ΚΕΦΑΛΑΙΟ 1 : Ανάλυση προβλήματος

Πρόβλημα:

Εννοούμε μία κατάσταση που χρήζει αντιμετώπισης (επίλυσης), η δε λύση της δεν είναι γνωστή και ούτε προφανής.

Στάδια αντιμετώπισης ενός προβλήματος:

Κατανόηση προβλήματος:

Εξαρτάται από δύο παράγοντες:

1) Τη σαφή διατύπωση από εκείνον που θέτει το πρόβλημα. Να μην αφήνει δηλαδή παρερμηνείες και ασάφειες)
2) Τη σωστή ερμηνεία από εκείνον που θα κληθεί να επιλύσει το πρόβλημα. Να το έχει δηλαδή κατανοήσει – καταλάβει.

Ανάλυση προβλήματος:

Σημαίνει ότι ξεκινάμε να αποκαλύπτουμε τη *δομή του προβλήματος*, δηλαδή να χωρίσουμε το πρόβλημα σε μικρότερα και απλούστερα υπο-προβλήματα, καθένα από τα οποία να λύνεται ευκολότερα.

Με τον όρο ***δομή προβλήματος***, εννοούμε τα συστατικά μέρη από τα οποία συντίθεται το πρόβλημα, δηλαδή τα επιμέρους τμήματα που το αποτελούν καθώς και τον τρόπο με τον οποίο αυτά συνδέονται μεταξύ τους.

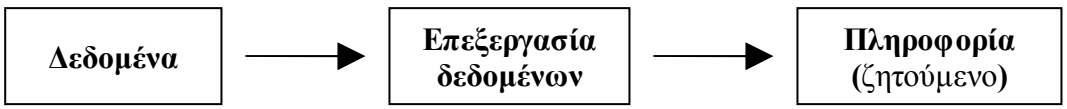
Η *ανάλυση* μπορεί να γίνει με δομημένο λεκτικό τρόπο ή καλύτερα χρησιμοποιώντας ένα *ιεραρχικό διάγραμμα*.

Επίλυση προβλήματος:

Για τη σωστή επίλυση του προβλήματος βασική προϋπόθεση είναι ο *καθορισμός απαιτήσεων*, δηλαδή να:

α) προσδιορίσουμε τα **δεδομένα** που παρέχονται, και να

β) προσδιορίσουμε τα **ζητούμενα**, δηλαδή τι περιμένουμε σας αποτέλεσμα.



- Τα **δεδομένα**, είναι οτιδήποτε στοιχείο μπορεί να γίνει αντιληπτό από τον άνθρωπο.

- Η **επεξεργασία δεδομένων** είναι μία διαδικασία κατά την οποία ένας μηχανισμός (π.χ. ο Η/Υ) κάνει πράξεις και υπολογισμούς στα δεδομένα ώστε να προκύψει μία χρήσιμη πληροφορία.
  Για παράδειγμα: η μέση θερμοκρασία κατά τον μήνα Μάρτιο (πληροφορία), προκύπτει από τον υπολογισμό του μέσου όρου (επεξεργασία) των ημερήσιων θερμοκρασιών του μήνα (δεδομένα).
- Η **πληροφορία** είναι γνωσιακό στοιχείο που προκύπτει από την επεξεργασία των δεδομένων.

Κατηγορίες προβλημάτων:

Με κριτήριο τη δυνατότητα επίλυσης, τα χωρίζουμε σε:

Επιλύσιμα	Η λύση τους είναι γνωστή κι έχει διατυπωθεί *(π.χ. το πρόβλημα του υπολογισμού του εμβαδού του κύκλου)*
Ανοικτά	Η λύση τους δεν έχει βρεθεί αλλά παράλληλα δεν έχει αποδειχτεί ότι δεν επιδέχονται λύση *(π.χ. το πρόβλημα της ενοποίησης των τεσσάρων δυνάμεων της φυσικής).*
Άλυτα	Έχουμε παραδεχτεί ότι δεν υπάρχει λύση *(π.χ. ο τετραγωνισμός του κύκλου)*

Τα *επιλύσιμα προβλήματα*, ανάλογα με το βαθμό δόμησης των λύσεων τους (πόσο αυτόματα δηλαδή βγαίνει η λύση τους) διακρίνονται σε:

Δομημένα	Η λύση τους είναι μία αυτοματοποιημένη διαδικασία *(π.χ. η επίλυση της δευτεροβάθμιας εξίσωσης)*
Ημιδομημένα	Η λύση τους μπορεί να προέλθει από πλήθος πιθανών λύσεων που εμείς επιλέγουμε *(π.χ. το πρόβλημα του πώς θα πάμε εκδρομή σε ένα μέρος. Μπορούμε να διαλέξουμε ένα πλήθος από πιθανούς τρόπους κτλ)*
Αδόμητα	Η λύση τους δεν είναι αυτοματοποιημένη, δηλαδή δεν μπορούμε να βρούμε ένα συγκεκριμένο τρόπο λύσης. Για την επίλυσή τους βασιζόμαστε στην ανθρώπινη διαίσθηση και εμπειρία *(π.χ. ο τρόπος οργάνωσης ενός πάρτι)*

Με κριτήριο το είδος επίλυσης, τα χωρίζουμε σε:

Απόφασης	Η λύση σε αυτά τα προβλήματα είναι του τύπου «Ναι» και «Όχι». *(π.χ. ο αριθμός 101 είναι πρώτος;)*
Υπολογιστικά	Για την επίλυση απαιτείται η διενέργεια υπολογισμών. *(π.χ. για τον αριθμό Ν να βρεθεί το Ν!)*
Βελτιστοποίησης	Η λύση που ζητάμε είναι το βέλτιστο αποτέλεσμα για τα συγκεκριμένα δεδομένα. *(π.χ. ποιος είναι ο συντομότερος δρόμος για να πάμε σε ένα μέρος)*

Οι λόγοι που αναθέτουμε την επίλυση ενός προβλήματος σε Η/Υ είναι:

1) Η πολυπλοκότητα των υπολογισμών.
2) Η ταχύτητα εκτέλεσης των πράξεων.
3) Ο μεγάλος όγκος δεδομένων.

4) Η επαναληπτικότητα των διαδικασιών.

Ο Η/Υ στα κυκλώματά του, εκτελεί μόνο τρεις λειτουργίες:

1) Πρόσθεση (όλες οι άλλες αριθμητικές πράξεις γίνονται μέσω πρόσθεσης).
2) Σύγκριση (που αποτελεί βασική λειτουργία των λογικών πράξεων).
3) Μεταφορά δεδομένων.

Γενικότερα, η ικανότητα του Η/Υ εκφράζεται σε *ποσοτικό* επίπεδο ενώ η ικανότητα του ανθρώπου σε *ποιοτικό* επίπεδο. Σκεφτείτε πόσους υπολογισμούς κάνει ο Η/Υ στο σκάκι για να βρει μια καλή κίνηση, ενώ ο άνθρωπος το κάνει με πολύ λιγότερες.

ΚΕΦΑΛΑΙΟ 2 : Βασικές έννοιες αλγορίθμων

Αλγόριθμος:

Είναι ένα σύνολο βημάτων, αυστηρά καθορισμένων και εκτελεσμένων σε πεπερασμένο χρόνο, που οδηγούν στην επίλυση ενός προβλήματος.

Χαρακτηριστικά (κριτήρια) ενός σωστού αλγόριθμου, είναι:

Είσοδος	Καμία, μία ή περισσότερες τιμές δεδομένων δίνονται ως είσοδοι
Έξοδος	Πρέπει να δημιουργηθεί μία τιμή δεδομένων ως αποτέλεσμα
Περατότητα	Να τελειώνει, φτάνοντας στο επιθυμητό αποτέλεσμα, σε πεπερασμένο αριθμό βημάτων ή χρόνο. Αν δεν συμβαίνει αυτό τότε είναι απλώς *υπολογιστική διαδικασία*.
Σαφήνεια	Τα επιμέρους βήματα να είναι απλά και σαφή.
Εκτελεσιμότητα	Τα βήματα να είναι έτσι διατυπωμένα, ώστε να μπορούν να εκτελεστούν από τον Η/Υ.
Πληρότητα	Ο αλγόριθμος να προβλέπει κάθε πιθανό ενδεχόμενο κατά την εκτέλεση του *(π.χ. μία διαίρεση με το μηδέν)*
Αποτελεσματικότητα	Μετά το πέρας της εκτέλεσης των βημάτων να έχει επιτευχθεί ο στόχος *(το ζητούμενο)*

Τρόποι αναπαράστασης ενός αλγορίθμου:

1) **Με ελεύθερο κείμενο:** Αποτελεί έναν αδόμητο τρόπο περιγραφής, που μπορεί να καταστρατηγήσει εύκολα τα κριτήρια του σωστού αλγόριθμου (ασάφειες, μη εκτελεσιμότητα κτλ).
2) **Με διαγραμματικές τεχνικές:** Αποτελεί έναν γραφικό τρόπο παρουσίασης των εντολών. Η πιο γνωστή τεχνική είναι το ***διάγραμμα ροής*** *(flow chart)* που παλαιότερα ήταν αρκετά δημοφιλή.
3) **Με φυσική γλώσσα κατά βήματα:** Χρειάζεται προσοχή, διότι μπορεί να παραβιαστεί το κριτήριο της πληρότητας. Δεν είναι προτιμητέος τρόπος.
4) **Με κωδικοποίηση:** Στην περίπτωση αυτή, δημιουργούμε τις εντολές κατευθείαν σε γλώσσα προγραμματισμού.

Στα παρακάτω παραδείγματα, χρησιμοποιούμε μια κωδικοποίηση σε μία μη υπαρκτή δομημένη γλώσσα προγραμματισμού, καλούμενη ως *ψευδογλώσσα*. Φτιάχνοντας τον αλγόριθμο με ψευδογλώσσα, μπορούμε κατόπιν να τον μετατρέψουμε χωρίς πολλές μετατροπές σε μία υπαρκτή γλώσσα προγραμματισμού.

Βασικά στοιχεία αλγορίθμου:

1) **Σταθερές:** Αφορούν ποσότητες που δεν μεταβάλλονται κατά τη διάρκεια εκτέλεσης του αλγόριθμου. Είναι τριών ειδών:
   i. *Αριθμητικές* π.χ. 100, -5, π
   ii. *Αλφαριθμητικές* π.χ. "Νίκος", "Νομός Ηρακλείου". Ονομάζονται και *λεκτικά* ή *συμβολοσειρές*. Περικλείονται σε ".
   iii. *Λογικές* π.χ. True, False.
2) **Μεταβλητές:** Αφορούν ποσότητες που μεταβάλλονται κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Παριστάνουν μια θέση μνήμης που περιέχει μία τιμή. Το περιεχόμενο αυτής της θέσης (η τιμή της) αλλάζει.

X ← Όνομα μεταβλητής ➔ Name

100 ← Τιμή ➔ "Νίκος"

Στις μεταβλητές δίνουμε ένα όνομα (αναγνωριστικό) και αυτό χρησιμοποιούμε στον αλγόριθμο. Η τιμή της αλλάζει με μία εντολή εκχώρησης τιμής π.χ. **X ← 100** ή **Name ← "Νίκος".**

Ανάλογα με το είδος του δεδομένου που εκχωρείται η μεταβλητή διακρίνεται σε:
   i. *Αριθμητική* π.χ. X ← 100
   ii. *Αλφαριθμητική* π.χ. Name ← "Νίκος"
   iii. *Λογική* π.χ. Έγγαμος ← False.

Επιπλέον, οι *αριθμητικές* μεταβλητές διακρίνονται σε:
   i. *Ακέραιες*, αν δέχονται ακέραια τιμή π.χ. 5, -8
   ii. *Πραγματικές*, αν δέχονται πραγματική τιμή π.χ. 5.8, -102.345

3) **Τελεστές:** Είναι τα γνωστά σύμβολα. Είναι 3 ειδών:
   i. *Αριθμητικοί:* +, -, * (πολλ/σμός), / (διαίρεση), ^ (ύψωση σε δύναμη), DIV (ακέραια διαίρεση), MOD (ακέραιο υπόλοιπο). Π.χ. X ← 5 * 2, X ← 3^4 (=81), X ← 5 DIV 2 (= 2 πηλίκο), X ← 5 MOD 2 (= 1 υπόλοιπο)
   ii. *Λογικοί:* AND, OR, NOT π.χ. Έγκυρος ← (X>1) AND (X<=20)
   iii. *Συγκριτικοί:* >, >=, <, <=, <> (διάφορο)
4) **Εκφράσεις:** Συνδυάζουν όλα τα παραπάνω, π.χ. Έγκυρος ← (X>1) AND (X<=20)

Βασικές εντολές αλγορίθμου:

1) *Εντολή εισόδου:* **Διάβασε**, π.χ. Διάβασε X. Στη μεταβλητή X εισάγεται μια τιμή.
2) *Εντολή εξόδου:* **Τύπωσε ή Εμφάνισε** π.χ. Τύπωσε X
3) *Εντολή εκχώρησης τιμής:* **Μεταβλητή ← Έκφραση**. Αριστερά βάζουμε το όνομα της μεταβλητής και δεξιά μία τιμή ή έκφραση. π.χ. X ← (2 * 4)/3. Το αποτέλεσμα της πράξης εκχωρείται στη μεταβλητή.

