Θεωρία-Οδηγίες ΑΕΠΠ

1ο ΓΕΛ Καστοριάς

Ενότητα 1 Εισαγωγή-Ανάλυση προβλήματος (κεφ. 1 σχολικού βιβλίου)

Το πρόβλημα αποτελεί έννοια που απαντάται σε όλες τις επιστήμες και τους κλάδους τους, αλλά παράλληλα και στην καθημερινή μας ζωή.
Με τον όρο **Πρόβλημα** εννοείται μια κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, η δε λύση της δεν είναι γνωστή, ούτε προφανής.
Η οποιαδήποτε προσπάθεια **αντιμετώπισης** ενός προβλήματος είναι καταδικασμένη σε αποτυχία αν προηγουμένως δεν έχει γίνει απόλυτα κατανοητό το πρόβλημα που τίθεται.
Η **κατανόηση** ενός προβλήματος αποτελεί συνάρτηση δύο παραγόντων, της σωστής **διατύπωσης** εκ μέρους του δημιουργού του και της αντίστοιχα σωστής **ερμηνείας** από τη μεριά εκείνου που καλείται να το αντιμετωπίσει.

Η κατανόηση ενός προβλήματος εξαρτάται σε μεγάλο βαθμό από τη **διατύπωσή** του. Οποιοδήποτε **μέσο** μπορεί να χρησιμοποιηθεί για να αποδοθεί η διατύπωση ενός προβλήματος. Συνηθέστερο από όλα είναι ο **λόγος**, είτε ο προφορικός, είτε ο γραπτός.
Σημαντικός ακόμα παράγοντας στη σωστή αντιμετώπιση ενός προβλήματος είναι η αποσαφήνιση του **χώρου** στον οποίο αναφέρεται. Η πληροφορία αυτή παρέχεται επίσης από την εκφώνηση του προβλήματος. Τα δεδομένα του προβλήματος είναι αυτά που θα μας παρέχουν αυτήν την πληροφορία.

Είναι σαφές ότι τα προβλήματα δεν απαιτούν για τη λύση τους την παρουσία υπολογιστών. Οι υπολογιστές έκαναν την εμφάνισή τους πολύ αργότερα από την εμφάνιση των προβλημάτων.

Με τον όρο **πρόβλημα** δεν αναφερόμαστε μόνο σε μαθηματικού τύπου προβλήματα, τα οποία αποτελούν ένα μόνο είδος προβλημάτων. Προβλήματα υπάρχουν και θέτονται σε όλες τις επιστήμες και τους επιστημονικούς κλάδους, αλλά και σε κοινωνικό, πολιτισμικό, εκπαιδευτικό και σε κάθε άλλο επίπεδο της ανθρώπινης δραστηριότητας, καθώς επίσης και σε καταστάσεις της καθημερινής ζωής.
Αφού κάποτε θα βρεθείς οπωσδήποτε στη θέση να διατυπώσεις ένα πρόβλημα, θα πρέπει να δώσεις προσοχή στη διατύπωσή του, έτσι ώστε να μην δημιουργεί παρερμηνείες και συγχύσεις σε κάποιον που θα κληθεί να το αντιμετωπίσει. Ιδιαίτερα μεγάλη προσοχή απαιτείται, αν το πρόβλημα “εκφράζεται” προς υπολογιστή, αφού η μηχανή δεν έχει την ευχέρεια να καταλάβει αυτά που θέλεις να δηλώσεις, αν δεν είναι απολύτως σωστά διατυπωμένο.

Με τον όρο **δεδομένο** δηλώνεται οποιοδήποτε στοιχείο μπορεί να γίνει αντιληπτό από έναν τουλάχιστον παρατηρητή με μία από τις πέντε αισθήσεις του. Με άλλα λόγια, παράσταση γεγονότων, εννοιών ή εντολών σε τυποποιημένη μορφή που είναι κατάλληλη για επικοινωνία, ερμηνεία ή επεξεργασία από άνθρωπο ή αυτόματα μέσα. Τα δεδομένα

(data) είναι η **αφαιρετική αναπαράσταση** της πραγματικότητας και συνεπώς μία απλοποιημένη όψη της. Τα δεδομένα, λοιπόν, είναι ακατέργαστα γεγονότα, και κάθε φορά η επιλογή τους εξαρτάται από τον τύπο του προβλήματος. Η συλλογή των ακατέργαστων δεδομένων και ο συσχετισμός τους δίνει ως αποτέλεσμα την πληροφορία.
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. Άρα η συστηματική εκτέλεση πράξεων σε δεδομένα. Παραδείγματα: χειρισμός, συγχώνευση, ταξινόμηση, μεταγλώττιση.
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.

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. Η δυσκολία αντιμετώπισης των προβλημάτων ελαττώνεται όσο περισσότερο προχωράει η **ανάλυση** τους σε απλούστερα προβλήματα.
Η ανάλυση αυτή του προβλήματος σε άλλα απλούστερα αναδεικνύει παράλληλα και τη δομή του προβλήματος. Για τη γραφική απεικόνιση της δομής ενός προβλήματος χρησιμοποιείται συχνότατα η **διαγραμματική αναπαράσταση**. Σύμφωνα με αυτή:
α. το αρχικό πρόβλημα αναπαρίσταται από ένα ορθογώνιο παραλληλόγραμμο
β. κάθε ένα από τα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα, αναπαρίσταται επίσης από ένα ορθογώνιο παραλληλόγραμμο.
γ. τα παραλληλόγραμμα που αντιστοιχούν στα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα, σχηματίζονται ένα επίπεδο χαμηλότερα. Έτσι, σε κάθε κατώτερο επίπεδο, δημιουργείται η γραφική αναπαράσταση των προβλημάτων στα οποία αναλύονται τα προβλήματα του αμέσως ψηλότερου επιπέδου.
Η διαγραμματική αναπαράσταση προσφέρει μια **απτή απεικόνιση της δομής** του προβλήματος. Η δημιουργία του σχετικού διαγράμματος βοηθάει τόσο στην καλύτερη κατανόηση του ίδιου του προβλήματος, όσο και στη σχεδίαση της λύσης του.

Συμπερασματικά 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 επί μέρους απλούστερα προβλήματα.
✔ **επίλυση**, όπου υλοποιείται η λύση του προβλήματος, μέσω της λύσης των επιμέρους προβλημάτων.

Η διαφορετική φύση των προβλημάτων επιτρέπει την **κατηγοριοποίησή** τους σύμφωνα με ποικίλα κριτήρια.

1. Με κριτήριο τη **δυνατότητα επίλυσης** ενός προβλήματος, διακρίνουμε τρεις κατηγορίες προβλημάτων:
✔ **Επιλύσιμα**, είναι εκείνα τα προβλήματα για τα οποία η λύση τους είναι ήδη γνωστή και έχει διατυπωθεί. Επιλύσιμα μπορεί επίσης να χαρακτηριστούν και προβλήματα, των οποίων η λύση δεν έχει ακόμα διατυπωθεί, αλλά ή συνάφειά τους με άλλα ήδη

επιλυμένα μας επιτρέπει να θεωρούμε σαν βέβαιη τη δυνατότητα επίλυσής τους.
✔ **Ανοικτά**, ονομάζονται εκείνα τα προβλήματα για τα οποία η λύση τους δεν έχει μεν ακόμα βρεθεί, αλλά παράλληλα δεν έχει αποδειχθεί, ότι δεν επιδέχονται λύση.
✔ **Άλυτα**, χαρακτηρίζονται εκείνα τα προβλήματα για τα οποία έχουμε φτάσει στην παραδοχή, ότι δεν επιδέχονται λύση.

2. Με κριτήριο το **βαθμό δόμησης** των λύσεών τους, τα επιλύσιμα προβλήματα μπορούν να διακριθούν σε τρεις επίσης κατηγορίες:
✔ **Δομημένα**, χαρακτηρίζονται εκείνα τα προβλήματα των οποίων η επίλυση προέρχεται από μια αυτοματοποιημένη διαδικασία. Για παράδειγμα, η επίλυση της δευτεροβάθμιας εξίσωσης αποτελεί ένα δομημένο πρόβλημα, αφού ο τρόπος επίλυσης της εξίσωσης είναι γνωστός και αυτοματοποιημένος.
✔ **Ημιδομημένα**, ονομάζονται τα προβλήματα εκείνα των οποίων η λύση επιδιώκεται στα πλαίσια ενός εύρους πιθανών λύσεων, αφήνοντας στον ανθρώπινο παράγοντα περιθώρια επιλογής της.
✔ **Αδόμητα**, χαρακτηρίζονται τα προβλήματα εκείνα των οποίων οι λύσεις δεν μπορούν να δομηθούν ή δεν έχει διερευνηθεί σε βάθος η δυνατότητα δόμησής τους. Πρωτεύοντα ρόλο στην επίλυση αυτού του τύπου προβλημάτων κατέχει η ανθρώπινη διαίσθηση.

3. Το κάθε πρόβλημα σε ότι αφορά στην επίλυσή του, είναι στενά συνδεδεμένο με την έννοια του αλγόριθμου που παρουσιάζουμε αναλυτικά στο επόμενο κεφάλαιο. Με κριτήριο το **είδος της επίλυσης** που επιζητούν, τα προβλήματα διακρίνονται σε τρεις κατηγορίες:
✔ **Απόφασης**, όπου η απόφαση που πρόκειται να ληφθεί σαν λύση του προβλήματος που τίθεται, απαντά σε ένα ερώτημα και πιθανόν αυτή να είναι ένα "Ναι" ή ένα "Όχι". Αυτό που θέλουμε να διαπιστώσουμε σε ένα πρόβλημα απόφασης είναι αν υπάρχει απάντηση που ικανοποιεί τα δεδομένα που θέτονται από το πρόβλημα.
✔ **Υπολογιστικά**, όπου το πρόβλημα που τίθεται απαιτεί τη διενέργεια υπολογισμών, για να μπορεί να δοθεί μία απάντηση στο πρόβλημα. Σε ένα υπολογιστικό πρόβλημα ζητάμε να βρούμε την τιμή της απάντησης που ικανοποιεί τα δεδομένα που παρέχει το πρόβλημα.
✔ **Βελτιστοποίησης**, όπου το πρόβλημα που τίθεται επιζητά το βέλτιστο αποτέλεσμα για τα συγκεκριμένα δεδομένα που διαθέτει. Σε ένα πρόβλημα βελτιστοποίησης, αναζητούμε την απάντηση που ικανοποιεί κατά τον καλύτερο τρόπο (με κριτήριο το χρόνο, το κόστος, την ασφάλεια) τα δεδομένα που παρέχει το πρόβλημα.

Με μια απλοποιημένη προσέγγιση, α**λγόριθμος** είναι μια "συνταγή" που προσδιορίζει τι πρέπει να κάνουμε κάτω από ορισμένες συνθήκες, έτσι ώστε να φτάσουμε στον επιθυμητό σκοπό.

Η "σύγκριση" λειτουργιών ανθρώπου και υπολογιστή επιφέρει βέβαια μια τεράστια ποιοτική διαφορά υπέρ του ανθρώπου.
Οι **λόγοι** που αναθέτουμε την επίλυση ενός προβλήματος σε υπολογιστή σχετίζονται με
✔ την **πολυπλοκότητα** των υπολογισμών,
✔ την **επαναληπτικότητα** των διαδικασιών,
✔ την **ταχύτητα** εκτέλεσης των πράξεων,
✔ το μεγάλο **πλήθος** των δεδομένων.

