτή με τη σειρά της καλεί μια συνάρτηση. Σε αυτήν την περίπτωση οι διευθύνσεις επιστροφής εμφανίζονται στη στοίβα με σειρά B, A. Μετά την εκτέλεση κάθε υποπρογράμματος, η διεύθυνση επιστροφής απωθείται από τη στοίβα κι ο έλεγχος μεταβιβάζεται στη διεύθυνση αυτή.(χρήση μεθόδου LIFO)



ΣΤΟΙΒΑ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ

ΕΜΒΕΛΕΙΑ ΜΕΤΑΒΛΗΤΩΝ ΚΑΙ ΣΤΑΘΕΡΩΝ

Κάθε πρόγραμμα και υποπρόγραμμα έχει τις δικές του μεταβλητές, οι οποίες είναι τοπικές δηλαδή ισχύουν μόνο στο πρόγραμμα ή το υποπρόγραμμα. Ο μόνος τρόπος για να περάσει μια τιμή από ένα υποπρόγραμμα στο πρόγραμμα ή σε κάποιο άλλο υποπρόγραμμα είναι δια μέσου των παραμέτρων κατά το στάδιο της κλήσης του υποπρογράμματος και μετά το τέλος της εκτέλεσης του υποπρογράμματος.
Το τμήμα του προγράμματος που ισχύουν οι μεταβλητές/σταθερές λέγεται **εμβέλεια** μεταβλητών/σταθερών.

Πολλές γλώσσες προγραμματισμού επιτρέπουν τη χρήση των μεταβλητών/σταθερών όχι μόνο στο τμήμα του προγράμματος που δηλώνονται αλλά και σε άλλα ή ακόμα και σε όλα τα υπόλοιπα υποπρογράμματα. Αυτό που καθορίζει την περιοχή που ισχύουν οι μεταβλητές/σταθερές είναι η εμβέλεια των μεταβλητών της γλώσσας.

ΑΠΕΡΙΟΡΙΣΤΗ ΕΜΒΕΛΕΙΑ

Όλες οι μεταβλητές/σταθερές είναι γνωστές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, άσχετα με το που δηλώθηκαν. Όλες οι μεταβλητές δηλαδή είναι **καθολικές.**
Η απεριόριστη εμβέλεια καταστρατηγεί την αρχή της αυτονομίας των υποπρογραμμάτων, δημιουργεί πολλά προβλήματα κι είναι αδύνατη σε μεγάλα προγράμματα που χρησιμοποιούν υποπρογράμματα αφού ο καθένας που γράφει ένα υποπρόγραμμα θα πρέπει να γνωρίζει τα ονόματα όλων των μεταβλητών που χρησιμοποιούνται στα υπόλοιπα υποπρογράμματα.

ΠΕΡΙΟΡΙΣΜΕΝΗ ΕΜΒΕΛΕΙΑ

Όλες οι μεταβλητές/σταθερές που χρησιμοποιούνται σε ένα τμήμα προγράμματος δηλώνονται σε αυτό μόνο το τμήμα, είναι δηλαδή **τοπικές** και ισχύουν μόνο στο υποπρόγραμμα στο οποίο δηλώθηκαν. Στη ΓΛΩΣΣΑ έχουμε περιορισμένη εμβέλεια.
Πλεονεκτήματα της περιορισμένης εμβέλειας είναι ότι τα υποπρογράμματα είναι αυτόνομα και μπορούμε να χρησιμοποιούμε οποιοδήποτε όνομα μεταβλητής χωρίς να μας ενδιαφέρει αν το ίδιο όνομα χρησιμοποιείται σε άλλο υποπρόγραμμα.

ΜΕΡΙΚΩΣ ΠΕΡΙΟΡΙΣΜΕΝΗ ΕΜΒΕΛΕΙΑ

Άλλες μεταβλητές είναι τοπικές και άλλες καθολικές. Αυτό εξαρτάται από τους κανόνες της κάθε γλώσσας προγραμματισμού.
Προσφέρει κάποια πλεονεκτήματα σε έναν πεπειραμένο προγραμματιστή αλλά για έναν αρχάριο προγραμματιστή περιπλέκει το πρόγραμμα.