Μπορούμε λοιπόν να τοποθετήσουμε τιμές σε μεταβλητές:

i. Με εντολή εισόδου **Διάβασε**, και
ii. Κατευθείαν με μία εντολή εκχώρησης.

Αλγοριθμικές δομές:

Εννοούμε τον τρόπο που θέτουμε τα βήματα (εντολές) στον αλγόριθμο, ώστε να έχουμε σωστή και αυστηρή (δομημένη) σύνταξη και να πληρούνται τα χαρακτηριστικά του. Οι αλγοριθμικές δομές είναι τρεις:

i. *Ακολουθιακή:* Οι εντολές εκτελούνται η μία μετά την άλλη.
ii. *Επιλογής:* Μία ομάδα εντολών εκτελούνται ανάλογα με την τιμή μιας συνθήκης (απόφαση).
iii. *Επαναληπτική:* Μία ομάδα εντολών εκτελούνται συνεχώς επαναληπτικά ανάλογα με την τιμή μιας συνθήκης (βρόχος – loop).

Ακολουθιακή δομή:

Παράδειγμα: Να γραφεί αλγόριθμος που διαβάζει τη βάση και το ύψος ενός τριγώνου και υπολογίζει και τυπώνει το Εμβαδόν του.

Σημείωση: Πρέπει να εισαχθούν δύο αριθμητικά δεδομένα (Βάση και Ύψος) και να εξαχθεί σαν αποτέλεσμα το Εμβαδόν. Για την αποθήκευση των τιμών θέλουμε τρεις μεταβλητές.

Αλγόριθμος Εμβαδόν_τριγώνου
Μεταβλητές
Ακέραιοι: Βάση, Ύψος
Πραγματικός: Εμβαδόν
Αρχή
Διάβασε Βάση
Διάβασε Ύψος
Εμβαδόν ← (Βάση * Ύψος) / 2
Τύπωσε Εμβαδόν

Τέλος Εμβαδόν_τριγώνου

Δομή επιλογής:

Έχει τρεις μορφές:

```
Αν <συνθήκη> τότε
        Εντολές
Τέλος_αν
```

Μορφή I

```
Αν <συνθήκη> τότε
        Εντολές1
Αλλιώς
        Εντολές2
Τέλος_αν
```

Μορφή II

```
Επίλεξε μεταβλητή
  Περίπτωση 1 : Εντολές1
  Περίπτωση 2 : Εντολές2
  .....
  Περίπτωση ν : Εντολέςν
Τέλος_επιλογών
```

Μορφή III

- **Μορφή I:** Εάν η συνθήκη δώσει True (αληθής), τότε εκτελείται η ομάδα εντολών.

Παράδειγμα:

Να γραφεί αλγόριθμος που διαβάζει έναν ακέραιο αριθμό κι αν αυτός είναι θετικός να βγάζει το μήνυμα «Είναι θετικός».

Αλγόριθμος Θετικός
Μεταβλητές
Ακέραιος: Χ *! ο αριθμός θα εισαχθεί στην μεταβλητή με όνομα Χ*
Αρχή
Διάβασε Χ
Αν Χ > 0 **τότε**
Τύπωσε «Είναι θετικός»
Τέλος_αν
Τέλος Θετικός

- **Μορφή ΙΙ:** Αν η συνθήκη δώσει True (αληθής), τότε εκτελείται η ομάδα των Εντολών1, ενώ αν δώσει False (ψευδής) εκτελείται η ομάδα Εντολών2.

Παράδειγμα:

Να γραφεί αλγόριθμος που διαβάζει έναν ακέραιο αριθμό κι αν αυτός είναι θετικός να βγάζει το μήνυμα «Είναι θετικός», ενώ αν είναι αρνητικός να βγάζει το μήνυμα «Δεν είναι θετικός».

Αλγόριθμος Έλεγχος_αριθμού
Μεταβλητές
Ακέραιος: Χ *! ο αριθμός θα εισαχθεί στην μεταβλητή με όνομα Χ*
Αρχή
Διάβασε Χ
Αν Χ > 0 **τότε**
Τύπωσε «Είναι θετικός»
αλλιώς
Τύπωσε «Δεν είναι θετικός»
Τέλος_αν
Τέλος Έλεγχος_αριθμού

- **Μορφή ΙΙΙ:** Στην *πολλαπλή επιλογή* εξετάζεται η τιμή μιας μεταβλητής μέσα σε ένα πλήθος τιμών. Αν ισούται με κάποια από τις περιπτώσεις τότε εκτελείται η αντίστοιχη ομάδα εντολών.

Παράδειγμα:

Να γραφεί αλγόριθμος που διαβάζει τον αριθμό της ημέρας και τυπώνει την αντίστοιχη περιγραφή.

Αλγόριθμος Μέρα_εβδομάδας
Μεταβλητές
Ακέραιος: Μέρα

```
Αρχή
        Διάβασε Μέρα
        Επίλεξε Μέρα
                Περίπτωση 1: Τύπωσε «Είναι Κυριακή»
                Περίπτωση 2: Τύπωσε «Είναι Δευτέρα»
                Περίπτωση 3: Τύπωσε «Είναι Τρίτη»
                Περίπτωση 4: Τύπωσε «Είναι Τετάρτη»
                Περίπτωση 5: Τύπωσε «Είναι Πέμπτη»
                Περίπτωση 6: Τύπωσε «Είναι Παρασκευή»
                Περίπτωση 7: Τύπωσε «Είναι Σάββατο»
        Τέλος_επιλογών
Τέλος Μέρα_εβδομάδας
```

Εμφωλευμένες δομές ελέγχου:

Πρόκειται για τις περιπτώσεις που έχουμε μία δομή **Αν**...**τότε**... μέσα σε μία άλλη.

```
Αν <συνθήκη1> τότε
        Εντολές1
αλλιώς
        Αν <συνθήκη2> τότε
                Εντολές2
        Τέλος_αν
Τέλος_αν
```

Πιο απλά γράφεται

```
Αν <συνθήκη1> τότε
        Εντολές1
αλλιώς_αν <συνθήκη2> τότε
        Εντολές2
Τέλος_αν
```

Μπορούμε να έχουμε πολλά **αλλιώς_αν** καταλήγοντας σε μία μορφή *πολλαπλής επιλογής*, όπου είναι καλύτερα να εκφραστεί με την εντολή **Επίλεξε... Τέλος_επιλογών**.

Παράδειγμα:

Να γραφεί αλγόριθμος που διαβάζει το βαθμό ενός μαθητή και τυπώνει το χαρακτηρισμό του, π.χ. εάν διαβάζει 19 να τυπώνει «Άριστα».

```
Αλγόριθμος Χαρακτηρισμός_βαθμού
Μεταβλητές
        Πραγματικός: Β    ! ο βαθμός που θα διαβαστεί
Αρχή
        Διάβασε Β
        Αν Β < 10 τότε
                Τύπωσε «Καλώς»
        αλλιώς_αν Β >= 10 AND Β < 12.5 τότε
                Τύπωσε «Μέτρια»
        αλλιώς_αν Β >= 12.5 AND Β < 15.5 τότε
                Τύπωσε «Καλά»
        αλλιώς_αν Β >= 15.5 AND Β < 18.5 τότε
                Τύπωσε «Πολύ καλά»
```

αλλιώς_αν Β >= 18.5 AND Β <= 20 **τότε**
Τύπωσε «Άριστα»
Τέλος_αν

Τέλος Χαρακτηρισμός_βαθμού

Η παραπάνω εμφωλευμένη επιλογή μπορεί να γραφτεί ισοδύναμα και πιο σύντομα με την επιλογή **Επίλεξε...Τέλος_επιλογών**.

Πώς?

Επαναληπτική δομή:

Έχει τρεις μορφές:

Όσο <συνθήκη> **επανέλαβε** Εντολές **Τέλος_επανάληψης**

Μορφή I

Αρχή_επανάληψης Εντολές **Μέχρις ότου** <συνθήκη>

Μορφή II

Για <μετρητής> **από** <αρχική τιμή> **μέχρι** <τελική τιμή> [**με βήμα** Βήμα] Εντολές **Τέλος_επανάληψης**

Μορφή III

Σημείωση: Στις περισσότερες περιπτώσεις (και πάντα στη μορφή ΙΙΙ) χρειαζόμαστε μία ***μεταβλητή-μετρητή****. Ο μετρητής (συνήθως τον ονομάζουμε i) μετρά τον αριθμό των επαναλήψεων.*

- **Μορφή I:** Όσο η συνθήκη δίνει True οι Εντολές εκτελούνται συνεχώς. Αν δώσει False τότε σταματάει η επανάληψη. Να σημειωθεί ότι πρώτα ελέγχεται η συνθήκη. Συνεπώς οι Εντολές μπορούν να εκτελεστούν μηδέν, μία ή περισσότερες φορές.

  Παράδειγμα:

  Να γραφεί αλγόριθμος που διαβάζει 10 αριθμούς και υπολογίζει το άθροισμά τους.

```
Αλγόριθμος Άθροισμα_αριθμών
Μεταβλητές
        Ακέραιος: X          ! ο αριθμός που διαβάζεται
        Ακέραιος: SUM        ! εδώ θα αποθηκευτεί το άθροισμα
        Ακέραιος: i          ! ο μετρητής
Αρχή
        i ← 1                ! ο μετρητής αρχικοποιείται πριν ξεκινήσει η επανάληψη
        SUM ← 0              ! το ίδιο και το άθροισμα
        Όσο i <= 10 επανέλαβε
                Διάβασε X            ! κάνε εισαγωγή του αριθμού στη μεταβλητή X
                SUM ← SUM + X        ! πρόσθεσέ τον στο άθροισμα
                i ← i + 1            ! αυξάνουμε τον μετρητή πριν κάνει την επόμενη
                                     ! επανάληψη
        Τέλος_επανάληψης
        Τύπωσε "Το άθροισμα είναι :", SUM
Τέλος Άθροισμα_αριθμών
```

Σε αυτό το παράδειγμα ο αριθμός των επαναλήψεων είναι γνωστός. Όταν είναι γνωστός ο αριθμός των επαναλήψεων προτιμότερη είναι η Τρίτη μορφή **Για...από...μέχρι... .**
Παράδειγμα 2: (επαναληπτική είσοδος αγνώστου πλήθους στοιχείων)

Να γραφεί αλγόριθμος που διαβάζει μια ακολουθία αριθμών όπου ο τελευταίος είναι 0 και να υπολογίζει το άθροισμά τους.

Αλγόριθμος Άθροισμα_αριθμών
Μεταβλητές
Ακέραιος: X
Ακέραιος: SUM
! δεν χρειαζόμαστε μετρητή μιας και το πλήθος των αριθμών είναι άγνωστο. Το μόνο που
! γνωρίζουμε είναι ότι ο τελευταίος αριθμός είναι το 0. Άρα θα διαβάζει ο Η/Υ μέχρι να
! συναντήσει το 0.
Αρχή
SUM ← 0
Διάβασε X *! ο πρώτος διαβάζεται πριν μπει στην επανάληψη*
Όσο X <> 0 **επανέλαβε**
SUM ← SUM + X *! πρόσθεσέ τον στο άθροισμα*
Διάβασε X *! διάβασε τον επόμενο*
Τέλος_επανάληψης
Τύπωσε "Το άθροισμα είναι :", SUM

Τέλος Άθροισμα_αριθμών

- **Μορφή ΙΙ:** Εκτελείται πρώτα η ομάδα των εντολών και μετά ελέγχεται η συνθήκη. Συνεπώς οι εντολές εκτελούνται μία ή περισσότερες φορές. Εδώ, όσο η συνθήκη δίνει False η επανάληψη εκτελείται, εάν δώσει True σταματά. Ισχύει δηλαδή το αντίθετο από την **Όσο...επανέλαβε**.