Όσο και αν τυχόν ξαφνιάζει, ο υπολογιστής δεν μπορεί να εκτελεί παρά μόνο τρεις **λειτουργίες**:

✔ **πρόσθεση**, η οποία αποτελεί τη βασική αριθμητική πράξη, δεδομένου ότι και οι άλλες αριθμητικές πράξεις μπορούν να αντιμετωπιστούν, σαν διαδικασίες πρόσθεσης.

✔ **σύγκριση**, η οποία συνιστά τη βασική λειτουργία για την επιτέλεση όλων των λογικών πράξεων (ΚΑΙ, Η, ΟΧΙ).

✔ **μεταφορά** δεδομένων, λειτουργία που προηγείται και έπεται της επεξεργασίας δεδομένων.

Ενότητα 2 Βασικές Έννοιες Αλγορίθμων – Δομή Ακολουθίας (κεφ. 2 και 7 σχολ. βιβλίου)

Αναλύεται η έννοια του αλγορίθμου και οι τρόποι αναπαράστασής του, οι βασικές έννοιες προγραμματισμού που απαιτούνται για την υλοποίηση ενός αλγορίθμου σε ένα προγραμματιστικό περιβάλλον (σταθερές, μεταβλητές, εντολές, πράξεις) και τέλος περιγράφεται η πρώτη αλγοριθμική δομή (**ακολουθία**).

Βήματα αντιμετώπισης ασκήσεων:
1. Αναλύστε το πρόβλημα
✔ να αναγνωρίσετε τα **ζητούμενα** του προβλήματος; Τι στοιχεία, ποιες ποσότητες θα βρείτε λύνοντας το πρόβλημα.
✔ να αναγνωρίσετε τα **δεδομένα** του προβλήματος; Τι στοιχεία, ποιες ποσότητες χρειάζεστε για να μπορέσετε να υπολογίσετε τα ζητούμενα.
2. Λύστε το πρόβλημα
✔ Περιγράψτε τις ενέργειες που κάνατε για να λύσετε το πρόβλημα.
✔ **Επαναδιατυπώστε** όλες τις εντολές του αλγόριθμου ώστε να αποθηκεύουν τα αποτελέσματά τους σε θέσεις μνήμης και να αξιοποιούν τα αποτελέσματα που βρίσκονται φυλαγμένα σε θέσεις μνήμης από προηγούμενες ενέργειες. Να δίνετε στις θέσεις μνήμης κατάλληλα ονόματα.
3. Σχεδιάστε τον αλγόριθμο
4. Εκτελέστε τον αλγόριθμο
Ο αλγόριθμός σας θα λειτουργεί χωρίς αλλαγές για πολλές διαφορετικές τιμές δεδομένων (στιγμιότυπα).

Με άλλα λόγια:
1. Διαβάζουμε καλά την εκφώνηση
2. Κάνουμε στο πρόχειρο ένα σκαρίφημα με τις αλγοριθμικές σκέψεις μας
3. Υπογραμμίζουμε τις λέξεις κλειδιά
4. Μεταφράζουμε σε κώδικα τις λέξεις κλειδιά
5. Κωδικοποίηση

Σημείωση: Η απάντησή σας σ' ένα υποερώτημα μπορεί να περιλαμβάνει αρκετά βήματα. Σχεδιάστε τα, αναλύστε τον τρόπο σκέψης σας και μετά προχωρήστε στην υλοποίησή τους. Ίσως ένα υποερώτημα να μην μπορείτε να το λύσετε, προχωρήστε στο επόμενο. Καλό είναι να γράψετε ένα σχετικό **σχόλιο** για να βοηθήσετε το βαθμολογητή.

Αλγόριθμος είναι μια **πεπερασμένη** σειρά ενεργειών, αυστηρά **καθορισμένων** και **εκτελέσιμων** σε πεπερασμένο χρόνο, που στοχεύουν στην **επίλυση** ενός προβλήματος.
Η έννοια του αλγόριθμου **δεν** συνδέεται αποκλειστικά και μόνο με προβλήματα της Πληροφορικής.

Κάθε Αλγόριθμος απαραίτητα ικανοποιεί τα επόμενα **κριτήρια**.
✔ **Είσοδος** (input). Καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται, όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη

βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια άλλων απλών εντολών.
✔ **Έξοδος** (output). Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν άλλον αλγόριθμο.
✔ **Καθοριστικότατη** (definiteness). Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της. π.χ. μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέτης της λαμβάνει μηδενική τιμή.
✔ **Περατότητα** (finiteness). Ο Αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων, δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά υπολογιστική διαδικασία .
✔ **Αποτελεσματικότητα** (effectiveness). Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μία εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη.

Στη βιβλιογραφία συναντώνται διάφοροι τρόποι α**ναπαράστασης** ενός αλγορίθμου:
1. με **ελεύθερο κείμενο** (free text), που αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Έτσι εγκυμονεί τον κίνδυνο ότι μπορεί εύκολα να οδηγήσει σε μη εκτελέσιμη παρουσίαση, παραβιάζοντας το τελευταίο χαρακτηριστικό των αλγορίθμων, δηλαδή την αποτελεσματικότητα.
2. με **διαγραμματικές τεχνικές**, που συνιστούν ένα γραφικό τρόπο παρουσίασης του αλγορίθμου. Από τις διάφορες διαγραμματικές τεχνικές που έχουν επινοηθεί, η πιο παλιά και η πιο γνωστή ίσως, είναι το διάγραμμα ροής (flow chart).
3. με **φυσική γλώσσα** (natural language) κατά βήματα. Στην περίπτωση αυτή χρειάζεται προσοχή, γιατί μπορεί να παραβιασθεί το κριτήριο του καθορισμού.
4. με **κωδικοποίηση** (coding), δηλαδή με ένα πρόγραμμα που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

Η **ακολουθιακή δομή** εντολών (σειριακών βημάτων) χρησιμοποιείται πρακτικά για την αντιμετώπιση απλών προβλημάτων, όπου είναι δεδομένη η σειρά εκτέλεσης ενός συνόλου ενεργειών

Σταθερές (constants). Με τον όρο αυτό αναφερόμαστε σε προκαθορισμένες τιμές που παραμένουν αμετάβλητες σε όλη τη διάρκεια της εκτέλεσης ενός αλγορίθμου. Οι σταθερές διακρίνονται σε:
✔ **αριθμητικές**, π.χ. 134, +7, -1,28
✔ **αλφαριθμητικές** π.χ. “Μανταρίνι”, “Hallo World!!”
✔ **λογικές** που είναι ακριβώς δύο, **Αληθής** και **Ψευδής**

Μεταβλητές (variables). Μια μεταβλητή είναι ένα γλωσσικό αντικείμενο, που χρησιμοποιείται για να παραστήσει ένα στοιχείο δεδομένου. Στη μεταβλητή εκχωρείται μια τιμή, η οποία μπορεί να αλλάζει κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Ανάλογα με το είδος της τιμής που μπορούν να λάβουν, οι μεταβλητές διακρίνονται σε αριθμητικές, αλφαριθμητικές και λογικές. Οι μεταβλητές πρέπει να **δηλώνονται στα προγράμματα** γιατί ο μεταγλωττιστής θέλει να γνωρίζει τον τύπο της πληροφορίας που θα αποθηκεύσει στη θέση μνήμης της RAM στην οποία ανατίθεται η μεταβλητή ώστε α) να δεσμεύσει τον κατάλληλο χώρο και β) να κάνει τον έλεγχο τύπων στις εκφράσεις που η μεταβλητή χρησιμοποιείται.

Δήλωση μεταβλητής: πρόταση που ενημερώνει το μεταγλωττιστή για το συμβολικό της όνομα και τον τύπο της. Καλό είναι να επιλέγετε **ονόματα μεταβλητών** σχετικά με το σκοπό που εξυπηρετεί η μεταβλητή και που θα είναι απίθανο να μπερδευτούν τυπογραφικά. Προτείνεται να χρησιμοποιείται σύντομα ονόματα για δείκτες βρόχων και μεγαλύτερα ονόματα για τις υπόλοιπες μεταβλητές.

Τελεστές (operators). Πρόκειται για τα γνωστά σύμβολα που χρησιμοποιούνται στις διάφορες πράξεις. Οι τελεστές διακρίνονται σε αριθμητικούς, λογικούς και συγκριτικούς. Οι μεταβλητές και οι σταθερές είναι τα βασικά αντικείμενα δεδομένων που χειρίζεται ένα πρόγραμμα. Οι τελεστές καθορίζουν τι θα γίνει μ' αυτές. Ο **τύπος** ενός αντικειμένου προσδιορίζει το σύνολο τιμών που μπορεί να έχει το αντικείμενο, καθώς και σε ποιες πράξεις μπορεί να συμμετέχει.

Εκφράσεις (expressions). Οι εκφράσεις διαμορφώνονται από τους τελεστέους (operands), που είναι σταθερές και μεταβλητές και από τους τελεστές. Η διεργασία αποτίμησης μιας έκφρασης συνίσταται στην απόδοση τιμών στις μεταβλητές και στην εκτέλεση των πράξεων. Η τελική τιμή μιας έκφρασης εξαρτάται από την ιεραρχία των πράξεων και τη χρήση των παρενθέσεων. Μια έκφραση μπορεί να αποτελείται από μια μόνο μεταβλητή ή σταθερά μέχρι μια πολύπλοκη μαθηματική παράσταση

Ένα **πρόγραμμα** αποτελείται από κώδικα και δεδομένα.
Κώδικας είναι προτάσεις-εντολές που εκτελούνται στο χρόνο φόρτωσης και εκτέλεσης του προγράμματος.
Τα **δεδομένα** είναι μεταβλητές ή σταθερές.

Δεσμευμένη λέξη: λεκτική μονάδα της οποίας η σημασία καθορίζεται από τους κανόνες της γλώσσας και δεν μπορεί να αλλάζει.

Ακολουθούν και μερικοί χρήσιμοι ορισμοί:
✔ **αλφαριθμητικό:** Σύνολο χαρακτήρων που μπορεί να εμπεριέχει γράμματα, ψηφία και ειδικά σύμβολα, όπως π.χ. σημεία στίξης.
✔ **Εκχώρηση:** Μηχανισμός τιμοδότησης μιας μεταβλητής.
✔ **Εντολή:** Σε μια γλώσσα προγραμματισμού μια έκφραση που έχει νόημα και η οποία καθορίζει μια πράξη και προσδιορίζει τους τελεστέους της, αν υπάρχουν.
✔ **φυσική γλώσσα** natural language: Γλώσσα οι κανόνες της οποίας βασίζονται στην τρέχουσα χρήση, χωρίς να είναι αυστηρά προδιαγεγραμμένο
✔ **σταθερά** constant**:** Γλωσσικό αντικείμενο που παίρνει μόνο μια ειδική τιμή.

Για να εκτελεστεί ένας αλγόριθμος στον υπολογιστή πρέπει να γραφτεί ένα πρόγραμμα που να τον υλοποιεί, διαδικασία γνωστή και ως **κωδικοποίηση**.

Τα **σχόλια** περιγράφουν τη λειτουργία ενός προγράμματος ή γενικότερα ενός τμήματος κώδικα και γράφονται για να βοηθήσουν τους ανθρώπους και όχι τον υπολογιστή στην κατανόηση και συντήρηση ενός προγράμματος. Όταν ένα πρόγραμμα μεταφράζεται, τα σχόλια αγνοούνται. Τα σχόλια που περιγράφουν τη λειτουργία ενός προγράμματος, το ρόλο των μεταβλητών και τη λειτουργία πολύπλοκων τμημάτων κώδικα, αποτελούν παράδειγμα καλού προγραμματιστικού στυλ.