ΚΕΦΑΛΑΙΟ 13-ΕΚΣΦΑΛΜΑΤΩΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΚΑΤΗΓΟΡΙΕΣ ΛΑΘΩΝ

Σε ένα πρόγραμμα είναι δυνατό να παρουσιαστούν διαφορετικής μορφής λάθη, τα οποία μπορούν να χωριστούν σε τρεις βασικές κατηγορίες:

1. Λάθη κατά την υλοποίηση

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.
Ένα λάθος που προκαλείται κατά τη συγγραφή του προγράμματος ανιχνεύεται από το μεταγλωττιστή, ο οποίος εμφανίζει προς τον προγραμματιστή κάποιο προειδοποιητικό μήνυμα. Αν το πρόγραμμα περιέχει ένα λάθος αυτής της μορφής, δεν επιτρέπεται η εκτέλεσή του, μέχρι να το διορθώσει ο προγραμματιστής.
Τα σύγχρονα προγραμματιστικά περιβάλλοντα μας προφυλάσσουν αυτόματα από τα λάθη κατά την υλοποίηση, αφού παρέχουν εργαλεία αυτόματου ελέγχου σύνταξης των εντολών και παρακολουθούν τον προγραμματιστή κατά τη συγγραφή του προγράμματος. Μόλις διαπιστώσουν κάποιο συντακτικό λάθος, σταματούν και απαιτούν τη διόρθωσή του. Συνήθως αντιλαμβάνονται ακριβώς το λάθος που δημιουργήθηκε και προτείνουν αναλυτικά τον τρόπο διόρθωσής του, εμφανίζοντας σε ενημερωτικό πλαίσιο την ορθή σύνταξη της εντολής που προκλήθηκε το λάθος.

2. Λάθη κατά την εκτέλεση

Τα λάθη που προκαλούνται κατά το χρόνο εκτέλεσης του προγράμματος είναι πιο επώδυνα γιατί συνήθως εμφανίζονται σε πραγματικό περιβάλλον εκτέλεσης και τις περισσότερες φορές προκαλούν τον αντικανονικό τερματισμό της εφαρμογής και το *κρέμασμα* (crash) του συστήματος. Όταν ένα λάθος προκληθεί κατά την εκτέλεση της εφαρμογής, είναι δυνατό να αντιμετωπισθεί μόνο με τη χρήση εντολών προγράμματος που το παγιδεύουν και εκτελούν τις κατάλληλες διαδικασίες χειρισμού του.
Η πρόληψη τέτοιων λαθών είναι αρκετά δύσκολη, αφού συνήθως οφείλονται σε καταστάσεις που δεν είναι εύκολο να ελεγχθούν από τον προγραμματιστή, ενώ πολλές φορές εμφανίζονται μετά από μεγάλο χρονικό διάστημα. Τέτοια λάθη είναι δυνατό να προκληθούν από την κλήση μιας διαδικασίας με δεδομένα που δεν μπορεί να χειριστεί, όπως η αναζήτηση διαγραμμένων αρχείων, η προσπάθεια διαίρεσης ενός αριθμού με το μηδέν, 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.

3. Λογικά λάθη

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.

ΕΚΣΦΑΛΜΑΤΩΣΗ

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. Η εργασία της εκσφαλμάτωσης δεν είναι εύκολη, απαιτεί βαθιά γνώση της γλώσσας προγραμματισμού και φυσικά αντίστοιχες ικανότητες από τον προγραμματιστή. Τα λάθη που κυρίως μας απασχολούν στη φάση της εκσφαλμάτωσης είναι τα λογικά λάθη και τα λάθη που παρουσιάζονται κατά το χρόνο εκτέλεσης του προγράμματος.