Για το ίδιο παράδειγμα 2, με τη μορφή ΙΙ θα γράφαμε:

Αλγόριθμος Άθροισμα_αριθμών
Μεταβλητές
Ακέραιος: X
Ακέραιος: SUM
! δεν χρειαζόμαστε μετρητή μιας και το πλήθος των αριθμών είναι άγνωστο. Το μόνο που
! γνωρίζουμε είναι ότι ο τελευταίος αριθμός είναι το 0. Άρα θα διαβάζει ο Η/Υ μέχρι να
! συναντήσει το 0.
Αρχή
SUM ← 0
Αρχή_επανάληψης
Διάβασε X
SUM ← SUM + X
μέχρις_ότου X = 0
Τύπωσε "Το άθροισμα είναι :", SUM

Τέλος Άθροισμα_αριθμών

- **Μορφή ΙΙΙ:** Είναι η καταλληλότερη κι ενδείκνυται όταν γνωρίζουμε τον αριθμό των επαναλήψεων. Χρησιμοποιεί απαραίτητα μετρητή. Η μορφή αυτή ενσωματώνει:
  - Την αρχικοποίηση του μετρητή.
  - Τον έλεγχο της συνθήκης, αν φτάσαμε στην τελική τιμή.
  - Την αύξηση του μετρητή. Αν δεν καθορίσουμε βήμα, τότε εννοείται βήμα 1.

Το παραπάνω παράδειγμα 1 ισοδύναμα γράφεται:

```
Αλγόριθμος Άθροισμα_αριθμών
Μεταβλητές
        Ακέραιος: X          ! ο αριθμός που διαβάζεται
        Ακέραιος: SUM        ! εδώ θα αποθηκευτεί το άθροισμα
        Ακέραιος: i          ! ο μετρητής
Αρχή
        SUM ← 0
        Για i από 1 μέχρι 10
                Διάβασε X
                SUM ← SUM + X
        Τέλος_επανάληψης
        Τύπωσε "Το άθροισμα είναι :", SUM

Τέλος Άθροισμα_αριθμών
```

Σύνδεση λογικών προτάσεων:

Στις δομές ελέγχου και στις επαναληπτικές δομές χρησιμοποιούμε <συνθήκη> για να καθοδηγήσουμε τον Η/Υ προς ένα ή άλλο μονοπάτι εκτέλεσης εντολών. Η <συνθήκη> είναι μια λογική πρόταση που δίνει True ή False και μπορεί να είναι:

Απλή: π.χ. i < 10 ή
Σύνθετη: π.χ. B >= 18.5 AND B <=20

Στη δεύτερη περίπτωση, χρησιμοποιούμε λογικές προτάσεις που συνδέονται μεταξύ τους με τους λογικούς τελεστές **AND** ή **OR**. Επίσης, χρησιμοποιούμε και το **NOT** (όχι) για την αντιστροφή της τιμής μιας πρότασης.

πρόταση **A**	πρόταση **B**	A **OR** B	A **AND** B	**NOT** A
True	True	True	True	False
True	False	True	False	False
False	True	True	False	True
False	False	False	False	True

Γενικά, στην περίπτωση OR, όταν μία πρόταση είναι αληθής (True) τότε βγαίνει συνολικά αληθής. Στην περίπτωση του AND, όταν μία πρόταση είναι ψευδής (False) τότε βγαίνει συνολικά ψευδής.

Πολλαπλασιασμός αλά ρώσικα:

Είναι μία μέθοδος πολλαπλασιασμού δύο αριθμών. Συγκεκριμένα, άν έχουμε δύο αριθμούς Χ και Υ και θέλουμε Χ*Υ τότε αυτό γίνεται ως εξής:

i. **Βήμα 1:** Ο Χ διπλασιάζεται και ο Υ υποδιπλασιάζεται, κρατώντας μόνο το ακέραιο μέρος.
ii. **Βήμα 2:** Συνεχίζουμε το Βήμα 1, μέχρι ο Υ να φτάσει το 1.
iii. **Βήμα 3:** Το γινόμενο θα είναι το άθροισμα των στοιχείων Χ στα βήματα που το Υ είναι περιττός.

π.χ. Χ = 45, Υ = 19

X	**Y**	
45	19	45
90	9	90
180	4	--
360	2	--
720	1	720
Άθροισμα		**855**

```
Αλγόριθμος Πολλαπλασιασμός_αλά_ρώσικα
Μεταβλητές
        Ακέραιοι: X, Y
        Ακέραιος: P             ! το αποτέλεσμα-γινόμενο
Αρχή
        P ← 0                   ! αρχικοποίηση γινομένου
        Όσο Y > 0 επανέλαβε
                Αν (Y MOD 2) = 1 τότε    ! αν το υπόλοιπο της διαίρεση του Y με το 2 είναι 1
                        P ← P + X        ! πρόσθεσε το X στο άθροισμα
                Τέλος_αν
                X ← X * 2                ! διπλασιασμός του X
                Y ← Y DIV 2              ! υποδιπλασιασμός του Y. Παίρνουμε το πηλίκο του
                                         !Y με το 2
        Τέλος_επανάληψης
        Τύπωσε "Το γινόμενο είναι :", P

Τέλος Πολλαπλασιασμός_αλά_ρώσικα
```

Σπουδαιότητα των αλγορίθμων στην Πληροφορική:

Η Πληροφορική μελετά τους αλγορίθμους από τις ακόλουθες σκοπιές:

1) **Υλικού:** Η ταχύτητα εκτέλεσης ενός αλγορίθμου επηρεάζεται από το υπάρχον hardware του Η/Υ. Δηλαδή, από την ταχύτητα του επεξεργαστή, της μνήμης, των δίσκων κτλ.
2) **Γλωσσών προγραμματισμού:** Κάθε γλώσσα προγραμματισμού έχει τα δικά της ιδιαίτερα χαρακτηριστικά που επηρεάζουν τον αριθμό και τον τρόπο δόμησης των εντολών του αλγόριθμου που θα χρησιμοποιηθούν για την επίλυση του προβλήματος.
3) **Θεωρητική:** Εξετάζουμε αν υπάρχει ή όχι κάποιος αποδοτικός αλγόριθμος για την επίλυση ενός συγκεκριμένου προβλήματος.
4) **Αναλυτική:** Εξετάζουμε τους υπολογιστικούς πόρους που σπαταλά ένας αλγόριθμος κατά την εκτέλεσή του. Δηλαδή, πόσο χρόνο απασχολεί τον επεξεργαστεί, πόση RAM δεσμεύει, πόσο χρόνο απαιτεί για λειτουργίες εισόδου / εξόδου κτλ.

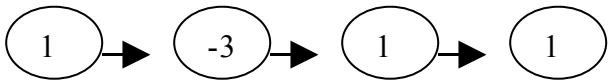
ΚΕΦΑΛΑΙΟ 3 : Δομές Δεδομένων και Αλγόριθμοι

Δομή δεδομένων:

Είναι ένας τρόπος αποθήκευσης (οργάνωσης) δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.

Η δομή αποτελείται από ένα σύνολο *κόμβων*. Κάθε κόμβος περιέχει μια απλή τιμή (αριθμός, αλφαριθμητικό κτλ) η μία πιο σύνθετη τιμή (π.χ. μια εγγραφή).

π.χ. δομή λίστας που περιέχει ακέραιους αριθμούς:



π.χ. δομή λίστας που περιέχει εγγραφές (Εγγραφή = μία ομάδα τιμών διαφορετικού τύπου σχετικά με ένα αντικείμενο. Π.χ. μία εγγραφή μαθητή περιέχει το όνομα, το επώνυμο, το έτος γέννησης, την τάξη του κτλ.)

Βασικές λειτουργίες (πράξεις) επί των δομών:

Προσπέλαση (access)	Είναι η δυνατότητα πρόσβασης σε ένα κόμβο με σκοπό την ανάγνωση ή τροποποίηση του περιεχομένου του.
Αναζήτηση (searching)	Προσπελαύνονται οι κόμβοι μιας δομής με σκοπό να εντοπιστεί κάποιος που περιέχει μία συγκεκριμένη τιμή.
Εισαγωγή (insertion)	Προσθήκη νέου κόμβου στη δομή.
Διαγραφή (deletion)	Αφαίρεση ενός κόμβου από τη δομή.
Ταξινόμηση (sorting)	Διατάσουμε τους κόμβους της δομής σε αύξοντα ή φθίνουσα σειρά (π.χ. αλφαβητικά κατά επώνυμο ή όνομα).
Αντιγραφή (copying)	Όλοι ή μερικοί κόμβοι αντιγράφονται σε μία άλλη δομή.
Συγχώνευση (merging)	Δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή
Διαχωρισμός (separation)	Μία δομή διασπάται σε δύο ή περισσότερες. Είναι το αντίστροφο της συγχώνευσης.

Για λειτουργία δημιουργείται και ένας αλγόριθμος. Αρκετές φορές υπάρχουν διαφορετικοί αλγόριθμοι που υλοποιούν μία συγκεκριμένη λειτουργία (π.χ. υπάρχουν αρκετοί αλγόριθμοι για τη λειτουργία της ταξινόμησης). Όταν συμβαίνει αυτό, επιλέγουμε ένα αλγόριθμο που είναι πιο αποδοτικός (π.χ. πιο γρήγορος) για τα συγκεκριμένα δεδομένα.

ΠΡΟΓΡΑΜΜΑ = ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ + ΑΛΓΟΡΙΘΜΟΙ

Κατηγορίες δομών δεδομένων:

Στατικές	**Δυναμικές**
Το μέγεθός τους είναι καθορισμένο (σταθερό) και δεν μεταβάλλεται κατά την διάρκεια εκτέλεσης του προγράμματος.	Το μέγεθός τους δεν είναι καθορισμένο (δεν είναι σταθερό) και μεταβάλλεται κατά τη διάρκεια εκτέλεσης του προγράμματος. Μπορούν να εισάγονται νέοι κόμβοι και να διαγράφονται υπάρχοντες.
Οι κόμβοι αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη.	Οι κόμβοι δεν αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη.

Πίνακες:

Χαρακτηριστικός εκπρόσωπος των στατικών δομών. Περιέχει ένα σταθερό σύνολο κόμβων (θέσεις) που αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη. Επίσης, όλα τα στοιχεία είναι του ίδιου τύπου (δηλαδή ακέραιοι, πραγματικοί κτλ).

1) Μονοδιάστατοι πίνακες (μιας γραμμής ή μιας στήλης)

Π.χ.

Πίνακας ακεραίων Π[5]

10	-3	20	-12	43
1	2	3	4	5

Πίνακας αλφαριθμητικών Π[4]

“Νίκος”	“Γιάννης”	“Μαρία”	“Τάσος”
1	2	3	4

Οι αριθμοί κάτω από τις θέσεις του πίνακα δηλώνουν τον *αριθμό θέσης* του πίνακα. Ο αριθμός θέσης λέγεται και *δείκτης*. Ο πίνακας ορίζεται ως εξής: **τύπος όνομα_πίνακα [διαστάσεις]**. Π.χ. ακέραιος Π[5], δηλώνει ένα πίνακα μιας γραμμής πέντε θέσεων (στήλες) που περιέχει ακέραιους.

Για να αναφερθούμε στο περιεχόμενο μιας θέσης του πίνακα βάζουμε το **όνομα[θέση]**. Π.χ., Π[1] = το περιεχόμενο της θέσης 1, Π[6] = το περιεχόμενο της θέσης 6.

2) Δισδιάστατοι πίνακες (πολλών γραμμών και στηλών)

Εδώ έχουμε περισσότερες από μία γραμμές. Κάθε γραμμή έχει ένα σύνολο θέσεων (στήλες). Στο παρακάτω παράδειγμα ο πίνακας ορίζεται ως εξής: ακέραιος Π[3,4]. Η πρώτη διάσταση αναφέρεται στις γραμμές και η δεύτερη στις στήλες.
Γενικά, ορίζεται ως Π[Μ,Ν] για ένα πίνακα ΜxΝ.

	1	2	3	4
1	Π[1,1]	Π[1,2]	Π[1,3]	Π[1,4]
2	Π[2,1]	Π[2,1]	Π[2,3]	Π[2,4]
3	Π[3,1]	Π[3,2]	Π[3,3]	Π[3,4]

Για να αναφερθούμε στο *περιεχόμενο μιας θέσης* του δισδιάστατου πίνακα βάζουμε το ***όνομα[γραμμή,στήλη]***. Δηλαδή, η θέση προσδιορίζεται από τον αριθμό της γραμμής και της στήλης.
Π.χ., Π[1,2] = το περιεχόμενο της θέσης στη γραμμή 1 και στήλη 2,
Π[3,2] = το περιεχόμενο της θέσης στη γραμμή 3 και στήλη 2.