Άλλωστε ένα καλό **προγραμματιστικό στυλ** χτίζεται και από:

- Σχόλια
- Εσοχές
- Σχετικά ονόματα μεταβλητών
- Κατανοήσιμο κώδικα (more is not always better)
- Πριν την εντολή Διάβασε μία επεξηγηματική-προτρεπτική Γράψε.

Στον προγραμματισμό πρέπει πάντα να σκεφτόμαστε από πολλές οπτικές γωνίες:

- **Προγραμματιστής**
- **Η/Υ** (οθόνη, επεξεργαστής, μνήμη) (βλέπε πίνακας τιμών)
- Τελικός **χρήστης**

Χρησιμοποιώντας αποκλειστικά τη **δομή ακολουθίας**, μπορείτε να λύσετε μόνο προβλήματα στα οποία:

- ✔ η σειρά των βημάτων είναι καθορισμένη
- ✔ όλα τα βήματα εκτελούνται πάντοτε
- ✔ δεν υπάρχουν εξαιρέσεις!

Ακολουθούν μερικές **παρατηρήσεις**:

✔ Για τελεστές ίδιου βαθμού προτεραιότητας στην ίδια έκφραση, ισχύει η προτεραιότητα από τα αριστερά προς τα δεξιά.

✔ Η εντολή Διάβασε είναι **εκτελεστέα** εντολή, ενώ η εντολή Αλγόριθμος είναι **δηλωτική** εντολή.

✔ Οι πράξεις μέσα σε **παρενθέσεις** εκτελούνται κατά προτεραιότητα.

✔ Η **προτεραιότητα** των λογικών τελεστών θα θεωρούμε ότι έχει την ίδια προτεραιότητα και όταν γράφουμε σύνθετες συνθήκες, θα χρησιμοποιούμε πάντα παρενθέσεις.

✔ Δεν ορίζονται στη ΓΛΩΣΣΑ πράξεις μεταξύ τελεστέων τύπου χαρακτήρα.

✔ Σε τελεστέους λογικού τύπου, ορίζονται μόνο οι λογικές πράξεις και από συγκρίσεις η ισότητα και η ανισότητα.

✔ Οι συγκριτικές πράξεις γίνονται μεταξύ τελεστέων ίδιου τύπου.

✔ Συχνά κατά την ποσοστιαία αύξηση ή μείωση ξεχνάμε να πολλαπλασιάσουμε το ποσοστό με την αρχική τιμή. Γράφουμε για παράδειγμα χ←χ+10/100, ενώ εννοούμε χ←χ+ 10/100*χ (ή χ←1.1*χ).

✔ Η απόσταση μεταξύ δύο τιμών εκφράζεται από την απόλυτη τιμή της διαφοράς τους

✔ **Δεδομένα** και **Αποτελέσματα** μόνο σε αλγόριθμο (δηλωτικές εντολές).

✔ Η εντολή **Αντιμετάθεσε** δεν ορίζεται στη ΓΛΩΣΣΑ (μόνο σε αλγόριθμο).

✔ Το ΚΑΙ έχει την ίδια προτεραιότητα με το Ή. Επομένως καλό είναι να χρησιμοποιούνται παρενθέσεις στις λογικές εκφράσεις για να μην υπάρχει αμφιβολία για τη σειρά εκτέλεσή τους.

✔ Γενικά, θεωρούμε ότι κάθε ΓΡΑΨΕ τυπώνει σε μία γραμμή τη λίστα των ορισμάτων της και γενικά ότι το αποτέλεσμα που τυπώνει είναι ευκρινές δηλαδή όχι κολλημένο το ένα με το άλλο. Επίσης θεωρούμε ότι ως όρισμα στη **ΓΡΑΨΕ** μπορεί να μπει και μία έκφραση.

✔ Αν και προγραμματίζοντας στο χαρτί δεν αντιμετωπίζουμε πρόβλημα μεγέθους στους αριθμούς, στον υπολογιστή έχουμε πάντα περιορισμένο μέγεθος, όσο μεγάλο και αν είναι αυτό.

✔ Δεν υπάρχει διαφορετικός τύπος δεδομένων για ένα χαρακτήρα και για πολλούς χαρακτήρες.

✔ Υπάρχει διευκρινιστική οδηγία από το Παιδαγωγικό Ινστιτούτο (πλέον ΙΕΠ), η οποία

αναφέρει ότι δεν πρέπει να χρησιμοποιούνται σε ασκήσεις με αρνητικούς αριθμούς ή πραγματικούς οι τελεστές **MOD, DIV**.

✔ Αποδεκτά ονόματα α**ναγνωριστικών** (δηλαδή όνομα προγράμματος, σταθεράς ή μεταβλητής) είναι αυτά που ξεκινούν με γράμμα (είτε ελληνικό είτε αγγλικό) και στη συνέχεια περιλαμβάνουν γράμματα, αριθμούς ή την κάτω παύλα (_).

✔ Η **εντολή ανάθεσης** τιμής μπορεί να μεταφραστεί ως εξής: «Υπολόγισε την τιμή της έκφρασης δεξιά από το σύμβολο ανάθεσης τιμής και βάλε το αποτέλεσμα στη μεταβλητή που βρίσκεται αριστερά από την ανάθεση τιμής». Στο δεξί μέρος μπορεί να βρίσκεται οτιδήποτε έχει τιμή, ενώ στο αριστερό μία και μόνο μεταβλητή (ή στοιχείο πίνακα) ίδιου τύπου δεδομένων με την έκφραση.

✔ Η εντολή **ΔΙΑΒΑΣΕ** διαβάζει από το πληκτρολόγιο μία λίστα μεταβλητών, χωρισμένων από κόμματα. Δεν επιτρέπεται να διαβαστεί κάτι που δεν είναι μεταβλητή.

✔ Η συνάρτηση A_M επιστρέφει το ακέραιο τμήμα ενός πραγματικού αριθμού, για παράδειγμα A_M(6.98) = 6, ενώ A_M(-9.77) = -9.

✔ Μια παράσταση σε παρενθέσεις: οι παρενθέσεις χρησιμοποιούνται για να υποδείξουν τη σειρά των πράξεων. Μια παράσταση σε παρενθέσεις είναι από μόνη της ένας όρος που αντιμετωπίζεται από τον μεταγλωττιστή ως μια μονάδα που πρέπει να υπολογιστεί πριν να συνεχιστεί ο υπολογισμός. Μια παράσταση (expression) αποτελείται από: **όρους** (terms) που παριστάνουν τιμές δεδομένων και **τελεστές** (operators) που υποδεικνύουν μια υπολογιστική πράξη.

✔ χ←χ * 10 και χ←χ div 10 μπορούμε να πούμε ότι είναι η ολίσθηση αριστερά και η ολίσθηση δεξιά αντίστοιχα στο δεκαδικό σύστημα αρίθμησης.

✔ Η **ιεραρχία** (προτεραιότητα) των τελεστών σε μια έκφραση είναι: αριθμητικοί, συγκριτικοί, λογικοί.

✔ Χρήσεις των τελεστών MOD/DIV: έλεγχος άρτιου, πολλαπλάσια, απομόνωση ψηφίων.

✔ Για τη σύνταξη μιας λογικής έκφρασης ή **συνθήκης** χρησιμοποιούνται σταθερές, μεταβλητές, αριθμητικές παραστάσεις, συγκριτικοί και λογικοί τελεστές, καθώς και παρενθέσεις.

Τώρα που αρχίζεις να γράφεις προγράμματα, χρήσιμο θα ήταν να μάθεις να ακολουθείς γενικές αρχές έτσι ώστε, η συγγραφή, η κατανόηση και η τροποποίηση των προγραμμάτων σου να γίνεται εύκολα και γρήγορα. Τα προγράμματά σου πρέπει να είναι **απλά** και **κατανοητά**. Όχι μόνο για τους βαθμολογητές, αλλά και για σένα που θα επανέλθεις σε παλαιότερο σου πρόγραμμα για να το τροποποιήσεις ή να το επεκτείνεις.

Μερικά σημεία για να είναι κατανοήσιμος ο κώδικας:

✔ η χρήση **κενών γραμμών** διευκολύνει την ανάγνωση του προγράμματος και οριοθετεί τις ενότητες του.

✔ Η χρήση **εσοχών.**

✔ η χρησιμοποίηση **σταθερών** σε διευκολύνει σε πιθανές επόμενες αλλαγές και σε προστατεύει από αθέλητες τροποποιήσεις.

✔ Θα πρέπει να αποδίδεις **αρχικές τιμές** στις μεταβλητές που χρησιμοποιείς στο πρόγραμμα. Επιπλέον η απόδοση αρχικών τιμών βοηθάει στην καλύτερη κατανόηση του προγράμματος και στην ευκολότερη συντήρησή του.

✔ Να αποφεύγεις να χρησιμοποιείς μεγάλους υπολογισμούς. Η διάσπαση ενός υπολογισμού σε απλούστερους, διευκολύνει τους άλλους στην κατανόηση του προγράμματος και σένα στην αποφυγή λαθών.

✔ Είναι καλύτερο να χρησιμοποιείς **παρενθέσεις**, έστω και αν δεν είναι απαραίτητο, σε

προφυλάσσει από πιθανά λάθη και αβλεψίες, ενώ ταυτόχρονα κάνει το πρόγραμμα πιο εύκολο στην κατανόηση του.

Στην πραγματικότητα όμως μόνο η **εξάσκηση** και η πείρα θα σου εξασφαλίσουν τη δυνατότητα να συντάσσεις εύκολα και γρήγορα σωστά προγράμματα.
Ιδιαίτερη έμφαση πρέπει να δίνεται στη **σχεδίαση** του αλγορίθμου. Οι μαθητές ενθαρρύνονται να επιλύουν το πρόβλημα στο πρόχειρο, να σχεδιάζουν τα βήματα του αλγόριθμου και μετά να το μετατρέπουν σε κώδικα. Αυτό πρέπει να το συνηθίσουν από την αρχή, ακόμη και για τα πιο απλά προβλήματα.

Ενότητα 3 Δομή Επιλογής (κεφ. 2 και 8 σχολ. Βιβλίου)

Χρησιμοποιώντας αποκλειστικά τη **δομή ακολουθίας**, μπορείτε να λύσετε μόνο προβλήματα στα οποία:
- ✔ όλα τα βήματα εκτελούνται πάντοτε
- ✔ δεν υπάρχουν απρόοπτα-εξαιρέσεις
- ✔ η σειρά των βημάτων είναι καθορισμένη

Για να δώσετε λύση σε πιο σύνθετα προβλήματα πρέπει να μπορείτε να δημιουργείτε στον αλγόριθμό λογικά μονοπάτια, εξετάζοντας απλά λογικά ερωτήματα. Σε αυτή την ενότητα θα ασχοληθείτε με μία νέα δομή, τη **δομή επιλογής**.
Οι ασκήσεις αυτής της ενότητας θα σας βοηθήσουν να κατανοήσετε πώς να σχεδιάζετε έναν αλγόριθμο, όταν αυτός περιλαμβάνει κάποιο βήμα που **δεν θα εκτελείται πάντοτε.** Έχετε πλέον τη δυνατότητα να δημιουργείτε **λογική** στους αλγορίθμους σας και να προσδιορίζετε εναλλακτικά μονοπάτια, ορίζοντας φίλτρα-κριτήρια επιλογής.

Η εντολή απλής επιλογής **ΑΝ** συνθήκη **ΤΟΤΕ** ορίζεται διαφορετικά στη Ψευδογλώσσα και διαφορετικά στη ΓΛΩΣΣΑ. Στη ΓΛΩΣΣΑ απαιτείται ένα **ΤΕΛΟΣ_ΑΝ** για κάθε ΑΝ, ενώ στη Ψευδογλώσσα αν στο τότε έχεις μόνο μια εντολή, επιτρέπεται να γραφεί στην ίδια ευθεία. Δηλαδή: **ΑΝ** συνθήκη **ΤΟΤΕ** εντολή

Η διαδικασία της επιλογής περιλαμβάνει τον έλεγχο κάποιας συνθήκης που μπορεί να έχει δύο τιμές και ακολουθεί η απόφαση εκτέλεσης κάποιας ενέργειας, ανάλογα με την τιμή της συνθήκης.

Και λίγοι **ορισμοί**:
Εμφωλευμένα ΑΝ ονομάζονται δύο ή περισσότερες εντολές της μορφής **ΑΝ** ...**ΑΛΛΙΩΣ** που περιέχονται η μία μέσα στην άλλη.
Συνθήκη: Μια έκφραση σε πρόγραμμα που μπορεί να εκτιμηθεί είτε ως αληθής είτε ως ψευδής, όταν εκτελείται το πρόγραμμα.
Μεταβλητή ελέγχου: Μεταβλητή της οποίας η τιμή ελέγχει τον αριθμό εκτελέσεων ενός βρόχου.

Εντολές ελέγχου: εντολές που επηρεάζουν τον τρόπο με τον οποίο εκτελούνται άλλες εντολές. Οι εντολές ελέγχου χωρίζονται σε δύο βασικές κατηγορίες:
- ✔ Εντολές **υπό συνθήκη**: χρησιμοποιούνται για την επιλογή δύο ή περισσότερων ανεξάρτητων διαδρομών σε ένα πρόγραμμα, ανάλογα με το αποτέλεσμα κάποιου ελέγχου συνθήκης.
- ✔ Εντολές **επανάληψης (Βλέπε Ενότητα 4)**: χρησιμοποιούνται όταν χρειάζεται να επαναλαμβάνεται στο πρόγραμμα μια λειτουργία ένα καθορισμένο πλήθος φορών (**ΓΙΑ**) ή όσο ισχύει μια συγκεκριμένη συνθήκη (**ΟΣΟ/ΜΕΧΡΙΣ_ΟΤΟΥ**).

Οι μεταβλητές **λογικού τύπου** χρησιμοποιούνται συχνά στα προγράμματα ως **σημαίες** (flags), για να σηματοδοτήσουν δηλαδή αν έχετε ολοκληρώσει ή όχι κάποια φάση της λειτουργίας.

Η εντολή **ΑΛΛΙΩΣ_ΑΝ** είναι διαφορετική από την **ΑΛΛΙΩΣ ΑΝ.**
Πριν χρησιμοποιήσεις **εμφωλευμένα ΑΝ**, σκέψου μήπως το ίδιο πρόγραμμα μπορεί να υλοποιηθεί απλούστερα με σύνθετες λογικές εκφράσεις ή την **πολλαπλή ΑΝ**.

Μην ξεχνάτε: η χρήση **εσοχών** καθιστά πολύ πιο ευκολονόητη τη δομή του προγράμματος, αυξάνοντας τη σαφήνειά του.

Ενότητα 4 Δομή Επανάληψης (κεφ. 2 και 8 σχολ. Βιβλίου)

Η διαδικασία της επανάληψης είναι ιδιαίτερα συχνή, αφού πλήθος προβλημάτων μπορούν να επιλυθούν με κατάλληλες επαναληπτικές διαδικασίες. Η λογική των επαναληπτικών διαδικασιών εφαρμόζεται στις περιπτώσεις, όπου μία ακολουθία εντολών πρέπει να εφαρμοσθεί σε ένα σύνολο περιπτώσεων, που έχουν κάτι κοινό.

Η δομή **επανάληψης** συναντάται κυρίως σε προβλήματα όπου έχουμε να επεξεργαστούμε μια σειρά από παρόμοιες τιμές δεδομένων αρκετές φορές. Τα προβλήματα αυτά άλλοτε, έχουν άγνωστο πλήθος φορών της παραπάνω επεξεργασίας και άλλοτε γνωστό. Πρέπει να είμαστε σε θέση μέσα από τις προδιαγραφές του προβλήματος, να καταλάβουμε σε ποια περίπτωση είμαστε.

Βρόχος loop: Σύνολο εντολών που μπορεί να εκτελεστεί επανειλημμένα, όσο ισχύει μια ορισμένη συνθήκη. Το τμήμα του αλγόριθμου που επαναλαμβάνεται.
Επανάληψη: Η διαδικασία επαναληπτικής εκτέλεσης ενός συνόλου εντολών μέχρι την ικανοποίηση κάποιας συνθήκης.
Λογική των εντολών επανάληψης (βρόχων): εφαρμόζεται σε περιπτώσεις, οι οποίες έχουν κάτι κοινό, όπου μία ακολουθία εντολών πρέπει να εκτελεστεί περισσότερες από μία φορά. Οι εκτελέσεις ελέγχονται πάντοτε από κάποια λογική έκφραση η οποία καθορίζει την έξοδο από το **βρόχο.**
Η τιμή **φρουρός** χρησιμοποιείται για τη διακοπή εκτέλεσης ενός βρόχου.

Η εντολή **Για** είναι η πιο απλή εντολή επανάληψης. Χρησιμοποιεί ένα μετρητή για να μετράει πόσες επαναλήψεις γίνονται. Επιλέγουμε να τη χρησιμοποιήσουμε όταν γνωρίζουμε από πριν πόσες επαναλήψεις θέλουμε. Ο αριθμός των επαναλήψεων που πραγματοποιείται ισούται με Ακέραιο Μέρος του (τελική τιμή - αρχική τιμή)/βημα+1.
Η εντολή ΓΙΑ χρησιμοποιείται μόνο για προκαθορισμένο αριθμό επαναλήψεων. Αν λοιπόν ξέρεις τον αριθμό των επαναλήψεων ή μπορείς να τον υπολογίσεις, τότε να χρησιμοποιείς την εντολή ΓΙΑ. Ποτέ μη χρησιμοποιείς εντολές που αλλάζουν την αρχική τιμή, την τελική τιμή, το βήμα ή τη μεταβλητή που ελέγχει την επανάληψη μέσα σε ένα βρόχο ΓΙΑ. Διαφυγή (έξοδος από τον χρήστη) βρόχου δε γίνεται στη ΓΙΑ.

Η εντολή **Όσο** είναι η πιο **ισχυρή** εντολή επανάληψης. Χρησιμοποιείται συνήθως όταν δεν ξέρουμε τον αριθμό των επαναλήψεων, αλλά οι επαναλήψεις εξαρτώνται από κάποια συνθήκη. Την προτιμούμε από την **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** όταν είναι πιθανό να μη γίνει καμία επανάληψη, γιατί στην **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** γίνεται πάντα τουλάχιστον μία.
Αν θέλετε να βάλετε μετρητή στην **ΟΣΟ**, θυμηθείτε τα τρία βήματα του μετρητή:
- ✔ Αρχικοποίηση πριν την **ΟΣΟ** (π.χ. i←1)
- ✔ Μεταβολή πριν το **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ** (π.χ. i←i+1)
- ✔ Συνθήκη τέλους στην **ΟΣΟ** (π.χ. **ΟΣΟ** i<=20)

Η εντολή **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** χρησιμοποιείται όταν δεν υπάρχει συγκεκριμένος αριθμός επαναλήψεων, αλλά οι επαναλήψεις εξαρτώνται από κάποια συνθήκη. Την προτιμούμε από την **ΟΣΟ** όταν θέλουμε να γίνει τουλάχιστον μία επανάληψη. **(**Ελεγχος εισαγωγής ακεραίου κλπ).
Αρχή_ επανάληψης

Διάβασε χ
Μέχρις_ότου χ=Α_Μ(χ)

Οι μεταβλητές που ελέγχουν την επανάληψη του βρόχου **ΟΣΟ** και **ΜΕΧΡΙΣ_ΟΤΟΥ** πρέπει υποχρεωτικά να αλλάζουν τιμή μέσα στο σώμα του βρόχου, αλλιώς ή δεν εκτελείται ποτέ ή συνηθέστερα δεν σταματάει η εκτέλεση του (**ατέρμων** βρόχος).
Οι επαναλήψεις που υλοποιούνται με την εντολή **ΟΣΟ**, μπορεί να μην εκτελεστούν ούτε μία φορά, αφού ο έλεγχος γίνεται στην είσοδο του βρόχου, αντίθετα οι επαναλήψεις **ΜΕΧΡΙΣ_ΟΤΟΥ** θα πραγματοποιηθούν τουλάχιστον μία φορά.

Φώλιασμα: Η ένθεση βρόχου ή τμήματος κώδικα μέσα σε άλλους βρόχους ή μπλοκ κώδικα.
Στη χρήση των **εμφωλευμένων** βρόχων ισχύουν συγκεκριμένοι κανόνες που πρέπει να ακολουθούνται αυστηρά για την σωστή λειτουργία των προγραμμάτων. Συγκεκριμένα:

- ✔ Ο εσωτερικός βρόχος πρέπει να βρίσκεται ολόκληρος μέσα στον εξωτερικό. Ο βρόχος που ξεκινάει τελευταίος, πρέπει να ολοκληρώνεται πρώτος.
- ✔ Η είσοδος σε κάθε βρόχο υποχρεωτικά γίνεται από την αρχή του.
- ✔ Δεν μπορεί να χρησιμοποιηθεί η ίδια μεταβλητή ως μετρητής δύο ή περισσοτέρων βρόχων που ο ένας βρίσκεται στο εσωτερικό του άλλου.

Ενότητα 5 Δομές δεδομένων - Πίνακες (κεφ. 3 και 9 σχολ. Βιβλίου)

Οι **δομές δεδομένων** είναι ο μηχανισμός της γλώσσας που μας δίνει τη δυνατότητα να ομαδοποιήσουμε δεδομένα που σχετίζονται μεταξύ τους ώστε να τα επεξεργαστούμε καλύτερα.

Οι δομές δεδομένων διακρίνονται σε δύο μεγάλες κατηγορίες: τις **στατικές** (static) και τις **δυναμικές** (dynamic). Οι δυναμικές δομές δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης, αλλά στηρίζονται στην τεχνική της λεγόμενης δυναμικής παραχώρησης μνήμης (dynamic memory allocation). Με άλλα λόγια, 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 δεδομένα αντίστοιχα. Όλες οι σύγχρονες γλώσσες προγραμματισμού προσφέρουν τη δυνατότητα δυναμικής παραχώρησης μνήμης.

Συχνά δεν μας αρκούν οι μεταβλητές! Πώς θα αποθηκεύαμε;
- ✔ Ένα σύνολο μαθητών;
- ✔ Μία λίστα από πόλεις;
- ✔ Μαθητές μαζί με τους βαθμούς τους;

Άρα βοηθούν οι πίνακες στην επίλυση προβλημάτων που δεν θα μπορούσαμε να λύσουμε μόνο με μεταβλητές.
Οι **πίνακες** αποτελούν ένα βολικό τρόπο διαχείρισης πολλών δεδομένων **ίδιου τύπου**. Η χρήση τους είναι αναγκαία όταν τα δεδομένα ενός προγράμματος πρέπει να διατηρηθούν στη μνήμη του μέχρι το τέλος της εκτέλεσης του. **Πίνακας** είναι ένα σύνολο αντικειμένων ίδιου τύπου, τα οποία αναφέρονται με ένα κοινό όνομα. Κάθε ένα από τα αντικείμενα λέγεται **στοιχείο** του πίνακα. Η αναφορά σε ατομικά στοιχεία του πίνακα γίνεται με το όνομα του πίνακα ακολουθούμενο από ένα δείκτη.