Τυπικές επεξεργασίες (λειτουργίες σε έναν πίνακα:

- **Διάβασμα** των στοιχείων του πίνακα, δηλαδή εισαγωγή τιμών στις θέσεις του,
- **Εκτύπωση** των στοιχείων του πίνακα,
- Υπολογισμός του **αθροίσματος** των στοιχείων του,
- Εύρεση του **ελάχιστου ή μέγιστου** στοιχείου του,
- **Αναζήτηση** ενός στοιχείου,
- **Ταξινόμηση** του πίνακα,
- **Συγχώνευση** δύο πινάκων.

Στους αλγόριθμους του πίνακα χρησιμοποιούμε ως επί το πλείστον, τη δομή **Για...από...μέχρι...**

Αλγόριθμοι μονοδιάστατου πίνακα:

1) Διάβασμα στοιχείων (δηλαδή εισαγωγή στοιχείων στις θέσεις του πίνακα)

```
Αλγόριθμος Διάβασμα_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Ν]          ! Ν θέσεις
        Ακέραιος: i             ! ο δείκτης θέσεις
Αρχή
        Για i από 1 μέχρι Ν
                Διάβασε Π[i]
        Τέλος_επανάληψης

Τέλος Διάβασμα_στοιχείων
```

2) Εκτύπωση στοιχείων

```
Αλγόριθμος Εκτύπωση_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Ν]          ! Ν θέσεις
        Ακέραιος: i             ! ο δείκτης θέσεις
Αρχή
        Για i από 1 μέχρι Ν
                Εκτύπωσε Π[i]
        Τέλος_επανάληψης

Τέλος Εκτύπωση_στοιχείων
```

3) Υπολογισμός αθροίσματος στοιχείων

```
Αλγόριθμος Άθροισμα_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Ν]          ! Ν θέσεις
        Ακέραιος: i             ! ο δείκτης θέσεις
        Ακέραιος: SUM
Αρχή
        SUM ← 0
        Για i από 1 μέχρι Ν
                SUM ← SUM + Π[i]
        Τέλος_επανάληψης

        Τύπωσε "Το άθροισμα είναι:", SUM
Τέλος Άθροισμα_στοιχείων
```

4) Υπολογισμός μέσου όρου (ΜΟ) στοιχείων

```
Αλγόριθμος ΜΟ_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Ν]          ! Ν θέσεις
        Ακέραιος: i             ! ο δείκτης θέσεις
        Ακέραιος: SUM
        Πραγματικός: ΜΟ
Αρχή
        SUM ← 0                 ! αρχικοποίηση πριν μπει στην επανάληψη
        Για i από 1 μέχρι Ν
                SUM ← SUM + Π[i]
        Τέλος_επανάληψης

        ΜΟ ← SUM/Ν
        Τύπωσε "Το άθροισμα είναι:", ΜΟ

Τέλος ΜΟ_στοιχείων
```

5) Εύρεση μέγιστου στοιχείου

Αλγόριθμος Max_στοιχείων
Μεταβλητές
Ακέραιος: Π[Ν] *! Ν θέσεις*
Ακέραιος: i *! ο δείκτης θέσεις*
Ακέραιος: Max
Αρχή
Max ← Π[1] *! θέτουμε σαν αρχική τιμή στο Max το πρώτο στοιχείο του ! πίνακα*
Για i **από** 2 **μέχρι** Ν
Αν Π[i] > Max **τότε** *! αν το i στοιχείο είναι μεγαλύτερο από την ! τρέχουσα τιμή του Max, τότε*
Max ← Π[i] *! βάλε στο Max το i στοιχείο*
Τέλος_αν
Τέλος_επανάληψης

Τύπωσε "Το μέγιστο στοιχείο του πίνακα είναι:", Max

Τέλος Max_στοιχείων

Η εύρεση του *ελαχίστου* είναι παρόμοια. Αλλάζει μόνο η ανισότητα, δηλαδή ελέγχουμε εάν Π[i] < Min.

6) Αναζήτηση στοιχείου

Εδώ θα δούμε την *σειριακή μέθοδο (γραμμική αναζήτηση)*. Η λογική είναι: Σαρώνουμε τον πίνακα και εξετάζουμε σε κάθε θέση αν το στοιχείο της θέσης αυτής είναι ίσο με αυτό που ψάχνουμε. Αν ναι, τότε σταματάμε.

Πότε χρησιμοποιούμε την μέθοδο αυτή?

i. Όταν ο πίνακας είναι μη ταξινομημένος
ii. Ο πίνακας είναι μικρού μεγέθους (π.χ. Ν <=20)
iii. Η αναζήτηση να πραγματοποιείται σπάνια, διότι η μέθοδος αυτή είναι σχετικά αργή.

Θα χρειαστούμε μια μεταβλητή **βρέθηκε** *λογικού τύπου*, που θα γίνει True αν το στοιχείο βρεθεί. Επίσης, δεν γνωρίζουμε σε ποια θέση βρίσκεται Μπορεί να είναι στην 1η θέση, αλλά μπορεί να είναι και στη Νστη θέση. Θα χρησιμοποιήσουμε και μια μεταβλητή **θέση**, που θα κρατήσει τη θέση όπου βρέθηκε.

Αλγόριθμος Σειριακή_αναζήτηση
Μεταβλητές
Ακέραιος: Π[Ν] *! Ν θέσεις*
Ακέραιος: i *! ο δείκτης θέσεις*

```
        Ακέραιος: key          ! αυτό που ψάχνουμε
        Ακέραιος: Θέση         ! ο αριθμός θέσης όπου βρίσκεται το στοιχείο
        Λογική: Βρέθηκε        ! θα γίνει True εάν βρεθεί το στοιχείο
Αρχή
        Βρέθηκε ← False
        Θέση ← 0
        i ← 1
        Όσο (i <= N) AND (Βρέθηκε = False) επανέλαβε
                Αν Π[i] = key τότε
                        Βρέθηκε ← True
                        Θέση ← i
                αλλιώς
                        i ← i + 1          ! προχώρα στην επόμενη θέση
                Τέλος_αν
        Τέλος_επανάληψης

        Αν Βρέθηκε = True τότε
                Τύπωσε "Το στοιχείο βρέθηκε στη θέση:", Θέση
        αλλιώς
                Τύπωσε "Το στοιχείο δεν βρέθηκε"
        Τέλος_αν

Τέλος Σειριακή_αναζήτηση
```

Σημείωση: *Εάν ο πίνακας είναι ταξινομημένος είναι προτιμότερο να χρησιμοποιείται μια πιο αποδοτική (γρήγορη) μέθοδος, που ονομάζεται* ***δυαδική μέθοδος****.*

> Ο αλγόριθμος δυαδικής αναζήτησης για ένα μονοδιάστατο ταξινομημένο πίνακα με αύξουσα σειρά έχει ως εξής:
> Συγκρίνουμε το μεσαίο στοιχείο του πίνακα με τη ζητούμενη τιμή. Αν είναι ίσα, επιστρέφουμε τη θέση του μεσαίου στοιχείου. Αν η ζητούμενη τιμή είναι μικρότερη, σημαίνει ότι βρίσκεται στο πρώτο μισό του πίνακα, αν υπάρχει. Αν η ζητούμενη τιμή είναι μεγαλύτερη, σημαίνει ότι βρίσκεται στο δεύτερο μισό του πίνακα, αν υπάρχει. Επομένως, με την πρώτη σύγκριση περιορίζουμε στο μισό τον χώρο αναζήτησης. Επαναλαμβάνουμε τη διαδικασία για το τμήμα του πίνακα που επιλέξαμε στο προηγούμενο βήμα έως ότου, με διαδοχικές διχοτομήσεις, περιοριστούμε σε ένα στοιχείο. Τότε, αν είναι ίσο με τη ζητούμενη τιμή επιστρέφουμε τη θέση του αλλιώς η ζητούμενη τιμή δεν περιέχεται στον πίνακα.

7) Ταξινόμηση πίνακα

Θα δούμε τη *μέθοδο της φυσαλίδας (bubble sort) ή ευθείας ανταλλαγής*. Βασίζεται στην αρχή της σύγκρισης γειτονικών στοιχείων του πίνακα και ανταλλαγής τους μέχρι να διαταχθούν όλα σε μία σειρά (αύξουσα ή φθίνουσα).

Η λογική είναι η εξής: Κάνουμε διαδοχικές σαρώσεις στον πίνακα. Σε κάθε σάρωση, το μικρότερο στοιχείο (για αύξουσα) μετακινείται προς την κορυφή του πίνακα. Αφού γίνουν όλες οι σαρώσεις, θα έχει επιτευχθεί η ταξινόμηση.

```
Αλγόριθμος Bubble_sort
Μεταβλητές
        Ακέραιος: Π[Ν]          ! Ν θέσεις
        Ακέραιος: i             ! για τις διαδοχικές σαρώσεις από την αρχή προς το τέλος
        Ακέραιος: j             ! για τη σάρωση που ελέγχει για το μικρότερο στοιχείο,
                                ! ξεκινά από το τέλος
        Ακέραιος: temp          ! βοηθητική μεταβλητή για την ανταλλαγή
Αρχή
        Για i από 2 μέχρι Ν         ! το i σαρώνει από την 2η θέση ως το τέλος
            Για j από Ν μέχρι i με_βήμα -1
                Αν Π[j -1] > Π[j] τότε
                    temp ← Π[j-1]
                    Π[j-1] ← Π[j]
                    Π[j] ← temp
                Τέλος_αν
            Τέλος_επανάληψης
        Τέλος_επανάληψης

Τέλος Bubble_sort
```

Τα γραμμοσκιασμένα βήματα, είναι εκείνα που κάνουν την αντιμετάθεση στοιχείων. Δηλαδή το j στοιχείο θα πάει στη θέση του j-1 και το j-1 στη θέση του j. Επειδή δεν μπορεί να γίνει άμεσα διότι θα χαθεί το στοιχείο j-1, προσωρινά τοποθετείται στη βοηθητική μεταβλητή temp.

Παράδειγμα: Να ταξινομηθούν οι ακέραιοι 12, 7, 5, 3, 10 ενός πίνακα. Πώς διαμορφώνεται ο πίνακας σε κάθε βήμα του αλγορίθμου?

Δείκτης		**Θέσεις Πίνακα**				
i	**j**	**1**	**2**	**3**	**4**	**5**
2	5	12	7	5	3	10
	4	12	7	3	5	10
	3	12	3	7	5	10
	2	**3**	12	7	5	10
3	5	3	12	7	5	10
	4	3	12	5	7	10
	3	3	**5**	12	7	10
4	5	3	5	12	7	10
	4	3	5	**7**	12	10
5	5	3	5	7	**10**	12

Ο παραπάνω αλγόριθμος με χρήση, αντί της μορφής **Για...από...μέχρι**, της μορφή **Όσο...επανάλαβε**, γίνεται:

```
Αλγόριθμος Bubble_sort
Μεταβλητές
        Ακέραιος: Π[Ν]
        Ακέραιος: i
        Ακέραιος: j
        Ακέραιος: temp
Αρχή
        i ← 2
        Όσο i <= N επανάλαβε
                j ← N
                Όσο j >= i επανάλαβε
                        Αν Π[j -1] > Π[j] τότε
                                temp ← Π[j-1]
                                Π[j-1] ← Π[j]
                                Π[j] ← temp
                        Τέλος_αν
                        j ← j -1
                Τέλος_επανάληψης
                i ← i + 1
        Τέλος_επανάληψης

Τέλος Bubble_sort
```

Αλγόριθμοι δισδιάστατου πίνακα:

1) Διάβασμα στοιχείων (δηλαδή εισαγωγή στοιχείων στις θέσεις του πίνακα)

```
Αλγόριθμος Διάβασμα_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Μ,Ν]      ! Μ γραμμές, Ν στήλες
        Ακέραιος: i           ! ο δείκτης γραμμής
        Ακέραιος: j           ! ο δείκτης στήλης
Αρχή
        Για i από 1 μέχρι Μ
                Για j από 1 μέχρι Ν
                        Διάβασε Π[i, j]
                Τέλος_επανάληψης
        Τέλος_επανάληψης

Τέλος Διάβασμα_στοιχείων
```

2) Εκτύπωση στοιχείων

```
Αλγόριθμος Εκτύπωση_στοιχείων
Μεταβλητές
        Ακέραιος: Π[Μ,Ν]      ! Μ γραμμές, Ν στήλες
        Ακέραιος: i           ! ο δείκτης γραμμής
        Ακέραιος: j           ! ο δείκτης στήλης
Αρχή
        Για i από 1 μέχρι Μ
                Για j από 1 μέχρι Ν
                        Τύπωσε Π[i, j]
                Τέλος_επανάληψης
        Τέλος_επανάληψης

Τέλος Εκτύπωση_στοιχείων
```