Με τη χρήση του πίνακα, όλα τα δεδομένα καταχωρούνται κάτω από το ίδιο όνομα μεταβλητής και έχει δύο χαρακτηριστικά:
1. Ο πίνακας είναι **διατεταγμένος:** τα μεμονωμένα συστατικά στοιχεία ενός πίνακα μπορούμε να τα ξεχωρίσουμε ως πρώτο, δεύτερο, κ.ο.κ.
2. Ο πίνακας είναι **ομοιογενής:** όλες οι τιμές που αποθηκεύονται σε έναν πίνακα είναι του ίδιου τύπου.

Οι πίνακες θεωρούμε ότι είναι στατικές δομές και άρα πρέπει να ορίζονται στην αρχή κάθε προγράμματος.

Το όνομα του πίνακα μπορεί να είναι οποιοδήποτε δεκτό όνομα της **ΓΛΩΣΣΑΣ** και ο δείκτης είναι μία ακέραια έκφραση, σταθερή ή μεταβλητή που περικλείεται μέσα στα σύμβολα [και]. Κάθε πίνακας πρέπει υποχρεωτικά να περιέχει δεδομένα του ιδίου τύπου, δηλαδή ακέραια, πραγματικά, λογικά, ή αλφαριθμητικά. Ο τύπος του πίνακα δηλώνεται μαζί με τις άλλες μεταβλητές του προγράμματος στο τμήμα δήλωσης μεταβλητών. Εκτός από τον τύπο του πίνακα πρέπει να δηλώνεται και ο αριθμός των στοιχείων που περιέχει ή καλύτερα ο μεγαλύτερος αριθμός στοιχείων που μπορεί να έχει ο συγκεκριμένος πίνακας και αυτό για να δεσμευτούν οι αντίστοιχες συνεχόμενες θέσεις

μνήμης.

Η χρήση πινάκων είναι ένας **βολικός τρόπος** για τη διαχείριση πολλών δεδομένων ιδίου τύπου, αλλά συχνά η χρήση τους είναι **περιττή** και επιζήμια στην ανάπτυξη του προγράμματος.

- ✔ **Οι πίνακες απαιτούν μνήμη**. Κάθε πίνακας δεσμεύει από την αρχή του προγράμματος πολλές θέσεις μνήμης. Σε ένα μεγάλο και σύνθετο πρόγραμμα η άσκοπη χρήση μεγάλων πινάκων μπορεί να οδηγήσει ακόμη και σε αδυναμία εκτέλεσης του προγράμματος.
- ✔ **Οι πίνακες περιορίζουν τις δυνατότητες του προγράμματος.** Αυτό γιατί οι πίνακες είναι στατικές δομές και το μέγεθος τους πρέπει να δηλώνεται στην αρχή του προγράμματος, ενώ παραμένει υποχρεωτικά σταθερό κατά την εκτέλεση του προγράμματος.

Κάθε πίνακας έχει δύο βασικές ιδιότητες: τον **τύπο στοιχείου**, δηλαδή τον τύπο των τιμών οι οποίες είναι δυνατό να αποθηκευτούν στα στοιχεία του πίνακα και το **μέγεθος πίνακα**, δηλαδή το πλήθος των στοιχείων που περιέχει. Κάθε φορά που δημιουργείτε έναν πίνακα στο πρόγραμμα σας, θα πρέπει να καθορίζετε τόσο τον τύπο των στοιχείων όσο και το μέγεθος του πίνακα:

Όταν χρησιμοποιείτε πίνακες στα προγράμματά σας πρέπει να εξασφαλίζετε ότι οι τιμές των αριθμοδεικτών που χρησιμοποιείτε για να επιλέγετε στοιχεία από τους πίνακες θα παραμένουν **εντός των ορίων** των πινάκων.

Με τους πίνακες γράφεις και διαβάζεις σε όποια θέση θέλεις άμεσα:
π.χ. Α[5] = 14
Άρα εύκολη σκέψη και χρήση.
Υπάρχουν δύο τρόποι για να προσπελάσουμε τα στοιχεία ενός πίνακα έστω Α.
Άμεσα, π.χ. Α[i].
Έμμεσα π.χ. Α[Β[i]] όπου ο δείκτης είναι κελί άλλου πίνακα (υποχρεωτικά ακεραίου με θετικές τιμές).

Οδηγία 1: Η ανάγνωση, η επεξεργασία και η εμφάνιση των στοιχείων ενός πίνακα υλοποιείται από βρόχους και συνήθως με την εντολή **ΓΙΑ**. Στους δισδιάστατους πίνακες συνήθως έχουμε εμφωλευμένες επαναλήψεις. Η αναφορά στα στοιχεία των πινάκων γίνεται με τη χρήση **ακέραιων θετικών δεικτών** (τόσοι όσες και οι διαστάσεις). Όταν θέλουμε να προσπελάσουμε τον δισδιάστατο πίνακα κατά γραμμές, βολεύει να βάλουμε πρώτα την εντολή επανάληψης που θα επεξεργαστεί τις γραμμές (αντίστοιχα για τις στήλες).

Οδηγία 2: Δύο ή περισσότεροι πίνακες λέγονται **παράλληλοι** αν τα περιεχόμενα που βρίσκονται στις ίδιες θέσεις τους, αντιστοιχούν σε χαρακτηριστικά της ίδιας οντότητας. Άρα στους παράλληλους πίνακες να προσέχουμε να μεταβάλλουμε τους δείκτες τους ταυτόχρονα και με τον ίδιο τρόπο. Φυσικά δεν είναι υποχρεωτικό να είναι του ίδιου τύπου.

Η χρήση των πινάκων είναι ένας βολικός τρόπος για την αποθήκευση μεγάλου αριθμού δεδομένων ιδίου τύπου. Συνήθως οι νέοι προγραμματιστές χρησιμοποιούν πίνακες

ακόμη και όταν η χρήση τους δεν είναι απαραίτητη! Εξέτασε αν πραγματικά χρειάζεται πίνακας για την επίλυση του προβλήματος. Αν δεν είναι απαραίτητος μην τον χρησιμοποιείς. Να έχεις πάντα στο νου σου ότι οι πίνακες ξοδεύουν μεγάλα ποσά μνήμης.

Για να αποφύγεις τα πλέον κοινά λάθη στη χρήση των πινάκων να προσέχεις πάντα:
✔ Να δίνεις αρχικές τιμές αν χρειάζεται στους πίνακες.
✔ Μην ξεπερνάς τα **όρια** του πίνακα σου. Το πιο συνηθισμένο λάθος στη χρήση των πινάκων είναι η προσπάθεια ανάγνωσης ή εκχώρησης τιμής έξω από τα όρια του πίνακα.
✔ Η επεξεργασία γίνεται στα στοιχεία του πίνακα. Άρα σε όλες τις εντολές πρέπει να εμφανίζονται τα στοιχεία του πίνακα και όχι το όνομα του ίδιου του πίνακα.
✔ Όλα τα στοιχεία του πίνακα έχουν τον ίδιο τύπο, για παράδειγμα όλα είναι ακέραια ή όλα είναι χαρακτήρες όπως ορίστηκαν στο τμήμα δηλώσεων.

Συνήθεις ασκήσεις με πίνακες.
➔ Υπολογισμός αθροισμάτων στοιχείων του πίνακα.
➔ Εύρεση του μέγιστου ή του ελάχιστου στοιχείου.
➔ Ταξινόμηση των στοιχείων του πίνακα.
➔ Αναζήτηση ενός στοιχείου του πίνακα.
➔ Συγχώνευση δύο πινάκων.

Λειτουργίες Δ.Δ.
Δομή Δεδομένων είναι ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών. Κάθε μορφή δομής δεδομένων αποτελείται από ένα σύνολο **κόμβων** (nodes).
Οι βασικές **λειτουργίες** (ή αλλιώς πράξεις) επί των δομών δεδομένων είναι οι ακόλουθες:
✔ **Προσπέλαση** (access), πρόσβαση σε έναν κόμβο με σκοπό να εξετασθεί ή να τροποποιηθεί το περιεχόμενό του.
✔ **Εισαγωγή** (insertion), δηλαδή η προσθήκη νέων κόμβων σε μία υπάρχουσα δομή.
✔ **Διαγραφή** (deletion), που αποτελεί το αντίστροφο της εισαγωγής, δηλαδή ένας κόμβος αφαιρείται από μία δομή.
✔ **Αναζήτηση** (searching), κατά την οποία προσπελάζονται οι κόμβοι μιας δομής, προκειμένου να εντοπιστούν ένας ή περισσότεροι που έχουν μια δεδομένη ιδιότητα.
✔ **Ταξινόμηση** (sorting), όπου οι κόμβοι μιας δομής διατάσσονται κατά αύξουσα ή φθίνουσα σειρά.
✔ **Αντιγραφή** (copying), κατά την οποία όλοι οι κόμβοι ή μερικοί από τους κόμβους μίας δομής αντιγράφονται σε μία άλλη δομή.
✔ **Συγχώνευση** (merging), κατά την οποία δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή.
✔ **Διαχωρισμός** (separation), που αποτελεί την αντίστροφη πράξη της συγχώνευσης.

Δεν μπορούν να εφαρμοστούν η εισαγωγή και η διαγραφή σε πίνακα. Συγκεκριμένα, στη στατική δομή δεδομένων το ακριβές μέγεθος της απαιτούμενης μνήμης καθορίζεται κατά τη στιγμή του προγραμματισμού τους. Αυτό συνεπάγεται ότι αυτό το μέγεθος καθορίζεται κατά τη στιγμή μετάφρασης τους και όχι κατά τη στιγμή εκτέλεσης (on the fly) του προγράμματος. Οι πράξεις της εισαγωγής και της διαγραφής αυξάνουν και μειώνουν αντίστοιχα το μέγεθος μίας δομής δεδομένων, επομένως μπορούν να εφαρμοστούν μόνο σε δυναμικές δομές δεδομένων. Στην πράξη σπάνια χρησιμοποιούνται και οι οκτώ

λειτουργίες για κάποια δομή.

Δείκτης: Στον προγραμματισμό, ένας ακέραιος που αναγνωρίζει τη θέση ενός στοιχείου δεδομένων σε μια ακολουθία στοιχείων δεδομένων.
Εξίσωση **Wirth**: Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα.

Στοίβα stack: Δομή δεδομένων με ένα άκρο στην οποία το τελευταίο στοιχείο που εισάγεται είναι και το πρώτο που μπορεί να εξαχθεί. Δύο είναι οι κύριες λειτουργίες σε μία **στοίβα**:
Η **ώθηση** (push) στοιχείου στην κορυφή της στοίβας, και η **απώθηση** (pop) στοιχείου από τη στοίβα.
Η διαδικασία της ώθησης πρέπει οπωσδήποτε να ελέγχει, αν η στοίβα είναι γεμάτη, οπότε λέγεται ότι συμβαίνει **υπερχείλιση** (overflow) της στοίβας. Αντίστοιχα, η διαδικασία απώθησης ελέγχει, αν υπάρχει ένα τουλάχιστον στοιχείο στη στοίβα, δηλαδή ελέγχει αν γίνεται **υποχείλιση** (underflow) της στοίβας.
Μια στοίβα μπορεί να υλοποιηθεί πολύ εύκολα με τη βοήθεια ενός μονοδιάστατου πίνακα, Μια βοηθητική μεταβλητή (με όνομα συνήθως **top**) χρησιμοποιείται για να δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας. Για την εισαγωγή ενός νέου στοιχείου στη στοίβα (ώθηση) αρκεί να αυξηθεί η μεταβλητή top κατά ένα και στη θέση αυτή να εισέλθει το στοιχείο. Αντίθετα για την εξαγωγή ενός στοιχείου από τη στοίβα (απώθηση) εξέρχεται πρώτα το στοιχείο που δείχνει η μεταβλητή top και στη συνέχεια η top μειώνεται κατά ένα για να δείχνει τη νέα κορυφή.