3) Υπολογισμός ΜΟ ανά γραμμή

```
Αλγόριθμος ΜΟ_ανά_γραμμή
Μεταβλητές
        Ακέραιος: Π[Μ,Ν]      ! Μ γραμμές, Ν στήλες
        Ακέραιος: i           ! ο δείκτης γραμμής
        Ακέραιος: j           ! ο δείκτης στήλης
        Ακέραιος: SUM
        Πραγματικός: ΜΟ
Αρχή
        Για i από 1 μέχρι Μ
                SUM ← 0       ! αρχικοποίηση πριν υπολογίσει για την νέα γραμμή
                Για j από 1 μέχρι Ν
                        SUM ← SUM + Π[i,j]
                Τέλος_επανάληψης
                ΜΟ ←SUM/N
                Τύπωσε "Ο Μέσος Όρος της γραμμής", i , "είναι: ", ΜΟ
        Τέλος_επανάληψης

Τέλος ΜΟ_ανά_γραμμή
```

4) **Υπολογισμός ΜΟ ανά γραμμή αλλά αποθήκευσή του σε ξεχωριστό πίνακα**

```
Αλγόριθμος ΜΟ_ανά_γραμμή_και_αποθήκευση_σε_άλλον_πίνακα
Μεταβλητές
        Ακέραιος: Π[Μ,Ν]      ! Μ γραμμές, Ν στήλες
        Ακέραιος: i           ! ο δείκτης γραμμής
        Ακέραιος: j           ! ο δείκτης στήλης
        Ακέραιος: SUM
        Πραγματικός: ΜΟ[Μ]    ! Ο πίνακας ΜΟ θα αποθηκεύει τους μέσους όρους ανά
                              ! γραμμή
Αρχή
        Για i από 1 μέχρι Μ
```

```
            SUM ← 0        ! αρχικοποίηση πριν υπολογίσει για την νέα γραμμή
            Για j από 1 μέχρι N
                  SUM ← SUM + Π[i,j]
            Τέλος_επανάληψης
      MO[i] ←SUM/N      ! εδώ αποθηκεύει στην Ι θέση του πίνακα ΜΟ τον μέσο όρο
      Τέλος_επανάληψης

      Για i από 1 μέχρι N
            Τύπωσε "Ο ΜΟ της γραμμής", i , "είναι :", MO[i]
      Τέλος_επανάληψης

Τέλος ΜΟ_ανά_γραμμή_και_αποθήκευση_σε_άλλον_πίνακα
```

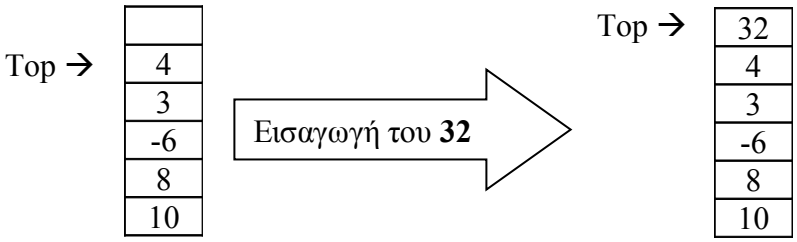
Στοίβα:

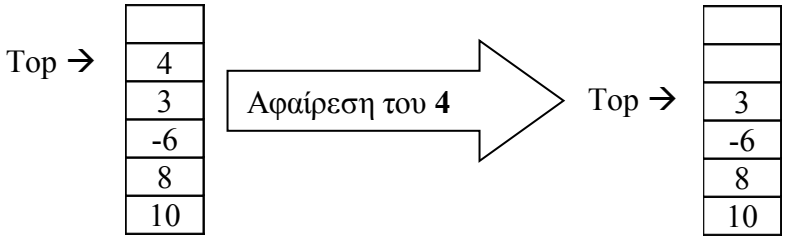
Η στοίβα υλοποιεί τη λογική **LIFO** (Last In First Out). Τα δεδομένα εισάγονται στην κορυφή της στοίβας, ενώ η αφαίρεση ενός στοιχείου γίνεται πάντα από την κορυφή της στοίβας. Η στατική στοίβα υλοποιείται με μονοδιάστατο πίνακα και χρησιμοποιεί ένα ***δείκτη Top*** που δείχνει στην κορυφή της στοίβας.

Οι λειτουργίες είναι δύο:

- Η ***εισαγωγή-ώθηση (Push)*** *ενός στοιχείου στην κορυφή της στοίβας.*
- Η ***αφαίρεση-απώθηση (Pop)*** ενός στοιχείου από την κορυφή της στοίβας.

Παράδειγμα:





Σημείωση:

Κατά της εισαγωγή ενός στοιχείου (ώθηση), πρέπει να γίνεται έλεγχος μήπως και η στοίβα είναι γεμάτη (δηλαδή δεν υπάρχει άλλη ελεύθερη θέση στον πίνακα). Ελέγχεται δηλαδή μήπως συμβεί ***υπερχείλιση (overflow)***.

Αντίστοιχα, κατά την αφαίρεση ενός στοιχείου (απώθηση), πρέπει να γίνεται έλεγχος μήπως η στοίβα είναι άδεια. Δηλαδή, ελέγχει μήπως συμβεί ***υποχείλιση (underflow)***.

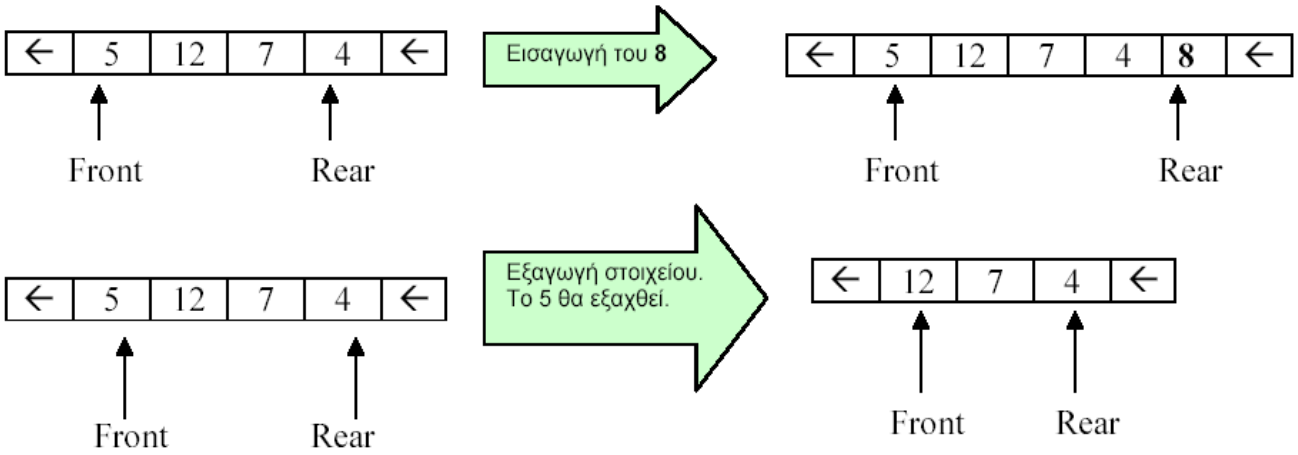
Ουρά:

Η ουρά υλοποιεί τη λογική **FIFO** (First In First Out). Τα δεδομένα εισάγονται στο πίσω μέρος της ουράς, ενώ η εξαγωγή ενός στοιχείου γίνεται πάντα από το μπροστινό μέρος της ουράς (όπως συμβαίνει σε μια ουρά σε τράπεζα!). Η στατική ουρά υλοποιείται με μονοδιάστατο πίνακα και χρησιμοποιεί δύο δείκτες:

i. Τον **Front** που δείχνει στο μπροστινό μέρος της ουράς και
ii. τον **Rear** που δείχνει στο πίσω μέρος της ουράς.

Οι λειτουργίες είναι δύο:

- Η ***εισαγωγή*** ενός στοιχείου στο πίσω μέρος της ουράς.
- Η ***εξαγωγή*** ενός στοιχείου από το μπροστινό μέρος της ουράς.



Σημείωση:

Κατά την εισαγωγή ενός στοιχείου, πρέπει να γίνεται έλεγχος μήπως η ουρά είναι γεμάτη (δηλαδή μήπως δεν υπάρχει άλλη ελεύθερη θέση στον πίνακα). Αντίστοιχα, κατά την εξαγωγή ενός στοιχείου, πρέπει να γίνεται έλεγχος μήπως η ουρά είναι άδεια.

Πλεονεκτήματα / Μειονεκτήματα των πινάκων:

Πλεονεκτήματα: Είναι ένας βολικός κι εύκολος τρόπος διαχείρισης δεδομένων του *ίδιου τύπου*.

Μειονεκτήματα:

- **Απαιτούν μνήμη**. Ο πίνακας δεσμεύει από την αρχή του προγράμματος αρκετές θέσης μνήμης. Έτσι, αν χρησιμοποιούμε πολλούς πίνακες σε ένα πρόγραμμα το επιβαρύνουμε όσον αφορά στη μνήμη.
- **Περιορίζουμε τις δυνατότητες του προγράμματος**. Επειδή ο πίνακας είναι *στατική δομή*, δεν μπορεί να αλλάξει το μέγεθός του κατά την εκτέλεση του προγράμματος.

Πότε είναι προτιμότερο να χρησιμοποιούμε τους πίνακας?

Όταν τα δεδομένα που εισάγουμε, απαιτείται να βρίσκονται στη μνήμη RAM καθ' όλη τη διάρκεια εκτέλεσης του προγράμματος.

ΚΕΦΑΛΑΙΟ 4 : Τεχνικές σχεδίασης αλγορίθμων

Σκοπός αυτού του κεφαλαίου είναι να παρουσιάσει κάποιες γενικές (πρότυπες) μεθοδολογίες ανάπτυξης-σχεδίασης αλγορίθμων και αν ένα συγκεκριμένο πρόβλημα μπορεί να επιλυθεί χρησιμοποιώντας μία από αυτές.

Ανάλυση προβλήματος

Η ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον, περιλαμβάνει τα εξής:

- Την *κατανόηση* του προβλήματος και τον *καθορισμό απαιτήσεων* (ποια δεδομένα παρέχονται και ποια είναι τα ζητούμενα, βλ. Κεφάλαιο 1)
- Διερεύνηση εάν το πρόβλημα έχει κάποιες *ιδιαιτερότητες*.
- Ποιες είναι οι *συνθήκες* και οι *προϋποθέσεις* για την επίλυσή του.
- Ποια είναι η πλέον *αποδοτική μέθοδος επίλυσής του και πως αυτή θα γραφτεί*. (π.χ. σχεδίαση με ψευδογλώσσα).
- Την *τελική επίλυση του προβλήματος σε ένα περιβάλλον Η/Υ*, χρησιμοποιώντας μία γλώσσα προγραμματισμού.

Οι λόγοι για τους οποίους οι μέθοδοι ανάλυσης και επίλυσης προβλημάτων παρουσιάζουν ενδιαφέρον είναι οι εξής:

- Παρέχουν έναν *τυποποιημένο τρόπο για την επίλυση προβλημάτων μεγάλης κλίμακας*.
- Μπορούν να *χρησιμοποιούν δομές δεδομένων και ελέγχου* που υποστηρίζονται από όλες τις σύγχρονες γλώσσες προγραμματισμού.
- Παρέχουν την δυνατότητα *να εκτιμήσουμε το «κόστος» της κάθε μεθόδου (χρονικές και χωρικές απαιτήσεις)*, δηλαδή το πόσο χρόνο θα απαιτήσει η μέθοδος για την επίλυση και το πόσους πόρους του Η/Υ θα χρειαστεί (π.χ. το μέγεθος της μνήμης). Με βάση το κόστος αυτό θα κρίνουμε εάν η μέθοδος είναι αποτελεσματική.

Φανταστείτε το πρόβλημα της μηχανοργάνωσης ενός οργανισμού. Πρόκειται για ένα μεγάλης κλίμακας πρόβλημα όπου εμπλέκονται πλήθος ανθρώπων διαφορετικών ειδικοτήτων και ποικίλων μηχανημάτων. Όλα αυτά πρέπει να συνδυαστούν και να λειτουργήσουν αρμονικά. Η επίλυση του προβλήματος θα ήταν πολύ δύσκολη αν δεν χρησιμοποιούσαμε κάποια συγκεκριμένη μεθοδολογία επίλυσης.

Μέθοδοι (τεχνικές) σχεδίασης αλγορίθμων

Μετά το πέρας της ανάλυσης του προβλήματος προκύπτουν κάποιες μέθοδοι επίλυσης που οδηγούν στη σχεδίαση του αλγορίθμου. Για λόγους διευκόλυνσης και τυποποίησης έχουμε ξεχωρίσει τρεις:

1) Μέθοδος *«Διαίρει και Βασίλευε»*
2) Μέθοδος *δυναμικού προγραμματισμού*
3) *Άπληστη μέθοδος*

Κάθε μέθοδος πρέπει να υποστηρίζει τα εξής:

- Να αντιμετωπίζει με τον δικό της τρόπο τα δεδομένα.
- Να έχει την δική της ακολουθία εντολών.

- Να έχει την δική της αποδοτικότητα.

Σε περιπτώσεις προβλημάτων που απαιτούν μία νέα και διαφορετική προσέγγιση στην επίλυσή τους, χρησιμοποιούμε *ευριστικές μεθόδους*.

ΚΕΦΑΛΑΙΟ 6 : Εισαγωγή στον προγραμματισμό

Ο **προγραμματισμός** είναι η διατύπωση του αλγορίθμου σε μορφή κατανοητή από τον Η/Υ, ώστε να τον εκτελέσει (*«τρέξει»*). Η *διατύπωση* γίνεται χρησιμοποιώντας μια γλώσσα προγραμματισμού.

Κατηγορίες γλωσσών προγραμματισμού:

Γλώσσα μηχανής	Το πρόγραμμα περιέχει εντολές που είναι σε δυαδική μορφή, άμεσα κατανοητή από τον Η/Υ (όχι όμως από τον άνθρωπο). Δηλαδή, το πρόγραμμα αποτελείται από ακολουθίες 0 και 1 π.χ. 10101000 00001010 11000000 00000001 .... **Πλεονεκτήματα:** ▪ Ταχύτατη εκτέλεση εντολών ▪ Δεν απαιτείται μεταφραστικό πρόγραμμα **Μειονεκτήματα:** ▪ Το γράψιμο του προγράμματος είναι μια ιδιαίτερα επίπονη και χρονοβόρα διαδικασία ▪ Απαιτείται βαθιά γνώση της αρχιτεκτονικής του Η/Υ. ▪ Το πρόγραμμα «τρέχει» μόνο στο συγκεκριμένο τύπο του Η/Υ.
Γλώσσες χαμηλού επιπέδου ή συμβολικές γλώσσες	Οι εντολές που είναι σε μορφή 0 και 1, αντικαθίστανται από μνημονικά (συμβολικά) ονόματα. Για παράδειγμα η εντολή 100001100 αντικαθίσταται από το ADD. Ένα παράδειγμα χρήσης θα ήταν: INDEX = $01 (Βάλε στην INDEX την τιμή 1) ADD INDEX (Πρόσθεση την τιμή της INDEX στον συσσωρευτή) LDA #10 (φόρτωσε στον συσσωρευτή την τιμή 10) CLA (καθάρισε τον συσσωρευτή) ..... **Πλεονεκτήματα:** ▪ Ταχύτατη εκτέλεση των εντολών ▪ Η μορφή του προγράμματος είναι καλύτερα κατανοητή από τον άνθρωπο σε σχέση με την γλώσσα μηχανής. **Μειονεκτήματα:** ▪ Η αντιστοιχία ένα προς ένα με τις εντολές της γλώσσας μηχανής παρέμεινε. ▪ Απαιτείται η χρήση ενός μεταφραστικού προγράμματος ώστε οι συμβολικές εντολές να μετατραπούν στις αντίστοιχες δυαδικές. Το ειδικό αυτό πρόγραμμα ονομάζεται *συμβολομεταφραστής (assembler)*. ▪ Το γράψιμο του προγράμματος εξακολουθεί να είναι μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία. ▪ Απαιτείται βαθιά γνώση της αρχιτεκτονικής του Η/Υ. ▪ Το πρόγραμμα «τρέχει» μόνο στο συγκεκριμένο τύπο του Η/Υ.
Γλώσσες υψηλού επιπέδου	Λέγονται έτσι, διότι τα προγράμματα διατυπωμένα σε μία τέτοια γλώσσα, είναι άμεσα κατανοητά από τον άνθρωπο (αλλά όχι από τον Η/Υ) αφού χρησιμοποιείται μια γλώσσα που είναι αρκετά περιγραφική, όπως μια φυσική γλώσσα. Π.χ.

	INPUT "Δώσε την τελική τιμή:"; N SUM = 0 For INDEX = 1 to N SUM = SUM + INDEX Next **Πλεονεκτήματα:** ▪ Η μορφή του προγράμματος είναι εύκολα κατανοητή από τον άνθρωπο σε σχέση με τη γλώσσα μηχανής ή τη συμβολική γλώσσα. ▪ Το γράψιμο του προγράμματος δεν είναι πλέον μια ιδιαίτερα επίπονη και χρονοβόρα διαδικασία όπως συμβαίνει με τη γλώσσα μηχανής ή τη συμβολική γλώσσα. ▪ Δεν απαιτεί σχεδόν καμιά γνώση της αρχιτεκτονικής του Η/Υ. Συνεπώς τα προγράμματα είναι ανεξάρτητα από την αρχιτεκτονική του Η/Υ. ▪ Το πρόγραμμα «τρέχει» σε όλους τους τύπους Η/Υ, αρκεί να υπάρχει το κατάλληλο μεταφραστικό πρόγραμμα. Συνεπώς, ένα χαρακτηριστικό τους είναι η *μεταφερσιμότητα*, δηλαδή ένα πρόγραμμα υψηλού επιπέδου να εκτελείται με ελάχιστες μετατροπές, σε πολλούς τύπους Η/Υ . ▪ Η εκμάθηση της γλώσσας είναι εύκολη. ▪ Η διόρθωση λαθών και η συντήρηση των προγραμμάτων είναι ευκολότερη. **Μειονεκτήματα:** ▪ Απαιτείται η χρήση ενός μεταφραστικού προγράμματος, ώστε οι εντολές να μετατραπούν σε πολλές δυαδικές εντολές (δεν υπάρχει εδώ αντιστοιχία *ένα προς ένα*). Έχουμε δύο ειδών *μεταφραστικά προγράμματα*: - Τους *μεταγλωττιστές (compilers)* και τους *διερμηνείς (interpreters)*. ▪ Το πρόγραμμα τρέχει πιο αργά σε σχέση με τα προγράμματα των συμβολικών γλωσσών ή της γλώσσας μηχανής.

Κατηγοριοποίηση γλωσσών υψηλού επιπέδου:

- **Διαδικασιακές ή αλγοριθμικές γλώσσες (Procedural):** Λέγονται έτσι διότι επιτρέπουν την εύκολη υλοποίηση αλγορίθμων π.χ. Pascal, Basic.
- **Αντικειμενοστραφείς γλώσσες (object – oriented)** π.χ. C++
- **Συναρτησιακές γλώσσες (functional)** π.χ. LISP
- **Μη-διαδικασιακές γλώσσες** π.χ. PROLOG
- **Γλώσσες ερωταπαντήσεων (Query languages) ή 4ης γενιάς** π.χ. SQL.

Μία άλλη κατηγοριοποίηση των γλωσσών υψηλού επιπέδου είναι:

- Γλώσσες **γενικής χρήσης**: Σκοπός τους είναι να επιλύουν πάσης φύσεων προβλήματα (αριθμητικά, εμπορικά, επιστημονικά). Τέτοιες είναι η Basic, Pascal. Μερικές γλώσσες όμως έχουν δημιουργηθεί αποκλειστικά για να επιλύουν ευκολότερα συγκεκριμένους τύπους προβλημάτων όπως:

- Γλώσσες **επιστημονικής κατεύθυνσης,** π.χ. FORTRAN
- Γλώσσες **εμπορικής κατεύθυνσης,** π.χ. COBOL
- Γλώσσες **προγραμματισμού συστημάτων,** π.χ. C
- Γλώσσες **τεχνητής νοημοσύνης,** π.χ. LISP, PROLOG
- Γλώσσες **ειδικής χρήσης.** Σκοπός τους είναι να επιλύουν ειδικού τύπου προβλήματα, όπως διαχείριση Βάσεων Δεδομένων κ.α., π.χ. SQL.

Διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης:

FORTRAN (1957)	Κατάλληλη για επίλυση επιστημονικών προβλημάτων (αριθμητικές εφαρμογές.
COBOL (1960)	Κατάλληλη για επίλυση εμπορικών προγραμμάτων (εφαρμογές μισθοδοσία, λογιστικές κτλ)
ALGOL (1960)	Δημιουργήθηκε με σκοπό την ανάπτυξη προγραμμάτων κυρίως για επιστημονικές εφαρμογές ως ανταγωνιστική της Fortran. Δεν χρησιμοποιήθηκε όμως ευρέως στην πράξη.
PL/1 (μέσα ’60)	Προσπάθησε να συνδυάσει τις δυνατότητες των γλωσσών που ήταν προσανατολισμένες για εμπορικές και επιστημονικές εφαρμογές, χωρίς όμως να γνωρίσει επιτυχία.
BASIC (μέσα ’60)	Δημιουργήθηκε με σκοπό την εκπαίδευση των αρχάριων στον προγραμματισμό. Σκοπός της BASIC είναι να γράφονται μικρά προγράμματα που κατόπιν εκτελούνται με τη βοήθεια διερμηνέα. Στις μέρες μας αποτελεί μια πανίσχυρη, γενικής χρήσης γλώσσα.
PASCAL (1970)	Γλώσσα γενικής χρήσης. Στηρίχθηκε στην ALGOL. Είναι η καταλληλότερη γλώσσα για να μάθει κανείς δομημένο προγραμματισμό.
C (αρχές ’70)	Περιέχει αρκετά κοινά χαρακτηριστικά με την PASCAL για την ανάπτυξη δομημένων εφαρμογών, αλλά παράλληλα ενσωματώνει και χαρακτηριστικά γλώσσας χαμηλού επιπέδου. Θεωρείται κατάλληλη για την ανάπτυξη λειτουργικών συστημάτων (π.χ. Unix).
Ada (1979)	Γλώσσα γενικής χρήσης που δίνει έμφαση στο θέμα της αξιοπιστίας των προγραμμάτων. Για αυτό τον λόγο χρησιμοποιείται πρωτίστως για στρατιωτικές εφαρμογές.

Αντικειμενοστραφείς γλώσσες κατά χρονολογική σειρά:

Smalltalk (αρχές ’80)	Η πρώτη αντικειμενοστραφής γλώσσα με ολοκληρωμένο μάλιστα περιβάλλον ανάπτυξης προγραμμάτων.
C++ (τέλη ’80)	Αποτελεί μια εξέλιξη της C στο χώρο του αντικειμενοστραφούς προγραμματισμού και χρησιμοποιείται αρκετά στην ανάπτυξη λειτουργικών προγραμμάτων (π.χ. Windows) αλλά και άλλου τύπου εφαρμογών. Θεωρείται σήμερα μια κορυφαία γλώσσα.
JAVA (μέσα ’90)	Γλώσσα ειδικά σχεδιασμένη για την ανάπτυξη εφαρμογών στο Internet. Σκοπός της είναι να γράφονται προγράμματα που θα εκτελούνται, χωρίς μετατροπές σε Η/Υ με διαφορετικά λειτουργικά συστήματα. Περιέχει αρκετά στοιχεία από την C++.
C# (2002)	Δημιουργήθηκε ως ανταγωνιστής της JAVA. Σκοπός της είναι να συνδυάσει την ευχρηστία της BASIC και τη δυναμική της C++ για την

	ανάπτυξη εφαρμογών που θα εκτελούνται σε Η/Υ με διαφορετικά λειτουργικά συστήματα.

Συναρτησιακές γλώσσες κατά χρονολογική σειρά εμφάνισης:

LISP (μέσα ’60)	Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης.

Μη διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης:

PROLOG (αρχές ’70)	Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης.

Γλώσσες 4ης γενιάς ή ερωταπαντήσεων:

SQL	Δεν απευθύνεται μόνο σε προγραμματιστές αλλά και σε χρήστες. Ο χρήστης μπορεί σχετικά εύκολα, να υποβάλλει ερωτήσεις στο σύστημα ή να αναζητά πληροφορίες από μία Βάση Δεδομένων.

Από τι εξαρτάται η επιλογή μιας γλώσσας προγραμματισμού:

- Από το είδος του προβλήματος (εμπορικό, αριθμητικό κ.α.)
- Από τον Η/Υ στον οποίο θα εκτελεστεί το πρόγραμμα.
- Από τη διαθέσιμη γλώσσα ή προγραμματιστικό περιβάλλον, στο οποίο θα αναπτυχθεί το πρόγραμμα.
- Από τις γνώσεις και την εμπειρία του προγραμματιστή.