Ουρά queue: Δομή δεδομένων με δύο άκρα, στην οποία το πρώτο στοιχείο που εισάγεται είναι και το πρώτο που μπορεί να εξαχθεί. Δύο είναι οι κύριες λειτουργίες που εκτελούνται σε μία **ουρά**:
Η **εισαγωγή** (enqueue) στοιχείου στο πίσω άκρο της ουράς, και η εξαγωγή (**dequeue**) στοιχείου από το εμπρός άκρο της ουράς. απαιτούνται **δύο δείκτες**: ο εμπρός (front) και ο πίσω (rear) δείκτης, που μας δίνουν τη θέση του στοιχείου που σε πρώτη ευκαιρία θα εξαχθεί και τη θέση του στοιχείου που μόλις εισήλθε. π.χ. σε μια αποθήκη φαρμάκων η εξαγωγή-πώληση αυτών γίνεται με τη μέθοδο **FIFO.**

Τα στοιχεία ενός **αρχείου** ονομάζονται **εγγραφές** .
Κάθε εγγραφή αποτελείται από ένα τουλάχιστον **πεδίο** (**πρωτεύων κλειδί**) που ταυτοποιεί την εγγραφή και από επιπλέον πεδία που καθορίζουν χαρακτηριστικά της.

Ενότητα 6 Εισαγωγή στον Προγραμματισμό (κεφ. 6 σχολ. βιβλίου)

Κάθε γλώσσα προγραμματισμού έχει το δικό της λεξιλόγιο και τα προγράμματα της ακολουθούν αυστηρούς γραμματικούς και συντακτικούς κανόνες. Για τη δημιουργία σωστών προγραμμάτων είναι απαραίτητη η γνώση των εντολών και του τρόπου σύνταξής τους.

Ορισμοί:
Ανάλυση analysis: Η μεθοδική μελέτη ενός προβλήματος και η διαδικασία της διάσπασης του σε μικρότερες μονάδες για περαιτέρω έρευνα σε λεπτομέρεια.
Αντικείμενο πρόγραμμα: Πρόγραμμα υπολογιστή σε μια τελική γλώσσα που έχει μεταφραστεί από πηγαίο πρόγραμμα.
Γεγονός: Ενέργεια του χρήστη μιας εφαρμογής που αναγκάζει την εφαρμογή να ανταποκριθεί. Τα γεγονότα συχνά σχετίζονται με κινήσεις του ποντικιού ή πληκτρολογήσεις.
Γλώσσα μηχανής machine language: Γλώσσα χαμηλού επιπέδου που οι εντολές της αποτελούνται μόνο από τα δυαδικά ψηφία.
Γλώσσα προγραμματισμού programming language: Τεχνητή γλώσσα σχεδιασμένη για να δημιουργεί ή να εκφράζει προγράμματα.
Διερμηνευτής interpreter: Πρόγραμμα που μεταφράζει και εκτελεί κάθε εντολή μια γλώσσας προγραμματισμού υψηλού επιπέδου πριν τη μετάφραση και εκτέλεση της επόμενης.
Δομημένος προγραμματισμός: Τρεις τύποι ελέγχου χρησιμοποιούνται στο δομημένο προγραμματισμό: ακολουθιακός, υπό συνθήκη και επαναληπτικός.
Μεταγλωττιστής compiler: Πρόγραμμα υπολογιστή που χρησιμοποιείται για τη μετάφραση σε γλώσσα χαμηλού επιπέδου ενός προγράμματος εκφρασμένου σε γλώσσα.
Οδηγούμενο από γεγονότα event driven: Η ιδιότητα ενός λειτουργικού συστήματος ή περιβάλλοντος, κατά την οποία όταν συμβεί ένα γεγονός εκτελείται κατάλληλο τμήμα κώδικα για την εξυπηρέτησή του.
Πρόγραμμα (υπολογιστή) program: Ακολουθία εντολών κατάλληλων για επεξεργασία. Η επεξεργασία περιλαμβάνει τη χρήση μεταφραστικού προγράμματος για να προετοιμάσει το πρόγραμμα για εκτέλεση, καθώς και την ίδια την εκτέλεση του προγράμματος.
Προγραμματισμός programming: Η διαδικασία δημιουργίας προγραμμάτων υπολογιστή.
Συμβολική γλώσσα assembly language: Γλώσσα χαμηλού επιπέδου εξαρτώμενη από το υλικό και η οποία έχει άμεση αντιστοιχία με τη γλώσσα μηχανής. Αποτελεί συμβολική αναπαράσταση του δυαδικού κώδικα της γλώσσας μηχανής και χρειάζεται συμβολομετάφραση.
Συμβολομεταφραστής assembler: Πρόγραμμα που μεταφράζει συμβολική γλώσσα σε γλώσσα μηχανής του δεδομένου υπολογιστή.
Κώδικας code: Ένα ή περισσότερα προγράμματα ή τμήμα προγράμματος.
Περιβάλλον ανάπτυξης λογισμικού: Σύνολο μεταφραστικών προγραμμάτων και άλλων εργαλείων ανάπτυξης λογισμικού που χρησιμοποιούνται στη δημιουργία προγραμμάτων εφαρμογών.
Πληροφορία information: Γνώση που αφορά πράγματα όπως πράξεις, έννοιες, αντικείμενα, γεγονότα, ιδέες ή διεργασίες που μέσα σε συγκεκριμένο κείμενο έχουν μια

ιδιαίτερη σημασία**.**

Η **επίλυση** ενός προβλήματος με τον υπολογιστή περιλαμβάνει τρία εξίσου σημαντικά στάδια.

- ✔ Τον ακριβή προσδιορισμό του προβλήματος.
- ✔ Την ανάπτυξη του αντίστοιχου αλγορίθμου.
- ✔ Τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή.

Ο **προγραμματισμός** ασχολείται με το τρίτο αυτό στάδιο, τη δημιουργία του προγράμματος, δηλαδή του συνόλου των εντολών που πρέπει να δοθούν στον υπολογιστή, ώστε να υλοποιηθεί ο αλγόριθμος για την επίλυση του προβλήματος. Ο προγραμματισμός είναι αυτός που δίνει την εντύπωση ότι, οι υπολογιστές είναι έξυπνες μηχανές που επιλύουν τα πολύπλοκα προβλήματα. Η εντύπωση αυτή όμως είναι απλώς μία ψευδαίσθηση.
Οι γλώσσες προγραμματισμού αναπτύχθηκαν με σκοπό την επικοινωνία του ανθρώπου (προγραμματιστή) με τη μηχανή (υπολογιστή).

Ένα πρόγραμμα σε **γλώσσα μηχανής** είναι μια ακολουθία δυαδικών ψηφίων, που αποτελούν εντολές προς τον επεξεργαστή για στοιχειώδεις λειτουργίες Οι εντολές σε **συμβολική γλώσσα** αποτελούνται από συμβολικά ονόματα που αντιστοιχούν σε εντολές της γλώσσας μηχανής. Οι συμβολικές γλώσσες είναι συνδεδεμένες με την αρχιτεκτονική κάθε υπολογιστή
Οι γλώσσες **υψηλού επιπέδου** χρησιμοποιούν ως εντολές απλές λέξεις της αγγλικής γλώσσας ακολουθώντας αυστηρούς κανόνες σύνταξης, οι οποίες μεταφράζονται από τον ίδιο τον υπολογιστή σε εντολές σε γλώσσα μηχανής.

Στα **πλεονεκτήματα** των γλωσσών προγραμματισμού υψηλού επιπέδου σε σχέση με τις συμβολικές μπορούν να αναφερθούν:
1. Ο φυσικότερος και πιο "α**νθρώπινος" τρόπος έκφρασης** των προβλημάτων. Τα προγράμματα σε γλώσσα υψηλού επιπέδου είναι πιο κοντά στα προβλήματα που επιλύουν.
2. Η α**νεξαρτησία από** τον τύπο του υπολογιστή. Προγράμματα σε μία γλώσσα υψηλού επιπέδου μπορούν να εκτελεστούν σε οποιονδήποτε υπολογιστή με ελάχιστες ή καθόλου μετατροπές. Η δυνατότητα της **μεταφερσιμότητας** των προγραμμάτων είναι σημαντικό προσόν.
3. Η ευκολία της **εκμάθησης** και εκπαίδευσης ως απόρροια των προηγουμένων.
4. Η **διόρθωση λαθών** και η **συντήρηση** προγραμμάτων σε γλώσσα υψηλού επιπέδου είναι πολύ ευκολότερο έργο.

Συνολικά οι γλώσσες υψηλού επιπέδου ελάττωσαν σημαντικά το **χρόνο** και το **κόστος** παραγωγής νέων προγραμμάτων, αφού λιγότεροι προγραμματιστές μπορούν σε μικρότερο χρόνο να αναπτύξουν προγράμματα που χρησιμοποιούνται σε περισσότερους υπολογιστές.

Μπορούμε να ισχυριστούμε με βεβαιότητα ότι μία γλώσσα προγραμματισμού που να είναι αντικειμενικά καλύτερη από τις άλλες δεν υπάρχει, ούτε πρόκειται να υπάρξει.

Κάθε γλώσσα προσδιορίζεται από το αλφάβητο της, το λεξιλόγιο της, τη γραμματική της

και τη σημασιολογία της.
✔ **Αλφάβητο** μίας γλώσσας καλείται το σύνολο των στοιχείων που χρησιμοποιούνται από τη γλώσσα.
✔ Το Λ**εξιλόγιο** αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από την γλώσσα.
✔ Η **Γραμματική** αποτελείται από το τυπικό ή τυπολογικό και το συντακτικό.
Τυπικό είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία λέξη είναι αποδεκτή.
Συντακτικό είναι το σύνολο των κανόνων που καθορίζει τη νομιμότητα της διάταξης και της σύνδεσης των λέξεων της γλώσσας για τη δημιουργία προτάσεων.
✔ Η Σ**ημασιολογία** (Semantics) είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα. Στις γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο **δημιουργός** της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας.

Ο **δομημένος** προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.
Ο **Δομημένος** Προγραμματισμός αποτελεί μία μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων. Οι γλώσσες προγραμματισμού εξελίσσονται ενσωματώνοντας νέα χαρακτηριστικά και δυνατότητες, όμως η βασική τους φιλοσοφία εξακολουθεί να στηρίζεται στις αρχές του Δομημένου Προγραμματισμού.

Επιγραμματικά μπορούμε να αναφέρουμε τα εξής **πλεονεκτήματα** του δομημένου προγραμματισμού.
✔ Δημιουργία απλούστερων προγραμμάτων.
✔ Άμεση μεταφορά των αλγορίθμων σε προγράμματα.
✔ Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα.
✔ Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
✔ Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
✔ Ευκολότερη διόρθωση και συντήρηση.

Εντολή **Goto**: ρητή διακλάδωση σε συγκεκριμένη θέση. Η χρήση της είναι αντίθετη με τις αρχές του δομημένου προγραμματισμού.