Με την ευρεία διάδοση των γραφικών περιβαλλόντων επικοινωνίας (π.χ. Windows, MacOS κ.α.) δημιουργήθηκαν παραλλαγές κάποιων γλωσσών που απευθύνονται σε αυτά. Τέτοιες είναι οι Visual Basic, Visual C++, Delphi (Visual Pascal), C# κ.α. Αυτές οι γλώσσες ακολουθούν την φιλοσοφία του *οπτικού* και του *καθοδηγούμενου από γεγονότα* προγραμματισμού, χωρίς να απορρίπτουν τις αρχές του δομημένου προγραμματισμού.

Φυσικές και Τεχνητές Γλώσσες:

Οι γλώσσες προγραμματισμού είναι τεχνητές γλώσσες που απευθύνονται σε ανθρώπους που επιθυμούν να επικοινωνήσουν με τον Η/Υ.

Κάθε γλώσσα προγραμματισμού προσδιορίζεται από:

- Το αλφάβητό της
- Το λεξιλόγιό της
- Τη γραμματική της
- Τη σημασιολογία της (Semantics)

Αλφάβητο Γλώσσας:

Ως αλφάβητο ορίζουμε το σύνολο των αποδεκτών χαρακτήρων της γλώσσας. Από τους χαρακτήρες αυτούς σχηματίζονται οι λέξεις τις γλώσσας.

Σε μία φυσική γλώσσα όπως τα Ελληνικά, η λέξη ΔΙΑΒΑΖΩ είναι αποδεκτή, ενώ η λέξη ΖΩΒΑΓΩ όχι.

Γραμματική της Γλώσσας:

Η γραμματική περιλαμβάνει το **τυπολογικό** και το **συντακτικό**.

Το **τυπολογικό** ορίζει τους κανόνες σύμφωνα με του οποίους μία λέξη θα είναι αποδεκτή. Για παράδειγμα, στην Ελληνική γλώσσα για την λέξη «δίνω», αποδεκτές μορφές είναι και το «δίνεις», «δίνουν», αλλά όχι το «δίνουτ».

Το **συντακτικό** είναι ένα σύνολο κανόνων που ορίζει το πως πρέπει να σχηματίζονται οι προτάσεις από τις λέξεις της γλώσσας, ώστε οι προτάσεις αυτές να είναι έγκυρες και αποδεκτές. Σε μία γλώσσα προγραμματισμού αυτό που ενδιαφέρει είναι η σωστή σύνταξη των εντολών.

Σημασιολογία της Γλώσσας:

Είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και προτάσεων της γλώσσας. Σε μία γλώσσα προγραμματισμού αυτό καθορίζεται από τον δημιουργό της, ενώ σε μία φυσική γλώσσα από αυτόν που εκφέρει την πρόταση.

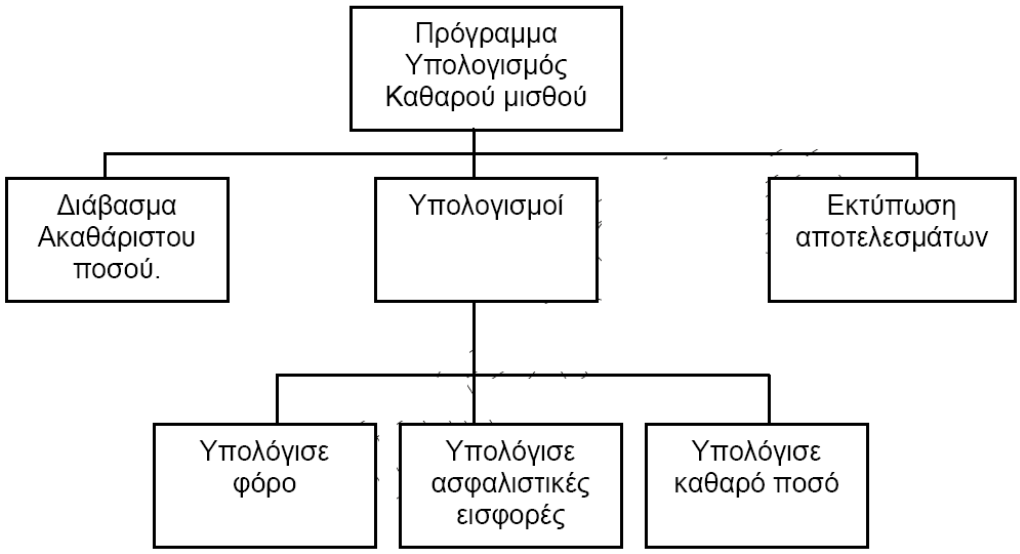
Διαφορές μεταξύ Φυσικών και Τεχνητών Γλωσσών:

Φυσικές	**Τεχνητές**
Χρησιμοποιούνται για την επικοινωνία μεταξύ ανθρώπων.	Χρησιμοποιούνται για την επικοινωνία μεταξύ ανθρώπου και Η/Υ.
Έχουν μεγάλες δυνατότητες εξέλιξης. Νέες λέξεις μπορεί να εισαχθούν, κανόνες γραμματικής και σύνταξης να αλλάξουν κλπ.	Οι δυνατότητες εξέλιξης είναι περιορισμένες. Τις περισσότερες φορές, η εξέλιξη αυτή αφορά την επέκταση του ρεπερτορίου των εντολών της γλώσσας (π.χ. Basic και Visual Basic)

Τεχνικές Σχεδίασης Προγραμμάτων:

Δεν αρκεί κάποιος να γνωρίζει μια γλώσσα προγραμματισμού. Θα πρέπει να ακολουθήσει και μία τεχνική σχεδίασης του προγράμματός του.
Για την σύνταξη σωστών, κατανοητών και εύκολα συντηρήσιμων προγραμμάτων ακολουθήθηκαν διάφορες μεθοδολογίες ανάπτυξης που παρουσιάζονται παρακάτω:

1) **Ιεραρχική σχεδίαση ή Top-down σχεδίαση:** Η τεχνική αυτή συνιστά στον καθορισμό βασικών λειτουργιών του προγράμματος σε ανώτερο επίπεδο και στη συνέχεια τη διάσπαση καθεμιάς σε μικρότερες και απλούστερες μέχρι του σημείου να είναι τόσο απλές που μπορούν να επιλυθούν άμεσα. **Σκοπός της ιεραρχικής σχεδίασης** είναι η διάσπαση του προβλήματος σε μικρότερα και απλούστερα υποπροβλήματα, τα οποία είναι ευκολότερο να επιλυθούν. Για την Top-down σχεδίαση χρησιμοποιούμε το *ιεραρχικό διάγραμμα*. **Παράδειγμα: Πρόγραμμα Υπολογισμός Καθαρού Μισθού:**



2) **Τμηματικός Προγραμματισμός:** Η ιεραρχική σχεδίαση πραγματοποιείται με τον τμηματικό προγραμματισμό. Μετά την ανάλυση του προβλήματος σε μικρότερα και απλούστερα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί μία ξεχωριστεί και ανεξάρτητη **ενότητα** (module). Τώρα, για κάθε ενότητα θα γραφεί το κατάλληλο πρόγραμμα ή τμήμα προγράμματος.

3) **Δομημένος Προγραμματισμός:** Είναι η μεθοδολογία που έχει επικρατήσει σήμερα. Εμπεριέχει τις αρχές τις ιεραρχικής σχεδίασης και του τμηματικού προγραμματισμού. Επιπλέον, αναφέρει ότι για την δημιουργία σωστών προγραμμάτων να χρησιμοποιούμε μόνο τις 3 στοιχειώδεις δομές: Ακολουθίας, Επιλογής και Επανάληψης. Όλα τα προγράμματα οποιουδήποτε μεγέθους, μπορούν να γραφτούν στηριζόμενα στη χρήση μόνο αυτών των δομών, αποφεύγοντας πλήρως την χρήση της GOTO. Επίσης, κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Πλεονεκτήματα του Δομημένου Προγραμματισμού:

- Άμεση υλοποίηση των αλγορίθμων σε πρόγραμμα.
- Διευκόλυνση της ανάλυσης του προγράμματος σε τμήματα (ενότητες). Το κάθε τμήμα μπορεί να γραφεί από διαφορετικές ομάδες προγραμματιστών.
- Ευκολότερη και συντομότερη ανάπτυξη προγραμμάτων.
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση του προγράμματος.

Αντικειμενοστραφής Προγραμματισμός:

Η φιλοσοφία του αντικειμενοστραφούς προγραμματισμού είναι η εξής: *Να θεωρήσουμε τα δεδομένα και τις αποδεκτές ενέργειες που γίνονται πάνω σε αυτά ως ένα ενιαίο αντικείμενο (object)*. Αυτό το αντικείμενο θα μπορεί να χρησιμοποιηθεί πολύ εύκολα οπουδήποτε αλλού, δηλαδή θα είναι *επαναχρησιμοποιήσιμο*.
Για παράδειγμα, θα μπορούσαμε να θεωρήσουμε μια στοίβα ως αντικείμενο: Ένας πίνακας Π[20] που θα δέχεται ακέραιους. Οι μόνες ενέργειες που θα ήταν επιτρεπτές θα ήταν η ώθηση ενός στοιχείου στη στοίβα και η αφαίρεση ενός στοιχείου από τη στοίβα. Αυτό το αντικείμενο θα μπορούσαμε να το δώσουμε και σε όποιον άλλο το έχει ανάγκη.
Ο αντικειμενοστραφής προγραμματισμός, ακολουθεί τις αρχές του δομημένου προγραμματισμού.

Οπτικός Προγραμματισμός:

Είναι η δυνατότητα να δημιουργούμε με γραφικό τρόπο, ολόκληρο το περιβάλλον της εφαρμογής, όπως για παράδειγμα τα μενού και τα πλαίσια διαλόγου και άλλα παράθυρα της εφαρμογής. Ο οπτικός προγραμματισμός εκμεταλλεύεται τις δυνατότητες των γραφικών περιβαλλόντων επικοινωνίας (π.χ. Windows κλπ).

Καθοδηγούμενος από γεγονότα Προγραμματισμός:

Είναι η δυνατότητα να εκτελούνται οι διάφορες λειτουργίες του προγράμματος με την ενεργοποίηση ενός γεγονότος. Για παράδειγμα, αν κάνουμε κλικ σε κάποια εντολή ενός μενού ή σε κάποιο κουμπί σε ένα παράθυρο της εφαρμογής, τότε θα εκτελεστεί μία λειτουργία.
Τα σύγχρονα προγραμματιστικά περιβάλλοντα είναι κτισμένα πάνω στις αρχές του οπτικού και καθοδηγούμενου από γεγονότα προγραμματισμού.

Παράλληλος Προγραμματισμός:

Σήμερα υπάρχουν μεγάλοι Η/Υ που διαθέτουν στο εσωτερικό τους πολλούς επεξεργαστές. Οι επεξεργαστές αυτοί μοιράζονται την ίδια μνήμη και λειτουργούν παράλληλα. Έτσι, την ίδια χρονική στιγμή, μπορούν να εκτελούνται διαφορετικές εντολές του ίδιου προγράμματος.
Για να εκμεταλλευτούμε αυτή την ιδιαίτερη ισχύ των Η/Υ, θα πρέπει τα προγράμματα να είναι φτιαγμένα με τέτοιο τρόπο, ώστε διαφορετικά τμήματά του να εκτελούνται παράλληλα. Για τον σκοπό αυτό έχουν αναπτυχθεί ιδιαίτερες γλώσσες προγραμματισμού.

Προγραμματιστικά Περιβάλλοντα:

Ένα πρόγραμμα που φτιάχνεται σε μία γλώσσα προγραμματισμού υψηλού επιπέδου, δεν είναι άμεσα κατανοητό από τον Η/Υ. Θα πρέπει να μεταφραστεί σε ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (δυαδική μορφή). Την διαδικασία μετάφρασης την πραγματοποιούν τα μεταφραστικά προγράμματα. Είναι δύο ειδών:

- *Μεταγλωττιστές (Compilers)*
- *Διερμηνευτές (Interpreters)*

Πηγαίο πρόγραμμα (source program):

Το πρόγραμμα που είναι φτιαγμένο σε γλώσσα υψηλού επιπέδου.

Αντικείμενο πρόγραμμα (object program:

Το πρόγραμμα που είναι μεταφρασμένο σε γλώσσα μηχανής, αλλά όχι άμεσα εκτελέσιμο.

Εκτελέσιμο πρόγραμμα (Executable program):

Το πρόγραμμα σε γλώσσα μηχανής που είναι έτοιμο πλέον να εκτελεσθεί.

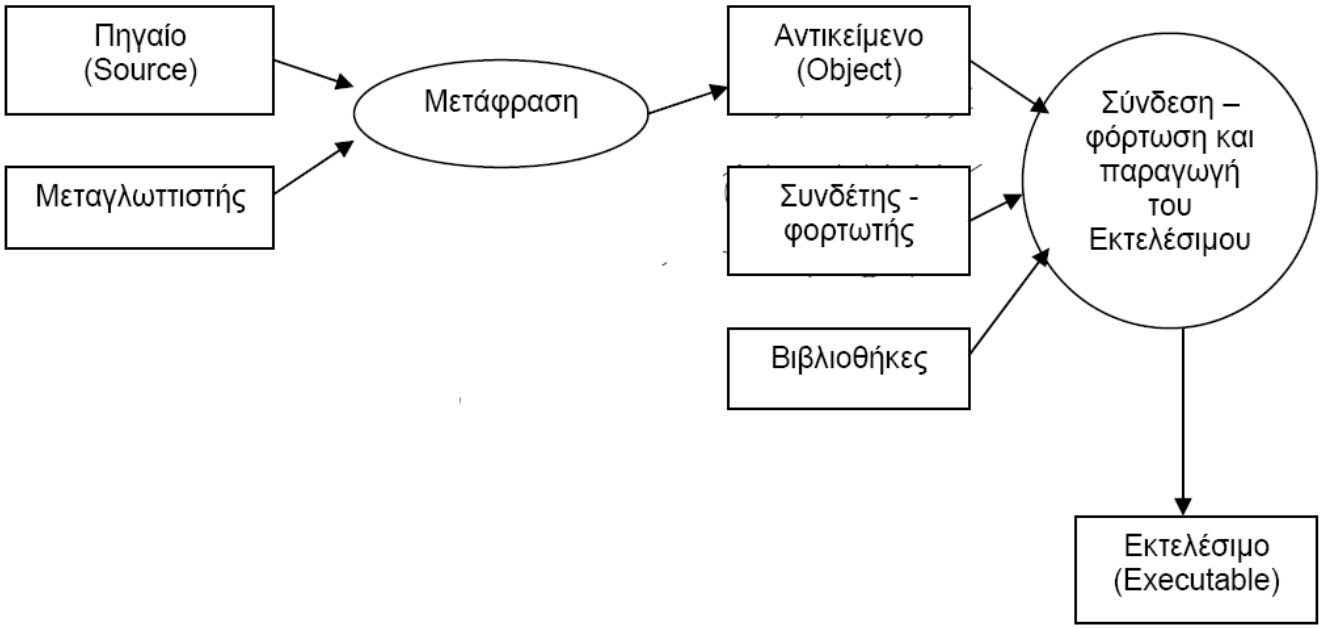
Μεταγλωττιστής:

Παίρνει ως είσοδο το πηγαίο πρόγραμμα και αναλαμβάνει να το μεταφράσει εξ' ολοκλήρου παράγοντας το αντικείμενο πρόγραμμα.

Διερμηνευτής:

Παίρνει ως είσοδο, μία-μία εντολή του πηγαίου προγράμματος, την μεταφράζει και την εκτελεί αμέσως. Η λειτουργία του μοιάζει με τον άνθρωπο-διερμηνέα που μεταφράζει επί τόπου κάθε πρόταση.

Η διαδικασία μεταγλώττισης και σύνδεσης ενός προγράμματος, είναι η εξής:



Βιβλιοθήκες:

Έτοιμες ενότητες (modules) αντικείμενου προγράμματος της γλώσσας, απαραίτητες για την παραγωγή του εκτελέσιμου προγράμματος.

Συνδέτης-Φορτωτής (linker-loader):

Ειδικό πρόγραμμα που αναλαμβάνει να συνδέσει το αντικείμενο πρόγραμμα με τις βιβλιοθήκες και να παράγει το εκτελέσιμο πρόγραμμα.

Κατά την δημιουργία ενός προγράμματος σχεδόν πάντα ενυπάρχουν **λάθη**. Τα λάθη τα χωρίζουμε σε δύο κατηγορίες:

- *Συντακτικά λάθη*
- *Λογικά λάθη*

Τα **συντακτικά** λάθη, ανιχνεύονται κατά την διάρκεια της μεταγλώττισης ή διερμήνευσης. Αφορούν σε παραβιάσεις του τυπολογικού και συντακτικού της γλώσσας (π.χ. μία εντολή έχει γραφτεί συντακτικά λάθος). Στην περίπτωση αυτή ο προγραμματιστής πρέπει να επιστρέψει στο πηγαίο πρόγραμμα, να διορθώσει τα λάθη και να ξαναϋποβάλλει το πηγαίο πρόγραμμα σε μεταγλώττιση.

Τα **λογικά** λάθη, είναι τα πλέον δύσκολα στην ανίχνευσή τους. Έχουν να κάνουν με σφάλματα στη λογική επίλυσης του προβλήματος ή λανθασμένης διατύπωσης του αλγορίθμου (π.χ. το πρόγραμμα παράγει άλλα αποτελέσματα και όχι τα ζητούμενα!).

Για την σύνταξη των προγραμμάτων, χρησιμοποιούμαι ένα ειδικό πρόγραμμα, που ονομάζεται **συντάκτης (editor)**. Μοιάζει με επεξεργαστή κειμένου με επιπλέον δυνατότητες που διευκολύνουν την γρήγορη σύνταξη των πηγαίων προγραμμάτων.

Τα σύγχρονα προγραμματιστικά περιβάλλοντα, έχουν υποχρεωτικώς τριών ειδών προγράμματα:

1) Τον συντάκτη
2) Το μεταγλωττιστή, και
3) Το συνδέτη – φορτωτή.

Πέραν αυτών, περιέχουν όλα εκείνα τα εργαλεία που διευκολύνουν την εύκολη, ταχύτατη σύνταξη, διόρθωση και συντήρηση των προγραμμάτων γι’ αυτό και πολλές φορές ονομάζονται RAD περιβάλλοντα (Rapid Application Development).
Όλα αυτά παρέχονται έναν ενιαίο και λειτουργικό τρόπο στον προγραμματιστή.

ΚΕΦΑΛΑΙΟ 10 : Υποπρογράμματα

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

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

Χαρακτηριστικά (ιδιότητες) υποπρογραμμάτων:

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 έλεγχος του. Αν εκτελεί περισσότερες λειτουργίες τότε μάλλον πρέπει να διασπαστεί σε ακόμα μικρότερα υποπρογράμματα.

Πλεονεκτήματα του τμηματικού προγραμματισμού:

- **Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντίστοιχου προγράμματος.** Αντί να προσπαθούμε να επιλύσουμε μεμιάς το συνολικό πρόβλημα είναι προτιμότερο να το σπάσουμε σε απλούστερα υπό-προβλήματα για τα οποία θα γράψουμε αντίστοιχους αλγορίθμους 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 μια λειτουργία δεν υποστηρίζεται απευθείας από την 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 μπορούνε να καλούνε αυτό το υποπρόγραμμα.

Συγγράφοντας πολλά υποπρογράμματα, μπορούμε να δημιουργήσουμε ολόκληρες *βιβλιοθήκες (libraries)* και ουσιαστικά να επεκτείνουμε την ίδια την γλώσσα προγραμματισμού για λειτουργίες που δεν τις υποστηρίζει απευθείας.

Παράμετροι:

Τα υποπρογράμματα, ενεργοποιούνται από κάποιο άλλο πρόγραμμα (π.χ. το κύριο πρόγραμμα) ή υποπρόγραμμα για να εκτελέσουνε την λειτουργία τους.
Η επικοινωνία μεταξύ του υποπρογράμματος και αυτού που το καλεί, γίνεται διαμέσου κάποιων **ειδικών μεταβλητών** που ονομάζονται **παράμετροι**. Το καλούν πρόγραμμα (π.χ. κύριο πρόγραμμα) «περνάει» (μεταβιβάζει) τιμές στο υποπρόγραμμα μέσω των παραμέτρων του. Ενδεχομένως, να επιστρέφει και τιμές στο καλούν πρόγραμμα (π.χ. κύριο πρόγραμμα), μέσω των παραμέτρων.

Παράμετρος είναι μία μεταβλητή, η οποία επιτρέπει το πέρασμα τιμής από ένα τμήμα προγράμματος σε ένα άλλο.

Είδη υποπρογραμμάτων:

- **Διαδικασίες**: Μια 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**: Μια συνάρτηση είναι ένας τύπος υποπρογράμματος, που **υπολογίζει ΜΙΑ μόνο τιμή** και την επιστρέφει στο καλούν πρόγραμμα, μέσω του ονόματός της.

Ορισμός και κλήση συναρτήσεων:

ΣΥΝΑΡΤΗΣΗ όνομα (λίστα παραμέτρων)**:** τύπος συνάρτησης
Τμήμα δηλώσεων μεταβλητών
ΑΡΧΗ
....
Εντολές 1
Εντολές 2
....
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

Ορισμός και κλήση διαδικασιών:

ΔΙΑΔΙΚΑΣΙΑ όνομα (λίστα παραμέτρων)
Τμήμα δηλώσεων μεταβλητών
ΑΡΧΗ
....
Εντολές 1
Εντολές 2
....
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ

Παρατήρηση:

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

Παράδειγμα ορισμού ΣΥΝΑΡΤΗΣΗΣ και πως καλείται:

Να γραφτεί μια συνάρτηση που υπολογίζει το εμβαδόν ενός τριγώνου, όταν γνωρίζουμε την βάση και το ύψος.
Επειδή η λειτουργία που θα εκτελεί είναι ένας υπολογισμός και συνεπώς θα επιστρέφει ΜΙΑ τιμή, είναι καταλληλότερο να χρησιμοποιηθεί συνάρτηση. Η τιμή που επιστρέφει είναι ένας πραγματικός αριθμός.

ΣΥΝΑΡΤΗΣΗ Εμβαδόν_Τριγώνου (βάση, ύψος) **:** πραγματική
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: βάση, ύψος
ΑΡΧΗ
    Εμβαδόν_Τριγώνου <- (βάση * ύψος)/2
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

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

ΠΡΟΓΡΑΜΜΑ Υπολογισμος_εμβαδών_διαφόρων_τριγώνων
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: Β, Υ
    ΠΡΑΓΜΑΤΙΚΕΣ: Ε
ΑΡΧΗ

Β <- 12
Υ <- 4
Ε <- Εμβαδόν_Τριγώνου(Β, Υ)
ΤΥΠΩΣΕ “Το εμβαδόν είναι:”, Ε

Β <- 10
Υ <- 5
Ε <- Εμβαδόν_Τριγώνου(Β, Υ)
ΤΥΠΩΣΕ “Το εμβαδόν είναι:”, Ε

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

Παρατήρηση:

Η συνάρτηση καλείται μέσω του ονόματός της.

Παράδειγμα ορισμού ΔΙΑΔΙΚΑΣΙΑΣ και πως καλείται:

Θα χρησιμοποιήσουμε το ίδιο παράδειγμα, για να φανεί η διαφορά στην υλοποίηση με μία διαδικασία. Θα φτιάξουμε μια διαδικασία, η οποία υπολογίζει το εμβαδόν ενός τριγώνου, όταν είναι γνωστά η βάση και το ύψος. Εδώ, θα έχουμε δύο παραμέτρους για το πέρασμα τιμών (βάση, ύψος) και μια παράμετρο για την επιστροφή του αποτελέσματος (εμβαδόν):

ΔΙΑΔΙΚΑΣΙΑ Εμβαδόν_Τριγώνου (βάση, ύψος, εμβαδόν)
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: βάση, ύψος

ΠΡΑΓΜΑΤΙΚΕΣ: εμβαδόν
ΑΡΧΗ
Εμβαδόν <- (βάση * ύψος)/2
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ

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

ΠΡΟΓΡΑΜΜΑ Υπολογισμός_Εμβαδών_διαφόρων_Τριγώνων
ΜΕΤΑΒΛΗΤΕΣ
ΑΚΕΡΑΙΟΙ: Β, Υ
ΠΡΑΓΜΑΤΙΚΕΣ: Ε
ΑΡΧΗ
Β <- 12
Υ <- 4
ΚΑΛΕΣΕ Εμβαδόν_Τριγώνου (Β, Υ, Ε)
ΤΥΠΩΣΕ "Το εμβαδόν είναι:", Ε

! Θα μπορούσαμε να το κάνουμε και ως εξής:

ΚΑΛΕΣΕ Εμβαδόν_Τριγώνου (10, 5, Ε)
ΤΥΠΩΣΕ "Το εμβαδόν είναι:", Ε

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

Παρατήρηση:

Η διαδικασία καλείται με την εντολή **Κάλεσε.**

Άσκηση: Να γραφεί μία διαδικασία που να εκτυπώνει ένα πλήθος άστρων (δηλαδή τον χαρακτήρα *) στην οθόνη. Το πλήθος των άστρων που θα τυπώνει θα περνά ως παράμετρος. Για παράδειγμα, αν περάσουμε τον αριθμό 5 να τυπώνει *****, ενώ αν περάσουμε τον αριθμό 3 να τυπώνει ***.