Κάθε πρόγραμμα που γράφτηκε σε οποιαδήποτε γλώσσα προγραμματισμού, πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές **γλώσσας μηχανής.**
Το αρχικό πρόγραμμα λέγεται **πηγαίο πρόγραμμα** (source), ενώ το πρόγραμμα που παράγεται από το μεταγλωττιστή λέγεται α**ντικείμενο** πρόγραμμα (object).

Το αντικείμενο πρόγραμμα είναι μεν σε μορφή κατανοητή από τον υπολογιστή, αλλά συνήθως δεν είναι σε θέση να εκτελεστεί. Χρειάζεται να συμπληρωθεί και να συνδεθεί με άλλα τμήματα προγράμματος απαραίτητα για την εκτέλεσή του, τμήματα που είτε τα

γράφει ο προγραμματιστής είτε βρίσκονται στις **βιβλιοθήκες** (libraries) της γλώσσας. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση ονομάζεται **συνδέτης – φορτωτής** (linker loader).
Το αποτέλεσμα του συνδέτη είναι η παραγωγή του **εκτελέσιμου προγράμματος** (executable), το οποίο είναι το τελικό πρόγραμμα που εκτελείται από τον υπολογιστή. Για το λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση.

Διερμηνευτής vs μεταγλωττιστής

Ομοιότητες:
Και οι δύο μεταφράζουν το πηγαίο πρόγραμμα (υψηλού επιπέδου) σε γλώσσα μηχανής.
Και οι δύο ανιχνεύουν τα συντακτικά λάθη.

Διαφορές:
Ο μεταγλωττιστής μεταγλωττίζει όλο το πρόγραμμα και με τη βοήθεια του συνδέτη – φορτωτή παράγεται το εκτελέσιμο.
Ο διερμηνευτής εκτελεί μία μία τις εντολές και δε χρειάζεται συνδέτη-φορτωτή.
Ο διερμηνευτής αφού εκτελεί τις εντολές μία μία έχει το πλεονέκτημα της **άμεσης διόρθωσης τ**ων λαθών. Για το λόγο αυτό χρησιμοποιείται συνήθως κατά την συγγραφή-διόρθωση ενός προγράμματος.
Η εκτέλεση ενός προγράμματος με τον διερμηνευτή είναι πιο αργή, γιατί για να εκτελεστεί το πρόγραμμα, πρέπει κάθε φορά να ξαναγίνεται η διερμηνεία από την αρχή, ενώ ο μεταγλωττιστής παράγει μια φορά το αντικείμενο πρόγραμμα και δεν χρειάζεται ξανά μεταγλώττιση.
Για να εκτελεστεί ένα πρόγραμμα με τον διερμηνευτή είναι απαραίτητη η παρουσία του πηγαίου προγράμματος ενώ με τον μεταγλωττιστή μόνο την πρώτη φορά.

Τα λάθη του προγράμματος είναι γενικά δύο ειδών, **λογικά** και **συντακτικά**. Τα λογικά λάθη εμφανίζονται μόνο στην **εκτέλεση**, ενώ τα συντακτικά λάθη στο στάδιο της **μεταγλώττισης**.

Συντακτικά λάθη
Οφείλονται σε αναγραμματισμούς γραμμάτων εντολών, παράληψη δήλωσης δεδομένων κ.λπ. Εντοπίζονται από τον μεταγλωττιστή ή τον διερμηνευτή. Πρέπει να διορθωθούν ώστε να δημιουργηθεί το εκτελέσιμο πρόγραμμα. Οι γλώσσες προγραμματισμού διαθέτουν ένα σύνολο **συντακτικών κανόνων** (syntax rules), οι οποίοι καθορίζουν αν ένα πρόγραμμα είναι σωστά δομημένο. Ο μεταγλωττιστής ελέγχει το πρόγραμμά σας και αν κάποιος ή κάποιοι από αυτούς τους συντακτικούς κανόνες παραβιάζονται τότε αναφέρει ένα συντακτικό λάθος. Σε αυτή την περίπτωση τα λάθη θα πρέπει να διορθωθούν και το πρόγραμμα να μεταγλωττιστεί ξανά.

Λογικά λάθη
Οφείλονται σε σφάλματα κατά την υλοποίηση του αλγορίθμου.
Εντοπίζονται μόνο κατά την εκτέλεση του προγράμματος.
Είναι τα πλέον σοβαρά και δύσκολα στην διόρθωση.

Ο σημαντικότερος τύπος προγραμματιστικού λάθους δεν είναι τα συντακτικά, αλλά τα λάθη που έχουν ως αποτέλεσμα το πρόγραμμα να παράγει λάθος αποτελέσματα ή καθόλου αποτελέσματα. Τα λάθη αυτά, τα οποία εμποδίζουν το πρόγραμμά σας να

επιλύσει ένα πρόβλημα εξαιτίας ενός λάθους στη λογική σας, ονομάζονται σφάλματα (bugs) ή **λογικά λάθη**.

Για τη δημιουργία, τη μετάφραση και την εκτέλεση ενός προγράμματος απαιτούνται τουλάχιστον τρία προγράμματα: ο **συντάκτης**, ο **μεταγλωττιστής** και ο **συνδέτης**. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν αυτά τα προγράμματα με ενιαίο τρόπο.

Ενότητα 7 Υποπρογράμματα (κεφ. 10 σχολ. Βιβλίου)

Έχοντας πλέον εξασκηθεί σε πολλά σημεία της αλγοριθμικής επίλυσης προβλημάτων μπορείτε να αναγνωρίσετε πότε μία **ενέργεια**:

- ✔ **επαναλαμβάνεται** ώστε να χρησιμοποιήσετε τη **δομή επανάληψης** ή
- ✔ **εφαρμόζεται** σε πολλά **όμοια δεδομένα** ώστε να χρησιμοποιήσετε **πίνακες**.

Προχωρώντας σε πιο πολύπλοκα προβλήματα, θα δείτε ότι συχνά κάποια **ενέργεια** (ή και ολόκληρο αλγοριθμικό κομμάτι), επαναλαμβάνεται σχεδόν αυτούσια σε πολλά **διαφορετικά σημεία** του αλγόριθμου, αφού επεξεργάζεται **διαφορετικά δεδομένα** αλλά με τον ίδιο ακριβώς τρόπο. "Επαναλαμβάνεται" δηλαδή, αλλά με τρόπο που δεν ταιριάζει ώστε να χρησιμοποιηθεί απλά η δομή επανάληψης (με ή χωρίς τη χρήση πίνακα). Σε τέτοιες περιπτώσεις δεν είναι καλή λύση να ξαναγράψουμε το ίδιο "αλγοριθμικό κομμάτι" σε όλα τα σημεία που χρειάζεται.

Θα μάθετε πώς μπορείτε να αντιμετωπίσετε συστηματικά και αποτελεσματικά τέτοια προβλήματα εντοπίζοντας σε αυτά αυτόνομες λειτουργίες και κατασκευάζοντας τα αντίστοιχα αλγοριθμικά κομμάτια που θα τα ονομάσουμε **υποπρογράμματα**. Θα δείτε πώς μπορείτε να χρησιμοποιήσετε (καλέσετε) αυτά τα αλγοριθμικά κομμάτια σε πολλά διαφορετικά σημεία του αλγόριθμου χωρίς να τα ξαναγράψετε.

Η **βιβλιοθήκη** (library) είναι μια συλλογή εργαλείων που έχουν γραφτεί από άλλους προγραμματιστές προκειμένου να εκτελούν συγκεκριμένες λειτουργίες. Οι βιβλιοθήκες είναι πολύ σημαντικές για τον προγραμματισμό, γιατί μας δίνουν τη δυνατότητα να χρησιμοποιούμε τα εργαλεία που περιλαμβάνονται σε αυτές και μας απαλλάσσουν από τον κόπο να τα γράφουμε μόνοι μας.

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

Για να αποφύγεις τα πλέον κοινά λάθη να **προσέχεις** ιδιαίτερα:

✔ Πριν ξεκινήσεις να γράφεις το πρόγραμμα σου, να μελετήσεις πώς το πρόγραμμα μπορεί να αναλυθεί σε επιμέρους τμήματα και να αποφασίσεις για τα αντίστοιχα υποπρογράμματα. Χρήσιμο είναι να κάνεις ένα διάγραμμα που θα δείχνει την ιεραρχία ανάμεσα στα υποπρογράμματα, τότε να αναπτύσσεις τους αλγόριθμους για το κάθε υποπρόγραμμα και στη συνέχεια να γράφεις το πρόγραμμα.

✔ Να μελετάς αν ένα υποπρόγραμμα πρέπει να υλοποιηθεί με διαδικασία ή μπορεί να υλοποιηθεί με συνάρτηση.

✔ Εξέτασε αν κάποια υποπρογράμματα τα οποία έχεις ήδη γράψει ή υπάρχουν σε έτοιμες βιβλιοθήκες προγραμμάτων μπορούν να χρησιμοποιηθούν. Θα γλιτώσεις χρόνο και κόπο.

✔ Κάθε υποπρόγραμμα να προσπαθείς να είναι όσο το δυνατόν πιο ανεξάρτητο από τα άλλα. Αυτό σε προφυλάσσει από λάθη στο πρόγραμμα σου και σου επιτρέπει τη χρήση του σε άλλα προγράμματα αργότερα.

✔ Να ορίζεις τον τύπο της συνάρτησης. Οι συναρτήσεις παράγουν μόνο ένα αποτέλεσμα συγκεκριμένου τύπου, ακεραίου, πραγματικού κ.λπ. που πρέπει να ορίζεται.

✔ Να μην υπάρχουν λάθη στην αντιστοίχιση τυπικών και πραγματικών παραμέτρων. Πρόσεξε ότι οι λίστες πρέπει να περιέχουν τον ίδιο αριθμό παραμέτρων και κάθε τυπική

παράμετρος με την αντίστοιχη πραγματική πρέπει να είναι του ιδίου τύπου.

Οι μαθητές ενθαρρύνονται να χρησιμοποιούν υποπρογράμματα. Ιδιαίτερη έμφαση πρέπει να δοθεί στη σωστή ανάλυση του προγράμματος σε υποπρογράμματα και στην επιλογή του κατάλληλου τύπου συνάρτησης ή διαδικασίας. Η ανάλυση αυτή πρέπει να γίνεται πάντα πριν αρχίσουν να γράφουν το πρόγραμμα. Πρέπει να αποφασίζουν για κάθε υποπρόγραμμα το είδος του, τη λειτουργία του, καθώς και τον αριθμό των παραμέτρων που χρειάζεται.

Όταν ένα τμήμα προγράμματος επιτελεί ένα αυτόνομο έργο και έχει γραφεί χωριστά από το υπόλοιπο πρόγραμμα, τότε αναφερόμαστε σε υποπρόγραμμα (subprogram) Υπάρχουν πάντως τρεις **ιδιότητες** που πρέπει να διακρίνουν τα υποπρογράμματα:
✔ Κάθε υποπρόγραμμα έχει μόνο **μία είσοδο** και **μία έξοδο**. Στην πραγματικότητα κάθε υποπρόγραμμα ενεργοποιείται με την είσοδο σε αυτό, που γίνεται πάντοτε από την αρχή του, εκτελεί ορισμένες ενέργειες, και απενεργοποιείται με την έξοδο από αυτό που γίνεται πάντοτε από το τέλος του.
✔ Κάθε υποπρόγραμμα πρέπει να είναι **ανεξάρτητο** από τα άλλα. Αυτό σημαίνει ότι κάθε υποπρόγραμμα μπορεί να σχεδιαστεί, να αναπτυχθεί και να συντηρηθεί αυτόνομα χωρίς να επηρεαστούν άλλα υποπρογράμματα. Στην πράξη βέβαια η απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί.
✔ Κάθε υποπρόγραμμα πρέπει **να μην είναι πολύ μεγάλο.** Η έννοια του μεγάλου προγράμματος είναι υποκειμενική, αλλά πρέπει κάθε υποπρόγραμμα να είναι τόσο, ώστε να είναι εύκολα κατανοητό για να μπορεί να ελέγχεται. Γενικά κάθε υποπρόγραμμα πρέπει να εκτελεί μόνο μία λειτουργία. αν εκτελεί περισσότερες λειτουργίες, τότε συνήθως μπορεί και πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

Πλεονεκτήματα τμηματικού προγραμματισμού
1. **Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος.**
Επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι την αντιμετώπιση του συνολικού προβλήματος. Με τη σταδιακή επίλυση των υποπροβλημάτων και τη δημιουργία των αντιστοίχων υποπρογραμμάτων τελικά επιλύεται το συνολικό πρόβλημα.

2. **Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.**
Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση ενός συγκεκριμένου τμήματος του, χωρίς οι αλλαγές αυτές να επηρεάσουν όλο το υπόλοιπο πρόγραμμα. Επίσης διευκολύνει οποιονδήποτε χρειαστεί να διαβάσει και να κατανοήσει τον τρόπο που λειτουργεί το πρόγραμμα. Όπως έχει πολλές φορές τονιστεί, αυτό είναι πολύ σημαντικό χαρακτηριστικό του σωστού προγραμματισμού, αφού ένα μεγάλο πρόγραμμα στον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.

3. **Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.**
Πολύ συχνά χρειάζεται η ίδια λειτουργία σε διαφορετικά σημεία ενός προγράμματος. από τη στιγμή που ένα υποπρόγραμμα έχει γραφεί, μπορεί το ίδιο να καλείται από πολλά σημεία του προγράμματος. Έτσι μειώνονται το μέγεθος του προγράμματος, ο χρόνος που απαιτείται για τη συγγραφή του και οι πιθανότητες λάθους, ενώ ταυτόχρονα το πρόγραμμα γίνεται πιο εύληπτο και κατανοητό.

4. **Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.**
Ένα υποπρόγραμμα που έχει γραφεί μπορεί να χρησιμοποιηθεί πολύ εύκολα και σε άλλα προγράμματα. από τη στιγμή που έχει δημιουργηθεί, η χρήση του δεν διαφέρει από τη χρήση των ενσωματωμένων συναρτήσεων που παρέχει η γλώσσα προγραμματισμού, όπως για τον υπολογισμό του ημίτονου ή του συνημίτονου ή την εντολή με την οποία εκτελεί μία συγκεκριμένη διαδικασία, για παράδειγμα γράφει στην οθόνη (εντολή ΓΡΑΨΕ). Αν λοιπόν χρειάζεται συχνά κάποια λειτουργία που δεν υποστηρίζεται απευθείας από τη γλώσσα, όπως για παράδειγμα η εύρεση του μικρότερου δύο αριθμών, τότε μπορεί να γραφεί το αντίστοιχο υποπρόγραμμα. Η συγγραφή πολλών υποπρογραμμάτων και η δημιουργία βιβλιοθηκών με αυτά, ουσιαστικά επεκτείνουν την ίδια τη γλώσσα προγραμματισμού.

Οι **παράμετροι** είναι σαν τις κοινές μεταβλητές ενός προγράμματος με μία ουσιώδη διαφορά, χρησιμοποιούνται για να περνούν τιμές στα υποπρογράμματα. Μία παράμετρος είναι μία μεταβλητή που επιτρέπει το πέρασμα της τιμής της από ένα τμήμα προγράμματος σε ένα άλλο.

Υπάρχουν δύο είδη υποπρογραμμάτων: οι **διαδικασίες και οι συναρτήσεις.** Κάθε είδος υποπρογράμματος καθορίζεται από το είδος της λειτουργίας που καλείται να επιτελέσει, ενώ ο τρόπος κλήσης και σύνταξής των δύο αυτών ειδών είναι διαφορετικός.

Οι **διαδικασίες** μπορούν να εκτελέσουν οποιαδήποτε λειτουργία από αυτές που μπορεί να εκτελέσει ένα πρόγραμμα. Να εισάγουν δεδομένα, να εκτελέσουν υπολογισμούς, να μεταβάλλουν τις τιμές των μεταβλητών και να τυπώσουν αποτελέσματα. Με τη χρήση των παραμέτρων, αυτές τις τιμές μπορούν να τις μεταφέρουν και στα άλλα υποπρογράμματα. Η εντολή **ΚΑΛΕΣΕ** μας επιτρέπει να καλέσουμε μία διαδικασία (όχι όμως συναρτήσεις, αυτές καλούνται μέσα από εκφράσεις). Οι διαδικασίες είναι υποπρογράμματα που γράφονται μετά το κυρίως πρόγραμμα. Μπορούν να κληθούν με την εντολή ΚΑΛΕΣΕ από οποιοδήποτε σημείο του προγράμματος. Οι παράμετροι αναφέρονται ονομαστικά κατά τη δήλωση της διαδικασίας και στη συνέχεια ο τύπος τους δηλώνεται στο τμήμα «Μεταβλητές».

Σαν παράμετρος μπορεί να περαστεί οτιδήποτε έχει τιμή, όπως μεταβλητές, πίνακες, σταθερές, εκφράσεις.
Αν μία διαδικασία κληθεί με μεταβλητή σαν παράμετρο και αλλαχθεί η τιμή της μέσα στην διαδικασία, τότε η τιμή της μεταβλητής στο καλών υποπρόγραμμα θα ενημερωθεί (αλλαχθεί) μετά την εκτέλεση της εντολής **ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ**.
Αν θέλουμε να περάσουμε σε διαδικασία κάποια μεταβλητή-πραγματική παράμετρο χωρίς να επιστραφεί αλλαγμένη, τότε πρέπει να κάνουμε κάποια «πράξη» με αυτήν ώστε να πάψει να είναι μεταβλητή (π.χ. 1*χ, χ + 0).

Αντίθετα η λειτουργία των **συναρτήσεων** είναι πιο περιορισμένη. Οι συναρτήσεις υπολογίζουν μόνο μία τιμή, αριθμητική, χαρακτήρα ή λογική και μόνο αυτήν επιστρέφουν στο υποπρόγραμμα που την κάλεσε. Η συνάρτηση είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή με το όνομά της (όπως οι μαθηματικές συναρτήσεις). Η ΓΛΩΣΣΑ επιτρέπει τη δημιουργία συναρτήσεων από το χρήστη, οι οποίες μπορούν στη συνέχεια να χρησιμοποιηθούν όπως και οι **ενσωματωμένες** συναρτήσεις. Το αποτέλεσμα κάθε συνάρτησης έχει πάντα τιμή κάποιου τύπου δεδομένων και δηλώνεται μετά τις παραμέτρους της συνάρτησης (ΑΚΕΡΑΙΑ, ΠΡΑΓΜΑΤΙΚΗ, ΧΑΡΑΚΤΗΡΑΣ ή ΛΟΓΙΚΗ).

Οι συναρτήσεις δεν επιτρέπεται να επιστρέφουν πίνακες. Όλες οι συναρτήσεις πρέπει να περιέχουν μία εντολή του τύπου ΌνομαΣυνάρτησης ← τιμή, ώστε να επιστρέφουν κάποια τιμή στο καλών υποπρόγραμμα.
Οι παράμετροι στις συναρτήσεις περνιούνται έτσι ώστε οποιαδήποτε αλλαγή στις τυπικές παραμέτρους μιας συνάρτησης δεν επηρεάζει τις πραγματικές παραμέτρους της συνάρτησης. Μια συνάρτηση που αλλάζει το περιεχόμενο των παραμέτρων της δεν επιστρέφει τις αλλαγές αυτές στις πραγματικές παραμέτρους.

Η λίστα των **τυπικών** παραμέτρων καθορίζει τις παραμέτρους στη δήλωση του υποπρογράμματος. Η λίστα των **πραγματικών παραμέτρων** καθορίζει τις παραμέτρους στην κλήση του υποπρογράμματος.
Κανόνες περάσματος παραμέτρων
1. Ο αριθμός των πραγματικών και των τυπικών παραμέτρων πρέπει να είναι ίδιος.
2. Κάθε πραγματική παράμετρος αντιστοιχεί στην τυπική παράμετρο που βρίσκεται στην αντίστοιχη θέση. Για παράδειγμα, η πρώτη της λίστας των τυπικών παραμέτρων στην πρώτη της λίστας των πραγματικών παραμέτρων κ.ο.κ.
3. Η τυπική παράμετρος και η αντίστοιχη της πραγματική πρέπει να είναι του ιδίου τύπου.

Όλες οι μεταβλητές είναι γνωστές, έχουν ισχύ όπως λέγεται, μόνο για το τμήμα προγράμματος στο οποίο έχουν δηλωθεί, ισχύουν δηλαδή **τοπικά** για το συγκεκριμένο υποπρόγραμμα ή κυρίως πρόγραμμα.

Επιστρεφόμενη τιμή: η πληροφορία που επιστρέφει μια συνάρτηση όταν επιστρέφει τον έλεγχο ροής-εκτέλεσης στο σημείο από το οποίο κλήθηκε (διεύθυνση επιστροφής).
Επικεφαλίδα συνάρτησης-διαδικασίας: προσδιορίζει τη διεπαφή της συνάρτησης και άρα τον τρόπο αναφορής-χρήσης του υποπρογράμματος.

Αν στις πραγματικές παραμέτρους υπάρχει συγκεκριμένη θέση του πίνακα, τότε η αντίστοιχη τυπική παράμετρος πρέπει να είναι μεταβλητή.

Άρα, ένα υποπρόγραμμα δεν διαβάζει, ούτε εμφανίζει στην οθόνη, εκτός αν ορίζεται ρητά στην εκφώνηση.
Αν το αποτέλεσμα είναι ένα, μπορεί να υλοποιηθεί με Συνάρτηση η οποία "κουβαλάει" η ίδια το αποτέλεσμα.
Σε Διαδικασία, όσα και να είναι τα αποτελέσματα, είναι παράμετροι.
Οι μεταβλητές πάντα δηλώνονται τοπικά (εμβέλεια) και όλες, άρα οι μεταβλητές του προγράμματος δηλώνονται μόνο σε αυτό και των υποπρογραμμάτων μόνο σε αυτά.

Η έννοια της **στοίβας** είναι πολύ χρήσιμη στο ίδιο το λογισμικό των γλωσσών προγραμματισμού. Όταν μία διαδικασία ή συνάρτηση καλείται από το κύριο πρόγραμμα, τότε η αμέσως επόμενη διεύθυνση του κύριου προγράμματος, που ονομάζεται **διεύθυνση επιστροφής** (return address), αποθηκεύεται από το μεταφραστή σε μία στοίβα που ονομάζεται **στοίβα χρόνου εκτέλεσης** (execution time stack). Μετά την εκτέλεση της διαδικασίας ή της συνάρτησης, η διεύθυνση επιστροφής απωθείται από τη στοίβα και έτσι ο έλεγχος του προγράμματος μεταφέρεται και πάλι στο κύριο πρόγραμμα.
Η **τεχνική** αυτή εφαρμόζεται και γενικότερα, δηλαδή οποτεδήποτε μία διαδικασία ή συνάρτηση καλεί μία διαδικασία ή συνάρτηση.

1ο ΓΕΛ Καστοριάς (Δημήτρης Τζήμας)